An Investigation of Classification Methods for Fashion-MNIST

Austin Woo (ID: 29005890)

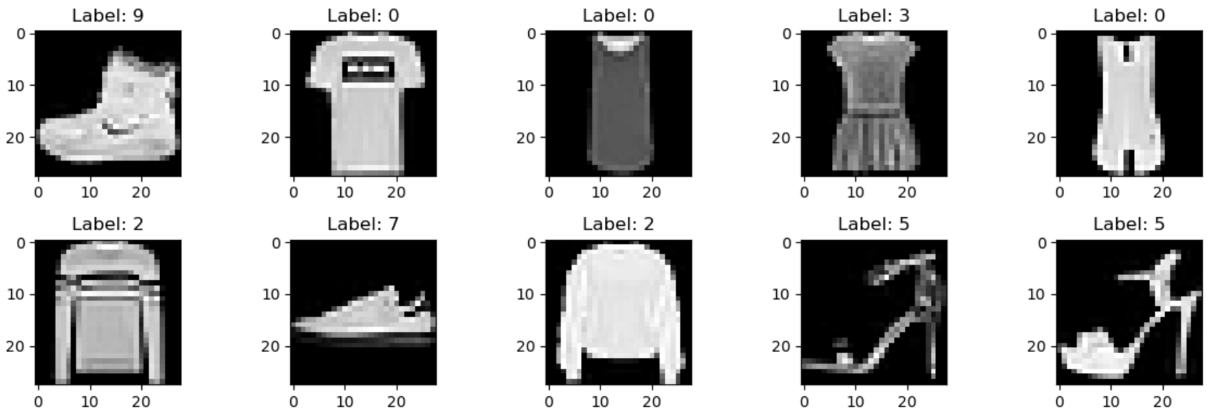
Laura Valko (ID: 66011613)

Nathan Gin (ID: 67117388)

1. Summary

An exploration of a variety of classifiers applied on the Fashion-MNIST dataset. The specific classification methods are as follows: K-Nearest Neighbors, Logistic Classifier, a Neural Network, and a Random Forest. We observe that the classifiers tend to make the same type of errors (visualized through confusion matrices) and analyze learning curves and prediction time. However, according to our results we found the Neural Networks classifier to perform the best because of its high accuracy whilst sustaining an efficient performance runtime.

2. Data Description

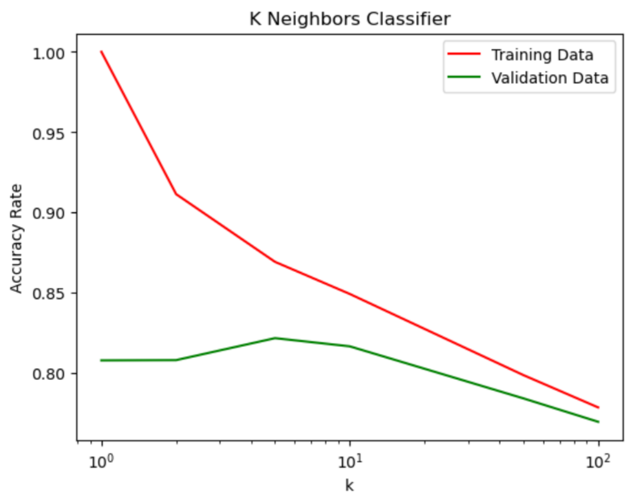


The Fashion-MNIST dataset consists of 70,000 examples split up with 6/7 making up the training set and 1/7 making up the testing set. The examples are 28x28 pixel grayscale images (resulting in 784 features) of 10 classes of clothes. The class labels are ordered from 0-9 as follows: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

3. Classifiers

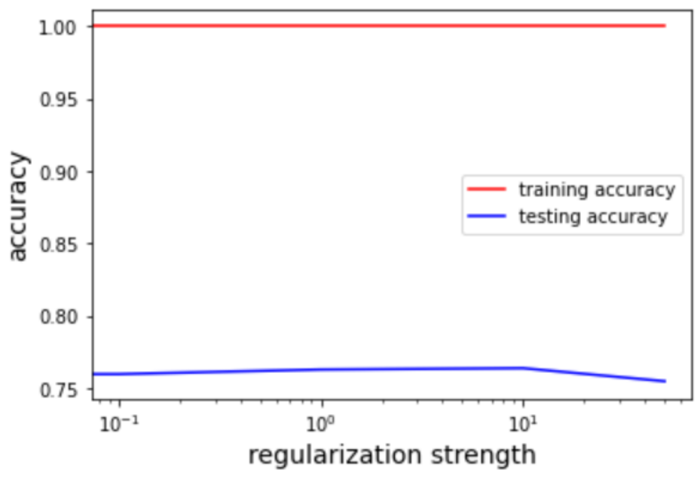
3i. KNN Classification

KNN classification takes in a datapoint and looks at the k points closest to it, assigning it the prediction label that is the majority of those k points. To choose the hyperparameter of k, an analysis of the testing error on the following values: 1, 2, 5, 10, 50, 100 with the scikit-learn implementation was conducted. The graph to the right displays this data.

