IT UNIVERSITY OF COPENHAGEN

Classifying Clothing Images

Machine Learning (Autumn 2023)
BSMALEA1KU

Group 3o:

Marija Trpenovska (mtrp@itu.dk)
Petya Petrova (pety@itu.dk)
Esther Dorthea Bøgelund Madsen (esma@itu.dk)

Github link:
`https://github.com/PetyaPetrova96/clasifying-clothing-types-`

BSc in Data Science
IT University of Copenhagen
January, 2024

Contents

1 Introduction

The main goal of this project was to distinguish between five different types of clothes from an image of an item. To achieve that, we implemented Linear Discriminant Analysis (LDA) as a feature reduction from scratch and fed the Naive Bayes classifier, which also had to be implemented and trained from scratch with the top two corresponding eigenvectors. Furthermore, k-Nearest Neighbour and Decision trees were also used to achieve this task. Both classifiers were trained three times: first on the entire dataset with pixel values from the grayscale, then on the dataset processed using two feature reduction methods—Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). Grid search was used to find the optimal hyperparameters of the Decision Trees fed with LDA.

2 Data Exploration

2.1 The dataset

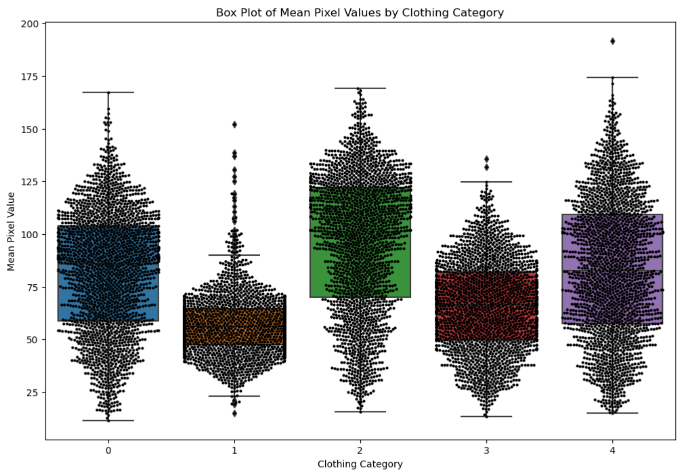
In this project, we will explore various methods to identify the category of clothing depicted in an image. The project dataset comprises 15000 labeled images of clothing, sourced from the Zalando website[1] where each image is a grayscale 28x28 depiction, representing either a t-shirt, trousers, a pullover, a dress, or a shirt.

2.1.1 Variable identification

The data set consists of categorical variables. The images are divided into a training set of 10000 images and a test set of 5000 images. The images and associated labels are available in NPY format as: fashion train.npy and fashion test.npy. Each line describes a piece of clothing. The first 784 columns are the pixel values of the 28x28 grayscale image, each taking an integer value between 0 and 255. The last column, number 785, is the category of clothing and takes values in 0, 1, 2, 3, 4

2.1.2 Univariate analysis

Here's a brief description of the results derived from the specific box plots generated for the Zalando fashion dataset:



The box plots reveal distinctive characteristics in the distribution of mean pixel values for different clothing categories. Here are key insights:

- T-Shirt/Top (Category 0): This category displays a relatively tight distribution of mean pixel values, with a moderate median. As show in the plot there are no visible outliers.
- Trousers (Category 1): Trousers exhibit a wider spread of mean pixel values, suggesting greater variability in grayscale patterns. The median value is moderate, and there are outliers indicative of diverse trouser styles.
- Pullover (Category 2): Pullovers show a concentrated distribution of mean pixel values with a slightly higher median. There are no outliers, suggesting consistent pixel patterns.
- Dress (Category 3): Dresses have a relatively tight distribution of mean pixel values, similar to T-Shirt/Top. The median value is moderate, and there are a few outliers indicating diverse dress designs.
- Shirt (Category 4): Shirts exhibit a broader distribution of mean pixel values, implying significant variability in grayscale patterns. The median is relatively high, and there are a minute amount of outliers, probably reflecting diverse shirt styles.

