# MNIST Fashion data Classification Task

CREATING 3 MACHINE LEARNING MODELS BY HAND TO FIND THE MOST ACCURATE

**Nicholas Stollmann**

# Table of Contents

# Introduction

The aim of this study is to find the best classifier to train a machine learning model that will be used to predict the content of pictures. By using the pixel breakdown of thousands of pictures, the model will attempt to classify each picture into its correct class. This report will focus on pictures of the following 10 clothing items: T-shirt/Top, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags and ankle boots. The aim of this model will be to train it using 30,000 pictures with assigned labels and then testing it on a further 5,000 pictures. Ideally, this model will be able to be fed any picture in the future and it will be able predict which of the 10 items listed above the picture is depicting.

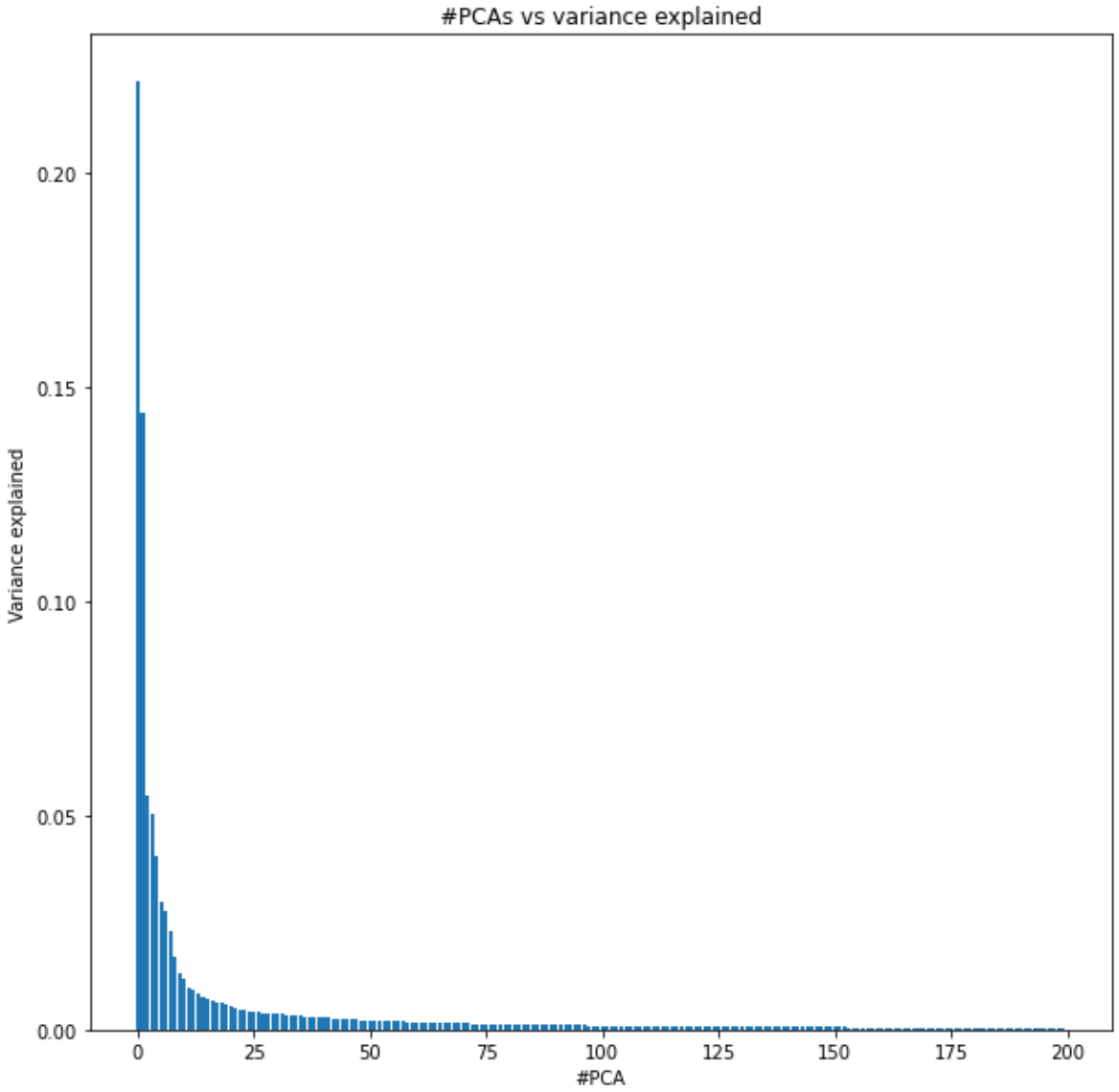
This is an important study because this practice could be used in many different aspects of life. It is teaching a computer to look at pictures and deduce what it is. Over the last few years machine learning and deep learning has had a big impact on many different fields of life, such as computer vision. Computer vision is a field that is always expanding, and this is the type of model that we are building in this study (Brand, 2019). Computer vision refers to the ability of computers to be able to identify patterns from pictures, and eventually the shape itself. This kind of model can be used in many different aspects of everyday life; for example, it could be used to filter though large files of pictures to better classify them, or even for social media software recognizing faces and people from pictures (Golemanova, 2019). With all the demand for improved facial recognition, using machine learning models to make these predictions is tool of great significance.

