

MNIST: Fashion Dataset Classifier

1st Given Edward Patch

Software Engineering and Artificial Intelligence (of MSc Year 4)

Artificial Intelligence

University of Wales Trinity St. Davids (of Dr. Gordan Dickers)

Swansea, Wales

Student ID: 1801492

Abstract—The research paper investigates two different models, k -Nearest Neighbours and Decision-Tree, to test different out-fitting and performance options. When comparing the results of the models, a Confusion Matrix and Classification Matrix Report measure the models' accuracy, precision and performance. The Jupyter Notebook (Python) tools, ML libraries, and the Fashion MNIST dataset aid the experiments in finding reliable results.

Index Terms—Machine Learning, Models, Fashion MNIST.

I. INTRODUCTION

This research paper provides insight into different supervised models featuring k -Nearest Neighbors (k NN) and Decision-Tree to sort untrained data into trained data. Inclusion of Detailing of the models, performance and accuracy test.

To test the supervised model's tools available to train and test data to the Fashion MNIST dataset using Comma Separated Values (CSV) that holds the datasets to aid the investigations. Han Xiao, Kashif Rasul and Roland Vollgraf [1], accessed from: <https://github.com/zalandoresearch/fashion-mnist> provide the training and test the Fashion MNIST dataset. The dataset contains sixty thousand of twenty-eight by twenty-eight greyscale images, meaning it is crucial to prepare the data in the best way possible to prevent performance and accuracy problems.

k NN is a non-parametric, lazy learning method for classification and regression. It is a type of instance-based learning where the model only stores the training data and does not learn a specific function to model it.

A Decision-Tree handles the data in a tree structure, where an algorithm would take advantage of the flow control to make its decisions. It breaks down a dataset into smaller subsets stopping at the designated depth while, at the same time, an associated Decision-Tree develops with an incrementation process. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches.

Preparation of the data to understand how to visualise, correlate and separate the data is crucial before any experimentation of the two mentioned models begins. The experiment should run the two models with the optimal parameters (after some testing), processing the Fashion MNIST dataset. The results of Confusion Matrices and Confusion Matrix Reports comprehend the contrast between the two models with two different model hyperparameter setups, allowing a discussion to understand which model suits the experiment better.

Furthermore, it is essential to mention that the Confusion Matrix can suggest any problems with parsing, preparing and executing the data, as it shows overfitting and underfitting to suggest incorrect handling of the data.

II. RELATED WORK IN LITERATURE

A. ML and Traversal Algorithms

Experience with past ML algorithms that support the research of MNIST includes Search Algorithms and Clustering Algorithms contained within the research paper, 'Artificial Intelligence and Machine Learning: Search and ML Clustering' [2]. Experience within the Search and Clustering algorithmics assists with understanding using ML algorithms to examine, sort and apply Fashion MNIST data within various scenarios.

Depth First Search (DFS) and Breadth First Search (BFS) traversal algorithms focus on searching for a node through a tree-node structure. From the 'Artificial Intelligence and Machine Learning: Search and ML Clustering' [2] research paper, a focus point is the performance of the two algorithms, as the DFS uses Stack memory logic and BFS uses Queue memory logic to perform the search. Vigneshwaran Palanisamy [3] helped establish the equation of time complexity of a non-matrix is $\mathcal{O}(\mathcal{V} + \mathcal{E})$ with $\mathcal{V}$ being the number of vertices and $\mathcal{E}$ being the number of edges. Palanisamy [3] states that a time complexity containing a matrix is $\mathcal{O}(\mathcal{V}^2)$, where $\mathcal{V}$ is the number of vertices. Another finding is that an algorithm similar to Dijkstra's algorithm is measured differently for the time complexity. Even though the base of the algorithm is BFS, the time complexity according to Michael Barbehenn [4] focuses on the weighted vertices rather than just the number of vertices, meaning the time complexity formula is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$. Barbehenn [4] mentions that the equation takes in the vertex-based cost functions and uses a binary heap implementation for the priority queue.

The time complexity of these two ML concepts applies to the Fashion MNIST data structure and logic, as the Fashion MNIST contains a vast amount of data that takes time to process. The prior research experience of acknowledging performance with different ML algorithms supports finding a better solution for Fashion MNIST processing. Moreover, the experience of knowing how to differentiate the time complexity and seeing how the equation changes to account for the weighted pathways rather than just the number of vertices

assists in identifying what type of algorithms would work best for the Fashion MNIST data structure.