2.1.3 Plotting the class means

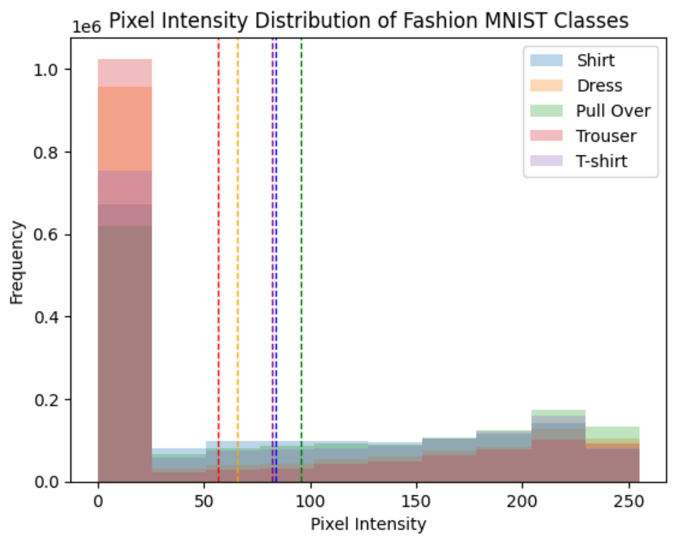


Figure 1: Pixel Intensity Distribution Of Fashion MNIST Classes

The histogram provides an overview of the pixel intensity distribution for five clothing categories: T-shirt, Trouser, Pullover, Dress, and Shirt. It is notable that each category exhibits a distinct peak in pixel intensity, indicating characteristic patterns specific to that type of clothing. However, we observe regions of overlap between the histograms, suggesting similarities in pixel intensity across certain categories. The dashed lines are representing the means for each category and they help identify the central tendency of pixel intensities.

2.1.4 Detecting and treating missing values

Upon a thorough examination of the dataset, it is noteworthy that no missing values have been identified. Every observation is complete, and there are no instances requiring imputation or treatment for missing data. This absence of missing values ensures the integrity of the dataset, providing a solid foundation for further analyses and modeling.

3 Methods

3.1 Dimensionality Reduction

3.1.1 Principal Component Analysis (PCA)

The primary objective of PCA is to identify the principal components, which are linear combinations of the original features. These principal components are orthogonal to each other and are ordered by the amount of variance they explain in the dataset.

The method identifies the directions (principal components) in which the data varies the most by computing the covariance matrix of the standardized data. The covariance between two features X_i and X_j is given by:

$$\operatorname{cov}(X_i, X_j) = \frac{1}{n-1}\sum_{k=1}^{n}(X_{ik} - \bar{X}_i)(X_{jk} - \bar{X}_j)$$

The covariance matrix is then formed using covariances between all pairs of features.

The next step involves finding the eigenvectors and eigenvalues of the covariance matrix. Eigenvectors represent the directions of maximum variance, and eigenvalues indicate the magnitude of the variance in those directions.

$$\Sigma v = \lambda v$$

The first principal component captures the maximum variance, followed by the second, third, and so on. The eigenvectors are ranked in descending order based on their corresponding eigenvalues. The top k eigenvectors, where k is the desired dimensionality, are selected to form the transformation matrix. We started with importing the PCA module from the scikit-learn library[2]. Subsequently, a PCA object is instantiated, specifying the desired number of components as 2, later in the project changed to 100 to capture more variance. This choice signifies the intention to project the original data into a two-dimensional space, thereby capturing the most salient features while discarding less significant information[3].

3.1.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis is used to reduce the number of features in a dataset, it is particularly useful in dealing with classification problems. LDA aims to maximise the separation between classes in the dataset. It does it by constructing a lower dimensional space which maximises the between class variance and minimizes the within class variance, through the following steps:[4]

1. Calculate the difference between the mean and the sample of each class, giving us the within-class scatter matric.

$$S_W = \sum_{i=1}^{c}\sum_{j=1}^{n_i}(x_{ij} - \mu_i)(x_{ij} - \mu_i)^T$$

c is the total number of classes, n_i is the number of samples in class i, x_{ij} is the j^{th} sample in class i, μ_i is the mean vector of class i, and T is matrix transpose.

In our case we can reduce the inner sum into taking the covariance of x_i transposed[5] so we get:

$$S_W = \sum_{i=1}^{c} \Sigma_i$$

2. Calculate the distance between the mean of different classes, giving us the between-class scatter matrix.

$$S_B = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T$$

S_B represents the sum, over all classes i, of the outer products of the difference between the mean vector of each class (μ_i) and the overall mean vector μ ($\mu = 1/k * \sum k = 1/5 * \sum \mu_i$), scaled by the number of samples in each class (n_i), where c is the total number of classes, and T denotes matrix transpose.

3. Find LDA projection vectors by getting discriminant matrix M:

$$M = S_W^{-1} * S_B$$

4. Get the eigenvalues and eigenvector pairs by solving: $M * v = \lambda * I * v$ *where v is the eigenvector and λ is eigenvalue of the matrix product M and I is the identity matrix.* We used the implementation from the Scipy linalg library called eigh[6]

5. Sort the eigenvalue-eigenvector pairs based on eigenvalues and select the top 2.

We made these steps into a function which takes an array with images, applies the LDA projection vectors we calculated to each image row, and returns an array with 3 columns representing the corresponding $[X1, X2, Class]$ per image and same amount of rows as the input array.

3.2 Classifiers

3.2.1 Naive Bayes

The Naive Bayes Classifier is a probabilistic machine learning classifier. It is based on Bayes Theorem. It is called Naive because of its strong assumption of the features looked at are unrelated to each other, considering the class is already known. It is based on Bayes Theorem which is written as:

$$P(\text{Class — Features}) = \frac{P(\text{Features — Class}) \cdot P(\text{Class})}{P(\text{Features})}$$

Bayes theorem can be used to calculate the posterior probability for each class. The class with the highest posterior probability is the predicted class.

