VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

LÊ QUỐC HUY – 520H0536
TĂNG ĐẠI – 520H0523
NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120
NGUYỄN THIÊN HUY - 521H0072
TRƯƠNG HUỲNH ĐĂNG KHOA – 521H0503

FINAL REPORT

MASSIVE DATA PROCESSING TECHNIQUES IN DATA SCIENCE

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

LÊ QUỐC HUY – 520H0536
TĂNG ĐẠI – 520H0523
NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120
NGUYỄN THIÊN HUY - 521H0072
TRƯƠNG HUỲNH ĐĂNG KHOA – 521H0503

FINAL REPORT

MASSIVE DATA PROCESSING TECHNIQUES IN DATA SCIENCE

Instructor

MSc. Nguyễn Thành An

HO CHI MINH CITY, 2024

ACKNOWLEDGEMENT

We sincerely thank MSc. Nguyễn Thành An for teaching us the Massive Data Processing Techniques in Data Science course with great enthusiasm. We want to express our deep appreciation for the dedication and professional knowledge that you shared with us. Through your classes, we gained a better understanding of the fundamental aspects of the course, thanks to your detailed explanations and practical applications. You helped us grasp the knowledge and apply it effectively. We have given our best effort on researching this subject. Therefore, we hope you can give your most sincere evaluation on our subject. We sincerely thank you and wish your health, success, and happiness.

Ho Chi Minh City, May 26, 2024

Authors:

(Sign and write full name)

Lê Quốc Huy

Tăng Đại

Nguyễn Đình Việt Hoàng

Nguyễn Thiên Huy

Trương Huỳnh Đăng Khoa

DECLARATION OF AUTHORSHIP

Our group assures that this is our own report and was guided by MSc. Nguyễn Thành An. The research content and results in this report are honest and have not been published in any form before. The figures in the tables used for analysis, comments, and evaluations were collected by the authors from various sources clearly stated in the reference section.

Additionally, the report includes some comments, evaluations, and data from other authors and organizations, all of which are cited and noted for their origin.

If any fraud is detected, we fully take responsibility for the content of our midterm report for the second semester of the 2023-2024 academic year. Ton Duc Thang University is not involved in any copyright or intellectual property violations that we may cause during the process (if any).

Ho Chi Minh, May 26, 2024

Authors:

(Sign and write full name)

Lê Quốc Huy

Tăng Đại

Nguyễn Đình Việt Hoàng

Nguyên Thiên Huy

Trương Huỳnh Đăng Khoa

ABSTRACT

This report investigates various machine learning techniques applied to diverse problem domains. Each chapter focuses on a specific task, providing pseudocode implementations and analyzing the resulting models. Chapter 1 explores clustering algorithms, with a focus on K-means for grouping similar data points. Chapter 2 delves into dimensionality reduction, utilizing Singular Value Decomposition (SVD) to extract essential features from high-dimensional datasets.

We further examine collaborative filtering for recommendation systems in Chapter 3, implementing the Alternating Least Squares (ALS) algorithm to predict user preferences. In Chapter 4, we apply linear regression to the challenge of stock price prediction, assessing its effectiveness in capturing market trends. Finally, Chapter 5 tackles multi-class classification, comparing the performance of Multi-layer Perceptron (MLP), Random Forest, and Linear Support Vector Machine (SVM) models.

Our results demonstrate the practical utility of these algorithms across a variety of applications. The K-means algorithm successfully identifies distinct clusters within datasets, while SVD efficiently reduces dimensionality without significant information loss. Collaborative filtering with ALS provides accurate recommendations, and linear regression offers insights into stock price movements. The multi-class classification comparison reveals the strengths and weaknesses of different models, aiding in informed algorithm selection.

TABLE OF CONTENTS

LIST OF FIGURES

Student list:

Full name	Student ID	Task	Completion (%)
Lê Quốc Huy	520H0536	Task 1, Report	100
Trương Huỳnh Đăng Khoa	521H0503	Task 2	100
Tăng Đại	520H0523	Task 3, Task 5	100
Nguyễn Thiên Huy	521H0072	Task 4	100
Nguyễn Đình Việt Hoàng	522H0120	Task 5, Report	100

CHAPTER 1 - CLUSTERING

1.1 Study Topic: Clustering, K-Means

Imagine you're organizing your bookshelf. You wouldn't jumble novels next to cookbooks, right? Clustering is similar.

The Art of Grouping: Clustering is an unsupervised learning technique that doesn't rely on pre-labeled data. Instead, it sifts through your dataset, identifying intrinsic similarities between data points and grouping them into distinct categories called "clusters." Think of it as automatically sorting a mountain of socks into pairs

Example: You have a dataset of customer purchases. Clustering can group customers with similar buying habits together. This can help target marketing campaigns or recommend products more effectively.

One of the most popular clustering algorithms is **K-means**. It's like a diligent organizer, dividing your data into a predetermined number of clusters (represented by the variable "k").

How it Works:

1. **Scatter the Seeds**: K-means starts by randomly scattering "k" points throughout your data. These points are like initial guesses for where the centers of your clusters might be.
2. **Gather the Flock**: Each data point is then assigned to the nearest cluster center, forming initial clusters.
3. **Re-center the Flock**: K-means recalculates the center of each cluster based on the average position of all the data points within it.
4. **Repeat Until Stability**: Steps 2 and 3 are repeated until the cluster centers stop moving significantly. This indicates that the clusters have stabilized and the algorithm has found the optimal grouping.

Advantages:

- **Simple and easy to implement:** K-means is a relatively simple algorithm in concept and is easy to implement in various programming languages.
- **Fast and efficient:** K-means can process large datasets quickly and efficiently.
- **Guaranteed convergence:** The K-means algorithm always converges to a result (although it is not guaranteed to be globally optimal).
- **Easy to interpret:** The clusters produced by K-means are often easy to understand and can be visualized.

Disadvantages:

- **Requires pre-defined number of clusters (K):** The user needs to specify the desired number of clusters (K) beforehand, which can be difficult without prior knowledge of the data.
- **Sensitive to initialization values:** The results of K-means can vary significantly depending on the initial placement of cluster centroids.
- **Sensitive to noise and outliers:** Noise and outliers in the data can significantly affect the position of cluster centroids and the clustering results.
- **Assumes spherical cluster shapes:** K-means assumes that clusters are spherical and of similar size, which may not hold true in real-world scenarios.