Clustering algorithms are essential when it comes to data organisation and grouping. Within the ‘Artificial Intelligence and Machine Learning: Search and ML Clustering’ [2] research paper, the study of unsupervised clustering algorithms focuses on K-Means, including equations and visualisation using MATLAB, testing the principles beforehand on **Jupyter Notebook** and RStudio. K-Means uses the $\mathcal{C}$ data structure in figure II-A, page 2: K as a constant representing the number of clusters, X and Y being the data points. Random placement of the Center of Gravity (CoG) starts the clustering process. The K vector only appends when $||\mathcal{C}_i||$ is less than $||\mathcal{C}_i|| * \frac{1}{2}$. The final step is to capture the relation of $\mathcal{C}_i$ assigning the value to the cluster group. The process repeats until there is no change.

$$\mathcal{C}_n = | \mathcal{K} \quad \mathcal{X} \quad \mathcal{Y} |$$

$$\mathcal{K}(n < \overline{\mathcal{C}_i} \frac{1}{2}) = | \mathcal{X} \quad \mathcal{Y} |$$

Fig. 1. K-Means Equation Representation [2]

Experience and knowledge gained from visually applying K-Means demonstrate the skillset that can apply to k -Nearest Neighbour (k NN) to visualise Fashion MNIST data. The difference between the k NN classification and K-Means is that k NN is a supervised classification algorithm rather than an unsupervised clustering algorithm. A supervised algorithm supports the training and validation rather than a randomised selection of values.

Having the ability to develop findings with algorithms helps the Fashion MNIST research process. The scenario above proves that additional research to find the purpose of algorithms’ strengths, weaknesses, and performance can change the outcome of answering some of the complex questions asked.

III. RESEARCH BACKGROUND

A. DataFrame Module

PyData [5] documents that Pandas DataFrame is a ‘two-dimensional, size-mutable, potentially heterogeneous tabular data’, having the ability to have a responsive scope for its internal data, making it a useful tool for data manipulation and analysis. Pandas DataFrame library offers a methodology to control labelled axis, containing alignment tools using Arithmetic operations. The Pandas library includes tools to manage data structures such as tuples, arrays and dictionaries. The extra support of the mentioned data structures means that the library can support semi-structured formats too. These functions are the ‘**read_csv**’, ‘**read_table**’ and ‘**read_clipboard**’. The Fashion MNIST data exists within a CSV format, hence the interest in the ‘**read_csv**’ function.

After opening the Fashion MNIST with Pandas, it was noticeable that the performance time was prolonged. Trying to conduct experiments with Jupyter Notebook and witnessing any kernel crashes meant a restart had to happen. If that was the case, then delaying further experimentation for Pandas to load the ‘fashion-mnist_test.csv’ and ‘fashion-mnist_train.csv’ files and store them in the Pandas DataFrame.

Dask library optimises data analysis on daily computing and tries to keep the same structure as the predecessors’ libraries to improve readability. The library sets to improve performance for multiple libraries, including Pandas, NumPy and Sci-Kit, by taking advantage of multi-threading techniques. According to the Dask Documentation [6], it is good practice to use the original libraries unless the handling of data is performance bound, justified with ‘Dask adds another layer of complexity which may get in the way.’ Moreover, multi-threading enables the execution of various tasks to run in parallel, initiating the processing power, for example, the system’s cores and threads available to execute the tasks. In contrast, Pandas library focuses on a single threading algorithm that takes a significant time to complete. Furthermore, with Python being a high-level programming language rather than a middle-level, programs generally take a long time to execute.

$$\mathcal{O}(n)$$

Fig. 2. Time Complexity of the ‘read_csv’ function

Figure 2, page 2 represents the time complexity of the procedure, ‘**read_csv**’, where n is the length of the CSV input. In the following performance tests, the assignment of $\mathcal{O}(\mathcal{X})$ is the value 21.15 (MB) of MNIST Test CVS and the assignment of $\mathcal{O}(\mathcal{Y})$ is value 126.88 (MB) of MNIST Train CVS.

Pandas library loaded $\mathcal{O}(\mathcal{X})$ within 11.0963900 seconds, with a Mean average time of 11.8792954 seconds with three repetitions. Dask DataFrame library loaded $\mathcal{O}(\mathcal{X})$ within 0.9104877 seconds, with a Mean average time of 1.7883162 seconds with three repetitions. The time difference of these two libraries are 10.1859024 seconds, demonstrating that Dask library saves execution time on average by 73.52%.

