

Machine Learning

ACCA6011

Josh Archer
2011907

Contents

1.	Report Overview	2
1.1	Abstract	2
1.2	Introduction	2
2.	Related Work	3
2.1	K nearest neighbours	3
2.2	Support Vector Machines	3
2.3	Random Forest	5
3.	Main Body	7
3.1	Machine Learning Pipeline	7
3.1.1	Data Collection	7
3.1.2	Data Preprocessing	7
3.1.3	Feature Selection	7
3.1.4	Model Selection	7
3.1.5	Model Training	8
3.1.6	Model Evaluation	8
3.1.7	Hyperparameter Tuning	8
3.1.8	Model Deployment	8
3.1.9	Monitoring And Maintenance	8
3.2	Methodology of Comparing different Classifiers	8
4.	Results	9
4.1	K Nearest Neighbour	9
4.2	Decision Tree	11
4.3	Effect of Decisions	12
5.	Conclusion	13
6.	References	14

1. Report Overview

1.1 Abstract

1.2 Introduction

In this assignment, we delve into the evaluation of two machine learning algorithms using the MNIST fashion image dataset, a curated collection of labelled images designed to replace the simplistic MNIST handwritten digit recognition dataset. The central concern lies in the inadequacy of the original MNIST dataset for assessing contemporary supervised classification ML algorithms due to its outdated simplicity. As the field of machine learning advances, it becomes imperative to adapt evaluation datasets to reflect the complexity of modern applications.

The MNIST fashion dataset serves as a challenging alternative, and the objectives of this report are to implement and assess two classifiers specified by the instructor. Leveraging the capabilities of Sci-kit learn and additional Python packages, the assignment mandates the exploration of various parameters to optimize the performance of each machine learning algorithm. Comprehensive evaluation using metrics such as confusion matrices, recall, precision, F1 score, and classification accuracy is required for both classifiers.

The report follows a standard machine learning workflow, encompassing dataset pre-processing (normalization, exploration, etc.), partitioning into training, testing, and evaluation subsets, and subsequent training, tuning, and evaluation of the two models. Furthermore, a literature review, experimental methodology details, results, and a thorough discussion with conclusions will be presented. This investigation aims to provide insights into the efficacy of machine learning algorithms on a challenging dataset, contributing valuable knowledge to the ever-evolving field of supervised classification.

2. Related Work

To be able to properly classify the Image dataset the correct machine learning classifier and model must first be selected. Doing this correctly will ensure that the accuracy of the classifications are as high as possible. Below is a breakdown of the selected models, how they are used, the benefits and disadvantages of these will also be discussed.

2.1 K nearest neighbours

K Nearest Neighbours (KNN) stands out as a straightforward and intuitive machine learning algorithm applicable to both classification and regression tasks. Operating on an instance-based learning approach, this model refrains from learning a global function; instead, it memorizes the training instances. Subsequently, predictions are made by assessing the similarity of new, unseen instances to the memorized training data. Here is a breakdown of how this model classifies datasets:

- **Training Phase:** In the training phase, the training dataset is stored.
- **Prediction Phase (Classification):** For classification tasks, when making predictions for new instances, KNN considers the 'k' nearest neighbours in the training data. The term "nearest" is typically defined by a distance metric, often the Euclidean distance. The algorithm assigns the majority class among these 'k' neighbours to the new instance.
- **Prediction Phase (Regression):** In regression tasks, instead of predicting a class label, KNN computes the average or weighted average of the target values of the 'k' neighbours to predict the target value for the new instance.
- **Hyperparameter 'k':** The choice of the hyperparameter 'k' (representing the number of neighbours to consider) is pivotal. A small 'k' can make the model sensitive to noise, while a large 'k' can reduce sensitivity but might smooth out important patterns.
- **Distance Metric:** The distance metric employed to measure similarity between instances is flexible. Euclidean distance is the most common [1].

This type of model proves especially beneficial in scenarios with spatial locality, often encountered in image datasets. However, it may encounter computational challenges with large datasets, a concern that modern computer systems help alleviate [2].