When to use K-means?

K-means is most suitable when:

- You need a simple and fast clustering algorithm.
- You know the desired number of clusters (K) beforehand.
- Your data consists of spherical clusters of similar size.
- You are not too concerned about noise and outliers in the data.

1.2 Approach to solve task

In Task 1, we use **mnist_mini.csv** for this task. Pseudocode:

function CLUSTER(data, k, seed=42) returns (model, results)

model ← KMEANS(data, k, seed)

results ← model.transform(data)

return (model, results)

function CALCULATE_AVERAGE_DISTANCES(model, results) returns distances

distances ← []

for each cluster_id in range(k):

cluster_points ← results.filter(results.prediction == cluster_id)

centroid ← model.clusterCenters()[cluster_id]

squared_distances ← [Vectors.squared_distance(point, centroid) for point in cluster_points]

average_distance ← sum(squared_distances) / len(squared_distances) distances.append(average_distance)

return distances

function VISUALIZE_DISTANCES(distances)

plt.bar(range(len(distances)), distances) plt.xlabel("Cluster ID")

plt.ylabel("Average Distance to Centroid")

plt.title("Average Distances to Cluster Centroids") plt.show()

1.3 Result

With k = 10, and weighted indices like the given task we have the result:

```
clusterer = Clustering(spark, data)
weighted_indices = [0, 1, 2, 3, 4, 7, 8, 11, 18, 61]
k = 10

model, transformed_df = clusterer.cluster(k)
avg_distances = clusterer.calculate_average_distances(model, transformed_df)
clusterer.visualize_distances(avg_distances)
```

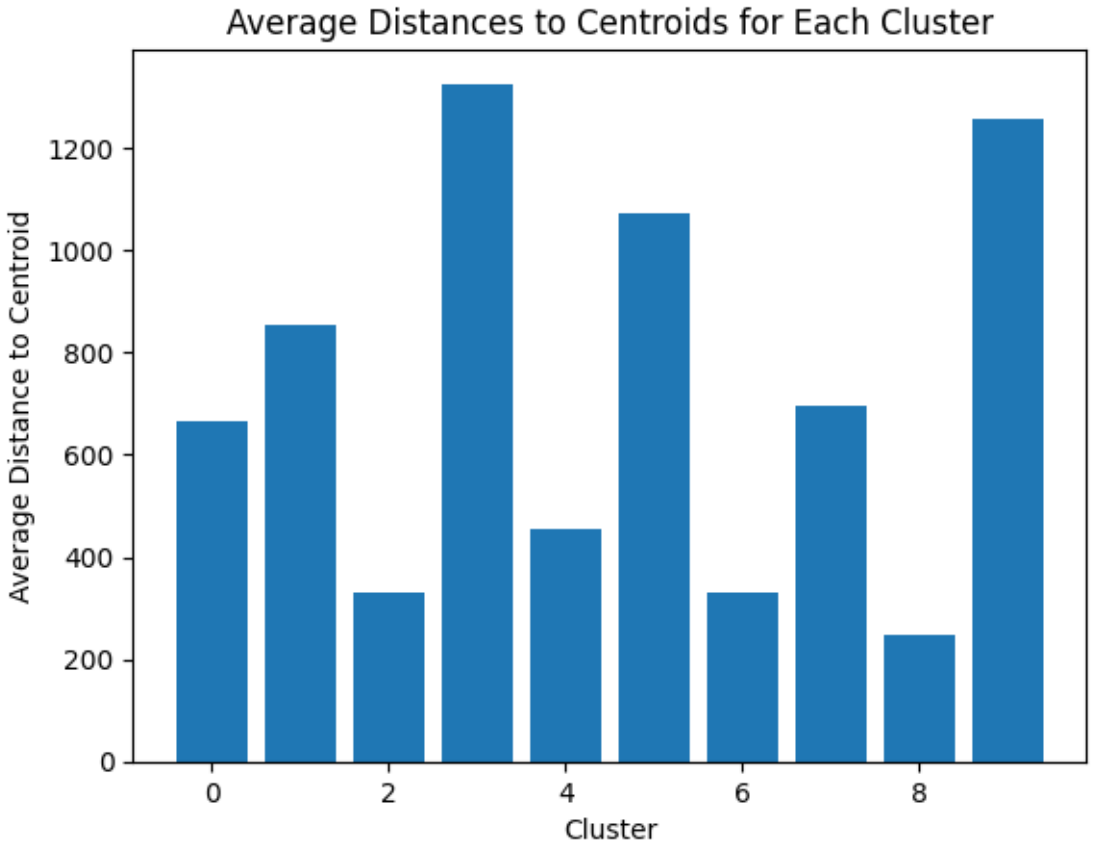

Figure 1. Task 1 Result

CHAPTER 2 – DIMENSIONALITY DEDUCTION WITH SVD

2.1 Study Topic: SVD

Dimensionality reduction with SVD (Singular Value Decomposition) is a powerful technique for simplifying complex data and extracting the most important information

SVD: Decomposing and Reconstructing

Imagine your data as a vast, multilayered matrix. SVD acts like a cosmic scalpel, dissecting this matrix into three fundamental components:

- **U: Left Singular Vector Matrix:** This matrix encapsulates information about the variability within your data. It tells us how the original data points are dispersed and in what directions they vary the most.
- **Σ (Sigma): The Singular Value Matrix:** This diagonal matrix holds the keys to the kingdom. The singular values within Σ are like a ranking system, quantifying the importance of each dimension in your data. The larger the singular value, the more influential the corresponding dimension.
- *V: **Right Singular Vector Matrix**:* This matrix provides a window into the underlying structure of your data. It reveals the original directions of the data and how those directions are related to the variability captured in U.

To understand the power of SVD, we need to confront the "curse of dimensionality." Imagine a dataset filled with information about customers: age, income, purchase history, browsing behavior, social media activity, and more. This treasure trove of data becomes overwhelming.

