Lecture 3

Classification and Prediction, overview
Review of Logistic Regression

Classification and prediction are essential data mining techniques used to make sense of data, discover patterns, and provide insights for various applications. Both techniques involve the analysis of historical data to build models that can be used to classify or predict future or unknown data points. Here's an overview of how classification and prediction are used in data mining:

Classification:
Classification is the process of categorizing data into predefined classes or categories based on the input attributes. It's a supervised learning technique, meaning it requires labeled training data, where each data point is associated with a known class or category.

Steps in Classification:

- *Data Collection*: Gather historical data that includes both the input attributes (features) and the corresponding class labels.
- *Data Preprocessing*: Clean and preprocess the data, handling missing values, outliers, and other data quality issues.
- *Feature Selection*: Identify and select the most relevant features for the classification task, which can help improve model accuracy and reduce computational complexity.
- *Training Data Split*: Divide the dataset into training and testing sets. The training set is used to build the classification model, while the testing set is used to evaluate the model's performance.
- *Model Building*: Select a classification algorithm (e.g., decision trees, support vector machines, or neural networks) and train the model using the training data.
- *Model Evaluation*: Assess the model's performance using the testing data. Common evaluation metrics include accuracy, precision, recall, F1 score, and the receiver operating characteristic (ROC) curve.
- *Model Deployment*: Once the model performs well on the testing data, deploy it to make predictions on new, unseen data.

Applications of Classification:

Email Spam Detection: Classify emails as either spam or not spam.
Medical Diagnosis: Categorize patient data into disease or non-disease groups.
Sentiment Analysis: Determine the sentiment of text data (e.g., positive, negative, or neutral).
Credit Scoring: Classify applicants as high-risk or low-risk for credit approval.

Prediction:

Prediction, also known as **regression**, is the process of estimating a continuous numerical value or outcome based on input attributes. It is used when the target variable is numeric and is not categorized into classes. Like classification, prediction is a supervised learning technique that requires labeled training data.

Steps in Prediction:

- *Data Collection*: Gather historical data with both the input attributes and the corresponding numerical target values.
- *Data Preprocessing*: Clean and preprocess the data, addressing issues such as missing values, outliers, and data quality.
- *Feature Selection*: Choose the most relevant input features for the prediction task.
- *Training Data Split*: Divide the dataset into training and testing sets for model evaluation.
- *Model Building*: Select a regression algorithm (e.g., linear regression, decision tree regression, or support vector regression) and train the model using the training data.
- *Model Evaluation*: Assess the model's performance using the testing data. Evaluation metrics for prediction tasks include mean squared error (MSE), root mean squared error (RMSE), and R-squared.
- *Model Deployment*: Deploy the model to make numerical predictions on new, unseen data.

Applications of Prediction:

Stock Price Forecasting: Predict future stock prices based on historical data.
Demand Forecasting: Estimate future demand for products or services.
Sales Revenue Prediction: Forecast sales revenue for a business.
Weather Forecasting: Predict temperature, precipitation, and other weather-related variables.

Classification and prediction are powerful tools in data mining and machine learning, enabling organizations to automate decision-making processes, improve accuracy in various domains, and gain valuable insights from data. These techniques are fundamental to building intelligent systems and making data-driven predictions and classifications.

We talked about many methods of regression (prediction) in 325. In this course, we will focus more on classification methods.

There are various **classification algorithms** used in data mining, and the choice of algorithm depends on the specific characteristics of the data and the problem you are trying to solve. Here are some of the most common classification algorithms in data mining:

Logistic Regression is a simple and widely used algorithm for binary classification tasks. It models the relationship between the independent variables and the probability of a binary outcome.

Decision Trees are used to partition the data into subsets based on the values of input features. They are interpretable and can handle both classification and regression tasks. Popular decision tree algorithms include CART and C4.5.

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification accuracy. It is robust and handles high-dimensional data well.

***Support Vector Machines (SVM*)** is a powerful classification algorithm that finds the optimal hyperplane to separate data into classes. It is effective in high-dimensional spaces and can handle non-linear data using kernel functions.

K-Nearest Neighbors (K-NN) is a simple and intuitive algorithm that classifies data points based on the majority class of their nearest neighbors. It's particularly useful for small to moderately sized datasets.