Pandas library loaded $\mathcal{O}(\mathcal{Y})$ within 60.4711797 seconds, with a Mean average time of 71.6725737 seconds with three repetitions. Dask DataFrame library loaded $\mathcal{O}(\mathcal{Y})$ within 1.2523577 seconds, with a Mean average time of 5.5200717 seconds. The time difference of these two libraries are 59.2188220 seconds, evidencing that Dask library saves execution time on average by 98.04%.

B. k -Nearest Neighbours

A supervised model named k NN exclusively aims to classify trained labels to untrained data. k NN model uses two combined formulae, Euclidean Distance (fig 4) and Majority Vote (fig 5). Euclidean Distance offers a k NN model to measure dissimilarities between unlabeled data sets. This formula tests the unlabelled data with the labelled data and gives a weighted fit. The Majority Vote formula takes the weighted fit and makes the final decision on the label of the data.

Raul Montoliu [7] mentions three drawbacks of k -NN stating ‘...its computational cost is very high in terms of vector distance calculations...’, ‘...its accuracy strongly depends on the hyperparameters chosen...’ and ‘...the best hyperparameters combination depends on every single scenario and environment’. These three statements summarises that k NN takes time setting-up, executing and having accurate results. This model is challenging to automate and implement with the current state, especially when repetition is a requirement to find the best accuracy.

$$\mathcal{O}(n_{tr} \times n_{ts})$$

Fig. 3. Time Complexity of k NN [7]

Research from Shiliang Sun and Rongqing Huang’s [8], Journal Article, ‘An Adaptive k -Nearest Neighbor Algorithm’ and Sahibsinh A. Dudani [9] formulates the understanding of the Euclidean Distance formula.

$$\mathcal{D}(\mathcal{X}_i, \mathcal{X}_j) = \sqrt{\sum_{r=1}^n \mathcal{W}_r (\mathcal{A}_r(\mathcal{X}_i) - \mathcal{A}_r(\mathcal{X}_j))^2}$$

Fig. 4. Euclidean Distance Formula [8] (1)

Let $\mathcal{X}$ assignment be an vector of $(a_1, a_2, \dots, a_n)$ and n be the dimension sizes of the $\mathcal{X}$ vector. $\mathcal{A}_r$ element is an example of r^{th} attribute. After the calculation stores within the variable, $\mathcal{D}(\mathcal{X}_i, \mathcal{X}_j)$ and according to Shiliang Sun [8], the smaller the value of $\mathcal{D}$, the most similar the two elements are. If this process were to loop, then elements append to the variable of $\mathcal{D}$ converting the variable to an array.

$$\mathcal{Y}(\mathcal{D}_i) = \arg \max_{\mathcal{K}} \sum_{\mathcal{X}_j \in \mathcal{KNN}} \mathcal{Y}(\mathcal{X}_j, \mathcal{C}_k)$$

Fig. 5. Majority Vote [8] (2)

Figure 5: $\mathcal{Y}$ function takes in account of the $\mathcal{D}_i$, where i is the index of the element, implying the $\mathcal{Y}$ function loops for $\mathcal{D}_i^{th}$ element for the comparison. The logic of the $\mathcal{Y}$ is for the class label assignment, which loops $\mathcal{X}_i$ being one of its k NN in the given training set. $\mathcal{Y}$ function contains a loop, which loops $\mathcal{X}_j$ if the element belongs to $\mathcal{NN}$. Finally, calling the $\mathcal{Y}$ function with the arguments, $\mathcal{X}_j$ and $\mathcal{C}_k$. The $\mathcal{NN}$ variable is a constant number, for example 3, which affects where the percentage of the data where the data belongs to based on similarity.

C. Decision-Tree

The SciKit-Learn [10] documentation states that the base of the Decision-Tree Regression model is Classification and Regression Trees (CART), ‘scikit-learn uses an optimized version of the CART algorithm; however, the scikit-learn implementation does not support categorical variables for now.’ Findings from the Journal Article, ‘An Assessment of Decision Tree based Classification and Regression Algorithms’, Soham Pathak [11] put together indicate that the CART algorithm

provides a version of classification and regression, whereas other algorithms lack. However, the algorithm lacks categorical variables such as ‘Gender’, ‘Marital Status’ and ‘Eye Colour’ to allow field filtering.