# Methods

The dataset used for this project has a training set of 30,000 examples, each with 784 (28 x 28 pixels) features and a test set of 5,000 examples. There were also 30,000 labels (from 0 to 10) supplied for the training set and 2,000 labels for the test set to calculate the accuracy of the model.

## Pre-processing

As this is a very large dataset, pre-processing was necessary to reduce the number of features. The method chosen to reduce the number of features was principle components analysis (PCA). To do this, the data needs to be standardized first. Therefore, the mean of each column was removed from each value in that column to normalize each value; this gave each column an average mean of 0. This normalisation is an important step, as it centres the data and allows us to calculate the eigenvalues accurately. Then, each value was divided by the standard deviation of each column to be standardized; this is not necessary in all cases but was performed for this dataset. The next step is calculating the covariance matrix of the transpose of the standardized matrix, this will explain how similar or dissimilar all the features are with each other. Calculating the eigenvalues of this covariance matrix then allows us to establish the most important PCA features. Arranging the eigenvalues and eigenvectors in descending order lists the PCA features from most to least importance. It is necessary to know how much of the variance is explained by the different PCA features so we know how many PCAs to use when recreating a matrix with a certain number of PCAs. This pre-processing step was performed outside the classifiers in order to see this graph and to understand how much of the data is explained by the different number of PCAs.




This graph shows how much of the variance in the data is explained by the top 200 PCAs. However, for the classifiers, the PCA is completed within the classifier, and the number of PCAs can be selected when first initiating a classifier. This is completed to give an additional hyperparameter that can be tweaked when trying to optimise the model.

The training data must then be split into a validation set to give us a dataset used for tuning the hyperparameters. Therefore, the training set was split randomly 80-20, with 20% in the validation set and the remaining 80% in the new training set. The method to do this was a random function that gave random numbers without replacement between 0 and the length of the training set. The first 80% of those random indices were then assigned to the training set, and the remaining 20% to the validation set - the labels were also split the same way with the same indices. This was a necessary step to give a reasonably large training set to train the model with, but then also a validation set to help turn the required parameters. Since the test set is not to be used for tuning of the hyperparameters, creating this validation set is an ideal solution. Since we have such a large dataset, the reduced training data for training is acceptable.

### K-nearest neighbours classifier

The first classifier trained for this project is a K-nearest neighbours (Knn) classifier. It is a simple algorithm that calculates the distances between each training point and the query point. Based on the selected K-value, the model then finds the K nearest points and calculates the distances between each of those points and the query point. Whichever class has the combined lowest weighted distances will then be chosen as the class for the query point. This is a non-parametric model and can run very slowly when dealing with large datasets, like this one in this study.