2.2 Support Vector Machines

Support Vector Machines (SVM) is a powerful and versatile supervised machine learning algorithm used for both classification and regression tasks. It belongs to the family of linear models but is particularly effective in high-dimensional spaces. SVM is known for its ability to find optimal hyperplanes that separate different classes in the input feature space. As before below is a brief explanation on how SVMs are deployed and executed:

- **Objective:** The primary objective of SVM is to find a hyperplane that best separates the data into different classes. The optimal hyperplane is the one that maximizes the margin, which is the distance between the hyperplane and the nearest data point from each class, known as support vectors.
- **Linear Separation:** In its simplest form, SVM assumes a linear separation, meaning it looks for a straight line (or hyperplane in higher dimensions) to separate the classes.
- **Kernel Trick (Non-linear Separation):** SVM can be extended to handle non-linear relationships using the kernel trick. This involves transforming the input features into a higher-dimensional space where a hyperplane can linearly separate the data.
- **Margin:** The margin is a critical concept in SVM. A larger margin indicates a more robust model that is less likely to be affected by noise in the data.
- **C Parameter:** The SVM algorithm includes a regularization parameter 'C' that controls the trade-off between achieving a smooth decision boundary and classifying the training points correctly. A smaller 'C' encourages a larger margin but may allow some misclassifications, while a larger 'C' penalizes misclassifications more heavily.
- **Support Vectors:** Support vectors are the data points that lie closest to the decision boundary. These are the most influential points in determining the optimal hyperplane [3].

With regards to the given problem of having to classify the image dataset, SVMs are powerful classifiers that can work well for image classification. They are particularly effective in high-dimensional spaces, making them suitable for image data. SVMs can handle non-linear decision boundaries and are known for their robust performance [4].

Support Vector Machines (SVM) is a potent supervised machine learning algorithm used for classification and regression tasks. It's part of linear models but excels in high-dimensional spaces. SVM aims to find a hyperplane that maximizes the margin between classes, acting as a robust decision boundary. Key points are below:

- **Objective:** SVM's primary goal is to locate a hyperplane that best separates data classes, maximizing the margin—the distance from the hyperplane to the nearest data point in each class (support vectors).

- **Linear Separation:** In its basic form, SVM assumes linear separation, searching for a straight line or hyperplane to distinguish classes.
- **Kernel Trick (Non-linear Separation):** SVM extends to handle non-linear relationships using the kernel trick, transforming input features into a higher-dimensional space for linear separation.
- **Margin:** A critical SVM concept, the margin signifies a robust model less affected by data noise.
- **C Parameter:** SVM incorporates a regularization parameter 'C,' balancing a smooth decision boundary with correct classification. A smaller 'C' encourages a larger margin but may allow some misclassifications, while a larger 'C' penalizes misclassifications more.
- **Support Vectors:** These are data points closest to the decision boundary, significantly influencing the optimal hyperplane [3].

Regarding image dataset classification, SVMs excel, especially in high-dimensional spaces, making them apt for image data. They handle non-linear decision boundaries and exhibit robust performance [4].

2.3 Random Forest

Random Forest, an ensemble learning algorithm, constructs numerous decision trees during training and outputs the class mode of individual trees for classification. It's a versatile model for both classification and regression tasks. Key points:

Decision Trees: Random Forest builds on decision trees, where each internal node signifies a decision based on an attribute, branches represent decision outcomes, and leaf nodes indicate the final class or numerical value.

Ensemble Learning: It forms an ensemble by creating multiple decision trees during training. Each tree is built by randomly selecting feature and training data subsets, introducing diversity.

Bagging (Bootstrap Aggregating): Training datasets for each tree involve random sampling with replacement from the original dataset, ensuring each tree is trained on a slightly different dataset.

Voting Mechanism: In classification, each tree independently predicts the class, and the final prediction results from a majority vote. In regression, it averages individual tree predictions.

Reducing Overfitting: Randomness in feature selection and data sampling mitigates overfitting, enhancing robustness and reducing sensitivity to training data noise.