Figures 6 displays the CART algorithm time complexity, formulated from ‘Computational complexity analysis of decision tree algorithms’ conducted by Sani, Habiba M. [12], where $\mathcal{A}$ is the attributes, and n is the number of observations.

$$\mathcal{O}(\mathcal{A}n \log_2 n)$$

Fig. 6. Time Complexity of CART algorithm

IV. METHODOLOGY

The application of two models, k NN and Decision-Tree, assists the MNIST training experimentation. Testing these two models includes comparing performance, accuracy, and a confusion matrix report (precision, recall, f1-score and support) and assembling the dataset from a series of performance and calculation tests.

The arrangements of the Confusion Matrix and Confusion Matrix Report tables aid in visualising the results for both k NN and Decision-Tree Classifier experiments. The Confusion Matrix displays the position of the correct and incorrect predictions that the model decided. Whereas the Confusion Matrix Report provides a detailed analysis of results that the Confusion Matrix reported. The ‘sklearn.metrics’, ‘confusion_matrix’ and ‘classification_matrix’ functions help with the result reporting.

The Classification Matrix is a tool that displays Precision, Recall, F1-Score, Support, Accuracy, Macro Average and Weighted Average of the data. The Classification Matrix looks into the Precision, Recall, F1-Score, Support, Accuracy, Macro Average and Weighted Average of the data. The purpose of the Classification Matrix is to help observe the performance and accuracy of the model’s progress. It enables fine tweaking of the model to avoid underfitting and overfitting.

Precision measures the percentage of correct predictions made by the model with the figure 7 equation. Recall determines the percentage of correct and incorrect assumptions using the figure 8 equation. F1-Score measures the mean of Precision and Recall demonstrated in figure 9. Support shows the total number of predictions that the model favoured a class. Macro Average calculates the precision, recall, and F1-score average for each class and then calculates the overall average by dividing the sum of these averages by three, illustrated in figure 10. Weighted Average calculates the dot product of precision, recall, F1-score and Support for each class and divides it by the support’s dot product that figure 11 is displaying. Additionally, Macro Average and Weighted Average give a better understanding of the performance, strengths and weaknesses of the model used.

Key: — *Note:* These variables are matrices:

- $\mathcal{TP}$: True Positives - $\mathcal{FP}$: False Positives
- $\mathcal{S}$: Support $\mathcal{C}$: Classes

$$\mathcal{P}(\mathcal{TP}, \mathcal{FP}) = \frac{\mathcal{TP}}{\mathcal{FP}}$$

Fig. 7. Mathematic Equation of Precision

$$\mathcal{R}(\mathcal{TP}, \mathcal{FP}) = \frac{\mathcal{TP}}{(\mathcal{TP} + \mathcal{FP})}$$

Fig. 8. Mathematic Equation of Recall

$$\mathcal{F}(p, r) = 2 * \frac{(p \times r)}{(p + r)}$$

Fig. 9. Mathematic Equation of F1-Score

$$m = \frac{1}{3} \left(\frac{\sum_{i=1}^{i < |C|} \mathcal{P}(C_i)}{|C|} + \frac{\sum_{i=1}^{i < |C|} \mathcal{R}(C_i)}{|C|} + \frac{\sum_{i=1}^{i < |C|} \mathcal{F}(\mathcal{P}(C_i), \mathcal{R}(C_i))}{|C|} \right)$$

Fig. 10. Mathematic Equation of Macro Average

Section VIII, page 5 is the conducted experiment results. The discussion of the comparison between k NN and Decision-Tree is in section V, page 4.

A. k -Nearest Neighbour Model

Preparing the ‘training’, ‘test’, and ‘validation’ data, the ‘train_test_split’ function from the ‘sklearn.model_selection’ Python module to create training and validation variables to support the k NN experimentation. Figure 12 default parameters of the ‘train_test_split’ function sourced from the documentation of SciKit Learn [10]. For the k NN experiment, the function will populate two sets of lists, one for training and the other for validation. Training iteration, assigning sixty per cent to the train size (‘train_size=0.60’) and twenty-eight to the random state seed (‘random_state=28’). Validation iteration, assigning twenty per cent to the train size (‘train_size = 0.20’) and eighty-four to the random state seed (‘random_state = 84’). After numerous testing, these figures appear to populate good training data with decent accuracy and less compromise to the performance.