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem. It's often used for text classification, spam detection, and sentiment analysis.

Neural Networks, particularly deep learning models, are powerful for complex classification tasks. Convolutional Neural Networks (CNNs) are used for image classification, and Recurrent Neural Networks (RNNs) are applied to sequence data.

K-Means Clustering: While primarily a clustering algorithm, K-Means can be used for simple classification tasks by assigning new data points to the cluster with the closest centroid.

Gradient Boosting Algorithms like *XGBoost*, *LightGBM*, and *CatBoost* are widely used for classification tasks. They work well on structured data and are known for their high predictive accuracy.

Ensemble Methods combine the predictions of multiple base classifiers to improve accuracy and reduce overfitting. Besides random forests, other ensemble techniques include AdaBoost and Gradient Boosting.

Nearest Centroid Classifier: This simple algorithm classifies data points based on their similarity to the centroids of each class. It's suitable for text classification and image classification.

Gaussian Naive Bayes: A variation of the Naive Bayes algorithm that assumes a Gaussian distribution for continuous features. It's suitable for datasets with continuous attributes.

LDA (Linear Discriminant Analysis) is a dimensionality reduction technique that can also be used for classification tasks. It maximizes the separation between classes by projecting data into a lower-dimensional space.

The choice of classification algorithm depends on factors such as the nature of the data, the size of the dataset, the interpretability of the model, and the specific requirements of the problem at hand. It's common to experiment with multiple algorithms and fine-tune their parameters to achieve the best results for a given classification task.

Let’s review Logistic Regression briefly.

Logistic regression is a statistical method used for binary classification, where the outcome variable is categorical and has two classes (usually coded as 0 and 1). It models the relationship between one or more independent variables and the probability of a particular outcome occurring.

How Logistic Regression Works:
Sigmoid Function: Logistic regression uses the logistic function (sigmoid function) to transform the linear combination of input features into a probability between 0 and 1. The sigmoid function is defined as:

$$P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\ldots+\beta_n X_n)}}$$

where:

- $P(Y=1)$ is the probability of the event Y being class 1.
- e is the base of the natural logarithm.
- $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients to be estimated.
- $X_1, X_2, \ldots, X_n$ are the input features.

This equation is derived from a log-odds model that is a linear function. Solving for the probability results in the equation above.

Training: The logistic regression model is trained by adjusting the coefficients to maximize the likelihood of the observed data given the model.

Decision Boundary: The model predicts class 1 if the probability $P(Y=1)$ is greater than or equal to 0.5 and predicts class 0 otherwise. The decision boundary is determined by the threshold (0.5 by default). It can be adjusted if needed.

Pros:

- *Interpretability*: Logistic regression coefficients represent the log-odds, making it easy to interpret the impact of each feature on the probability of the outcome.
- *Efficiency*: Logistic regression is computationally efficient and requires less computational resources compared to more complex algorithms.
- *Scalability*: It can handle a large number of features without much risk of overfitting.
- *Probabilistic Output*: Logistic regression provides probabilistic outputs, allowing for more nuanced predictions and decision-making.
- *No Assumptions about Feature Distribution*: Logistic regression does not assume that the input features are normally distributed.

Cons:

- *Linear Decision Boundary*: Logistic regression assumes a linear decision boundary, which may not capture complex relationships in the data.
- *Sensitivity to Outliers*: Logistic regression is sensitive to outliers, and their presence can impact the model's performance.
- *Assumption of Independence*: It assumes that the observations are independent of each other, which may not hold in certain situations.
- *Limited to Binary Classification*: Logistic regression is primarily designed for binary classification tasks and may need modifications for multi-class problems.
- *Requires Large Sample Sizes*: It performs better with a large number of samples, and the risk of overfitting increases with a small dataset.
- *No Feature Importance Ranking*: Unlike some tree-based models, logistic regression does not provide a natural feature importance ranking.

In summary, logistic regression is a versatile and interpretable classification algorithm suitable for various applications. However, its simplicity and linearity might be limiting in situations where the underlying

relationships are highly complex or nonlinear. It's crucial to consider the characteristics of the data and the problem at hand when choosing a classification algorithm.