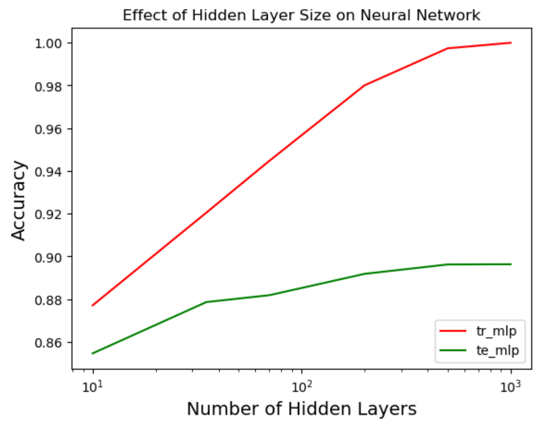


3ii. Logistic Classifier

Essentially, a logistic classifier provides an estimate of the probability that a datapoint receives a specific label. Another way to understand logistic classifiers is that they are basically neural networks, just without the use of any hidden layers (in addition, the sigmoid activation function is used in logistic classification). To choose my hyperparameters, tests similar to the one on the right (using different regularization strengths) were performed.

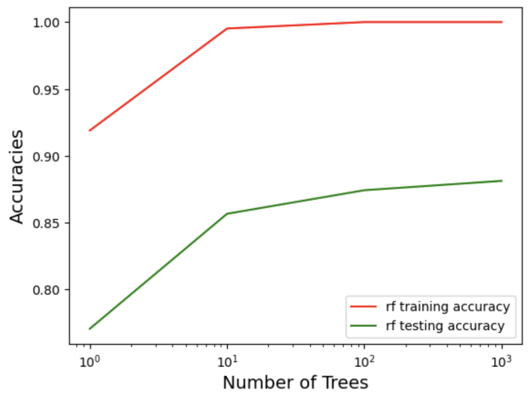


3iii. Neural Networks (MLP Classification)



Neural Networks (using MLP Classification) uses an algorithm that trains the data using backpropagation. In the Multilayer Perceptron, the network consists of the input layer, hidden layers, and output layer. When taking in the data, the goal of the neural network is to map an input to some label through the composition of multiple functions of layers. During the process of training the MLP model, the weights and bias are calculated into the input to produce the most accurate prediction before calculating loss and beginning backpropagation. Hidden layers are unique and important to the functionality of neural networks. Increasing the number of hidden layers often increases the accuracy but comes at the cost of complexity and overfitting. As shown in the corresponding graph, the training accuracy continues to increase which is expected since the model is continually getting more fitted to the data. Though the testing data also increases in accuracy, we can see the accuracy starting to converge as the number of hidden layers gets near 10^2. For layer sizes > 10^2, the cost of complexity is not worth the minimal increase in accuracy (begins to overfit the data since training data goes to near 100% accuracy but testing data converges around 89%). For this reason, 70 hidden layers were used in the MLP classification of the Fashion Dataset because of its balance of high accuracy and low complexity.

3iv. Random Forest Classification



The random forest classifier creates multiple decision trees and finds the average of them all to find the best prediction labels from the training data to use on the testing data. A single decision tree normally has high variance so taking the average of a forest of decision trees can help to return more accurate results. The main things that were tested were how the number of trees, criterion, and bootstrapping affects the accuracy. In this case having 100 trees was nearly equivalent to having 1000 trees but significantly more than just one decision tree. Additionally, bootstrapping was tested and found to be more accurate when set to “False”. This means that instead of taking random portions of the data to fit each decision tree it uses all of the data for every one. The random portion of random forest in this case would be decided by taking random subsets of features during each split of the decision trees. The last large factor was the criterion. Entropy was the best function to measure the quality of splits in this example. The default values in the remaining hyperparameters were all optimal in this case.

4. Experimental Setup

We fit our classifiers on the entire training set to ensure maximum accuracy. We adjusted our hyperparameters in an attempt to minimize the error rate by testing various inputs and keeping track of their corresponding accuracies. The specific hyperparameters we chose are as follows.