Within the k NN experiments, the Python library, ‘sklearn.neighbors’ assists with the classification and regression of untrained data providing the k NN model. In the documentation of SciKit Learn [10], the default parameters of the ‘KNeighborsClassifier’ class are as stated in figure 13.

The KNeighborsClassifier model has several hyperparameters (fig 13). The focus point of these hyperparameters that assist within the experiment includes ‘n_neighbours’, ‘algorithm’ and ‘p’. Firstly, ‘n_neighbours’ specifies the number of NN to consider when making a prediction; the default value is ‘5’; however, ‘6’ offers a 0.2249 per cent increase. Testing different values for this hyperparameter at the default value worked best for the Fashion MNIST dataset. Secondly, ‘algorithm’ specifies the algorithm to compute the NN; the default value is ‘auto’. Finally, ‘p’ specifies the power parameter for the Minkowski distance metric; the default value is ‘2’. Testing different values for these hyperparameters, the combination that results in the best performance on the Fashion MNIST dataset is determined. The default of the ‘algorithm’ parameter inherently determines an algorithm that fits best for the data provided. However, it sacrifices accuracy over performance. Using ‘ball_tree’ and ‘kd_tree’ algorithms to compare accuracy and performance instead of relying on ‘auto’ to give an understanding of the best analyst for the Fashion MNIST dataset. ‘p’ parameter selects Euclidean Distance as the default parameter, which provides an accurate measurement with a good performance time. The Manhattan Distance can increase accuracy with a higher dimensional data

set, affecting performance time. In this case, the priority will remain on Euclidean Distance to test k NN to compromise the stake of losing time.

B. Decision-Tree Model

To prepare the Fashion MNIST dataset for a Decision-Tree model experiment, twenty-five per cent of the data progresses towards testing, and the remaining seventy-five per cent advances towards training. To ensure reproducibility, we will randomly shuffle the data with a fixed seed of forty-two before dividing it into training and test sets. From the ‘sklearn.tree’, the ‘DecisionTreeClassifier’ model trains the test data for this experimentation. Figure 14 shows some of the default hyperparameters from the SciKit Learn’s [10] documentation. The ‘criterion’, ‘max_depth’, ‘min_samples_split’ and ‘min_samples_leaf’ hyperparameters will help with the current unbalance of train and test data.

Setting ‘max_depth’ to the value of twenty, after trying values of three, five, ten, thirty and twenty-five, has higher accuracy and better performance, accuracy and consistency, the lowest percentage with ‘max_depth’ assigned as three, being fifty-five per cent. Using ‘min_samples_split’ and ‘min_samples_leaf’ helps control the underfitting and overfitting of the training data, however, only increasing the accuracy by a marginal of ‘0.804’ from ‘0.801’. The final parameter constant ‘random_state’ has the assignment of value eighty-one. These parameters will remain the same for the experiment, and the attention will concentrate on the ‘criterion’, testing the accuracy and performance of ‘gini’ and ‘entropy’.

V. DISCUSSION

Tables IV, I: The two algorithms did not impact the test results and outputted the same results after several executions enabling the merge of the two datasets into one. However, the execution time of the ‘kd_test’ algorithm took two minutes and fifteen seconds longer than the ‘ball_test’ algorithm. The results of k NN tend to have a consistent precision, recall and F1-score with each class.

Tables V, VI, II, III: After testing the two criterion algorithms, the findings indicate that both models have an accuracy of eighty per cent and a difference of a seven-second execution time. Observing the Gini criterion model, results demonstrate a slightly higher precision and F1 scores for most classes, whereas the Entropy criterion model has a slightly higher recall. In comparison, the choice of the Entropy criterion model proves better in terms of recall.

$$w = \frac{\sum_{i=1}^{i < |C|} (\mathcal{P}(C_i) \cdot \mathcal{R}(C_i) \cdot \mathcal{F}(\mathcal{P}(C_i), \mathcal{R}(C_i)) \cdot S_i)}{\sum_{i=1}^{i < |C|} S_i}$$

Fig. 11. Mathematic Equation of Weighted Average

```
train_size=None,
random_state=None,
shuffle=True,
stratify=None
```

Fig. 12. Default Parameters of 'train_test_split' function [10]

```
n_neighbors=5,
weights='uniform',
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None
```

Fig. 13. Default Hyperparameters of the 'KNeighborsClassifier' model. [10]

VI. CONCLUSION

Decision-Tree with the 'Entropy' criterion appears to have higher precision than k NN. However, it lacks in the recall and F1-score elements and accuracy. With different hyperparameters, the Decision-Tree may have fitted the model better. However, with the Decision-Tree with both 'Gini' and 'Entropy' criterions, the model had the lowest accuracy score of fifty-five per cent, and any higher depths meant a compromise in performance. Granted, the k NN model had a highest of twenty-one minutes and twenty-nine seconds execution time for an extra four per cent of accuracy and slightly higher recall and f1-score. This fact alone promotes the Decision-Tree model with both 'Gini' and 'Entropy' configurations when exploring data where a requirement of accuracy is not an issue when working with a Fashion MNIST dataset.

