Statistical Machine Learning for Bioinformatics

A Crash Course for Data Analysis

Prof. Deisy Morselli Gysi, PhD.

2026-03-01

Table of contents

Preface

Welcome to the Statistical Machine Learning EVOP Course!

It is a pleasure to have you here, and I hope you will find the course engaging and informative. This course is designed to provide you with a comprehensive understanding of statistical machine learning concepts and techniques, equipping you with the skills needed to analyze and interpret complex data. The main goal is for you to understand the fundamental principles of statistical machine learning, including key algorithms, methodologies, and applications. We will cover a wide range of topics, from basic concepts to advanced techniques, ensuring that you have a solid foundation in the field. It is quite obvious that the course will not cover all the algorithms and techniques in the field, but we will focus on the most important and widely used ones. By the end of the course, you should be able to apply these concepts to real-world problems and have a deeper understanding of how statistical machine learning can be used to extract insights from data.

Part I

Foundations

1 Introduction

As researchers in the biological sciences, we are often swimming in high-dimensional data—whether it is proteomics, genomics, transcriptomics, single-cell sequencing or any other omics. Transitioning from the classical frequentist statistics to a Machine Learning (ML) mindset is a powerful move for any novel biological discovery. ML techniques can help us uncover patterns, make predictions, and gain insights from complex datasets that traditional methods might miss.

In this course, we will explore the fundamentals of Machine Learning with a focus on applications in the biological sciences. We will cover key concepts, algorithms, and practical implementations using R. By the end of this course, you will have a solid understanding of how to apply ML techniques to your own research questions.

1.1 What is Machine Learning?

Machine Learning is a field of study within computer science that gives computers the ability to learn without being explicitly programmed to perform specific tasks. Instead of following a set of predefined rules, ML algorithms identify patterns in data and use these patterns to make predictions or decisions. ML was first coined by Arthur Samuel in 1959, who defined it as "the field of study that gives computers the ability to learn without being explicitly programmed." Even though ML has been around for decades, it has gained significant traction in recent years due to advancements in computational power, the availability of large datasets, and improvements in algorithms. ML uses statistical techniques to enable computers to improve their performance on a specific task over time as they are exposed to more data (Figure 1.1). This is particularly useful in scenarios where traditional programming approaches are impractical or impossible due to the complexity of the data or the task at hand.

1.2 Why Machine Learning in Biology?

Biological data is inherently complex and high-dimensional. Traditional univariate statistical methods often fall short in capturing the intricate relationships within such data. Machine Learning offers a suite of tools that can handle this complexity, allowing us to:

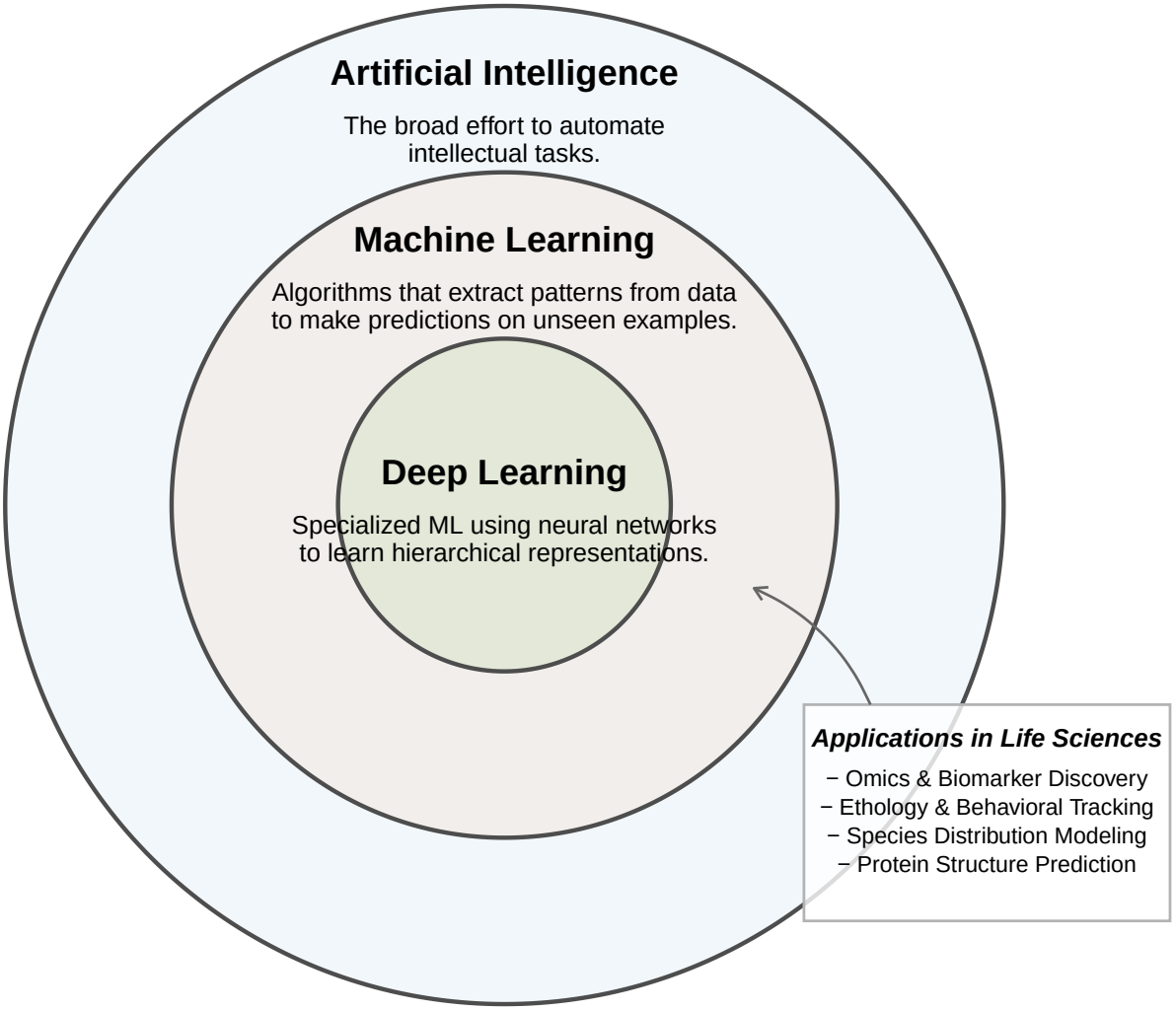


Figure 1.1: The Landscape of Machine Learning in Biology

- **Identify Patterns**: ML algorithms can uncover hidden patterns in large datasets that may not be apparent through traditional analysis.
- **Make Predictions**: ML models can predict outcomes based on input data, which is invaluable for tasks such as disease diagnosis, treatment response prediction, and more.
- **Automate Analysis**: ML can automate the analysis of large datasets, saving time and resources.
- **Integrate Diverse Data Types**: ML techniques can integrate various data types (e.g., genomic, proteomic, clinical) to provide a holistic view of biological systems.

By leveraging Machine Learning, we can enhance our ability to make sense of complex biological data and drive novel discoveries in the life sciences.

1.3 Course Structure

This course is structured into several modules, each focusing on different aspects of Machine Learning:

1. **Supervised Learning**: Techniques where the model is trained on labeled data to make predictions.
2. **Unsupervised Learning**: Methods for discovering patterns in unlabeled data.
3. **Model Evaluation**: Strategies for assessing the performance of ML models.

Each module will include theoretical concepts, practical examples, and hands-on exercises using R. We will also explore real-world biological datasets to illustrate the application of ML techniques in research.

1.4 Types of Machine Learning

Machine Learning can be broadly categorized into three main types: Supervised Learning, Unsupervised Learning, and Reinforcement Learning, as depicted in Figure 1.2. In this course, we will focus primarily on the first two types, as they are most relevant to biological data analysis.

- **Unsupervised Learning**: In unsupervised learning, the model is trained on an **unlabeled dataset**, meaning that the input data does not have corresponding output labels. The goal is to find hidden patterns or structures in the data. Common algorithms include clustering methods (e.g., K-means, hierarchical clustering) and dimensionality reduction techniques (e.g., PCA, t-SNE), Chapter 3. Applications in biology include identifying cell types from single-cell RNA-seq data and discovering gene expression patterns.

- **Supervised Learning**: In supervised learning, the model is trained on a **labeled dataset**, meaning that each *input data point* is paired with a corresponding *output label*. The goal is for the model to learn the mapping from inputs to outputs so that it can make accurate predictions on new, unseen data. Common algorithms include decision trees, support vector machines, and neural networks. Applications in biology include disease classification based on gene expression profiles and predicting protein functions. We will explore various supervised learning techniques in detail in Chapter 4.
- **Reinforcement Learning**: Although not the focus of this course, reinforcement learning is another type of ML where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards. This approach is less commonly used in biological data analysis but has potential applications in areas like drug discovery and personalized medicine.

1.5 Methodological Challenges in Biological Machine Learning

While Machine Learning offers powerful tools for biological data analysis, several methodological challenges must be addressed to ensure robust and meaningful results. Below, we outline some of the key challenges specific to the application of ML in biology.

1.5.1 Data Quality and Preprocessing

Biological datasets often suffer from issues such as missing values, batch effects, and technical variability. Proper data preprocessing is crucial to mitigate these issues.

- **Normalization and Scaling:** Techniques such as log transformation, z-score normalization, and batch effect correction (e.g., ComBat) are essential to ensure that the data is comparable across samples and conditions.
- **Feature Selection:** Given the high dimensionality of biological data, selecting relevant features (e.g., genes, proteins) is critical to reduce noise and improve model performance. Methods such as variance thresholding, recursive feature elimination, and domain knowledge-based selection can be employed.

1.5.2 The "Curse of Dimensionality" and Data Scarcity

In computational biology, we frequently encounter the $p \gg n$ problem, where the number of features (genes/ transcripts/ proteins/ metabolites) vastly exceeds the number of observations (biological replicates/ individuals).

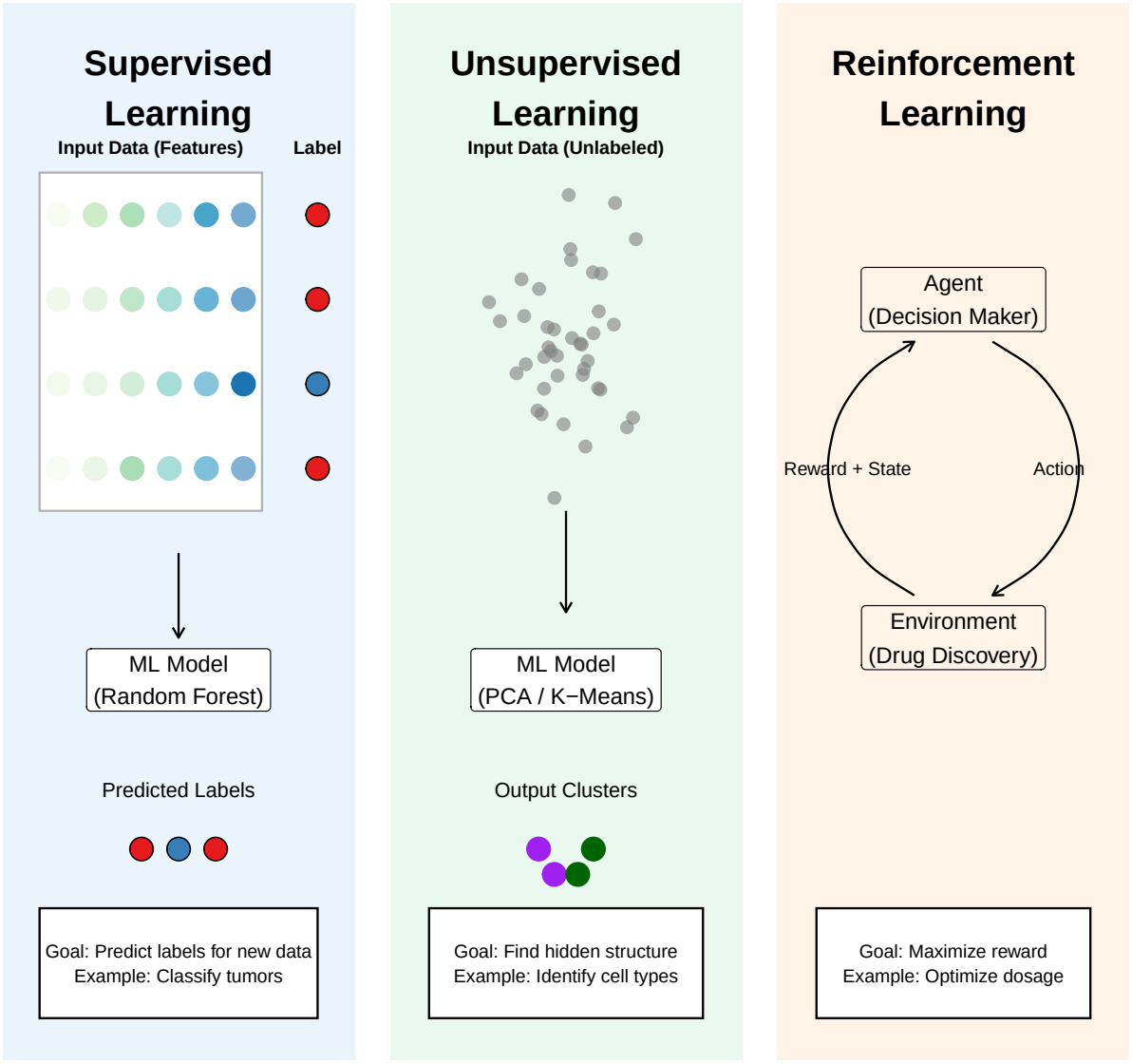


Figure 1.2: The Three Main Types of Machine Learning: Supervised, Unsupervised, and Reinforcement Learning.

- **Dimensionality Concerns:** As the feature space expands, the volume of the space increases so rapidly that the available data becomes sparse. This sparsity makes it mathematically trivial to find a separating hyperplane that appears significant but lacks biological reality. Often, models trained in such high-dimensional spaces will perform well on training data but fail to generalize to new samples.
- **Quality Constraints:** Biological data is inherently noisy, susceptible to batch effects, and expensive to generate. Models trained on insufficient or poorly normalized data will inevitably reflect technical artifacts rather than true physiological signals. Moreover, the limited sample sizes typical in biological studies exacerbate the risk of spurious correlations.

1.5.3 Overfitting and Generalization Error

Overfitting represents the most significant threat to the transition from computational predictions to biological insights. Overfitting occurs when a model possesses excessive degrees of freedom, allowing it to interpolate the stochastic noise within the training set rather than capturing the underlying biological manifold. In high-dimensional transcriptomics, where $p \gg n$, the risk of "memorizing" the training data is exceptionally high. Obviously, this is a critical issue in most of the biological discoveries, where the goal is to identify features that are predictive across diverse populations.

- **The Statistical Mechanism**: In high-dimensional spaces, points are naturally isolated, we say that the data is sparse. A complex algorithm, such as an unpruned Decision Tree or a high-capacity Support Vector Machine, can easily construct a decision boundary that achieves zero training error by isolating these specific coordinates. However, such a boundary is unlikely to represent the true population distribution, leading to a precipitous drop in performance when applied to independent validation cohorts.
- **Regularization and Constraint**: To mitigate this, we must introduce **penalization terms**—such as the L_1 (Lasso) or L_2 (Ridge) norms—which mathematically constrain the coefficient weights, effectively "shrinking" less informative features to zero. This enforces a simpler model architecture that prioritizes broad biological trends over sample-specific idiosyncrasies. We will explore these techniques in detail in Section 4.8.
- **Rigorous Validation Frameworks**: The antidote to overfitting is a robust validation architecture. This involves the strict separation of data into training, testing, and ideally, completely independent external validation sets. Different cross-validation strategies (e.g., k-fold, stratified) should be employed to ensure that the model's performance metrics (e.g., accuracy, AUC-ROC) reflect its ability to generalize beyond the training data. We will discuss these strategies in detail in Chapter 5.
- **The Biological Cost of High Variance**: From a translational perspective, an overfitted model produces "phantom biomarkers." These are genes that appear statistically

significant due to technical artifacts (e.g., batch effects or library prep variance) but fail to replicate in follow-up experiments. Identifying these errors early via ML validation saves significant time and capital that would otherwise be spent on futile *in vitro* or *in vivo* validation.

1.5.4 The Interpretability-Complexity Paradigm

The application of Machine Learning in *omics* often forces a trade-off between a model's **predictive capacity** and its **interpretability**. While high-capacity models—such as Gradient Boosted Trees, Random Forests, or Deep Neural Networks—can capture complex, non-linear gene-gene interactions, they often function as "black boxes," providing little intuition regarding the biological phenomena driving the classification.

- **Mechanistic Transparency vs. Predictive Accuracy**: In a clinical research context, the *rationale* behind a prediction is often as critical as the prediction itself. A model that achieves high AUC (Area Under the Curve) but lacks transparency cannot be used to generate new biological hypotheses or inform therapeutic interventions. To bridge this gap, we employ **explainable AI (XAI)** techniques, such as **SHAP (SHapley Additive exPlanations)** or **LIME**, which mathematically decompose the contribution of each gene to a specific sample's prediction.
- **Strategic Model Selection**: The choice of algorithm must be dictated by the research objective. If the goal is pure diagnostic accuracy, ensemble methods are appropriate. However, if the objective is **biomarker identification**, we must prioritize models with intrinsic feature selection properties (e.g., Sparse Partial Least Squares or Elastic Net). This allows us to map mathematical importance back to biological relevance—transforming "important features" into actionable "biomarkers" that can be validated via pathway enrichment or Gene Ontology (GO) analysis.
- **The Burden of Validation**: A complex, uninterpretable model is difficult to audit for "leakage" or technical bias. By maintaining interpretability, we ensure that the model is making decisions based on genuine biological signal (e.g., the upregulation of an oncogenic pathway) rather than a technical artifact (e.g., different RNA extraction kits), thereby increasing the likelihood of successful translation to the wet lab.

1.5.5 Computational Logistics and Scalability

The processing of large-scale *omics* datasets requires significant memory allocation and optimized algorithmic implementations.

- **Infrastructural Demands:** Complex architectures, particularly in deep learning or iterative manifold learning (UMAP/t-SNE), necessitate high-performance computing

(HPC) environments. Researchers must evaluate the trade-off between algorithmic complexity and the marginal gain in predictive power.

1.5.6 Algorithmic Bias and Ethical Validity

The translation of Machine Learning models into clinical diagnostics needs a profound commitment to data diversity and ethical transparency.

- **Representation Bias:** If the training data lacks ancestral or demographic diversity, the resulting selected genes may exhibit reduced efficacy across broader populations, inadvertently perpetuating healthcare disparities.
- **Accountability:** The use of automated decision-making in healthcare requires rigorous auditing of the training pipeline to ensure that confounding variables (e.g., age, sex, or technical batch) are not driving the model's classification.

2 The Core Workflow of ML & Biological Pre-processing

In the transition from classical biostatistics to Machine Learning, the most critical shift is moving from **explaining the past** to **predicting the future**.

This means that, unlike traditional hypothesis testing where we focus on p-values and confidence intervals, in ML we prioritize **out-of-sample performance**, or well can we generalize to new, unseen data.

While a p-value tells us how unlikely a result is under the null hypothesis, an ML model's utility is measured by its **Generalization Error**: how well it performs on a sample it has never seen before.

2.1 The ML Workflow Overview

A robust ML pipeline for biological data follows these non-negotiable steps:

1. **Data Cleaning**: Handling NAs and filtering low-variance features (e.g., genes with near-zero counts).
2. **Data Spending**: Splitting the data into training and testing sets to prevent data leakage.
3. **Biological Pre-processing**: Normalizing, filtering, and transforming biological data
4. **Model Training**: Fitting the algorithm to the training data.
5. **Evaluation**: Testing the "unseen" data to see if the model actually learned biology or just memorized noise.

2.2 `{tidymodels}`: A Unified Framework for ML in R

The `{tidymodels}` ecosystem provides a cohesive set of packages that streamline the entire ML workflow, from data pre-processing to model evaluation. It emphasizes a tidy, consistent syntax that integrates well with the broader `{tidyverse}`.

At its core, `{tidymodels}` is a unified framework for machine learning in R. Instead of learning different syntax for every single algorithm, you use one consistent language.

Think of it as a Lego set for data science: you snap different pieces together to build a complete pipeline.

To understand `{tidymodels}`, we just need to know the roles of its four main "workers":

2.2.1 1. `{rsample}` (The Splitter)

Before you do anything, you need to set aside data to test your model later. `{rsample}` handles the "Training vs. Testing" split.

- **The Goal:** To make sure the model doesn't just "memorize" the data it has already seen.

2.2.2 2. `{recipes}` (The Chef)

This is where you define your **preprocessing** steps. Just like a cooking recipe, it's a list of instructions to get your raw data ready for the model.

- **Common Steps:** Filling in missing values (imputation), scaling numbers so they are on the same range (0 to 1), or converting text categories into numbers (dummy variables).
- **Crucial Detail:** The recipe defines the steps, but it doesn't "cook" them until the model is actually run.

2.2.3 3. `{parsnip}` (The Interface)

In R, different models (like Random Forests vs. Linear Regression) often require completely different code styles. `{parsnip}` solves this by providing a **unified interface**.

- **Example:** You tell `{parsnip}` you want a "Random Forest," and it handles the background translation to whatever specific R package (the "engine") you want to use. You only have to learn one way to write it.

2.2.4 4. `{workflows}` (The Glue)

A **Workflow** bundles your `{recipe}` and your `{parsnip}` model together into a single object.

- It treats the entire process as one unit. When you want to predict new data, you just pass it to the workflow, and it automatically applies the same preprocessing steps before running the model.

2.2.5 Example

```
library(tidymodels)

# 1. Split the data
data_split <- initial_split(mtcars, prop = 0.8)
train_data <- training(data_split)

# 2. Define the Recipe (The Prep)
# "Predict mpg using all other variables, then scale them"
simple_recipe <- recipe(mpg ~ ., data = train_data) %>%
  step_normalize(all_predictors())

# 3. Define the Model (The Engine)
# "I want a linear regression model using the 'lm' engine"
lm_spec <- linear_reg() %>%
  set_engine("lm")

# 4. Create the Workflow (The Glue)
simple_workflow <- workflow() %>%
  add_recipe(simple_recipe) %>%
  add_model(lm_spec)

# 5. Fit the whole thing
final_model <- fit(simple_workflow,
                   data = train_data)
```

2.3 TCGA Data

For our course, we will use the TCGA Cholangiocarcinoma (CHOL) cohort. We will use the `{TCGAbiolinks}` package to download and prepare the data.

```
############
# Prepare our Data for Machine Learning with
# TCGA Cholangiocarcinoma (CHOL) Cohort
library(TCGAbiolinks)
library(SummarizedExperiment)
library(tidyverse)

# 1. Query only the Cholangiocarcinoma cohort
```

```r
query_chol <- GDCquery(
  project = "TCGA-CHOL",
  data.category = "Transcriptome Profiling",
  data.type = "Gene Expression Quantification",
  workflow.type = "STAR - Counts"
)
#  Load the data into a SummarizedExperiment object
chol_se <- GDCprepare(query_chol)
```

```
|                                                      |  0%
|=                                                     |2.272727% ~2 s remaining
|==                                                    |4.545455% ~2 s remaining
|===                                                   |6.818182% ~1 s remaining
|====                                                  |9.090909% ~1 s remaining
|=====                                                 |11.36364% ~1 s remaining
|=======                                               |13.63636% ~1 s remaining
|========                                              |15.90909% ~3 s remaining
|=========                                             |18.18182% ~3 s remaining
|==========                                            |20.45455% ~2 s remaining
|===========                                           |22.72727% ~2 s remaining
|=============                                         | 25% ~2 s remaining
|==============                                        |27.27273% ~2 s remaining
|===============                                       |29.54545% ~2 s remaining
|================                                      |31.81818% ~2 s remaining
|=================                                     |34.09091% ~1 s remaining
|==================                                    |36.36364% ~1 s remaining
|====================                                  |38.63636% ~2 s remaining
|=====================                                 |40.90909% ~2 s remaining
|======================                                |43.18182% ~2 s remaining
|========================                              |45.45455% ~1 s remaining
|=========================                             |47.72727% ~1 s remaining
|==========================                            | 50% ~1 s remaining
|============================                          |52.27273% ~1 s remaining
|=============================                         |54.54545% ~1 s remaining
|==============================                        |56.81818% ~1 s remaining
|================================                      |59.09091% ~1 s remaining
|=================================                     |61.36364% ~1 s remaining
|==================================                    |63.63636% ~1 s remaining
|====================================                  |65.90909% ~1 s remaining
|=====================================                 |68.18182% ~1 s remaining
|======================================                |70.45455% ~1 s remaining
```

```
|==================================                    |72.72727% ~1 s remaining
|===================================                   | 75% ~1 s remaining
|====================================                  |77.27273% ~1 s remaining
|=====================================                 |79.54545% ~0 s remaining
|======================================                |81.81818% ~0 s remaining
|=======================================               |84.09091% ~0 s remaining
|========================================              |86.36364% ~0 s remaining
|=========================================             |88.63636% ~0 s remaining
|==========================================            |90.90909% ~0 s remaining
|===========================================           |93.18182% ~0 s remaining
|============================================          |95.45455% ~0 s remaining
|=============================================         |97.72727% ~0 s remaining
|======================================================|100% ~0 s remaining
|======================================================|100%                     Completed af
```

```
#######

# 2. Download and prepare the data
# Extract fpkm using
fpkm_data <- assay(chol_se, "fpkm_unstrand")

# 3. Quick Metadata Link
metadata <- as.data.frame(colData(chol_se)) %>%
  select(barcode, sample_type)
head(metadata)
```

```
                                                 barcode         sample_type
TCGA-W5-AA39-01A-11R-A41I-07 TCGA-W5-AA39-01A-11R-A41I-07       Primary Tumor
TCGA-3X-AAVB-01A-31R-A41I-07 TCGA-3X-AAVB-01A-31R-A41I-07       Primary Tumor
TCGA-W5-AA2R-11A-11R-A41I-07 TCGA-W5-AA2R-11A-11R-A41I-07 Solid Tissue Normal
TCGA-W5-AA38-01A-11R-A41I-07 TCGA-W5-AA38-01A-11R-A41I-07       Primary Tumor
TCGA-W5-AA2G-01A-11R-A41I-07 TCGA-W5-AA2G-01A-11R-A41I-07       Primary Tumor
TCGA-W5-AA2Q-11A-11R-A41I-07 TCGA-W5-AA2Q-11A-11R-A41I-07 Solid Tissue Normal
```

```
#######

# 4. ML Preparation (Transpose)
# Often in ML, TPM or FPKM values are used directly
# We need the samples as rows and genes as columns
# We filter for low-expression genes to remove noise
# For fpkm, a common threshold is > 5 in a certain % of samples.
keep <- rowSums(fpkm_data > 5) >= ncol(fpkm_data) * 0.2 # Keep genes expressed in at least 2
```

```r
fpkm_filtered <- fpkm_data[keep, ]
dim(fpkm_filtered) # Check dimensions after filtering
```

```
[1] 8646   44
```

```r
## Combine with metadata
df_ml_fpkm <- as.data.frame(t(fpkm_filtered)) %>%
  rownames_to_column("barcode") %>%
  inner_join(metadata, by = "barcode") %>%
  select(-barcode) %>%
  mutate(sample_type = factor(sample_type))
dim(df_ml_fpkm) # Check final dimensions
```

```
[1]   44 8647
```

2.4 The Spending Plan: Data Splitting Strategies

In biology, samples are precious and often limited ($n < 100$). This scarcity makes the "Data Spending" phase the most high-stakes part of your project. If you use your data too aggressively during training, you will overfit; if you save too much for testing, your model will be too weak to learn the underlying biology. Finally, if you "peek" at the test set during pre-processing, you will introduce **Data Leakage**[1], leading to overly optimistic performance estimates.

2.4.1 The Initial Split: Training vs. Testing

To ensure that our model's performance is a true reflection of its ability to generalize, we must carefully partition our dataset at the very beginning of our analysis. For that, we divide our dataset into two primary components (Figure 2.1):

1. **The Training Set**: This subset is used to train the model. All pre-processing steps (normalization, feature selection, imputation) must be derived solely from this set. Often, we use 70% to 80% of the data for training.
2. **The Test Set**: This subset is held out and only used once at the very end to evaluate the model's performance. It must remain completely unseen during training and pre-processing.

[1] **Data Leakage** occurs when information from outside the training dataset is used to create the model. This can lead to overly optimistic performance estimates because the model has effectively "seen" parts of the test data during training.

> **Note:** Never perform normalization, feature selection, or imputation on the *entire* dataset before splitting. If you calculate the mean expression of a gene using all samples, the training set now "knows" something about the distribution of the test set. This is **Data Leakage**, and it leads to artificially inflated (and ultimately false) performance metrics.

The easiest way to implement this split in R is using the `initial_split()` function from the `{rsample}` package, which is part of the `{tidymodels}` ecosystem.

```
# Split the data into training and testing sets
library(tidymodels)
set.seed(12345) # For reproducibility
data_split <- initial_split(df_ml_fpkm,
                            prop = 0.8)
train_data_chol <- training(data_split)
test_data_chol  <- testing(data_split)

cat("Training samples:", nrow(train_data_chol), "\nTesting samples:", nrow(test_data_chol))
```

```
Training samples: 35
Testing samples: 9
```

```
cat("Proportion of sample types in training set:\n")
```

```
Proportion of sample types in training set:
```

```
print(prop.table(table(train_data_chol$sample_type)))
```

```
      Primary Tumor Solid Tissue Normal
                0.8                 0.2
```

```
cat("Proportion of sample types in testing set:\n")
```

```
Proportion of sample types in testing set:
```

```
print(prop.table(table(test_data_chol$sample_type)))
```

```
      Primary Tumor Solid Tissue Normal
          0.7777778           0.2222222
```

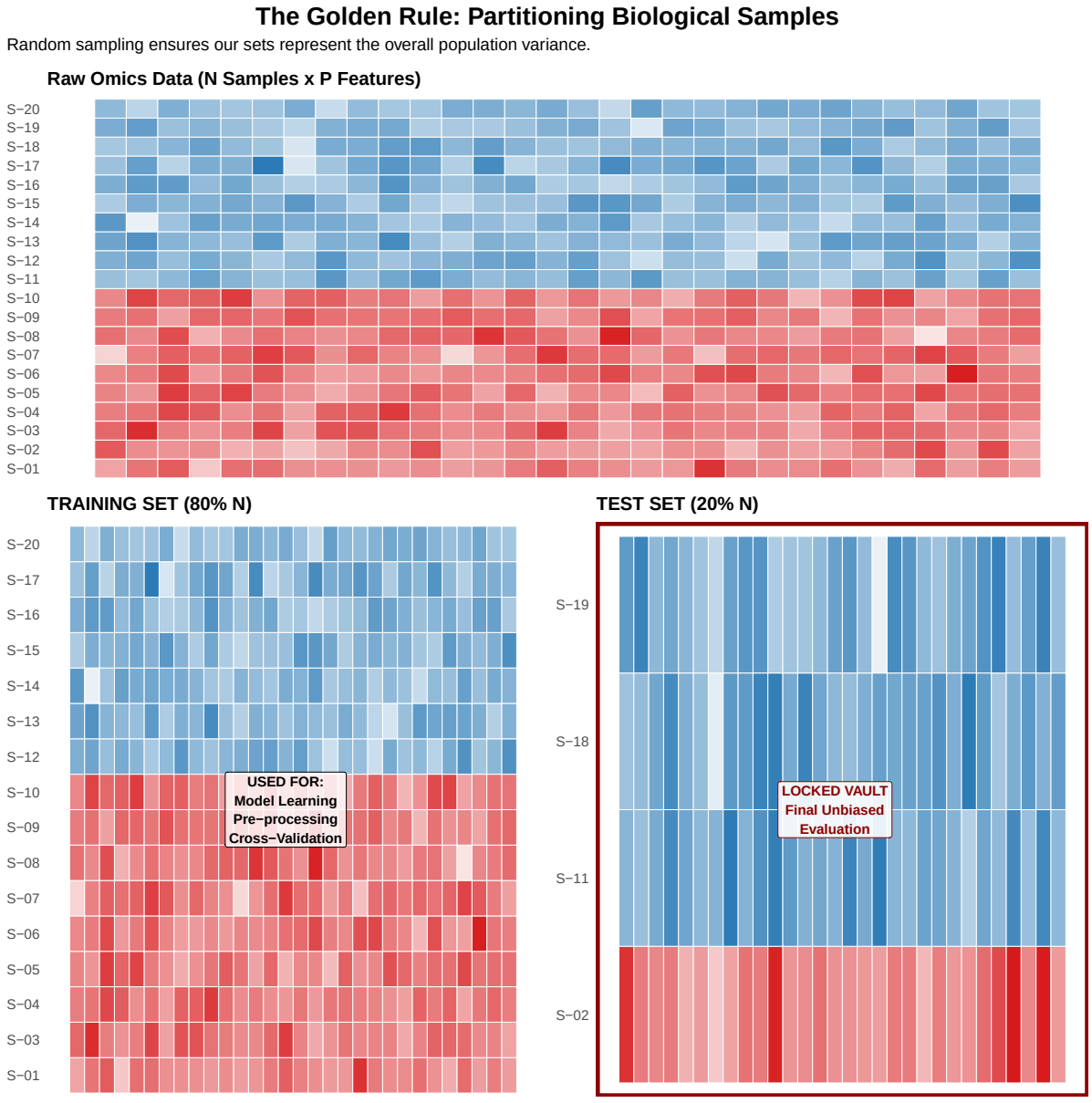


Figure 2.1: Data Spending: Training vs. Testing Sets

2.4.2 Stratified Sampling

In several biological contexts, the classes we are trying to predict are imbalanced, which can lead to misleading performance metrics if not handled properly. Menaing that one class (e.g., healthy controls) may vastly outnumber another (e.g., patients with a rare disease). If we randomly split the data, we risk creating training and testing sets that do not accurately represent the overall class distribution. This can lead to models that perform well on the majority class but poorly on the minority class, which is often of greater interest in biological studies.

To address this, we use **Stratified Sampling** to ensure that the proportion of the outcome (e.g., disease status) is preserved in both the training and testing sets.

In R, we will use the same function as before, but with a new argument.

```
# Split the data into training and testing sets
library(tidymodels)
set.seed(42) # For reproducibility
data_split <- initial_split(df_ml_fpkm,
                            prop = 0.8,
                            strata = sample_type)
train_data_chol <- training(data_split)
test_data_chol  <- testing(data_split)

cat("Training samples:", nrow(train_data_chol), "\nTesting samples:", nrow(test_data_chol))
```

```
Training samples: 35
Testing samples: 9
```

```
cat("Proportion of sample types in training set:\n")
```

```
Proportion of sample types in training set:
```

```
print(prop.table(table(train_data_chol$sample_type)))
```

```
      Primary Tumor Solid Tissue Normal
                0.8                 0.2
```

```r
cat("Proportion of sample types in testing set:\n")
```

```
Proportion of sample types in testing set:
```

```r
print(prop.table(table(test_data_chol$sample_type)))
```

```
      Primary Tumor Solid Tissue Normal
          0.7777778           0.2222222
```

2.5 Pre-processing for Biological Data

Biological data is rarely "model-ready." We must address two major issues: **Scale** and **Skewness**.

2.5.1 Scaling and Centering

Most ML algorithms (like SVM or Lasso) use distance-based metrics. If *Gene A* has expression values in the thousands and *Gene B* in the decimals, the model will unfairly prioritize *Gene A*.

2.5.2 Near-Zero Variance (NZV)

In RNA-seq, many genes show little to no variation across samples. These are "noise" for a predictive model and increase the "Curse of Dimensionality."

2.5.3 The recipe Approach

In `{tidymodels}`, we define a "recipe" – a blueprint of transformations.

```r
library(tidymodels)
# Define a recipe for pre-processing
# We will scale, center, and remove near-zero variance predictors

# Define a recipe for pre-processing
# Focus specifically on numeric predictors to avoid scaling the outcome factor
ml_recipe <- recipe(sample_type ~ ., data = train_data_chol) %>%
```

```
  step_zv(all_predictors()) %>%            # Remove zero variance (constant) genes
  step_nzv(all_numeric_predictors()) %>%   # Remove genes with very little variation
  step_normalize(all_numeric_predictors()) # Z-score transformation (mean=0, sd=1)

# Statistical Note for Students:
# We prep() using ONLY train_data_chol to ensure the mean and SD
# are not influenced by our test set (avoiding "Data Leakage").
ml_prep <- prep(ml_recipe, training = train_data_chol)

# Apply (bake) to both sets
train_processed <- bake(ml_prep, new_data = NULL) # NULL defaults to the training data
test_processed  <- bake(ml_prep, new_data = test_data_chol)

# Check dimensions: How many "noisy" genes did we drop?
cat("Original Training Dim:", dim(train_data_chol), "\n")
```

```
Original Training Dim: 35 8647
```

```
cat("Processed Training Dim:", dim(train_processed), "\n")
```

```
Processed Training Dim: 35 8647
```

```
# How about the test set?
cat("Original Testing Dim:", dim(test_data_chol), "\n")
```

```
Original Testing Dim: 9 8647
```

```
cat("Processed Testing Dim:", dim(test_processed), "\n")
```

```
Processed Testing Dim: 9 8647
```

With our data now properly split and pre-processed, we still have an issue: our training set is small. To build a robust model, we need to maximize the utility of our training data without overfitting. This is where **Cross-Validation** comes into play.

2.6 Cross-Validation: Maximizing Training Data Utility

Cross-Validation (CV) is a resampling technique used to evaluate ML models on a limited data sample. The most common form is **k-Fold Cross-Validation**. In k-Fold CV, the training data is divided into k subsets (or "folds"). The model is trained on *k-1* folds and validated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once (Figure 2.2). The final performance metric is averaged across all folds.

Term	Definition
Fold	A subset of the training data.
Resampling	Repeating the split process.
Hyperparameter	Settings of the model (not learned from data).

To implement k-Fold CV in R, we use the `vfold_cv()` function from `{rsample}`.

```
# Create stratified 5-fold cross-validation
set.seed(123)
cv_folds <- vfold_cv(train_data_chol,
                     v = 5)

cv_folds
```

```
#  5-fold cross-validation
# A tibble: 5 x 2
  splits         id
  <list>         <chr>
1 <split [28/7]> Fold1
2 <split [28/7]> Fold2
3 <split [28/7]> Fold3
4 <split [28/7]> Fold4
5 <split [28/7]> Fold5
```

2.6.1 Stratified K-Fold: Ensuring Biological Representation

As we have discussed in Section Section 2.4, biological datasets are often imbalanced. If we use standard K-Fold CV, we risk an iteration where the "Assessment" fold accidentally contains zero cases of a specific disease subtype. **Stratified K-Fold** ensures that each internal fold preserves the same ratio of classes as the original training set (Figure 2.3).