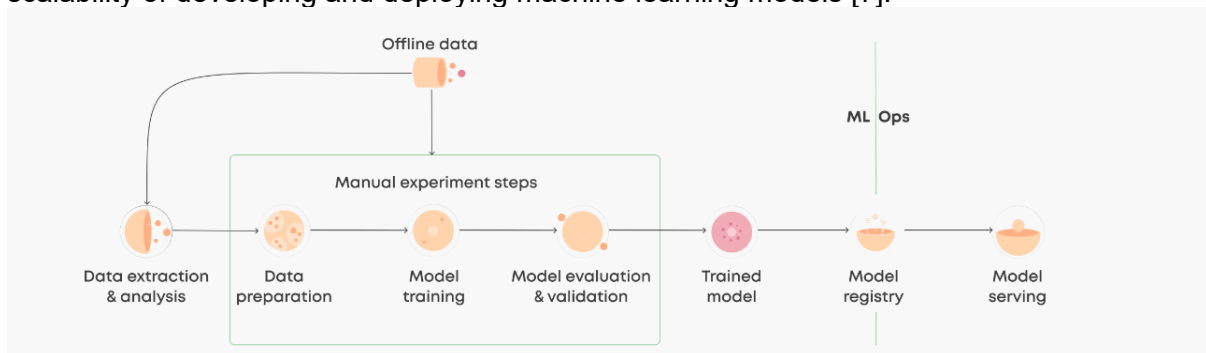
Feature Importance: Random Forests offer feature importance metrics, revealing which features contribute more to the model. This information aids in feature selection and interpretation [5].

Widely used for high accuracy, robustness, and handling high-dimensional data, Random Forests find applications in diverse areas, including image classification, where the ensemble captures intricate patterns [6].

3. Main Body

3.1 Machine Learning Pipeline

A machine learning pipeline encompasses the comprehensive journey of creating, training, and deploying a machine learning model. It comprises a sequence of interlinked steps, seamlessly managing the flow of data and operations. Starting from the initial data preparation, it progresses through various stages, culminating in the deployment of the trained model to generate predictions. The primary objective of a machine learning pipeline is to automate and structure these processes, enhancing the efficiency, reproducibility, and scalability of developing and deploying machine learning models [7].



[7]

The efficiently compose a Machine Learning Pipeline some basic components and stages must be set out. These are stated below:

3.1.1 Data Collection

At this stage the composer should Gather relevant data for their machine learning task. This may involve acquiring datasets, accessing databases, or collecting data through other means.

3.1.2 Data Preprocessing

During this phase, the raw data undergoes cleaning and preprocessing to address issues such as missing values, outliers, and other data quality concerns. Additionally, this stage may encompass feature engineering, involving the creation of new features or the transformation of existing ones to enhance the overall performance of the model.

3.1.3 Feature Selection

At this stage the composer should Choose the most relevant features for your model. Feature selection helps reduce dimensionality and may improve the model's efficiency and interpretability.

3.1.4 Model Selection

The composer here will choose the appropriate machine learning algorithm or model architecture based on the characteristics of your data and the goals of your task.

3.1.5 Model Training

This phase guarantees that the composer trains the chosen model with a labelled dataset. It entails supplying the model with input data and fine-tuning its internal parameters to minimize the disparity between predicted and actual outcomes.

3.1.6 Model Evaluation

At this juncture, the composer ought to evaluate the performance of the trained model using metrics like accuracy, precision, recall, and F1 score. This process aids in comprehending how effectively the model generalizes to novel, unseen data.

3.1.7 Hyperparameter Tuning

This phase necessitates the composer to fine-tune the model's hyperparameters for enhanced performance. This step might encompass employing methods such as grid search or randomized search to identify the optimal combination of hyperparameters.

3.1.8 Model Deployment

At this point the composer should now deploy the trained and tuned model to a production environment where it can make predictions on new data. Deployment may involve creating APIs, deploying on cloud platforms, or integrating the model into existing software systems.

3.1.9 Monitoring And Maintenance

The concluding phase involves the ongoing monitoring of the model's performance in the production environment. Periodically retraining the model with fresh data or updated features is essential to maintain its accuracy and relevance [8] [9].

