MNIST Classification Project

Magesh Babu Madhana Gopal

Machine Learning

Kunskapskontroll-2

2024-03

Abstract

This project explores the task of classifying handwritten digits using the popular MNIST dataset. Leveraging the scikit-learn library, several machine learning models including Logistic Regression, K-Nearest Neighbors, Decision Tree, and Random Forest Classifier were trained and evaluated. The performance of each model was assessed based on accuracy metrics, with the Random Forest Classifier emerging as the top-performing algorithm. Subsequently, the model was deployed into a Streamlit application, allowing for real-time digit recognition. This project showcases the effectiveness of machine learning techniques in tackling the MNIST classification problem and demonstrates the practical implementation of the selected model in a user-friendly interface.

Table of Contents

1 Introduction

The objective of this project is to employ a machine learning model for the detection of handwritten digits. Each individual possesses a unique handwriting style, and humans can effortlessly discern the exact digit by observing handwritten samples due to lifelong exposure and training. However, for a machine learning model to achieve similar accuracy, it necessitates extensive training on thousands of handwritten digit samples to predict unseen data effectively. As this is a classification problem, we will explore various machine learning classifier models using the MNIST dataset.

1.1 About MNIST Dataset

The MNIST dataset is a widely used benchmark dataset in the field of machine learning and computer vision. It consists of a collection of handwritten digits, ranging from 0 to 9, each represented as grayscale images of size 28x28 pixels. MNIST stands for Modified National Institute of Standards and Technology, where it originated from a larger dataset developed for training various machine learning algorithms.

Originally constructed for the task of handwritten digit recognition, the MNIST dataset has become a standard benchmark for evaluating the performance of different machine learning models, particularly in the realm of image classification. It comprises a training set of 60,000 examples and a test set of 10,000 examples, making it a convenient resource for both training and evaluating models. Due to its simplicity and availability, MNIST has served as a foundational dataset for researchers and practitioners alike, enabling the development and comparison of various algorithms and techniques in the domain of image recognition and classification.

2 Theory

2.1 Classification Algorithm

The classification algorithm is a supervised learning technique that is used to identify the category of new observation on the basis of training data. There are four types of classification problem. They are Binary classification, multi-class classification, multi-label classification and imbalanced classification. Our project comes under multiclass classification problem because we have to predict values (0-9) which are more than 2 classes from the given input data (MNIST dataset). (Ramakrishnan, 2022)

Based on the nature of problem and computational power, I have selected the four algorithms to train the MNIST dataset.

2.1.1 Logistic Regression

Logistic regression is a machine learning classification algorithm that is used to predict the probability of certain classes based on some dependent variables. The main advantages of logistic regression are easier implantation, suitable for linearly separable datasets and provides valuable insights. In order to predict the outcome and their probabilities, a logistic function called a sigmoid function is used. It is a function refers to an S-shaped curve that converts any real value to a range between 0 and 1. It is a fundamental component used to model the probability that a given input belongs to a particular class. (Kanade, 2022)

$$f(x) = \frac{1}{1+e^{x}}$$ Sigmoid function

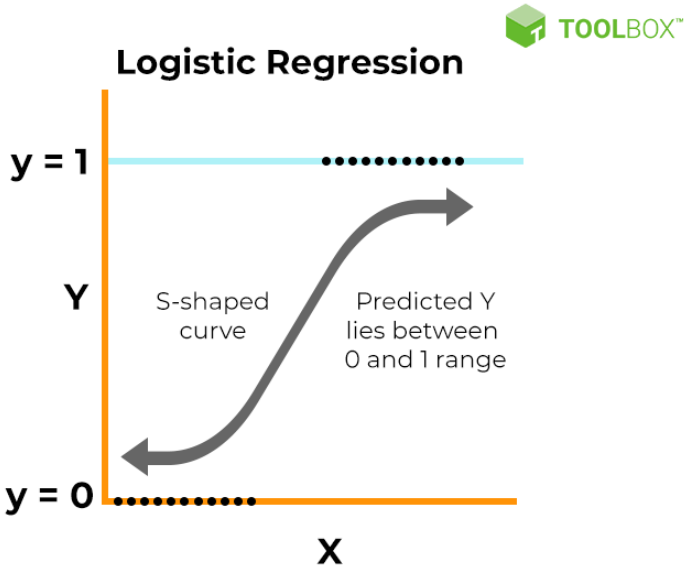

Figure 1- Logistic Regression Curve

Generally logistic regression is designed for binary class classification which uses binomial probability distribution function, but we can do some modification to make it work in multiclass classification problem through multinomial probability distribution function. (Brownlee, 2020)

2.1.2 KNearestNeighbors

The K-Nearest Neighbors (KNN) algorithm is a popular machine learning technique in Classification problem. When training the model, it stores the entire training dataset and use it as a reference when making the prediction on the new data. It calculates the distance between new data point and existing data point from the stored training dataset. The most common formula to calculate the distance is Euclidean distance.