In R, implementing this is as simple as adding the `strata` argument to your resampling function:

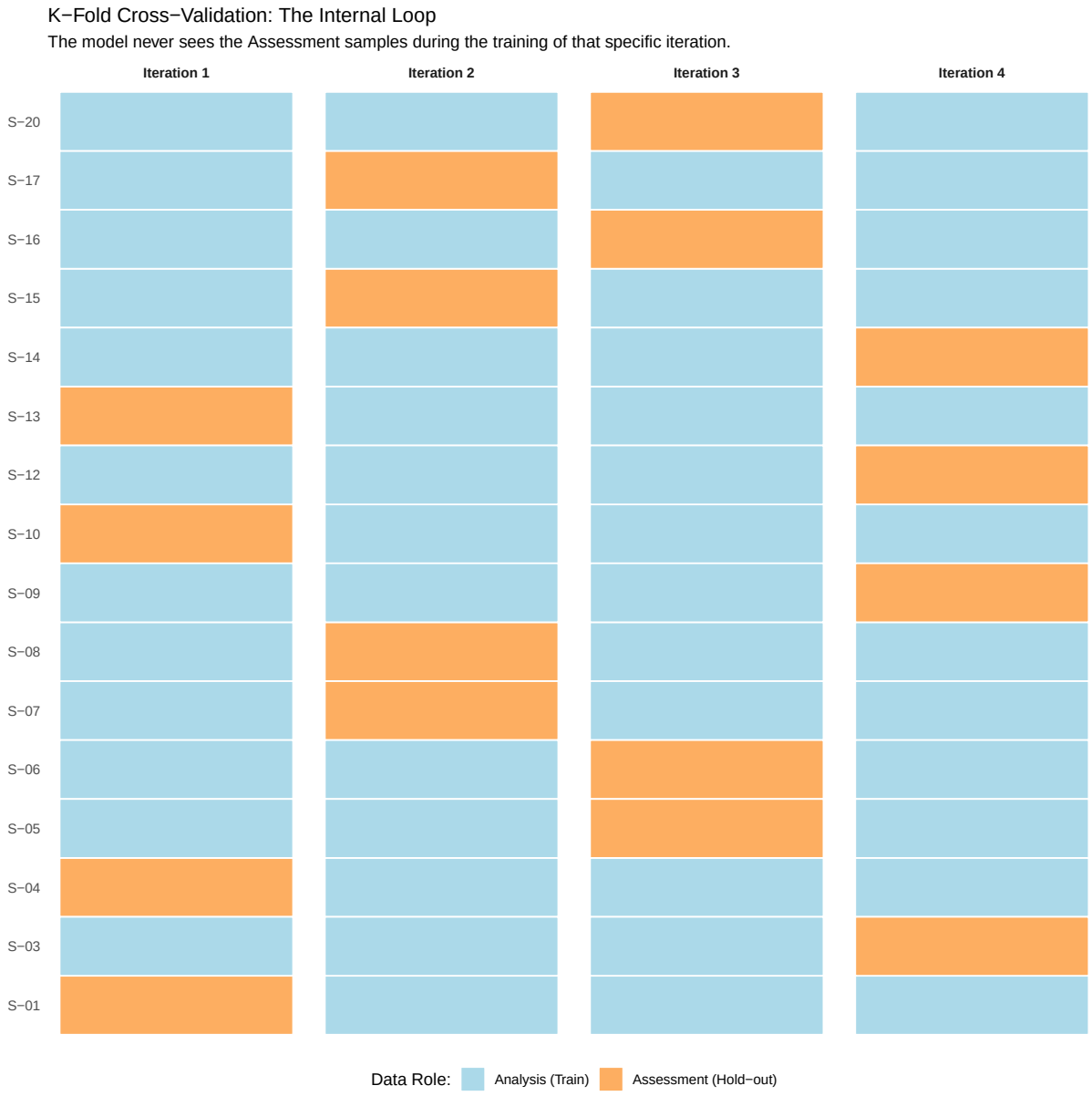


Figure 2.2: K-Fold Cross-Validation: The Internal Loop

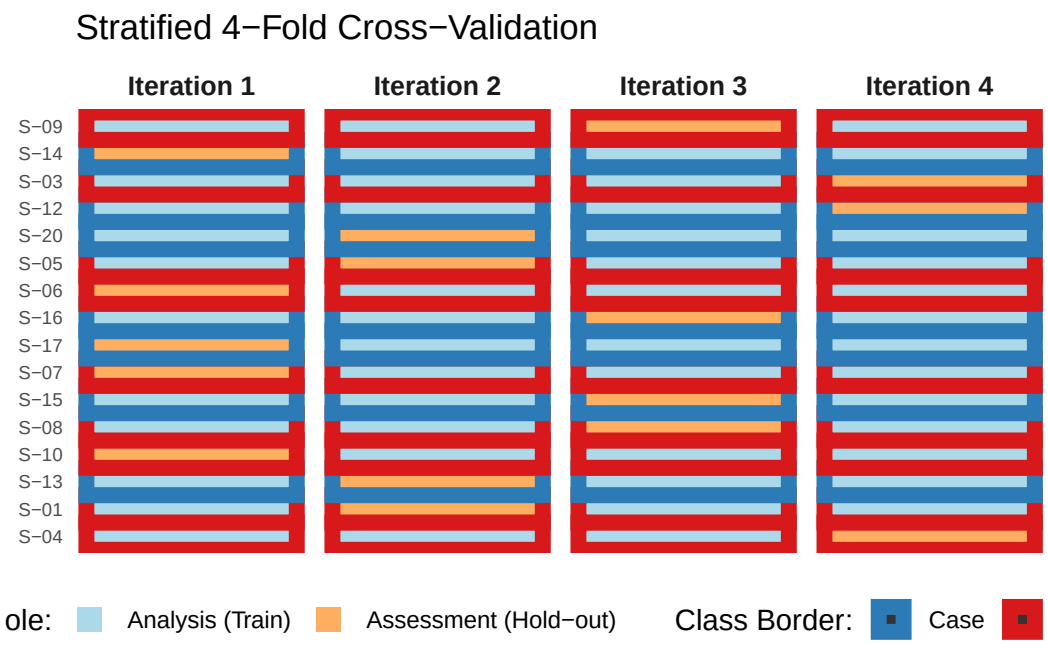


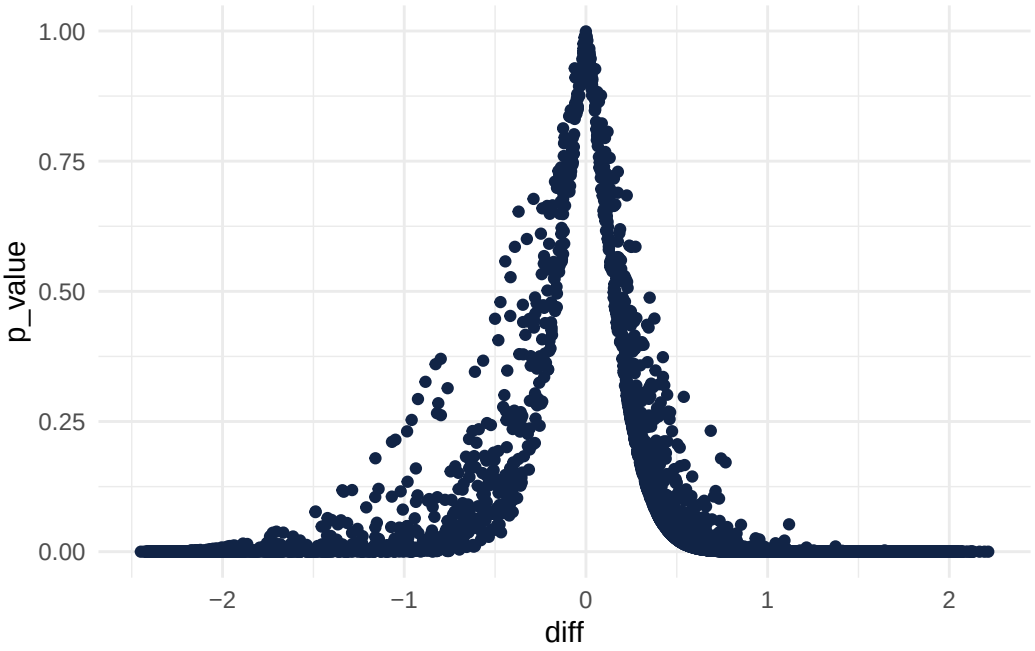
Figure 2.3: Stratified K-Fold Cross-Validation

```
# Create stratified 5-fold cross-validation
set.seed(12345)
cv_folds <- vfold_cv(train_data_chol,
                     v = 5,
                     strata = sample_type)
cv_folds
```

```
#  5-fold cross-validation using stratification
# A tibble: 5 x 2
  splits         id
  <list>         <chr>
1 <split [27/8]> Fold1
2 <split [27/8]> Fold2
3 <split [28/7]> Fold3
4 <split [29/6]> Fold4
5 <split [29/6]> Fold5
```

```
###### Before start, let us do a quick t-test just to check

# Perform t-tests for each gene
t_test_results <- train_processed %>%
```

```r
  pivot_longer(cols = -sample_type,
               names_to = "Gene",
               values_to = "Expression") %>%
  group_by(Gene) %>%
  summarize(
    p_value = t.test(Expression ~ sample_type)$p.value,
    mean_tumor = mean(Expression[sample_type == "Primary Tumor"]),
    mean_normal = mean(Expression[sample_type == "Solid Tissue Normal"]),
    diff = mean_tumor - mean_normal
  ) %>%
  arrange(p_value)

# Visualize using Esquisse :)

ggplot(t_test_results) +
  aes(x = diff, y = p_value) +
  geom_point(colour = "#112446") +
  theme_minimal()
```



2.7 Summary and Concluding Remarks

In this Chapter, we have established the "infrastructure" of a machine learning project. We moved beyond simple data loading to a rigorous pipeline that respects biological complexity and statistical integrity.

2.7.1 Key Takeaways

- **The Mindset Shift**: We moved from p-values (inference) to Generalization Error (prediction).
- **Data Spending**: We learned that the Test Set is a "Locked Vault" that must remain untouched until the very end to prevent **Data Leakage**.
- **Feature Engineering**: Using `{tidymodels}` recipes, we automated the removal of Near-Zero Variance (NZV) genes and normalized high-throughput counts to prevent feature scale bias.
- **Validation**: We implemented Cross-Validation to maximize our small biological sample size (n) while maintaining a "mock" testing environment.

2.7.2 Final Remarks

You now have a "processed" dataset (`train_processed`) and a validation strategy (`cv_folds`). However, we haven't actually looked at our data yet. In the next chapter, we will explore **Unsupervised Learning**. We will use Dimensionality Reduction (PCA, tSNE, UMAP) to see if our biological groups (Cases vs. Controls) naturally separate before we ever try to "force" a model to learn them.

Part II

The ML Workflow

3 Unsupervised Machine Learning - Dimensionality Reduction and Clustering

In biological research, unsupervised learning serves as the primary tool for **exploratory data analysis** (EDA). Unlike supervised methods that require prior labeling, unsupervised algorithms **identify inherent structures, patterns, and anomalies within high-dimensional datasets without external guidance**. For datasets such as the TCGA-CHOL transcriptomics cohort, these techniques are essential for assessing data quality, identifying technical artifacts (batch effects), and discovering novel biological subgroups.

3.1 The Geometry of High-Dimensional Biological Data

Transcriptomics data typically resides in a space where $p \gg n$. Visualizing 20,000 dimensions is physically impossible; therefore, we rely on dimensionality reduction to project this information into a 2D or 3D manifold.

The primary objectives in this phase are:

1. **Feature Compression**: Reducing the number of variables while retaining the maximum amount of biological variance.
2. **Noise Reduction**: Filtering out stochastic variation that does not contribute to the global structure of the data.
3. **Visualization**: Mapping samples into a coordinate system where proximity represents biological similarity.

3.2 Principal Component Analysis (PCA)

PCA is a **linear transformation** that identifies the axes (Principal Components) along which the variance of the data is maximized. It is an orthogonal transformation, meaning each subsequent component is **uncorrelated** with the previous ones.

3.2.1 Mathematical Intuition

Each Principal Component (PC) is a linear combination of the original variables:

$$PC_i = \phi_{1i}X_1 + \phi_{2i}X_2 + \cdots + \phi_{pi}X_p$$

Where ϕ represents the "loadings" or the weight of each gene's contribution to that component.

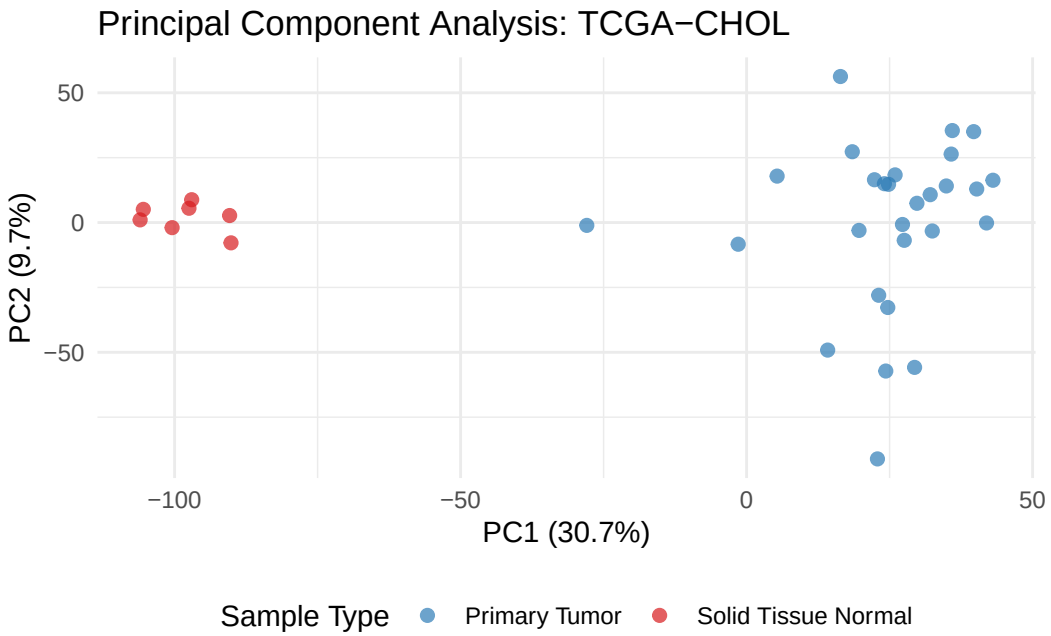
3.2.2 Implementation in `tidymodels`

We utilize the `{embed}` package extension for `{tidymodels}` to incorporate PCA directly into our reproducible pipeline.

```
library(embed)
```

```
Warning: package 'embed' was built under R version 4.4.3
```

```
# Define the PCA Recipe
pca_rec <- recipe(sample_type ~ .,
                  data = train_processed) %>%
  step_pca(all_predictors(), num_comp = 5)

# Train the recipe on the CHOL training set
pca_estimates <- prep(pca_rec)

# Extract the coordinates for visualization
pca_plot_data <- juice(pca_estimates)

# Visualization of PC1 vs PC2
pca_plot_data %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6", "Solid Tissue Normal" = "#d7191
  theme_minimal() +
  labs(title = "Principal Component Analysis: TCGA-CHOL",
       x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
       y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
       color = "Sample Type") +
  theme(legend.position = "bottom")
```



When we use the PCA for dimensionality reduction, we can visualize the first two principal components (PC1 and PC2) to assess the separation between "Primary Tumor" and "Solid Tissue Normal" samples. The percentage of variance explained by each component is also displayed on the axes, providing insight into how much of the original data's variability is captured in this 2D representation. Note that, in this case, we are able to see a clear separation between the two sample types, indicating that the PCA has successfully captured the underlying structure of the data. However, PCA may not always be sufficient to capture complex non-linear relationships, which is where manifold learning techniques like t-SNE and UMAP come into play.

3.3 Non-Linear Dimensionality Reduction

While PCA is restricted to linear projections, biological systems often exhibit **non-linear relationships** and hierarchical structures that require manifold learning. These techniques aim to preserve the high-dimensional proximity of samples in a low-dimensional space.

3.3.1 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE is a probabilistic technique specifically designed to visualize high-dimensional clusters. It converts Euclidean distances between samples into conditional probabilities that represent similarities.

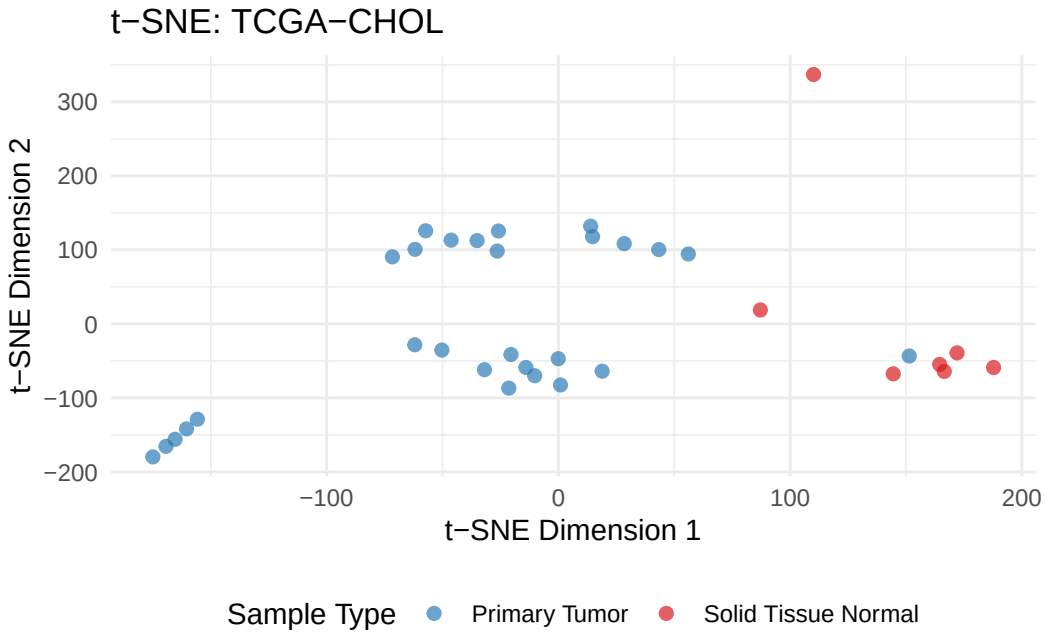
- **Mechanism**: It uses a Student's t-distribution in the low-dimensional space to alleviate the "crowding problem," where points in the center of a cluster tend to collapse onto each other.
- **Perplexity**: The most critical hyperparameter. It balances the attention between local and global aspects of the data. Meaning that, a low perplexity (e.g., 5) focuses on local structure, while a high perplexity (e.g., 50) captures more global relationships. Values between 5 and 50 are typical for biological datasets.
- **Limitation**: t-SNE often fails to preserve global structure; the distance between clusters in the plot is not always meaningful.

```
library(Rtsne)
# Prepare data for t-SNE
# Assuming 'train_data_chol' is a data frame with the sample type in the first column and ge
# We will use the gene expression data for t-SNE
tsne_data <- train_processed %>%
  select(-sample_type) %>%
  as.matrix()
# Run t-SNE
# Set a random seed for reproducibility
set.seed(123)
tsne_result <- Rtsne(tsne_data,
                     perplexity = 1, # Play with the perplexity parameter to see how it affe
                     verbose = TRUE,
                     max_iter = 1000)
```

```
Performing PCA
Read the 35 x 35 data matrix successfully!
Using no_dims = 2, perplexity = 1.000000, and theta = 0.500000
Computing input similarities...
Building tree...
Done in 0.00 seconds (sparsity = 0.142041)!
Learning embedding...
Iteration 50: error is 79.651283 (50 iterations in 0.00 seconds)
Iteration 100: error is 64.081333 (50 iterations in 0.00 seconds)
Iteration 150: error is 68.740650 (50 iterations in 0.00 seconds)
Iteration 200: error is 68.047803 (50 iterations in 0.00 seconds)
Iteration 250: error is 69.798722 (50 iterations in 0.00 seconds)
Iteration 300: error is 1.806691 (50 iterations in 0.00 seconds)
Iteration 350: error is 1.979903 (50 iterations in 0.00 seconds)
Iteration 400: error is 1.668650 (50 iterations in 0.00 seconds)
Iteration 450: error is 1.884399 (50 iterations in 0.00 seconds)
```

```
Iteration 500: error is 1.806041 (50 iterations in 0.00 seconds)
Iteration 550: error is 1.615140 (50 iterations in 0.00 seconds)
Iteration 600: error is 1.624831 (50 iterations in 0.00 seconds)
Iteration 650: error is 1.338770 (50 iterations in 0.00 seconds)
Iteration 700: error is 1.266358 (50 iterations in 0.00 seconds)
Iteration 750: error is 0.807965 (50 iterations in 0.00 seconds)
Iteration 800: error is 1.257645 (50 iterations in 0.00 seconds)
Iteration 850: error is 0.972807 (50 iterations in 0.00 seconds)
Iteration 900: error is 1.026641 (50 iterations in 0.00 seconds)
Iteration 950: error is 1.005357 (50 iterations in 0.00 seconds)
Iteration 1000: error is 0.972446 (50 iterations in 0.00 seconds)
Fitting performed in 0.02 seconds.
```

```r
# Create a data frame for plotting
tsne_plot_data <- data.frame(
  X = tsne_result$Y[, 1],
  Y = tsne_result$Y[, 2],
  sample_type = train_data_chol$sample_type
)

# Visualization of t-SNE results
# Using ggplot2 for visualization

tsne_plot_data %>%
  ggplot() +
  aes(x = X, y = Y, color = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6",
                                "Solid Tissue Normal" = "#d7191c")) +
  theme_minimal() +
  labs(title = "t-SNE: TCGA-CHOL",
       x = "t-SNE Dimension 1",
       y = "t-SNE Dimension 2",
       color = "Sample Type") +
  theme(legend.position = "bottom")
```



Note that t-SNE can be computationally intensive, especially with large datasets, and the results can vary between runs if the random seed is not set. Additionally, while t-SNE can reveal clusters, it may not accurately represent the distances between those clusters, which is a critical consideration when interpreting the results in a biological context. Moreover, we can clearly see that t-SNE has successfully separated the "Primary Tumor" and "Solid Tissue Normal" samples, but the relative positioning of the clusters does not necessarily reflect their biological relationships.

3.3.2 Uniform Manifold Approximation and Projection (UMAP)

UMAP has largely superseded t-SNE in computational biology (particularly in single-cell transcriptomics) due to its superior speed and better preservation of global data topology.

3.3.2.1 The Theoretical Foundation

UMAP is grounded in Riemannian geometry and algebraic topology. It assumes that the data is uniformly distributed on a locally connected manifold. The algorithm constructs a fuzzy simplicial set representation of this manifold and then finds a low-dimensional layout that has the most similar topological structure.

3.3.2.2 Advantages for Biological Data

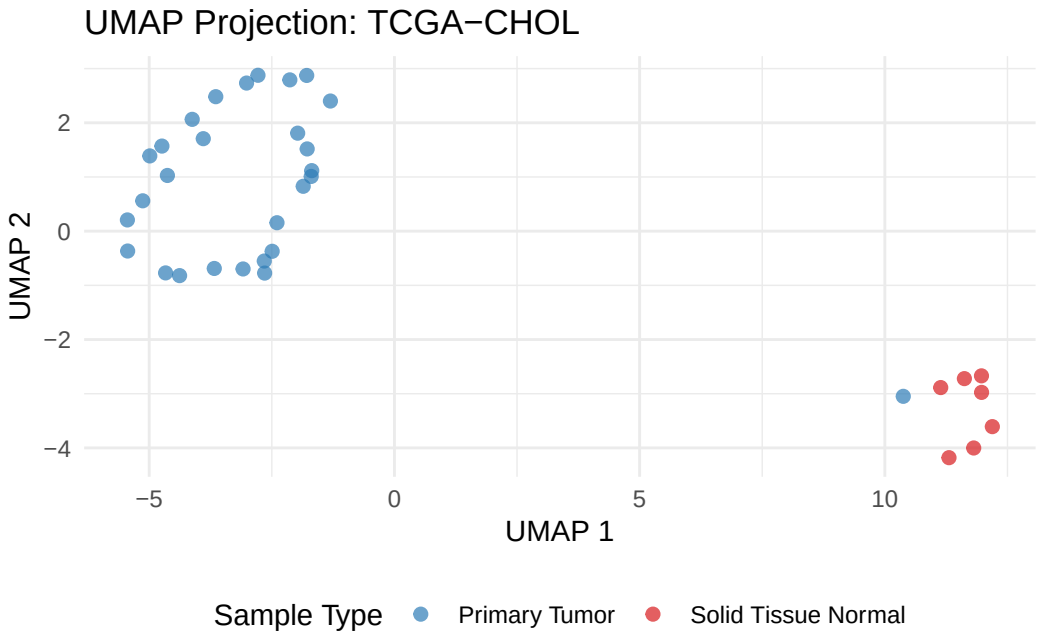
- **Global vs. Local Balance**: Unlike t-SNE, the relative distances between clusters in a UMAP plot often reflect the biological distance between groups (e.g., developmental trajectories or metabolic similarity).
- **Performance**: UMAP is significantly faster and more memory-efficient than t-SNE, making it suitable for large-scale multi-omic integrations.
- **Consistency**: UMAP is more stable across different runs, provided a random seed is set.

3.3.2.3 Hyperparameter Optimization

1. **`n_neighbors`**: This defines the size of the local neighborhood used to learn the manifold. Small values focus on fine-grained local structure (e.g., identifying rare cell subtypes), while large values focus on the global structure of the entire dataset.
2. **`min_dist`**: Determines how tightly the algorithm is allowed to pack points together. Lower values lead to tighter, more discrete clusters, while higher values preserve the broad sense of the data's shape.

```
library(tidymodels)
library(embed)

# Define UMAP recipe
umap_rec <- recipe(sample_type ~ ., data = train_processed) %>%
  step_umap(
    all_predictors(),
    num_comp = 2, # Plotting in 2D, but try 3D as well
    neighbors = 5, # Try different values (5, 10, 15) to see how it affects the visualizatio
    min_dist = 0.1 # Try different values (0.1, 0.5, 0.9) to see how it affects the clusteri
  )

# Train recipe
umap_estimates <- prep(umap_rec)

# Extract coordinates
umap_plot_data <- juice(umap_estimates)

# Visualization
umap_plot_data %>%
  ggplot() +
  aes(x = UMAP1, y = UMAP2, color = sample_type) +
```

```
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6",
                                "Solid Tissue Normal" = "#d7191c")) +
  theme_minimal() +
  labs(title = "UMAP Projection: TCGA-CHOL",
       x = "UMAP 1",
       y = "UMAP 2",
       color = "Sample Type") +
  theme(legend.position = "bottom")
```



In this UMAP visualization, we can observe that the "Primary Tumor" and "Solid Tissue Normal" samples are well-separated, and the relative distances between clusters may provide insights into their biological relationships. We do find, however, one PT within the ST cluster. UMAP's ability to preserve both local and global structures makes it a powerful tool for exploratory data analysis in high-dimensional biological datasets.

3.4 Clustering

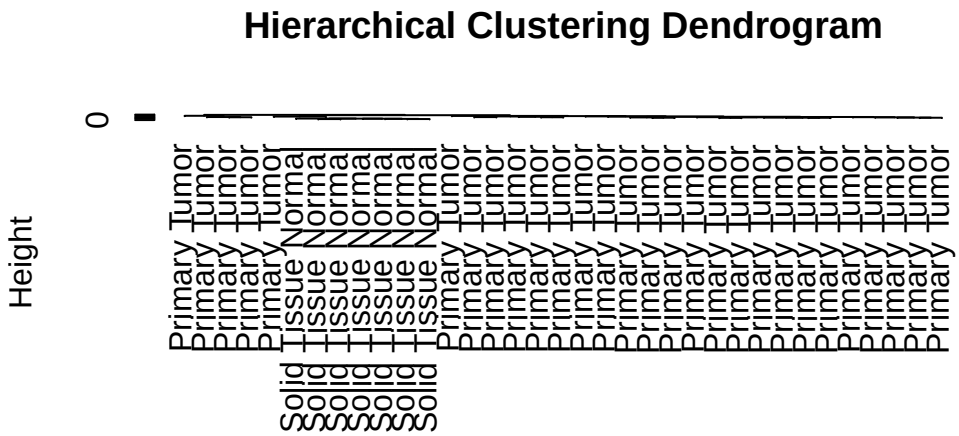
Clustering algorithms are used to group samples based on their similarity in the high-dimensional space. Common methods include hierarchical clustering, k-means, and density-based clustering (e.g., DBSCAN). These techniques can help identify subgroups of samples that may correspond to distinct biological states or disease subtypes.

3.5 Hierarchical Clustering

Hierarchical clustering builds a tree-like structure (dendrogram) that represents the nested grouping of samples based on their pairwise distances. The choice of distance metric (e.g., Euclidean, Manhattan) and linkage method (e.g., complete, average) can significantly influence the resulting clusters.

In R, we can perform hierarchical clustering using the `hclust` function, and visualize the results with a dendrogram.

```
# Compute distance matrix
# A common choice for gene expression data is the Euclidean distance
distance_matrix <- dist(
  train_processed %>%
    select(-sample_type),
  method = "minkowski") # different distance metrics can be tried (euclidean, manhattan, mink
## Euclidean is used for continuous data, while Manhattan can be more robust to outliers.
## Minkowski is a generalization of both

# Perform hierarchical clustering
# We can use the complete linkage method, which considers the maximum distance between points
hc <- hclust(distance_matrix,
             method = "complete")
# Plot the dendrogram

plot(hc,
     labels = train_processed$sample_type,
     main = "Hierarchical Clustering Dendrogram",
     xlab = "",
     sub = "")
```



In this dendrogram, samples that are more similar to each other (based on their gene expression profiles) will be grouped together. We can visually inspect the clusters to see if they correspond to the "Primary Tumor" and "Solid Tissue Normal" sample types. Additionally, we can cut the dendrogram at a specific height to define distinct clusters and analyze their biological significance.

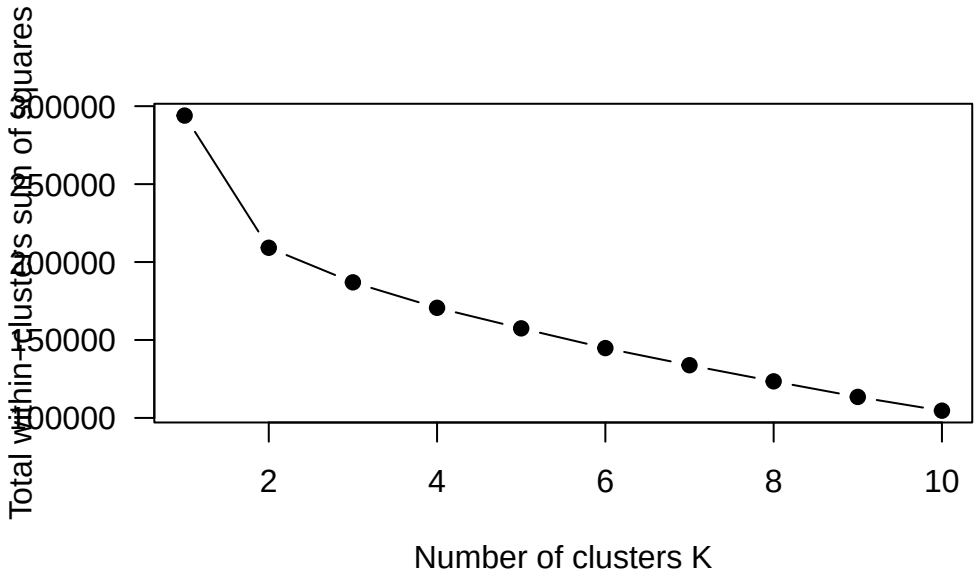
> **Note**: We can also use the Hierarchical Cluter to confirm the results from the Supervised Machine Learning chapter, where we will build predictive models based on the insights gained from our unsupervised analyses. By comparing the clusters identified through hierarchical clustering with the predictions from our supervised models, we can validate the biological relevance of our findings and potentially uncover novel subtypes or patterns within the data.

3.5.1 K-Means Clustering

K-means clustering partitions the data into K distinct clusters by minimizing the within-cluster sum of squares. It is a simple and efficient algorithm but requires the number of clusters (K) to be specified in advance, which can be a limitation when the optimal number of clusters is unknown.

A way to identify the optimal number of clusters is to use the *Elbow Method*, which plots the total within-cluster sum of squares against the number of clusters and looks for an "elbow" point where the rate of decrease sharply changes.

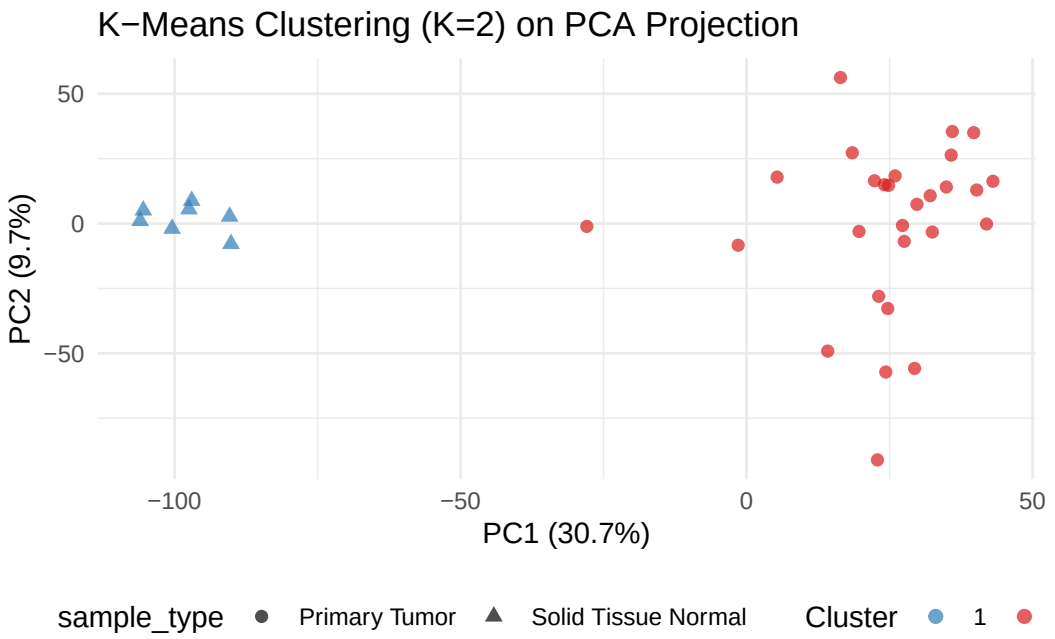
```
# Determine the optimal number of clusters using the Elbow Method
# We will compute the total within-cluster sum of squares for a range of K values
wss <- sapply(1:10, function(k) {
  kmeans(train_processed %>%
           select(-sample_type),
         centers = k,
         nstart = 25)$tot.withinss
})
# Plot the Elbow Method
plot(1:10, wss,
     type = "b",
     las = 1,
     pch = 19,
     xlab = "Number of clusters K",
     ylab = "Total within-clusters sum of squares")
```



Once we have determined the optimal number of clusters (let us use K=2), we can perform K-means clustering and visualize the results.

```
# Perform K-means clustering with K=2
set.seed(123)
kmeans_result <- kmeans(train_processed %>%
                          select(-sample_type),
                        centers = 2,
                        nstart = 25)

cluster <- as.factor(kmeans_result$cluster)
# Visualize the clusters
# We can use PCA to visualize the clusters in a 2D space

pca_plot_data %>%
  mutate(cluster = cluster) %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = cluster, shape = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("1" = "#2c7bb6",
                                "2" = "#d7191c")) +
  theme_minimal() +
  labs(title = "K-Means Clustering (K=2) on PCA Projection",
       x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
       y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
       color = "Cluster") +
```

```
  theme(legend.position = "bottom")
```



In this visualization, we can see how the K-means algorithm has partitioned the samples into two clusters. We can also compare these clusters with the original sample types (Primary Tumor vs Solid Tissue Normal) to assess how well the clustering corresponds to known biological categories. There are few misclassifications, but overall, the clusters seem to align well with the sample types, indicating that K-means has successfully captured the underlying structure of the data.

3.5.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based clustering algorithm that identifies clusters based on the density of data points. It is particularly effective at identifying clusters of arbitrary shape and can handle noise (outliers) in the data. DBSCAN requires two parameters: `eps`, which defines the radius of the neighborhood around a point, and `minPts`, which specifies the minimum number of points required to form a dense region.

```
library(dbscan)
```

```
Warning: package 'dbscan' was built under R version 4.4.3
```

```
Attaching package: 'dbscan'
```

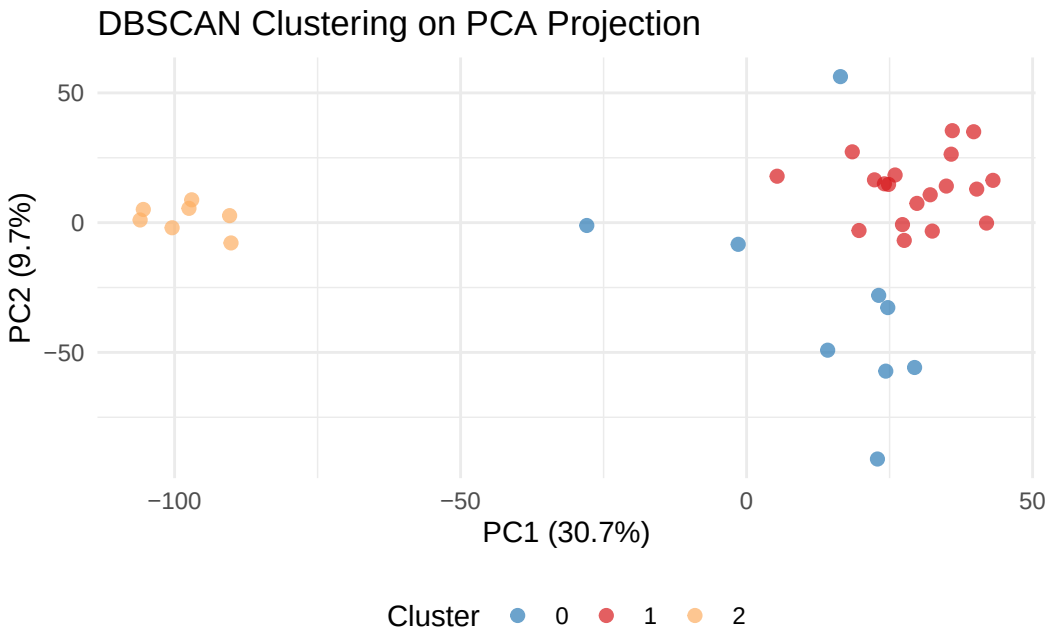
```
The following object is masked from 'package:stats':

    as.dendrogram
```

```r
# Prepare data for DBSCAN
# We will use the PCA-reduced data for DBSCAN to reduce computational complexity
dbscan_data <- pca_plot_data %>%
  select(PC1, PC2) %>%
  as.matrix()
# Run DBSCAN
# Set eps to a value that captures the local density of points and minPts to a value that re
dbscan_result <- dbscan(dbscan_data,
                        eps = 20,
                        minPts = 5,
                        borderPoints = T
)
dbscan_result
```

```
DBSCAN clustering for 35 objects.
Parameters: eps = 20, minPts = 5
Using euclidean distances and borderpoints = TRUE
The clustering contains 2 cluster(s) and 9 noise points.

 0  1  2
 9 19  7

Available fields: cluster, eps, minPts, metric, borderPoints
```

```r
# Add cluster assignments to the original data
pca_plot_data$cluster <- as.factor(dbscan_result$cluster)
# Visualize the DBSCAN clusters
pca_plot_data %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = cluster) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("0" = "#2c7bb6", # Cluster 0 (noise)
                                "1" = "#d7191c", # Cluster 1
                                "2" = "#fdae61", # Cluster 2
                                "3" = "#abdda4", # Cluster 3
                                "4" = "#2b83ba")) + # Cluster 4
  theme_minimal() +
  labs(title = "DBSCAN Clustering on PCA Projection",
```

```
      x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
      y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
      color = "Cluster") +
  theme(legend.position = "bottom")
```



When applying DBSCAN, we cannot identify clusters based on the density of points in the PCA-reduced space. The algorithm will classify points as core points (part of a cluster), border points (on the edge of a cluster), or noise (outliers). In this example, we can see that DBSCAN has identified only noise and no clusters. Depending on the chosen parameters, DBSCAN may classify some samples as noise, which can be useful for identifying outliers in the data.

3.6 Conclusion

We have explored various unsupervised machine learning techniques for dimensionality reduction and clustering in the context of high-dimensional biological data. PCA provided a linear projection that captured the major sources of variance, while t-SNE and UMAP offered non-linear embeddings that revealed more complex relationships between samples. Clustering algorithms like hierarchical clustering, K-means, and DBSCAN allowed us to identify groups of similar samples based on their gene expression profiles.

We will now proceed to the Supervised Machine Learning chapter, where we will build predictive models based on the insights gained from our unsupervised analyses.

4 Supervised Machine Learning

The "Supervised Machine Learning" chapter will cover various algorithms and techniques used in supervised learning, where the model is trained on labeled data. It is important to keep in mind that we now have a label for each observation, and the label is what we are trying to predict, therefore, if this is a classification problem, the label is a class, and if this is a regression problem, the label is a continuous value.

> **Note**: The quality of your label is crucial for the performance of your model. If your label is noisy or inaccurate, your model will struggle to learn the underlying patterns in the data, and it will likely perform poorly on unseen data. Therefore, it is important to ensure that your labels are as accurate and clean as possible before training your model.

4.1 Introduction

There are many algorithms and techniques used in supervised machine learning, and the choice of which one to use depends on the specific problem you are trying to solve, the size and complexity of your dataset, and the computational resources available to you. In this chapter, we will cover some of the most commonly used algorithms and techniques in supervised machine learning, including: Random Forests, Gradient Boosting, Logistic Regression, Regression, Regularization Techniques, and Support Vector Machines, and will discuss some of the Model Interpretability.

The coice of which algorithm to use will depend on the specific problem you are trying to solve, and the nature of your label. For example, if you are trying to solve a classification problem, you may want to use algorithms such as Random Forests, Gradient Boosting, Logistic Regression, Support Vector Machines, or K-Nearest Neighbors. If you are trying to solve a regression problem, you may want to use algorithms such as Linear Regression, Ridge Regression, Lasso Regression, or Neural Networks. It is important to keep in mind that there is no one-size-fits-all algorithm for supervised machine learning, and it is often necessary to try multiple algorithms and compare their performance on your specific problem to find the best one for your needs. Additionally, it is important to consider the interpretability of your model, as some algorithms may be more interpretable than others, which can be important for understanding the underlying patterns in your data and for communicating your results to others. For example, if we are trying to better characterize the underlying transcriptomic

relationships between different genes in our cancer example, it makes more sense for us to have a model that is more interpretable, such as a Random Forest or a Logistic Regression, rather than a Neural Network, which will not directly explain which genes - and their associated weights - to our disease. On the other hand, if we are trying to achieve the highest possible accuracy in our predictions, and we are less concerned about interpretability, such as, I need to develop a diagnostic tool to predict if a patient has or not the disease, and I am not concerned about the underlying mechanism, then a Neural Network may be a better choice. Ultimately, the choice of which algorithm to use will depend on the specific problem you are trying to solve, and the trade-offs between accuracy and interpretability that you are willing to make.

> **Note**: Aways keep in mind the questions you are trying to answer with your model, and the specific problem you are trying to solve, as this will guide you in choosing the right algorithm for your needs. Additionally, it is important to consider the computational resources available to you, as some algorithms may be more computationally expensive than others, and may not be feasible to run on large datasets or with limited resources.

4.2 Overall Model Evaluation

Before we dive into the specific algorithms, it is important to discuss how we will evaluate the performance of our models. There are many metrics that can be used to evaluate the performance of a supervised machine learning model, and the choice of which one to use depends on the specific problem you are trying to solve, and the nature of your label. For classification problems, common metrics include **accuracy**, **precision**, **recall**, and **F1-score**. For regression problems, common metrics include **mean squared error** and **mean absolute error**. It is important to choose the right metric for your specific problem, as different metrics may give different insights into the performance of your model.

The Chapter 5 covers in more detail the different metrics that can be used to evaluate the performance of a supervised machine learning model, and how to interpret them. It is important to keep in mind that no single metric can capture all aspects of the performance of a model, and it is often necessary to use multiple metrics to get a complete picture of how well your model is performing. Additionally, it is important to consider the trade-offs between different metrics, as improving one metric may come at the cost of another metric. For example, improving precision may come at the cost of recall, and vice versa. Therefore, it is important to carefully consider which metrics are most relevant for your specific problem, and to use them in conjunction with each other to evaluate the performance of your model.

If you are not familiar with the different metrics used to evaluate the performance of a supervised machine learning model, I recommend that you read the Chapter 5 before proceeding with the rest of this chapter, as it will provide you with the necessary background to understand how to evaluate the performance of your models.

4.3 Decision Trees

Let us start with the most basic algorithm in supervised machine learning, which is the *Decision Tree*. A Decision Tree is a simple algorithm that can be used for both classification and regression problems. It works by recursively splitting the data into subsets based on the values of the features, and it makes predictions by following the path of the tree from the root to a leaf node, where a prediction is made based on the majority class (for classification) or the average value (for regression) of the observations in that leaf node.

The goal of the Decision Tree algorithm is to find the best splits of the data that **maximize the separation between the classes** (for classification) or **minimize the mean squared error** (for regression). The algorithm uses a greedy approach, where it selects the best split at each step without considering the global structure of the tree. *This can lead to overfitting*, where the tree becomes too complex and captures noise in the data rather than the underlying patterns. To prevent overfitting, we can use techniques such as **pruning**, which involves removing branches of the tree that do not contribute significantly to the predictive power of the model.