3.2 Methodology of Comparing different Classifiers

Comparing different classifiers with different settings in machine learning involves a thoughtful methodology to systematically evaluate their performance under various configurations. First the creator of the model must Define the problem. To they must clearly define the classification problem. They must build an understanding of the nature of the data and goals of the classification. Select Evaluation Metrics should be the next stage in the methodology. An appropriate evaluation metric based on the characteristics of the problem at hand. These metrics include accuracy, precision, recall and F1 score [10].

Next for the model creator would be to Split your dataset into training, validation, and test sets. The training set is used for model training, the validation set for hyperparameter tuning, and the test set for final evaluation. The composer should then Choose the classifiers you want to compare, such as k-Nearest Neighbours (KNN) or Decision Trees. Experiment with different hyperparameter settings, such as the number of nearest neighbours, distance metrics (e.g., Euclidean, Manhattan), or tree depth. Define a range of values for each hyperparameter to explore during the evaluation.

The modeller should Apply consistent preprocessing steps across all classifiers. This may include feature scaling, handling missing values, and encoding categorical variables. Ensure that preprocessing steps are applied consistently to maintain fairness in comparisons [11].

The follow should then be considered within the methodology:

- **Cross-Validation:** Implement cross-validation, such as k-fold cross-validation, to obtain robust performance estimates. This step helps mitigate the impact of variations in the dataset splits and provides a more reliable evaluation.
- **Hyperparameter Tuning:** Use the validation set to tune hyperparameters for each classifier. Perform grid search or randomized search to explore different combinations of hyperparameter values. This process helps identify the optimal settings for each classifier.
- **Train and Evaluate Models:** Train each classifier on the training set using the chosen hyperparameters. Evaluate their performance on the validation set and iteratively adjust hyperparameters as needed. Once satisfied, assess the final performance on the test set.
- **Statistical Significance Testing:** Conduct statistical significance testing, if possible, to assess whether observed differences in performance are statistically significant. This step ensures that performance differences are not due to random chance.
- **Documentation and Reporting:** Document the results comprehensively, including the selected metrics, hyperparameter configurations, and any observations or insights gained during the evaluation. Provide clear reports, possibly with visualizations, to facilitate communication of the comparative performance of the classifiers.
- **Iterate and Refine:** If necessary, iterate and refine your methodology. Experiment with additional classifiers or hyperparameter settings to gain a deeper understanding of how different configurations impact model performance.

By following this methodology, you can systematically compare different classifiers with varying settings, facilitating informed decision-making on the selection of models and configurations for your specific machine learning task [12].

4. Results

4.1 K Nearest Neighbour

Upon executing the code with a K Nearest Neighbour classifier model, the obtained results are presented below. To comprehend these results, it is crucial to elucidate the key findings. The classification report furnishes a comprehensive breakdown of the model's performance for each class, encompassing precision, recall, and F1-score, along with support, indicating the number of true instances for each class. These metrics are delineated as follows:

- **Precision:** This metric quantifies the accuracy of positive predictions, calculated as the number of true positives divided by the sum of true positives and false positives.
- **Recall:** Also known as sensitivity or true positive rate, recall gauges the ability of the classifier to capture all positive instances. It is computed as the number of true positives divided by the sum of true positives and false negatives.
- **F1-score:** This metric, representing the harmonic mean of precision and recall, offers a balanced measure that considers both false positives and false negatives.

The results for the model are as follows:

Accuracy: 0.8559

Classification Report:

	precision	recall	f1-score	support
0	0.75	0.87	0.81	1394
1	0.99	0.97	0.98	1402
2	0.72	0.80	0.76	1407
3	0.91	0.84	0.87	1449
4	0.79	0.76	0.77	1357
5	1.00	0.84	0.91	1449
6	0.69	0.60	0.64	1407
7	0.88	0.95	0.92	1359
8	0.98	0.95	0.96	1342
9	0.89	0.96	0.93	1434
accuracy		0.86		14000
macro avg	0.86	0.86	0.86	14000
weighted avg	0.86	0.86	0.86	14000

These can be interpreted as:

Accuracy: 0.8559

The overall accuracy of the model on the test set is approximately 85.59%. This means that the model correctly classified about 85.59% of the instances in the test set.