### Naïve Bayes classifier

The second classifier used for this study is a Naïve Bayes classifier. This classifier uses the likelihood of a point being part of a class given the features. These values are calculated using a gaussian distribution and the assumption that the data is from a normal distribution. This is another reason for the initial standardisation of the data. During the training stage, this algorithm separates all the points into the different classes. The algorithm then calculates the mean and standard deviation for each of those classes. When giving the model new values to test, it will calculate the probability of each point being part of each class using a gaussian probability function and the previously summarised mean and standard deviation. The model then returns the class for which the point has the largest probability.

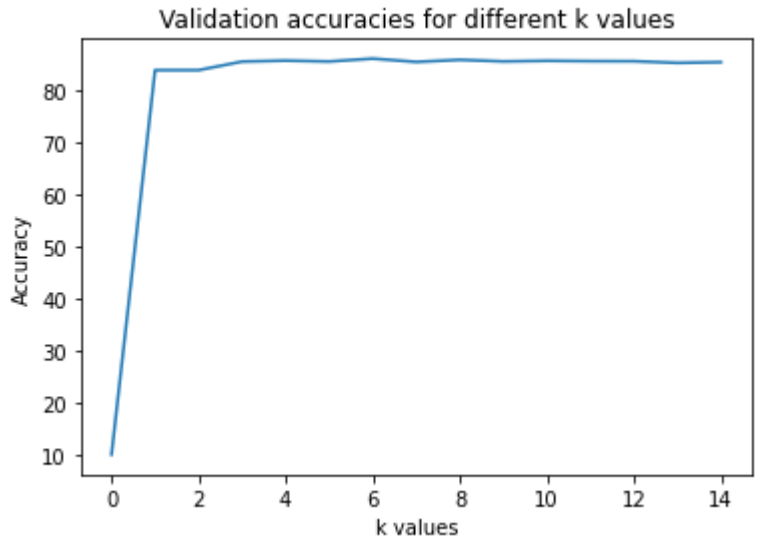
### Neural Network classifier

The third classifier trained on this dataset is a Neural Network classifier. This classifier works with one hidden layer between the input and output. The data is multiplied by a set of weights (initially randomly assigned) and is then passed through an activation function (ReLu function in this case). These values then make up the hidden layer, after which those values are again multiplied by another set of weights to give the output layer. The values must be passed through a SoftMax function so that the results become a probability between 0 and 1. The result is 10 different probabilities, each belonging to one of the classes. From that point, the model iterates backwards to change the weights as much as possible to minimise the loss (the difference between the right predictions and the actual classes). When those final weights have been calculated, we then pass the test data through the forward propagation state with the optimised weights to find the highest probability class. This neural network is a simple classifier, but still runs extremely slowly, which makes it very hard to tune and to optimise.