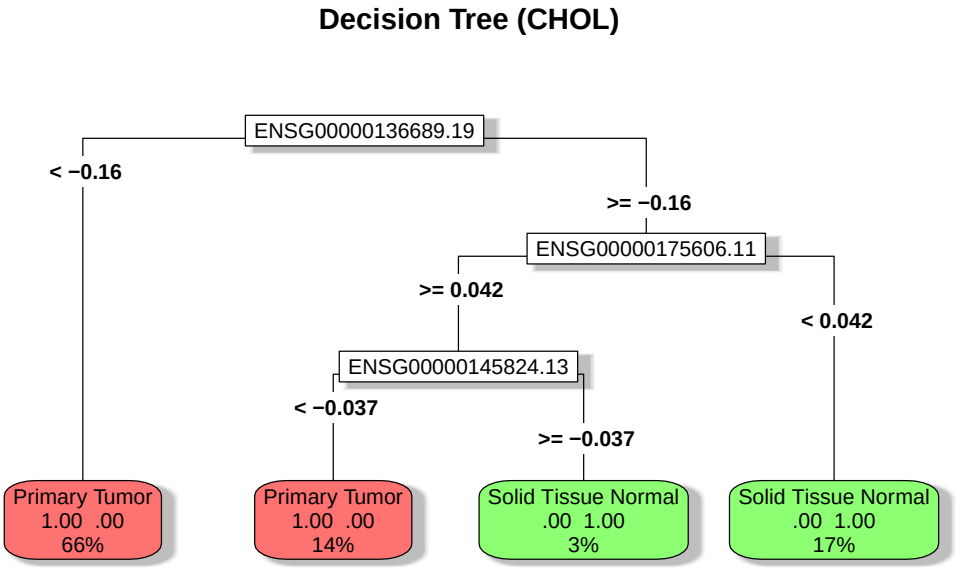
How do we determine the best split of the data? For classification problems, we can use metrics such as **Gini impurity** or **Information Gain** to evaluate the quality of a split. For regression problems, we can use metrics such as **Mean Squared Error** or **Mean Absolute Error** to evaluate the quality of a split. The algorithm will select the split that maximizes the separation between the classes (for classification) or minimizes the mean squared error (for regression).

4.3.1 Running a Decision Tree in R: Usual Framework

To run a Decision Tree in R, we can use the `rpart` package, which provides a simple interface for fitting decision trees. We can also use the `{tidymodels}` framework, which is fancier and easier to reproduce. Here is an example of how to fit a decision tree for a classification problem:

```
library(rpart)
library(rpart.plot)

### Create a mini Train, so it does not take a long time to run
main_genes <- t_test_results %>%
  mutate(p_adj = p.adjust(p_value, method = "BH")) %>%
  filter(p_adj > 0.8) %>%
  head(50) %>%
  pull(Gene)

train_processed_mini <- train_processed %>%
  select(sample_type, main_genes)
```

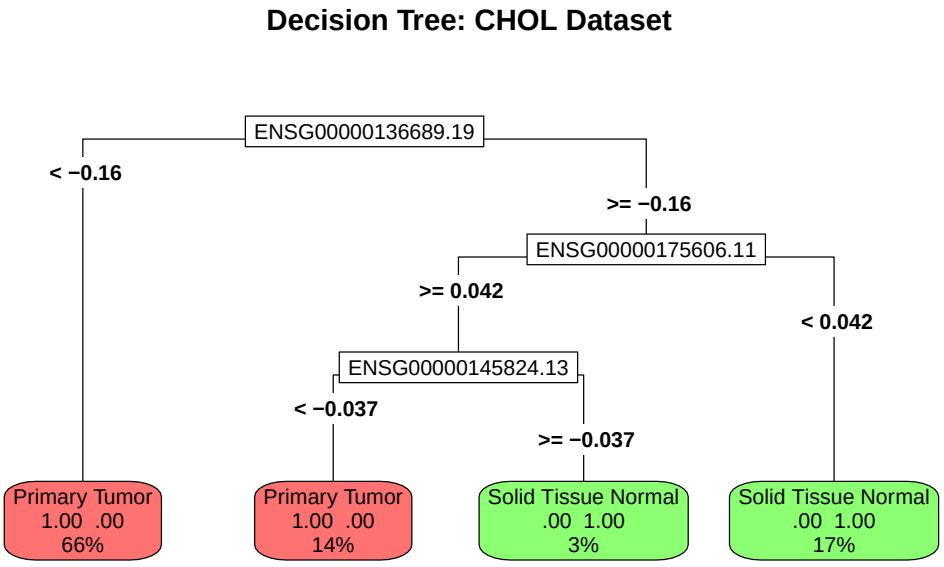
```r
# 1. Prepare the Data.
# We have already done it ;)

# 2. Factorize
train_processed_mini$sample_type <- as.factor(train_processed_mini$sample_type)
train_processed_mini$sample_type <- as.factor(train_processed_mini$sample_type)

# 3. Fit Model with Complexity Control
# cp (complexity parameter) helps prevent the tree from getting too "wild"
tree_model <- rpart(sample_type ~ .,
                    data = train_processed_mini,
                    method = "class",
                    control = rpart.control(cp = 0.001,
                                            minsplit = 2))

# 4. Plot
# 'type = 5' shows the split labels clearly; 'extra = 104' shows probabilities and percentage
rpart.plot(tree_model,
           type = 5,
           extra = 104,
           box.palette = "RdYlGn",
           shadow.col = "gray",
           main = "Decision Tree (CHOL)")
```



4.3.2 Running a Decision Tree in R: {tidymodels}

```r
# 1. Define the Recipe (Pre-processing)
# We select our outcome and predictors, then filter for the top most variable genes
tree_recipe <- recipe(sample_type ~ .,
                      data = train_processed_mini)
full_recipe <- recipe(sample_type ~ .,
                                     data = train_processed )
# 2. Define the Model Specification
# Here we specify 'rpart' as the engine for a classification task
tree_spec <- decision_tree(cost_complexity = 0.01,
                           tree_depth = 5,
                           min_n = 2
                           ) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# 3. Create a Workflow
# This bundles the recipe and the model together
tree_workflow <- workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

# 4. Fit the model
tree_fit <- tree_workflow %>%
  fit(data = train_processed)

# 5. Extract the fitted model to plot the tree
extract_fit_engine(tree_fit) %>%
  rpart.plot(type = 5,
             extra = 104,
             box.palette = "RdYlGn",
             main = "Decision Tree: CHOL Dataset")
```

```
Warning: Cannot retrieve the data used to build the model (so cannot determine roundint and i
To silence this warning:
    Call rpart.plot with roundint=FALSE,
    or rebuild the rpart model with model=TRUE.
```



In this example, we are using the `sample_type` as the label, which is a binary variable indicating whether the sample is a tumor or a normal tissue. The tree will show us how the different genes are used to split the data and how they contribute to the prediction of the sample type.

- **How do we read the plot**: The root node label shows the transcript that is used to split the data, and the value of the split. The left branch corresponds to samples that have a value of the transcript less than or equal to the split value, and the right branch corresponds to samples that have a value of the transcript greater than the split value. The leaf nodes show the predicted class for each group of samples, as well as the proportion of samples in each class, the percentage value indicates the percentage of samples in that node.

4.3.3 Model Evaluation

When evaluating our model ability to predict the sample type, we should always use our test dataset, which is a subset of our data that was not used to train the model. This will give us an unbiased estimate of the performance of our model on unseen data. We can use metrics such as accuracy, precision, recall, and F1-score to evaluate the performance of our model on the test dataset.

```
# Make predictions on the test dataset
test_processed$sample_type <- as.factor(test_processed$sample_type)
predictions <- predict(tree_model, newdata = test_processed, type = "class")
# Evaluate the accuracy of the model
# Create a confusion matrix
```

```r
caret::confusionMatrix(predictions, test_processed$sample_type)
```

```
Confusion Matrix and Statistics

                    Reference
Prediction           Primary Tumor Solid Tissue Normal
  Primary Tumor                  5                   1
  Solid Tissue Normal            2                   1

               Accuracy : 0.6667
                 95% CI : (0.2993, 0.9251)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.8822

                  Kappa : 0.1818

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.7143
            Specificity : 0.5000
         Pos Pred Value : 0.8333
         Neg Pred Value : 0.3333
             Prevalence : 0.7778
         Detection Rate : 0.5556
   Detection Prevalence : 0.6667
      Balanced Accuracy : 0.6071

       'Positive' Class : Primary Tumor
```

4.4 Random Forests

The Random Forest algorithm is an ensemble learning method that combines **multiple decision trees** to improve the predictive performance and reduce overfitting. It works by creating a **large number of decision trees**, each trained on a **random subset of the data and a random subset of the features**. The final prediction is made by aggregating the predictions of all the individual trees, either by taking the majority vote (for classification) or by averaging the predictions (for regression). The Random Forest algorithm has several advantages over a single decision tree. First, it can handle a large number of features and is less likely to

Table 4.1: Interpretation Guide for Caret Confusion Matrix Outputs

Caret Confusion Matrix Glossary

Standardized Performance Evaluation Metrics

Metric Name	Brief Explanation
Accuracy	The overall proportion of correct predictions (both positive and nega
No Information Rate (NIR)	The accuracy achievable by always predicting the majority class.
P-Value [Acc > NIR]	Statistical test checking if the model is significantly better than a bli
Kappa	A measure of agreement between predicted and observed, adjusted fo
Sensitivity (Recall)	The ability to correctly identify the 'Positive' class (e.g., catching the
Specificity	The ability to correctly identify the 'Negative' class (e.g., identifying
Pos Pred Value (Precision)	How often the model is correct when it predicts the 'Positive' class.
Balanced Accuracy	The arithmetic mean of Sensitivity and Specificity; vital for imbalanc
Mcnemar's Test P-Value	Tests for systematic bias—whether the model fails more in one direc

overfit the data compared to a single decision tree. Second, it can provide estimates of **feature importance**, which can help us understand which features are most important for making predictions. Third, it can handle missing values and maintain accuracy even when a large proportion of the data is missing. However, it can be computationally expensive to train a large number of trees, especially on large datasets, and it may not be as interpretable as a single decision tree.

4.4.1 Running a Random Forest in R: The Usual Framework

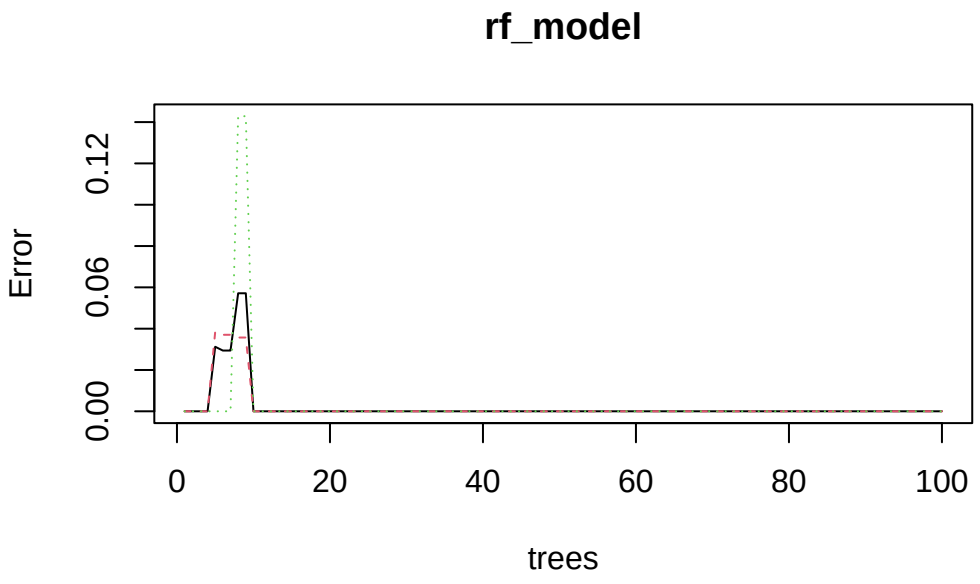
To run a Random Forest in R, we can use the `randomForest` package, which provides a simple interface for fitting random forest models. Here is an example of how to fit a random forest for a classification problem:

```
library(randomForest)
set.seed(12345) # For reproducibility
# Fit a random forest model
#
rf_model <- randomForest(sample_type ~ .,
                         mtry = 10, # Number of variables randomly sampled as candidates at e
                         data = train_processed,
                         maxnodes = 5,
                         nodesize = 2,
                         ntree = 100) # Number of trees to grow
```

```r
# Print the random forest model
rf_model
```

```
Call:
 randomForest(formula = sample_type ~ ., data = train_processed,      mtry = 10, maxnodes = 5
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 10

        OOB estimate of  error rate: 0%
Confusion matrix:
                   Primary Tumor Solid Tissue Normal class.error
Primary Tumor                 28                   0           0
Solid Tissue Normal            0                   7           0
```

```r
#######

# Plot the random forest model
plot(rf_model)
```



The visualization produced by `plot(rf_model)` illustrates the *Out-of-Bag (OOB)* error rate as a function of the number of trees in the forest. The black line represents the overall error for the entire model, while the colored lines track the specific error rates for each class (e.g., Primary Tumor vs. Solid Tissue Normal). This plot is essential for diagnosing model convergence: *as*

we add more trees, the error should initially drop and then "plateau" or flatten out. If the lines are still trending downward at 100 trees, it suggests that the model hasn't reached its full potential and **ntree** should be increased. However, if the lines have been horizontal for several hundred trees, it confirms that our forest is stable and further computation would offer diminishing returns.

```
# Get feature importance
importance(rf_model) %>%
  as.data.frame() %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(10) %>%
  knitr::kable(caption = "Top 10 Most Important Features in the Random Forest Model")
```

Table 4.2: Top 10 Most Important Features in the Random Forest Model

	MeanDecreaseGini
ENSG00000196072.12	0.1680000
ENSG00000136872.20	0.1577143
ENSG00000140107.12	0.1508571
ENSG00000101266.19	0.1428571
ENSG00000157657.14	0.1428571
ENSG00000159210.10	0.1428571
ENSG00000168495.13	0.1428571
ENSG00000170638.10	0.1428571
ENSG00000215305.10	0.1428571
ENSG00000035115.22	0.1337143

```
library(ggplot2)

imp_df <- importance(rf_model) %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(20)

ggplot(imp_df) +
  aes(x = reorder(Gene, MeanDecreaseGini),
      y = MeanDecreaseGini) +
  geom_point(size = 3,
             color = "steelblue") +
  geom_segment(aes(x = Gene,
```

```
                    xend = Gene,
                    y = 0,
                    yend = MeanDecreaseGini),
                color = "skyblue") +
coord_flip() +
labs(title = "Variable Importance (Random Forest)",
     x = "Genes",
     y = "Mean Decrease in Gini Index") +
theme_minimal()
```

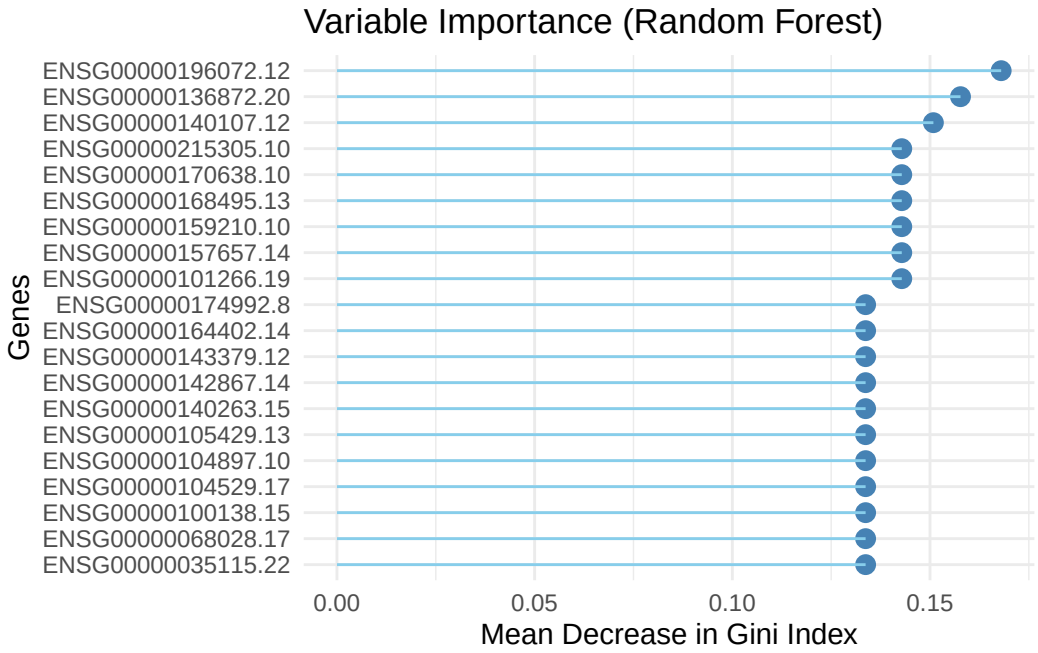


Figure 4.1: Top 20 Gene Predictors by Gini Importance

Note that in this example, we are using the full data, and the sample type as the label. The random forest model will show us how the different genes are used to split the data and how they contribute to the prediction of the sample type. We can also look at the importance of each feature in the random forest, which can help us understand which genes are most important for predicting the sample type. In the previous example, we saw that only one gene was used to split the data in the decision tree, but in the random forest, we can see that multiple genes are used to split the data, and we can also see the importance of each gene in the model.

How to visualize the tree: First of all, when dealing with trees, we do not have a single tree, but rather a forest of trees, and each tree is different from the others, as they are trained on

different subsets of the data and different subsets of the features. Therefore, it is not possible to visualize the entire random forest, but we can visualize individual trees from the forest. The `randomForest` package does not provide a built-in function to visualize individual trees in the forest, but we can use the `getTree` function to extract a single tree from the random forest and then visualize it using the `rpart.plot` package. Here is an example of how to visualize a single tree from the random forest:

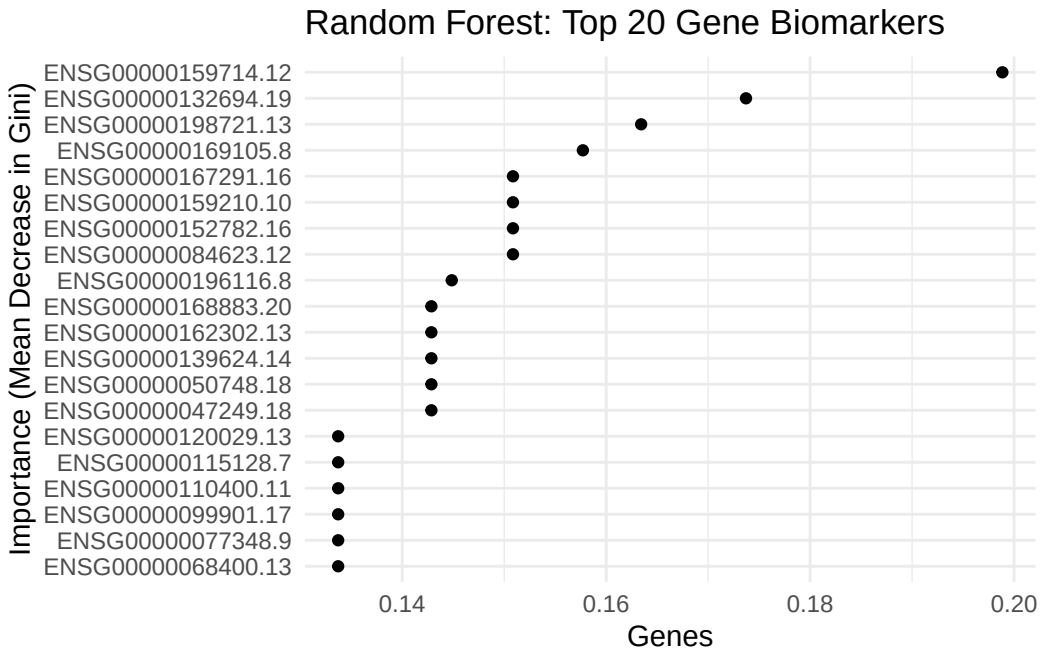
```
# Look at the structure of the 1st tree
# k = 1 is the tree number
# labelVar = TRUE ensures it uses gene names instead of index numbers
randomForest::getTree(rf_model,
                      k = 1,
                      labelVar = TRUE) %>%
  head(10) # Showing the first 10 nodes
```

```
  left daughter right daughter          split var split point status
1             2              3 ENSG00000141279.17    -1.21143      1
2             0              0               <NA>     0.00000     -1
3             0              0               <NA>     0.00000     -1
           prediction
1                <NA>
2 Solid Tissue Normal
3       Primary Tumor
```

4.4.2 Running a Random Forest in R: {tidymodels}

```
library(tidymodels)
set.seed(123) # For reproducibility
## The data step is the same as previously, we need only to update our model specification a

# 1. Define the Random Forest Specification
# We'll grow 100 trees (trees = 100)
# mtry is the number of genes randomly sampled at each split
rf_spec <- rand_forest(
  mtry = tune(),      # We can tune this later
  trees = 100,
  min_n = 2
) %>%
  set_engine("ranger",
             importance = "impurity") %>%
```

```
  set_mode("classification")

# 2. Update the Workflow
# We reuse the 'tree_recipe' from the previous section
rf_workflow <- workflow() %>%
  add_recipe(full_recipe) %>% # Reusing the recipe
  add_model(rf_spec %>% finalize_model(list(mtry = 2))) # Setting a starting mtry

# 3. Fit the Model
rf_fit <- rf_workflow %>%
  fit(data = train_processed)

# 4. Extract the fitted model to get feature importance
rf_fit %>%
  extract_fit_parsnip() %>%
  vip::vip(num_features = 20,
           geom = "point") +
  theme_minimal() +
  labs(title = "Random Forest: Top 20 Gene Biomarkers",
       x = "Importance (Mean Decrease in Gini)",
       y = "Genes")
```



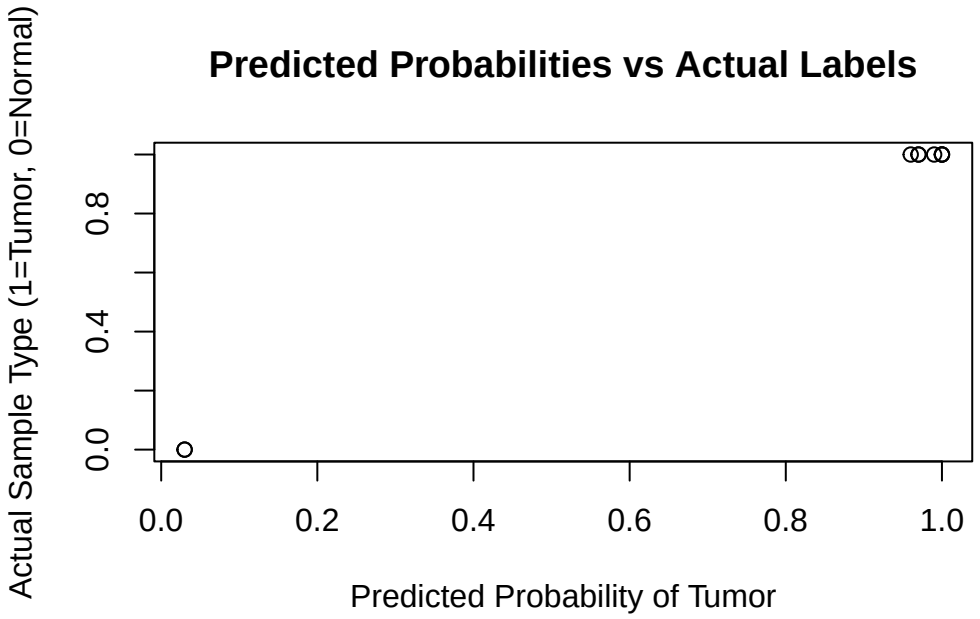
4.4.3 Model Accuracy

To evaluate the accuracy of our random forest model, we can use the `predict` function to make predictions on a test dataset, and then compare the predicted labels to the true labels using metrics such as **accuracy**, **precision**, **recall**, and **F1-score** for classification problems or **mean squared error** and **mean absolute error** for regression problems. Here is an example of how to evaluate the accuracy of a random forest model for a classification problem:

```
# Make predictions on the test dataset
# We will use our test data here
predictions <- predict(rf_model, newdata = test_processed)
# Evaluate the accuracy of the model
# Create a confusion matrix
require(caret)
confusionMatrix(predictions, test_processed$sample_type)
```

```
Confusion Matrix and Statistics

                    Reference
Prediction           Primary Tumor Solid Tissue Normal
  Primary Tumor                  7                   0
  Solid Tissue Normal            0                   2

               Accuracy : 1
                 95% CI : (0.6637, 1)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.1042

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.7778
         Detection Rate : 0.7778
   Detection Prevalence : 0.7778
      Balanced Accuracy : 1.0000

       'Positive' Class : Primary Tumor
```

```
#######
## AUC and AUC-PR
require(ROCR)
# Get predicted probabilities for the positive class
pred_prob <- predict(rf_model,
                     newdata = test_processed,
                     type = "prob")[, 1] # Tumour Prob is the First Column
sample_type <- ifelse(test_processed$sample_type == "Primary Tumor", 1, 0) # Convert to binar

plot(pred_prob, sample_type,
     xlab = "Predicted Probability of Tumor",
     ylab = "Actual Sample Type (1=Tumor, 0=Normal)",
     main = "Predicted Probabilities vs Actual Labels")
```



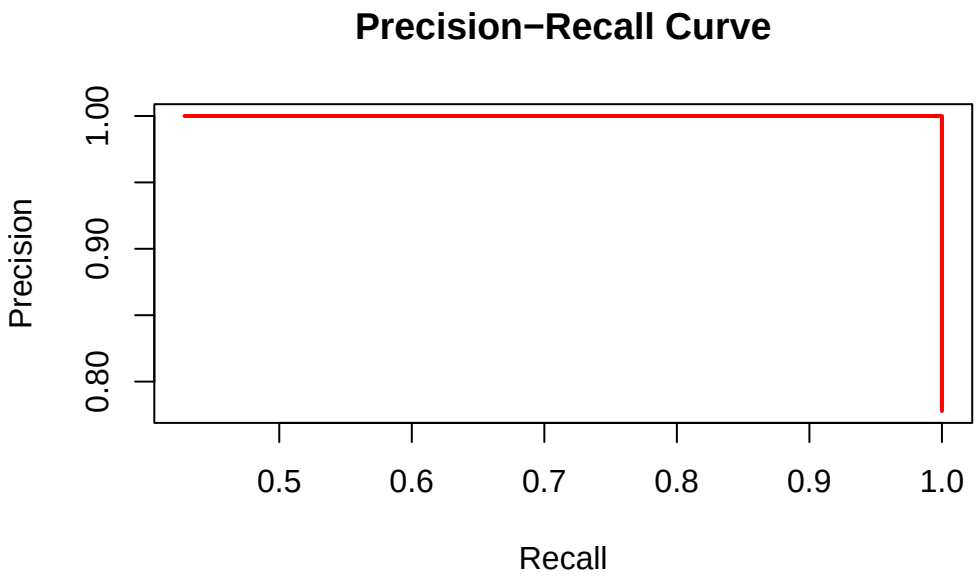

```
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)
# Calculate AUC
auc <- ROCR::performance(pred, measure = "auc")@y.values[[1]]
auc
```

```
[1] 1
```

```
#######
# Calculate AUC-PR
auc_pr <- performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr
```

```
[1] 1
```

```
#######

# plot the Precision-Recall curve

pr_perf <- performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")
```



In conclusion, the Random Forest algorithm is a powerful and versatile method for supervised machine learning that can be used for both classification and regression problems. It can handle a large number of features and is less likely to overfit the data compared to a single decision tree. Additionally, it can provide estimates of feature importance, which can help us understand which features are most important for making predictions. However, it can be computationally expensive to train a large number of trees, especially on large datasets, and it may not be as interpretable as a single decision tree.

4.5 Gradient Boosting

The Gradient Boosting algorithm is another ensemble learning method that combines multiple decision trees to improve the predictive performance. It works by sequentially adding trees to the model, where each tree is trained to correct the errors of the previous tree. The final prediction is made by aggregating the predictions of all the individual trees, either by taking the majority vote (for classification) or by averaging the predictions (for regression). The Gradient Boosting algorithm has several advantages over a single decision tree, including improved predictive performance and reduced overfitting. However, it can be computationally expensive to train a large number of trees, especially on large datasets, and it may not be as interpretable as a single decision tree. The main difference between Random Forests and Gradient Boosting is that Random Forests build trees independently and in parallel, while Gradient Boosting builds trees sequentially, where each tree is trained to correct the errors of the previous tree. This means that Gradient Boosting can potentially achieve better performance than Random Forests, but it can also be more sensitive to overfitting if not properly tuned.

4.5.1 Running a Gradient Boosting Model in R: Basic Framework

To run a Gradient Boosting model in R, we can use the `{xgboost}` package, which provides a simple interface for fitting gradient boosting models. Here is an example of how to fit a gradient boosting model for a classification problem:

```
library(xgboost)
set.seed(12345) # For reproducibility
# Prepare the data for xgboost
# We need to convert the data into a matrix format and the labels into a numeric format
train_matrix <- train_processed_mini %>%
  select(-sample_type) %>% # Exclude the label
  as.matrix()
train_labels <- as.numeric(train_processed_mini$sample_type) - 1 # Convert to numeric (0 and
# Fit a gradient boosting model
gb_model <- xgboost(data = train_matrix,
                    label = as.factor(train_labels),
                    nrounds = 100, # Number of boosting rounds
                    objective = "binary:logistic") # Binary classification
# Print the gradient boosting model
print(gb_model)

#######

# Get feature importance
```

```r
importance_matrix <- xgb.importance(feature_names = colnames(train_matrix), model = gb_model)
importance_matrix %>%
  head(10) %>%
  knitr::kable(caption = "Top 10 Most Important Features in the Gradient Boosting Model")
```

4.5.2 Running a Gradient Boosting in R: {tidymodels}

```r
#######
library(tidymodels)
library(xgboost)
library(vip)

set.seed(123)

# 1. Define the Gradient Boosting Specification
# Note: XGBoost usually needs many more trees than a Random Forest
# because it learns slowly (boosting vs bagging).
gb_spec <- boost_tree(
  min_n = 2,
  mode = "classification",
  mtry = 5,
  trees = 500,
  tree_depth = 5,
  learn_rate = 0.0001
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# 2. Update the Workflow (reusing your tree_recipe)
gb_workflow <- workflow() %>%
  add_recipe(full_recipe) %>%
  add_model(gb_spec)

# 3. Fit the Model
gb_fit <- gb_workflow %>%
  fit(data = train_processed)

# 4. Extract and Plot Importance
# In tidymodels, you simply pass the parsnip object directly to vip().
# It will automatically interface with xgboost and retrieve the feature names.
```

```r
gb_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 20,
      geom = "point",
      aesthetics = list(color = "midnightblue", size = 3)) +
  theme_minimal() +
  labs(title = "Gradient Boosting: Top 20 Gene Biomarkers",
       subtitle = "Importance calculated via Gain",
       x = "Importance",
       y = "Genes")
```

4.5.3 Model Evaluation

Not surprisingly, we can evaluate the accuracy of our gradient boosting model using the same approach as we did for the random forest model. We can use the `predict` function to make predictions on a test dataset. Here is an example of how to evaluate the accuracy of a gradient boosting model for a classification problem:

```r
# Prepare the test data for xgboost
test_matrix <- as.matrix(test_processed %>% select (-sample_type)) # Exclude the label
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert to numeric (0 and 1)
# Make predictions on the test dataset
predictions <- predict(gb_model, newdata = test_matrix)
# Convert predicted probabilities to class labels (0 or 1)
predicted_labels <- ifelse(predictions > 0.5, 1, 0 ) %>%
  as.factor()

# Evaluate the accuracy of the model
caret::confusionMatrix(predicted_labels, as.factor(test_labels))
```

```r
## AUC and AUC-PR
## Get predicted probabilities for the positive class
pred_prob <- predict(gb_model,
                     newdata = test_matrix) # Predicted probabilities for the positive class
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)
# Calculate AUC
auc <- ROCR::performance(pred,
                         measure = "auc")@y.values[[1]]
auc
```

```
#######
# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")
#######
# Calculate AUC-PR
auc_pr <- ROCR::performance(pred,
                            measure = "aucpr")@y.values[[1]]
auc_pr

# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred,
                             measure = "prec",
                             x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")
```

4.6 Regression

A regression model is a type of supervised machine learning algorithm that is used to predict a **continuous outcome variable based on one or more predictor variables**. The goal of a regression model is to find the best-fitting line or curve that describes the relationship between the predictor variables and the outcome variable. There are many different types of regression models, including linear regression, polynomial regression, and logistic regression (which we will see later on). The choice of which regression model to use depends on the specific problem you are trying to solve, and the nature of your data, meaning that, you must know beforehand the data distribution and the relationship between the predictor variables and the outcome variable to choose the appropriate regression model. For example, if you have a linear relationship between the predictor variables and the outcome variable, then a linear regression model may be appropriate. However, if you have a non-linear relationship, then a polynomial regression model may be more appropriate.

Mathematically, a regression model can be represented as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon.$$

Where Y is the outcome variable, $X_1, X_2, ..., X_p$ are the predictor variables, β_0 is the intercept, $\beta_1, \beta_2, ..., \beta_p$ are the coefficients for the predictor variables, and ϵ is the error term.

It is important to note that regression models make several assumptions about the data, including *linearity*, *independence of errors*, *homoscedasticity* (variance is constant), and *normality of errors*. It is important to check these assumptions before fitting a regression model, as violations of these assumptions can lead to biased estimates and incorrect inferences. Additionally, it is important to consider the potential for multicollinearity between the predictor variables, as this can lead to unstable estimates of the coefficients and can affect the interpretability of the model. In addition to that, data should be independent, meaning that the observations should not be correlated with each other. If the data is not independent, then a regression model may not be appropriate, and we may need to use other algorithms that can account for the dependencies between the observations, such as a generalized linear mixed model (GLMM) allowing for both fixed and random effects, and can be used to model data with complex dependencies, such as repeated measures or hierarchical data.

4.6.1 Running a Regression Model in R: Basic Framework

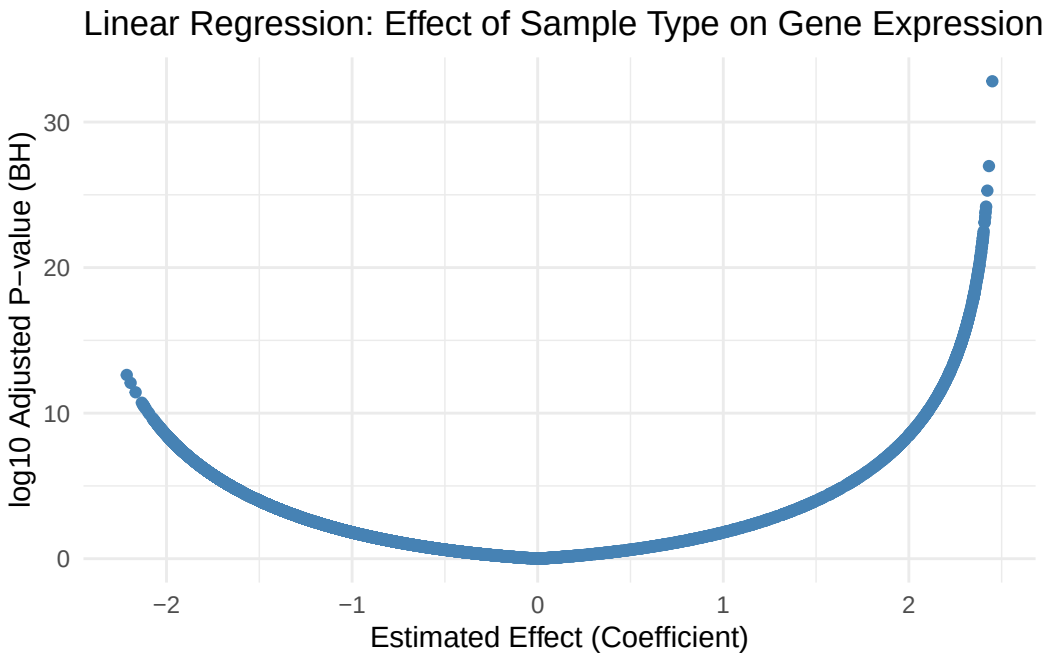
To run a regression model in R, we can use the `lm` function for linear regression, or the `glm` function for generalized linear models (which includes logistic regression). Here is an example of how to fit a linear regression model:

```
# Fit a linear regression model

linear_model <- lm(ENSG00000001629.10 ~ sample_type, data = train_processed)
# Print the linear regression model
summary(linear_model)
```

```

Call:
lm(formula = ENSG00000001629.10 ~ sample_type, data = train_processed)

Residuals:
    Min      1Q  Median      3Q     Max
-1.4458 -0.3428 -0.0210  0.1604  3.2922

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)                         0.2936     0.1541   1.906 0.065382 .
sample_typeSolid Tissue Normal  -1.4682     0.3445  -4.262 0.000159 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8152 on 33 degrees of freedom
Multiple R-squared:  0.355, Adjusted R-squared:  0.3355
F-statistic: 18.17 on 1 and 33 DF,  p-value: 0.000159
```

```r
### Ok, but I don't want to do one gene at a time, or put into a for loop.
require(broom)
fit = train_processed %>%
  pivot_longer(cols = -sample_type,
               names_to = "Gene",
               values_to = "Expression") %>%
  group_by(Gene) %>%
  do(tidy(lm(Expression ~ sample_type, data = .))) %>%
  filter(term != "(Intercept)") %>%
  mutate(p_adj = p.adjust(p.value, method = "BH"))

fit %>%
  ggplot() +
  aes(x = estimate, y = -log10(p_adj)) +
  geom_point(color = "steelblue") +
  theme_minimal() +
  labs(title = "Linear Regression: Effect of Sample Type on Gene Expression",
       x = "Estimated Effect (Coefficient)",
       y = "log10 Adjusted P-value (BH)") +
  theme(legend.position = "bottom")
```



4.6.2 Running a Regression Model in R: {tidymodels}

```
#######
library(tidymodels)
# 1. Define the Recipe

regression_recipe <- recipe(ENSG00000001629.10 ~ sample_type, data = train_processed)

regression_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# 3. Create a Workflow
regression_workflow <- workflow() %>%
  add_recipe(regression_recipe) %>%
  add_model(regression_spec)

# 4. Fit the model
regression_fit <- regression_workflow %>%
  fit(data = train_processed)
```

```r
# 5. Print the model summary
extract_fit_parsnip(regression_fit)
```

```
parsnip model object


Call:
stats::lm(formula = ..y ~ ., data = data)

Coefficients:
                   (Intercept)  sample_typeSolid Tissue Normal
                        0.2936                         -1.4682
```

4.6.3 Model Evaluation

To evaluate the accuracy of a regression model, we can use metrics such as **mean squared error (MSE)**, **mean absolute error (MAE)**, and **R-squared**. Here is an example of how to evaluate the accuracy of a linear regression model:

```r
## We start by making predictions on the test dataset
predictions <- predict(linear_model,
                       newdata = test_processed)

# Calculate Mean Squared Error (MSE)
mse <- mean((predictions - test_processed$ENSG00000001629.10)^2)
mse
```

```
[1] 0.2082772
```

```r
# Calculate Mean Absolute Error (MAE)
mae <- mean(abs(predictions - test_processed$ENSG00000001629.10))
mae
```

```
[1] 0.3658356
```

```r
# Calculate R-squared
ss_total <- sum((test_processed$ENSG00000001629.10 - mean(test_processed$ENSG00000001629.10))
ss_residual <- sum((test_processed$ENSG00000001629.10 - predictions)^2)
r_squared <- 1 - (ss_residual / ss_total)
r_squared
```

```
[1] 0.5113006
```

4.7 Logistic Regression

The Logistic Regression algorithm is a linear model that is used for classification problems. It works by **modeling the relationship between the features and the probability of the positive class (e.g., tumor) using a logistic function**. The key idea behind logistic regression is to model the log-odds of the positive class as a linear combination of the features. The logistic function is used to convert the log-odds into a probability, which can then be used to make predictions. Meaning that, in the end, we are modeling the probability of a sample being a tumor or a normal tissue based on the expression levels of the genes. The model estimates the coefficients for each feature, which can be interpreted as the log-odds of the positive class given a one-unit increase in the feature, while holding all other features constant. Logistic regression is a **simple, yet interpretable model** that can be used for both binary and multiclass classification problems. But, beware of a multiclass logistic regression, which is a generalization of binary logistic regression to handle multiple classes, and it can be often refered as "multinomial logistic regression".

The logistic regression algorithm has several advantages, including its simplicity and interpretability. It can also be regularized to prevent overfitting (we will see it later on), and it can handle both continuous and categorical features. However, it may not perform well when the relationship between the features and the log-odds of the positive class is not linear, or when there are complex interactions between the features. Additionally, logistic regression assumes that the observations are **independent of each other**, which may not always be the case in real-world data. It means that, if we have a dataset where the samples are not independent, such as in a time series or in a spatial dataset, logistic regression may not be the best choice, and we may need to use other algorithms that can account for the dependencies between the observations, such as a generalized linear mixed model (GLMM), which is an extension of a Generalized Linear Model (GLM) that allows for both fixed and random effects, and can be used to model data with complex dependencies, such as repeated measures or hierarchical data.

The mathematical formulation of logistic regression is as follows:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p)}}$$

Where $P(Y = 1|X)$ is the probability of the positive class given the features X, β_0 is the intercept, $\beta_1, \beta_2, ..., \beta_p$ are the coefficients for the features $X_1, X_2, ..., X_p$, and e is the base of the natural logarithm.

It is important to note that the Logistic Regression, oposite to the Random Forest and Gradient Boosting algorithms, do not perform any kind of feature selection, and it will use all the features in the dataset to make predictions. Therefore, it is important to perform feature selection before fitting a logistic regression model, especially when dealing with high-dimensional data, such as gene expression data. This can be done using techniques such as **variance filtering**, **correlation filtering**, or **regularization** (which we will see later on). Additionally, it is

important to check for multicollinearity between the features, as this can lead to unstable estimates of the coefficients and can affect the interpretability of the model.