We have implemented it as follows with our two features X_1 and X_2 per image we got from applying our LDA to each image in our dataset through the steps described in 3.1.2:

$$P(C_i|X_1, X_2) = \frac{P(C_i) \cdot P(X_1|C_i) \cdot P(X_2|C_i)}{P(X_1) \cdot P(X_2)}$$

Where: *$P(C_i|X_1, X_2)$: Posterior probability of class C_i given features X_1 and X_2, $P(X_1|C_i)$: Likelihood of observing X_1 given class C_i, $P(X_2|C_i)$: Likelihood of observing X_2 given class C_i, $P(C_i)$: Prior probability of class C_i (before considering the features), $P(X_1)$: Probability of observing X_1, $P(X_2)$: Probability of observing X_2.*
Since our marginal probabilities $P(X_1)$ and $P(X_2)$ are the same for each class there is no need to divide with them:

$$P(C_i|X_1, X_2) = P(C_i) \cdot P(X_1|C_i) \cdot P(X_2|C_i)$$

1. Get the simple probabilities $P(C_i)$ for each class, and save them in a dictionary called *simpprob*, where the key is the class name and the values are the simple probabilities.
2. Calculate $P(X_i|C_i)$, the likelihood of observing each given feature X, given class C.
   (a) apply our projection vectors X1, X2 to our data divided into classes, so we get eg. $tshirt\_proj\_x1$ *and* $tshirt\_proj\_x2$ per class, and save them in $x1\_data$ and $x2\_data$ arrays respectively
   (b) for each class i:
      i. get *k,bins* from the numpy.histogram function per feature, set the parameter 'density = True' so we get the probability density function of the bins instead of the frequency. k is an array containing the bin divided probability density function values of the histogram, *bins* is an array which contains the bin-edges of the histogram[7]
      ii. get the *bin_index* using the numpy.digitize function per feature, which returns the index of the bin the respective feature belongs in in the respective X_i histogram.
      iii. if the *bin_index* is within the histogram bins, calculate the probability of observing each feature given the class i through getting the *bin_index* index of the k array containing the bin-divided probability density function results, else set the probability to 0. Save the feature specific probabilities in *binprop1* and *binprop2* respectively.
      iv. get *class_simp_prob* through taking index i in the *categories* array, containing the classes in string form, and using that as the key in the *simpprob* dictionary.
      v. calculate the probability of the image being in class i through taking $binprop1 * binprop2 * class\_simp\_prob$
      vi. append the result to a *results* dictionary with the class text name as the key and the calculated posterior class probability as the value.
3. Taking the class with the highest calculated posterior probability from the *results* dictionary and returning that as the predicted class along with the probability of it.

3.2.2 KNN

KNN or k nearest neighbour is a widely used supervised machine learning algorithm that can handle both numerical and categorical data. The classifier identifies the K closest neighbours to a given data point, using a distance metric like Euclidean distance by the following formula :

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

Once these nearest neighbors are determined, their labels are examined to classify the original data point. [8]

3.2.3 Decision Trees

Decision trees classifier is a powerful supervised machine learning algorithm that can be used for both regression and classification tasks. In the training process, the Decision Tree algorithm picks the most suitable feature to divide the data, considering measures like entropy or Gini impurity. In our case, we have decided to use entropy, because it gave us slightly better performance. The aim is to identify the feature that provides the most information gain. After identifying the feature with the most information gain, the decision tree uses it to partition the data and continues this process until it constructs a tree capable of making predictions on new, unseen data. The algorithm follows the following steps for our dataset:
Step 1: Calculate the Entropy of the entire dataset with five classes (a measure of uncertainty or disorder in a dataset) by this formula:

$$H = -\sum_{i=1}^{5} P_i \cdot \log_2(P_i)$$

Step 2: Calculate the Information Gain(a measure of the effectiveness of a feature) for each feature given by:

$$IG = H_{\text{parent}} - \sum_{j=1}^{k} \left(\frac{N_j}{N} \times H_j \right)$$

Where:
The entropy (H_{parent}) of the parent node, representing the entire dataset, is calculated based on the total number of samples (N). For each child node (j), where N_j is the number of samples after the split, the entropy (H_j) of that child node is determined. The relationships among these variables are integral to assessing the information gain or impurity reduction in decision tree algorithms.
The feature with the highest Information Gain becomes the root node. It's the most useful feature for the first split.

Step 3: Recursive Splitting
After identifying the root node, the next step involves recursively splitting the dataset. For each branch created by the root node, the following sub-steps are performed:

- Calculate Information Gain for the features.
- Select the feature with the highest Information Gain.
- Create child nodes based on the chosen feature.
- Recursively repeat for each child node, refining the tree iteratively.