The curse of dimensionality refers to the phenomenon where, as the number of dimensions (features) increases, the data becomes increasingly sparse. This

sparsity makes it difficult for algorithms to find meaningful patterns. Additionally, distances between data points become less meaningful, hindering tasks like clustering and classification.

Advantages of SVD-Based Dimensionality Reduction

The benefits of employing SVD for dimensionality reduction are manifold:

1. **Computational Efficiency**: Machine learning algorithms often scale poorly with high-dimensional data. SVD reduces the computational burden without sacrificing critical information.
2. **Enhanced Visualization**: High-dimensional data is inherently difficult to visualize. SVD enables the projection of data onto a lower-dimensional subspace, facilitating visual exploration and pattern recognition.
3. **Noise Mitigation**: SVD can act as a noise filter by discarding dimensions associated with small singular values, which are often dominated by noise.
4. **Feature Selection**: The singular vectors associated with the largest singular values can be interpreted as the most important features, aiding in feature selection tasks.

Disadvantages:

1. **Computational Cost:** SVD can be computationally expensive for large datasets, especially when calculating all singular values and vectors. This can limit its applicability in real-time or large-scale applications.
2. **Interpretability:** The resulting singular vectors can be difficult to interpret, as they represent linear combinations of the original features. This makes it challenging to understand the meaning of the reduced dimensions.
3. **Loss of Information:** While SVD aims to preserve the most important information in the data, some information loss is inevitable when reducing dimensionality. This can impact the performance of downstream tasks that rely on the reduced representation.

4. **Sensitivity to Noise:** SVD can be sensitive to noise in the data, especially when the signal-to-noise ratio is low. This can lead to inaccurate dimensionality reduction and affect the quality of the results.
5. **Assumption of Linearity:** SVD assumes that the underlying data relationships are linear. If the data exhibits non-linear patterns, SVD might not be the most effective dimensionality reduction technique.
6. **Not Suitable for Sparse Data:** SVD is not well-suited for sparse datasets, where most entries are zero. This is because it operates on the entire matrix, including the zero entries, leading to unnecessary computational overhead.

Applicability of SVD

SVD is a versatile tool with applications across diverse domains. It is particularly well-suited for scenarios where:

1. The dataset comprises a large number of features, potentially with redundancies.
2. The primary objective is to capture the underlying structure of the data, rather than precise reconstruction.
3. Visualization and interpretability are crucial components of the analysis.

2.2 Approach to solve task

In task 2, we use **mnist_mini.csv** for this task and use **PySpark** and the **SVD** algorithm to reduce the dimensionality of data points from 784 to 3.

This class performs dimensionality reduction on Spark DataFrames using Singular Value Decomposition (SVD).

Pseudocode:

```
function DIMENSIONALITY_REDUCTION(data, r=3) returns (U, s, V, reduced_data)
        indexed_row_rdd ← PREPROCESS_DATA(data)
        matrix ← IndexedRowMatrix(indexed_row_rdd)
        (U, s, V) ← matrix.computeSVD(r, computeU=True)
        reduced_data ← []
        for row in U:
            reduced_data.append(row[:r].toArray())
        return (U, s, V, reduced_data)
function GET_RANDOM_SAMPLES(reduced_data, sample_size) returns samples
        if reduced_data is None:
            raise ValueError("No reduced data available. Run dimensionality reduction first.")
        indices ← random_sample(range(len(reduced_data)), sample_size)
        samples ← [reduced_data[i] for i in indices]
        return samples
function PLOT_SAMPLES(samples)
        fig ← plt.figure()
        ax ← fig.add_subplot(111, projection='3d')
        ax.scatter(*zip(*samples), c='b', marker='o')
        ax.set_xlabel('Component 1')
        ax.set_ylabel('Component 2')
        ax.set_zlabel('Component 3')
        plt.show()
```

2.3 Result

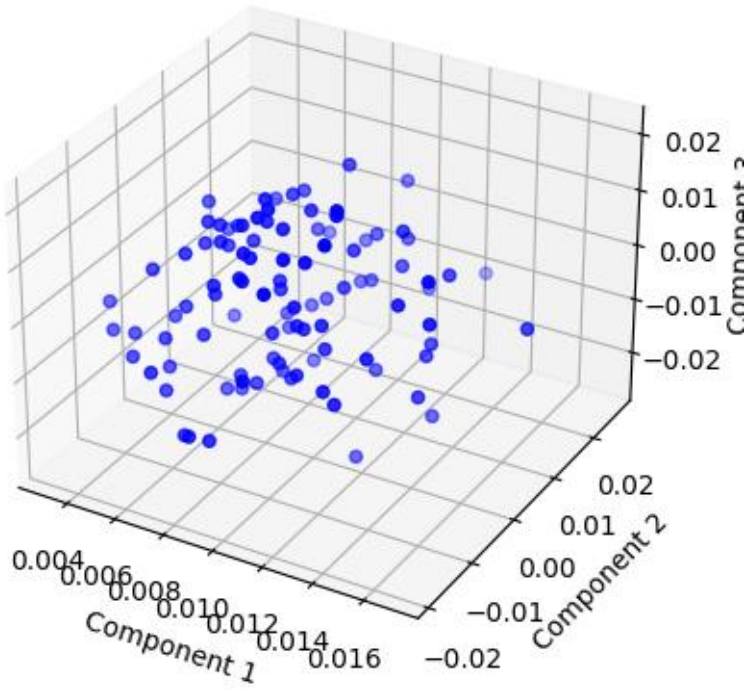

Figure 2. Task 2 Result

CHAPTER 3 – RECOMMENDATION SYSTEM

3.1 Study Topic

Recommender systems, essential tools in various digital platforms, aim to provide users with personalized recommendations. Collaborative filtering (CF) is a prominent technique within recommender systems, leveraging the collective behavior of users to predict preferences. Within CF, the Alternating Least Squares (ALS) algorithm and the Mean Squared Error (MSE) metric play crucial roles in model optimization and evaluation.

Alternating Least Squares (ALS) Algorithm

ALS is a matrix factorization method that is particularly well-suited to the collaborative filtering problem. It operates on a user-item matrix, where rows

represent users, columns represent items, and cell values signify user ratings for items. This matrix is often sparse, meaning many ratings are missing.