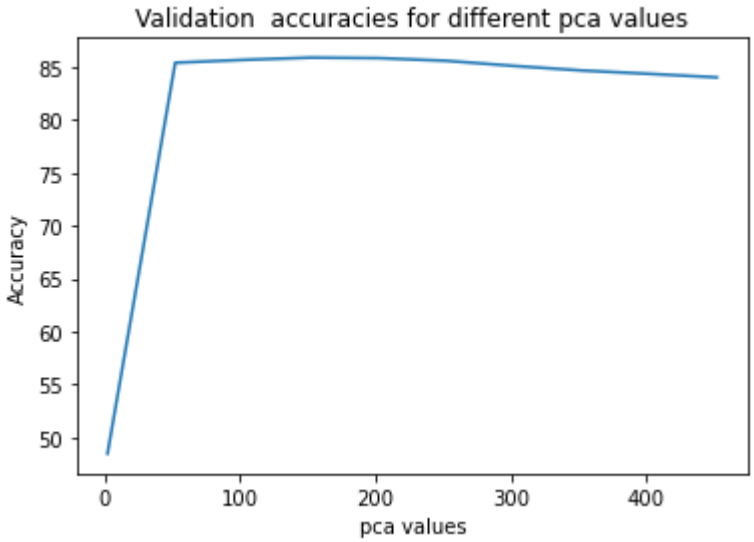
## Experiments and Results

### K-nearest neighbours classifier

The Knn classifier produced accurate results but did need some tuning of the hyperparameters. The three options available to tune this model are the value of k, the number of PCAs used and the distance metric used. To do this tuning, the model was trained with the reduced training data (80% of the original amount) and then the accuracy was calculated on the unseen validation data. Then a plot was created to show the accuracy of the model for different values of k on the validation set.




This graph shows that the accuracies plateau for values of k very early on. Using this graph, it was decided that a k-value of 5 would be used. It gives close to the maximum accuracy and it is not worth using larger values as the accuracy increase is minimal. The second hyperparameter that we could adjust for this model was the number of PCAs used. Since the PCA step is within the classifier this can be performed quite easily by running a loop over suitable values for the PCA and graphing the accuracies. Again, this was completed using the validation set.




This graph shows that the accuracy again plateaus at a rather small number of PCAs. By calculating the PCA value to which the largest accuracy corresponded (maxima of the graph above) it was decided that the best number of PCAs to use is 152.

The final parameter that can be adjusted is the distance metric, the two most common metrics were used to see if either of them would perform better. Below is a table of the accuracy scores for the Knn algorithm using the Euclidean distance and the Manhattan distance.

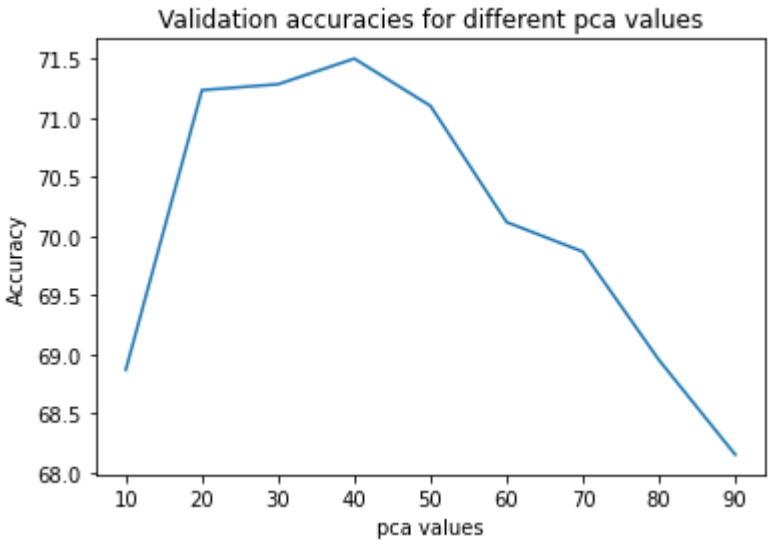
| Distance Type | Validation Accuracy | Test Accuracy |
|---|---|---|
| Euclidean | 85.783333 | 83.45 |
| Manhattan | 85.883333 | 83.60 |

Whilst the difference is quite small, the Manhattan model performs slightly better than the Euclidean model, so it will be the one implemented into the model by default. Note, that whilst the test accuracies are shown above, those are not used for tuning -only looking at the validation accuracies leads to changes in the hyperparameters.

Using the three parameters of k = 5, PCA = 152 and Manhattan distance, the classifier gave a maxed-out accuracy of 85.88% on the validation set. Finally, using these same parameters to calculate the accuracy on the 2,000 test values for which we have labels, gave an accuracy of 83.60%. This Knn classifier is reasonably accurate, which may be due to the type of data we are using. By comparing pixels, using a measure of distance like this classifier proved to be effective. The downside of this model is that it can run quite slow and it makes tuning the hyperparameters a tedious task that can take a long time.