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

The K value in the model denotes the number of neighbors to consider before classifying the new data point. If we assume value of k is 3, then the algorithm will only consider three nearest neighbors to classify the new data points. Among the 3 neighbors, the algorithm chooses to classify the new data points using majority votes. In the picture below shows that new green data point classified as red category because among three of the nearest neighbors, two neighbors are from red class. (Abba, 2023)

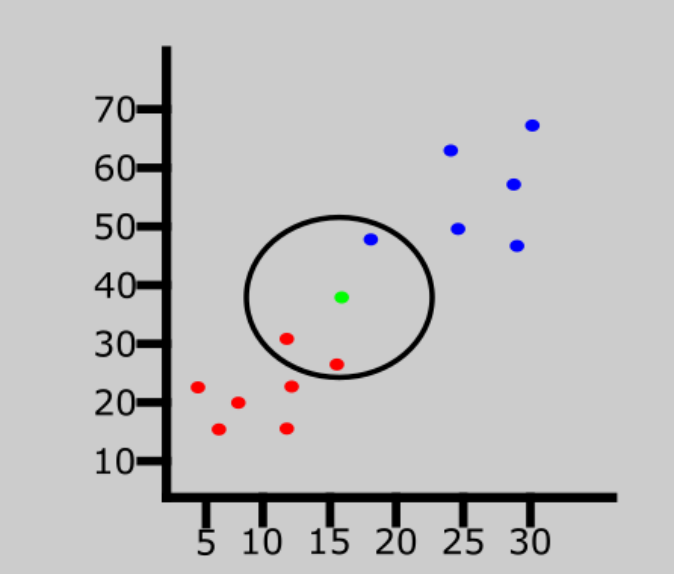

Figure 2 - KNN Classification example

2.1.3 Decision Tree Classifier

A decision tree is a supervised learning algorithm for classification and regression. It has a hierarchical tree structure consisting of a root node, branches, internal nodes and leaf nodes. It divides the dataset in tree like structure and it starts with a root node and ends with a decision made by leaves. It works by recursively partitioning the input data into regions and assigning a class label to each region based on the majority class of the training examples that fall into that region. In figure 3, it shows basic partitioning of dataset in tree like structure. (Chauhan, 2022)

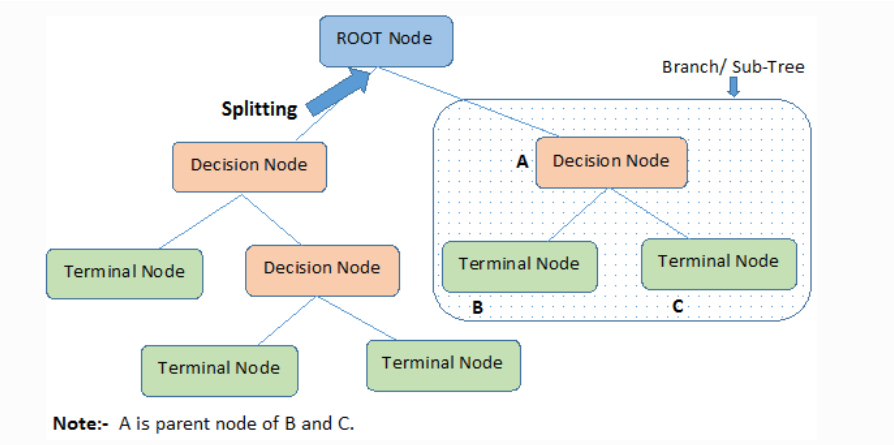

Figure 3 - Decision Tree working principle

In a decision tree, the algorithm begins at the root node and compares the values of attributes with the dataset's records, proceeding down the tree based on comparisons until it reaches a leaf node. The working principle involves:

1. Initialization: Start with the root node containing the entire dataset.
2. Attribute Selection: Choose the best attribute using an **Attribute Selection Measure (ASM)** like Information Gain or Gini Index.
3. Partitioning: Split the dataset into subsets based on the selected attribute.
4. Node Creation: Generate decision tree nodes representing the chosen attribute.
5. Recursion: Repeat the process recursively for each subset until reaching leaf nodes.

The primary challenge in decision tree construction lies in selecting the optimal attribute for both the root node and subsequent sub-nodes. To address this, the Attribute Selection Measure (ASM) technique offers two prominent methods: Information Gain and Gini Index.

Information Gain: Information gain measures the change in entropy after dataset segmentation based on an attribute. It quantifies how much information a feature provides about a class and is calculated as:

$$\text{Information Gain} = \text{Entropy(S)} - (\text{Weighted Avg}) \times \text{Entropy (each feature)}$$

Entropy: Entropy measures the impurity in an attribute and can be calculated as:

$$\text{Entropy(s)} = -P(\text{yes})\log_2 P(\text{yes}) - P(\text{no})\log_2 P(\text{no})$$

Where, S = Total number of samples

P(yes) = probability of yes

P(no) = probability of no

Gini Index: Gini index quantifies impurity or purity and is used in creating decision trees. It calculates the probability of misclassifying a randomly chosen sample and is given by:

$$\text{Gini Index} = 1 - \sum_j P_j^2 \quad , \quad \text{where } P_j \text{ is the probability of class } j$$

Pruning: Pruning is the process of removing unnecessary nodes from the tree to obtain the optimal decision tree. It prevents overfitting and enhances the tree's generalization ability. Cost Complexity Pruning and Reduced Error Pruning are common pruning techniques employed to achieve an optimal decision tree. (Java Point, u.d.)

2.1.4 Random Forest Classifier

The random forest classifier is widely used ensemble learning method for classification problem. Ensemble learning means a collection of models is used to make predictions rather than an individual model. The idea behind the random forest is building multiple decision trees on various subsets of given dataset and combining their predictions to obtain a final classification through majority voting. (R, 2024)

Given that a random forest is composed of multiple decision trees, its operational mechanism is briefly explained in Chapter 2.1.3.