In the next lecture, we'll look at two specific classification algorithms, Support Vector Machines (SVM) and K-Nearest Neighbors (KNN).

Resources:

1. https://www.geeksforgeeks.org/basic-concept-classification-data-mining/
2. https://www.upgrad.com/blog/classification-in-data-mining/
3. https://www.ibm.com/docs/en/db2/10.1.0?topic=algorithms-classification
4. https://stats.oarc.ucla.edu/r/dae/logit-regression/
5. https://www.datacamp.com/tutorial/logistic-regression-R
6. https://www.geeksforgeeks.org/logistic-regression-in-r-programming/
7. https://www.r-bloggers.com/2015/09/how-to-perform-a-logistic-regression-in-r/
8. https://uc-r.github.io/logistic_regression
9. https://towardsdatascience.com/how-to-do-logistic-regression-in-r-456e9cfec7cd
10. https://www.tutorialspoint.com/r/r_logistic_regression.htm
11. https://www.mastersindatascience.org/learning/machine-learning-algorithms/logistic-regression/
12. https://www.statology.org/types-of-logistic-regression/
13. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/
14. https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

Lecture 4

Support Vector Machines
K-Nearest Neighbors

A **Support Vector Machine (SVM)** is a powerful and versatile machine learning algorithm used for both classification and regression tasks. SVMs are known for their ability to handle high-dimensional data and their effectiveness in separating data into different classes. Here's how the SVM algorithm works:

1. *Data Representation*: SVM starts with a labeled dataset, where each data point is represented as a feature vector and assigned to one of two classes: positive or negative (for binary classification).

2. *Objective*: The primary objective of SVM is to find a hyperplane that best separates the data points of different classes with the maximum margin. The margin is the distance between the hyperplane and the nearest data points (support vectors) of each class.

3. *Hyperplane Selection*: The SVM algorithm selects the hyperplane that maximizes the margin. This hyperplane is called the "maximum-margin hyperplane" or the "optimal hyperplane." The hyperplane can be linear, meaning it is a straight line in 2D, a plane in 3D, or a hyperplane in higher dimensions. The dimensionality depends on the number of features in the dataset.

4. *Support Vectors*: Support vectors are data points that are closest to the decision boundary (the margin) and have the smallest margin. These support vectors play a crucial role in defining the optimal hyperplane. The optimal hyperplane is determined by a subset of support vectors, so only a few data points have significant influence on the decision boundary.

5. *Kernel Trick* (Optional): In cases where the data is not linearly separable, SVMs can use the kernel trick to transform the data into a higher-dimensional space, where it becomes linearly separable. Common kernel functions include the linear kernel, polynomial kernel, and radial basis function (RBF) kernel.

6. *Margin Calculation*: The margin is calculated as the perpendicular distance between the optimal hyperplane and the closest support vectors from each class. Maximizing this margin is essential for SVM's robustness and generalization.

7. *Optimization Problem*: The training of an SVM involves solving an optimization problem to find the optimal hyperplane. The optimization problem aims to maximize the margin while ensuring that all data points are correctly classified or have a margin of at least one. This leads to the formulation of a convex quadratic programming problem.

8. *Soft Margin* (Optional): In real-world data, it's often not possible to find a perfect hyperplane due to noise and outliers. SVM can be adapted to handle such situations by introducing a "soft margin" that allows for some misclassification. This is known as the "soft-margin SVM."

9. *Classification or Regression*: Once the optimal hyperplane is determined, it can be used for classification tasks. New data points are classified based on which side of the hyperplane they fall. For regression tasks, SVM can be used to predict numerical values (although, we will not use it for this purpose in this course.)

10. *Regularization* (C Parameter): The regularization parameter "C" is used to control the trade-off between maximizing the margin and minimizing classification errors. Higher values of C prioritize accuracy, while lower values prioritize margin maximization.

SVMs are particularly effective when dealing with high-dimensional data or cases where a clear margin exists between classes. They are used in various applications, including image classification, text classification, and anomaly detection. The kernel trick and the ability to handle nonlinear data make SVMs a versatile choice for many machine learning problems.

Let's walk through a simple example of using a Support Vector Machine (SVM) for a binary classification problem. In this example, we'll classify iris flowers into two classes based on their sepal length and sepal width.