ALS decomposes this sparse matrix into two lower-dimensional matrices:

- User Factor Matrix: Each row represents a user's latent preferences, where each latent factor captures a distinct aspect of their taste.
- Item Factor Matrix: Each row encapsulates an item's latent characteristics, with latent factors representing different attributes of the item.
- The ALS algorithm iteratively alternates between fixing one factor matrix and optimizing the other, hence the name "Alternating Least Squares." During each iteration, it refines these matrices to minimize the discrepancy between predicted ratings and observed ratings. This process is guided by a loss function, often the Mean Squared Error (MSE).

Mean Squared Error (MSE)

MSE is a commonly used loss function in recommender systems, quantifying the average of the squared differences between predicted ratings and actual ratings:

```
MSE = (1 / n) * Σ(actual rating - predicted rating)^2
```

where *n* is the number of observed ratings. A lower MSE value indicates a better fit of the model to the data, signifying that the predicted ratings are closer to the true preferences of users.

Advantages:

1. **Scalability**: ALS is highly scalable and can handle very large datasets efficiently. It's well-suited for distributed computing environments, allowing parallel processing across multiple machines.
2. **Handles Sparsity**: Recommender system datasets are often sparse, meaning users rate only a small fraction of items. ALS effectively deals with this sparsity by using regularization techniques to prevent overfitting and produce meaningful recommendations.

3. **Implicit Feedback**: ALS can be readily adapted to work with implicit feedback data (e.g., purchase history, browsing behavior) in addition to explicit ratings. This is valuable as implicit feedback is often more abundant than explicit ratings.
4. **Interpretability**: The latent factors learned by ALS can provide some interpretability about user preferences and item characteristics. This can be helpful for understanding why certain recommendations are made.
5. **Flexibility**: ALS allows for incorporating additional information, such as user demographics or item features, to improve recommendations. This flexibility enhances the algorithm's adaptability to different scenarios.

Disadvantages:

1. **Cold Start Problem**: ALS struggles with the cold start problem, which refers to the difficulty of making recommendations for new users or items with limited interaction history. It requires sufficient data to learn meaningful latent factors.
2. **Assumption of Linearity**: ALS assumes linear relationships between users and items, which might not always hold true in real-world scenarios. Non-linear relationships can lead to suboptimal recommendations.
3. **Parameter Tuning**: ALS requires tuning hyperparameters like regularization strength and the number of latent factors. The optimal values depend on the dataset and may require experimentation to find.
4. **Computationally Intensive**: While scalable, ALS can still be computationally intensive for very large datasets, especially when dealing with a high number of latent factors.
5. **Limited to Matrix Factorization**: ALS is primarily a matrix factorization technique, meaning it focuses on finding latent factors. It

might not capture complex interactions that go beyond simple user-item relationships.

The Connection Between ALS and MSE

In the ALS algorithm, the MSE serves as the optimization objective. By iteratively minimizing the MSE, ALS refines the user and item factor matrices, gradually improving the accuracy of its predictions. As the MSE decreases, the model's ability to capture user-item relationships and generate meaningful recommendations increases.

combination of ALS and MSE is potent for building effective recommender systems. ALS's ability to handle large-scale, sparse data, coupled with MSE's straightforward interpretation and optimization properties, has made this approach a popular choice in real-world applications.

3.2 Approach to solve task

In task 3, we use **ratings2k.csv** for this task. Split the given data set into training and test sets with the fraction 7 : 3

Pseudocode:

function BUILD_MODEL(data, rank, maxIter=10, regParam=0.1) returns model

print("Building model with rank:", rank)

als ← ALS(maxIter=maxIter, regParam=regParam, rank=rank, userCol="user", itemCol="item", ratingCol="rating", coldStartStrategy="drop")

model ← als.fit(data)

return model

function EVALUATE_MODEL(model, data) returns mse

print("Evaluating model...")

```
        predictions ← model.transform(data)
        evaluator ← RegressionEvaluator(metricName="mse")
        mse ← evaluator.evaluate(predictions)
        print(f"Evaluation results - Mean Squared Error (MSE): {mse}")
        return mse
    function RUN_ALS_EXPERIMENT(data, rank_range) returns mse_values
(training, test) ← preprocess_data(data)
        mse_values ← []
        for rank in rank_range:
            model ← BUILD_MODEL(training, rank)
            mse ← EVALUATE_MODEL(model, test)
            mse_values.append(mse)
            print(f"Rank: {rank}, MSE: {mse}")
        VISUALIZE_RESULTS(rank_range, mse_values)
        return mse_values
    function VISUALIZE_RESULTS(rank_range, mse_values)
        plt.bar(rank_range, mse_values)
        plt.xlabel("Rank")
        plt.ylabel("Mean Squared Error (MSE)")
        plt.title("Model Performance by Rank")
        plt.show()
```