Stopping Criteria:

- Continue splitting until a stopping criterion is met, such as reaching a maximum depth, having a minimum number of samples in a node, or no further improvement in information gain[9],[10].

3.3 Other methods

3.3.1 K-fold Cross-Validation

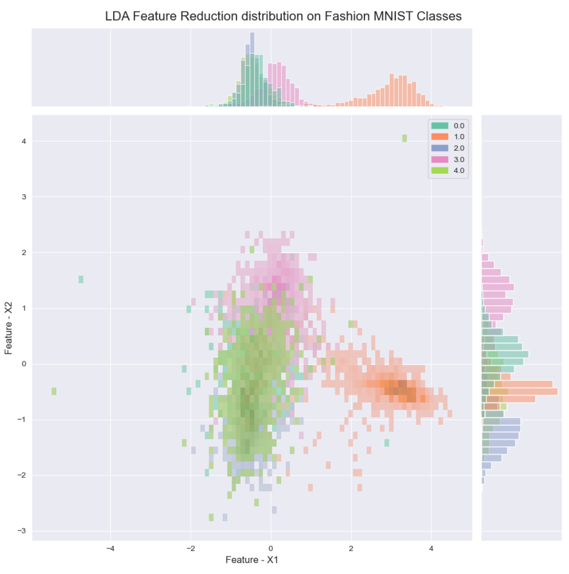
Cross-validation is a technique in machine learning that helps evaluate and choose the best model for a specific task. It's a simple and effective method that avoids biases, making it a powerful tool for assessing a model's performance. The process involves training and testing the model on different parts of the dataset, ensuring a fair and reliable evaluation. K-Fold cross-validation divides the dataset into k equal parts, or folds to enhance the assessment process[11].

3.3.2 Grid Search

GridSearchCV is a method used to find the best settings for a model's hyperparameters. Hyperparameters are values that influence a model's performance. Instead of manually trying different values, GridSearchCV automatically tests various combinations, helping discover the optimal ones[12].

4 Result and Analysis

4.1 Results of LDA



Through applying the LDA projection vectors to our dataset and visualising it through the combined histograms and scatterplot seen to the right, we can see that the LDA has managed to separate the class instances fairly well, especially regarding the Trousers class, which is well separated on both the x and y axis while still keeping its discriminate features. The second projection vector X2 has separated the data and the means distinctively, since there's not a lot of overlap in the histograms. In the histograms showing the feature X1 it is very visible that class 0, class 2 and class 4 have a lot of overlap between them in the histograms and scatterplot, which can make it tough to distinguish between the different classes and predict a clear result for them when training a classifier on it and doing predictions. A possible way to remedy this would be to include more features, than just the two we have used so far. This would allow the LDA to generate a clearer distinction between those three classes, as the ideal number of LDA features for a dataset with classes k would be k-1, so in our case, 4 features would be an ideal next step to try and improve our feature reduction and the separation of classes.

4.2 Results of PCA vs LDA 2 components and PCA 100 components

We have used two feature reduction methods PCA and LDA. We have visualized how by increasing the principle components, the clarity of the pictures increases as shown in the figure below.

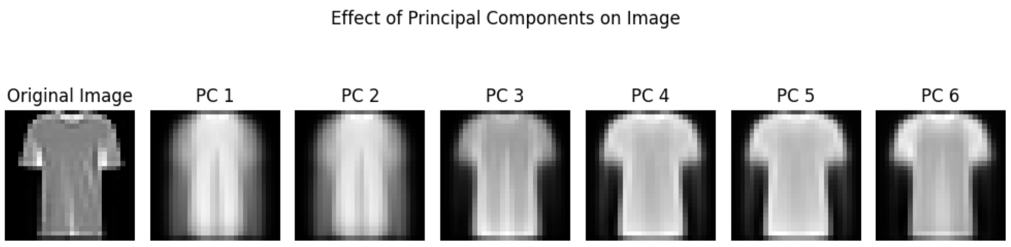
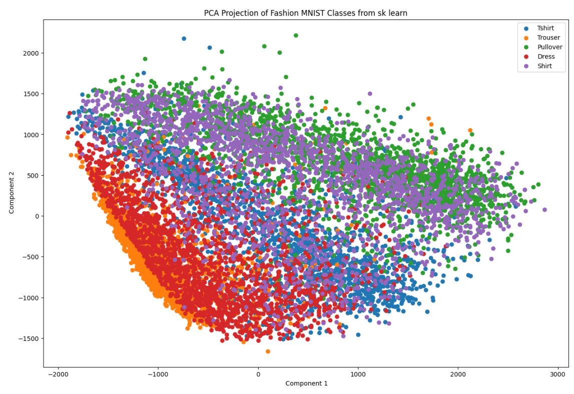
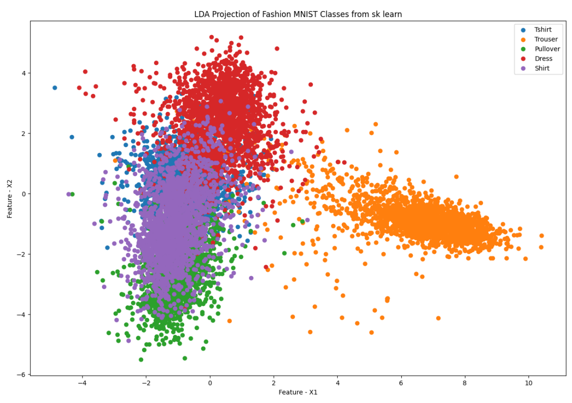