The stepwise selection method is a common approach for feature selection in logistic regression, where features are added or removed from the model based on some criteria (such as AIC). However, it is important to note that stepwise selection does not always lead to the best model, and it can be prone to overfitting, especially when dealing with high-dimensional data. Oftentimes, it can also select features that are correlated, therefore, it should - as all models - be used with caution.

4.7.1 Running a Logistic Regression Model in R: Basic Framework

The basic framework for running a logistic regression model in R is similar to the one we used for the decision tree and random forest models. We can use the `glm` function to fit a logistic regression model, and then use the `predict` function to make predictions on a test dataset. Here is an example of how to fit a logistic regression model for a classification problem:

```
logistic_model <- glm(sample_type ~ ENSG00000001629.10,
                      data = train_processed,
                      family = binomial)
# Print the logistic regression model
summary(logistic_model)
```

```
Call:
glm(formula = sample_type ~ ENSG00000001629.10, family = binomial,
    data = train_processed)

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)          -20.48      15.13  -1.354    0.176
ENSG00000001629.10   -19.21      13.67  -1.405    0.160

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 35.0282  on 34  degrees of freedom
Residual deviance:  7.7216  on 33  degrees of freedom
AIC: 11.722

Number of Fisher Scoring iterations: 11
```

```r
## Stepwise selection
## # To reduce the amount of genes we test, we will use the genes that were found in the dec

keep_imp = t_test_results %>%
  mutate(p_adj = p.adjust(p_value, method = "BH")) %>%
  filter(p_adj < 0.00001) %>%
  head(50) %>%
  pull(Gene)
train_processed_cl = train_processed %>%
  select(sample_type, all_of(keep_imp))
null_model <- glm(sample_type ~ 1,
                  data = train_processed_cl,
                  family = binomial) # Model with only the intercept
full_model <- glm(sample_type ~ .,
                  data = train_processed_cl,
                  family = binomial) # Model with all features
stepwise_model <- stats::step(null_model,
                              scope = list(lower = null_model,
                                           upper = full_model),
                              direction = "both",
                              trace = 0)
summary(stepwise_model)
```

```
Call:
glm(formula = sample_type ~ ENSG00000117305.15, family = binomial,
    data = train_processed_cl)

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)          -39.76   57039.39  -0.001    0.999
ENSG00000117305.15    37.20   53152.09   0.001    0.999

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3.5028e+01  on 34  degrees of freedom
Residual deviance: 7.7559e-10  on 33  degrees of freedom
AIC: 4

Number of Fisher Scoring iterations: 25
```

4.7.2 Running a Logistic Regression Model in R: {tidymodels}

Oftentimes, we can use the {tidymodels} framework to fit a logistic regression model, which provides a more consistent and reproducible workflow. Here is an example of how to fit a logistic regression model using {tidymodels}:

```
library(tidymodels)
# 1. Define the Recipe (Pre-processing)
log_recipe <- recipe(sample_type ~ .,
                     data = train_processed_cl %>%
                       select(sample_type, 2:5))

# 2. Define the Model Specification
# Here we specify 'glm' as the engine for a classification task
logistic_spec <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")
# 3. Create a Workflow
# adding the stepwise selection is a bit tricky in tidymodels, as it does not have a built-in

logistic_workflow <- workflow() %>%
  add_recipe(log_recipe) %>% # Reusing the recipe with Variance Filtering
  add_model(logistic_spec)

# 4. Fit the model

logistic_fit <- logistic_workflow %>%
  fit(data = train_processed_cl)
```

4.7.3 Model Evaluation

To evaluate the accuracy of our logistic regression model, we can use the `predict` function to make predictions on a test dataset, and then compare the predicted labels to the true labels using the accuracy metrics. Here is an example of how to evaluate the accuracy of a logistic regression model for a classification problem:

```
# Make predictions on the test dataset
predictions <- predict(logistic_model,
                       newdata = test_processed,
                       type = "response")
```

```r
predicted_labels <- ifelse(predictions < 0.5, "Primary Tumor", "Solid Tissue Normal") %>%
  as.factor()

# Evaluate the accuracy of the model
caret::confusionMatrix(predicted_labels, test_processed$sample_type)
```

```
Confusion Matrix and Statistics

                    Reference
Prediction           Primary Tumor Solid Tissue Normal
  Primary Tumor                  7                   0
  Solid Tissue Normal            0                   2

               Accuracy : 1
                 95% CI : (0.6637, 1)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.1042

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.7778
         Detection Rate : 0.7778
   Detection Prevalence : 0.7778
      Balanced Accuracy : 1.0000

       'Positive' Class : Primary Tumor
```

4.8 Regularization Techniques

Often, when dealing with high-dimensional data, such as gene expression data, we may have more features than samples, which can lead to overfitting and poor generalization performance. Having many more features than samples, mathematically, means that we have more parameters to estimate than data points, which can lead to overfitting, mostly, we have an underdetermined

system of equations, where there are infinitely many solutions that can fit the training data perfectly. **Regularization** is the statistical "penalty" we apply to prevent the model from becoming too complex and memorizing the noise in the data. We do this by adding a penalty term to our loss function[^1]. [^1]: The loss function is what the model tries to minimize during training. For example, in linear regression, the loss function is typically the mean squared error (MSE), which measures the average squared difference between the predicted values and the true values.

4.8.1 Understanding Regularization

Regularization techniques are a set of methods that can be used to prevent overfitting by adding a penalty term to the loss function of the model. The most common regularization techniques are **Lasso (L1)** and **Ridge (L2)** regularization, and the **Elastic Net**, which is a combination of Lasso and Ridge regularization.

4.8.2 Lasso (L1) Regularization

LASSO, Least Absolute Shrinkage and Selection Operator, is a regularization technique that adds a penalty term to the loss function of the model, which is proportional to the absolute value of the coefficients. The Lasso penalty **encourages sparsity in the model**, meaning that it set some **coefficients to zero**, which essencially performs a feature selection. This can be particularly useful when dealing with high-dimensional data, as it can help to identify the most important features for making predictions.

Mathematically, the L1 norm is defined as the sum of the absolute values of the coefficients:

$$L1 = \lambda \sum_{j=1}^{p} |\beta_j|.$$

Where λ is the regularization parameter that controls the strength of the penalty, and β_j are the coefficients of the model.

> **Note**: If you do have high multicollinearity in your data, Lasso may **arbitrarily** select one feature from a group of correlated features and set the others to zero, which can lead not only to instability in feature selection, but also to a loss of interpretability, as the selected feature may not be the most biologically relevant one.

4.8.2.1 Understanding Lasso Coefficients and Feature Selection

Before evaluating our results, we must carefully interpret what the coefficients in a Lasso (L1-regularized) logistic regression actually signify.

1. The Log-Odds Interpretation In a Lasso logistic regression, the non-zero coefficients represent the change in the **log-odds** of the outcome (e.g., "Tumor" vs. "Normal") for every one-unit increase in gene expression, assuming all other variables in the model remain constant.

2. The Power of Sparsity The defining feature of Lasso is **sparsity**. By penalizing the absolute magnitude of coefficients, Lasso performs automatic feature selection.

- **Non-zero coefficients**: These identify the "minimal set" of genes required to achieve optimal prediction.
- **Zero coefficients**: These are genes the model deemed redundant or non-informative *in the presence of the other selected genes*.

3. The Magnitude and Direction The **sign** (+/-) tells us the direction of the association: a positive coefficient suggests that higher expression of that gene increases the probability of the sample being a tumor. The **magnitude** reflects the strength of that gene's contribution to the decision boundary.

4. A Warning on Multicollinearity In transcriptomics, genes often operate in tightly correlated networks (multicollinearity). If three genes are perfectly correlated, Lasso will typically pick **only one** and set the others to zero.

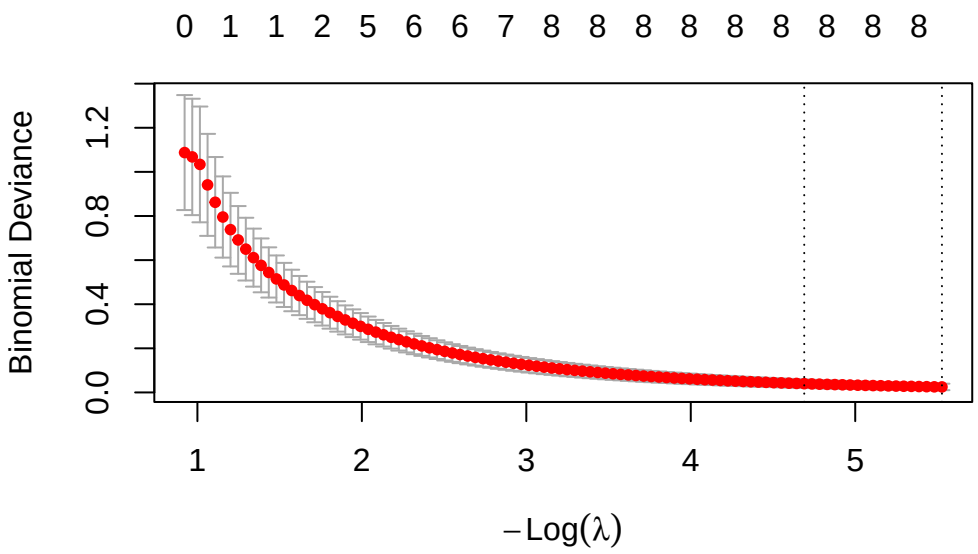
> **Note:** A coefficient of zero does *not* necessarily mean a gene has no biological relationship with the disease; it may simply be highly correlated with another gene already in the model. If your goal is to understand entire biological pathways rather than finding a minimal biomarker panel, you should consider **Ridge Regression** or **Elastic Net**, which distribute the coefficients across correlated groups rather than discarding them.

4.8.2.2 Running a Lasso Regression Model in R: Basic Framework

To run a Lasso regression model in R, we can use the `{glmnet}` package, which provides a simple interface for fitting Lasso regression models. Here is an example of how to fit a Lasso regression model for a classification problem:

```
library(glmnet)
set.seed(123) # For reproducibility
# Prepare the data for glmnet
# Data must be in matrix format and labels must be numeric
train_matrix <- train_processed %>%
```

```
  select(-sample_type) %>%
  as.matrix()
train_labels <- as.numeric(train_processed$sample_type) - 1 # Convert

# Fit a Lasso regression model
# Before fitting a LASSO we have to find the best lambda (regularization parameter) using cr
cv_lasso <- cv.glmnet(train_matrix,
                      train_labels,
                      alpha = 1, # Lasso
                      family = "binomial")
plot(cv_lasso)
```




```
best_lambda <- cv_lasso$lambda.min
best_lambda
```

```
[1] 0.003976754
```

```
lasso_model <- glmnet(train_matrix,
                      train_labels,
                      alpha = 1,
                      lambda = best_lambda,
                      family = "binomial")
# Print the Lasso regression model
# The coefficients of the model can be extracted using the coef function
coef(lasso_model) %>%
```

```
  as.matrix() %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
  filter(s0 != 0) %>%
  arrange(desc(abs(s0))) %>%
  filter(Gene != "(Intercept)") %>%
  knitr::kable(caption = "Most Important Features in the Lasso Regression Model")
```

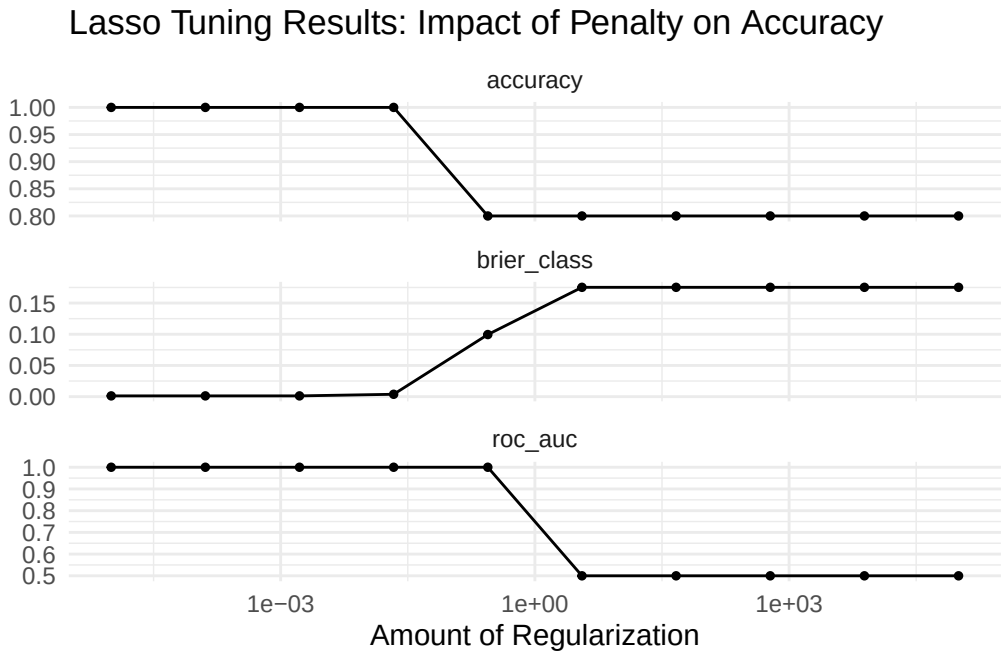
Table 4.3: Most Important Features in the Lasso Regression Model

Gene	s0
ENSG00000204653.10	3.0925746
ENSG00000109576.14	0.4336123
ENSG00000122787.15	0.4300169
ENSG00000213398.8	0.1975483
ENSG00000165140.11	0.1194624
ENSG00000005421.9	0.1178844
ENSG00000055957.11	0.0908859
ENSG00000250056.8	0.0415200
ENSG00000009724.17	0.0394340
ENSG00000101981.12	0.0031845

4.8.2.3 Running a Lasso Regression Model in R: {tidymodels}

```
library(tidymodels)

# 1. Define Lasso Specification
lasso_spec <- logistic_reg(
  penalty = tune(), # We will find this via CV
  mixture = 1       # 1 = Lasso
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# 2. Define Cross-Validation Folds (10-fold)
set.seed(123)
folds <- vfold_cv(train_processed, v = 10)

# 3. Create a Grid of Lambda values to test
```

```r
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 10)

# 4. Run the Tuning
lasso_grid <- tune_grid(
  workflow() %>%
    add_recipe(full_recipe) %>%
    add_model(lasso_spec),
  resamples = folds,
  grid = lambda_grid
)

# 5. Visualize the Tuning Results
# This replaces the plot(cv_lasso) call
autoplot(lasso_grid) +
  theme_minimal() +
  labs(title = "Lasso Tuning Results: Impact of Penalty on Accuracy")
```




```r
#######

# 6. Select the best penalty (highest ROC AUC)
best_penalty <- lasso_grid %>%
  select_best(metric = "roc_auc")
```

```r
# 7. Finalize and Fit
final_lasso <- workflow() %>%
  add_recipe(full_recipe) %>%
  add_model(lasso_spec) %>%
  finalize_workflow(best_penalty) %>%
  fit(data = train_processed)

# 8. Extract Non-Zero Coefficients (Biomarkers)
final_lasso %>%
  extract_fit_parsnip() %>%
  tidy() %>%
  filter(estimate != 0 & term != "(Intercept)") %>%
  arrange(desc(abs(estimate))) %>%
  knitr::kable(caption = "Biomarkers Selected by Lasso Regression")
```

Table 4.4: Biomarkers Selected by Lasso Regression

term	estimate	penalty
ENSG00000204653.10	3.2397359	1e-05
ENSG00000122787.15	0.3480214	1e-05
ENSG00000109576.14	0.2106497	1e-05
ENSG00000005421.9	0.1776182	1e-05
ENSG00000250056.8	0.1627249	1e-05
ENSG00000055957.11	0.1602819	1e-05
ENSG00000101981.12	0.1456805	1e-05
ENSG00000213398.8	0.1190364	1e-05

4.8.2.4 Model Evaluation

To evaluate the accuracy of our Lasso regression model, we can use the `predict` function to make predictions on a test dataset. Here is an example of how to evaluate the accuracy of a Lasso regression model for a classification problem:

```r
# Prepare the test data for prediction
# We need to ensure that the test data has the same features as the training data, and that t

test_matrix <- test_processed %>%
  select(-sample_type) %>% # Exclude the label
  as.matrix()
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert
```

```r
# Make predictions on the test dataset

predictions <- predict(lasso_model,
                       newx = test_matrix,
                       type = "response")
predicted_labels <- ifelse(predictions > 0.5, 1, 0
) %>%
  as.factor()
# Evaluate the accuracy of the model
confusionMatrix(predicted_labels, as.factor(test_labels))
```

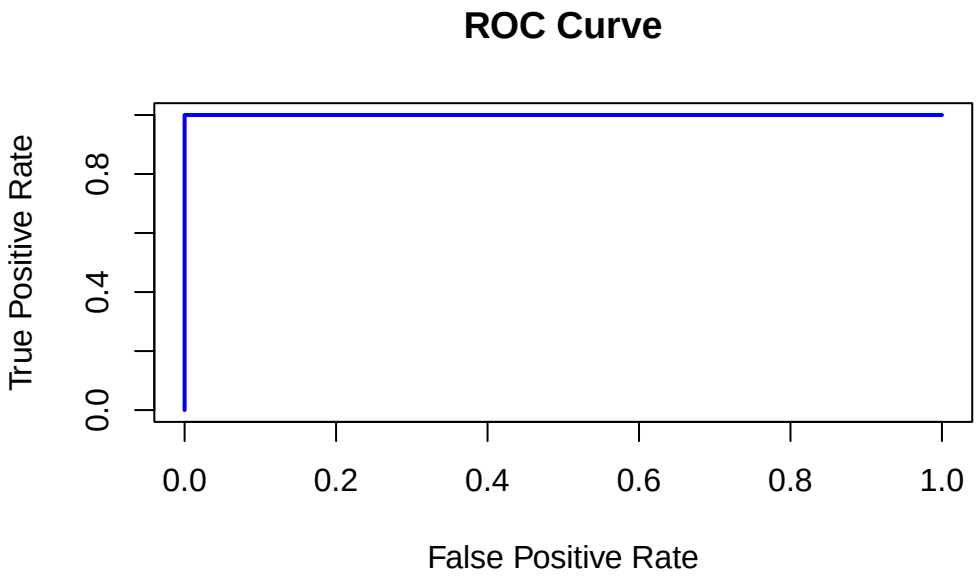
```
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 7 0
         1 0 2

               Accuracy : 1
                 95% CI : (0.6637, 1)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.1042

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.7778
         Detection Rate : 0.7778
   Detection Prevalence : 0.7778
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

```r
## AUC and AUC-PR
# Get predicted probabilities for the positive class
```

```r
pred_prob <- predict(lasso_model, newx = test_matrix, type = "response")
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)

# Calculate AUC
auc <- ROCR::performance(pred,
                         measure = "auc")@y.values[[1]]
auc
```

```
[1] 1
```

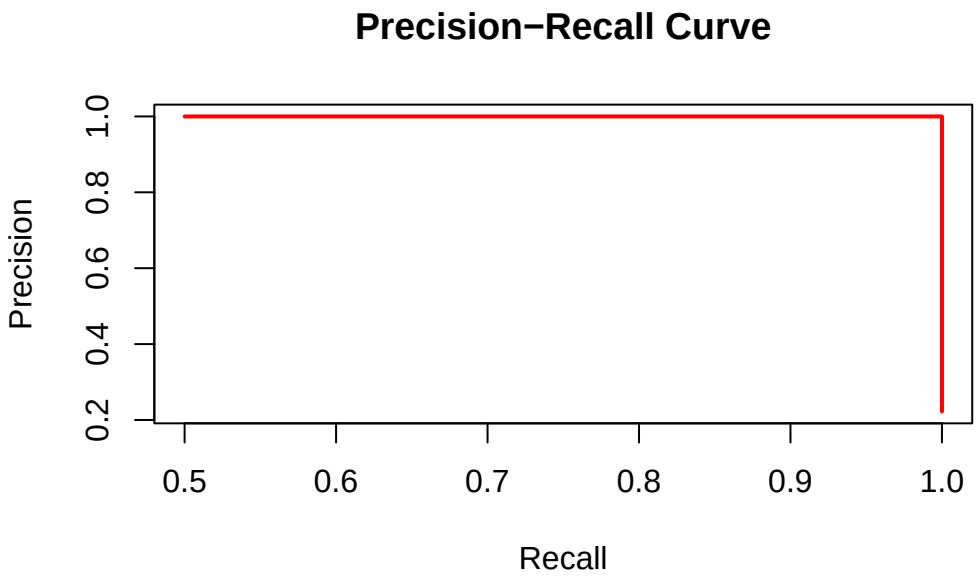
```r
# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")
```




```r
# Calculate AUC-PR
auc_pr <- ROCR::performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr
```

```
[1] 1
```

```r
# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")
```



4.8.3 Ridge (L2) Regularization

Ridge regression is a regularization technique that adds a penalty term to the loss function of the model, which is proportional to the **square of the coefficients**. The Ridge penalty **encourages small coefficients**, but it does not set any coefficients to zero, meaning that it **does not perform feature selection**. This can be useful when we have a large number of features and we want to keep all of them in the model, but we want to prevent overfitting by shrinking the coefficients towards zero.

Mathematically, the L2 norm is defined as the sum of the squares of the coefficients:

$$L2 = \lambda \sum_{j=1}^{p} \beta_j^2.$$

Where λ is the regularization parameter that controls the strength of the penalty, and β_j are the coefficients of the model.

4.8.3.1 Understanding Ridge Coefficients and the L2 Norm

While Lasso (L1) aims for a sparse model by zeroing out coefficients, **Ridge Regression (L2)** takes a different approach to regularization. It is particularly powerful when dealing with the high degree of correlation (multicollinearity) common in biological systems.

1. The "Shrinkage" Effect The L2 norm penalizes the **square** of the magnitude of the coefficients. Mathematically, this means that while the penalty gets smaller as the coefficient approaches zero, it never quite reaches it. As a result, Ridge Regression "shrinks" all coefficients toward zero, but it **almost never sets them to exactly zero.**

2. Handling Gene Networks (Multicollinearity) This is where L2 shines for a biologist. If you have a group of 10 genes that are all part of the same signaling pathway and highly correlated with each other:

- **Lasso (L1)** would likely pick one gene arbitrarily and discard the other nine.
- **Ridge (L2)** will keep all 10 genes in the model, shrinking their coefficients collectively.

3. Interpretation of "Small" Coefficients In an L2 model, every gene in your dataset technically contributes to the prediction. Instead of "important vs. unimportant," we interpret the **relative magnitude**. A gene with a relatively larger coefficient is more influential in the model, but the model still relies on a "consensus" of many small effects rather than a few "superstar" genes.

4. When to Choose L2 over L1?

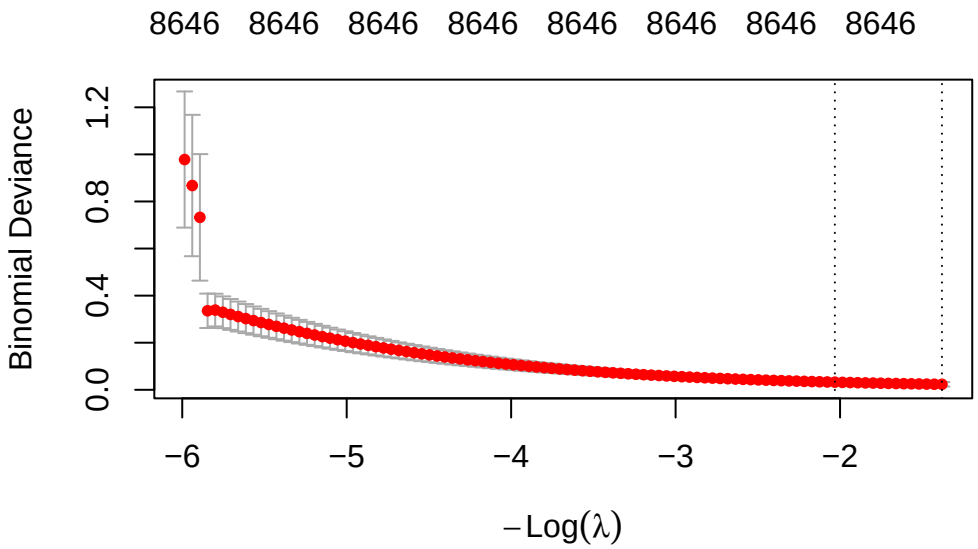
- **Predictive Stability:** Ridge often provides more stable predictions than Lasso when predictors are highly correlated.
- **Biological Complexity:** If you believe the biological trait you are studying is **polygenic** (influenced by many genes with small effects), Ridge is a more appropriate statistical assumption than Lasso.
- **The Trade-off:** The downside of Ridge is that it does not perform feature selection. You will still have 1000+ genes in your final model, which makes it harder to develop a "cheap" diagnostic PCR test for the wet lab.

  Note: Think about the Ridge as the "Democratic" model. It gives every gene a vote, but it reduces the power of the loudest voices to prevent any single gene from dominating the results due to random noise.

4.8.3.2 Running a Ridge Regression Model in R: Basic Framework

To run a Ridge regression model in R, we can use the `{glmnet}` package, which provides a simple interface for fitting Ridge regression models. The approach is very similar to running a Lasso, the main difference is the alpha parameter, which is set to 0 for Ridge regression. Here is an example of how to fit a Ridge regression model for a classification problem:

```
library(glmnet)
set.seed(123) # For reproducibility
# We use the same data as prepared for the Lasso regression, but we will set alpha = 0 for R
cv_ridge <- cv.glmnet(train_matrix,
                      train_labels,
                      alpha = 0, # alpha = 0 for Ridge
                      family = "binomial")

plot(cv_ridge)
```




```
best_lambda_ridge <- cv_ridge$lambda.min
best_lambda_ridge
```

```
[1] 3.976754
```

```
ridge_model <- glmnet(train_matrix,
                      train_labels,
                      alpha = 0, # alpha = 0 for Ridge
```

```
                      lambda = best_lambda_ridge,
                      family = "binomial")