3.3 Result

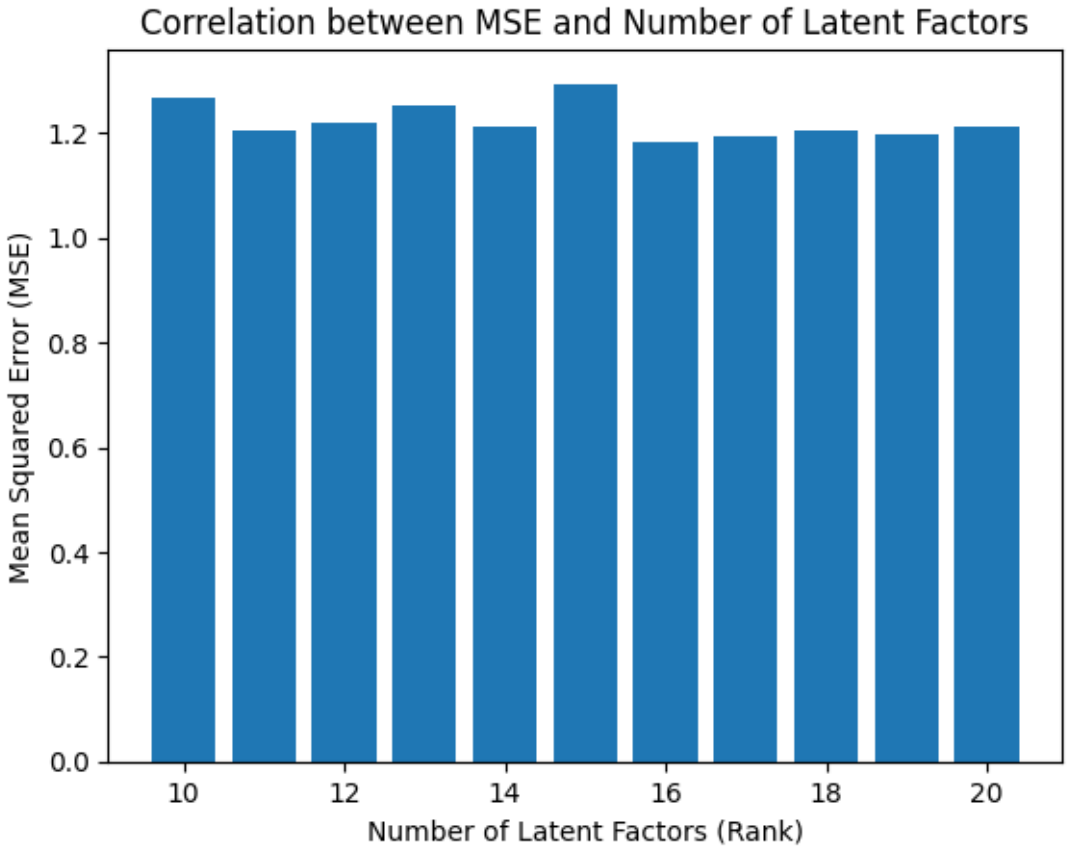

Figure 3. Task 3 Result

CHAPTER 4 – STOCK PRICE REGRESSION

4.1 Study topic

Linear Regression in Stock Price Prediction: A Critical Analysis

In the realm of financial forecasting, the prediction of stock prices remains a complex and elusive endeavor. While numerous sophisticated models have been developed to address this challenge, linear regression, a foundational statistical technique, continues to hold a prominent position. Despite its inherent limitations, linear regression offers a valuable starting point for understanding the intricate relationship between variables influencing stock prices.

Linear Regression Model: A Primer

Linear regression is a statistical method that establishes a linear relationship between a dependent variable (the variable being predicted) and one or more independent variables (factors presumed to influence the dependent variable). In the context of stock price prediction, this translates to:

- Dependent Variable: The stock price, typically represented by the closing price.
- Independent Variables: A plethora of factors that may impact the stock price, encompassing:
  - Company-specific financials (earnings, revenue, etc.)
  - Broad market trends (e.g., major stock indices)
  - Industry performance metrics
  - Macroeconomic indicators (interest rates, inflation)
  - Trading volume

The linear regression model aims to create a best-fit line that minimizes the difference between the actual stock prices and the prices predicted by the equation. This line represents the general trend in the data, providing insights into the average relationship between the dependent and independent variables.

Advantages:

1. **Simplicity and Interpretability:** Linear regression models are easy to understand and interpret. The relationship between the predictor variables (e.g., historical prices, economic indicators) and the stock price is represented by a linear equation. This makes it easier for analysts and investors to grasp the factors influencing the prediction.
2. **Computational Efficiency:** Linear regression models are computationally efficient to train and make predictions. This is particularly beneficial for real-time stock price prediction scenarios where quick decisions are required.
3. **Baseline Model:** Linear regression can serve as a baseline model to compare with more complex prediction models. It helps assess the added value of sophisticated techniques and determine if their complexity is justified by improved performance.
4. **Handles Linear Relationships:** In some cases, stock prices may exhibit linear relationships with certain predictor variables. Linear regression can effectively capture and model such relationships, providing accurate predictions in these scenarios.
5. **Feature Importance:** Linear regression models can provide insights into the importance of different features in predicting stock prices. The coefficients associated with each predictor variable indicate their relative contribution to the prediction, aiding in feature selection and understanding the underlying factors

Limitations and Cautions

Despite its simplicity and ease of interpretation, linear regression possesses inherent limitations that must be acknowledged:

1. **Non-Linearity**: Stock prices often exhibit non-linear behavior due to the influence of various unpredictable events, market sentiments, and

sudden shifts in investor behavior. Linear regression may struggle to capture these complexities accurately.

2. **Historical Bias**: The model relies heavily on historical data, assuming that past relationships between variables will persist in the future. This assumption can be problematic as market dynamics are subject to change.
3. **Exogenous Factors**: External events, such as geopolitical developments, regulatory changes, or unexpected news, can trigger significant fluctuations in stock prices, independent of the variables incorporated into the model.

Utility of Linear Regression

While these limitations are substantial, linear regression should not be dismissed as a useful tool for stock price analysis:

1. Relationship Identification: Linear regression can reveal correlations between specific factors (e.g., earnings, market trends) and stock prices, providing a basic understanding of their influence.
2. Trend Detection: The model can effectively identify broad upward or downward trends in the data, offering a simplified overview of price movements over time.
3. Foundation for Advanced Models: Linear regression can serve as a building block for more sophisticated models. By incorporating additional factors, non-linear relationships, and advanced statistical techniques, the predictive power of the model can be enhanced.