Figure 2: Effect of principle components

To see how both methods perform, we have reduced the dimension of our data by two components by using them. The results should be seen on the scatterplots below where the axes on the scatterplots represent the reduced components. As indicated by the figure, LDA effectively distinguishes the pants category from the others and, to some extent, the dress category. However, it faces challenges in distinguishing the remaining categories, as they appear to overlap. In contrast, PCA also exhibits overlaps among categories, but the data points are less clustered compared to LDA.



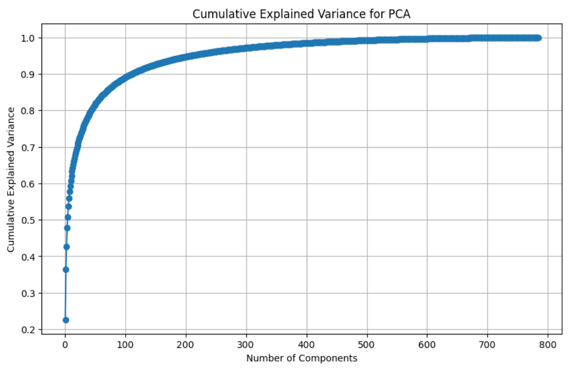
PCA 2 components



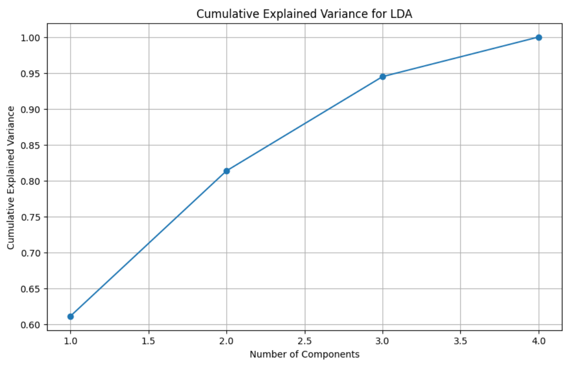
LDA 2 components

Figure 3: Scatterplots of LDA and PCA projection per class

It is a challenge to judge which one is better at distinguishing, therefore we have decided to feed our optional classifiers with both reduced datasets. Our goal was to capture at least 90 percent of the variance in the data. Specifically, in PCA, a hundred components explain up to 90 percent of the data, while LDA, with four components, accounts for the entire dataset. While both classifiers performed equally well overall, there were instances where the model using the PCA components outperformed the one using LDA components and vice versa.



PCA 100 components



LDA 4 components

Figure 4: PCA LDA explained variance

4.3 Results of Naive Bayes

Through applying our Naive Bayes to our LDA reduced test dataset, as described in 3.1.2 and 3.2.1 we got an overall average accuracy of 88.8%. The overall results show a balanced and consistent classifier throughout several classes. The model has a good balance between recall and precision with an F1 score of 71% showing that it can generalise well throughout differing classes as seen in Table 1 below.

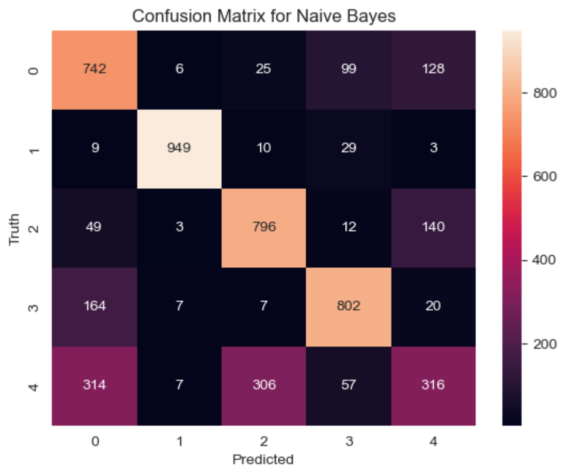
Table 1: Performance Metrics for Each Class and Average

Class	**Precision**	**Recall**	**F1 Score**	**Accuracy**
T-shirt (0)	0.580595	0.742	0.651449	0.8412
Trouser (1)	0.976337	0.949	0.962475	0.9852
Pull-over (2)	0.695804	0.796	0.742537	0.8896
Dress (3)	0.802803	0.802	0.802401	0.9210
Shirt (4)	0.520593	0.316	0.393279	0.8050
Average	0.715226	0.721	0.710428	0.8884