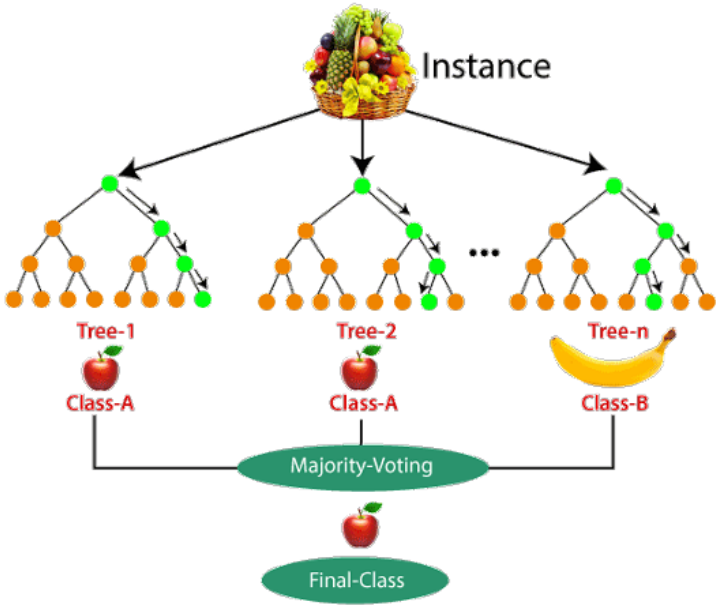

Figure 4 - Random Forest working principle

2.2 Feature Engineering

Feature engineering is the process of selecting, manipulating and transforming raw data into informative features that enhance the model's ability to make accurate classifications. In our dataset, specific tasks such as scaling and applying the PCA technique are deemed necessary. However, we won't apply scaling and PCA to random forest and decision trees since their hierarchical nature makes them less sensitive, potentially affecting their performance.

2.2.1 Scaling Technique

Feature scaling is a data preprocessing technique used to transform the values of features or variables in a dataset to a similar scale. It ensures all features contribute equally to the model and prevents domination by features with larger values. It's necessary when dealing with features of different ranges, units, or magnitudes to avoid biased model performance. Common techniques include standardization, normalization, and min-max scaling, preserving relative relationships and distributions. Scaling facilitates accurate modelling, improves convergence, and prevents features from overshadowing others. (Bhandari, 2024)

In our data preprocessing, we employ standardization, involving subtracting the mean and dividing by the standard deviation for each feature. This centers the data around zero with a standard deviation of 1.

$$X_{scaled} = \frac{X - \mu}{\sigma}$$

2.2.2 PCA Technique

Dimensionality reduction is a crucial technique in data analysis and machine learning, especially when dealing with high-dimensional datasets. As datasets become larger and more complex, the sheer number of features or dimensions can overwhelm computational resources, increase the risk of overfitting, and make models harder to interpret. One prominent technique for dimensionality reduction is Principal Component Analysis (PCA).

PCA is widely used in data analysis and machine learning for transforming high-dimensional data into a lower-dimensional representation while retaining the most important information. The motivation behind PCA lies in achieving a representation of the data where the dimensions are independent and widely dispersed. This is achieved by transforming the original data into a new set of variables called principal components, which are linear combinations of the original variables.

The working principle of PCA involves several key steps:

1. **Standardization**: Before applying PCA, it's essential to standardize the data, especially when features are measured in diverse units. Standardization involves subtracting the mean and dividing by the standard deviation for each feature to ensure that all features contribute equally to the analysis.

2. **Compute the Covariance Matrix**: Next, the covariance matrix of the standardized data is computed. The covariance matrix quantifies the relationships between different dimensions of the data.

   Covariance matrix formula, $\frac{1}{n-1}(X - \overline{X})^T(X - \overline{X})$

3. **Calculate Eigenvectors and Eigenvalues**: The eigenvectors and eigenvalues of the covariance matrix are then calculated. Eigenvectors represent the directions or principal components of the data, while eigenvalues represent the magnitude of variance in those directions.

4. **Sort Eigenvalues**: The eigenvalues are sorted in descending order. The eigenvectors corresponding to the highest eigenvalues capture the most variance in the data and are chosen as the principal components.
5. **Select Principal Components**: Based on the desired level of explained variance, a subset of the principal components is selected. Typically, enough principal components are retained to explain a significant portion of the total variance in the data.
6. **Transform the Data:** Finally, the original data is transformed into the new lower-dimensional space defined by the selected principal components. This transformation allows for a more concise representation of the data while preserving as much meaningful information as possible.

Our goal is to ensure that the data exhibits high variance across its dimensions and eliminates correlated dimensions, indicating linear independence. We achieve this by transforming the data such that its covariance matrix resembles a diagonal matrix, where significant values lie along the main diagonal and zero values occupy the off-diagonal elements. This transformation, known as diagonalization, is the fundamental motivation behind Principal Component Analysis (PCA). (Baruah, 2023)

2.3 Performance Metrics

When building a classification system, we need a way to evaluate the performance of the classifier. And we want to have evaluation metrics that are reflective of the classifier's true performance. The most commonly used Performance metrics for classification problem are as follows,

- Accuracy
- Confusion Matrix
- Precision, Recall, and F1 score
- ROC AUC
- Log-loss

Accuracy: Accuracy measures the proportion of correctly classified instances out of the total number of instances. It's calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Confusion Matrix: A confusion matrix provides a tabular representation of the model's performance. It shows the number of true positives, true negatives, false positives, and false negatives, allowing for a more in-depth analysis of the model's behaviour across different classes.