## Naïve Bayes classifier

The Naïve Bayes classifier initially did not give very high accuracy on the validation model. There is not a lot of hyperparameter tuning that can be completed with a Naïve Bayes classifier; the only element we can experiment with changing is the number of PCAs to use. Similarly, to how the previous model was completed, this classifier was trained with the updated training data (80% of the initial training data) and the accuracy was measured on the validation set. Calculating the accuracies for different PCA values provided the graph below.




The accuracy initially increases with the number of PCAs, but then the drop-off is substantial and continues as we add more PCAs back in. Calculating the maxima of this graph on python gave an optimal number of 40 PCAs, which gave an accuracy on the validation set of 71.50%. Using this value of PCAs to test the accuracy on the test set returned an accuracy of 72.00%. Whilst the accuracy is substantially lower than that of the Knn classifier, this model has the advantage it runs much faster.

The reason this model may not be as accurate as that of the Knn classifier, is that the features of the data are not necessarily independent; since they are pictures, the pixels are affected by what is around them. There is not a lot of other parameters that can be adjusted for the Naïve Bayes classifier. Changing the type of distribution from a gaussian is an option, but that did not show to have much of an effect.

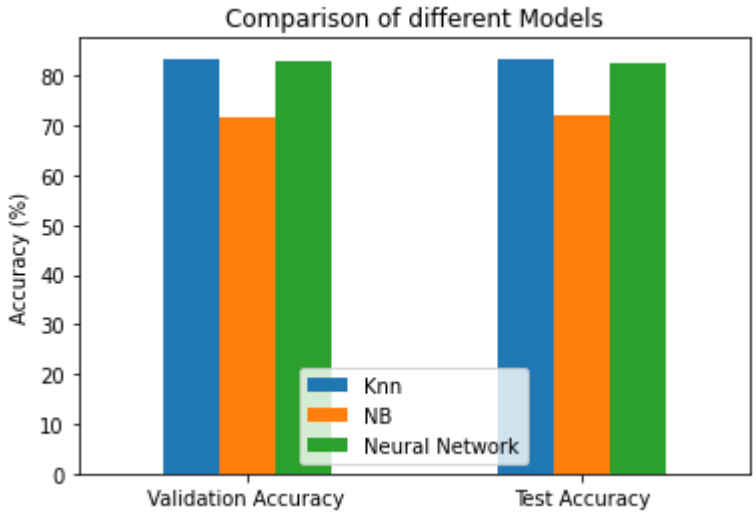
## Neural Network classifier

The final model trained on this data is a neural network model. This specific model is quite basic as it only has 1 hidden layer made up of only 15 neurons. The reasoning for this is that this algorithm is very computational intense, even at this basic level it took a very long time to run, up to 30 minutes at times. It did however give accuracies in the low 80% range, which is superior to the Naïve Bayes classifier. Some tuning of the hyperparameters was performed, however the time this model takes to run is a limitation when it comes to tuning. This model is giving reasonably good results for this data, so it should be considered when making models for image recognition. The neural network did not use PCA reduction as we can select the number of neutrons in the first layer, so the weights can be used to adjust the features. There is however one parameter that can be changed in this algorithm and that is the activation function. Either the sigmoid function or the relu function can be selected when initiating the network. It was found that the model performed much worse using the sigmoid function so it will now use the relu function by default.