# Print the Ridge regression model
coef(ridge_model) %>%
  as.matrix() %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
  arrange(desc(abs(s0))) %>%
  filter(abs(s0) > 0.001) %>% # Filter for coefficients with a magnitude greater than a certa
  filter(Gene != "(Intercept)") %>%
  knitr::kable(caption = "Most Important Features in the Ridge Regression Model")
```

Table 4.5: Most Important Features in the Ridge Regression Model

Gene	s0
ENSG00000215644.10	0.0020081
ENSG00000126231.15	0.0019705
ENSG00000204653.10	0.0019398
ENSG00000109758.9	0.0019326
ENSG00000158104.11	0.0019150
ENSG00000121690.11	0.0019047
ENSG00000213398.8	0.0019040
ENSG00000130649.10	0.0018988
ENSG00000214855.9	0.0018848
ENSG00000143257.12	0.0018815
ENSG00000118514.14	0.0018814
ENSG00000149124.11	0.0018782
ENSG00000138823.13	0.0018771
ENSG00000162460.7	0.0018757
ENSG00000021826.17	0.0018748
ENSG00000173531.15	0.0018740
ENSG00000286024.1	0.0018738
ENSG00000240935.6	0.0018617
ENSG00000109576.14	0.0018612
ENSG00000166391.15	0.0018607
ENSG00000165140.11	0.0018603
ENSG00000134538.3	0.0018593
ENSG00000055957.11	0.0018545
ENSG00000163586.10	0.0018542
ENSG00000196616.14	0.0018532

Gene	s0
ENSG00000005421.9	0.0018509
ENSG00000009724.17	0.0018505
ENSG00000205866.4	0.0018496
ENSG00000167798.17	0.0018448
ENSG00000172425.12	0.0018439
ENSG00000090512.12	0.0018413
ENSG00000135929.9	0.0018398
ENSG00000138315.13	0.0018385
ENSG00000145708.11	0.0018337
ENSG00000123454.12	0.0018309
ENSG00000148803.12	0.0018275
ENSG00000083807.10	0.0018212
ENSG00000140505.7	0.0018190
ENSG00000137491.15	0.0018190
ENSG00000168306.13	0.0018177
ENSG00000172497.9	0.0018169
ENSG00000113905.5	0.0018153
ENSG00000145826.9	0.0018151
ENSG00000163631.17	0.0018150
ENSG00000183549.10	0.0018132
ENSG00000170509.12	0.0018067
ENSG00000180432.6	0.0018061
ENSG00000132837.15	0.0017993
ENSG00000248709.3	0.0017987
ENSG00000140107.12	0.0017963
ENSG00000060971.19	0.0017959
ENSG00000156006.5	0.0017931
ENSG00000163687.14	0.0017930
ENSG00000255974.8	0.0017908
ENSG00000155666.12	0.0017897
ENSG00000170458.14	0.0017872
ENSG00000052802.13	0.0017861
ENSG00000187758.8	0.0017861
ENSG00000099834.19	0.0017843
ENSG00000112337.10	0.0017822
ENSG00000160339.16	0.0017812
ENSG00000161031.13	0.0017810
ENSG00000250799.11	0.0017787
ENSG00000132840.10	0.0017784
ENSG00000196177.13	0.0017763
ENSG00000120054.12	0.0017758

Gene	s0
ENSG00000140263.15	0.0017745
ENSG00000132855.5	0.0017740
ENSG00000169738.8	0.0017738
ENSG00000002933.9	0.0017728
ENSG00000135447.17	0.0017724
ENSG00000116882.15	0.0017716
ENSG00000147647.13	0.0017702
ENSG00000106823.12	0.0017701
ENSG00000159403.18	0.0017638
ENSG00000151224.13	0.0017568
ENSG00000162365.12	0.0017566
ENSG00000174992.8	0.0017563
ENSG00000188725.8	-0.0017558
ENSG00000197408.10	0.0017556
ENSG00000204444.11	0.0017547
ENSG00000187048.13	0.0017544
ENSG00000186115.13	0.0017539
ENSG00000159650.9	0.0017532
ENSG00000187824.9	0.0017505
ENSG00000129925.11	-0.0017501
ENSG00000271992.1	0.0017472
ENSG00000144908.14	0.0017465
ENSG00000147100.11	0.0017449
ENSG00000111700.13	0.0017446
ENSG00000160868.15	0.0017443
ENSG00000186910.4	0.0017419
ENSG00000117305.15	0.0017408
ENSG00000145781.9	-0.0017391
ENSG00000146233.8	0.0017391
ENSG00000145321.13	0.0017384
ENSG00000142168.15	0.0017378
ENSG00000184999.12	0.0017358
ENSG00000129596.5	0.0017342
ENSG00000118137.10	0.0017314
ENSG00000130988.13	0.0017310
ENSG00000154262.13	0.0017298
ENSG00000168237.18	0.0017264
ENSG00000198732.11	0.0017252
ENSG00000095203.14	0.0017251
ENSG00000100577.19	0.0017176
ENSG00000106565.18	0.0017168

Gene	s0
ENSG00000107186.17	0.0017154
ENSG00000157131.11	0.0017145
ENSG00000166183.16	0.0017141
ENSG00000122787.15	0.0017141
ENSG00000182220.15	-0.0017140
ENSG00000137561.5	0.0017134
ENSG00000145916.19	-0.0017126
ENSG00000138109.11	0.0017126
ENSG00000182327.8	0.0017104
ENSG00000108515.18	0.0017093
ENSG00000114200.10	0.0017090
ENSG00000261701.7	0.0017081
ENSG00000138030.13	0.0017066
ENSG00000165410.15	0.0017050
ENSG00000111275.13	0.0017030
ENSG00000134716.11	0.0017002
ENSG00000103569.10	0.0016970
ENSG00000123453.18	0.0016959
ENSG00000100652.5	0.0016955
ENSG00000111181.12	0.0016936
ENSG00000160282.14	0.0016929
ENSG00000229005.2	0.0016919
ENSG00000142748.13	0.0016910
ENSG00000130707.18	0.0016909
ENSG00000080910.13	0.0016905
ENSG00000105607.13	0.0016900
ENSG00000124155.18	-0.0016900
ENSG00000175287.19	0.0016887
ENSG00000062282.15	0.0016867
ENSG00000162882.15	0.0016857
ENSG00000172955.17	0.0016847
ENSG00000161267.12	0.0016842
ENSG00000164329.13	-0.0016838
ENSG00000175336.10	0.0016816
ENSG00000023330.15	0.0016807
ENSG00000250056.8	0.0016797
ENSG00000145192.13	0.0016774
ENSG00000161671.17	-0.0016719
ENSG00000182551.14	0.0016715
ENSG00000164344.16	0.0016701
ENSG00000173334.4	0.0016697

Gene	s0
ENSG00000101981.12	0.0016659
ENSG00000251424.1	0.0016652
ENSG00000149131.17	0.0016635
ENSG00000141485.17	0.0016609
ENSG00000183258.12	-0.0016574
ENSG00000182902.14	0.0016573
ENSG00000134389.10	0.0016564
ENSG00000133027.18	0.0016555
ENSG00000198099.9	0.0016547
ENSG00000152422.16	-0.0016543
ENSG00000176087.15	-0.0016535
ENSG00000137204.15	0.0016535
ENSG00000100554.12	-0.0016510
ENSG00000070010.19	-0.0016506
ENSG00000100665.12	0.0016494
ENSG00000100889.12	0.0016491
ENSG00000145692.15	0.0016482
ENSG00000130203.10	0.0016466
ENSG00000130208.9	0.0016463
ENSG00000174990.8	0.0016460
ENSG00000287770.1	0.0016449
ENSG00000166840.13	0.0016442
ENSG00000059769.20	0.0016435
ENSG00000198373.13	-0.0016431
ENSG00000249173.6	0.0016430
ENSG00000157379.14	0.0016430
ENSG00000256847.1	0.0016423
ENSG00000144852.19	0.0016405
ENSG00000168275.16	-0.0016397
ENSG00000182326.15	0.0016375
ENSG00000131781.13	0.0016365
ENSG00000152078.10	0.0016358
ENSG00000168159.14	-0.0016347
ENSG00000113575.10	-0.0016347
ENSG00000136710.10	-0.0016344
ENSG00000051382.9	-0.0016343
ENSG00000162433.15	0.0016341
ENSG00000168924.15	-0.0016331
ENSG00000109929.10	0.0016330
ENSG00000158435.8	-0.0016316
ENSG00000139547.7	0.0016264

Gene	s0
ENSG00000109917.11	-0.0016219
ENSG00000115255.12	0.0016190
ENSG00000131187.10	0.0016175
ENSG00000171560.16	0.0016157
ENSG00000137106.18	0.0016153
ENSG00000123561.15	0.0016151
ENSG00000138207.14	0.0016140
ENSG00000139194.8	0.0016135
ENSG00000132541.11	0.0016116
ENSG00000185386.15	-0.0016109
ENSG00000166598.15	-0.0016076
ENSG00000204619.8	-0.0016072
ENSG00000145782.13	-0.0016064
ENSG00000162688.17	0.0016051
ENSG00000128654.14	-0.0016047
ENSG00000065000.19	-0.0016038
ENSG00000175003.15	0.0016037
ENSG00000135778.12	-0.0016036
ENSG00000149930.18	-0.0016001
ENSG00000100714.17	0.0015990
ENSG00000213512.3	0.0015978
ENSG00000105398.4	0.0015974
ENSG00000084674.15	0.0015971
ENSG00000174891.13	-0.0015964
ENSG00000272333.7	-0.0015958
ENSG00000102967.12	0.0015931
ENSG00000115718.18	0.0015923
ENSG00000134365.13	0.0015920
ENSG00000140961.14	0.0015915
ENSG00000141505.12	0.0015901
ENSG00000072080.11	0.0015901
ENSG00000119711.13	0.0015892
ENSG00000121691.7	0.0015882
ENSG00000138115.14	0.0015880
ENSG00000114744.9	-0.0015859
ENSG00000091583.11	0.0015833
ENSG00000171564.12	0.0015823
ENSG00000183864.5	-0.0015817
ENSG00000113141.18	-0.0015814
ENSG00000134440.12	-0.0015812
ENSG00000008282.9	-0.0015808

Gene	s0
ENSG00000283342.1	0.0015800
ENSG00000118271.12	0.0015755
ENSG00000135069.14	0.0015740
ENSG00000180210.15	0.0015732
ENSG00000163702.20	0.0015727
ENSG00000214135.8	-0.0015727
ENSG00000158825.6	0.0015721
ENSG00000107566.14	0.0015710
ENSG00000084734.9	0.0015700
ENSG00000088926.14	0.0015689
ENSG00000113600.11	0.0015684
ENSG00000111667.14	-0.0015673
ENSG00000244414.7	0.0015667
ENSG00000162267.12	0.0015664
ENSG00000148672.9	0.0015640
ENSG00000073734.10	0.0015640
ENSG00000058262.10	-0.0015635
ENSG00000005007.13	-0.0015626
ENSG00000124713.6	0.0015618
ENSG00000100003.18	0.0015611
ENSG00000171557.17	0.0015610
ENSG00000079557.5	0.0015603
ENSG00000103043.15	-0.0015591
ENSG00000183044.12	0.0015586
ENSG00000036473.8	0.0015584
ENSG00000132313.15	-0.0015569
ENSG00000136881.12	0.0015568
ENSG00000152782.16	0.0015552
ENSG00000130222.11	0.0015543
ENSG00000214274.10	0.0015534
ENSG00000073754.6	0.0015534
ENSG00000172270.21	-0.0015531
ENSG00000163581.14	0.0015523
ENSG00000100024.15	0.0015515
ENSG00000103051.20	-0.0015511
ENSG00000022277.13	-0.0015506
ENSG00000112996.11	-0.0015497
ENSG00000094631.21	0.0015490
ENSG00000156973.14	-0.0015480
ENSG00000113492.14	0.0015480
ENSG00000135052.16	-0.0015476

Gene	s0
ENSG00000146007.11	-0.0015475
ENSG00000091262.16	0.0015467
ENSG00000157557.13	0.0015462
ENSG00000142327.13	-0.0015458
ENSG00000147164.12	-0.0015452
ENSG00000048544.6	-0.0015451
ENSG00000164244.21	-0.0015449
ENSG00000066813.14	0.0015443
ENSG00000175348.11	-0.0015414
ENSG00000113504.21	-0.0015403
ENSG00000014824.14	-0.0015387
ENSG00000092203.15	-0.0015386
ENSG00000141026.6	-0.0015369
ENSG00000132604.11	-0.0015352
ENSG00000117682.17	-0.0015348
ENSG00000196419.13	-0.0015342
ENSG00000174695.10	-0.0015334
ENSG00000143774.17	-0.0015325
ENSG00000244038.11	-0.0015316
ENSG00000125967.17	-0.0015304
ENSG00000118900.15	-0.0015295
ENSG00000118520.15	0.0015289
ENSG00000123933.17	-0.0015282
ENSG00000002834.18	-0.0015282
ENSG00000119673.14	0.0015280
ENSG00000108924.14	0.0015266
ENSG00000003989.18	0.0015259
ENSG00000197150.13	-0.0015239
ENSG00000166816.15	0.0015238
ENSG00000119844.15	-0.0015238
ENSG00000138107.13	-0.0015236
ENSG00000138308.6	0.0015211
ENSG00000167588.13	0.0015210
ENSG00000156222.12	0.0015208
ENSG00000069345.12	-0.0015180
ENSG00000171759.10	0.0015178
ENSG00000120509.11	-0.0015176
ENSG00000151365.2	0.0015167
ENSG00000171234.14	0.0015151
ENSG00000119682.17	-0.0015149
ENSG00000175899.15	0.0015147

Gene	s0
ENSG00000107537.14	0.0015144
ENSG00000129055.13	-0.0015140
ENSG00000131508.16	-0.0015136
ENSG00000123104.12	0.0015114
ENSG00000101323.5	0.0015113
ENSG00000143761.16	-0.0015111
ENSG00000030110.13	-0.0015103
ENSG00000138796.17	0.0015089
ENSG00000149428.19	-0.0015089
ENSG00000110169.11	0.0015081
ENSG00000168256.18	-0.0015076
ENSG00000278967.1	0.0015072
ENSG00000164902.14	-0.0015063
ENSG00000079739.17	0.0015057
ENSG00000106538.10	0.0015053
ENSG00000171848.16	-0.0015042
ENSG00000143740.14	-0.0015041
ENSG00000101019.22	-0.0015039
ENSG00000277893.2	0.0015028
ENSG00000106993.12	0.0015025
ENSG00000168118.12	-0.0015015
ENSG00000100116.17	0.0014989
ENSG00000119013.9	-0.0014973
ENSG00000099290.17	-0.0014969
ENSG00000275152.5	0.0014956
ENSG00000019169.10	0.0014954
ENSG00000115685.15	-0.0014948
ENSG00000198612.11	-0.0014942
ENSG00000173692.14	-0.0014942
ENSG00000120158.12	0.0014927
ENSG00000167711.14	0.0014918
ENSG00000021852.13	0.0014912
ENSG00000008283.16	-0.0014912
ENSG00000138356.14	0.0014889
ENSG00000106682.15	-0.0014885
ENSG00000174132.9	-0.0014883
ENSG00000176871.9	-0.0014880
ENSG00000163950.13	-0.0014879
ENSG00000162881.6	0.0014877
ENSG00000205707.11	0.0014865
ENSG00000004864.14	0.0014858

Gene	s0
ENSG00000140284.11	0.0014852
ENSG00000158874.11	0.0014846
ENSG00000127022.15	-0.0014837
ENSG00000132842.14	-0.0014834
ENSG00000109181.12	0.0014828
ENSG00000112695.11	-0.0014805
ENSG00000100197.22	0.0014795
ENSG00000171109.19	-0.0014794
ENSG00000161204.11	-0.0014788
ENSG00000188338.15	0.0014787
ENSG00000110887.8	0.0014783
ENSG00000134970.14	-0.0014773
ENSG00000129484.14	-0.0014772
ENSG00000249948.6	0.0014768
ENSG00000119718.11	-0.0014768
ENSG00000182704.8	0.0014766
ENSG00000234771.3	-0.0014759
ENSG00000168724.17	-0.0014757
ENSG00000116761.12	0.0014752
ENSG00000184992.13	-0.0014742
ENSG00000170445.16	-0.0014740
ENSG00000135956.9	-0.0014737
ENSG00000159720.12	-0.0014732
ENSG00000181035.14	0.0014730
ENSG00000122971.9	0.0014724
ENSG00000178537.10	0.0014717
ENSG00000272335.1	-0.0014711
ENSG00000170871.12	-0.0014709
ENSG00000114209.15	-0.0014705
ENSG00000160862.13	0.0014704
ENSG00000124356.16	-0.0014704
ENSG00000113649.11	-0.0014703
ENSG00000261012.2	0.0014702
ENSG00000173409.14	-0.0014700
ENSG00000165802.23	-0.0014699
ENSG00000159921.19	0.0014689
ENSG00000063854.13	0.0014687
ENSG00000163166.15	-0.0014686
ENSG00000176393.11	-0.0014677
ENSG00000106327.13	0.0014673
ENSG00000161217.12	-0.0014669

Gene	s0
ENSG00000166747.13	-0.0014668
ENSG00000116785.14	0.0014656
ENSG00000119231.11	-0.0014648
ENSG00000162889.11	-0.0014638
ENSG00000141084.11	-0.0014637
ENSG00000134153.10	-0.0014623
ENSG00000180346.4	0.0014622
ENSG00000140830.9	-0.0014621
ENSG00000121073.15	-0.0014617
ENSG00000204344.14	-0.0014616
ENSG00000100335.15	-0.0014611
ENSG00000113648.16	-0.0014609
ENSG00000151552.12	0.0014603
ENSG00000124253.11	0.0014602
ENSG00000144867.13	-0.0014598
ENSG00000117594.10	0.0014596
ENSG00000181789.14	-0.0014590
ENSG00000049860.14	-0.0014590
ENSG00000186529.16	0.0014586
ENSG00000142494.13	0.0014578
ENSG00000173456.5	-0.0014565
ENSG00000127241.18	0.0014565
ENSG00000066044.15	-0.0014560
ENSG00000113569.16	-0.0014560
ENSG00000101544.9	-0.0014557
ENSG00000118705.17	-0.0014550
ENSG00000163655.16	-0.0014548
ENSG00000130005.13	0.0014526
ENSG00000146085.8	0.0014525
ENSG00000179364.14	-0.0014517
ENSG00000142230.13	-0.0014513
ENSG00000159346.13	-0.0014511
ENSG00000065054.14	0.0014504
ENSG00000125166.13	0.0014498
ENSG00000184840.12	-0.0014494
ENSG00000256162.2	0.0014492
ENSG00000143742.14	-0.0014487
ENSG00000105171.10	-0.0014480
ENSG00000141551.14	-0.0014472
ENSG00000100997.20	-0.0014470
ENSG00000113734.18	-0.0014448

Gene	s0
ENSG00000033011.13	-0.0014435
ENSG00000172828.13	0.0014430
ENSG00000204386.12	-0.0014429
ENSG00000132912.13	-0.0014426
ENSG00000100138.15	-0.0014426
ENSG00000101150.18	-0.0014426
ENSG00000139209.16	0.0014423
ENSG00000175224.16	-0.0014420
ENSG00000179218.14	-0.0014419
ENSG00000081307.14	-0.0014409
ENSG00000152942.19	-0.0014398
ENSG00000139344.8	0.0014397
ENSG00000114573.10	-0.0014395
ENSG00000186575.19	-0.0014394
ENSG00000108187.16	0.0014392
ENSG00000138075.14	0.0014389
ENSG00000197136.4	-0.0014388
ENSG00000171766.17	0.0014385
ENSG00000257017.9	0.0014380
ENSG00000248144.6	0.0014378
ENSG00000215193.13	-0.0014377
ENSG00000156639.12	-0.0014369
ENSG00000087076.9	0.0014367
ENSG00000163918.11	-0.0014361
ENSG00000013306.16	-0.0014359
ENSG00000102158.20	-0.0014359
ENSG00000120915.14	0.0014357
ENSG00000075239.14	0.0014355
ENSG00000125144.14	0.0014352
ENSG00000151726.15	0.0014351
ENSG00000161533.12	0.0014347
ENSG00000104969.11	-0.0014345
ENSG00000109066.14	-0.0014337
ENSG00000159792.10	-0.0014327
ENSG00000186432.10	-0.0014322
ENSG00000136527.18	-0.0014316
ENSG00000167491.17	-0.0014312
ENSG00000065518.8	-0.0014311
ENSG00000048140.18	-0.0014299
ENSG00000138231.13	-0.0014298
ENSG00000178988.11	-0.0014296

Gene	s0
ENSG00000100744.15	-0.0014294
ENSG00000132703.4	0.0014290
ENSG00000120437.9	0.0014281
ENSG00000114354.15	-0.0014278
ENSG00000186204.15	0.0014277
ENSG00000101152.11	-0.0014271
ENSG00000089220.5	0.0014269
ENSG00000112964.14	0.0014269
ENSG00000168883.20	-0.0014268
ENSG00000125356.7	-0.0014268
ENSG00000130935.10	-0.0014265
ENSG00000146282.18	-0.0014264
ENSG00000178188.14	-0.0014261
ENSG00000116771.6	0.0014252
ENSG00000181396.13	-0.0014249
ENSG00000184743.13	-0.0014244
ENSG00000167522.16	-0.0014238
ENSG00000179761.12	0.0014238
ENSG00000131408.15	-0.0014237
ENSG00000010244.19	-0.0014231
ENSG00000161202.20	-0.0014230
ENSG00000115363.14	0.0014230
ENSG00000244474.6	0.0014229
ENSG00000197386.12	-0.0014211
ENSG00000168397.17	-0.0014199
ENSG00000205629.12	-0.0014194
ENSG00000114378.17	0.0014192
ENSG00000205765.9	-0.0014192
ENSG00000113312.11	-0.0014191
ENSG00000162961.14	-0.0014184
ENSG00000124532.15	-0.0014182
ENSG00000146416.19	0.0014180
ENSG00000172775.17	-0.0014173
ENSG00000106927.12	0.0014164
ENSG00000113300.13	-0.0014163
ENSG00000174529.7	-0.0014162
ENSG00000145476.16	0.0014156
ENSG00000161904.12	-0.0014155
ENSG00000175376.9	-0.0014153
ENSG00000131467.11	-0.0014148
ENSG00000121361.5	0.0014146

Gene	s0
ENSG00000225921.7	-0.0014123
ENSG00000139540.12	0.0014122
ENSG00000225756.1	0.0014122
ENSG00000169019.11	-0.0014122
ENSG00000213699.9	-0.0014119
ENSG00000198695.2	0.0014118
ENSG00000152620.13	0.0014108
ENSG00000089234.16	-0.0014108
ENSG00000178913.8	-0.0014104
ENSG00000100387.9	-0.0014102
ENSG00000153015.16	-0.0014097
ENSG00000167004.13	-0.0014096
ENSG00000085978.22	-0.0014091
ENSG00000171720.10	-0.0014090
ENSG00000149476.16	0.0014080
ENSG00000127884.5	0.0014076
ENSG00000179115.11	-0.0014074
ENSG00000204351.12	-0.0014070
ENSG00000144895.12	-0.0014066
ENSG00000119471.15	0.0014065
ENSG00000144231.11	-0.0014058
ENSG00000197451.12	-0.0014054
ENSG00000103512.15	-0.0014053
ENSG00000169083.18	0.0014047
ENSG00000154511.12	0.0014046
ENSG00000099940.12	-0.0014042
ENSG00000131165.16	-0.0014034
ENSG00000174903.16	-0.0014025
ENSG00000113068.10	-0.0014022
ENSG00000214160.10	-0.0014021
ENSG00000169871.13	-0.0014009
ENSG00000156411.10	-0.0013999
ENSG00000198610.11	0.0013994
ENSG00000131979.20	0.0013987
ENSG00000173660.12	-0.0013963
ENSG00000106245.11	-0.0013956
ENSG00000085231.14	-0.0013954
ENSG00000157103.12	0.0013948
ENSG00000101193.8	-0.0013942
ENSG00000145191.15	-0.0013938
ENSG00000141741.12	-0.0013938

Gene	s0
ENSG00000006194.10	-0.0013931
ENSG00000184208.12	-0.0013929
ENSG00000151779.13	-0.0013926
ENSG00000103111.15	-0.0013925
ENSG00000177889.10	-0.0013924
ENSG00000140319.11	-0.0013910
ENSG00000106636.9	-0.0013908
ENSG00000162840.4	0.0013902
ENSG00000155329.12	-0.0013898
ENSG00000088038.19	-0.0013896
ENSG00000149658.18	-0.0013891
ENSG00000159593.15	-0.0013888
ENSG00000105852.11	0.0013883
ENSG00000213523.10	-0.0013882
ENSG00000178951.9	-0.0013882
ENSG00000137288.10	-0.0013878
ENSG00000155463.14	-0.0013876
ENSG00000214753.4	-0.0013870
ENSG00000140632.17	-0.0013868
ENSG00000109072.14	0.0013865
ENSG00000100227.18	-0.0013862
ENSG00000112685.14	-0.0013862
ENSG00000141522.12	-0.0013859
ENSG00000157869.16	-0.0013854
ENSG00000137409.19	-0.0013853
ENSG00000163608.15	-0.0013852
ENSG00000161999.13	-0.0013849
ENSG00000182446.14	-0.0013848
ENSG00000204231.11	-0.0013848
ENSG00000187546.14	0.0013846
ENSG00000081154.12	-0.0013841
ENSG00000129562.11	-0.0013837
ENSG00000177951.17	-0.0013825
ENSG00000087274.17	-0.0013819
ENSG00000171161.13	-0.0013802
ENSG00000181192.12	0.0013798
ENSG00000079805.18	-0.0013798
ENSG00000135336.15	-0.0013796
ENSG00000112739.17	-0.0013795
ENSG00000132305.21	-0.0013793
ENSG00000114796.16	-0.0013789

Gene	s0
ENSG00000182149.21	-0.0013785
ENSG00000115561.16	-0.0013777
ENSG00000198898.14	-0.0013772
ENSG00000124164.16	-0.0013768
ENSG00000128928.10	0.0013764
ENSG00000100519.12	-0.0013761
ENSG00000135373.13	-0.0013748
ENSG00000254858.10	-0.0013748
ENSG00000133706.19	-0.0013748
ENSG00000132386.11	0.0013747
ENSG00000008394.13	0.0013744
ENSG00000049541.11	-0.0013743
ENSG00000137642.13	0.0013738
ENSG00000204439.4	-0.0013735
ENSG00000129460.16	-0.0013734
ENSG00000125826.21	-0.0013733
ENSG00000166986.15	-0.0013727
ENSG00000112941.14	-0.0013720
ENSG00000137055.15	-0.0013719
ENSG00000140259.7	-0.0013715
ENSG00000226137.5	-0.0013714
ENSG00000143376.14	-0.0013714
ENSG00000160695.15	-0.0013704
ENSG00000197226.13	-0.0013703
ENSG00000153406.14	-0.0013702
ENSG00000241058.4	0.0013702
ENSG00000172115.9	-0.0013695
ENSG00000138792.10	0.0013690
ENSG00000115806.13	-0.0013686
ENSG00000131148.9	-0.0013685
ENSG00000135269.18	-0.0013682
ENSG00000107929.15	-0.0013675
ENSG00000113558.19	-0.0013671
ENSG00000148057.16	0.0013656
ENSG00000165471.7	0.0013656
ENSG00000125970.12	-0.0013656
ENSG00000125651.14	-0.0013649
ENSG00000145703.16	0.0013642
ENSG00000099937.11	0.0013639
ENSG00000149809.16	0.0013629
ENSG00000080371.6	-0.0013625

Gene	s0
ENSG00000100325.15	-0.0013621
ENSG00000131759.18	-0.0013620
ENSG00000146247.14	-0.0013613
ENSG00000232119.8	-0.0013613
ENSG00000203950.6	-0.0013612
ENSG00000167283.8	-0.0013600
ENSG00000170348.9	-0.0013598
ENSG00000022840.16	-0.0013594
ENSG00000084090.13	-0.0013592
ENSG00000063978.16	-0.0013590
ENSG00000112033.14	-0.0013582
ENSG00000108587.16	-0.0013580
ENSG00000113194.13	-0.0013578
ENSG00000105058.12	-0.0013570
ENSG00000111885.7	0.0013564
ENSG00000204560.10	-0.0013563
ENSG00000182871.16	0.0013556
ENSG00000181666.18	-0.0013555
ENSG00000134014.17	-0.0013554
ENSG00000119541.10	-0.0013550
ENSG00000205403.13	0.0013546
ENSG00000115761.16	-0.0013540
ENSG00000162496.9	0.0013538
ENSG00000117899.11	-0.0013538
ENSG00000105697.9	0.0013533
ENSG00000196510.12	-0.0013532
ENSG00000108242.13	0.0013522
ENSG00000130803.15	-0.0013519
ENSG00000101843.19	-0.0013513
ENSG00000185262.9	-0.0013513
ENSG00000122194.19	0.0013497
ENSG00000181524.6	-0.0013497
ENSG00000278259.4	-0.0013495
ENSG00000132376.20	-0.0013493
ENSG00000129472.15	-0.0013492
ENSG00000163825.4	0.0013492
ENSG00000087302.9	-0.0013491
ENSG00000133606.11	-0.0013489
ENSG00000077721.16	-0.0013489
ENSG00000186480.13	0.0013489
ENSG00000013374.16	-0.0013488

Gene	s0
ENSG00000170522.10	0.0013483
ENSG00000139726.11	-0.0013468
ENSG00000205358.4	0.0013467
ENSG00000065457.10	-0.0013460
ENSG00000125148.7	0.0013453
ENSG00000179010.14	-0.0013453
ENSG00000116857.17	-0.0013452
ENSG00000158882.15	-0.0013450
ENSG00000171475.14	-0.0013449
ENSG00000131558.15	-0.0013447
ENSG00000163902.12	-0.0013439
ENSG00000103591.13	-0.0013434
ENSG00000119977.21	-0.0013431
ENSG00000185721.13	-0.0013428
ENSG00000001561.7	-0.0013427
ENSG00000184432.11	-0.0013423
ENSG00000155393.13	-0.0013421
ENSG00000053900.11	-0.0013421
ENSG00000166847.10	-0.0013421
ENSG00000196290.16	-0.0013416
ENSG00000189114.8	-0.0013412
ENSG00000254004.7	-0.0013411
ENSG00000050748.18	-0.0013408
ENSG00000144043.12	-0.0013403
ENSG00000171503.12	0.0013403
ENSG00000185787.15	-0.0013401
ENSG00000213585.11	-0.0013399
ENSG00000100258.18	-0.0013395
ENSG00000204228.4	0.0013391
ENSG00000240857.2	-0.0013390
ENSG00000115540.15	-0.0013387
ENSG00000204564.12	-0.0013386
ENSG00000179933.6	-0.0013385
ENSG00000101557.15	-0.0013383
ENSG00000100023.19	-0.0013373
ENSG00000156599.11	-0.0013368
ENSG00000164880.16	-0.0013367
ENSG00000171311.13	-0.0013367
ENSG00000130254.12	-0.0013366
ENSG00000113790.11	0.0013364
ENSG00000268205.1	-0.0013361

Gene	s0
ENSG00000163714.18	-0.0013360
ENSG00000134824.14	0.0013360
ENSG00000170270.5	-0.0013360
ENSG00000166347.19	0.0013359
ENSG00000259865.1	-0.0013356
ENSG00000125386.16	-0.0013355
ENSG00000185359.14	-0.0013350
ENSG00000105700.11	-0.0013345
ENSG00000104853.16	-0.0013340
ENSG00000115289.14	-0.0013335
ENSG00000187051.9	-0.0013325
ENSG00000185298.13	-0.0013324
ENSG00000049656.14	-0.0013320
ENSG00000067248.10	-0.0013318
ENSG00000131871.15	-0.0013314
ENSG00000130725.8	-0.0013311
ENSG00000138069.18	-0.0013311
ENSG00000102974.16	-0.0013310
ENSG00000139178.11	0.0013303
ENSG00000166135.14	-0.0013297
ENSG00000140093.10	0.0013296
ENSG00000128524.5	-0.0013292
ENSG00000025423.11	0.0013287
ENSG00000099785.10	-0.0013283
ENSG00000132581.10	-0.0013282
ENSG00000062194.16	-0.0013276
ENSG00000134910.14	-0.0013273
ENSG00000099769.6	0.0013273
ENSG00000163956.13	-0.0013270
ENSG00000105669.14	-0.0013268
ENSG00000136522.14	-0.0013263
ENSG00000185432.12	0.0013255
ENSG00000134287.10	-0.0013245
ENSG00000183723.13	-0.0013241
ENSG00000196235.14	-0.0013234
ENSG00000143921.9	0.0013234
ENSG00000169926.11	-0.0013224
ENSG00000164284.15	-0.0013216
ENSG00000159335.18	0.0013211
ENSG00000078668.14	-0.0013203
ENSG00000127526.15	-0.0013201

Gene	s0
ENSG00000139508.15	0.0013199
ENSG00000101464.11	-0.0013192
ENSG00000127837.10	-0.0013188
ENSG00000187193.9	0.0013185
ENSG00000204396.11	-0.0013182
ENSG00000068912.14	-0.0013181
ENSG00000146066.3	-0.0013173
ENSG00000139697.14	-0.0013167
ENSG00000130706.13	-0.0013165
ENSG00000176422.14	0.0013164
ENSG00000101940.18	-0.0013160
ENSG00000090861.17	-0.0013156
ENSG00000105671.12	-0.0013155
ENSG00000130311.11	-0.0013155
ENSG00000115694.15	-0.0013151
ENSG00000161013.17	-0.0013150
ENSG00000154914.17	-0.0013150
ENSG00000166908.18	-0.0013147
ENSG00000089280.19	-0.0013146
ENSG00000105401.9	-0.0013140
ENSG00000148584.15	0.0013133
ENSG00000124209.4	-0.0013131
ENSG00000139597.18	0.0013131
ENSG00000068654.16	-0.0013119
ENSG00000130734.10	-0.0013116
ENSG00000274523.5	-0.0013115
ENSG00000113407.14	-0.0013109
ENSG00000174437.17	-0.0013108
ENSG00000112855.17	-0.0013105
ENSG00000110917.8	-0.0013103
ENSG00000128708.13	-0.0013103
ENSG00000116750.14	-0.0013101
ENSG00000100030.15	-0.0013099
ENSG00000119616.11	-0.0013096
ENSG00000185798.8	-0.0013092
ENSG00000145919.11	-0.0013090
ENSG00000162604.12	-0.0013088
ENSG00000131966.14	-0.0013087
ENSG00000186111.10	-0.0013086
ENSG00000163848.20	-0.0013082
ENSG00000110958.16	-0.0013081

Gene	s0
ENSG00000044115.21	-0.0013079
ENSG00000175197.12	-0.0013077
ENSG00000109180.15	-0.0013075
ENSG00000163686.15	0.0013074
ENSG00000233377.1	0.0013074
ENSG00000143278.5	0.0013073
ENSG00000166913.13	-0.0013068
ENSG00000198911.12	-0.0013066
ENSG00000136872.20	0.0013063
ENSG00000136631.15	-0.0013061
ENSG00000169188.5	-0.0013060
ENSG00000070610.14	-0.0013060
ENSG00000131653.13	-0.0013059
ENSG00000153179.13	-0.0013051
ENSG00000119559.17	-0.0013051
ENSG00000166788.10	-0.0013050
ENSG00000188917.15	-0.0013050
ENSG00000128463.13	-0.0013047
ENSG00000138085.17	-0.0013045
ENSG00000105568.18	-0.0013041
ENSG00000124383.9	-0.0013034
ENSG00000100949.14	-0.0013029
ENSG00000112977.16	-0.0013028
ENSG00000183971.10	0.0013026
ENSG00000105726.17	-0.0013025
ENSG00000131069.20	0.0013022
ENSG00000103356.18	-0.0013022
ENSG00000178772.7	0.0013020
ENSG00000173744.18	-0.0013018
ENSG00000106049.9	0.0013018
ENSG00000224389.9	0.0013010
ENSG00000142444.7	-0.0013010
ENSG00000198618.5	-0.0013005
ENSG00000064726.10	-0.0013004
ENSG00000196961.13	-0.0013003
ENSG00000076984.18	-0.0013003
ENSG00000135926.15	-0.0012997
ENSG00000152102.18	-0.0012995
ENSG00000265241.7	-0.0012993
ENSG00000112308.13	-0.0012991
ENSG00000180370.10	-0.0012988

Gene	s0
ENSG00000145050.19	-0.0012978
ENSG00000144567.11	-0.0012976
ENSG00000064995.18	-0.0012976
ENSG00000146834.14	-0.0012975
ENSG00000115365.12	-0.0012971
ENSG00000156642.17	-0.0012971
ENSG00000011485.15	-0.0012968
ENSG00000111011.18	-0.0012967
ENSG00000170260.9	-0.0012966
ENSG00000103485.19	0.0012965
ENSG00000144233.10	-0.0012962
ENSG00000205362.11	0.0012962
ENSG00000177885.15	-0.0012960
ENSG00000156860.16	-0.0012959
ENSG00000186501.14	-0.0012954
ENSG00000163694.15	-0.0012950
ENSG00000243955.6	0.0012948
ENSG00000153187.20	-0.0012945
ENSG00000140350.15	-0.0012943
ENSG00000122390.19	-0.0012938
ENSG00000197498.13	-0.0012938
ENSG00000157916.20	-0.0012938
ENSG00000161888.11	-0.0012934
ENSG00000068366.20	-0.0012930
ENSG00000048162.21	-0.0012927
ENSG00000101412.13	-0.0012925
ENSG00000168092.14	-0.0012924
ENSG00000130414.13	-0.0012923
ENSG00000100124.15	-0.0012923
ENSG00000013275.8	-0.0012922
ENSG00000115204.15	-0.0012920
ENSG00000138246.17	-0.0012919
ENSG00000157326.19	0.0012915
ENSG00000146678.10	0.0012915
ENSG00000169217.9	-0.0012911
ENSG00000204463.13	-0.0012909
ENSG00000205250.9	-0.0012905
ENSG00000011009.11	-0.0012900
ENSG00000164406.8	0.0012900
ENSG00000166557.14	-0.0012895
ENSG00000124733.5	-0.0012890

Gene	s0
ENSG00000141560.15	-0.0012889
ENSG00000100056.12	-0.0012888
ENSG00000070413.20	-0.0012886
ENSG00000077463.15	-0.0012883
ENSG00000168591.16	-0.0012883
ENSG00000160447.7	-0.0012874
ENSG00000149483.12	-0.0012872
ENSG00000006451.8	-0.0012872
ENSG00000170430.10	0.0012870
ENSG00000176974.20	0.0012864
ENSG00000010270.14	-0.0012864
ENSG00000056998.20	0.0012863
ENSG00000132437.18	0.0012858
ENSG00000223501.9	-0.0012857
ENSG00000138382.15	-0.0012855
ENSG00000101337.16	-0.0012852
ENSG00000113360.16	-0.0012851
ENSG00000099991.18	-0.0012850
ENSG00000130382.9	-0.0012847
ENSG00000164331.10	-0.0012847
ENSG00000214046.9	-0.0012845
ENSG00000278311.5	-0.0012845
ENSG00000286398.1	0.0012842
ENSG00000156711.17	-0.0012840
ENSG00000011304.22	-0.0012838
ENSG00000167987.11	-0.0012836
ENSG00000183828.15	-0.0012836
ENSG00000185049.16	-0.0012833
ENSG00000122299.12	-0.0012831
ENSG00000181610.13	-0.0012829
ENSG00000103005.12	-0.0012825
ENSG00000104870.13	0.0012825
ENSG00000114631.11	-0.0012823
ENSG00000170439.7	0.0012814
ENSG00000114491.14	-0.0012811
ENSG00000167881.15	-0.0012807
ENSG00000197249.14	0.0012805
ENSG00000109133.13	-0.0012804
ENSG00000172534.14	-0.0012803
ENSG00000141564.15	-0.0012803
ENSG00000170099.6	0.0012800

Gene	s0
ENSG00000087206.17	-0.0012793
ENSG00000213024.12	-0.0012793
ENSG00000100526.20	-0.0012792
ENSG00000141002.20	-0.0012791
ENSG00000169446.5	-0.0012788
ENSG00000139428.12	0.0012788
ENSG00000136738.15	-0.0012788
ENSG00000151445.16	-0.0012784
ENSG00000103429.11	-0.0012783
ENSG00000166035.11	0.0012782
ENSG00000059122.17	-0.0012781
ENSG00000161057.13	-0.0012780
ENSG00000128607.14	-0.0012778
ENSG00000087263.17	-0.0012775
ENSG00000213588.6	-0.0012775
ENSG00000111540.16	-0.0012775
ENSG00000127445.14	-0.0012773
ENSG00000140943.17	-0.0012773
ENSG00000146963.18	-0.0012773
ENSG00000151239.14	-0.0012765
ENSG00000105323.17	-0.0012763
ENSG00000100811.14	-0.0012762
ENSG00000139620.13	-0.0012762
ENSG00000134463.15	0.0012761
ENSG00000144034.16	-0.0012760
ENSG00000102100.16	-0.0012760
ENSG00000163923.10	-0.0012755
ENSG00000100603.14	-0.0012754
ENSG00000204525.17	-0.0012751
ENSG00000167378.8	-0.0012751
ENSG00000204310.13	-0.0012748
ENSG00000104915.15	-0.0012748
ENSG00000180901.11	-0.0012747
ENSG00000136715.19	-0.0012746
ENSG00000094880.11	-0.0012741
ENSG00000095906.17	-0.0012737
ENSG00000104980.8	-0.0012736
ENSG00000189306.11	-0.0012731
ENSG00000108604.16	-0.0012730
ENSG00000140939.14	-0.0012728
ENSG00000138433.16	-0.0012728

Gene	s0
ENSG00000140474.14	-0.0012726
ENSG00000015676.18	-0.0012719
ENSG00000123384.14	0.0012718
ENSG00000170892.12	-0.0012718
ENSG00000132485.14	-0.0012715
ENSG00000115207.13	-0.0012713
ENSG00000178397.13	-0.0012711
ENSG00000163798.13	-0.0012708
ENSG00000140157.15	-0.0012706
ENSG00000184831.14	-0.0012700
ENSG00000164609.10	-0.0012699
ENSG00000185651.15	-0.0012698
ENSG00000259974.3	0.0012695
ENSG00000115233.12	-0.0012694
ENSG00000167930.17	-0.0012689
ENSG00000161921.17	-0.0012688
ENSG00000128789.21	-0.0012682
ENSG00000134057.15	-0.0012681
ENSG00000113460.13	-0.0012679
ENSG00000011451.21	-0.0012678
ENSG00000119777.20	-0.0012677
ENSG00000186301.8	0.0012675
ENSG00000178307.10	-0.0012675
ENSG00000244731.8	0.0012675
ENSG00000130545.16	-0.0012674
ENSG00000187630.17	0.0012672
ENSG00000100029.18	-0.0012672
ENSG00000176783.15	-0.0012670
ENSG00000100442.11	-0.0012669
ENSG00000143537.14	-0.0012668
ENSG00000118246.14	-0.0012664
ENSG00000114982.19	-0.0012661
ENSG00000115514.12	-0.0012661
ENSG00000100243.21	-0.0012658
ENSG00000173702.7	-0.0012654
ENSG00000113732.9	-0.0012653
ENSG00000105723.13	-0.0012653
ENSG00000250722.6	0.0012653
ENSG00000172340.15	0.0012652
ENSG00000163754.18	-0.0012649
ENSG00000132612.16	-0.0012643

Gene	s0
ENSG00000141580.16	-0.0012642
ENSG00000176720.6	0.0012639
ENSG00000157181.16	-0.0012633
ENSG00000136448.13	-0.0012632
ENSG00000112146.17	-0.0012629
ENSG00000105612.9	-0.0012627
ENSG00000130985.17	-0.0012626
ENSG00000013288.9	-0.0012625
ENSG00000133884.10	-0.0012621
ENSG00000172500.13	-0.0012618
ENSG00000155380.12	0.0012618
ENSG00000087365.16	-0.0012618
ENSG00000170515.14	-0.0012615
ENSG00000092199.18	-0.0012612
ENSG00000051009.11	-0.0012611
ENSG00000170265.12	-0.0012611
ENSG00000196670.14	-0.0012607
ENSG00000138375.13	-0.0012606
ENSG00000116903.7	-0.0012605
ENSG00000082781.12	-0.0012604
ENSG00000137656.12	-0.0012600
ENSG00000132744.8	0.0012597
ENSG00000129214.15	0.0012597
ENSG00000108523.16	-0.0012593
ENSG00000110245.12	0.0012593
ENSG00000162236.13	-0.0012592
ENSG00000256612.7	0.0012592
ENSG00000165475.15	0.0012588
ENSG00000174227.16	-0.0012586
ENSG00000131475.7	-0.0012585
ENSG00000126461.15	-0.0012584
ENSG00000106853.20	0.0012582
ENSG00000079785.16	-0.0012582
ENSG00000078140.14	-0.0012581
ENSG00000169727.12	-0.0012579
ENSG00000107140.16	-0.0012575
ENSG00000114850.7	-0.0012570
ENSG00000043093.15	-0.0012569
ENSG00000116918.15	-0.0012568
ENSG00000134294.14	0.0012567
ENSG00000161091.13	-0.0012566

Gene	s0
ENSG00000108592.17	-0.0012563
ENSG00000088986.11	-0.0012563
ENSG00000113391.19	-0.0012559
ENSG00000169715.15	0.0012557
ENSG00000140995.17	-0.0012551
ENSG00000131153.9	-0.0012550
ENSG00000198848.13	0.0012550
ENSG00000279059.1	-0.0012550
ENSG00000166164.16	-0.0012548
ENSG00000164089.9	0.0012546
ENSG00000116704.8	0.0012543
ENSG00000141699.11	-0.0012542
ENSG00000108039.18	-0.0012541
ENSG00000065427.15	-0.0012540
ENSG00000162073.14	-0.0012539
ENSG00000170113.16	-0.0012539
ENSG00000170293.9	0.0012539
ENSG00000171155.8	-0.0012537
ENSG00000179091.5	-0.0012536
ENSG00000169221.14	-0.0012533
ENSG00000140553.18	-0.0012532
ENSG00000146535.14	-0.0012531
ENSG00000188566.14	-0.0012528
ENSG00000136238.18	-0.0012527
ENSG00000151806.14	-0.0012521
ENSG00000176142.13	-0.0012518
ENSG00000075856.12	-0.0012517
ENSG00000104131.13	-0.0012514
ENSG00000158195.11	-0.0012514
ENSG00000120253.14	-0.0012512
ENSG00000102241.12	-0.0012507
ENSG00000086589.12	-0.0012506
ENSG00000120029.13	-0.0012505
ENSG00000078808.19	-0.0012503
ENSG00000115446.12	-0.0012503
ENSG00000196588.20	-0.0012502
ENSG00000153786.13	-0.0012501
ENSG00000117984.15	-0.0012501
ENSG00000127184.13	-0.0012500
ENSG00000072849.11	-0.0012500
ENSG00000198077.10	0.0012499

Gene	s0
ENSG00000146281.6	-0.0012499
ENSG00000100865.15	-0.0012498
ENSG00000183624.14	-0.0012498
ENSG00000141753.7	0.0012497
ENSG00000170322.14	-0.0012494
ENSG00000121579.13	-0.0012494
ENSG00000123575.9	-0.0012490
ENSG00000168411.14	-0.0012489
ENSG00000124380.11	-0.0012489
ENSG00000177042.14	-0.0012488
ENSG00000122566.22	-0.0012487
ENSG00000102401.20	-0.0012487
ENSG00000087470.18	-0.0012486
ENSG00000100632.11	-0.0012486
ENSG00000077254.14	-0.0012485
ENSG00000120837.8	-0.0012481
ENSG00000093000.19	-0.0012480
ENSG00000136718.10	-0.0012480
ENSG00000115234.11	-0.0012480
ENSG00000130305.17	-0.0012478
ENSG00000213593.10	-0.0012477
ENSG00000146828.18	-0.0012476
ENSG00000148358.20	-0.0012470
ENSG00000159200.18	0.0012469
ENSG00000156802.13	-0.0012468
ENSG00000100410.8	-0.0012467
ENSG00000164631.19	-0.0012466
ENSG00000163161.14	-0.0012465
ENSG00000112294.14	0.0012460
ENSG00000163378.14	-0.0012460
ENSG00000181852.18	-0.0012459
ENSG00000205436.7	0.0012457
ENSG00000130158.14	-0.0012455
ENSG00000100938.18	-0.0012454
ENSG00000135723.14	-0.0012451
ENSG00000136457.10	0.0012449
ENSG00000196453.8	-0.0012448
ENSG00000086666.19	-0.0012448
ENSG00000230989.7	-0.0012446
ENSG00000175203.17	-0.0012443
ENSG00000130202.10	-0.0012441

Gene	s0
ENSG00000099942.13	-0.0012427
ENSG00000114416.18	-0.0012427
ENSG00000215717.7	-0.0012427
ENSG00000144524.18	-0.0012424
ENSG00000117054.14	0.0012424
ENSG00000185896.11	-0.0012422
ENSG00000102225.16	-0.0012422
ENSG00000187522.16	-0.0012419
ENSG00000175581.14	-0.0012413
ENSG00000214517.10	-0.0012409
ENSG00000083312.17	-0.0012404
ENSG00000189091.13	-0.0012404
ENSG00000099917.17	-0.0012401
ENSG00000101367.9	-0.0012399
ENSG00000101421.4	-0.0012397
ENSG00000205531.13	-0.0012397
ENSG00000132356.11	-0.0012395
ENSG00000127948.16	0.0012392
ENSG00000173011.12	-0.0012392
ENSG00000131100.13	-0.0012391
ENSG00000082213.18	-0.0012387
ENSG00000155959.12	-0.0012386
ENSG00000183513.9	-0.0012385
ENSG00000143207.21	-0.0012383
ENSG00000237190.4	-0.0012383
ENSG00000103249.18	-0.0012382
ENSG00000184640.19	-0.0012381
ENSG00000102858.13	-0.0012380
ENSG00000204469.13	-0.0012375
ENSG00000056097.16	-0.0012374
ENSG00000163872.16	-0.0012373
ENSG00000240344.9	-0.0012372
ENSG00000007202.15	-0.0012370
ENSG00000068903.20	-0.0012369
ENSG00000071626.17	-0.0012368
ENSG00000240230.6	-0.0012363
ENSG00000116698.21	-0.0012362
ENSG00000025770.19	-0.0012362
ENSG00000008311.15	0.0012359
ENSG00000140694.17	-0.0012359
ENSG00000122545.20	-0.0012356

Gene	s0
ENSG00000112851.15	-0.0012354
ENSG00000115128.7	-0.0012351
ENSG00000063244.13	-0.0012351
ENSG00000166855.9	0.0012350
ENSG00000144848.11	-0.0012348
ENSG00000155868.8	-0.0012348
ENSG00000099219.14	-0.0012346
ENSG00000149182.15	-0.0012346
ENSG00000164885.13	-0.0012345
ENSG00000146083.12	-0.0012345
ENSG00000170043.12	-0.0012341
ENSG00000113716.13	-0.0012339
ENSG00000111361.13	-0.0012338
ENSG00000241258.7	-0.0012337
ENSG00000228278.5	0.0012335
ENSG00000159202.18	-0.0012332
ENSG00000132024.17	-0.0012331
ENSG00000139433.11	-0.0012322
ENSG00000102317.18	-0.0012316
ENSG00000124541.7	-0.0012315
ENSG00000129351.17	-0.0012314
ENSG00000121766.16	-0.0012312
ENSG00000160877.6	-0.0012308
ENSG00000005175.10	-0.0012307
ENSG00000169223.15	-0.0012305
ENSG00000142409.6	-0.0012303
ENSG00000100220.12	-0.0012303
ENSG00000198356.12	-0.0012302
ENSG00000152291.14	-0.0012301
ENSG00000119801.13	-0.0012298
ENSG00000027847.14	-0.0012298
ENSG00000166037.11	-0.0012298
ENSG00000168393.13	-0.0012297
ENSG00000134049.6	-0.0012294
ENSG00000100804.19	-0.0012293
ENSG00000106012.18	-0.0012293
ENSG00000113387.12	-0.0012293
ENSG00000178234.13	-0.0012292
ENSG00000077235.18	-0.0012292
ENSG00000069329.18	-0.0012292
ENSG00000143933.19	-0.0012285

Gene	s0
ENSG00000131495.9	-0.0012283
ENSG00000130529.16	-0.0012282
ENSG00000184545.11	-0.0012278
ENSG00000158480.11	-0.0012277
ENSG00000177646.19	-0.0012274
ENSG00000106638.17	-0.0012272
ENSG00000140545.15	-0.0012270
ENSG00000131263.13	-0.0012270
ENSG00000126934.14	-0.0012268
ENSG00000188002.10	-0.0012266
ENSG00000104964.15	-0.0012266
ENSG00000068394.11	-0.0012266
ENSG00000198492.16	-0.0012265
ENSG00000105738.11	-0.0012265
ENSG00000165671.21	-0.0012265
ENSG00000182117.6	-0.0012261
ENSG00000165804.16	-0.0012260
ENSG00000162222.14	-0.0012256
ENSG00000171204.13	-0.0012249
ENSG00000241635.8	0.0012248
ENSG00000186660.15	-0.0012245
ENSG00000104142.11	-0.0012245
ENSG00000125912.11	-0.0012242
ENSG00000166166.13	-0.0012242
ENSG00000182180.14	-0.0012240
ENSG00000122203.15	-0.0012239
ENSG00000113013.16	-0.0012238
ENSG00000163781.14	-0.0012237
ENSG00000167701.14	0.0012237
ENSG00000050820.17	-0.0012236
ENSG00000116906.13	-0.0012236
ENSG00000185104.20	-0.0012233
ENSG00000103502.14	-0.0012232
ENSG00000150403.18	-0.0012231
ENSG00000157657.14	-0.0012231
ENSG00000105993.15	-0.0012230
ENSG00000227345.8	-0.0012230
ENSG00000172731.14	-0.0012229
ENSG00000034713.8	-0.0012229
ENSG00000121774.18	-0.0012228
ENSG00000014641.21	-0.0012225

Gene	s0
ENSG00000125505.17	-0.0012224
ENSG00000154582.17	-0.0012223
ENSG00000050130.18	-0.0012222
ENSG00000112186.12	-0.0012219
ENSG00000073712.15	0.0012213
ENSG00000148229.13	-0.0012211
ENSG00000065491.8	-0.0012211
ENSG00000241468.8	-0.0012209
ENSG00000168246.6	-0.0012207
ENSG00000100412.17	-0.0012206
ENSG00000173599.15	0.0012205
ENSG00000184983.11	-0.0012203
ENSG00000117614.10	-0.0012203
ENSG00000124228.14	-0.0012200
ENSG00000064652.11	-0.0012199
ENSG00000173875.14	-0.0012197
ENSG00000150316.12	-0.0012195
ENSG00000151882.11	-0.0012193
ENSG00000100462.16	-0.0012193
ENSG00000082515.18	-0.0012192
ENSG00000092010.15	-0.0012190
ENSG00000063322.15	-0.0012186
ENSG00000105447.13	-0.0012186
ENSG00000254470.3	-0.0012183
ENSG00000101216.11	-0.0012180
ENSG00000198380.13	-0.0012173
ENSG00000146109.5	-0.0012170
ENSG00000130204.13	-0.0012168
ENSG00000166685.12	-0.0012165
ENSG00000186222.5	-0.0012162
ENSG00000108774.15	-0.0012160
ENSG00000156026.14	-0.0012158
ENSG00000071051.14	-0.0012158
ENSG00000129103.19	-0.0012158
ENSG00000165660.8	-0.0012154
ENSG00000054611.14	-0.0012151
ENSG00000105576.15	-0.0012150
ENSG00000156096.14	0.0012148
ENSG00000105325.15	-0.0012145
ENSG00000166946.14	-0.0012142
ENSG00000184602.6	-0.0012141

Gene	s0
ENSG00000108788.12	-0.0012140
ENSG00000119574.13	-0.0012140
ENSG00000197694.18	-0.0012137
ENSG00000173511.10	-0.0012137
ENSG00000152990.14	0.0012130
ENSG00000197021.9	-0.0012130
ENSG00000109103.12	-0.0012130
ENSG00000144035.4	0.0012129
ENSG00000150756.14	-0.0012127
ENSG00000009950.16	0.0012127
ENSG00000253352.10	-0.0012127
ENSG00000179134.15	-0.0012126
ENSG00000032389.13	-0.0012126
ENSG00000131051.24	-0.0012124
ENSG00000050393.11	-0.0012124
ENSG00000154473.18	-0.0012123
ENSG00000143771.12	-0.0012122
ENSG00000110906.13	-0.0012119
ENSG00000061936.10	-0.0012117
ENSG00000161980.6	-0.0012114
ENSG00000164414.18	-0.0012113
ENSG00000114771.14	0.0012112
ENSG00000145868.17	-0.0012111
ENSG00000135587.9	-0.0012110
ENSG00000115839.18	-0.0012107
ENSG00000147257.15	-0.0012107
ENSG00000206527.10	-0.0012105
ENSG00000158604.15	-0.0012105
ENSG00000118363.12	-0.0012104
ENSG00000092201.10	-0.0012102
ENSG00000175166.17	-0.0012100
ENSG00000101997.13	-0.0012098
ENSG00000204220.12	-0.0012093
ENSG00000100982.12	-0.0012093
ENSG00000160190.14	-0.0012084
ENSG00000250535.1	-0.0012082
ENSG00000165782.11	-0.0012081
ENSG00000146842.17	-0.0012081
ENSG00000213676.13	-0.0012080
ENSG00000196262.15	-0.0012078
ENSG00000213903.9	-0.0012078

Gene	s0
ENSG00000103671.10	-0.0012078
ENSG00000163468.15	-0.0012076
ENSG00000106397.12	-0.0012074
ENSG00000119760.16	-0.0012073
ENSG00000156504.16	-0.0012072
ENSG00000142002.17	-0.0012071
ENSG00000164209.17	-0.0012070
ENSG00000158545.16	-0.0012068
ENSG00000149792.9	-0.0012064
ENSG00000132406.12	-0.0012063
ENSG00000160075.12	-0.0012061
ENSG00000155506.18	-0.0012061
ENSG00000170471.15	-0.0012060
ENSG00000141867.18	-0.0012058
ENSG00000183747.12	0.0012057
ENSG00000143845.15	0.0012056
ENSG00000105771.14	-0.0012056
ENSG00000009954.11	-0.0012056
ENSG00000175137.11	-0.0012055
ENSG00000197157.11	-0.0012055
ENSG00000155508.14	-0.0012052
ENSG00000066117.15	-0.0012049
ENSG00000157870.17	-0.0012049
ENSG00000058799.15	-0.0012049
ENSG00000115468.12	0.0012046
ENSG00000128272.15	-0.0012046
ENSG00000196141.14	-0.0012045
ENSG00000272398.6	-0.0012041
ENSG00000114988.12	-0.0012041
ENSG00000141030.13	-0.0012036
ENSG00000138095.19	-0.0012035
ENSG00000054148.18	-0.0012035
ENSG00000100216.6	-0.0012034
ENSG00000044574.8	-0.0012033
ENSG00000127452.9	-0.0012033
ENSG00000162613.17	-0.0012032
ENSG00000100348.10	-0.0012032
ENSG00000013561.18	-0.0012031
ENSG00000165637.13	-0.0012031
ENSG00000125755.19	-0.0012027
ENSG00000178605.13	-0.0012027

Gene	s0
ENSG00000089053.13	-0.0012026
ENSG00000166477.13	-0.0012025
ENSG00000231925.12	-0.0012021
ENSG00000149657.20	-0.0012017
ENSG00000174243.10	-0.0012016
ENSG00000005486.17	-0.0012012
ENSG00000185361.9	0.0012011
ENSG00000035928.16	-0.0012010
ENSG00000186298.12	-0.0012007
ENSG00000114544.17	-0.0012006
ENSG00000104824.17	-0.0012005
ENSG00000103275.21	-0.0012000
ENSG00000115966.17	-0.0012000
ENSG00000136897.8	-0.0011998
ENSG00000100575.14	-0.0011994
ENSG00000101294.18	-0.0011993
ENSG00000184164.15	-0.0011992
ENSG00000198952.8	-0.0011992
ENSG00000121390.19	-0.0011992
ENSG00000126768.12	-0.0011991
ENSG00000143878.10	0.0011991
ENSG00000136699.19	-0.0011991
ENSG00000136758.19	-0.0011989
ENSG00000103148.17	-0.0011989
ENSG00000087191.13	-0.0011989
ENSG00000106615.10	-0.0011988
ENSG00000099341.11	-0.0011988
ENSG00000130638.17	-0.0011987
ENSG00000023041.11	-0.0011986
ENSG00000165105.10	-0.0011986
ENSG00000180329.14	-0.0011985
ENSG00000143627.19	0.0011984
ENSG00000141562.18	-0.0011980
ENSG00000175707.9	-0.0011979
ENSG00000177963.14	-0.0011977
ENSG00000136436.15	-0.0011975
ENSG00000160087.21	-0.0011974
ENSG00000106052.13	-0.0011974
ENSG00000167085.13	-0.0011973
ENSG00000074071.15	-0.0011972
ENSG00000136731.13	-0.0011971

Gene	s0
ENSG00000031823.14	-0.0011969
ENSG00000101350.8	-0.0011968
ENSG00000206341.7	-0.0011968
ENSG00000099364.18	-0.0011968
ENSG00000166526.18	-0.0011967
ENSG00000158290.18	-0.0011967
ENSG00000203485.14	-0.0011967
ENSG00000146729.10	-0.0011965
ENSG00000141971.13	-0.0011964
ENSG00000148337.21	-0.0011963
ENSG00000087460.27	-0.0011961
ENSG00000197296.6	-0.0011956
ENSG00000182379.10	-0.0011956
ENSG00000102898.12	-0.0011951
ENSG00000082898.18	-0.0011951
ENSG00000090316.16	-0.0011948
ENSG00000129933.21	-0.0011948
ENSG00000100813.14	-0.0011945
ENSG00000101255.11	-0.0011943
ENSG00000103353.16	-0.0011943
ENSG00000102580.15	-0.0011942
ENSG00000118004.18	0.0011942
ENSG00000159377.11	-0.0011941
ENSG00000102978.13	-0.0011940
ENSG00000101266.19	-0.0011938
ENSG00000004779.10	-0.0011937
ENSG00000100353.18	-0.0011935
ENSG00000167766.18	-0.0011934
ENSG00000184009.12	-0.0011933
ENSG00000146670.10	-0.0011930
ENSG00000152683.14	-0.0011927
ENSG00000105968.19	-0.0011927
ENSG00000159579.14	-0.0011926
ENSG00000168385.18	-0.0011926
ENSG00000171421.13	-0.0011923
ENSG00000090581.10	-0.0011922
ENSG00000130559.19	-0.0011922
ENSG00000112984.12	-0.0011916
ENSG00000089335.21	-0.0011915
ENSG00000105127.9	-0.0011915
ENSG00000138073.14	-0.0011913

Gene	s0
ENSG00000108599.15	-0.0011912
ENSG00000151465.14	-0.0011912
ENSG00000090372.15	-0.0011910
ENSG00000149636.16	-0.0011910
ENSG00000065548.18	-0.0011907
ENSG00000100266.19	-0.0011899
ENSG00000176340.4	-0.0011898
ENSG00000177879.17	-0.0011896
ENSG00000150776.18	-0.0011895
ENSG00000039650.12	-0.0011895
ENSG00000167967.16	-0.0011894
ENSG00000147274.14	-0.0011894
ENSG00000114503.11	-0.0011891
ENSG00000146731.11	-0.0011890
ENSG00000101654.18	-0.0011889
ENSG00000167863.12	-0.0011887
ENSG00000198677.12	-0.0011886
ENSG00000164253.14	-0.0011886
ENSG00000106665.16	-0.0011885
ENSG00000141543.12	-0.0011885
ENSG00000100823.12	-0.0011884
ENSG00000094914.14	-0.0011883
ENSG00000113583.8	-0.0011882
ENSG00000130640.13	-0.0011882
ENSG00000143727.16	-0.0011877
ENSG00000198417.7	0.0011876
ENSG00000151116.17	-0.0011874
ENSG00000184752.13	-0.0011873
ENSG00000173207.13	-0.0011873
ENSG00000072803.17	-0.0011873
ENSG00000129071.10	-0.0011870
ENSG00000089289.16	-0.0011870
ENSG00000151655.19	0.0011864
ENSG00000134240.12	0.0011862
ENSG00000115350.12	-0.0011862
ENSG00000173065.13	-0.0011858
ENSG00000121022.14	-0.0011858
ENSG00000176407.18	-0.0011855
ENSG00000264281.3	-0.0011855
ENSG00000176108.9	-0.0011855
ENSG00000211460.12	-0.0011851

Gene	s0
ENSG00000132603.15	-0.0011850