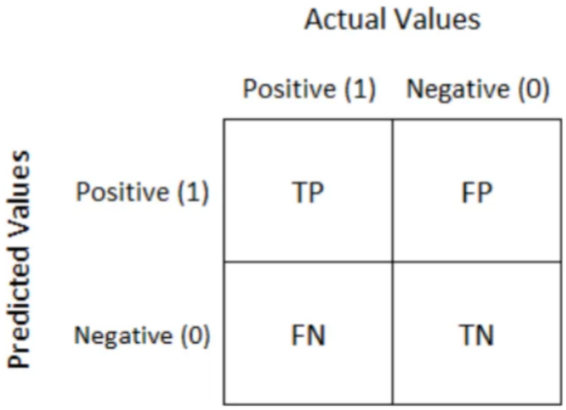

Figure 5 - Confusion Matrix

Precision: Precision quantifies the proportion of true positive predictions among all positive predictions made by the model. It's particularly useful when the cost of false positives is high. Precision is calculated as:

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall: Recall measures the proportion of true positive predictions among all actual positive instances in the dataset. It's particularly important when the cost of false negatives is high. Recall is calculated as:

$$\text{Recall} = \frac{TP}{TP+FN}$$

F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. It's calculated as:

$$F1 = 2 \times \frac{\text{Precision x Recall}}{\text{Precision}+\text{Recall}}$$

ROC Curve and AUC: Receiver Operating Characteristic (ROC) curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The Area Under the Curve (AUC) summarizes the ROC curve, providing a single scalar value representing the model's ability to distinguish between classes. (Vidiyala, 2020)

2.4 Streamlit Application

Streamlit stands out as a remarkable open-source Python library, empowering users to effortlessly craft captivating and customizable web applications tailored for machine learning and data science endeavours. Its appeal lies in its user-friendly interface and seamless integration with Python, rendering the process of transforming Python scripts into dynamic web apps a breeze, without requiring any proficiency in JavaScript or HTML.

Streamlit revolutionizes the landscape of web app development by placing a paramount emphasis on data and machine learning, while providing a frictionless avenue for sharing insights through immersive web applications. From exploring weather datasets to constructing complex machine learning models, Streamlit empowers users to unleash the full potential of their data-driven projects with unparalleled ease and efficiency. (Codes, 2023)

3 Methodology

To tackle our classification problem using machine learning algorithms, Scikit-learn emerges as the indispensable tool. Known as sklearn, Scikit-learn stands as a prominent open-source library for machine learning in Python. It offers straightforward and effective utilities for both data mining and analysis, leveraging the robustness of Python's core libraries like NumPy, SciPy, and Matplotlib. Scikit-learn boasts a diverse array of machine learning algorithms catering to various tasks such as classification, regression, clustering, dimensionality reduction, and model selection.

Overall, scikit-learn is a versatile and user-friendly library that is widely used by data scientists, researchers, and developers. It serves as a go-to solution for implementing machine learning algorithms, prototyping models, and crafting predictive analytics solutions. With its blend of simplicity, efficiency, and comprehensive documentation, Scikit-learn stands as an indispensable asset within the Python ecosystem for machine learning endeavors.

3.1 Import Data

In scikit-learn, the MNIST dataset is available through the fetch_openml function, which allows users to fetch datasets from the OpenML repository. The MNIST dataset is one of the datasets hosted on OpenML. When you call fetch_openml('mnist_784'), scikit-learn downloads the MNIST dataset directly from the OpenML repository and loads it into your Python environment. In figure 6 shows code snippet to load the MNIST dataset using scikit-learn:

```
# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1, parser='auto', cache=True, as_frame=False)

X = mnist["data"]
y = mnist["target"].astype(np.uint8)
```

Figure 6 - Import Data

3.2 Splitting the Data

After importing the data, the next crucial step involves splitting it into train, validation, and test sets. This segmentation enables us to methodically train, assess, and refine machine learning models, ensuring their effectiveness and reliability on new data.

The training set plays a pivotal role in model development, as it exposes the model to a subset of the data, allowing it to discern underlying patterns and relationships between features and target variables. Subsequently, the validation set assumes its significance in evaluating the trained model's performance. By scrutinizing how well the model generalizes to unseen data, the validation set aids in fine-tuning hyperparameters, thus enhancing model efficacy.

Finally, the test set serves as the ultimate arbiter of the model's proficiency. By offering an unbiased estimate of performance on entirely unseen data, the test set validates the model's ability to generalize, free from any influence stemming from model training or hyperparameter tuning. In figure 7 shows the code snippet.

```
# Split the data into training, validation and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=10000, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=10000, random_state=42)
```

Figure 7- Split Data

In splitting the dataset, I allocated 50,000 records for the training set, 10,000 records for the validation set, and another 10,000 records for the test set. By setting the random state to 42, I ensured that the data splitting process remains consistent across multiple runs of the code. This prevents any shuffling of the data in subsequent executions, maintaining the integrity of the dataset partitioning.

3.3 Data Preprocessing

Scaling and PCA (Principal Component Analysis) are crucial preprocessing techniques applied to feature data (X) to prepare it for machine learning algorithms, while target data (y) remains unchanged as it represents the labels or outcomes of interest. Scaling methods, such as standardization or min-max scaling, ensure that features are brought onto a similar scale, promoting uniformity in their influence on model training. In Chapter 2.2.1, I elaborated briefly on scaling techniques, emphasizing the usage of the StandardScaler() function from scikit-learn. Here, the fit_transform() method is employed to compute the mean and standard deviation of the training data (X_train) and subsequently standardize it, combining fitting and transformation into a single step. On the other hand, only the transform() method is utilized on the validation and test sets (X_val and X_test) to ensure that they are scaled consistently based on the statistics learned from the training set (X_train).