Detailed below is an interpretation of the results for each class:

Class 0 (Label 0):

Precision: 0.75

Recall: 0.87

F1-score: 0.81

Support: 1394

Class 1 (Label 1):

Precision: 0.99

Recall: 0.97

F1-score: 0.98

Support: 1402

In summary, the classification report gives you a detailed understanding of how well the model performs on each individual class and provides an overall assessment through macro and weighted averages. The high precision, recall, and F1-scores indicate good performance, but it's essential to consider the specific requirements of your application when interpreting these metrics.

4.2 Decision Tree

Accuracy: 0.7940

Classification Report:

	precision	recall	f1-score	support
0	0.74	0.76	0.75	1394
1	0.94	0.95	0.95	1402
2	0.69	0.68	0.68	1407
3	0.80	0.78	0.79	1449
4	0.66	0.67	0.67	1357
5	0.89	0.89	0.89	1449
6	0.54	0.53	0.54	1407
7	0.87	0.88	0.88	1359
8	0.91	0.90	0.90	1342
9	0.90	0.90	0.90	1434
accuracy		0.79		14000
macro avg	0.79	0.79	0.79	14000
weighted avg	0.79	0.79	0.79	14000

These results can be interpreted as Class 0 (Label 0):

Precision: 0.74

Recall: 0.76

F1-score: 0.75

Support: 1394

Class 1 (Label 1):

Precision: 0.94

Recall: 0.95

F1-score: 0.95

Support: 1402

4.3 Effect of Decisions

The decisions you make regarding hyperparameters and other settings in machine learning models can have a significant impact on the overall performance of the code. Let's discuss the effects of some specific decisions you might make in the context of the code provided, focusing on the number of nearest neighbours for K-Nearest Neighbours, the distance metric used, and the tree depth for Decision Trees.

For example, the choice of the number of nearest neighbours (K) can impact the model's performance. Smaller values of K might lead to a more flexible model, but it can be sensitive to noise and outliers in the data.

Larger values of K might provide a smoother decision boundary but might struggle with capturing local patterns in the data.

Another example would be the tree depth. The maximum depth of a decision tree influences the complexity of the tree and its ability to capture intricate patterns in the data. Shallow trees might lead to underfitting, where the model is too simple and cannot capture the complexity of the data.

Deep trees might lead to overfitting, where the model memorizes the training data but struggles to generalize to new, unseen data.

5. Conclusion

In conclusion, this report provided insights into three different machine learning algorithms: K Nearest Neighbours (KNN), Support Vector Machines (SVM), and Random Forest. Each algorithm was explained in detail, highlighting their key concepts, objectives, and applications. The report focused on the KNN algorithm's training and prediction phases, the SVM algorithm's optimization of hyperplanes, and the Random Forest's ensemble learning approach.

The practical implementation of the KNN algorithm for image classification was presented, emphasizing the importance of hyperparameters such as the number of neighbours (k) and the distance metric. The classification report demonstrated the model's effectiveness, with an accuracy of approximately 85.59%.

The Random Forest algorithm, an ensemble of decision trees, was introduced as a versatile and powerful model. The concept of bagging, the voting mechanism, and the reduction of overfitting through randomness were explained. Random Forests were praised for their high accuracy, robustness, and feature importance analysis.

The machine learning pipeline, comprising stages from data collection to model deployment and maintenance, was outlined as a systematic approach to building and deploying machine learning models. The methodology for comparing different classifiers was discussed, emphasizing the importance of defining the problem, selecting appropriate evaluation metrics, and conducting rigorous testing.

Finally, the results of applying the KNN and Decision Tree classifiers to the image dataset were presented. The classification reports provided detailed insights into the performance of each model across different classes. The impact of decisions, such as the number of nearest neighbours and tree depth, on the overall performance was discussed.