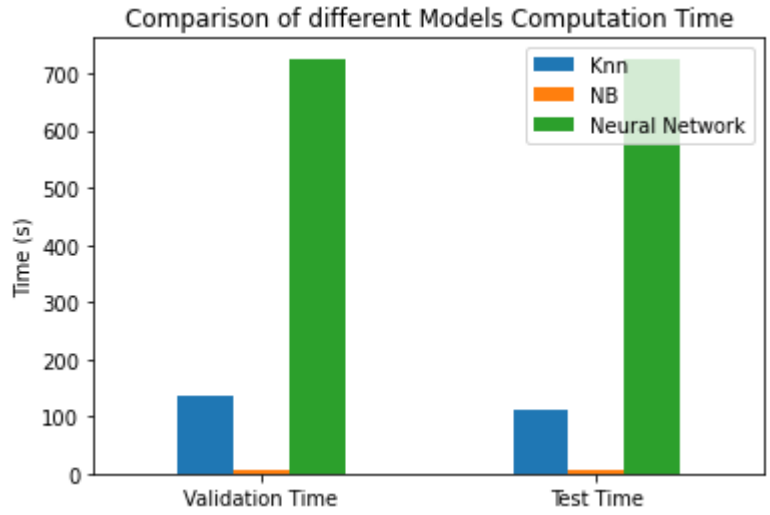
Trying to improve this model is not an easy task, and adding additional layers or parameters is something that was unsuccessfully attempted. An improvement on the model would also be to implement a bias term for each neuron, this would give another parameter that could be tweaked along with the weights to further improve accuracy (Ognjanovski, 2020). After trying to implement a bias term the model was not functioning at all so it is an aspect of the model that will need to be improved upon after further research. In the future I would like to try and add these additional features to the model to try and improve accuracy.

## Comparing the models

From a standpoint of accuracy, the best model was the Knn model and it seems to be the better of the three models when analysing the data for this study. Since the type of features we are using are pixel colours, using a measure of distance seems to be the best method to classify the different points. The Naïve Bayes classifier assumes independence in the features, however that is not the case for this example, which may be a reason why the model is performing poorly in comparison. Although the Naïve Bayes classifier does return a respectable accuracy of 72%, it would not be suitable in real world situations, as more than a quarter of pictures would be misclassified. The Neural Network also performed reasonably well, however it has very severe limitations. The graph below compares the accuracy results on the validation set and the partial test data between the models. Note, the test accuracy was not used for the tuning of parameters.




Comparing the time consumption of the algorithms does suggest that the Naïve Bayes classifier is the more efficient of the three. This is likely related to the way the Knn classifier works; it must calculate the distances between each training point and each test point, but with such a large dataset, this can take a long time. On the other hand, the Naïve Bayes classifier needs to only calculate average statistics of each class in the training set (mean and standard deviation) and uses a gaussian distribution to calculate the probability of each test point being in each class. The Neural Network was by the far the most time consuming; this is related to all the iterations it must do during the back-propagation phase -this combined with the large dataset means it takes a very long time to finish. This results in the Naïve Bayes classifier being much less computationally intensive. Training and producing results for the test set for the Knn classifier took 114 seconds, the Naïve Bayes classifier only took 4 seconds and the Neural Network took 831 seconds.




Whilst the time difference between the Naïve Bayes and Knn classifiers is not that impactful for this example, if we extrapolated those results out for much larger datasets, the difference in time would have a big impact on the efficiency of the model. This difference in time consumption also had a big impact during the training stage of the model when tuning the hyperparameters needed to be done, as running the models for different hyperparameters took a very long time.

With all factors being considered, the Knn model was the one chosen for this data. Whilst the Knn model is computationally more expensive and slower than the Naïve Bayes model, the difference in accuracy was too great to ignore. The Neural Network was also considered because of its relatively high accuracy, but the fact that it runs very slowly meant it was not as good of a model as the others.