4i. KNN Classification

k = 7

4ii. Logistic Classifier

Fit Intercept = True, **Penalty** = ‘l1’ **Solver** = ‘liblinear’, **C** (regularization) = 1

4iii. Neural Networks (MLP Classification)

Hidden Layers = 70, **Activation/Solver** = relu/adam, **Learning Rate** = 0.00005 (constant), **Batch Size** = 360

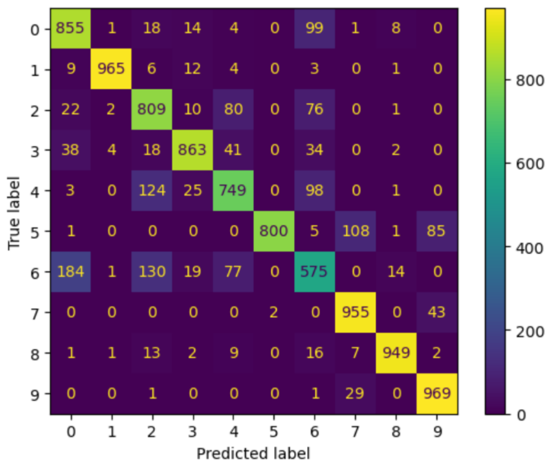
4iv. Random Forest Classification

N Estimators = 1000, **Criterion** = ‘entropy’, **Bootstrap** = False

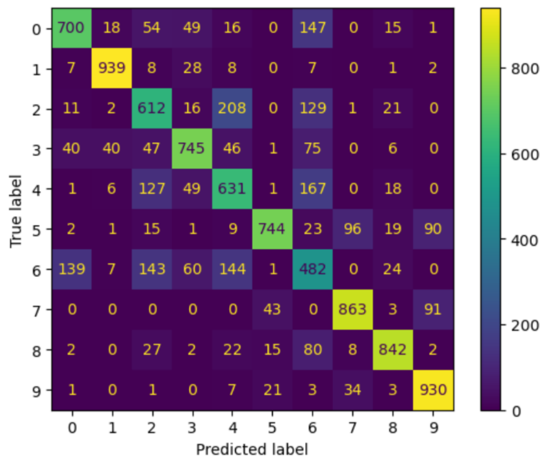
5. Experimental Results

5i. Confusion Matrices

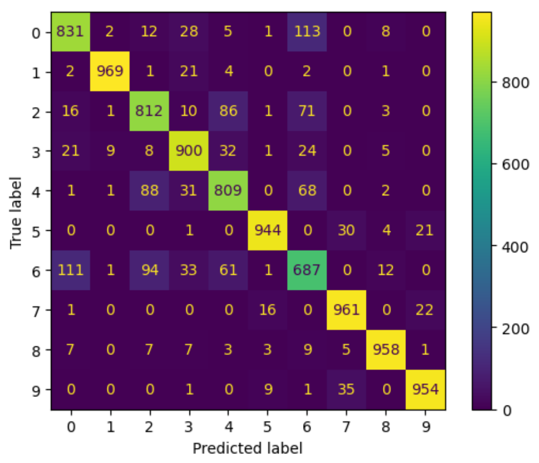
The confusion matrices below show the error that each classifier makes separated by labels. The diagonal colored line represents the correct predictions that the classifiers made for the specified labels.