PCA serves as a dimensionality reduction technique, converting high-dimensional feature data into a lower-dimensional representation while retaining as much variance as possible. As explained briefly in Chapter 2.2.2, the PCA(n_components=0.95) function from scikit-learn is employed for this purpose, selecting the minimum number of principal components that explain 95% of the variance in the data. Here, the fit_transform() method is utilized to fit the PCA model to the training data (X_train) and

transform it into its principal components. Similar to scaling, the transform() method is applied exclusively to the validation and test sets (X_val and X_test) to ensure that they are transformed consistently using the principal components derived from the training set (X_train). This approach maintains consistency in dimensionality reduction across different datasets. Figure 8 illustrates the code snippet showcasing the implementation of scaling and PCA techniques.

```
In [8]: # Standardizing the data.
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_val_scaled = scaler.transform(X_val)  # Only transforming the validation data.
        X_test_scaled = scaler.transform(X_test)  # Only transforming the test data.
```

```
In [9]: # Apply PCA
        pca = PCA(n_components=0.95)  # Retain 95% of variance
        X_train_pca = pca.fit_transform(X_train_scaled)
        X_val_pca = pca.transform(X_val_scaled)
        X_test_pca = pca.transform(X_test_scaled)
```

Figure 8- Scaling and PCA

3.4 Define Parameter Grid and Grid Search CV

Parameter grids are defined for hyperparameter tuning across various machine learning classifiers, with each grid specifying a range of hyperparameters to be explored during the tuning process. Considering computational constraints, I opted not to create extensive lists of values when defining the parameter grid to avoid lengthy processing times.

In logistic regression, the parameter 'C' represents the inverse of regularization strength. For decision trees, hyperparameters control tree depth, the minimum number of samples for node splitting, and the minimum number of samples at leaf nodes. The grid encompasses various combinations of these hyperparameters, including 'None' for unlimited depth. Similarly, for random forests, parameters dictate the number of trees, maximum tree depth, and minimum samples for node splitting, with multiple combinations explored. In the case of k-nearest neighbors, parameters determine the number of neighbors considered for predictions and the weight function used, with options including 'uniform' and 'distance'.

These parameter grids serve as a roadmap for exploring each classifier's hyperparameter space, facilitating the selection of optimal parameters through techniques like grid search. Figure 9 illustrates the code snippet shows defining the parameter grid.

```
In [10]: # Define parameter grids for hyperparameter tuning
param_grid_lr = {
    'classifier__C': [0.01, 0.1, 1]
}

param_grid_dt = {
    'classifier__max_depth': [None, 10, 20, 30],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4]
}

param_grid_rf = {
    'classifier__n_estimators': [50, 100, 150],
    'classifier__max_depth': [None, 5, 10],
    'classifier__min_samples_split': [2, 5, 7]
}

param_grid_knn = {
    'classifier__n_neighbors': [3, 5, 7],
    'classifier__weights': ['uniform', 'distance']
}
```

Figure 9 - Parameter Grid

The GridSearchCV function provided by scikit-learn is utilized to perform hyperparameter tuning for machine learning models by exhaustively searching through a specified grid of hyperparameters. This technique allows us to find the optimal combination of hyperparameters that results in the best model performance. Pipelines are created to encapsulate all the classifiers, the working principles of which are thoroughly explained in Chapter 2.1. For each model, hyperparameter grids are defined, with the cv parameter determining the number of folds for cross-validation and the scoring parameter indicating the performance metric used for comparison and selection of the best hyperparameter combination. Figure 10 illustrates the code snippet shows applying GridSearchCV.

```
In [12]: # Create GridSearchCV instances for each model
grid_search_lr = GridSearchCV(pipe_lr, param_grid_lr, cv=5, scoring='accuracy')
grid_search_dt = GridSearchCV(pipe_dt, param_grid_dt, cv=5, scoring='accuracy')
grid_search_rf = GridSearchCV(pipe_rf, param_grid_rf, cv=5, scoring='accuracy')
grid_search_knn = GridSearchCV(pipe_knn, param_grid_knn, cv=5, scoring='accuracy')
```

Figure 10 - Applying GridSearchCV

3.5 Model Fitting

In training the machine learning models, the .fit method serves as a fundamental tool within scikit-learn. With GridSearchCV objects, this method not only trains the model but also orchestrates the hyperparameter tuning procedure by fitting the model with various hyperparameter combinations and cross-validating them. Following the completion of tuning, the optimal hyperparameters are determined based on the designated scoring metric. Once trained, the GridSearchCV object retains crucial information regarding the best hyperparameters, facilitating subsequent analysis or prediction tasks. Notably, during this process, the scaled and PCA-transformed data were exclusively fitted to the logistic regression and KNN models, as applying them to the decision tree and random forest models could potentially impact their performance. Figure 11 and 12 illustrates the code snippet shows model fitting and best parameter respectively.

```
In [13]: # Fit GridSearchCV to training data
         grid_search_lr.fit(X_train_pca, y_train)
         grid_search_dt.fit(X_train, y_train)
         grid_search_rf.fit(X_train, y_train)
         grid_search_knn.fit(X_train_pca, y_train)
```

Out[13]:

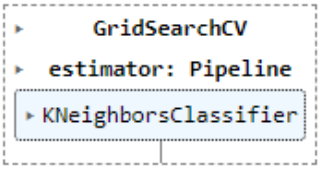

Figure 11- fitting the model

```
In [16]: # Print best parameter
         print("Best Parameter - Logistic Regression:",best_lr)
         print("Best Parameter  - Decision Tree:", best_dt)
         print("Best Parameter  - Random Forest:", best_rf)
         print("Best Parameter  - KNN:", best_knn )

         Best Parameter - Logistic Regression: {'classifier__C': 0.1}
         Best Parameter  - Decision Tree: {'classifier__max_depth': 20, 'classifier__min_samples_leaf': 1, 'classifier__min_samples_spli
         t': 2}
         Best Parameter  - Random Forest: {'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimator
         s': 150}
         Best Parameter  - KNN: {'classifier__n_neighbors': 3, 'classifier__weights': 'distance'}
```