4.2 Approach to solve task

Pseudo code:

```
class STOCK_FLUC_MODEL:
    def TRAIN_MODEL(self) returns LinearRegressionModel:
```

```
        lr ← LinearRegression(featuresCol='features', labelCol='label')
self.model ← lr.fit(self.train_data) return self.model
        def EVALUATE_MODEL(self) returns (float, float):
                train_predictions ← self.model.transform(self.train_data)
                test_predictions ← self.model.transform(self.test_data)
                train_mse ← self.evaluator.evaluate(train_predictions)
                test_mse ← self.evaluator.evaluate(test_predictions)
                return train_mse, test_mse def PREDICT(self) returns
        DataFrame:
                predictions ← self.model.transform(self.test_data)
                return predictions.select('prediction')
        def PLOT_MSE(self, train_mse, test_mse):
                plt.bar(['Train MSE', 'Test MSE'], [train_mse, test_mse],
color=['blue', 'orange'])
                plt.ylabel('Mean Square Error')
                plt.title('MSE for Train and Test Sets') plt.show()


def MAIN():
        # DATA PREPARATION
        processor ← STOCK_DATA_PROCESSOR()
        data                                                    ←
processor.LOAD_DATA('/content/drive/MyDrive/Colab_Notebooks/stockH
VN2022.csv')
        print("Samples of dataset:")
        data.show(5)
        print("The number of rows of dataset:", data.count())
        print("Columns of dataset:") data.printSchema() (train_data, test_data)
← processor.SPLIT_DATA()
```

```
prepared_train_data ← processor.PREPARE_DATA(train_data)
prepared_test_data ← processor.PREPARE_DATA(test_data)
# MODEL TRAINING AND EVALUATION
model ← STOCK_FLUC_MODEL(prepared_train_data, prepared_test_data)
model.TRAIN_MODEL()
(train_mse, test_mse) ← model.EVALUATE_MODEL()
model.PLOT_MSE(train_mse, test_mse)
# PREDICTION
predictions ← model.PREDICT()
print("Predictions:") predictions.show(5)
# VISUALIZATION
actual_values ← [row['label'] for row in prepared_test_data.collect()]
predicted_values ← [row['prediction'] for row in predictions.collect()]
dates ← [row['Ngay'] for row in prepared_test_data.collect()]
plt.figure(figsize=(10, 6))
plt.plot(dates, actual_values, label='Actual Values', marker='o')
plt.plot(dates, predicted_values, label='Predicted Values', marker='x')
plt.xlabel('Date')
plt.ylabel('Fluctuation')
plt.title('Actual vs Predicted Fluctuation')
plt.legend() plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

4.3 Result

```
Samples of dataset:
+----------+-----+
|      Ngay|  HVN|
+----------+-----+
|2022-01-04| 23.3|
|2022-01-05| 23.1|
|2022-01-06|22.85|
|2022-01-07|22.65|
|2022-01-10| 23.2|
+----------+-----+
only showing top 5 rows

The number of rows of dataset:  219
Columns of dataset:
root
 |-- Ngay: date (nullable = true)
 |-- HVN: float (nullable = true)
```

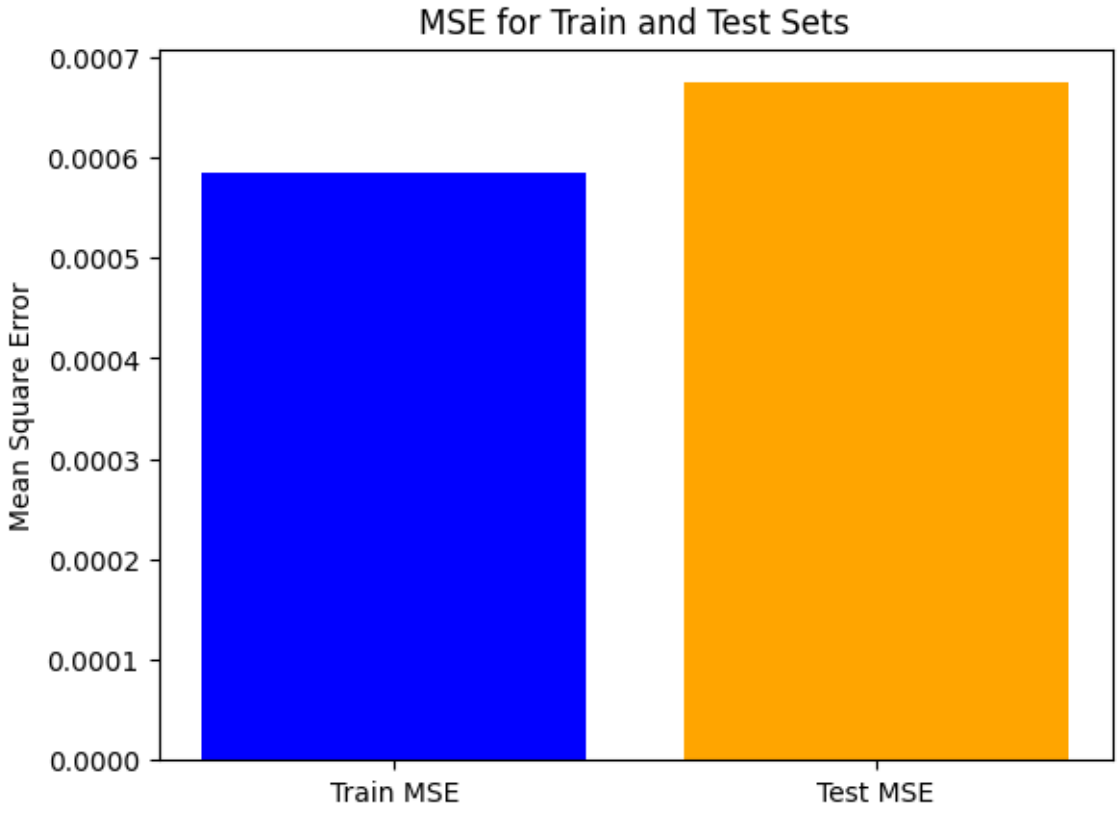

Figure 4. Mean Square Error