4.3.1 Strengths & Weaknesses

The model does exceptionally well in classifying Trousers and Dresses. It attains high precision, recall and F1 score values. This shows that the classifier combined with the LDA has a good ability to differentiate between those classes. This also aligns with the scatterplot created from the LDA features seen in section 4.1 showcasing a big separation in the datapoints between Trousers and the rest of the data.
The model shows a bigger struggle with classifying Shirts specifically with really low precision and recall values. The low recall of 31% shows that the model seems to struggle with identifying a big proportion of the actual shirts, therefore also lowering the overall accuracy of the model. The low F1 score of Shirts suggests that the model has issues with minimizing both false positives and negatives, where in this case it has a lot of false negatives, meaning that it misses a lot of Shirts when identifying.

Looking at the confusion matrix it can be seen that it often identifies a Shirt as either a T-Shirt or a Pullover, which when looking at the means of those classes, they are fairly close as seen in Figure 1 along with the box plot in Section 2.1.2 showing a lot of similarities between those three classes specifically. Even looking at the real life composition of those three clothing types, they have similar shapes and features in general. To mitigate this issue it could be further looked into other feature reduction methods, changing the number of components of our LDA or other types of classifiers that would work better on this particular dataset.



4.4 Results of KNN

We conducted three iterations of kNN training on our dataset. Initially, the entire dataset was utilized, followed by two additional runs with dimensionality reduction techniques. The first reduction involved applying PCA with 100 components, and the second utilized LDA with 4 features. To assess performance and identify the optimal dataset, all three classifiers employed default hyperparameters from scikit-learn and were subsequently compared. They were tested all on both - the training dataset by using k-fold cross validation having 5 folds and the test data. By using k-fold cross-validation, we improved how we checked the classifier's performance. Instead of just splitting the data once into training and test sets, we did it multiple times (k=5 times) in different ways. This helped us get a better and more reliable understanding of how well the classifier works on various parts of our dataset. Here are the results:

Metrics	kNN no PCA no LDA		kNN PCA		kNN LDA	
	Test	Cross-val	Test	Cross-val	Test	Cross-val
Precision	0,82	0,83	0,83	0,84	0,80	0,86
Recall	0,82	0,83	0,83	0,84	0,81	0,86
F1 Score	0,82	0,83	0,83	0,84	0,81	0,86
Accuracy	0,82	0,83	0,83	0,84	0,80	0,86

Table 2: Results kNN default parameters,k = 5

To further enhance our model, we undertook an additional exploration to determine the ideal number of neighbors. This involved multiple rounds of training the classifier with PCA since it had the best performance out of them all, systematically evaluating different neighbor counts, and identifying the point where both accuracy and F1 score reached their peaks. When analyzing the graph, it becomes clear that both accuracy and F1 score attain their highest values at specific points. Notably, when examining the performance of the model, it is observed that the values for k = 5 and k = 15 are nearly identical. This similarity in performance suggests that a simpler model is sufficient, leading to the conclusion that k = 5 is the preferable choice.

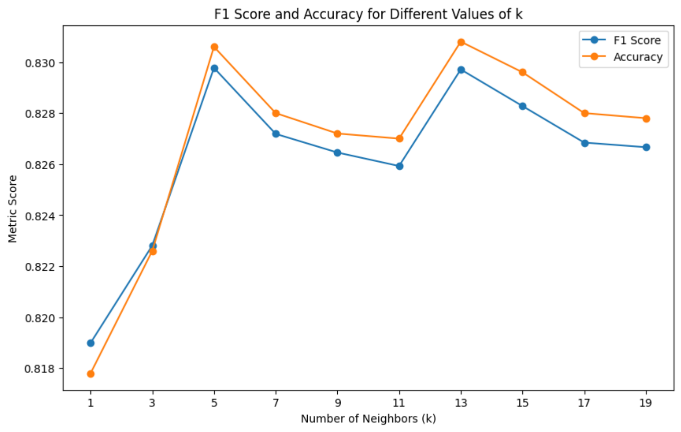


Figure 5: F1 Score and Accuracy for different values of k in KNN

4.5 Results of Decision Trees

We have trained the decision tree classifier from the scikit-learn library[2] with default settings, including no specified maximum depth, a minimum of 2 samples required to split a node, and a minimum of 2 samples for a leaf node. However, we modified the splitting criterion from 'gini' to 'entropy' as it yielded slightly improved results compared to the Gini criterion. We conducted tests on various datasets by testing them on unseen data and using k-fold cross-validation on the training data: one with 784 features representing the pixels, another with the pixels features reduced to 100 components using PCA, and a third with the same dataset reduced to 4 components through LDA. Here are the results:

Metrics	DT no PCA no LDA		DT PCA		DT LDA	
	Test	Cross-val	Test	Cross-val	Test	Cross-val
Precision	0,77	0,79	0,74	0,75	0,76	0,81
Recall	0,77	0,79	0,74	0,75	0,76	0,81
F1 Score	0,77	0,79	0,74	0,75	0,76	0,76
Accuracy	0,77	0,79	0,74	0,75	0,76	0,81

Table 3: Results DT default parameters, depth=26, min sample split=2, min samples leaf= 1

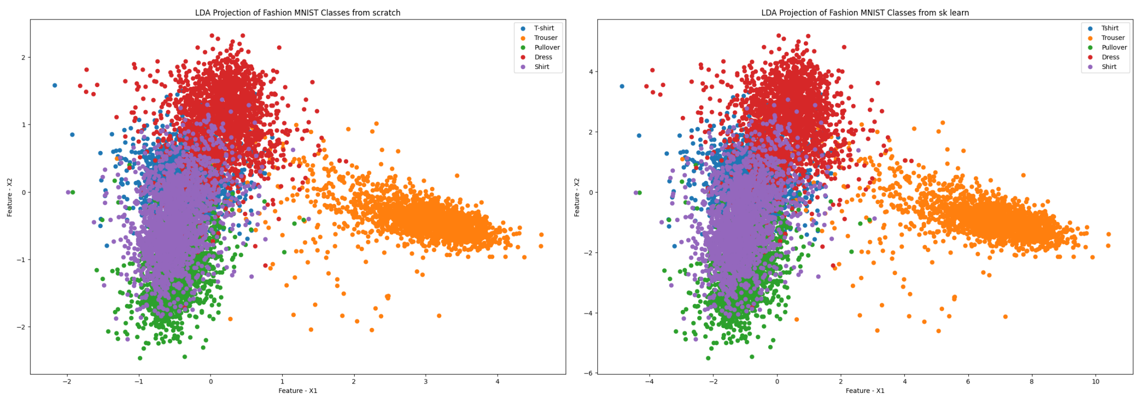
The model without any feature reduction performed almost the same as the model that used the 4 components through LDA. Therefore, we decided to tune the parameters of the model using LDA for better performance since the model is simpler and performs almost the same way. We have used Grid search to find the best combination of maximum depth, minimum samples split, and minimum samples leaf. **The Best Parameters for Decision Tree LDA:** maximum tree depth = 5, minimum samples leaf = 4, minimum samples split = 2 and criterion = 'entropy'

5 Discussion

5.1 Assertion of the correctness of our implementations

5.1.1 LDA implementation

To assess the correctness of our implementation of our Linear Discriminant Analysis as a dimensionality reduction method, we have compared it to the one implemented by the sklearn package. As seen below we have plotted a scatterplot of our own implementation of the LDA projection vectors applied to the dataset, and also instantiated and applied the sklearn LDA method to our dataset and created a similar scatterplot. Looking at the two scatterplots and comparing them, it is evident that they are if not completely, then extremely close to being the same. This comparison gives us enough confidence in our implementation being correct and returning the right results on our dataset.



Our LDA implementation sklearn LDA implementation

Figure 6: Scatterplots of LDA projection per class

5.1.2 Naive Bayes implementation

To assess the correctness of our Naive Bayes implementation we have opted into comparing it to a similar Naive Bayes implementation from a library, we have decided to use the same LDA 2 feature dimensionality reduction on the data before comparing the Naive Bayes'.

Table 4: Changes in Performance Metrics between sklearn Gaussian NB and our NB

Class	**Precision**	**Recall**	**F1 Score**	**Accuracy**
T-shirt	0.014701	-0.008	0.005963	0.0058
Trouser	0.003023	0.000	0.001466	0.0006
Pull-over	0.005437	-0.005	0.000884	0.0012
Dress	-0.008336	0.002	-0.003196	-0.0018
Shirt	-0.008435	0.021	0.013234	-0.0018
Average	0.001278	0.002	0.00367	0.0008

As seen in Table 4 the difference in how the Gaussian NB performs compared to our NB classifier it can be seen that changes in metrics are minor, which gives us a good indication that our Naive Bayes implementation has been done correctly, since it returns basically the same results as one from the well-established python library sklearn. The minor difference could be a result of us using the Gaussian Naive Bayes to compare with, since we have implemented our naive Bayes with histograms instead of a Gaussian distribution. Overall it looks though as our implementations have been done correctly and are returning the expected results.

5.2 Performance comparison of each method

5.2.1 Performance Comparison kNN

All three models- kNN without LDA or PCA, kNN with PCA a 100 components, and kNN with LDA 4 components exhibit good performance on both the test and cross-validation datasets, as indicated by high accuracy and F1 scores. The effect of Dimensionality Reduction performs comparably well or even slightly better than the other models in terms of accuracy and F1 score. This suggests that reducing the number of features using PCA did not significantly compromise model performance and may have even improved it. It achieves high performance on both the test and cross-validation datasets, suggesting good generalization to unseen data, while also benefiting from dimensionality reduction.
All the models as mentioned were trained with default parameters from sk-learn, meaning that the neighbours were five, k=5. After plotting the performance for different numbers of neighbors, we found that k = 5 gives the best results(k=15 gives the same result, but we opted to do a simpler model) as mentioned in Figure 5.