Step 1: Import Necessary Libraries
First, you need to import the required libraries in R. In this example, we'll use the e1071 package for SVM.

```
# Install and load the e1071 package
install.packages("e1071")
library(e1071)
```

Step 2: Load and Prepare the Data

We'll use the built-in Iris dataset from R, which contains measurements of sepal length and width for three species of iris flowers. For this example, we'll focus on classifying the Setosa species.

```
# Load the Iris dataset
data(iris)

# Subset the data to include only Setosa and non-Setosa samples
iris_setosa <- iris[iris$Species == "setosa", ]
iris_non_setosa <- iris[iris$Species != "setosa", ]

# Create a binary classification task (Setosa vs. non-Setosa)
data <- rbind(iris_setosa, iris_non_setosa)
```

Step 3: Split the Data into Training and Testing Sets
Divide the dataset into a training set and a testing set for model evaluation.

```
# Set a random seed for reproducibility
set.seed(123)

# Create an index for splitting the data (70% training, 30% testing)
index <- sample(1:nrow(data), 0.7 * nrow(data))

# Split the data
train_data <- data[index, ]
test_data <- data[-index, ]
```

Step 4: Train an SVM Model
In this step, you'll train an SVM model using the training data. We'll use a linear kernel for simplicity.

```
# Train an SVM model with a linear kernel
svm_model <- svm(Species ~ Sepal.Length + Sepal.Width, data =
train_data, kernel = "linear")
```

Step 5: Make Predictions
Use the trained SVM model to make predictions on the testing data.

```
# Make predictions on the testing data
predictions <- predict(svm_model, newdata = test_data)
```

Step 6: Evaluate the Model
Finally, evaluate the SVM model's performance using appropriate metrics such as accuracy, precision, recall, or an ROC curve. In this example, we'll calculate the accuracy.

```
# Calculate the accuracy
accuracy <- mean(predictions == test_data$Species)
cat("Accuracy:", accuracy, "\n")
```

This is a basic example of using an SVM for binary classification. In practice, you would typically perform more data preprocessing, explore different kernel functions, tune hyperparameters, and use cross-validation for a more comprehensive evaluation of the model's performance. However, this example provides a starting point for applying SVM to a real-world classification problem. See the code examples file for this week for a more detailed procedure.

The **k-Nearest Neighbors (k-NN)** algorithm is a simple and intuitive machine learning algorithm used for both classification and regression tasks. It operates on the principle that data points with similar features

tend to belong to the same class (in the case of classification) or have similar target values (in the case of regression). Here's how the k-NN algorithm works:

1. *Data Representation*: k-NN starts with a labeled dataset, where each data point is represented as a feature vector and assigned to one of multiple classes (for classification) or has a numerical target value (for regression).

2. *Determine the Value of k*: The number "k" is a hyperparameter that you need to specify before applying k-NN. It represents the number of nearest neighbors to consider when making a prediction. The choice of "k" can significantly impact the algorithm's performance. A small "k" may lead to noisy predictions, while a large "k" may result in overly smooth predictions. Even values of k can lead to ties in binary classification, which the algorithm will decide by chance. This can make the model unstable.

3. *Calculate Distances*: For a new data point, calculate the distance (similarity) between that point and all other data points in the dataset. Common distance metrics include Euclidean distance, Manhattan distance, or cosine similarity. The choice of distance metric can be tailored to the nature of the data and the problem. Before calculating the distances, you may need to consider rescaling variables so that one variable does not dominate the others.

4. *Find the Nearest Neighbors*: Select the "k" data points with the shortest distances to the new data point. These are the "k" nearest neighbors. These nearest neighbors will be used to make predictions for the new data point.

5. *For Classification*: In the case of classification, count the occurrences of each class among the "k" nearest neighbors. The new data point is assigned to the class that is most common among the nearest neighbors (a majority vote).

6. *For Regression*: In the case of regression, calculate the average of the target values of the "k" nearest neighbors. The new data point is assigned the average value as its prediction.

7. *Make Predictions*: For each new data point, repeat the above steps to make predictions based on the "k" nearest neighbors.

8. *Evaluate and Tune*: After applying k-NN, evaluate the model's performance using appropriate metrics such as accuracy (for classification) or mean squared error (for regression). You can experiment with different values of "k" to find the best-performing model for your dataset.