```
+--------------------+
|          prediction|
+--------------------+
|-0.00337246015779...|
|-6.40023528657115E-4|
|-0.00951425708381701|
|-5.47631591134960...|
|0.003675510125960...|
+--------------------+
only showing top 5 rows
```

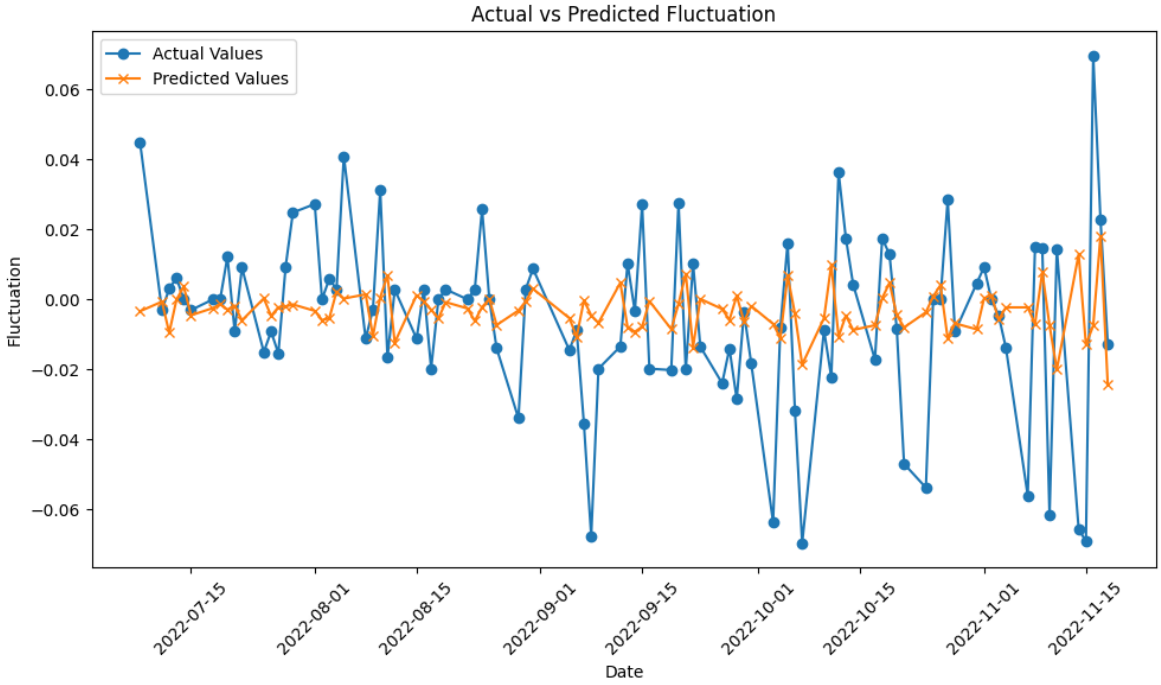

Figure 5. Task 4 Result

CHAPTER 5 – MULTI-CLASS CLASSIFICATION

5.1 Study Topic

Apache Spark, a powerful engine for big data processing, offers a robust library of machine learning algorithms through its MLlib and ML components. In this report, we delve into three prominent algorithms: Multi-Layer Perceptron (MLP), Random Forest, and Linear Support Vector Machine (SVM), comparing their characteristics, strengths, weaknesses, and optimal use cases.

Multi-Layer Perceptron (MLP)

Type: Artificial Neural Network (ANN)

Strengths:

- **Universal Approximation:** MLPs possess the capability to approximate arbitrary complex, non-linear functions, making them well-suited for tasks where intricate relationships exist between features and target variables.
- **High-Dimensional Data:** MLPs are adept at handling high-dimensional data, extracting meaningful patterns and representations.
- **Adaptability:** Through backpropagation and gradient descent, MLPs can learn from data and adjust their parameters to improve prediction accuracy.

Weaknesses:

- **Computational Cost:** Training MLPs can be computationally intensive, especially with deep architectures and large datasets.
- **Hyperparameter Tuning:** Achieving optimal performance often necessitates meticulous tuning of hyperparameters such as learning rate, activation functions, and the number of hidden layers/neurons.
- **Overfitting:** Without proper regularization techniques (e.g., dropout, early stopping), MLPs can be susceptible to overfitting, hindering their generalization to unseen data.
- **Interpretability:** The inherent complexity of MLPs can make them less interpretable compared to simpler models.

When to Use: MLPs are particularly valuable for complex classification and regression tasks where simpler algorithms may falter. Their ability to handle high-dimensional data and model non-linear relationships makes them a powerful tool for a wide array of applications, including image recognition, natural language processing, and time series forecasting.

Random Forest

Type: Ensemble learning method (combines multiple decision trees)

Strengths:

- **Robustness to Overfitting:** Random Forests are generally less prone to overfitting than individual decision trees due to the averaging of predictions across multiple trees.
- **Feature Handling:** They can effectively handle both categorical and numerical features, requiring minimal preprocessing.
- **Feature Importance:** Random Forests can provide feature importance scores, indicating the relative contribution of each feature to the model's predictions, enhancing interpretability.

Weaknesses:

- **Model Complexity:** While less complex than deep neural networks, Random Forests can still be computationally expensive to train, particularly with large datasets or a high number of trees.
- **Data Requirement:** In some cases, Random Forests may require a larger amount of data to achieve optimal performance compared to simpler models.
- **Black-Box Nature:** Although feature importance scores offer some interpretability, the decision-making process of a Random Forest can still be challenging to fully comprehend.

When to Use: Random Forests are versatile and applicable to a wide range of classification and regression problems. Their robustness, feature handling capabilities, and interpretability make them a popular choice for tasks where accuracy, stability, and understanding feature contributions are important.

Linear Support Vector Machine (SVM)

Type: Kernel method (finds the optimal hyperplane separating classes)

Strengths:

- **Efficiency:** Linear SVMs are computationally efficient, especially for high-dimensional datasets with sparse features.

- **Linear Separability:** They excel in problems where classes are clearly separable by a linear boundary.
- **Interpretability:** Support vectors (data points closest to the decision boundary) can offer insights into the model's decision-making process.

Weaknesses:

- **Non-Linear Relationships:** Linear SVMs struggle with datasets exhibiting complex, non-linear relationships. Non-linear kernels can be used, but they introduce computational overhead and may require careful parameter tuning.
- **Sensitivity:** SVMs are sensitive to outliers and the scaling of features. Proper preprocessing is essential to mitigate these sensitivities.

When to Use: Linear SVMs are a suitable choice for classification tasks with well-defined class boundaries, particularly when dealing with high-dimensional data. They can also be applied to linear regression problems.

5.2 Approach to solve task

In task 5, we use **mnist_mini.csv** for this task:

Pseudocode:

function TRAIN_EVALUATE(model, train_data, test_data) returns (train_accuracy, test_accuracy)

model ← model.fit(train_data)

train_predictions ← model.transform(train_data)

test_predictions ← model.transform(test_data)

evaluator ← MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

train_accuracy ← evaluator.evaluate(train_predictions)

test_accuracy ← evaluator.evaluate(test_predictions)

return (train_accuracy, test_accuracy)

