# Machine Learning with MNIST Fashion Dataset

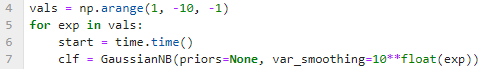
The objective of this assignment is to use various classification algorithms to identify clothing items from pictures using the fashion data from MNIST (<https://github.com/zalandoresearch/fashion-mnist>). The Gradient Boosted Decision trees yielded the best accuracy score of the algorithms that were evaluated with 88.76%, and it only took 76.4 seconds to train the model. Table 1 below shows model results for all three classifiers tested.

**Table 1: Classifier Models and Results**

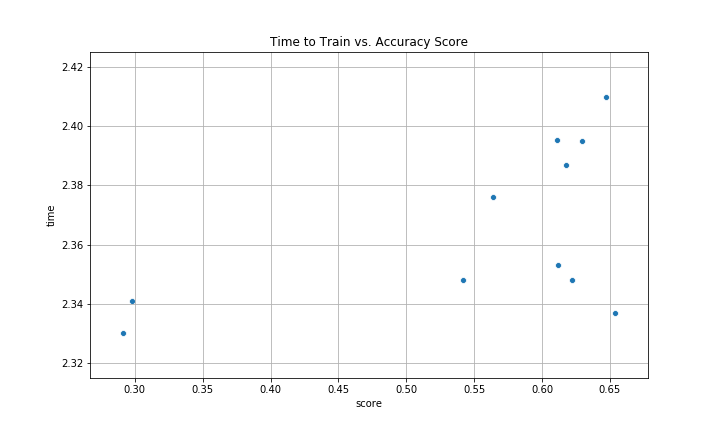
|  |  |  |
| --- | --- | --- |
| Classifier | Best Accuracy Score | Time to Train Model |
| Naïve Bayes  *sklearn.naive\_bayes.GaussianNB* | 65.35% | 2.4 seconds |
| Multilayer Perceptron *sklearn.neural\_network.MLPClassifier* | 88.63% | 550.8 seconds |
| Gradient Boosted Trees  *XGBoost* | 88.76% | 77.5 seconds |

# Naïve Bayes

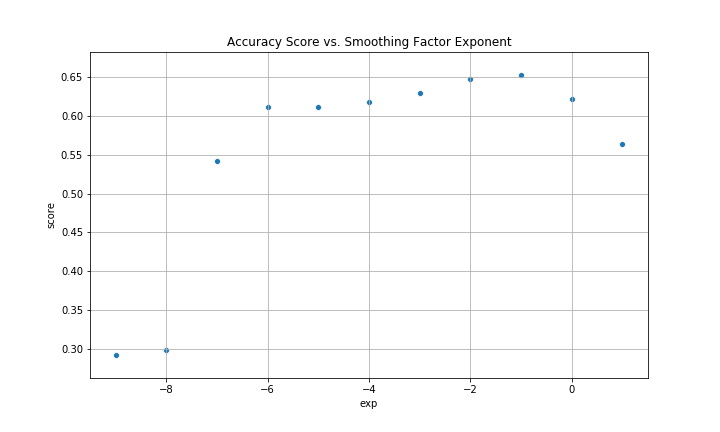
The first classifier built was a Naïve Bayes classifier, using the GaussianNB classifer from sklearn. This classifier is very fast as it was able to fit a model to the training data in less than 2.5 seconds for several iterations of smoothing.



That said, it did not achieve very good accuracy scores on the test dataset.



**Figure 1: Naïve Bayes Classifier, Training Time vs. Accuracy Score**

****

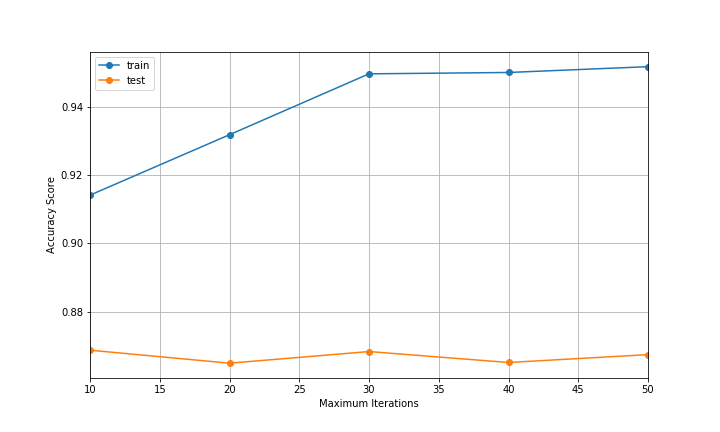
**Figure 2: Naïve Bayes Classifier, Accuracy Score vs. Smoothing Factor Exponent**

Overall, the best score on the test data was just over 65% accurate, which is not very good. The Naïve Bayes classifier performs best when the features follow the normal distribution, so additional transformations on the inputs should be considered to improve performance.