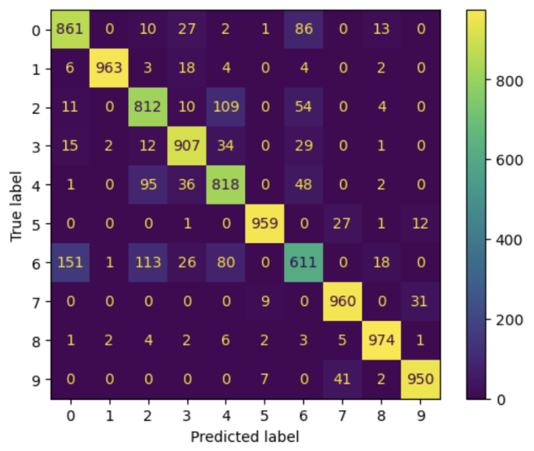


KNN Classifier



Logistic Classifier



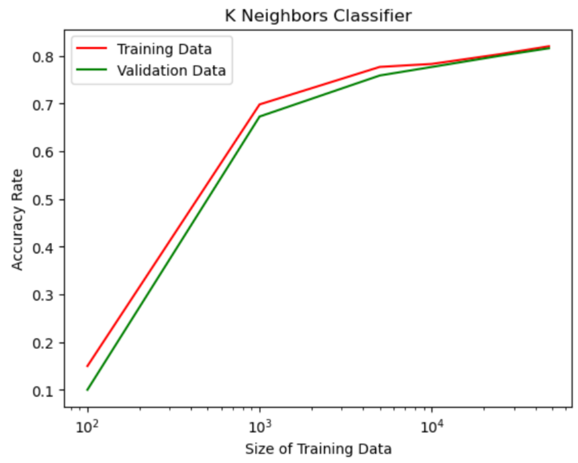


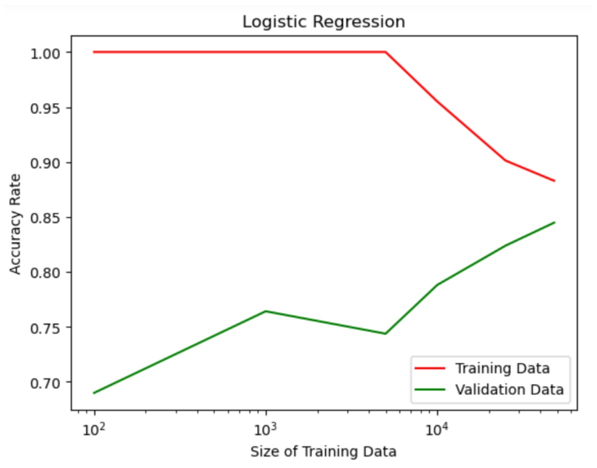
Neural Network- MLP Classifier

Random Forest Classifier

5ii. Learning Curves

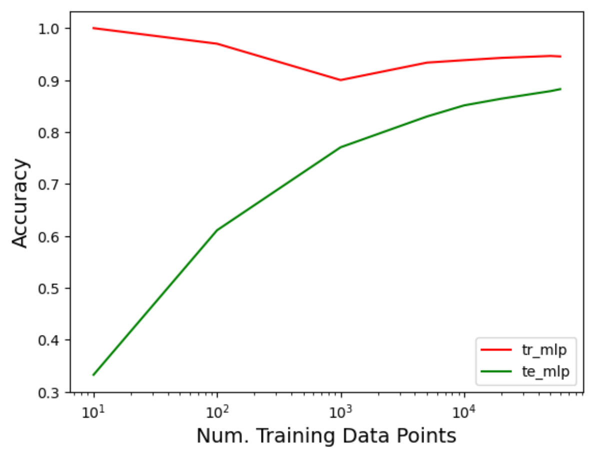
The graphs below represent the change in accuracy of each classifier as we increase the size of the training data.

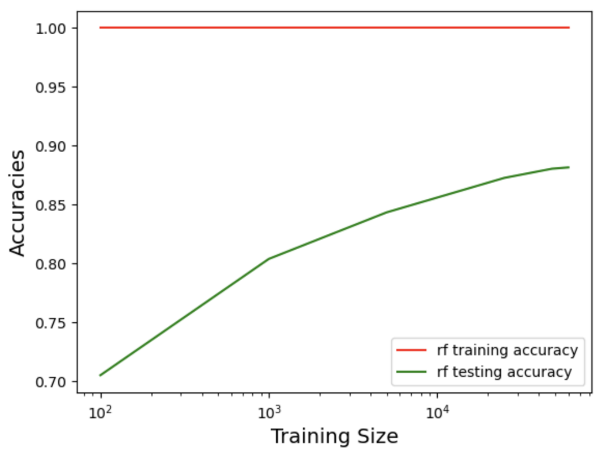




KNN Classifier

Logistic Classifier





Neural Network- MLP Classifier

Random Forest Classifier

5ii. Prediction Speed of Classifiers

The classifiers performed on their predictions (where test size = 10,000) as follows (in decreasing order):

Logistic Classifier	→	CPU Time Total: 141 ms
Neural Networks	→	CPU Time Total: 384 ms
Random Forest	→	CPU Time Total: 420 ms
KNN Classifier	→	CPU Time Total: 39.6 sec

6. Insights

The most noticeable result that we found was the common errors displayed in the confusion matrices. These errors make sense intuitively: they are mostly between items of the same type (such as shirt vs. pullover or sandals vs. sneakers). For our learning curves, a larger training set resulted in better test accuracy for all classifiers but training accuracy varied erratically. In terms of prediction times, the logistic classifier was the quickest and KNN Classification was by far the least efficient. This is because every prediction performed for KNN necessitates the calculation of the K nearest neighbors for that specific datapoint.

7. Contributions

The team collaborated on the majority of the report, with individual completion as follows:

-Austin Woo worked on Logistic Classification and KNN Classification.

-Laura Valko worked on Neural Networks (MLP Classification).

-Nathan Gin worked on Random Forest Classification.