VII. TERMINOLOGY

List of terminologies used in this document:-

- MNIST - Modified National Institute of Standards and Technology Dataset.
- k NN - k -Nearest Neighbour.
- DFS - Depth First Search.
- BFS - Breadth First Search.
- CoG - Center of Gravity.
- CART - Classification and Regression Trees.

VIII. RESULTS - CLASSIFICATION REPORT

TABLE I
CONFUSION MATRIX REPORT FOR THE 'KNEIGHBORSCLASSIFIER'
MODEL WITH THE BALL-TREE AND KD-TREE ALGORITHM.

Class	Precision	Recall	F1-Score	Support
T-Shirt/Top	0.732	0.886	0.802	3565
Trouser	0.988	0.955	0.971	3602
Pullover	0.713	0.790	0.750	3622
Dress	0.877	0.854	0.865	3578
Coat	0.753	0.722	0.737	3580
Sandal	0.990	0.809	0.890	3635
Shirt	0.639	0.530	0.580	3578
Sneaker	0.860	0.951	0.903	3655
Bag	0.977	0.938	0.957	3622
Ankle Boot	0.890	0.953	0.921	3563
Accuracy			0.839	36000
Macro Avg.	0.842	0.839	0.838	36000
Weighted Avg.	0.842	0.839	0.838	36000
Accuracy Score: 84% Execution Time: BT: 9m 0s KDT: 11m 15s				

```
criterion='gini',
splitter='best',
max_depth=None,
min_samples_split=2,
min_samples_leaf=1
```

Fig. 14. Default Hyperparameters of 'DecisionTreeClassifier'[10]

TABLE II
CONFUSION MATRIX REPORT FOR THE 'DECISIONTREECLASSIFIER'
MODEL WITH THE GINI CRITERION.

Class	Precision	Recall	F1-Score	Support
T-Shirt/Top	0.736	0.760	0.748	1519
Trouser	0.947	0.949	0.948	1479
Pullover	0.699	0.707	0.703	1494
Dress	0.796	0.809	0.802	1541
Coat	0.666	0.688	0.677	1483
Sandal	0.881	0.894	0.888	1435
Shirt	0.558	0.506	0.531	1561
Sneaker	0.892	0.873	0.882	1541
Bag	0.915	0.916	0.915	1447
Ankle Boot	0.911	0.917	0.914	1500
Accuracy			0.800	15000
Macro Avg.	0.800	0.802	0.801	15000
Weighted Avg.	0.798	0.800	0.799	15000
Accuracy Score: 80%				Execution Time: 44s

TABLE III
CONFUSION MATRIX REPORT FOR THE 'DECISIONTREECLASSIFIER'
MODEL WITH THE ENTROPY CRITERION.

Class	Precision	Recall	F1-Score	Support
T-Shirt/Top	0.747	0.760	0.753	1519
Trouser	0.947	0.944	0.945	1479
Pullover	0.679	0.714	0.696	1494
Dress	0.823	0.803	0.813	1541
Coat	0.670	0.680	0.675	1483
Sandal	0.884	0.887	0.886	1435
Shirt	0.575	0.545	0.560	1561
Sneaker	0.887	0.877	0.882	1541
Bag	0.930	0.925	0.927	1447
Ankle Boot	0.905	0.919	0.912	1500
Accuracy			0.804	15000
Macro Avg.	0.805	0.805	0.805	15000
Weighted Avg.	0.803	0.804	0.803	15000
Accuracy Score: 80%				Execution Time: 37s

IX. RESULTS - CONFUSION MATRICES

TABLE IV

CONFUSION MATRIX FOR THE 'KNEIGHBORSClassifier' MODEL WITH THE BALL-TREE AND KD-TREE ALGORITHM.