# Multilayer Perceptron

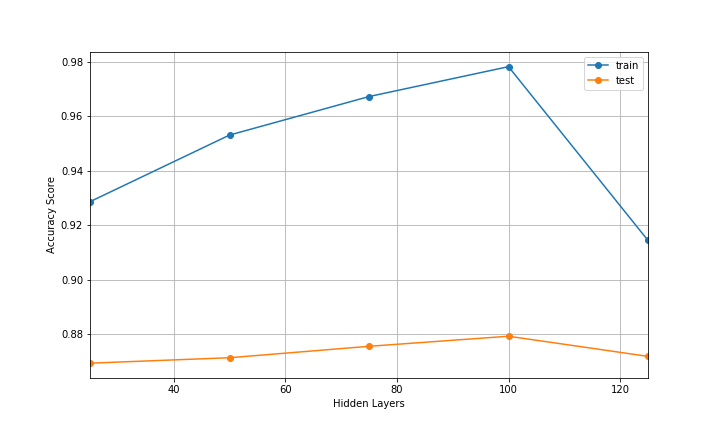
Next we used a multilayer perceptron classifier from the sklearn library. For this classifier, there are several different hyper parameters, or inputs that determine how the classifier builds the model with the training data. A few of these hyper parameters were explored to determine their impact to the accuracy of the model.

The first parameter evaluated was the maximum number of iterations. Figure 3 below shows that as the maximum number of iterations increases, the accuracy of the model also increases, though with the default inputs for other the other model parameters the accuracy is levelling off after only 30 or 40 iterations. It is also interesting to note that the accuracy on the test data doesn’t appear to follow the accuracy with the training data. This suggests we need to tune other model parameters to better match the accuracy from the training data.



**Figure 3: MLP Classifier, Accuracy vs. Iterations**

Next, we varied the size of the hidden layers in the MLP classifier while running each model out to 50 iterations. In Figure 4 below, we see that more hidden layers are generating better scores on the training data up to about 100, where we see a steep drop off in the accuracy above 100. We are getting better accuracy with these models than with the default inputs and it is starting to follow the same shape as the accuracy in the training data, but the peak accuracy is still only about 88% for the test data.



**Figure 4: MLP Classifier, Accuracy vs. Hidden Layer Size**

Further experimentation with the input parameters to the MLPClassifier did not yield significantly better results. The highest achieved accuracy was 88.63%, and the model took 550.8 seconds to train. The model input parameters are specified below in Table 2.

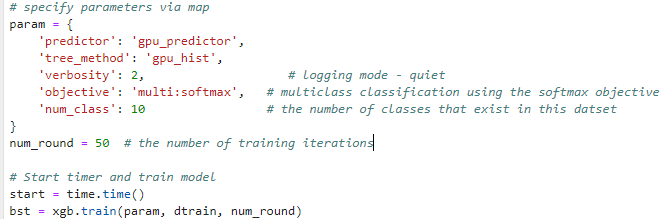
**Table 2: MLP Classifier Input Parameters**

|  |  |
| --- | --- |
| Model Parameter | Input |
| hidden\_layer\_sizes | (175, 75) |
| max\_iter | 250 |
| activation | ‘logistic’ |
| solver | ‘sgd’ |
| random\_state | True |
| learning\_rate\_init | 0.05 |
| learning\_rate | ‘adaptive’ |

# Gradient Boosted Trees

The third method tested was Gradient Boosted decision trees using the XGBoost library. XGBoost takes advantage of GPU acceleration to create many decision trees in parallel, and the “boost” comes from weeding out trees that are less accurate and keeping those that are more accurate. Thus, you achieve a sort of “wisdom of the crowd” model from aggregating the predictions of several decision trees.

By using mostly default inputs to the XGBoost classifier, and only setting inputs to determine the max number of classes in the data to 10, the max iterations to 50, and ensuring that GPU optimization was active, we achieved an accuracy score of 88.76% in only 77.54 seconds. This accuracy just barely beat out the MLPClassifier in terms of accuracy, but in an order of magnitude less time.



**Figure 4: XGBoost Classifier Basic Inputs**