ENSG00000134779.15	-0.0011850
ENSG00000076258.10	0.0011847
ENSG00000173821.19	-0.0011845
ENSG00000112167.10	-0.0011844
ENSG00000161021.13	-0.0011841
ENSG00000167657.14	-0.0011840
ENSG00000115241.11	-0.0011837
ENSG00000166925.9	-0.0011836
ENSG00000136521.13	-0.0011834
ENSG00000108468.15	-0.0011833
ENSG00000111786.9	-0.0011833
ENSG00000138032.21	-0.0011832
ENSG00000134597.16	-0.0011831
ENSG00000072401.15	-0.0011825
ENSG00000105732.13	-0.0011823
ENSG00000113048.17	-0.0011822
ENSG00000174938.14	-0.0011818
ENSG00000135686.13	-0.0011813
ENSG00000171903.16	0.0011808
ENSG00000144283.22	-0.0011807
ENSG00000129515.20	-0.0011806
ENSG00000115211.16	-0.0011805
ENSG00000103769.10	-0.0011804
ENSG00000129151.9	0.0011804
ENSG00000128989.11	-0.0011802
ENSG00000135127.11	-0.0011802
ENSG00000150990.8	-0.0011802
ENSG00000132591.12	-0.0011802
ENSG00000100347.15	-0.0011797
ENSG00000172889.16	-0.0011796
ENSG00000142751.15	-0.0011794
ENSG00000115539.14	-0.0011793
ENSG00000112640.16	-0.0011792
ENSG00000168259.17	-0.0011790
ENSG00000114383.10	-0.0011790
ENSG00000080189.15	-0.0011789
ENSG00000178691.11	-0.0011788
ENSG00000165119.21	-0.0011787
ENSG00000135924.16	-0.0011782
ENSG00000129518.9	-0.0011782

Gene	s0
ENSG00000100139.13	-0.0011781
ENSG00000173230.15	-0.0011776
ENSG00000158769.18	-0.0011776
ENSG00000100372.15	-0.0011775
ENSG00000115271.11	-0.0011775
ENSG00000125445.11	-0.0011774
ENSG00000127616.19	-0.0011770
ENSG00000140307.11	-0.0011767
ENSG00000120742.11	-0.0011765
ENSG00000100395.15	-0.0011765
ENSG00000184216.14	-0.0011763
ENSG00000126903.16	-0.0011760
ENSG00000064419.13	-0.0011760
ENSG00000166887.16	-0.0011759
ENSG00000114125.14	-0.0011758
ENSG00000039068.19	-0.0011757
ENSG00000113643.9	-0.0011757
ENSG00000070476.15	-0.0011755
ENSG00000111237.19	-0.0011755
ENSG00000038532.16	-0.0011754
ENSG00000184863.11	-0.0011753
ENSG00000187778.14	-0.0011753
ENSG00000090432.7	-0.0011753
ENSG00000184900.16	-0.0011750
ENSG00000127540.12	-0.0011749
ENSG00000135900.4	-0.0011748
ENSG00000004897.12	-0.0011748
ENSG00000198646.14	-0.0011744
ENSG00000174109.5	-0.0011744
ENSG00000167965.18	-0.0011743
ENSG00000135632.12	-0.0011741
ENSG00000205937.12	-0.0011741
ENSG00000068438.15	-0.0011741
ENSG00000129083.13	-0.0011737
ENSG00000197580.13	0.0011736
ENSG00000122565.19	-0.0011735
ENSG00000136986.10	-0.0011734
ENSG00000134697.13	-0.0011734
ENSG00000174996.12	-0.0011733
ENSG00000128191.16	-0.0011732
ENSG00000125249.7	-0.0011728

Gene	s0
ENSG00000115307.17	-0.0011725
ENSG00000179958.10	-0.0011724
ENSG00000099194.6	0.0011722
ENSG00000197045.13	-0.0011722
ENSG00000168434.13	-0.0011721
ENSG00000170759.11	-0.0011715
ENSG00000158417.11	-0.0011715
ENSG00000134590.14	-0.0011714
ENSG00000140829.12	-0.0011712
ENSG00000006744.19	-0.0011711
ENSG00000135974.10	-0.0011710
ENSG00000105438.9	-0.0011710
ENSG00000018610.15	-0.0011709
ENSG00000145214.14	-0.0011707
ENSG00000126524.10	-0.0011702
ENSG00000116199.12	-0.0011701
ENSG00000103194.16	-0.0011701
ENSG00000198168.9	-0.0011699
ENSG00000102893.16	-0.0011699
ENSG00000121058.5	-0.0011697
ENSG00000139168.9	-0.0011695
ENSG00000125351.12	-0.0011694
ENSG00000149357.10	-0.0011694
ENSG00000167118.11	-0.0011688
ENSG00000135972.9	-0.0011688
ENSG00000109854.14	-0.0011687
ENSG00000175324.10	-0.0011686
ENSG00000134056.12	-0.0011686
ENSG00000124207.17	-0.0011685
ENSG00000131779.11	-0.0011685
ENSG00000147162.14	-0.0011682
ENSG00000005194.15	-0.0011679
ENSG00000064225.12	0.0011677
ENSG00000135457.10	-0.0011675
ENSG00000136518.17	-0.0011670
ENSG00000189043.10	-0.0011668
ENSG00000172269.19	-0.0011667
ENSG00000077514.9	-0.0011667
ENSG00000144580.14	-0.0011666
ENSG00000166165.13	-0.0011663
ENSG00000136709.12	-0.0011663

Gene	s0
ENSG00000197879.17	-0.0011663
ENSG00000175573.7	-0.0011661
ENSG00000168066.20	-0.0011658
ENSG00000104812.15	-0.0011653
ENSG00000077380.16	-0.0011651
ENSG00000176102.13	-0.0011649
ENSG00000066583.12	0.0011648
ENSG00000204356.14	-0.0011647
ENSG00000101199.13	-0.0011646
ENSG00000130749.10	-0.0011643
ENSG00000127334.11	-0.0011643
ENSG00000275700.5	-0.0011642
ENSG00000101138.12	-0.0011641
ENSG00000108094.16	-0.0011637
ENSG00000148343.18	-0.0011637
ENSG00000108671.11	-0.0011634
ENSG00000103274.11	-0.0011634
ENSG00000163993.7	-0.0011634
ENSG00000124659.7	-0.0011632
ENSG00000155115.7	-0.0011631
ENSG00000104805.16	-0.0011629
ENSG00000005471.19	0.0011626
ENSG00000138078.16	-0.0011626
ENSG00000125868.16	-0.0011626
ENSG00000170027.7	-0.0011625
ENSG00000132254.13	-0.0011621
ENSG00000138629.16	-0.0011621
ENSG00000141076.18	-0.0011618
ENSG00000063245.15	-0.0011616
ENSG00000077232.18	-0.0011616
ENSG00000204604.12	-0.0011614
ENSG00000130741.11	-0.0011612
ENSG00000186566.13	-0.0011611
ENSG00000168389.18	0.0011607
ENSG00000101363.12	-0.0011607
ENSG00000119689.15	-0.0011606
ENSG00000104671.8	-0.0011606
ENSG00000165092.13	0.0011604
ENSG00000076003.5	-0.0011603
ENSG00000167778.9	-0.0011600
ENSG00000168476.12	-0.0011600

Gene	s0
ENSG00000105887.11	-0.0011600
ENSG00000125648.15	-0.0011600
ENSG00000198663.17	-0.0011598
ENSG00000118579.13	-0.0011598
ENSG00000146223.16	-0.0011597
ENSG00000113658.18	-0.0011596
ENSG00000078699.22	-0.0011596
ENSG00000144118.14	-0.0011591
ENSG00000071553.18	-0.0011588
ENSG00000125449.7	-0.0011588
ENSG00000164266.11	-0.0011587
ENSG00000123159.16	-0.0011587
ENSG00000101246.20	-0.0011584
ENSG00000172725.14	-0.0011583
ENSG00000177030.17	-0.0011582
ENSG00000124568.12	0.0011578
ENSG00000107862.6	-0.0011576
ENSG00000047249.18	-0.0011575
ENSG00000103415.12	-0.0011575
ENSG00000183751.15	-0.0011573
ENSG00000162817.7	0.0011572
ENSG00000167513.9	-0.0011571
ENSG00000139645.10	-0.0011569
ENSG00000247077.7	-0.0011567
ENSG00000152894.15	-0.0011566
ENSG00000009335.18	-0.0011563
ENSG00000099381.18	-0.0011563
ENSG00000174165.8	-0.0011562
ENSG00000166889.14	-0.0011559
ENSG00000204673.11	-0.0011558
ENSG00000166783.22	-0.0011557
ENSG00000160959.8	-0.0011554
ENSG00000278730.1	-0.0011554
ENSG00000132676.16	-0.0011552
ENSG00000122965.11	-0.0011552
ENSG00000162458.13	-0.0011551
ENSG00000152684.11	-0.0011551
ENSG00000276293.4	-0.0011550
ENSG00000197324.9	-0.0011550
ENSG00000079246.16	-0.0011548
ENSG00000125534.10	-0.0011548

Gene	s0
ENSG00000166794.5	-0.0011546
ENSG00000174327.7	0.0011543
ENSG00000177731.16	-0.0011540
ENSG00000114737.16	0.0011537
ENSG00000168958.20	-0.0011536
ENSG00000083750.13	-0.0011535
ENSG00000120697.9	-0.0011534
ENSG00000100439.10	-0.0011533
ENSG00000143612.21	-0.0011532
ENSG00000148297.16	-0.0011531
ENSG00000198026.8	-0.0011530
ENSG00000275285.1	0.0011526
ENSG00000158863.22	-0.0011523
ENSG00000167186.11	-0.0011523
ENSG00000100324.14	-0.0011520
ENSG00000126247.11	-0.0011520
ENSG00000144021.3	-0.0011519
ENSG00000122126.17	-0.0011518
ENSG00000112062.11	-0.0011514
ENSG00000196305.18	-0.0011512
ENSG00000129473.11	-0.0011512
ENSG00000006125.18	-0.0011512
ENSG00000109606.13	-0.0011511
ENSG00000100991.12	-0.0011511
ENSG00000104325.7	0.0011510
ENSG00000282826.2	-0.0011510
ENSG00000145293.16	-0.0011509
ENSG00000115649.16	-0.0011509
ENSG00000106635.8	-0.0011509
ENSG00000264364.3	-0.0011508
ENSG00000125977.7	-0.0011507
ENSG00000139291.14	-0.0011507
ENSG00000177239.15	-0.0011505
ENSG00000166266.14	-0.0011505
ENSG00000120686.12	-0.0011505
ENSG00000084463.8	-0.0011504
ENSG00000146540.15	-0.0011504
ENSG00000108509.21	-0.0011501
ENSG00000169903.7	-0.0011500
ENSG00000099995.19	-0.0011500
ENSG00000126005.17	-0.0011498

Gene	s0
ENSG00000149639.15	-0.0011495
ENSG00000140941.14	-0.0011495
ENSG00000170619.10	-0.0011495
ENSG00000125447.18	-0.0011494
ENSG00000166471.11	-0.0011494
ENSG00000142453.12	-0.0011494
ENSG00000266472.6	-0.0011493
ENSG00000183726.11	-0.0011489
ENSG00000131507.11	-0.0011479
ENSG00000061794.13	-0.0011478
ENSG00000163479.14	-0.0011477
ENSG00000151148.14	-0.0011476
ENSG00000140497.16	-0.0011476
ENSG00000135387.21	-0.0011475
ENSG00000104660.19	-0.0011472
ENSG00000156831.8	-0.0011469
ENSG00000182952.5	-0.0011467
ENSG00000115053.17	-0.0011467
ENSG00000163257.11	-0.0011466
ENSG00000143643.13	-0.0011464
ENSG00000119888.11	-0.0011462
ENSG00000176171.11	0.0011462
ENSG00000167325.15	-0.0011457
ENSG00000177169.10	-0.0011455
ENSG00000108651.10	-0.0011454
ENSG00000183161.6	-0.0011454
ENSG00000101126.18	-0.0011453
ENSG00000204348.10	-0.0011452
ENSG00000173801.17	-0.0011450
ENSG00000050327.15	-0.0011447
ENSG00000244462.8	-0.0011447
ENSG00000104635.15	0.0011446
ENSG00000186212.4	0.0011446
ENSG00000145912.9	-0.0011445
ENSG00000088298.13	-0.0011445
ENSG00000100221.10	-0.0011445
ENSG00000137310.12	-0.0011443
ENSG00000070495.15	-0.0011437
ENSG00000106263.18	-0.0011436
ENSG00000102054.18	-0.0011435
ENSG00000124783.14	-0.0011435

Gene	s0
ENSG00000165322.18	-0.0011434
ENSG00000277258.5	-0.0011434
ENSG00000188612.12	-0.0011427
ENSG00000221869.5	0.0011426
ENSG00000168522.13	-0.0011425
ENSG00000116690.13	0.0011425
ENSG00000172239.14	-0.0011424
ENSG00000167291.16	-0.0011422
ENSG00000242299.1	-0.0011420
ENSG00000115762.16	-0.0011420
ENSG00000023839.12	0.0011420
ENSG00000187097.13	0.0011418
ENSG00000103202.13	-0.0011417
ENSG00000130726.12	-0.0011417
ENSG00000123200.17	-0.0011416
ENSG00000055950.16	-0.0011414
ENSG00000143621.17	-0.0011414
ENSG00000156110.15	0.0011413
ENSG00000167900.12	-0.0011412
ENSG00000159479.17	-0.0011411
ENSG00000160917.15	-0.0011408
ENSG00000177946.6	-0.0011408
ENSG00000151414.15	-0.0011407
ENSG00000101558.14	-0.0011404
ENSG00000172757.13	-0.0011403
ENSG00000140262.18	-0.0011402
ENSG00000255112.3	-0.0011400
ENSG00000130733.11	-0.0011399
ENSG00000172531.16	-0.0011399
ENSG00000257103.8	-0.0011398
ENSG00000131482.10	0.0011394
ENSG00000241973.11	-0.0011394
ENSG00000156976.17	-0.0011393
ENSG00000081791.9	-0.0011384
ENSG00000136485.15	-0.0011383
ENSG00000100403.12	-0.0011382
ENSG00000164877.19	-0.0011382
ENSG00000055917.16	-0.0011382
ENSG00000172661.19	-0.0011382
ENSG00000132823.11	-0.0011381
ENSG00000167323.11	-0.0011380

Gene	s0
ENSG00000157837.16	-0.0011377
ENSG00000101181.17	-0.0011376
ENSG00000166145.14	-0.0011375
ENSG00000070081.17	-0.0011373
ENSG00000168488.18	-0.0011371
ENSG00000149136.9	-0.0011371
ENSG00000119446.14	-0.0011371
ENSG00000129315.11	-0.0011370
ENSG00000125753.14	-0.0011370
ENSG00000184708.18	-0.0011369
ENSG00000041357.16	-0.0011368
ENSG00000177427.13	-0.0011368
ENSG00000149269.10	-0.0011365
ENSG00000088256.9	-0.0011365
ENSG00000007933.13	0.0011365
ENSG00000119048.8	-0.0011364
ENSG00000133056.13	-0.0011364
ENSG00000100596.7	-0.0011364
ENSG00000039123.16	-0.0011363
ENSG00000125731.13	-0.0011361
ENSG00000166801.16	-0.0011359
ENSG00000102178.13	-0.0011358
ENSG00000226479.4	-0.0011358
ENSG00000165733.8	-0.0011358
ENSG00000079950.14	-0.0011353
ENSG00000168569.8	-0.0011351
ENSG00000198791.12	-0.0011351
ENSG00000106367.15	-0.0011349
ENSG00000145833.16	-0.0011349
ENSG00000144028.15	-0.0011347
ENSG00000084652.16	-0.0011347
ENSG00000111707.12	-0.0011347
ENSG00000112081.17	-0.0011345
ENSG00000198863.7	-0.0011344
ENSG00000175334.8	-0.0011343
ENSG00000112787.13	-0.0011338
ENSG00000196547.15	-0.0011338
ENSG00000196850.6	-0.0011337
ENSG00000169592.15	-0.0011336
ENSG00000001036.14	-0.0011336
ENSG00000125834.13	-0.0011336

Gene	s0
ENSG00000134265.13	-0.0011334
ENSG00000162430.17	-0.0011332
ENSG00000288670.1	-0.0011332
ENSG00000143819.12	0.0011332
ENSG00000125633.11	-0.0011331
ENSG00000112118.20	-0.0011330
ENSG00000106028.11	-0.0011327
ENSG00000066322.15	-0.0011327
ENSG00000169241.19	-0.0011327
ENSG00000139641.13	-0.0011325
ENSG00000101365.21	-0.0011323
ENSG00000163156.12	-0.0011323
ENSG00000131174.7	-0.0011322
ENSG00000124802.12	-0.0011322
ENSG00000254986.7	-0.0011322
ENSG00000184162.15	-0.0011321
ENSG00000177370.5	-0.0011321
ENSG00000088833.17	-0.0011321
ENSG00000101310.17	-0.0011318
ENSG00000166200.15	-0.0011318
ENSG00000111670.16	-0.0011318
ENSG00000160410.15	-0.0011318
ENSG00000134882.16	-0.0011317
ENSG00000198522.14	-0.0011317
ENSG00000185305.11	-0.0011316
ENSG00000218426.5	-0.0011316
ENSG00000113889.14	0.0011313
ENSG00000159111.13	-0.0011311
ENSG00000167088.11	-0.0011310
ENSG00000147155.11	0.0011307
ENSG00000235174.1	-0.0011305
ENSG00000075711.21	-0.0011304
ENSG00000162819.12	-0.0011303
ENSG00000104442.10	-0.0011303
ENSG00000244734.4	0.0011303
ENSG00000185551.15	-0.0011302
ENSG00000137842.7	-0.0011301
ENSG00000167608.12	-0.0011301
ENSG00000090615.15	-0.0011301
ENSG00000124562.10	-0.0011301
ENSG00000086598.11	-0.0011300

Gene	s0
ENSG00000144591.19	-0.0011300
ENSG00000134375.11	-0.0011299
ENSG00000171843.17	-0.0011297
ENSG00000175806.15	0.0011295
ENSG00000125875.14	-0.0011294
ENSG00000147669.11	-0.0011292
ENSG00000104413.18	-0.0011292
ENSG00000066136.20	-0.0011290
ENSG00000210144.1	-0.0011289
ENSG00000119927.14	0.0011288
ENSG00000060069.17	-0.0011281
ENSG00000112796.10	-0.0011280
ENSG00000185800.12	-0.0011276
ENSG00000141582.15	-0.0011276
ENSG00000126858.18	-0.0011276
ENSG00000126883.17	-0.0011276
ENSG00000179562.3	-0.0011275
ENSG00000108176.15	0.0011274
ENSG00000176953.12	-0.0011274
ENSG00000129559.13	-0.0011272
ENSG00000115275.14	-0.0011272
ENSG00000152904.11	-0.0011271
ENSG00000101146.13	-0.0011271
ENSG00000129636.13	-0.0011270
ENSG00000214114.9	-0.0011267
ENSG00000138385.16	-0.0011266
ENSG00000166454.10	-0.0011266
ENSG00000159348.13	-0.0011265
ENSG00000108344.15	-0.0011262
ENSG00000133703.13	-0.0011256
ENSG00000162909.18	-0.0011255
ENSG00000128228.5	-0.0011254
ENSG00000067596.12	-0.0011252
ENSG00000172053.18	-0.0011251
ENSG00000182307.14	-0.0011247
ENSG00000101391.21	-0.0011247
ENSG00000141568.21	-0.0011247
ENSG00000146425.11	-0.0011241
ENSG00000161800.13	-0.0011241
ENSG00000181929.13	-0.0011240
ENSG00000069509.6	-0.0011240

Gene	s0
ENSG00000012963.15	-0.0011239
ENSG00000168509.20	0.0011235
ENSG00000133318.14	-0.0011234
ENSG00000055483.19	-0.0011230
ENSG00000070831.17	-0.0011226
ENSG00000172336.5	-0.0011225
ENSG00000165458.14	-0.0011224
ENSG00000167315.18	0.0011221
ENSG00000167670.16	-0.0011221
ENSG00000151725.12	-0.0011220
ENSG00000145907.16	-0.0011219
ENSG00000183520.12	-0.0011219
ENSG00000136628.18	-0.0011218
ENSG00000196655.12	-0.0011217
ENSG00000085760.15	-0.0011217
ENSG00000087586.18	-0.0011217
ENSG00000179151.13	-0.0011216
ENSG00000125871.14	-0.0011215
ENSG00000103266.11	-0.0011214
ENSG00000105552.15	-0.0011213
ENSG00000204590.12	-0.0011211
ENSG00000119684.16	-0.0011210
ENSG00000160360.13	-0.0011209
ENSG00000174744.14	-0.0011208
ENSG00000060339.14	-0.0011206
ENSG00000128335.14	-0.0011205
ENSG00000259956.2	-0.0011203
ENSG00000000419.13	-0.0011202
ENSG00000138071.14	-0.0011197
ENSG00000122643.22	-0.0011196
ENSG00000173418.12	-0.0011195
ENSG00000159166.15	-0.0011195
ENSG00000162521.19	-0.0011194
ENSG00000170312.16	-0.0011193
ENSG00000116521.11	-0.0011192
ENSG00000162704.16	-0.0011189
ENSG00000250317.8	-0.0011188
ENSG00000147853.17	0.0011185
ENSG00000132670.20	-0.0011185
ENSG00000204209.13	-0.0011177
ENSG00000088247.17	-0.0011174

Gene	s0
ENSG00000128908.17	-0.0011174
ENSG00000175455.16	-0.0011173
ENSG00000171302.17	-0.0011172
ENSG00000204315.4	-0.0011171
ENSG00000134058.12	-0.0011168
ENSG00000105197.11	-0.0011167
ENSG00000185504.17	-0.0011165
ENSG00000100151.16	-0.0011163
ENSG00000078369.18	-0.0011162
ENSG00000158864.13	-0.0011162
ENSG00000068308.14	-0.0011162
ENSG00000172732.12	-0.0011162
ENSG00000070882.13	-0.0011161
ENSG00000150753.12	-0.0011160
ENSG00000227500.10	-0.0011160
ENSG00000168495.13	-0.0011160
ENSG00000072682.19	-0.0011158
ENSG00000136279.21	-0.0011157
ENSG00000160446.19	-0.0011152
ENSG00000006715.16	-0.0011152
ENSG00000070501.12	-0.0011152
ENSG00000047579.20	-0.0011150
ENSG00000079432.7	-0.0011149
ENSG00000105705.16	-0.0011147
ENSG00000181523.13	-0.0011147
ENSG00000131748.16	-0.0011146
ENSG00000168887.11	-0.0011146
ENSG00000134453.16	-0.0011144
ENSG00000106244.13	-0.0011143
ENSG00000143862.8	-0.0011143
ENSG00000158571.11	0.0011139
ENSG00000198888.2	0.0011139
ENSG00000122783.17	-0.0011139
ENSG00000135932.11	-0.0011136
ENSG00000204070.10	-0.0011135
ENSG00000168216.13	-0.0011135
ENSG00000180891.13	-0.0011133
ENSG00000104973.19	-0.0011127
ENSG00000172586.8	-0.0011126
ENSG00000111481.10	-0.0011123
ENSG00000173575.22	-0.0011122

Gene	s0
ENSG00000107949.17	-0.0011122
ENSG00000198721.13	0.0011122
ENSG00000151502.11	-0.0011121
ENSG00000234745.12	-0.0011120
ENSG00000120805.14	-0.0011120
ENSG00000086758.16	-0.0011117
ENSG00000105486.14	-0.0011116
ENSG00000176624.12	-0.0011114
ENSG00000096401.8	-0.0011114
ENSG00000174953.14	-0.0011114
ENSG00000228474.6	-0.0011112
ENSG00000141428.17	-0.0011111
ENSG00000158941.16	-0.0011107
ENSG00000132341.12	-0.0011107
ENSG00000215305.10	-0.0011103
ENSG00000075151.21	-0.0011103
ENSG00000214194.9	-0.0011101
ENSG00000123989.14	-0.0011099
ENSG00000135506.16	-0.0011097
ENSG00000129625.13	-0.0011097
ENSG00000136451.9	-0.0011095
ENSG00000080618.17	0.0011092
ENSG00000117222.14	-0.0011091
ENSG00000168612.4	-0.0011091
ENSG00000171792.11	-0.0011088
ENSG00000101608.13	-0.0011087
ENSG00000119203.14	-0.0011085
ENSG00000138430.16	-0.0011081
ENSG00000185825.17	-0.0011081
ENSG00000115073.8	-0.0011080
ENSG00000156697.13	-0.0011078
ENSG00000125450.11	-0.0011076
ENSG00000163785.13	-0.0011073
ENSG00000140395.9	-0.0011072
ENSG00000265354.4	-0.0011072
ENSG00000099956.20	-0.0011072
ENSG00000126767.18	-0.0011071
ENSG00000011243.18	-0.0011070
ENSG00000107021.16	-0.0011069
ENSG00000148688.14	-0.0011068
ENSG00000164535.15	-0.0011066

Gene	s0
ENSG00000155975.10	-0.0011064
ENSG00000197894.11	0.0011064
ENSG00000090863.12	-0.0011064
ENSG00000073050.12	-0.0011064
ENSG00000173786.17	-0.0011060
ENSG00000013810.20	-0.0011060
ENSG00000156261.13	-0.0011060
ENSG00000229320.3	-0.0011060
ENSG00000108091.11	-0.0011059
ENSG00000123136.15	-0.0011058
ENSG00000140854.13	-0.0011058
ENSG00000164576.12	-0.0011057
ENSG00000226084.5	-0.0011056
ENSG00000185753.13	-0.0011053
ENSG00000165527.7	-0.0011051
ENSG00000105321.14	-0.0011050
ENSG00000134308.14	-0.0011050
ENSG00000135677.11	-0.0011044
ENSG00000130299.17	-0.0011044
ENSG00000116678.20	0.0011043
ENSG00000103978.16	-0.0011043
ENSG00000124596.17	-0.0011043
ENSG00000029993.15	-0.0011041
ENSG00000172613.8	-0.0011039
ENSG00000114956.20	-0.0011038
ENSG00000168101.14	-0.0011038
ENSG00000204843.12	-0.0011038
ENSG00000101161.8	-0.0011037
ENSG00000085982.14	-0.0011036
ENSG00000083123.15	0.0011035
ENSG00000185627.18	-0.0011033
ENSG00000101189.7	-0.0011032
ENSG00000164889.15	-0.0011030
ENSG00000135164.19	-0.0011029
ENSG00000010610.10	0.0011028
ENSG00000102981.10	-0.0011027
ENSG00000172172.8	-0.0011026
ENSG00000206053.13	-0.0011026
ENSG00000168792.5	0.0011026
ENSG00000168701.19	-0.0011021
ENSG00000113621.15	-0.0011017

Gene	s0
ENSG00000132463.14	-0.0011017
ENSG00000148719.15	-0.0011017
ENSG00000123130.17	-0.0011013
ENSG00000133398.4	-0.0011010
ENSG00000132664.12	-0.0011010
ENSG00000090534.20	0.0011010
ENSG00000070814.22	-0.0011009
ENSG00000116285.13	0.0011008
ENSG00000102119.11	-0.0011007
ENSG00000081177.18	-0.0011006
ENSG00000160551.12	-0.0011006
ENSG00000100014.20	-0.0011005
ENSG00000106524.9	-0.0011004
ENSG00000140400.17	-0.0011004
ENSG00000103126.15	-0.0011004
ENSG00000130520.11	-0.0011002
ENSG00000120705.13	-0.0011001
ENSG00000105229.7	-0.0011001
ENSG00000107651.13	-0.0011000
ENSG00000105063.19	-0.0011000
ENSG00000138459.9	-0.0011000
ENSG00000143811.19	-0.0010998
ENSG00000136270.14	-0.0010998
ENSG00000137547.9	-0.0010997
ENSG00000176261.15	-0.0010996
ENSG00000004478.8	-0.0010996
ENSG00000101158.15	-0.0010994
ENSG00000225507.1	-0.0010993
ENSG00000184076.13	-0.0010993
ENSG00000106290.15	-0.0010993
ENSG00000151923.17	-0.0010989
ENSG00000178982.10	-0.0010989
ENSG00000227671.4	-0.0010987
ENSG00000143368.10	-0.0010987
ENSG00000117868.17	-0.0010987
ENSG00000028528.15	-0.0010985
ENSG00000100568.11	-0.0010984
ENSG00000183742.13	-0.0010982
ENSG00000153879.9	-0.0010981
ENSG00000174943.11	-0.0010980
ENSG00000204387.14	-0.0010976

Gene	s0
ENSG00000139624.14	-0.0010976
ENSG00000100605.17	-0.0010976
ENSG00000186283.14	-0.0010974
ENSG00000256269.10	-0.0010972
ENSG00000100426.7	-0.0010971
ENSG00000118961.15	-0.0010971
ENSG00000170633.16	-0.0010968
ENSG00000133858.17	-0.0010968
ENSG00000198700.10	-0.0010967
ENSG00000104687.14	-0.0010967
ENSG00000188428.20	-0.0010967
ENSG00000137876.11	-0.0010965
ENSG00000100522.10	0.0010962
ENSG00000162607.13	-0.0010961
ENSG00000185246.18	-0.0010961
ENSG00000159069.14	-0.0010961
ENSG00000115946.8	-0.0010961
ENSG00000037042.9	-0.0010960
ENSG00000120733.14	-0.0010959
ENSG00000116120.11	-0.0010956
ENSG00000120963.12	-0.0010955
ENSG00000182628.13	-0.0010954
ENSG00000197006.14	-0.0010953
ENSG00000125817.8	-0.0010949
ENSG00000189159.16	-0.0010948
ENSG00000101166.16	-0.0010948
ENSG00000101407.13	-0.0010948
ENSG00000074319.13	-0.0010942
ENSG00000143553.11	-0.0010942
ENSG00000168175.15	-0.0010938
ENSG00000168096.15	-0.0010937
ENSG00000139531.13	0.0010936
ENSG00000267317.2	-0.0010932
ENSG00000184500.16	0.0010932
ENSG00000088356.6	-0.0010932
ENSG00000151690.15	-0.0010931
ENSG00000104880.18	-0.0010930
ENSG00000136045.12	-0.0010929
ENSG00000143793.13	-0.0010928
ENSG00000005893.16	-0.0010927
ENSG00000114107.9	-0.0010927

Gene	s0
ENSG00000111725.11	-0.0010927
ENSG00000136146.15	-0.0010926
ENSG00000155229.21	-0.0010925
ENSG00000158526.8	-0.0010924
ENSG00000152464.15	-0.0010923
ENSG00000175274.19	-0.0010923
ENSG00000151176.8	-0.0010923
ENSG00000040531.16	-0.0010922
ENSG00000244067.3	0.0010922
ENSG00000140416.23	-0.0010920
ENSG00000017797.13	-0.0010920
ENSG00000001497.18	-0.0010920
ENSG00000177733.7	-0.0010919
ENSG00000111843.14	-0.0010919
ENSG00000064607.17	-0.0010918
ENSG00000197147.15	-0.0010917
ENSG00000214022.11	-0.0010917
ENSG00000106105.15	-0.0010917
ENSG00000102934.10	-0.0010917
ENSG00000172057.10	0.0010916
ENSG00000162522.11	-0.0010914
ENSG00000155827.12	-0.0010914
ENSG00000171865.10	-0.0010911
ENSG00000108021.20	-0.0010909
ENSG00000132432.14	-0.0010902
ENSG00000125733.18	-0.0010902
ENSG00000171456.20	-0.0010902
ENSG00000141577.14	-0.0010901
ENSG00000126775.9	-0.0010899
ENSG00000119280.17	-0.0010899
ENSG00000159352.16	-0.0010899
ENSG00000106459.15	-0.0010896
ENSG00000178038.17	-0.0010895
ENSG00000141456.16	-0.0010894
ENSG00000149531.15	-0.0010893
ENSG00000170291.15	-0.0010891
ENSG00000164896.20	-0.0010889
ENSG00000161526.15	-0.0010888
ENSG00000160932.11	0.0010887
ENSG00000127804.13	-0.0010887
ENSG00000171163.16	-0.0010887

Gene	s0
ENSG00000128245.15	-0.0010887
ENSG00000146802.13	-0.0010884
ENSG00000117410.14	-0.0010883
ENSG00000112679.14	-0.0010882
ENSG00000101343.15	-0.0010879
ENSG00000168264.10	-0.0010877
ENSG00000141698.17	-0.0010876
ENSG00000204371.11	-0.0010875
ENSG00000136271.10	-0.0010874
ENSG00000167461.12	-0.0010873
ENSG00000167182.15	-0.0010872
ENSG00000166033.13	-0.0010871
ENSG00000138363.15	-0.0010870
ENSG00000141446.11	-0.0010867
ENSG00000198551.10	-0.0010864
ENSG00000134815.19	-0.0010864
ENSG00000120451.11	-0.0010863
ENSG00000170852.11	-0.0010862
ENSG00000162407.9	0.0010861
ENSG00000171813.14	-0.0010861
ENSG00000141644.17	-0.0010861
ENSG00000138594.14	-0.0010860
ENSG00000148296.7	-0.0010859
ENSG00000104388.15	-0.0010858
ENSG00000014216.16	-0.0010858
ENSG00000113712.19	-0.0010857
ENSG00000111731.12	-0.0010857
ENSG00000150459.12	-0.0010855
ENSG00000068697.7	-0.0010855
ENSG00000139350.12	-0.0010853
ENSG00000178952.11	-0.0010852
ENSG00000171219.9	-0.0010851
ENSG00000124193.16	-0.0010851
ENSG00000170385.10	0.0010847
ENSG00000198836.11	-0.0010847
ENSG00000167986.14	-0.0010847
ENSG00000107960.11	-0.0010845
ENSG00000144589.22	-0.0010845
ENSG00000100106.22	-0.0010845
ENSG00000004059.11	-0.0010842
ENSG00000100911.16	-0.0010841

Gene	s0
ENSG00000171681.12	-0.0010838
ENSG00000160688.19	-0.0010836
ENSG00000125741.5	-0.0010836
ENSG00000122882.11	-0.0010834
ENSG00000116171.19	0.0010833
ENSG00000167005.14	-0.0010832
ENSG00000185043.12	-0.0010829
ENSG00000116489.13	-0.0010828
ENSG00000198960.11	-0.0010826
ENSG00000139746.16	-0.0010825
ENSG00000099783.12	-0.0010825
ENSG00000011260.14	-0.0010825
ENSG00000175220.12	-0.0010824
ENSG00000137274.13	0.0010824
ENSG00000135250.17	-0.0010823
ENSG00000138036.18	-0.0010822
ENSG00000168994.14	0.0010820
ENSG00000179262.10	-0.0010820
ENSG00000182473.22	-0.0010820
ENSG00000176641.11	0.0010820
ENSG00000142892.15	-0.0010820
ENSG00000115484.15	-0.0010818
ENSG00000086232.13	-0.0010815
ENSG00000101856.10	0.0010815
ENSG00000107937.19	-0.0010812
ENSG00000112335.15	-0.0010810
ENSG00000070614.15	-0.0010810
ENSG00000158411.11	-0.0010806
ENSG00000036549.13	-0.0010804
ENSG00000089154.11	-0.0010803
ENSG00000145012.13	-0.0010802
ENSG00000136754.18	-0.0010800
ENSG00000104365.16	-0.0010799
ENSG00000171853.16	-0.0010799
ENSG00000253570.1	-0.0010797
ENSG00000160209.19	-0.0010797
ENSG00000237649.8	-0.0010796
ENSG00000172432.19	-0.0010795
ENSG00000125686.12	-0.0010792
ENSG00000106077.18	-0.0010789
ENSG00000265808.4	-0.0010789

Gene	s0
ENSG00000269867.1	-0.0010788
ENSG00000166484.20	-0.0010787
ENSG00000160310.18	-0.0010787
ENSG00000076555.15	0.0010786
ENSG00000169905.13	-0.0010786
ENSG00000108528.14	-0.0010784
ENSG00000123352.18	-0.0010784
ENSG00000130349.10	-0.0010782
ENSG00000143575.15	-0.0010782
ENSG00000124172.10	-0.0010782
ENSG00000140264.21	-0.0010781
ENSG00000125430.9	0.0010780
ENSG00000117153.16	-0.0010777
ENSG00000142039.4	-0.0010773
ENSG00000105701.16	-0.0010773
ENSG00000168894.10	-0.0010772
ENSG00000119041.11	-0.0010770
ENSG00000123562.18	-0.0010770
ENSG00000274561.1	-0.0010766
ENSG00000161179.14	-0.0010766
ENSG00000143321.19	-0.0010765
ENSG00000166140.17	-0.0010762
ENSG00000166170.10	-0.0010761
ENSG00000139579.13	-0.0010761
ENSG00000179051.14	-0.0010760
ENSG00000198763.3	0.0010758
ENSG00000104885.18	-0.0010757
ENSG00000196365.12	-0.0010757
ENSG00000101928.13	-0.0010756
ENSG00000132646.11	-0.0010756
ENSG00000116353.16	-0.0010755
ENSG00000197930.13	-0.0010754
ENSG00000165792.18	-0.0010753
ENSG00000179195.16	-0.0010753
ENSG00000164346.10	-0.0010750
ENSG00000087152.15	-0.0010750
ENSG00000106785.15	-0.0010747
ENSG00000015532.10	-0.0010747
ENSG00000135316.19	-0.0010747
ENSG00000111802.14	-0.0010746
ENSG00000133422.14	-0.0010744

Gene	s0
ENSG00000154001.14	-0.0010744
ENSG00000100083.19	-0.0010744
ENSG00000166848.7	-0.0010742
ENSG00000105186.16	-0.0010742
ENSG00000162076.13	-0.0010741
ENSG00000197122.12	-0.0010741
ENSG00000133704.10	-0.0010737
ENSG00000168936.11	-0.0010736
ENSG00000205302.7	-0.0010736
ENSG00000169599.13	-0.0010735
ENSG00000102103.17	-0.0010735
ENSG00000213762.12	-0.0010733
ENSG00000169976.7	-0.0010732
ENSG00000173545.5	-0.0010731
ENSG00000104897.10	-0.0010731
ENSG00000198858.10	-0.0010731
ENSG00000168214.20	-0.0010730
ENSG00000100567.13	-0.0010730
ENSG00000134291.12	-0.0010730
ENSG00000214113.11	-0.0010729
ENSG00000057593.14	0.0010729
ENSG00000258890.7	-0.0010727
ENSG00000106462.11	-0.0010727
ENSG00000090097.22	-0.0010724
ENSG00000103197.18	-0.0010723
ENSG00000097007.19	-0.0010723
ENSG00000115661.14	-0.0010722
ENSG00000182544.9	-0.0010720
ENSG00000073792.16	-0.0010718
ENSG00000163481.8	-0.0010717
ENSG00000129353.15	-0.0010715
ENSG00000122958.15	-0.0010714
ENSG00000123737.13	-0.0010712
ENSG00000143294.15	-0.0010712
ENSG00000187735.14	-0.0010710
ENSG00000115216.14	-0.0010710
ENSG00000198843.13	-0.0010709
ENSG00000137504.14	-0.0010709
ENSG00000168763.16	-0.0010709
ENSG00000117906.14	-0.0010706
ENSG00000164548.11	-0.0010702

Gene	s0
ENSG00000132589.16	-0.0010698
ENSG00000226328.7	-0.0010698
ENSG00000002919.15	-0.0010697
ENSG00000067066.17	-0.0010696
ENSG00000133313.15	-0.0010694
ENSG00000141127.15	-0.0010692
ENSG00000133243.10	-0.0010690
ENSG00000182944.18	-0.0010690
ENSG00000160606.11	-0.0010689
ENSG00000108469.15	-0.0010688
ENSG00000133872.14	-0.0010688
ENSG00000124333.16	-0.0010687
ENSG00000141759.15	-0.0010686
ENSG00000175826.12	-0.0010686
ENSG00000139687.16	-0.0010686
ENSG00000242372.9	-0.0010685
ENSG00000130787.14	-0.0010685
ENSG00000146701.12	-0.0010684
ENSG00000126267.11	-0.0010680
ENSG00000188976.11	-0.0010679
ENSG00000173465.8	-0.0010679
ENSG00000143486.16	-0.0010678
ENSG00000145817.17	-0.0010677
ENSG00000086475.15	-0.0010677
ENSG00000108861.9	-0.0010675
ENSG00000176155.19	-0.0010675
ENSG00000068323.17	-0.0010673
ENSG00000198727.2	0.0010673
ENSG00000162910.19	-0.0010672
ENSG00000130669.17	-0.0010666
ENSG00000203705.11	-0.0010665
ENSG00000101191.17	-0.0010664
ENSG00000106609.16	-0.0010664
ENSG00000141378.15	-0.0010663
ENSG00000168876.9	-0.0010660
ENSG00000089159.16	-0.0010660
ENSG00000084110.11	0.0010658
ENSG00000242802.9	-0.0010658
ENSG00000125734.15	-0.0010658
ENSG00000146433.9	-0.0010658
ENSG00000196998.19	-0.0010657

Gene	s0
ENSG00000055070.17	-0.0010656
ENSG00000128563.13	-0.0010656
ENSG00000131462.8	-0.0010655
ENSG00000145740.19	-0.0010654
ENSG00000158985.14	-0.0010654
ENSG00000168286.3	-0.0010651
ENSG00000130175.10	-0.0010651
ENSG00000131043.12	-0.0010650
ENSG00000126945.9	-0.0010648
ENSG00000057935.14	-0.0010648
ENSG00000126581.13	-0.0010645
ENSG00000152127.9	-0.0010644
ENSG00000169045.17	-0.0010643
ENSG00000173812.11	-0.0010642
ENSG00000112936.19	0.0010641
ENSG00000198742.10	-0.0010641
ENSG00000197785.14	-0.0010638
ENSG00000038358.15	-0.0010637
ENSG00000159423.17	0.0010637
ENSG00000105254.12	-0.0010637
ENSG00000065135.12	-0.0010636
ENSG00000118680.14	-0.0010634
ENSG00000103326.12	-0.0010633
ENSG00000044090.10	-0.0010633
ENSG00000145494.12	-0.0010631
ENSG00000183207.14	-0.0010631
ENSG00000175600.16	0.0010627
ENSG00000111229.16	-0.0010627
ENSG00000171497.5	0.0010626
ENSG00000167110.18	-0.0010626
ENSG00000025293.17	-0.0010625
ENSG00000160285.15	0.0010625
ENSG00000136247.15	-0.0010624
ENSG00000135486.19	-0.0010622
ENSG00000160993.4	-0.0010622
ENSG00000167635.12	-0.0010622
ENSG00000113812.14	-0.0010621
ENSG00000175029.17	-0.0010619
ENSG00000095485.18	-0.0010619
ENSG00000038382.20	-0.0010618
ENSG00000169230.10	-0.0010617

Gene	s0
ENSG00000100296.14	-0.0010615
ENSG00000118640.11	-0.0010613
ENSG00000135341.18	-0.0010612
ENSG00000105341.19	-0.0010611
ENSG00000227057.10	-0.0010610
ENSG00000280587.1	0.0010609
ENSG00000147548.17	-0.0010607
ENSG00000152944.9	-0.0010604
ENSG00000131373.15	0.0010599
ENSG00000137522.18	-0.0010598
ENSG00000100246.13	-0.0010593
ENSG00000177425.11	-0.0010592
ENSG00000068724.17	-0.0010592
ENSG00000173281.5	0.0010591
ENSG00000225648.5	-0.0010590
ENSG00000160014.17	-0.0010590
ENSG00000187257.16	-0.0010589
ENSG00000124198.9	-0.0010587
ENSG00000079691.18	-0.0010586
ENSG00000103549.22	-0.0010584
ENSG00000100401.20	-0.0010583
ENSG00000239305.7	-0.0010582
ENSG00000163162.9	-0.0010581
ENSG00000151327.13	-0.0010581
ENSG00000108061.12	-0.0010580
ENSG00000173171.14	-0.0010577
ENSG00000167977.9	-0.0010577
ENSG00000091164.13	-0.0010572
ENSG00000163964.17	-0.0010572
ENSG00000146576.13	-0.0010571
ENSG00000077312.9	-0.0010570
ENSG00000111445.14	-0.0010570
ENSG00000197562.10	-0.0010567
ENSG00000213639.10	-0.0010566
ENSG00000082996.20	-0.0010566
ENSG00000133895.17	-0.0010565
ENSG00000047457.14	0.0010565
ENSG00000139842.15	-0.0010562
ENSG00000198786.2	0.0010561
ENSG00000175115.13	-0.0010561
ENSG00000179632.10	-0.0010561

Gene	s0
ENSG00000109171.15	-0.0010560
ENSG00000185986.12	-0.0010559
ENSG00000166741.8	0.0010558
ENSG00000106144.20	-0.0010557
ENSG00000117691.10	-0.0010557
ENSG00000148773.14	-0.0010553
ENSG00000166123.14	0.0010552
ENSG00000108848.16	-0.0010552
ENSG00000116030.17	-0.0010551
ENSG00000138621.12	-0.0010550
ENSG00000165280.17	-0.0010550
ENSG00000105677.12	-0.0010549
ENSG00000143374.17	-0.0010549
ENSG00000213965.3	-0.0010548
ENSG00000110367.13	-0.0010548
ENSG00000198900.6	-0.0010548
ENSG00000113845.10	-0.0010541
ENSG00000119335.17	-0.0010540
ENSG00000137200.13	-0.0010539
ENSG00000118418.14	-0.0010539
ENSG00000257218.6	-0.0010537
ENSG00000120948.18	-0.0010535
ENSG00000106554.13	-0.0010534
ENSG00000179950.14	-0.0010533
ENSG00000125870.11	-0.0010532
ENSG00000135631.17	-0.0010532
ENSG00000068354.16	-0.0010531
ENSG00000237523.2	-0.0010531
ENSG00000186834.4	-0.0010531
ENSG00000136908.18	-0.0010529
ENSG00000174606.14	-0.0010528
ENSG00000164405.11	-0.0010528
ENSG00000218283.2	-0.0010527
ENSG00000116161.18	-0.0010527
ENSG00000143149.12	0.0010527
ENSG00000198231.13	-0.0010526
ENSG00000111832.13	-0.0010526
ENSG00000141570.11	-0.0010526
ENSG00000127870.17	-0.0010523
ENSG00000099246.17	-0.0010522
ENSG00000064961.19	-0.0010522

Gene	s0
ENSG00000169957.10	-0.0010519
ENSG00000064601.20	-0.0010519
ENSG00000035141.8	-0.0010519
ENSG00000100591.8	-0.0010516
ENSG00000160131.13	-0.0010514
ENSG00000177663.14	-0.0010512
ENSG00000137269.15	-0.0010512
ENSG00000114686.9	-0.0010511
ENSG00000096746.17	-0.0010510
ENSG00000163466.16	-0.0010507
ENSG00000100028.12	-0.0010507
ENSG00000188938.17	-0.0010501
ENSG00000196586.16	-0.0010500
ENSG00000167565.13	-0.0010498
ENSG00000181163.14	-0.0010496
ENSG00000126457.22	-0.0010495
ENSG00000154096.14	-0.0010494
ENSG00000137166.16	-0.0010494
ENSG00000079313.15	-0.0010487
ENSG00000117751.18	-0.0010486
ENSG00000104904.12	-0.0010486
ENSG00000048828.17	-0.0010485
ENSG00000166949.16	-0.0010485
ENSG00000197905.10	-0.0010483
ENSG00000204237.5	-0.0010483
ENSG00000164190.19	-0.0010482
ENSG00000175216.15	-0.0010482
ENSG00000079462.8	-0.0010479
ENSG00000068120.15	-0.0010476
ENSG00000196704.13	-0.0010475
ENSG00000018699.13	-0.0010474
ENSG00000137767.14	-0.0010472
ENSG00000166881.10	-0.0010472
ENSG00000121060.18	-0.0010470
ENSG00000218739.10	-0.0010468
ENSG00000174238.15	-0.0010468
ENSG00000127838.14	-0.0010465
ENSG00000164466.13	0.0010465
ENSG00000007168.14	-0.0010464
ENSG00000153827.14	-0.0010460
ENSG00000129158.11	-0.0010459

Gene	s0
ENSG00000178467.18	-0.0010458
ENSG00000139921.13	-0.0010457
ENSG00000109685.19	-0.0010457
ENSG00000124535.16	-0.0010457
ENSG00000105429.13	-0.0010457
ENSG00000185624.15	-0.0010454
ENSG00000135930.14	-0.0010454
ENSG00000142507.10	-0.0010450
ENSG00000100796.17	-0.0010449
ENSG00000152332.16	-0.0010448
ENSG00000136044.12	-0.0010448
ENSG00000073969.18	-0.0010444
ENSG00000108829.10	-0.0010438
ENSG00000080815.19	-0.0010436
ENSG00000163840.10	-0.0010436
ENSG00000113441.16	-0.0010436
ENSG00000118960.13	-0.0010435
ENSG00000116954.8	-0.0010435
ENSG00000089057.15	0.0010433
ENSG00000135624.16	-0.0010432
ENSG00000109118.14	-0.0010431
ENSG00000122359.18	-0.0010431
ENSG00000163001.12	-0.0010428
ENSG00000153944.12	-0.0010427
ENSG00000173226.17	-0.0010427
ENSG00000128581.17	-0.0010426
ENSG00000224870.7	-0.0010425
ENSG00000139131.13	-0.0010425
ENSG00000090054.15	-0.0010425
ENSG00000169180.12	-0.0010421
ENSG00000121644.19	-0.0010419
ENSG00000120314.18	-0.0010414
ENSG00000023171.18	-0.0010409
ENSG00000138735.16	-0.0010409
ENSG00000124171.9	-0.0010408
ENSG00000141696.13	-0.0010408
ENSG00000184207.9	-0.0010407
ENSG00000157778.9	-0.0010406
ENSG00000139651.11	-0.0010406
ENSG00000130762.15	-0.0010406
ENSG00000187531.14	-0.0010405

Gene	s0
ENSG00000233476.3	-0.0010400
ENSG00000239789.6	-0.0010399
ENSG00000067704.10	-0.0010399
ENSG00000106070.20	-0.0010396
ENSG00000163882.9	-0.0010395
ENSG00000091947.10	-0.0010391
ENSG00000112473.18	-0.0010389
ENSG00000116874.12	-0.0010389
ENSG00000121892.15	-0.0010389
ENSG00000117408.11	-0.0010387
ENSG00000197976.12	-0.0010387
ENSG00000167258.15	-0.0010386
ENSG00000080845.18	-0.0010386
ENSG00000185122.11	-0.0010384
ENSG00000124422.12	-0.0010384
ENSG00000164978.18	-0.0010383
ENSG00000188529.15	-0.0010381
ENSG00000071127.17	-0.0010380
ENSG00000138834.14	-0.0010378
ENSG00000157353.17	-0.0010378
ENSG00000171490.13	-0.0010378
ENSG00000007376.8	-0.0010378
ENSG00000148843.15	-0.0010377
ENSG00000108582.12	-0.0010377
ENSG00000241685.10	-0.0010376
ENSG00000104983.8	-0.0010376
ENSG00000058272.19	-0.0010375
ENSG00000121741.16	-0.0010374
ENSG00000181704.12	-0.0010373
ENSG00000188986.9	-0.0010373
ENSG00000018510.17	-0.0010372
ENSG00000037241.7	-0.0010369
ENSG00000166181.13	-0.0010368
ENSG00000105849.6	-0.0010367
ENSG00000169439.12	0.0010364
ENSG00000170776.22	-0.0010361
ENSG00000105289.15	-0.0010360
ENSG00000101844.18	-0.0010360
ENSG00000217128.12	-0.0010359
ENSG00000168061.17	-0.0010359
ENSG00000160633.13	-0.0010354

Gene	s0
ENSG00000179387.10	-0.0010353
ENSG00000123992.20	-0.0010352
ENSG00000198792.13	-0.0010351
ENSG00000134690.11	-0.0010350
ENSG00000221823.11	-0.0010350
ENSG00000113810.16	-0.0010350
ENSG00000007520.4	-0.0010349
ENSG00000163235.16	-0.0010348
ENSG00000157107.14	-0.0010347
ENSG00000129911.9	-0.0010346
ENSG00000124795.17	-0.0010346
ENSG00000115758.13	-0.0010346
ENSG00000173264.15	-0.0010346
ENSG00000107957.16	-0.0010345
ENSG00000137161.17	-0.0010345
ENSG00000204394.13	-0.0010344
ENSG00000132383.12	-0.0010342
ENSG00000236552.2	-0.0010340
ENSG00000155256.17	-0.0010338
ENSG00000095787.23	-0.0010338
ENSG00000140848.17	-0.0010338
ENSG00000141258.13	-0.0010336
ENSG00000115419.13	-0.0010334
ENSG00000100320.23	-0.0010332
ENSG00000163960.12	-0.0010332
ENSG00000020129.16	-0.0010332
ENSG00000141429.13	-0.0010331
ENSG00000211452.11	0.0010330
ENSG00000204256.14	-0.0010330
ENSG00000138614.15	-0.0010328
ENSG00000076108.12	-0.0010328
ENSG00000143977.14	-0.0010327
ENSG00000108100.18	-0.0010325
ENSG00000165983.14	-0.0010324
ENSG00000166971.17	-0.0010322
ENSG00000155097.12	-0.0010322
ENSG00000198815.9	-0.0010321
ENSG00000146833.16	-0.0010321
ENSG00000166508.18	-0.0010317
ENSG00000196396.10	-0.0010316
ENSG00000084733.11	-0.0010313

Gene	s0
ENSG00000129250.12	-0.0010312
ENSG00000182481.10	-0.0010312
ENSG00000058804.12	-0.0010312
ENSG00000140612.14	-0.0010310
ENSG00000075234.17	0.0010308
ENSG00000159199.14	-0.0010308
ENSG00000169100.14	-0.0010308
ENSG00000103335.22	-0.0010307
ENSG00000160714.10	-0.0010305
ENSG00000235173.7	-0.0010305
ENSG00000137815.14	-0.0010302
ENSG00000006453.14	-0.0010302
ENSG00000151929.10	-0.0010301
ENSG00000108826.16	-0.0010300
ENSG00000144655.15	0.0010299
ENSG00000185619.18	-0.0010297
ENSG00000029363.17	-0.0010296
ENSG00000155304.6	-0.0010296
ENSG00000123064.13	-0.0010295
ENSG00000173153.17	-0.0010295
ENSG00000141858.12	-0.0010293
ENSG00000224470.8	-0.0010292
ENSG00000108883.13	-0.0010290
ENSG00000131236.18	-0.0010289
ENSG00000105402.8	-0.0010288
ENSG00000108424.11	-0.0010288
ENSG00000160588.10	-0.0010287
ENSG00000102921.8	-0.0010286
ENSG00000168906.13	-0.0010285
ENSG00000133030.22	-0.0010284
ENSG00000182872.16	-0.0010283
ENSG00000115963.13	0.0010283
ENSG00000104960.15	-0.0010282
ENSG00000122952.17	-0.0010281
ENSG00000215114.10	-0.0010278
ENSG00000163909.8	-0.0010277
ENSG00000221978.12	-0.0010276
ENSG00000168090.10	-0.0010275
ENSG00000184489.12	-0.0010275
ENSG00000167862.10	-0.0010273
ENSG00000106330.12	-0.0010272

Gene	s0
ENSG00000169018.6	-0.0010272
ENSG00000087157.19	-0.0010272
ENSG00000172893.16	0.0010271
ENSG00000141569.12	-0.0010271
ENSG00000196367.14	-0.0010269
ENSG00000126254.12	-0.0010268
ENSG00000276180.1	-0.0010267
ENSG00000163510.14	-0.0010267
ENSG00000204531.20	-0.0010266
ENSG00000116001.17	-0.0010266
ENSG00000203879.12	-0.0010265
ENSG00000124786.12	-0.0010265
ENSG00000173327.8	-0.0010265
ENSG00000180917.18	-0.0010265
ENSG00000277791.5	-0.0010260
ENSG00000006625.18	-0.0010260
ENSG00000120008.16	-0.0010252
ENSG00000132005.9	-0.0010251
ENSG00000169764.16	0.0010250
ENSG00000151240.17	-0.0010249
ENSG00000172830.13	-0.0010249
ENSG00000170035.16	-0.0010248
ENSG00000197858.11	-0.0010247
ENSG00000141503.17	-0.0010246
ENSG00000148308.18	-0.0010242
ENSG00000071859.15	-0.0010242
ENSG00000204574.14	-0.0010238
ENSG00000105767.3	-0.0010236
ENSG00000104979.9	-0.0010236
ENSG00000126464.14	-0.0010235
ENSG00000185009.13	-0.0010234
ENSG00000130309.11	-0.0010231
ENSG00000100418.8	-0.0010223
ENSG00000072958.9	-0.0010219
ENSG00000158773.14	-0.0010219
ENSG00000172354.10	-0.0010215
ENSG00000198586.14	-0.0010215
ENSG00000140983.14	-0.0010215
ENSG00000168938.6	-0.0010214
ENSG00000204628.12	-0.0010213
ENSG00000074842.8	-0.0010206

Gene	s0
ENSG00000108946.15	-0.0010205
ENSG00000132768.14	-0.0010204
ENSG00000121578.13	-0.0010203
ENSG00000095139.14	-0.0010200
ENSG00000168172.9	-0.0010200
ENSG00000108479.12	0.0010200
ENSG00000110497.14	-0.0010197
ENSG00000068383.19	-0.0010196
ENSG00000141738.14	-0.0010194
ENSG00000186716.21	-0.0010194
ENSG00000186591.13	-0.0010191
ENSG00000120889.13	-0.0010188
ENSG00000128833.13	-0.0010186
ENSG00000129932.10	-0.0010183
ENSG00000204253.4	-0.0010181
ENSG00000100281.14	-0.0010180
ENSG00000176182.6	-0.0010177
ENSG00000115904.14	-0.0010177
ENSG00000175906.5	0.0010176
ENSG00000163382.12	-0.0010175
ENSG00000104883.8	0.0010173
ENSG00000235863.4	-0.0010172
ENSG00000075568.17	-0.0010171
ENSG00000166452.12	-0.0010171
ENSG00000143889.16	-0.0010171
ENSG00000140740.11	-0.0010171
ENSG00000178741.12	-0.0010170
ENSG00000120333.4	-0.0010170
ENSG00000109111.15	-0.0010169
ENSG00000100297.16	-0.0010169
ENSG00000105514.8	-0.0010166
ENSG00000145741.16	-0.0010165
ENSG00000157227.13	-0.0010165
ENSG00000119414.12	-0.0010165
ENSG00000103245.14	-0.0010164
ENSG00000140521.17	-0.0010162
ENSG00000138413.14	0.0010161
ENSG00000102393.14	-0.0010160
ENSG00000151893.15	-0.0010160
ENSG00000103174.13	-0.0010160
ENSG00000138772.13	-0.0010157

Gene	s0
ENSG00000106608.17	-0.0010157
ENSG00000105443.15	-0.0010155
ENSG00000162517.13	-0.0010151
ENSG00000143799.14	-0.0010151
ENSG00000090487.11	-0.0010150
ENSG00000119655.11	-0.0010149
ENSG00000239306.5	-0.0010148
ENSG00000132323.9	-0.0010148
ENSG00000196757.8	-0.0010147
ENSG00000182827.9	-0.0010147
ENSG00000105053.11	-0.0010144
ENSG00000127952.17	-0.0010143
ENSG00000124120.11	0.0010142
ENSG00000041802.11	-0.0010142
ENSG00000181026.15	-0.0010142
ENSG00000127191.18	-0.0010141
ENSG00000133597.11	-0.0010141
ENSG00000135124.15	-0.0010138
ENSG00000135720.13	-0.0010137
ENSG00000111711.10	-0.0010137
ENSG00000184916.9	-0.0010137
ENSG00000115738.10	0.0010131
ENSG00000176853.16	-0.0010131
ENSG00000126012.12	-0.0010131
ENSG00000188342.12	-0.0010131
ENSG00000149196.16	-0.0010129
ENSG00000108395.14	-0.0010125
ENSG00000171552.14	-0.0010123
ENSG00000148925.11	-0.0010123
ENSG00000134717.18	-0.0010122
ENSG00000112983.18	-0.0010122
ENSG00000182700.5	-0.0010121
ENSG00000111737.12	-0.0010118
ENSG00000160813.7	-0.0010118
ENSG00000204304.12	-0.0010118
ENSG00000165775.18	-0.0010117
ENSG00000136003.16	-0.0010116
ENSG00000176903.5	-0.0010115
ENSG00000151376.17	-0.0010115
ENSG00000103932.12	-0.0010114
ENSG00000090266.13	-0.0010114

Gene	s0
ENSG00000075415.13	-0.0010112
ENSG00000067829.19	-0.0010110
ENSG00000133134.12	-0.0010108
ENSG00000143256.5	-0.0010106
ENSG00000123595.8	-0.0010105
ENSG00000113368.12	-0.0010105
ENSG00000143390.18	-0.0010103
ENSG00000171425.10	-0.0010103
ENSG00000135018.14	-0.0010102
ENSG00000143476.18	-0.0010102
ENSG00000120756.13	-0.0010102
ENSG00000212978.6	-0.0010100
ENSG00000116560.11	-0.0010100
ENSG00000115756.13	-0.0010099
ENSG00000123353.10	-0.0010098
ENSG00000115457.10	0.0010097
ENSG00000082258.13	-0.0010096
ENSG00000154845.16	-0.0010095
ENSG00000111669.15	-0.0010093
ENSG00000159596.7	-0.0010093
ENSG00000280128.1	-0.0010092
ENSG00000130165.10	-0.0010092
ENSG00000104976.12	-0.0010091
ENSG00000029364.12	-0.0010091
ENSG00000185414.20	-0.0010090
ENSG00000133275.16	-0.0010089
ENSG00000004961.15	-0.0010087
ENSG00000153207.16	-0.0010086
ENSG00000010803.16	-0.0010085
ENSG00000130479.11	-0.0010080
ENSG00000130600.19	0.0010079
ENSG00000147459.18	-0.0010079
ENSG00000205903.7	-0.0010077
ENSG00000189343.7	-0.0010075
ENSG00000114978.18	-0.0010074
ENSG00000171992.13	-0.0010074
ENSG00000163638.13	-0.0010072
ENSG00000135698.10	-0.0010071
ENSG00000152492.15	-0.0010071
ENSG00000100225.18	-0.0010071
ENSG00000061676.15	-0.0010070

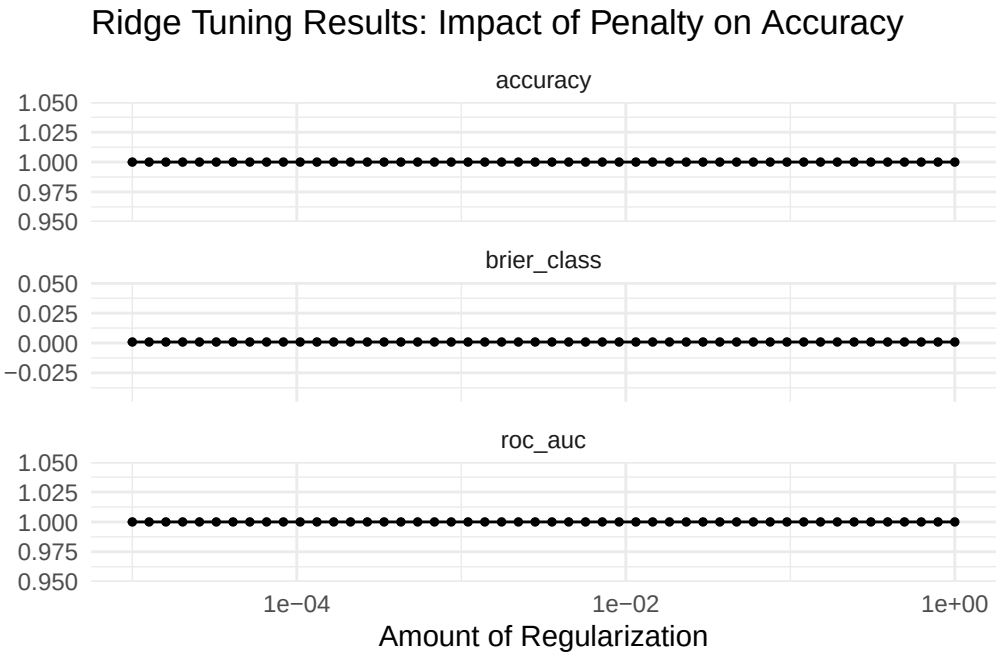
Gene	s0
ENSG00000130147.16	-0.0010070
ENSG00000130811.12	-0.0010069
ENSG00000196151.11	-0.0010067
ENSG00000213719.8	-0.0010064
ENSG00000040633.13	-0.0010061
ENSG00000104731.14	-0.0010061
ENSG00000115652.15	-0.0010061
ENSG00000122642.11	-0.0010056
ENSG00000286451.1	-0.0010054
ENSG00000214253.9	-0.0010053
ENSG00000134851.13	-0.0010053
ENSG00000204580.13	-0.0010052
ENSG00000205336.14	-0.0010052
ENSG00000133773.12	-0.0010052
ENSG00000188641.14	0.0010052
ENSG00000111845.5	-0.0010050
ENSG00000100201.23	-0.0010050
ENSG00000102897.10	0.0010048
ENSG00000182670.13	-0.0010048
ENSG00000061273.18	-0.0010046
ENSG00000166619.15	-0.0010046
ENSG00000103496.15	-0.0010045
ENSG00000100416.15	-0.0010045
ENSG00000090470.15	-0.0010039
ENSG00000100364.19	-0.0010039
ENSG00000167515.10	-0.0010038
ENSG00000109046.15	-0.0010038
ENSG00000197713.15	-0.0010037
ENSG00000116586.12	-0.0010037
ENSG00000163517.15	-0.0010036
ENSG00000244005.13	-0.0010036
ENSG00000066135.13	-0.0010035
ENSG00000066654.14	-0.0010035
ENSG00000163624.6	-0.0010035
ENSG00000127947.16	-0.0010033
ENSG00000185189.18	-0.0010031
ENSG00000136021.19	-0.0010030
ENSG00000108312.15	-0.0010030
ENSG00000261221.4	-0.0010029
ENSG00000153046.18	-0.0010029
ENSG00000125835.19	-0.0010029

Gene	s0
ENSG00000130227.17	-0.0010027
ENSG00000135365.16	-0.0010026
ENSG00000111640.15	-0.0010026
ENSG00000182087.14	-0.0010026
ENSG00000119411.11	-0.0010024
ENSG00000204536.14	-0.0010022
ENSG00000125991.20	-0.0010022
ENSG00000162066.16	-0.0010021
ENSG00000197771.13	-0.0010021
ENSG00000234608.8	-0.0010020
ENSG00000242247.11	-0.0010018
ENSG00000054965.11	-0.0010018
ENSG00000177981.11	-0.0010016
ENSG00000116459.11	-0.0010014
ENSG00000163875.16	-0.0010012
ENSG00000166228.9	0.0010011
ENSG00000184887.13	-0.0010009
ENSG00000140743.8	-0.0010008
ENSG00000171530.15	-0.0010006
ENSG00000102606.18	-0.0010005
ENSG00000205593.12	-0.0010005
ENSG00000119596.18	-0.0010005
ENSG00000178096.9	-0.0010003
ENSG00000100344.11	0.0010002
ENSG00000100162.15	-0.0010002
ENSG00000102144.15	-0.0010001
ENSG00000143549.21	-0.0010000

4.8.3.3 Running a Ridge Regression Model in R: `{tidymodels}`