Act.		Pred.									
		T-Shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
	T-Shirt/Top	3160	4	63	81	21	1	213	0	22	0
	Trouser	22	3439	29	85	5	0	22	0	0	0
	Pullover	69	3	2861	33	328	0	319	0	9	0
	Dress	201	29	29	3055	161	0	98	0	4	1
	Coat	25	3	481	140	2585	0	341	0	5	0
	Sandal	6	0	2	2	0	2941	27	400	7	250
	Shirt	803	2	477	69	296	0	1898	0	33	0
	Sneaker	0	0	0	0	0	13	1	3476	0	165
	Bag	30	1	67	18	34	4	46	22	3397	3
	Ankle Boot	0	0	3	1	1	12	4	145	0	3397

TABLE V

CONFUSION MATRIX FOR THE 'DECISIONTreeClassifier' MODEL WITH THE GINI CRITERION.

Act.		Pred.									
		T-Shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
	T-Shirt/Top	1155	8	20	82	11	7	219	0	16	1
	Trouser	14	1404	4	42	3	1	8	0	3	0
	Pullover	18	2	1057	33	203	1	165	0	14	1
	Dress	81	49	20	1246	75	3	55	0	11	1
	Coat	18	6	198	93	1021	1	136	1	8	1
	Sandal	3	2	2	3	0	1283	5	71	24	42
	Shirt	266	8	198	56	203	5	790	2	32	1
	Sneaker	0	1	0	0	0	102	0	1345	10	83
	Bag	15	2	11	11	12	21	36	9	1325	5
	Ankle Boot	0	0	3	0	4	32	1	80	5	1375

TABLE VI

CONFUSION MATRIX FOR THE 'DECISIONTreeClassifier' MODEL WITH THE ENTROPY CRITERION.

Act.		Pred.									
		T-Shirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
	T-Shirt/Top	1155	9	30	65	10	4	233	0	12	1
	Trouser	24	1396	1	36	6	1	10	0	4	1
	Pullover	26	5	1066	20	200	2	158	0	16	1
	Dress	90	43	23	1237	80	4	51	0	10	3
	Coat	13	5	236	72	1009	2	139	0	6	1
	Sandal	1	6	1	3	1	1273	6	92	15	37
	Shirt	230	9	198	52	186	5	850	1	28	2
	Sneaker	0	0	0	0	0	90	0	1351	6	94
	Bag	5	1	15	18	12	22	26	6	1338	4
	Ankle Boot	3	0	0	0	1	37	4	73	4	1378

X. REFERENCE LIST

- [1] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," Aug. 2017.
- [2] Edward Patch, "Artificial Intelligence and Machine Learning: Search and ML Clustering," p. 4.
- [3] V. Palanisamy and S. Vijayanathan, "A Novel Agent Based Depth First Search Algorithm," in *2020 IEEE 5th International Conference on Computing Communication and Automation (IC-CCA)*, Oct. 2020, pp. 443–448, iSSN: 2642-7354.
- [4] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263–, Feb. 1998.
- [5] PyData, "pandas.DataFrame — pandas 1.5.2 documentation." [Online]. Available: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html>
- [6] Dask, "Best Practices — Dask documentation." [Online]. Available: <https://docs.dask.org/en/stable/array-best-practices.html>
- [7] R. Montoliu, A. Pérez-Navarro, and J. Torres-Sospedra, "Efficient tuning of k NN hyperparameters for indoor positioning with N-TBEA," in *2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, Oct. 2022, pp. 39–44, iSSN: 2157-023X.
- [8] S. Sun and R. Huang, "An adaptive k-nearest neighbor algorithm," in *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 1, Aug. 2010, pp. 91–94.
- [9] S. A. Dudani, "The Distance-Weighted k-Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 4, pp. 325–327, Apr. 1976, conference Name: IEEE Transactions on Systems, Man, and Cybernetics.
- [10] SciKit Learn, "scikit-learn: machine learning in Python — scikit-learn 1.2.0 documentation." [Online]. Available: <https://scikit-learn.org/stable/index.html>
- [11] S. Pathak, I. Mishra, and A. Swetapadma, "An Assessment of Decision Tree based Classification and Regression Algorithms," in *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, Nov. 2018, pp. 92–95.
- [12] H. M. Sani, C. Lei, and D. Neagu, "Computational complexity analysis of decision tree algorithms," 2018, accepted: 2019-01-22T15:57:05Z. [Online]. Available: <https://bradscholars.brad.ac.uk/handle/10454/16762>