*Comparison of the three models and their results:*

| | **Knn classifier** | **Naïve Bayes classifier** | **Neural Network** |
|---|---|---|---|
| **Validation accuracy** | 85.88% | 71.50% | 82.85% |
| **Test accuracy** | 83.60% | 72.00% | 82.50% |
| **Validation computation time** | 136 seconds | 5 seconds | 831 seconds |
| **Test computation time** | 114 seconds | 4 seconds | 831 seconds |

Below is the confusion matrix for the 2000 provided test labels using the optimal Knn classifier:

| Predicted Class / Actual Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Percentage correct |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 157 | 0 | 2 | 4 | 2 | 1 | 24 | 0 | 2 | 0 | 82.0 |
| **1** | 1 | 183 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 99.0 |
| **2** | 6 | 0 | 151 | 5 | 27 | 0 | 17 | 0 | 0 | 0 | 73.0 |
| **3** | 7 | 0 | 3 | 172 | 12 | 0 | 13 | 0 | 0 | 0 | 83.0 |
| **4** | 1 | 0 | 23 | 11 | 162 | 0 | 23 | 0 | 0 | 0 | 74.0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 168 | 0 | 15 | 1 | 6 | 88.0 |
| **6** | 26 | 0 | 16 | 5 | 16 | 0 | 127 | 0 | 0 | 0 | 67.0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 177 | 0 | 13 | 92.0 |
| **8** | 2 | 0 | 1 | 2 | 2 | 1 | 6 | 1 | 210 | 2 | 93.0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 185 | 96.0 |

The classifier performs well (80% +) for most items, however, it misclassified the 2, 4 and 6 labels by the biggest margin. Those three labels correspond to pullovers, coats and shirts, so it makes sense that they are the most misclassified. The three items are quite similar and when reducing the features even more, it is clear how the algorithm could misclassify them. Surprisingly the three footwear items performed very well using the classifier (5, 7 and 9), although, visually looking at the pictures of the shoes, it suggests that they are quite different from each other.

## Conclusion

Finding the optimal model for problems of image classification is not an easy task - since there are many kinds of models available, it is important to try several different models to find the optimal image classification one. As this dataset deals with features that describe different pixels of photos,

a model that measures similarity (distance) between points has shown to be a suitable choice. The neural network classifier showed promise, as at its most basic stage it gave an accuracy of more than 80%. It is also a very common algorithm used in real world scenarios for this kind of data analysis. However, since it was not the best performing algorithm out of the three, and it was very time intensive, it was not chosen as the final algorithm.

Whilst machine learning algorithms have in many ways improved society there are also issues associated with them. In some more serious implementations, misclassifications have a far greater impact. For instance, twitter's facial recognition algorithm was found to be racist or biased, it would crop our black faces in pictures much more often than white faces (Hern, 2020). These kind of unintended side effects of algorithms are often related to the biased training data used for the models which then in turn can give biased results (Heaven, 2020). Whilst this is not as much of an issue for this dataset specifically, it is highly related to these kinds of algorithms, so future models of this type should be more accurate with such potential side effects avoided as much as possible.

## References


Brand, J. (2019, October 29). Deep Learning for Image Recognition - KwadigoAI. Medium. https://medium.com/kwadigoai/deep-learning-for-image-recognition-1d612be00bbb

But what is a Neural Network? | Deep learning, chapter 1. (2017, October 5). [Video]. YouTube. https://www.youtube.com/watch?v=aircAruvnKk

Golemanova, R. (2019, December 16). The Top 5 Uses of Image Recognition. Imagga Blog. https://imagga.com/blog/the-top-5-uses-of-image-recognition/#:%7E:text=This%20is%20a%20machine%20learning,formulate%20relevant%20categories%20and%20tags

Heaven, Will Douglas. "Predictive Policing Algorithms Are Racist. They Need to Be Dismantled." MIT Technology Review, MIT Technology Review, 17 July 2020, www.technologyreview.com/2020/07/17/1005396/predictive-policing-algorithms-racist-dismantled-machine-learning-bias-criminal-justice/

Neural Networks from Scratch - P.5 Hidden Layer Activation Functions. (2020, May 14). [Video]. YouTube. https://www.youtube.com/watch?v=gmjzbpSVY1A

Ognjanovski, G. (2020, June 07). Everything you need to know about Neural Networks and Backpropagation - Machine Learning Made Easy... Retrieved October 19, 2020, from https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a