6. References

- [1] S. J. D. Pádraig Cunningham, “k-Nearest Neighbour Classifiers - A Tutorial,” Association for Computing Machinery, New York, 2021.
- [2] G. B. Diego Furtado Silva, “How k-Nearest Neighbor Parameters Affect its,” University of Sao Paulo, Sao Paulo, 2009.
- [3] R. K. Mariette Awad, *Efficient Learning Machines*, Berkeley: Apress Berkeley, 2015.
- [4] Jair Cervantes, *Neurocomputing*, Elsevier, 2019.
- [5] A. Bosch, “Image Classification using Random Forests and Ferns,” International Conference on Computer Vision, Rio de Janeiro, 2007.
- [6] Y. Yile Ao, “The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling,” *Journal of Petroleum Science and Engineering*, vol. 174, pp. 776-789, 2019.
- [7] Valohai, “What Is a Machine Learning Pipeline?,” Valohai, 2022.
- [8] IBM, “What is a machine learning pipeline?,” IBM, 2022. [Online]. Available: <https://www.ibm.com/topics/machine-learning-pipeline>. [Accessed 04 01 2024].
- [9] C. N. Hannes Hapke, “Overview of Steps in a Machine Learning Pipeline,” in *Building Machine Learning Pipelines*, Oreilly, 2020, pp. 3-7.
- [10] Anita Rácz, “Multi-Level Comparison of Machine Learning Classifiers and Their Performance Metrics,” Medicinal Chemical Research Group, 2020.
- [11] Nisha P. Shetty, “Comparison study of machine learning classifiers,” Department of Information and Communication Technology, 2020.
- [12] Diego Raphael Amancio, “A Systematic Comparison of Supervised Classifiers,” Shanghai Jiaotong, 2014.

7. Appendix

7.1 Decision Tree Python Code

```
main.py × K Nearest Neighbor.py × Decision Tree.py ×
1 # Import necessary libraries
2 from sklearn.model_selection import train_test_split
3 from sklearn.tree import DecisionTreeClassifier
4 from sklearn.metrics import accuracy_score, classification_report
5 from sklearn.datasets import fetch_openml
6
7 # Load the MNIST Fashion dataset using scikit-learn's built-in datasets
8 fashion_mnist = fetch_openml(name="Fashion-MNIST", as_frame=False)
9
10 # Split the dataset into features (X) and labels (y)
11 X = fashion_mnist.data.astype('float32') / 255.0 # Normalize pixel values to [0, 1]
12 y = fashion_mnist.target.astype('int')
13
14 # Split the dataset into training and testing sets
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
16
17 # Initialize the Decision Tree classifier
18 decision_tree_classifier = DecisionTreeClassifier(random_state=42)
19 # Train the classifier
20 decision_tree_classifier.fit(X_train, y_train)
21 # Make predictions on the test set
22 y_pred = decision_tree_classifier.predict(X_test)
23
24 # Evaluate the classifier
25 accuracy = accuracy_score(y_test, y_pred)
26 print(f"Accuracy: {accuracy:.4f}")
27
28 # Print classification report
29 print("Classification Report:")
30 print(classification_report(y_test, y_pred))
31
```

PyCharm 2021.1.3 available
Update...

7.2 K nearest Neighbour

```
main.py x K Nearest Neighbor.py x Decision Tree.py x
1 # Import necessary libraries
2 from sklearn.model_selection import train_test_split
3 from sklearn.neighbors import KNeighborsClassifier
4 from sklearn.metrics import accuracy_score, classification_report
5 from sklearn.datasets import fetch_openml
6 import numpy as np
7 # Load the MNIST Fashion dataset using scikit-learn's built-in datasets
8 fashion_mnist = fetch_openml(name="Fashion-MNIST", as_frame=False, parser='auto')
9 # Split the dataset into features (X) and labels (y)
10 X = fashion_mnist.data.astype('float32') / 255.0 # Normalize pixel values to [0, 1]
11 y = fashion_mnist.target.astype('int')
12 # Split the dataset into training and testing sets
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
14
15 # Initialize the KNN classifier
16 knn_classifier = KNeighborsClassifier(n_neighbors=3) # You can adjust the number of neighbors as needed
17
18 # Train the classifier
19 knn_classifier.fit(X_train, y_train)
20
21 # Make predictions on the test set
22 y_pred = knn_classifier.predict(X_test)
23
24 # Evaluate the classifier
25 accuracy = accuracy_score(y_test, y_pred)
26 print(f"Accuracy: {accuracy:.4f}")
27
28 # Print classification report
29 print("Classification Report:")
30 print(classification_report(y_test, y_pred))
31
```

PyCharm 2021.1.3 available
[Update...](#)