Figure 12- Best Parameter

3.6 Predict and Evaluate on Validation set

In the subsequent steps, predictions are made on the validation set (X_val) using the machine learning models trained via GridSearchCV. The .predict method, a common tool in scikit-learn, is utilized to predict the target variable. Given that our dataset is not imbalanced, accuracy serves as a suitable performance metric, as discussed briefly in Chapter 2.3. The accuracy_score function from scikit-learn is then employed to compare predicted labels with actual labels from the validation set, providing a measure of the proportion of correctly classified instances. Based on the validation accuracy scores, it is evident that the random forest classifier achieved the highest performance, indicating that it should be selected to proceed with the test set. Figure 13 illustrates the code snippet shows prediction and evaluation of trained models.

```
In [17]: # Prediction on X validation set
         y_pred_gs_lr = grid_search_lr.predict(X_val_pca)
         y_pred_gs_dt = grid_search_dt.predict(X_val)
         y_pred_gs_rf = grid_search_rf.predict(X_val)
         y_pred_gs_knn = grid_search_knn.predict(X_val_pca)

In [18]: # Evaluate models on validation set
         val_accuracy_lr = accuracy_score(y_val, y_pred_gs_lr)
         val_accuracy_dt = accuracy_score(y_val, y_pred_gs_dt)
         val_accuracy_rf = accuracy_score(y_val, y_pred_gs_rf)
         val_accuracy_knn = accuracy_score(y_val, y_pred_gs_knn)

In [19]: # Print validation accuracies
         print("Validation Accuracy - Logistic Regression:", val_accuracy_lr)
         print("Validation Accuracy - Decision Tree:", val_accuracy_dt)
         print("Validation Accuracy - Random Forest:", val_accuracy_rf)
         print("Validation Accuracy - KNN:", val_accuracy_knn)

         Validation Accuracy - Logistic Regression: 0.9212
         Validation Accuracy - Decision Tree: 0.871
         Validation Accuracy - Random Forest: 0.972
         Validation Accuracy - KNN: 0.9517
```

Figure 13- Predict and Evaluate

3.7 Testing

After training the dataset and evaluating the validation set, the final stage involves assessing the model's performance on an unseen test set. This critical step ensures a comprehensive evaluation of the model's generalization capabilities to new, unseen data. In this process, predictions are made on the test set using the best-performing model which is random forest classifier, identified during validation, and its accuracy on the test set is computed. The obtained test accuracy provides valuable insights into the model's effectiveness in making predictions on real-world data, ultimately informing decisions about its deployment or further refinement. Figure 14 illustrates the code snippet shows evaluation of new unseen data(X_test) is 0.9649.

```
In [21]: # Evaluate the best model on the test set
         y_pred_test = grid_search_rf.predict(X_test)
         test_accuracy = accuracy_score(y_test, y_pred_test)
         print("Test Accuracy - Random Forest (Selected Model):", test_accuracy)

         Test Accuracy - Random Forest (Selected Model): 0.9649
```

Figure 14 - Evaluation on Testset

3.8 Streamlit Implementation

After evaluating our random forest model on a new, unseen test set, the next step is to deploy the model using a web-based Streamlit application. Streamlit, briefly explained in Chapter 2.4, provides a user-friendly interface for deploying machine learning models. The purpose of this application is to enable users to upload images of handwritten digits and predict the digit using the pre-trained random forest model. Upon uploading, the image is displayed using the st.image function. To ensure consistency with our trained data, the uploaded image undergoes several preprocessing steps. Firstly, it is converted to grayscale to match the color representation of digits in our dataset. Next, the image is inverted to align with the background color used during training. Finally, the image is resized to 28x28 pixels, matching the dimensions of the images in our training data. Subsequently, the image is flattened to 784 features using flatten(), and the reshape() function is utilized to align the input shape expected by the model. This preprocessing process ensures that the uploaded image undergoes the same transformations as the images used during model training, facilitating accurate predictions. Figure 15 illustrates the complete code for implementation of streamlit application.

```
# Main Streamlit app
def main():
    st.title('Welcome to the portal - Recognition of Handwritten digits')
    st.markdown('## Here you can upload your image')
    st.write('The purpose of this portal is to check whether the random forest model predicts the handwritten digits image corre

    # Upload image
    uploaded_image = st.file_uploader("Upload Image", type=["jpg", "jpeg", "png"])

    if uploaded_image is not None:

        # Display uploaded image
        st.image(uploaded_image, caption='Uploaded Image', use_column_width=True)

        # Open the image file
        image = Image.open(uploaded_image)

        # Preprocess the image
        grayscale_image = image.convert('L')  # Convert to grayscale
        inverted_image = ImageOps.invert(grayscale_image) # Inverting the color of digit and background
        resized_image = inverted_image.resize((28, 28))  # Resize image to 28x28 because our input data in model development is 2

        # Convert image to 784 features
        data = np.asarray(resized_image)  # Convert image to numpy array
        features = data.flatten()  # Flatten the array to 1D
        #features = features.reshape(1, -1)  # Reshape to match model input shape

        # Display prediction
        st.subheader("Prediction Interface")
        prediction = rfc.predict([features])[0]

        # Display preprocessed image
        st.subheader("Preprocessed Image")
        st.image(resized_image, caption='Preprocessed Image (Grayscale, 28x28)', use_column_width=True)

        # Display prediction
        st.markdown(f"## Predicted digit: {prediction}")

if __name__ == '__main__':
    main()
```

Figure 15- Streamlit code

4 Result and Discussion

The performance of four different machine learning algorithms was evaluated on the validation dataset, yielding the accuracy scores which shows in table 1. These accuracy scores provide insight into the classification capabilities of each model on unseen data.