```
library(tidymodels)
set.seed(123)

# 1. Define Ridge Specification
ridge_spec <- logistic_reg(
  penalty = tune(), # We will find this via CV
  mixture = 0       # 0 = Ridge
) %>%
  set_engine("glmnet") %>%
```

```r
  set_mode("classification")

# 2. Define Cross-Validation Folds (10-fold)
folds <- vfold_cv(train_processed, v = 10)

# 3. Create a Grid of Lambda values to test
lambda_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 50)

# 4. Run the Tuning
ridge_grid <- tune_grid(
  workflow() %>% add_recipe(full_recipe) %>% add_model(ridge_spec),
  resamples = folds,
  grid = lambda_grid
)

# 5. Visualize the Tuning Results
autoplot(ridge_grid) +
  theme_minimal() +
  labs(title = "Ridge Tuning Results: Impact of Penalty on Accuracy")
```



4.8.3.4 Model Evaluation

As we have seem, the model evaluation is the same as for all the previous models.

```
# Prepare the test data for prediction
test_matrix <- as.matrix(test_processed %>%
                           select (-sample_type)) %>% # Exclude the label
  as.matrix()
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert
# Make predictions on the test dataset
predictions <- stats::predict(ridge_model,
                              newx = test_matrix,
                              type = "response")
predicted_labels <- ifelse(predictions > 0.5, 1, 0)
predicted_labels <- as.factor(predicted_labels)
# Evaluate the accuracy of the model

# Evaluate the accuracy of the model
confusionMatrix(predicted_labels, as.factor(test_labels))
```

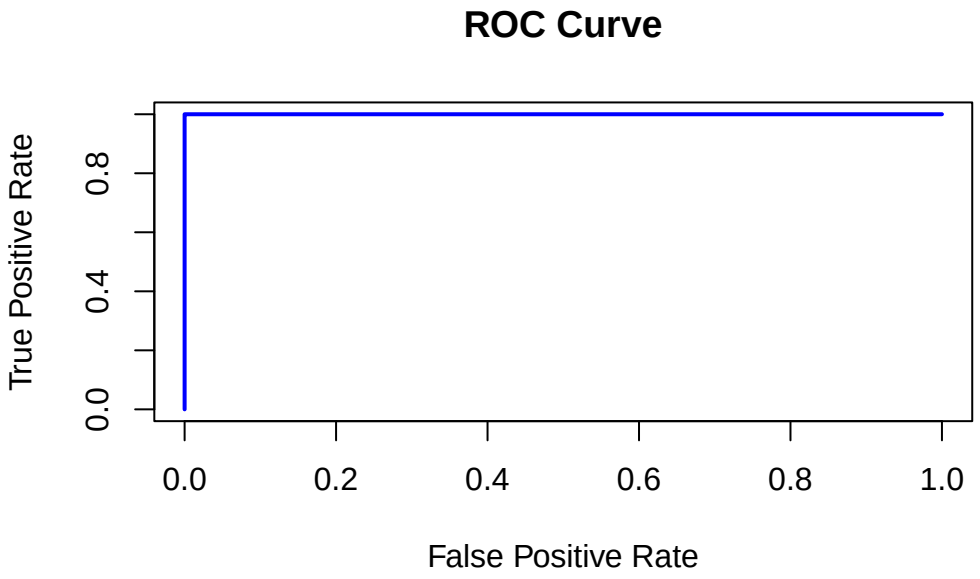
```
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 7 0
         1 0 2

               Accuracy : 1
                 95% CI : (0.6637, 1)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.1042

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.7778
         Detection Rate : 0.7778
```

```
   Detection Prevalence : 0.7778
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

```r
#######
## AUC and AUC-PR
# Get predicted probabilities for the positive class
pred_prob <- predict(lasso_model,
                     newx = test_matrix,
                     type = "response")
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)

# Calculate AUC
auc <- ROCR::performance(pred, measure = "auc")@y.values[[1]]
auc
```

```
[1] 1
```

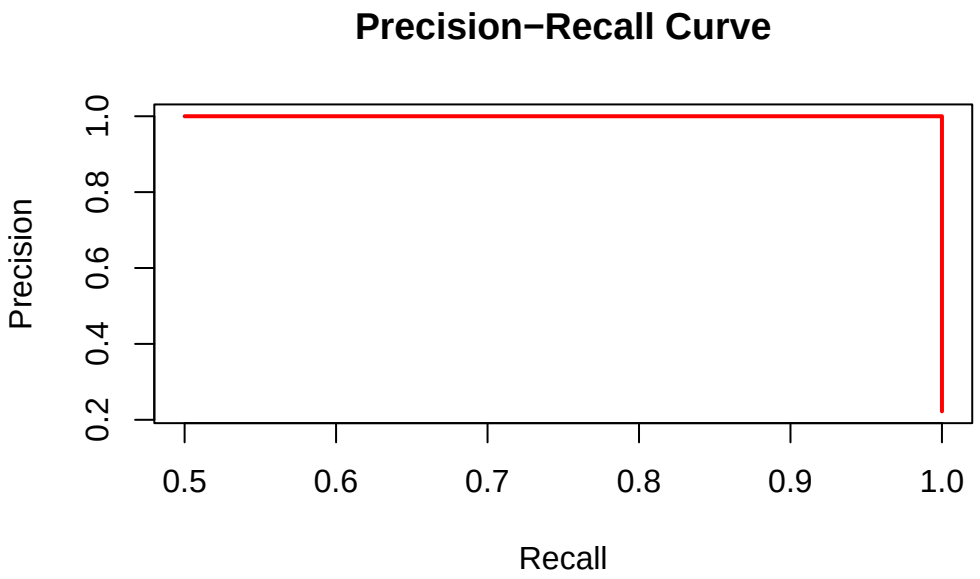
```r
# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")
```




```
# Calculate AUC-PR
auc_pr <- ROCR::performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr
```

```
[1] 1
```

```
# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")
```



4.9 Conclusion

In this section, we have explored how to implement decision trees, random forests, gradient boosting, regression models and also delved into regularization techniques, specifically Lasso (L1) and Ridge (L2) regression, which are essential for handling high-dimensional data like gene expression profiles. Understanding the differences between these regularization methods is crucial for selecting the appropriate model based on the biological context of your data.

For all model implementations, we have also discussed how to evaluate their performance using metrics such as accuracy, AUC, and AUC-PR, which are critical for assessing the predictive power of our models in a clinical setting.

5 Evaluating our model

When building a machine learning model, it is crucial to evaluate its performance to understand how well it generalizes to unseen data. In this section, we will discuss various evaluation metrics and techniques that can be used to assess the performance of our model.

Recall that our goal when building a machine learning model is to make good predictions for new and unseen data. To evaluate how well our model performs on unseen data, we can use a variety of metrics depending on the type of problem we are trying to solve (e.g., classification, regression).

As we have seen in previous chapters, we typically split our dataset into a training set and a test set. The training set is used to train the model, while the **test set is used to evaluate its performance**. It is important to ensure that the test set is representative of the data the model will encounter in real-world applications, and that it has not been used during the training process to avoid overfitting.

5.1 Evaluation Metrics for Classification

For classification problems, we often want to match the predicted class labels to the true class labels, and evaluate how well our model does this. We start by building a so-called confusion matrix (Figure 5.1), which is a table that summarizes the performance of a classification model by showing the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

From the confusion matrix, we can derive several important metrics:

- **True Positive (TP)**: The number of instances where the model correctly predicted the positive class.
- **True Negative (TN)**: The number of instances where the model correctly predicted the negative class.
- **False Positive (FP)**: The number of instances where the model incorrectly predicted the positive class (Type I error).
- **False Negative (FN)**: The number of instances where the model incorrectly predicted the negative class (Type II error).

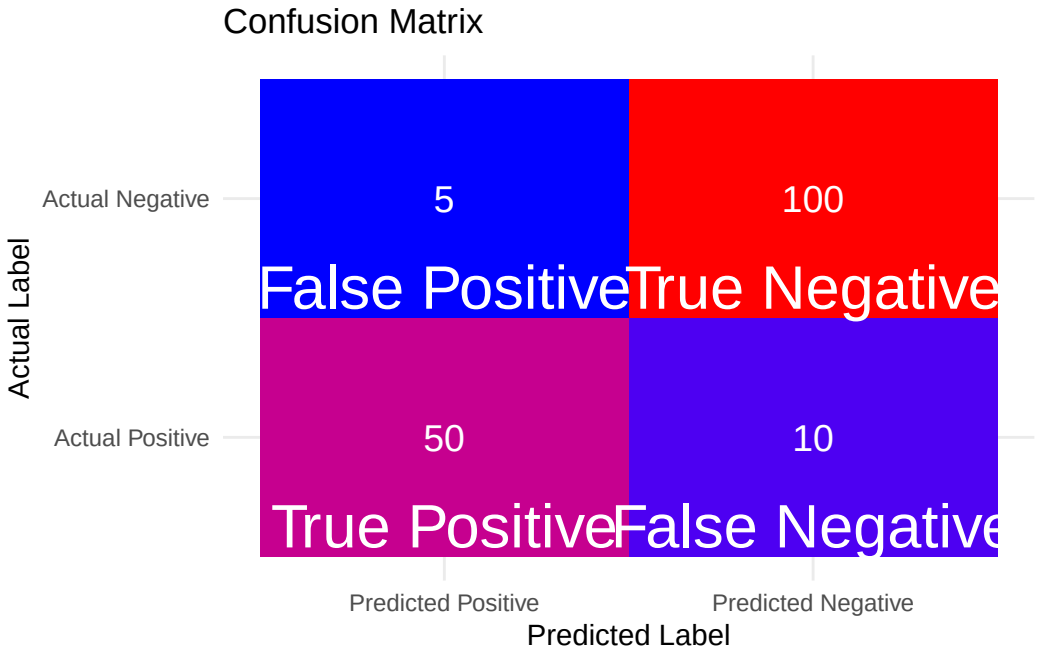


Figure 5.1: Example Confusion Matrix Visualization.

- **Accuracy**: The proportion of correct predictions (both true positives and true negatives) out of the total number of predictions. This metric is sensitive to class imbalance, as it can be misleading when one class is much more frequent than the other.

$$A = (TP + TN)/(TP + TN + FP + FN).$$

- **Balanced Accuracy**: The average of the true positive rate (sensitivity) and true negative rate (specificity), providing a more balanced view of performance in the presence of class imbalance.

$$BA = \frac{(TPR + TNR)}{2}.$$

- **Precision (Positive Predictive Value)**: The proportion of true positives out of all predicted positives.

$$P = TP/(TP + FP).$$

- **Recall (Sensitivity, True Positive Rate)**: The proportion of true positives out of all actual positives.

$$R = TP/(TP + FN).$$

- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two metrics.

$$F1 = 2 * (P * R)/(P + R).$$

- **Matthews Correlation Coefficient (MCC)**: A measure of the quality of binary classifications, taking into account true and false positives and negatives. It ranges from -1 to +1, where +1 indicates perfect prediction, 0 indicates random prediction, and -1 indicates total disagreement between prediction and observation.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}.$$

A complete diagnostic classification matrix with all these metrics and their definitions is found in Figure 5.3.