5.2.2 Performance Comparison DT

The Decision Tree model with LDA performs well in terms of cross-validation accuracy and test, suggesting good generalization to new data. While the test accuracy is slightly lower compared to the model without LDA, the overall performance and the ability to generalize to new data, we believe that that makes DT with LDA a potentially better choice in this scenario. Moreover, the Decision Tree model with LDA not only performs well in generalizing to new data but is also a better choice because of its simplicity. With just four features, it is a simpler model compared to the model with 784 features, making it a more practical option.
After finding that LDA with 4 features with entropy split criterion for DT gave us the best results, we decided to tune its parameters and improve its performance. This was done by tuning the max depth, the minimum sample leaf, and the minimum sample split by using Grid search. The table below indicates that by tuning the parameters slightly improved the performance of the model, but not significantly. Also based on the accuracy and the f1 score there is no sign of overfitting.

Metrics	DT LDA		Tuned DT LDA	
	Test	Cross-validation	Test	Cross-validation
F1 Score	0,76	0,76	0,80	0,86
Accuracy	0,76	0,81	0,80	0,86

Table 5: Default Decision Trees LDA vs Tuned DT LDA

5.2.3 Overall performance

Firstly, kNN with PCA, where k= 5 performs best from the kNN group. Secondly, DT fed with LDA 4 features, tuned to split criterion being entropy, max depth = 5, leaf samples

= 4, split samples = 2 did best from the the DT group. However, kNN fed with PCA 100 components with k= 5 is the winner, because it performs better compared to DT. It outperforms all the other classifiers with an accuracy of 0,83 and f1 score of 0,83 on the unseen data.

6 Conclusion

In conclusion, our report has undertaken a comprehensive exploration of various machine learning techniques applied to the dataset. We began by introducing Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) as feature reduction methods. The implementation of these methods involved visualizing the impact of principal components and assessing their effectiveness through scatter plots. Subsequently, we incorporated Naive Bayes as a classifier using the reduced features from LDA, achieving an overall average accuracy of 88.8%.

Moving forward, we explored k-Nearest Neighbors (kNN) with three iterations: utilizing the entire dataset, applying PCA with 100 components, and employing LDA with 4 features. The kNN classifiers were evaluated through k-fold cross-validation on the training data and tested on the unseen test data. We explored the ideal number of neighbors through systematic training with PCA, which exhibited superior performance. Analyzing accuracy and F1 score peaks revealed that k=5 provides comparable results to k=15. Hence, a simpler model suffices, making k=5 the preferred choice.

The decision tree classifier was then introduced, tested on datasets with 784 features, 100 PCA components, and 4 LDA components. We observed comparable performance between the model without feature reduction and the model with LDA-reduced features. Further optimization of the decision tree with LDA features was conducted using Grid Search, leading to improved parameter tuning with a 4% improvement on both F1 score and accuracy on the test data.

Overall, our extensive analysis and experimentation with diverse machine learning techniques contribute valuable insights into the classification of the dataset, highlighting the strengths and nuances of each approach.

7 References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms". In: *arXiv preprint arXiv:1708.07747* (2017).

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Jason Brownlee. *Principal Component Analysis for Dimensionality Reduction in Python - MachineLearningMastery.com — machinelearningmastery.com*. `https://machinelearningmastery.com/principal-components-analysis-for-dimensionality-reduction-in-python/`. [Accessed 03-01-2024].

[4] Gilbert Tanner. *Linear Discriminant Analysis (LDA) — ml-explained.com*. `https://ml-explained.com/blog/linear-discriminant-analysis-explained`. [Accessed 31-12-2023].

[5] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[6] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[7] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`. URL: `https://doi.org/10.1038/s41586-020-2649-2`.

[8] *K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks — geeksforgeeks.org*. `https://www.geeksforgeeks.org/k-nearest-neighbours/`. [Accessed 02-01-2024].

[9] *Decision Tree - GeeksforGeeks — geeksforgeeks.org*. `https://www.geeksforgeeks.org/decision-tree/`. [Accessed 02-01-2024].

[10] *1.10. Decision Trees — scikit-learn.org*. `https://scikit-learn.org/stable/modules/tree.html`. [Accessed 02-01-2024].

[11] Abhishek Jha Vladimir Lyashenko. *Cross-Validation in Machine Learning: How to Do It Right*. `https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right`. [Accessed 02-01-2024].

[12] Great Learning Team. *Hyperparameter Tuning with GridSearchCV — mygreatlearning.com*. `https://www.mygreatlearning.com/blog/gridsearchcv/`. [Accessed 02-01-2024].