Accuracy Scores on Validation set	
Logistic Regression	0.9212
Decision Tree	0.871
Random Forest	0.972
KNN	0.9517

Table 1: Accuracy scores.

Based on the validation results, the Random Forest model demonstrated the highest accuracy among the evaluated models, with an impressive accuracy score of 97.2%. Therefore, the Random Forest model was selected as the best performing model for further evaluation on the test dataset.

Subsequently, the selected Random Forest model was tested on the independent test dataset, where it achieved an accuracy score of 96.49%. This high accuracy score indicates that the Random Forest model generalizes well to unseen data and performs reliably in classifying handwritten digits.

After running the Streamlit application, the uploaded handwritten image is accurately classified by the Random Forest model, demonstrating the effectiveness of our implementation. This successful prediction showcases the capability of our model to correctly identify handwritten digits in real-time applications. This outcome reinforces the reliability of our solution and highlights the practical utility of the Random Forest approach for digit recognition tasks. Figure 16 shows the output snippet of streamlit application.

Figure 16- Streamlit Output

5 Conclusion

In summary, the Random Forest model stands out as the preferred approach for handwritten digit recognition, showcasing exceptional performance in comparison to alternative methodologies. Its remarkable accuracy on both validation and test datasets underscores the reliability and robust generalization capacity of the Random Forest model in effectively classifying handwritten digits. While the Random Forest approach has proven highly effective, there is potential to explore additional methodologies such as neural networks, TensorFlow, and deep learning for addressing the MNIST dataset problem.

6 Teoretiska frågor

1. Kalle delar upp sin data i ”Träning”, ”Validering” och ”Test”, vad används respektive del för?

För att träna maskininlärningsmodellerna använder vi träningsuppsättningsdata. Modellen använder en träningsdatauppsättning för att förstå mönstret och sambanden i datan. Så att den kan göra förutsägelser eller beslut från dem som lär sig utan att explicit programmeras för att utföra en specifik uppgift.

Valideringsdatauppsättningar innehåller olika datauppsättningar och används för att utvärdera tränade ML-modeller. När man arbetar med validering används data för att bedöma modellens prestanda. Vi kan fortfarande styra modellen genom att ställa in parametern eller hyperparametern på valideringsuppsättningen.

Testuppsättningen är återigen en annan uppsättning osynliga data. Testdatauppsättningen speglar verkliga data som maskininlärningsmodellen aldrig har sett förut. Huvudsyftet är att göra en rättvis och slutgiltig bedömning av hur modellen skulle prestera när den möter ny data i en levande operativ miljö.

2. Julia delar upp sin data i träning och test. På träningsdatan så tränar hon tre modeller; ”Linjär Regression”, ”Lasso regression” och en ”Random Forest modell”. Hur skall hon välja vilken av de tre modellerna hon skall fortsätta använda när hon inte skapat ett explicit ”valideringsdataset”?

Istället för att direkt anpassa respektive modell på träningssetet kan vi använda cross validering metod eller K-fold cross validering teknik som tränar modellen och validerar genom att dela upp data som tränings- och valideringsuppsättning för det givna antalet veck och sedan kan beräkna RMSE för varje iterativ veck och slutligen beräkna genomsnittlig RMSE bland alla folds.

På detta sätt kan vi utvärdera alla modeller och välja den lägsta RMSE-modellen att använda mot testdataset.

3. Vad är ”regressionsproblem? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden?

Regressionsproblem är en typ av supervised inlärningsproblem. Målet är att förutsäga en kontinuerlig utfallsvariabel baserat på en eller flera indatafunktioner. Vi kan också säga att regression syftar till att hitta sambandet mellan oberoende variabler(features) och en beroende variabel(target) och göra förutsägelser om target variabeln.

Några av regressionsmodellerna är linjär regression, lasso regression, ridge regression, support vector regression, beslutsträd och random forest regression.

Potentiella applikationsområden är

- prediktiv modelleringsuppgift såsom att förutsäga huspris, aktiekurs och försäljningsprognoser.
- Finansiering såsom kreditvärdering och riskbedömning.
- Marknadsföring och försäljning såsom kundbeteende.

4. Hur kan du tolka RMSE och vad används det till:
   RMSE = $\sqrt{\sum_i (y_i - \hat{y}_i)^2}$

y_i är det faktiska värdet av målvariabeln för den i:te datapunkten

$\hat{y}_i$ är det förutsagda värdet för målvariabeln för den i:te datapunkten.

RMSE står för Root Mean Squared Error och det används ofta prestandamått för att utvärdera regressionsmodellerna. Den mäter den genomsnittliga avvikelsen mellan förutsagda värden och faktiska värden i datamängden.

Lägsta RMSE indikerar bättre modellprestanda, eftersom det betyder att de förutsagda värdena ligger närmare de faktiska värdena i genomsnitt.

5. Vad är ”klassificieringsproblem? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden? Vad är en ”Confusion Matrix”?

Klassificeringsproblem är en typ av supervised inlärningsproblem. Målet är att kategorisera indata i fördefinierade klasser eller kategorier. Target variabeln är kategorisk, vilket innebär att den antar en diskret uppsättning värden eller etiketter.

Några av regressionsmodellerna är Logistic regression, support vector classifier, K-Nearest Neighbors(KNN), beslutsträd och random forest klassificering.