Another important metric for evaluating classification models is the **Receiver Operating Characteristic (ROC) curve** and the associated **Area Under the Curve (AUC)**:

- **Area Under the Receiver Operating Characteristic Curve (AUC-ROC)**: A measure of the model's ability to distinguish between classes (Figure 5.2a). The ROC curve plots the **true positive rate** against the **false positive rate** at various threshold settings, and the AUC represents the area under this curve. The AUC ranges from 0 to 1, with higher values indicating better performance. The AUC is highly affected by class imbalance, and it can be misleading in cases where the positive class is rare. In such cases, other metrics like Precision-Recall AUC may be more informative.
- **Precision-Recall AUC**: This metric is particularly useful when dealing with imbalanced datasets, as it focuses on the performance of the positive class. The Precision-Recall curve plots precision against recall at various threshold settings (Figure 5.2b), and the area under this curve provides a single scalar value to summarize the model's performance in terms of precision and recall. The Precision-Recall AUC ranges from 0 to 1, with higher values indicating better performance.

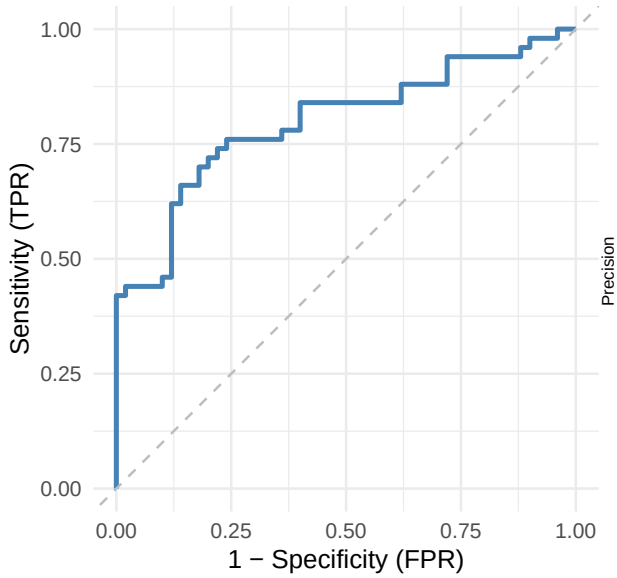
5.2 Evaluation Metrics for Regression

For regression problems, we typically evaluate the performance of our model using metrics that measure the **difference between the predicted values and the actual values**. Some common regression evaluation metrics include: - **Mean Absolute Error (MAE)**: The average of the absolute differences between the predicted values and the actual values.

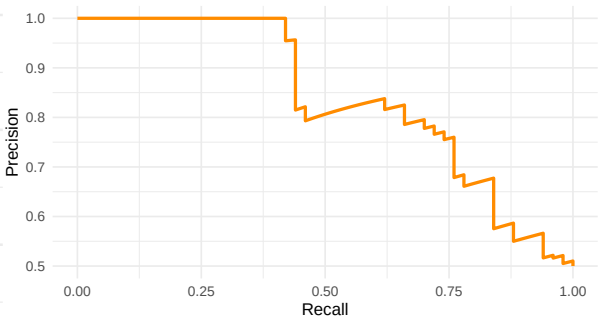
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|.$$

- **Mean Squared Error (MSE)**: The average of the squared differences between the predicted values and the actual values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$



(a) ROC Curve (Sensitivity vs FPR)



(b) Precision-Recall Curve

Figure 5.2: Classification Performance Curves

- **Root Mean Squared Error (RMSE)**: The square root of the mean squared error, providing a measure of the average magnitude of the errors in the same units as the target variable.

$$RMSE = \sqrt{MSE}.$$

- **R-squared (Coefficient of Determination)**: A measure of how well the regression model fits the data, representing the proportion of the variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

- **Adjusted R-squared**: A modified version of R-squared that adjusts for the number of predictors in the model, providing a more accurate measure of model fit when multiple predictors are used.

$$R_a^2 = 1 - \left(\frac{SS_{res}/(n-p-1)}{SS_{tot}/(n-1)}\right) = 1 - \left(\frac{(1-R^2)(n-1)}{n-p-1}\right).$$

When evaluating regression models, it is important to consider the context of the problem and the specific requirements of the application. For example, in some cases, we may prioritize minimizing large errors (in which case RMSE may be more appropriate), while in other cases, we may want to focus on the average error (in which case MAE may be more suitable).

Additionally, R-squared and Adjusted R-squared can provide insights into how well the model explains the variance in the data, but they should be interpreted with caution, especially when comparing models with different numbers of predictors.

5.3 Comprehensive Confusion Matrix and Metrics

Complete Diagnostic Classification Matrix

Recreated with all metrics, definitions, and error types

	Predicted Condition		Actual Condition Totals & Rates	
Actual Condition	Predicted Positive	Predicted Negative	Primary Rate	Secondary Rate
Real Positive (P) **[a] The number of real positive cases**	**True Positive (TP)** [b] Hit: Correct positive indication	**False Negative (FN)** [c] Miss, Type II error: Wrongly indicates absence	**True Positive Rate (TPR)** Recall, Sensitivity, Hit Rate $\frac{TP}{P} = 1 - FNR$	**False Negative Rate (FNR)** Miss rate, Type II error $\frac{FN}{P} = 1 - TPR$
Real Negative (N) **[d] The number of real negative cases**	**False Positive (FP)** [f] False alarm, Type I error: Wrongly indicates presence	**True Negative (TN)** [e] Correct rejection: Correct absence indication	**False Positive Rate (FPR)** Fall-out, Type I error $\frac{FP}{N} = 1 - TNR$	**True Negative Rate (TNR)** Specificity, Selectivity $\frac{TN}{N} = 1 - FPR$
Total Population **= P + N**	**Positive Predictive Value (PPV)** Precision = $\frac{TP}{TP+FP} = 1 - FDR$	**False Omission Rate (FOR)** $\frac{FN}{TN+FN} = 1 - NPV$	**Prevalence** $\frac{P}{P+N}$	**Accuracy (ACC)** $\frac{TP+TN}{P+N}$

Metric Legend & Compound Formulas:

- **[a] P:** Real Positives | **[b] TP:** Hits | **[c] FN:** Type II Error (Miss) | **[d] N:** Real Negatives | **[e] TN:** Correct Rejection | **[f] FP:** Type I Error (False Alarm)
- **F1 Score:** $\frac{2PPV \times TPR}{PPV+TPR} = \frac{2TP}{2TP+FP+FN}$
- **MCC:** $\sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$
- **Informedness (BM):** $TPR + TNR - 1$ | **Markedness (MK):** $PPV + NPV - 1$
- **Diagnostic Odds Ratio (DOR):** $\frac{LR+}{LR-} = \frac{TP \times TN}{FP \times FN}$
- **Threat Score (TS/CSI/Jaccard):** $\frac{TP}{TP+FN+FP}$
- **Prevalence Threshold (PT):** $\frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$

Figure 5.3: Complete Confusion Matrix and Evaluation Metrics

Part III

Source

6 Run the function

```
#' ML Crash Course: Dependency Installer

install_ml_course_deps <- function() {

  # 1. CRAN Packages
  cran_packages <- c(
    "tidyverse",    # Data manipulation and ggplot2
    "tidymodels",   # The ML framework (parsnip, recipes, rsample, etc.)
    "embed",        # Extra recipes for UMAP and target encoding
    "caret",        # For confusionMatrix and older ML utility
    "randomForest", # Random Forest engine
    "ranger",       # Fast Random Forest engine for tidymodels
    "xgboost",      # Gradient Boosting engine
    "glmnet",       # Lasso/Ridge engine
    "rpart",        # Decision tree base
    "rpart.plot",   # Visualization of trees
    "vip",          # Variable Importance Plots
    "ROCR",         # Performance metrics and ROC curves
    "dbscan",       # Density-based clustering
    "Rtsne",        # t-SNE dimensionality reduction
    "knitr"         # For neat table formatting
    "usethis",      # For R configuration, we use for git
    "gitcreds"      # For GitHub token management
  )

  # 2. Bioconductor Packages (Specific for Transcriptomics)
  bioc_packages <- c(
    "TCGAbiolinks",        # TCGA Data Access
    "SummarizedExperiment" # Data structure for biological assays
  )

  # Helper function to install missing CRAN packages
  new_cran <- cran_packages[!(cran_packages %in% installed.packages()[,"Package"])]
  if(length(new_cran)) {
```

```
    message("Installing missing CRAN packages: ", paste(new_cran, collapse = ", "))
    install.packages(new_cran, dependencies = TRUE)
  }

  # Helper function to install Bioconductor manager and packages
  if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
  }

  new_bioc <- bioc_packages[!(bioc_packages %in% installed.packages()[,"Package"])]
  if(length(new_bioc)) {
    message("Installing missing Bioconductor packages: ", paste(new_bioc, collapse = ", "))
    BiocManager::install(new_bioc, update = FALSE, ask = FALSE)
  }

  message("--- All dependencies checked and installed! ---")
}

install_ml_course_deps()

## Configure Git Account
library(usethis)
use_git_config(user.name = "User Name",
               user.email = "user.email@email.com")

## Create the token
usethis::create_github_token()

## Include your creds
gitcreds::gitcreds_set()
```

```
library(tidymodels)

# 1. Split the data
data_split <- initial_split(mtcars, prop = 0.8)
train_data <- training(data_split)

# 2. Define the Recipe (The Prep)
# "Predict mpg using all other variables, then scale them"
simple_recipe <- recipe(mpg ~ ., data = train_data) %>%
  step_normalize(all_predictors())
```

```r
# 3. Define the Model (The Engine)
# "I want a linear regression model using the 'lm' engine"
lm_spec <- linear_reg() %>%
  set_engine("lm")

# 4. Create the Workflow (The Glue)
simple_workflow <- workflow() %>%
  add_recipe(simple_recipe) %>%
  add_model(lm_spec)

# 5. Fit the whole thing
final_model <- fit(simple_workflow,
                   data = train_data)


############
# Prepare our Data for Machine Learning with
# TCGA Cholangiocarcinoma (CHOL) Cohort
library(TCGAbiolinks)
library(SummarizedExperiment)
library(tidyverse)

# 1. Query only the Cholangiocarcinoma cohort
query_chol <- GDCquery(
  project = "TCGA-CHOL",
  data.category = "Transcriptome Profiling",
  data.type = "Gene Expression Quantification",
  workflow.type = "STAR - Counts"
)
#  Load the data into a SummarizedExperiment object
chol_se <- GDCprepare(query_chol)


#######

# 2. Download and prepare the data
# Extract fpkm using
fpkm_data <- assay(chol_se, "fpkm_unstrand")

# 3. Quick Metadata Link
metadata <- as.data.frame(colData(chol_se)) %>%
  select(barcode, sample_type)
```

```r
head(metadata)

#######

# 4. ML Preparation (Transpose)
# Often in ML, TPM or FPKM values are used directly
# We need the samples as rows and genes as columns
# We filter for low-expression genes to remove noise
# For fpkm, a common threshold is > 5 in a certain % of samples.
keep <- rowSums(fpkm_data > 5) >= ncol(fpkm_data) * 0.2 # Keep genes expressed in at least 2(
fpkm_filtered <- fpkm_data[keep, ]
dim(fpkm_filtered) # Check dimensions after filtering
## Combine with metadata
df_ml_fpkm <- as.data.frame(t(fpkm_filtered)) %>%
  rownames_to_column("barcode") %>%
  inner_join(metadata, by = "barcode") %>%
  select(-barcode) %>%
  mutate(sample_type = factor(sample_type))
dim(df_ml_fpkm) # Check final dimensions

#######
# Split the data into training and testing sets
library(tidymodels)
set.seed(12345) # For reproducibility
data_split <- initial_split(df_ml_fpkm,
                            prop = 0.8)
train_data_chol <- training(data_split)
test_data_chol  <- testing(data_split)

cat("Training samples:", nrow(train_data_chol), "\nTesting samples:", nrow(test_data_chol))
cat("Proportion of sample types in training set:\n")
print(prop.table(table(train_data_chol$sample_type)))

cat("Proportion of sample types in testing set:\n")
print(prop.table(table(test_data_chol$sample_type)))

#######
# Split the data into training and testing sets
library(tidymodels)
set.seed(42) # For reproducibility
data_split <- initial_split(df_ml_fpkm,
                            prop = 0.8,
```

```
                              strata = sample_type)
train_data_chol <- training(data_split)
test_data_chol  <- testing(data_split)

cat("Training samples:", nrow(train_data_chol), "\nTesting samples:", nrow(test_data_chol))
cat("Proportion of sample types in training set:\n")
print(prop.table(table(train_data_chol$sample_type)))

cat("Proportion of sample types in testing set:\n")
print(prop.table(table(test_data_chol$sample_type)))
### Just to check the mean and sd of
### the expression values for each gene
### in the training set
train_data_chol %>%
  pivot_longer(cols = -sample_type,
               names_to = "Gene",
               values_to = "Expression") %>%
  group_by(Gene, sample_type) %>%
  summarize(mean_expression = mean(Expression),
            sd = sd(Expression))


#######
library(tidymodels)
# Define a recipe for pre-processing
# We will scale, center, and remove near-zero variance predictors

# Define a recipe for pre-processing
# Focus specifically on numeric predictors to avoid scaling the outcome factor
ml_recipe <- recipe(sample_type ~ ., data = train_data_chol) %>%
  step_zv(all_predictors()) %>%            # Remove zero variance (constant) genes
  step_nzv(all_numeric_predictors()) %>%   # Remove genes with very little variation
  step_normalize(all_numeric_predictors()) # Z-score transformation (mean=0, sd=1)

# Statistical Note for Students:
# We prep() using ONLY train_data_chol to ensure the mean and SD
# are not influenced by our test set (avoiding "Data Leakage").
ml_prep <- prep(ml_recipe, training = train_data_chol)

# Apply (bake) to both sets
train_processed <- bake(ml_prep, new_data = NULL) # NULL defaults to the training data
test_processed  <- bake(ml_prep, new_data = test_data_chol)
```

```r
# Check dimensions: How many "noisy" genes did we drop?
cat("Original Training Dim:", dim(train_data_chol), "\n")
cat("Processed Training Dim:", dim(train_processed), "\n")

# How about the test set?
cat("Original Testing Dim:", dim(test_data_chol), "\n")
cat("Processed Testing Dim:", dim(test_processed), "\n")

# Create stratified 5-fold cross-validation
set.seed(123)
cv_folds <- vfold_cv(train_data_chol,
                     v = 5)

cv_folds


# Create stratified 5-fold cross-validation
set.seed(12345)
cv_folds <- vfold_cv(train_data_chol,
                     v = 5,
                     strata = sample_type)
cv_folds

###### Before start, let us do a quick t-test just to check

# Perform t-tests for each gene
t_test_results <- train_processed %>%
  pivot_longer(cols = -sample_type,
               names_to = "Gene",
               values_to = "Expression") %>%
  group_by(Gene) %>%
  summarize(
    p_value = t.test(Expression ~ sample_type)$p.value,
    mean_tumor = mean(Expression[sample_type == "Primary Tumor"]),
    mean_normal = mean(Expression[sample_type == "Solid Tissue Normal"]),
    diff = mean_tumor - mean_normal
  ) %>%
  arrange(p_value)

# Visualize using Esquisse :)

ggplot(t_test_results) +
```

```
  aes(x = diff, y = p_value) +
  geom_point(colour = "#112446") +
  theme_minimal()


####### Sec 3: Unsupervised

library(embed)
# Define the PCA Recipe
pca_rec <- recipe(sample_type ~ .,
                  data = train_processed) %>%
  step_pca(all_predictors(), num_comp = 5)

# Train the recipe on the CHOL training set
pca_estimates <- prep(pca_rec)

# Extract the coordinates for visualization
pca_plot_data <- juice(pca_estimates)

# Visualization of PC1 vs PC2
pca_plot_data %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6", "Solid Tissue Normal" = "#d7191
  theme_minimal() +
  labs(title = "Principal Component Analysis: TCGA-CHOL",
       x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
       y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
       color = "Sample Type") +
  theme(legend.position = "bottom")

library(Rtsne)
# Prepare data for t-SNE
# Assuming 'train_data_chol' is a data frame with the sample type in the first column and ge
# We will use the gene expression data for t-SNE
tsne_data <- train_processed %>%
  select(-sample_type) %>%
  as.matrix()
# Run t-SNE
# Set a random seed for reproducibility
set.seed(123)
```

```r
tsne_result <- Rtsne(tsne_data,
                     perplexity = 1, # Play with the perplexity parameter to see how it affe
                     verbose = TRUE,
                     max_iter = 1000)

# Create a data frame for plotting
tsne_plot_data <- data.frame(
  X = tsne_result$Y[, 1],
  Y = tsne_result$Y[, 2],
  sample_type = train_data_chol$sample_type
)

# Visualization of t-SNE results
# Using ggplot2 for visualization

tsne_plot_data %>%
  ggplot() +
  aes(x = X, y = Y, color = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6",
                                "Solid Tissue Normal" = "#d7191c")) +
  theme_minimal() +
  labs(title = "t-SNE: TCGA-CHOL",
       x = "t-SNE Dimension 1",
       y = "t-SNE Dimension 2",
       color = "Sample Type") +
  theme(legend.position = "bottom")

library(tidymodels)
library(embed)

# Define UMAP recipe
umap_rec <- recipe(sample_type ~ ., data = train_processed) %>%
  step_umap(
    all_predictors(),
    num_comp = 2, # Plotting in 2D, but try 3D as well
    neighbors = 5, # Try different values (5, 10, 15) to see how it affects the visualizatio
    min_dist = 0.1 # Try different values (0.1, 0.5, 0.9) to see how it affects the clusteri
  )

# Train recipe
umap_estimates <- prep(umap_rec)
```

```r
# Extract coordinates
umap_plot_data <- juice(umap_estimates)

# Visualization
umap_plot_data %>%
  ggplot() +
  aes(x = UMAP1, y = UMAP2, color = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Primary Tumor" = "#2c7bb6",
                                "Solid Tissue Normal" = "#d7191c")) +
  theme_minimal() +
  labs(title = "UMAP Projection: TCGA-CHOL",
       x = "UMAP 1",
       y = "UMAP 2",
       color = "Sample Type") +
  theme(legend.position = "bottom")

# Compute distance matrix
# A common choice for gene expression data is the Euclidean distance
distance_matrix <- dist(
  train_processed %>%
    select(-sample_type),
  method = "minkowski") # different distance metrics can be tried (euclidean, manhattan, mink
## Euclidean is used for continuous data, while Manhattan can be more robust to outliers.
## Minkowski is a generalization of both

# Perform hierarchical clustering
# We can use the complete linkage method, which considers the maximum distance between points
hc <- hclust(distance_matrix,
             method = "complete")
# Plot the dendrogram

plot(hc,
     labels = train_processed$sample_type,
     main = "Hierarchical Clustering Dendrogram",
     xlab = "",
     sub = "")


# Determine the optimal number of clusters using the Elbow Method
# We will compute the total within-cluster sum of squares for a range of K values
wss <- sapply(1:10, function(k) {
```

```
  kmeans(train_processed %>%
           select(-sample_type),
         centers = k,
         nstart = 25)$tot.withinss
})
# Plot the Elbow Method
plot(1:10, wss,
     type = "b",
     las = 1,
     pch = 19,
     xlab = "Number of clusters K",
     ylab = "Total within-clusters sum of squares")



# Perform K-means clustering with K=2
set.seed(123)
kmeans_result <- kmeans(train_processed %>%
                          select(-sample_type),
                        centers = 2,
                        nstart = 25)

cluster <- as.factor(kmeans_result$cluster)
# Visualize the clusters
# We can use PCA to visualize the clusters in a 2D space

pca_plot_data %>%
  mutate(cluster = cluster) %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = cluster, shape = sample_type) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("1" = "#2c7bb6",
                                "2" = "#d7191c")) +
  theme_minimal() +
  labs(title = "K-Means Clustering (K=2) on PCA Projection",
       x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
       y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
       color = "Cluster") +
  theme(legend.position = "bottom")
```

```r
library(dbscan)
# Prepare data for DBSCAN
# We will use the PCA-reduced data for DBSCAN to reduce computational complexity
dbscan_data <- pca_plot_data %>%
  select(PC1, PC2) %>%
  as.matrix()
# Run DBSCAN
# Set eps to a value that captures the local density of points and minPts to a value that re
dbscan_result <- dbscan(dbscan_data,
                        eps = 20,
                        minPts = 5,
                        borderPoints = T
)
dbscan_result


# Add cluster assignments to the original data
pca_plot_data$cluster <- as.factor(dbscan_result$cluster)
# Visualize the DBSCAN clusters
pca_plot_data %>%
  ggplot() +
  aes(x = PC1, y = PC2, color = cluster) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("0" = "#2c7bb6", # Cluster 0 (noise)
                                "1" = "#d7191c", # Cluster 1
                                "2" = "#fdae61", # Cluster 2
                                "3" = "#abdda4", # Cluster 3
                                "4" = "#2b83ba")) + # Cluster 4
  theme_minimal() +
  labs(title = "DBSCAN Clustering on PCA Projection",
       x = paste0("PC1 (", round(pca_estimates$steps[[1]]$res$sdev[1]^2 / sum(pca_estimates$s
       y = paste0("PC2 (", round(pca_estimates$steps[[1]]$res$sdev[2]^2 / sum(pca_estimates$s
       color = "Cluster") +
  theme(legend.position = "bottom")



####### Supervised
library(rpart)
library(rpart.plot)

### Create a mini Train, so it does not take a long time to run
```

```r
main_genes <- t_test_results %>%
  mutate(p_adj = p.adjust(p_value, method = "BH")) %>%
  filter(p_adj > 0.8) %>%
  head(50) %>%
  pull(Gene)

train_processed_mini <- train_processed %>%
  select(sample_type, main_genes)
# 1. Prepare the Data.
# We have already done it ;)

# 2. Factorize
train_processed_mini$sample_type <- as.factor(train_processed_mini$sample_type)
train_processed_mini$sample_type <- as.factor(train_processed_mini$sample_type)

# 3. Fit Model with Complexity Control
# cp (complexity parameter) helps prevent the tree from getting too "wild"
tree_model <- rpart(sample_type ~ .,
                    data = train_processed_mini,
                    method = "class",
                    control = rpart.control(cp = 0.001,
                                            minsplit = 2))

# 4. Plot
# 'type = 5' shows the split labels clearly; 'extra = 104' shows probabilities and percentage
rpart.plot(tree_model,
           type = 5,
           extra = 104,
           box.palette = "RdYlGn",
           shadow.col = "gray",
           main = "Decision Tree (CHOL)")



#######

# 1. Define the Recipe (Pre-processing)
# We select our outcome and predictors, then filter for the top most variable genes
tree_recipe <- recipe(sample_type ~ .,
                      data = train_processed_mini)
full_recipe <- recipe(sample_type ~ .,
                                    data = train_processed )
```

```r
# 2. Define the Model Specification
# Here we specify 'rpart' as the engine for a classification task
tree_spec <- decision_tree(cost_complexity = 0.01,
                           tree_depth = 5,
                           min_n = 2
                           ) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# 3. Create a Workflow
# This bundles the recipe and the model together
tree_workflow <- workflow() %>%
  add_recipe(tree_recipe) %>%
  add_model(tree_spec)

# 4. Fit the model
tree_fit <- tree_workflow %>%
  fit(data = train_processed)

# 5. Extract the fitted model to plot the tree
extract_fit_engine(tree_fit) %>%
  rpart.plot(type = 5,
             extra = 104,
             box.palette = "RdYlGn",
             main = "Decision Tree: CHOL Dataset")

# Make predictions on the test dataset
test_processed$sample_type <- as.factor(test_processed$sample_type)
predictions <- predict(tree_model, newdata = test_processed, type = "class")
# Evaluate the accuracy of the model
# Create a confusion matrix
require(caret)
confusionMatrix(predictions, test_processed$sample_type)



#######

library(randomForest)
set.seed(12345) # For reproducibility
# Fit a random forest model
#
```

```
rf_model <- randomForest(sample_type ~ .,
                         mtry = 10, # Number of variables randomly sampled as candidates at e
                         data = train_processed,
                         maxnodes = 5,
                         nodesize = 2,
                         ntree = 100) # Number of trees to grow

# Print the random forest model
rf_model

#######

# Plot the random forest model
plot(rf_model)

#######

# Get feature importance
importance(rf_model) %>%
  as.data.frame() %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(10) %>%
  knitr::kable(caption = "Top 10 Most Important Features in the Random Forest Model")

#######

library(ggplot2)

imp_df <- importance(rf_model) %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(20)

ggplot(imp_df) +
  aes(x = reorder(Gene, MeanDecreaseGini),
      y = MeanDecreaseGini) +
  geom_point(size = 3,
             color = "steelblue") +
  geom_segment(aes(x = Gene,
                   xend = Gene,
                   y = 0,
```

```
                    yend = MeanDecreaseGini),
                color = "skyblue") +
  coord_flip() +
  labs(title = "Variable Importance (Random Forest)",
       x = "Genes",
       y = "Mean Decrease in Gini Index") +
  theme_minimal()

#######

# Look at the structure of the 1st tree
# k = 1 is the tree number
# labelVar = TRUE ensures it uses gene names instead of index numbers
randomForest::getTree(rf_model,
                      k = 1,
                      labelVar = TRUE) %>%
  head(10) # Showing the first 10 nodes

#######
library(tidymodels)
set.seed(123) # For reproducibility
## The data step is the same as previously, we need only to update our model specification as

# 1. Define the Random Forest Specification
# We'll grow 100 trees (trees = 100)
# mtry is the number of genes randomly sampled at each split
rf_spec <- rand_forest(
  mtry = tune(),      # We can tune this later
  trees = 100,
  min_n = 2
) %>%
  set_engine("ranger",
             importance = "impurity") %>%
  set_mode("classification")

# 2. Update the Workflow
# We reuse the 'tree_recipe' from the previous section
rf_workflow <- workflow() %>%
  add_recipe(full_recipe) %>% # Reusing the recipe
  add_model(rf_spec %>% finalize_model(list(mtry = 2))) # Setting a starting mtry

# 3. Fit the Model
```

```r
rf_fit <- rf_workflow %>%
  fit(data = train_processed)

# 4. Extract the fitted model to get feature importance
rf_fit %>%
  extract_fit_parsnip() %>%
  vip::vip(num_features = 20,
           geom = "point") +
  theme_minimal() +
  labs(title = "Random Forest: Top 20 Gene Biomarkers",
       x = "Importance (Mean Decrease in Gini)",
       y = "Genes")

#######
# Make predictions on the test dataset
# We will use our test data here
predictions <- predict(rf_model, newdata = test_processed)
# Evaluate the accuracy of the model
# Create a confusion matrix
require(caret)
confusionMatrix(predictions, test_processed$sample_type)

#######
# Make predictions on the test dataset
# We will use our test data here
predictions <- predict(rf_model, newdata = test_processed)
# Evaluate the accuracy of the model
# Create a confusion matrix
require(caret)
confusionMatrix(predictions, test_processed$sample_type)

#######
## AUC and AUC-PR
require(ROCR)
# Get predicted probabilities for the positive class
pred_prob <- predict(rf_model,
                     newdata = test_processed,
                     type = "prob")[, 1] # Tumour Prob is the First Column
sample_type <- ifelse(test_processed$sample_type == "Primary Tumor", 1, 0) # Convert to bina

plot(pred_prob, sample_type,
     xlab = "Predicted Probability of Tumor",
```

```
     ylab = "Actual Sample Type (1=Tumor, 0=Normal)",
     main = "Predicted Probabilities vs Actual Labels")
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)
# Calculate AUC
auc <- ROCR::performance(pred, measure = "auc")@y.values[[1]]
auc


#######
# Calculate AUC-PR
auc_pr <- performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr

#######

# plot the Precision-Recall curve

pr_perf <- performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")

#######

library(xgboost)
set.seed(12345) # For reproducibility
# Prepare the data for xgboost
# We need to convert the data into a matrix format and the labels into a numeric format
train_matrix <- train_processed_mini %>%
  select(-sample_type) %>% # Exclude the label
  as.matrix()
train_labels <- as.numeric(train_processed_mini$sample_type) - 1 # Convert to numeric (0 and
# Fit a gradient boosting model
gb_model <- xgboost(data = train_matrix,
                    label = as.factor(train_labels),
                    nrounds = 100, # Number of boosting rounds
                    objective = "binary:logistic") # Binary classification
# Print the gradient boosting model
```

```r
print(gb_model)

#######

# Get feature importance
importance_matrix <- xgb.importance(feature_names = colnames(train_matrix), model = gb_model)
importance_matrix %>%
  head(10) %>%
  knitr::kable(caption = "Top 10 Most Important Features in the Gradient Boosting Model")

#######
library(tidymodels)
library(xgboost)
library(vip)

set.seed(123)

# 1. Define the Gradient Boosting Specification
# Note: XGBoost usually needs many more trees than a Random Forest
# because it learns slowly (boosting vs bagging).
gb_spec <- boost_tree(
  trees = 100,
  tree_depth = 3,
  learn_rate = 0.1
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# 2. Update the Workflow (reusing your tree_recipe)
gb_workflow <- workflow() %>%
  add_recipe(full_recipe) %>%
  add_model(gb_spec)

# 3. Fit the Model
gb_fit <- gb_workflow %>%
  fit(data = train_processed)

# 4. Extract and Plot Importance
# In tidymodels, you simply pass the parsnip object directly to vip().
# It will automatically interface with xgboost and retrieve the feature names.
gb_fit %>%
  extract_fit_parsnip() %>%
```

```
  vip(num_features = 20,
      geom = "point",
      aesthetics = list(color = "midnightblue", size = 3)) +
  theme_minimal() +
  labs(title = "Gradient Boosting: Top 20 Gene Biomarkers",
       subtitle = "Importance calculated via Gain",
       x = "Importance",
       y = "Genes")

#######
# Prepare the test data for xgboost
test_matrix <- as.matrix(test_processed %>% select (-sample_type)) # Exclude the label
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert to numeric (0 and 1)
# Make predictions on the test dataset
predictions <- predict(gb_model, newdata = test_matrix)
# Convert predicted probabilities to class labels (0 or 1)
predicted_labels <- ifelse(predictions > 0.5, 1, 0 ) %>%
  as.factor()

# Evaluate the accuracy of the model
caret::confusionMatrix(predicted_labels, as.factor(test_labels))

#######
## AUC and AUC-PR
## Get predicted probabilities for the positive class
pred_prob <- predict(gb_model,
                     newdata = test_matrix) # Predicted probabilities for the positive class
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)
# Calculate AUC
auc <- ROCR::performance(pred,
                         measure = "auc")@y.values[[1]]
auc

#######
# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")
```

```
#######
# Calculate AUC-PR
auc_pr <- ROCR::performance(pred,
                            measure = "aucpr")@y.values[[1]]
auc_pr

# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred,
                             measure = "prec",
                             x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")


#######

# Fit a linear regression model

linear_model <- lm(ENSG00000001629.10 ~ sample_type, data = train_processed)
# Print the linear regression model
summary(linear_model)


### Ok, but I don't want to do one gene at a time, or put into a for loop.
require(broom)
fit = train_processed %>%
  pivot_longer(cols = -sample_type,
               names_to = "Gene",
               values_to = "Expression") %>%
  group_by(Gene) %>%
  do(tidy(lm(Expression ~ sample_type, data = .))) %>%
  filter(term != "(Intercept)") %>%
  mutate(p_adj = p.adjust(p.value, method = "BH"))

fit %>%
  ggplot() +
  aes(x = estimate, y = -log10(p_adj)) +
  geom_point(color = "steelblue") +
```

```
  theme_minimal() +
  labs(title = "Linear Regression: Effect of Sample Type on Gene Expression",
       x = "Estimated Effect (Coefficient)",
       y = "log10 Adjusted P-value (BH)") +
  theme(legend.position = "bottom")


#######
library(tidymodels)
# 1. Define the Recipe

regression_recipe <- recipe(ENSG00000001629.10 ~ sample_type, data = train_processed)

regression_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# 3. Create a Workflow
regression_workflow <- workflow() %>%
  add_recipe(regression_recipe) %>%
  add_model(regression_spec)

# 4. Fit the model
regression_fit <- regression_workflow %>%
  fit(data = train_processed)

# 5. Print the model summary
extract_fit_parsnip(regression_fit)
#######

## We start by making predictions on the test dataset
predictions <- predict(linear_model,
                       newdata = test_processed)

# Calculate Mean Squared Error (MSE)
mse <- mean((predictions - test_processed$ENSG00000001629.10)^2)
mse

# Calculate Mean Absolute Error (MAE)
mae <- mean(abs(predictions - test_processed$ENSG00000001629.10))
mae
```

```r
# Calculate R-squared
ss_total <- sum((test_processed$ENSG00000001629.10 - mean(test_processed$ENSG00000001629.10)
ss_residual <- sum((test_processed$ENSG00000001629.10 - predictions)^2)
r_squared <- 1 - (ss_residual / ss_total)
r_squared


#######
# Fit a logistic regression model


logistic_model <- glm(sample_type ~ ENSG00000001629.10,
                      data = train_processed,
                      family = binomial)
# Print the logistic regression model
summary(logistic_model)

## Stepwise selection
## # To reduce the amount of genes we test, we will use the genes that were found in the dec

keep_imp = t_test_results %>%
  mutate(p_adj = p.adjust(p_value, method = "BH")) %>%
  filter(p_adj < 0.00001) %>%
  head(50) %>%
  pull(Gene)
train_processed_cl = train_processed %>%
  select(sample_type, all_of(keep_imp))
null_model <- glm(sample_type ~ 1,
                  data = train_processed_cl,
                  family = binomial) # Model with only the intercept
full_model <- glm(sample_type ~ .,
                  data = train_processed_cl,
                  family = binomial) # Model with all features
stepwise_model <- stats::step(null_model,
                              scope = list(lower = null_model,
                                           upper = full_model),
                              direction = "both",
                              trace = 0)
summary(stepwise_model)

#######
```

```
library(tidymodels)
# 1. Define the Recipe (Pre-processing)
log_recipe <- recipe(sample_type ~ .,
                     data = train_processed_cl %>%
                       select(sample_type, 2:5))

# 2. Define the Model Specification
# Here we specify 'glm' as the engine for a classification task
logistic_spec <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")
# 3. Create a Workflow
# adding the stepwise selection is a bit tricky in tidymodels, as it does not have a built-in

logistic_workflow <- workflow() %>%
  add_recipe(log_recipe) %>% # Reusing the recipe with Variance Filtering
  add_model(logistic_spec)

# 4. Fit the model

logistic_fit <- logistic_workflow %>%
  fit(data = train_processed_cl)

#######
# Make predictions on the test dataset
predictions <- predict(logistic_model,
                       newdata = test_processed,
                       type = "response")


predicted_labels <- ifelse(predictions < 0.5, "Primary Tumor", "Solid Tissue Normal") %>%
  as.factor()

# Evaluate the accuracy of the model
caret::confusionMatrix(predicted_labels, test_processed$sample_type)

#######

library(glmnet)
set.seed(123) # For reproducibility
# Prepare the data for glmnet
# Data must be in matrix format and labels must be numeric
```

```r
train_matrix <- train_processed %>%
  select(-sample_type) %>%
  as.matrix()
train_labels <- as.numeric(train_processed$sample_type) - 1 # Convert

# Fit a Lasso regression model
# Before fitting a LASSO we have to find the best lambda (regularization parameter) using cr
cv_lasso <- cv.glmnet(train_matrix,
                      train_labels,
                      alpha = 1, # Lasso
                      family = "binomial")
plot(cv_lasso)

best_lambda <- cv_lasso$lambda.min
best_lambda


lasso_model <- glmnet(train_matrix,
                      train_labels,
                      alpha = 1,
                      lambda = best_lambda,
                      family = "binomial")
# Print the Lasso regression model
# The coefficients of the model can be extracted using the coef function
coef(lasso_model) %>%
  as.matrix() %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
  filter(s0 != 0) %>%
  arrange(desc(abs(s0))) %>%
  filter(Gene != "(Intercept)") %>%
  knitr::kable(caption = "Most Important Features in the Lasso Regression Model")


#######

library(tidymodels)

# 1. Define Lasso Specification
lasso_spec <- logistic_reg(
  penalty = tune(), # We will find this via CV
  mixture = 1       # 1 = Lasso
```

```
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# 2. Define Cross-Validation Folds (10-fold)
set.seed(123)
folds <- vfold_cv(train_processed, v = 10)

# 3. Create a Grid of Lambda values to test
lambda_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 10)

# 4. Run the Tuning
lasso_grid <- tune_grid(
  workflow() %>%
    add_recipe(full_recipe) %>%
    add_model(lasso_spec),
  resamples = folds,
  grid = lambda_grid
)

# 5. Visualize the Tuning Results
# This replaces the plot(cv_lasso) call
autoplot(lasso_grid) +
  theme_minimal() +
  labs(title = "Lasso Tuning Results: Impact of Penalty on Accuracy")


#######

# 6. Select the best penalty (highest ROC AUC)
best_penalty <- lasso_grid %>%
  select_best(metric = "roc_auc")

# 7. Finalize and Fit
final_lasso <- workflow() %>%
  add_recipe(full_recipe) %>%
  add_model(lasso_spec) %>%
  finalize_workflow(best_penalty) %>%
  fit(data = train_processed)

# 8. Extract Non-Zero Coefficients (Biomarkers)
final_lasso %>%
```

```
  extract_fit_parsnip() %>%
  tidy() %>%
  filter(estimate != 0 & term != "(Intercept)") %>%
  arrange(desc(abs(estimate))) %>%
  knitr::kable(caption = "Biomarkers Selected by Lasso Regression")

#######

# Prepare the test data for prediction
# We need to ensure that the test data has the same features as the training data, and that t

test_matrix <- test_processed %>%
  select(-sample_type) %>% # Exclude the label
  as.matrix()
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert
# Make predictions on the test dataset

predictions <- predict(lasso_model,
                       newx = test_matrix,
                       type = "response")
predicted_labels <- ifelse(predictions > 0.5, 1, 0
) %>%
  as.factor()
# Evaluate the accuracy of the model
confusionMatrix(predicted_labels, as.factor(test_labels))

## AUC and AUC-PR
# Get predicted probabilities for the positive class
pred_prob <- predict(lasso_model, newx = test_matrix, type = "response")
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)

# Calculate AUC
auc <- ROCR::performance(pred,
                         measure = "auc")@y.values[[1]]
auc

# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
```

```
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")

# Calculate AUC-PR
auc_pr <- ROCR::performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr

# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
     main = "Precision-Recall Curve",
     xlab = "Recall",
     ylab = "Precision")


#######

library(glmnet)
set.seed(123) # For reproducibility
# We use the same data as prepared for the Lasso regression, but we will set alpha = 0 for Ri
cv_ridge <- cv.glmnet(train_matrix,
                      train_labels,
                      alpha = 0, # alpha = 0 for Ridge
                      family = "binomial")

plot(cv_ridge)
best_lambda_ridge <- cv_ridge$lambda.min
best_lambda_ridge

ridge_model <- glmnet(train_matrix,
                      train_labels,
                      alpha = 0, # alpha = 0 for Ridge
                      lambda = best_lambda_ridge,
                      family = "binomial")

# Print the Ridge regression model
coef(ridge_model) %>%
  as.matrix() %>%
  as.data.frame() %>%
  tibble::rownames_to_column("Gene") %>%
```

```
  arrange(desc(abs(s0))) %>%
  filter(abs(s0) > 0.001) %>% # Filter for coefficients with a magnitude greater than a certa
  filter(Gene != "(Intercept)") %>%
  knitr::kable(caption = "Most Important Features in the Ridge Regression Model")


#######
library(tidymodels)
set.seed(123)

# 1. Define Ridge Specification
ridge_spec <- logistic_reg(
  penalty = tune(), # We will find this via CV
  mixture = 0       # 0 = Ridge
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# 2. Define Cross-Validation Folds (10-fold)
folds <- vfold_cv(train_processed, v = 10)

# 3. Create a Grid of Lambda values to test
lambda_grid <- grid_regular(penalty(range = c(-5, 0)), levels = 50)

# 4. Run the Tuning
ridge_grid <- tune_grid(
  workflow() %>% add_recipe(full_recipe) %>% add_model(ridge_spec),
  resamples = folds,
  grid = lambda_grid
)

# 5. Visualize the Tuning Results
autoplot(ridge_grid) +
  theme_minimal() +
  labs(title = "Ridge Tuning Results: Impact of Penalty on Accuracy")

#######
# Prepare the test data for prediction
test_matrix <- as.matrix(test_processed %>%
                           select (-sample_type)) %>% # Exclude the label
  as.matrix()
test_labels <- as.numeric(test_processed$sample_type) - 1 # Convert
```

```r
# Make predictions on the test dataset
predictions <- stats::predict(ridge_model,
                              newx = test_matrix,
                              type = "response")
predicted_labels <- ifelse(predictions > 0.5, 1, 0)
predicted_labels <- as.factor(predicted_labels)
# Evaluate the accuracy of the model

# Evaluate the accuracy of the model
confusionMatrix(predicted_labels, as.factor(test_labels))

#######
## AUC and AUC-PR
# Get predicted probabilities for the positive class
pred_prob <- predict(lasso_model,
                     newx = test_matrix,
                     type = "response")
sample_type <- test_labels # Binary labels (1 for Tumor, 0 for Normal)
# Create a prediction object
pred <- ROCR::prediction(pred_prob, sample_type)

# Calculate AUC
auc <- ROCR::performance(pred, measure = "auc")@y.values[[1]]
auc

# plot the ROC curve
roc_perf <- ROCR::performance(pred, measure = "tpr", x.measure = "fpr")
plot(roc_perf, col = "blue",
     lwd = 2, main = "ROC Curve",
     xlab = "False Positive Rate",
     ylab = "True Positive Rate")


# Calculate AUC-PR
auc_pr <- ROCR::performance(pred, measure = "aucpr")@y.values[[1]]
auc_pr

# plot the Precision-Recall curve
pr_perf <- ROCR::performance(pred, measure = "prec", x.measure = "rec")
plot(pr_perf,
     col = "red",
     lwd = 2,
```

```
    main = "Precision-Recall Curve",
    xlab = "Recall",
    ylab = "Precision")
```

6.1 Books

- https://statisticalmachinelearning.com/
- https://www.statlearning.com/
- https://bradleyboehmke.github.io/HOML/

References