Key Considerations:

- k-NN is a non-parametric algorithm, which means it doesn't make assumptions about the underlying data distribution.
- The choice of distance metric, "k," and the representation of data can significantly affect the results.
- k-NN is sensitive to the scale of features, so feature scaling (e.g., normalization or standardization) is often required.
- k-NN is computationally intensive, especially with large datasets, as it requires calculating distances for each data point.

- k-NN is a straightforward and interpretable algorithm suitable for small to moderately sized datasets. However, it may not perform well on high-dimensional data, and careful consideration of parameter tuning is essential for optimal results.

Let's walk through a small example of using the k-Nearest Neighbors (k-NN) algorithm for a binary classification problem. In this example, we'll classify data points as either "Red" or "Blue" based on their coordinates on a 2D plane.

Step 1: Import Necessary Libraries: In R, you can use the "class" package for k-NN classification. First, install and load the package:

```
install.packages("class")
library(class)
```

Step 2: Generate Sample Data: For simplicity, we'll create a small dataset with example data points. Each data point has x and y coordinates and is labeled as "Red" or "Blue."

```
# Sample data
data <- data.frame( x = c(2, 3, 5, 8, 10, 12, 15, 17, 19, 22), y =
c(3, 4, 6, 8, 9, 11, 13, 15, 16, 18), label = c("Red", "Red", "Red",
"Red", "Red", "Blue", "Blue", "Blue", "Blue", "Blue") )
```

Step 3: Split Data into Training and Testing Sets: Divide the dataset into a training set and a testing set for model evaluation.

```
# Set a random seed for reproducibility
set.seed(123)
# Create an index for splitting the data (70% training, 30% testing)
index <- sample(1:nrow(data), 0.7 * nrow(data))
# Split the data
train_data <- data[index, ]
test_data <- data[-index, ]
```

Step 4: Train the k-NN Model: Train a k-NN model using the training data. For this example, we'll set k = 3 (you can experiment with different values of k).

```
# Train a k-NN model with k = 3
k <- 3
knn_model <- knn(train = train_data[, c("x", "y")], test = test_data[,
c("x", "y")], cl = train_data$label, k = k)
```

Step 5: Make Predictions: Use the trained k-NN model to make predictions on the testing data.

```
# Predict labels for the testing data
predictions <- knn_model
```

Step 6: Evaluate the Model: Evaluate the k-NN model's performance by calculating accuracy or creating a confusion matrix.

```
# Calculate accuracy
accuracy <- mean(predictions == test_data$label)
cat("Accuracy:", accuracy, "\n")
# Create a confusion matrix
conf_matrix <- table(Actual = test_data$label, Predicted =
predictions)
print(conf_matrix)
```

This simple example demonstrates how to use k-NN for a binary classification problem. The model predicts whether data points should be labeled as "Red" or "Blue" based on their coordinates. In practice, you would typically apply k-NN to more complex datasets, preprocess the data, and experiment with different values of k to find the optimal model.

In the next lecture, we'll look at Decision Trees and Random Forest.

Resources:

1. https://www.datacamp.com/tutorial/support-vector-machines-r
2. https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/
3. https://uc-r.github.io/svm
4. https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf
5. https://www.edureka.co/blog/support-vector-machine-in-r/
6. https://www.simplilearn.com/tutorials/data-science-tutorial/svm-in-r
7. https://www.projectpro.io/recipes/use-svm-classifier-r
8. https://www.kaggle.com/code/meetnagadia/support-vector-machine-r-tutorial
9. https://learndatascienceskill.com/index.php/2020/07/15/support-vector-machines-svms-with-r/
10. https://www.datacamp.com/tutorial/k-nearest-neighbors-knn-classification-with-r-tutorial
11. https://www.geeksforgeeks.org/k-nn-classifier-in-r-programming/
12. https://rpubs.com/pmtam/knn
13. https://towardsdatascience.com/k-nearest-neighbors-algorithm-with-examples-in-r-simply-explained-knn-1f2c88da405c
14. https://www.analyticsvidhya.com/blog/2015/08/learning-concept-knn-algorithms-programming/
15. https://fderyckel.github.io/machinelearningwithr/knnchapter.html