```
function RUN_CLASSIFIERS(train_data, test_data) returns results
        results ← {}
        # Multi-Layer Perceptron (MLP)
        layers ← [784, 128, 64, 10] # Assuming 10 classes
        mlp ← MultilayerPerceptronClassifier(layers=layers,
labelCol="label", featuresCol="features", maxIter=100)
        results["MLP"] ← TRAIN_EVALUATE(mlp, train_data, test_data)
        # Random Forest
        rf ← RandomForestClassifier(labelCol="label",
featuresCol="features", numTrees=100)
        results["RandomForest"] ← TRAIN_EVALUATE(rf, train_data,
test_data)
        # One-vs-Rest Linear SVM
        lsvc ← LinearSVC(maxIter=100, regParam=0.1, labelCol="label",
featuresCol="features")
        ovr ← OneVsRest(classifier=lsvc)
        results["OneVsRest_LinearSVM"] ← TRAIN_EVALUATE(ovr,
train_data, test_data)
        return results

function PLOT_RESULTS(results)
        model_names ← list(results.keys())
        train_acc ← [results[model][0] for model in model_names]
        test_acc ← [results[model][1] for model in model_names] fig,
        ax ← plt.subplots() bar_width = 0.35
        x_pos = range(len(model_names))

        ax.bar(x_pos, train_acc, bar_width, label='Train Accuracy')
```

ax.bar([x + bar_width for x in x_pos], test_acc, bar_width, label='Test Accuracy')

ax.set_xlabel('Classifier')

ax.set_ylabel('Accuracy')

ax.set_title('Training and Testing Accuracy by Classifier')

ax.set_xticks([x + bar_width / 2 for x in x_pos])

ax.set_xticklabels(model_names) ax.legend() plt.show()

5.3 Result

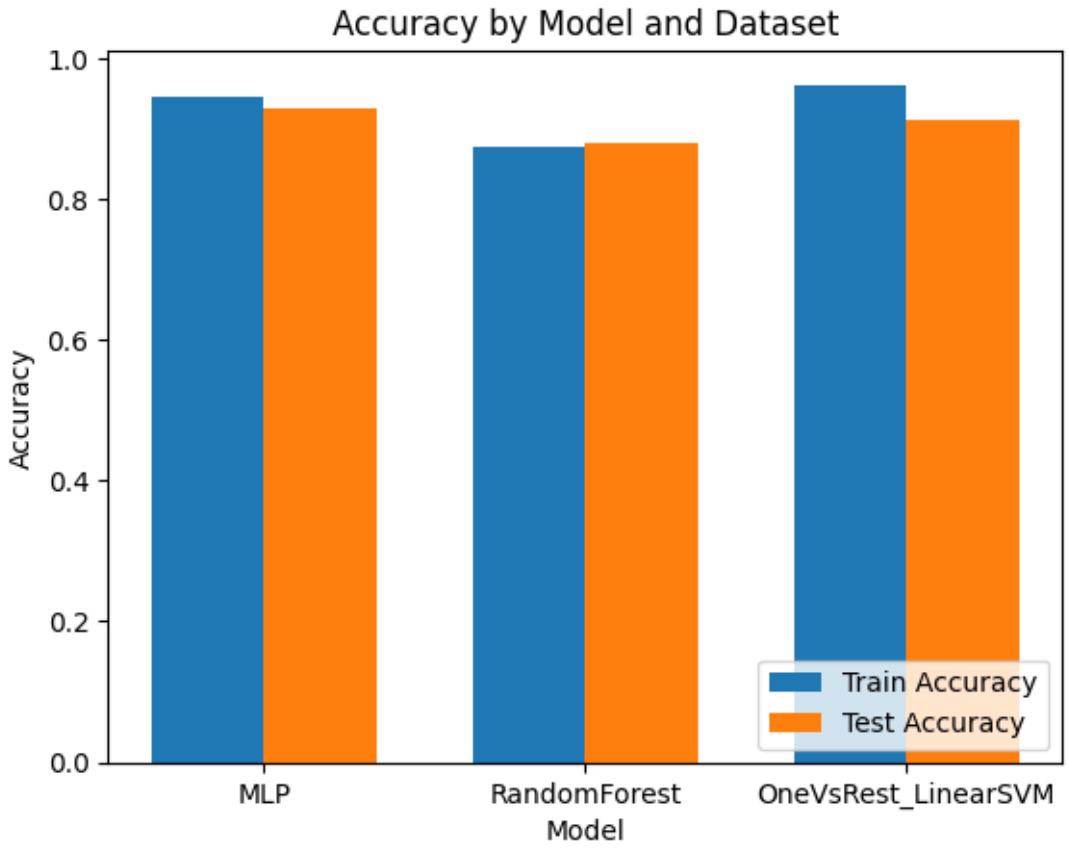

Figure 6. Task 5 Result

REFERENCES

[1] Spark MLlib Developers. "Clustering - KMeans." *PySpark Documentation*, [Online]. Available: https://spark.apache.org/docs/latest/ml-clustering.html#k-means

[2] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Franklin, M. J. (2016). MLlib: Machine learning in Apache Spark. *Journal of Machine Learning Research*, 17(1), 1235-1241.

[3] Spark MLlib Developers. "Dimensionality Reduction - RDD-based API." *PySpark Documentation*, [Online]. Available: https://spark.apache.org/docs/latest/mllib-dimensionality-reduction.html.

[4] Spark MLlib Developers. "RegressionEvaluator." *PySpark Documentation*, [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.RegressionEvaluator.html

[5] Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer Science & Business Media.

[6] Spark MLlib Developers. "Classification and Regression." *PySpark Documentation*, [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html.

Specifically, look at the sections on:

- Multilayer Perceptron Classifier
- Random Forest Classifier
- Linear Support Vector Machine
- One-vs-Rest Classifier (for multi-class SVM)