Potentiella applikationsområden är

- Upptäcka skräppost – Klassificera e-postmeddelanden som antingen skräppost eller inte-
- Förutsägelse av kundavgång – Förutsäg huruvida en kund sannolikt kommer att lämna kunder eller inte.
- Bildigenkänning – Klassificera bilder i olika kategorier eller etiketter.

Confusion Matrix

För att utvärdera prestandan hos en klassificeringsmodell används Confusion Matrix. Det är en tabell som gör att du kan visualisera prestandan för en algoritm genom att presentera en sammanfattning av de förutsägelser som modellen gjort jämfört med de faktiska värdena. Från Confusion Matrix tabellen kan vi beräkna Accuracy, Precision, Recall och F1-score. Med hjälp av dessa mätetal och problemets sammanhang kan vi bestämma modellens prestanda.

6. Vad är K-means modellen för något? Ge ett exempel på vad det kan tillämpas på.

Det är en populär unsupervised inlärningsalgoritm som används för att gruppera data i grupper eller kluster baserat på likhet. Algoritmen syftar till att dela upp data i K-kluster där varje datapunkt tillhör klustret med närmaste medelvärde (tyngdpunkt).

Det kan appliceras

- Kundsegmentering – Inom marknadsföring kan K-means-klustring användas för att segmentera kunder baserat på deras köpbeteenden eller andra funktioner.

- Anomalidetektering – K-means-klustring kan användas för att identifiera anomalier eller extremvärden i datauppsättningar.
- Bildkomprimering – K-betyder klustring kan användas för att komprimera bilder genom att minska antalet färger.

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding. Se mappen ”l8” på GitHub om du behöver repetition.

Ordinal encoding används för att koda kategoriska variabler där kategorierna har naturlig ordning eller rangordning. Varje kategori mappas till ett unikt heltalsvärde baserat på dess ordning.

Exempel: Första pris – 0, Andra pris – 1, Tredje pris – 2.

one-hot encoding används för att omvandla kategoriska variabler till binära vektorer. Varje kategorisk värde kan omvandlas till nya binära variabler.

Exempel: "Röd" – [1,0,0]

"Grön"- [0,1,0]

"Blå" – [0,0,1]

Dummy variable encoding liknar one-hot encoding. Om dess n kategoriska värden skapar den n-1 nya binära variabler. En kategori tas bort och återstående kategori representeras av binära funktioner. Det är för att undvika multikollinearitet i regressionsmodeller.

Exempel: "Röd" – [1,0]

"Grön"- [0,1]

"Blå" – [0,0]

8. Göran påstår att datan antingen är ”ordinal” eller ”nominal”. Julia säger att detta måste tolkas. Hon ger ett exempel med att färger såsom {röd, grön, blå} generellt sett inte har någon inbördes ordning (nominal) men om du har en röd skjorta så är du vackrast på festen (ordinal) – vem har rätt?

Julia har rätt. Vi kan inte direkt säga att data är ordinal eller nominal. Den bör tolkas utifrån problemets sammanhang och betydelsen av kategorierna inom den specifika problemdomänen.

9. Kolla följande video om Streamlit: https://www.youtube.com/watch?v=ggDa-RzPP7A&list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&index=12 Och besvara följande fråga: - Vad är Streamlit för något och vad kan det användas till?

Streamlit är ett kraftfullt verktyg som används för att bygga interaktiva webbapplikationer för datavetenskap och maskininlärningsprojekt. Det är ett pythonbibliotek med öppen källkod som tillåter användare att distribuera sin modell i webbapplikationer utan att behöva skriva HTML, CSS eller Javascript.

7 Självutvärdering

1. Utmaningar du haft under arbetet samt hur du hanterat dem.

   Jag hade utmaningar med att skapa parametergrid för hyperparameterjustering och tog också lång tid att lära mig bildkonvertering för den streamlit applikationen. Jag har hanterat med hjälp av google och stack overflow och med lite tålamod.

2. Vilket betyg du anser att du skall ha och varför.
   Jag kan klara mig bekvämt eftersom jag tror att jag uppfyllde alla krav för denna kurs.

3. Något du vill lyfta fram till Antonio?
   Inget verkar vara problem i kursen. Det var fantastiskt att arbeta i den här typen av projekt.

8 List of Figures

9 Bibliography

Abba, I. V. (2023, January 25). *Freecodecamp*. Retrieved from https://www.freecodecamp.org/news/k-nearest-neighbors-algorithm-classifiers-and-model-example/

Baruah, I. D. (2023, October 7). *Medium*. Retrieved from https://medium.com/nerd-for-tech/dimensionality-reduction-techniques-pca-lca-and-svd-f2a56b097f7c

Bhandari, A. (2024, January 4). *Analytics Vidhya*. Retrieved from https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Brownlee, J. (2020, September 1). Retrieved from https://machinelearningmastery.com/multinomial-logistic-regression-with-python/

Chauhan, N. S. (2022, February 9). *KDnuggets*. Retrieved from https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

Codes, F. (2023, September 11). *Medium*. Retrieved from https://medium.com/@fareedcodes/introduction-to-streamlit-build-data-apps-in-python-53e7aa20f9fd

Kanade, V. (2022, April 18). *Spiceworks*. Retrieved from https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/

R, S. E. (2024, January 3). *Analytics Vidhya*. Retrieved from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Ramakrishnan, M. (2022, November 23). *Emeritus*. Retrieved from https://emeritus.org/blog/artificial-intelligence-and-machine-learning-classification-in-machine-learning/

Vidiyala, R. (2020, July 26). *Towards Data Science*. Retrieved from https://towardsdatascience.com/performance-metrics-for-classification-machine-learning-problems-97e7e774a007