Data Classification of MNIST Fashion Data

Leoul K. Berhanu

University of Maryland, College Park

STAT 426

Vince Lyzinski

April 26, 2022

Introduction

The purpose of this project was to highlight how different classifier models predicted data from a given data set. The data set was the MNIST Fashion Dataset, and it consisted of 70,000 images of articles of clothing, where 60,000 were a part of the training set and the remaining 10,000 were a part of the test set. The images were 28x28 pixels, and were organized into 10 different categories of clothing. This data set provides an alternative to the popular MNIST data set, which consists of thousands of images of handwritten digits, and the reason people are gravitating away from this data set is because it is overused and some machine learning models have too easy of a time predicting it.

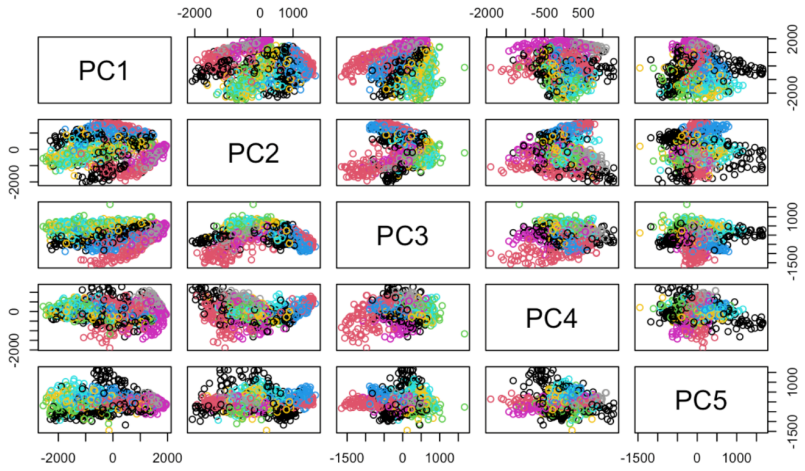
While the specifics will be covered in the later portions of this paper, the code is started off by reducing the dimensionality of the data using PCA. On this now reduced dimension data, a number of classifiers are run on it, starting with K nearest neighbors. After comparing across classifiers, support vector machines are covered - The first SVM is linear while the other is with the radial base kernel. Continued, Linear Discriminant Analysis (LDA) is then run on the data to predict the labels on the test set and also see what errors it yields. Lastly deep learning is covered by running a Deep Neural Network on the data.

This document will focus on analyzing the results of these classifiers (how well they performed, the amount of time they took to run) and why we performed these techniques on our data. By the end of the document, the reader should have a better understanding of a machine learning workflow and which classifiers worked the best and were easiest to implement.

Part 1: PCA

The first task consisted of reducing the dimensionality of the data. This was achieved by PCA. Principal Components Analysis is a method that takes high dimensional data and represents it in a lower dimension. PCA attempts to find this representation while also maximizing the variance of the data. It does this because it wants to make sure it isn't pulling identical features across different dimensions, and each of these dimensions are represented by a linear combination of different features. One may ask why reducing the dimensionality of data is even necessary. For one, working in a lower dimension means your machine will run a lot faster since it doesn't have to tend to as many features as before. Additionally, working in high dimensions means a lot of features, which can lead to overfitting, which would hinder your prediction accuracy.

The code began by loading in the data set and running PCA on the training set. It found that in order to capture 90% of the total variation of the data, it needed to be represented in 71 dimensions. We then looked at the pairs of Principal Components (which were colored by their class label) and saw this:



You can see that the pairs were bunched together, but if you focus on just the purple pairs, you can see that they transition from right to left as you move from the pc1 column to the pc5 column. Continued, we ran the same centering and projection onto the test data, and found that it was also embedded in 71 dimensions.

Part 2: K-NEAREST NEIGHBORS

For the second portion of the project, K-fold cross Validation was used to train the K-Nearest Neighbors Algorithm. K Nearest neighbors is a machine learning algorithm that attempts to classify a point by comparing that data point to the k nearest points around it from the test set. To describe K-fold cross validation, cross validation should first be explained. Cross validation is a way to see how well a model works by resampling data. It is an ideal way to see how accurate the model is, and lets the experimenter see things like if their model overfits the data. K-fold cross validation is a type of CV where the data is split into k folds, and the error is decided by one comparing one validation set to the remaining folds. Here a 5 and 10 fold CV was conducted, on both a full and reduced dimension dataset.

KNN: 5 CV

```
   user  system elapsed
 67.978   2.510  71.102
```

On the KNN trained by 5-fold CV, it took the machine 71.102 seconds to process the data. This long run time can be attributed to the fact that it was run on the full 784 dimensions, which gave the machine more features to run through. The optimal model was the one that yielded the highest accuracy, and in this case the highest accuracy (0.7707) was achieved with the hyperparameter k = 5.

KNN: 10 CV

a 10-fold CV was then run on the machine. This took 76.351 seconds to run, an addition of 5 seconds, but this makes sense because there were more folds for the validation set to be compared to. The model achieved its highest accuracy (0.7807) on the hyperparameter k = 5 as well. I would say the reason for the higher accuracy is because the additional folds provide the validation set with a better fit.

KNN: 5 CV (PCA)

KNN on 5-fold CV was run again, but this time on a data set that had its dimensionality reduced by PCA. this led to a runtime of 5.895 (significantly faster) and the highest accuracy (0.777) was achieved at k =7.

KNN: 10 CV (PCA)

On the 10-fold CV, it had a run time of 7.782, and this makes sense since there are more folds to compare the validating set to. The highest accuracy (0.7818) was achieved at k =9.

In terms of 5-fold, the unprojected data had an accuracy of 0.77, while the projected data had an accuracy of 0.78. This is a difference of ~0.01, which is not tremendously significant. In terms of the 10-fold, the accuracies of the unprojected versus projected data were 0.768 and 0.783, respectively. While this is a larger difference in comparison to 5 fold, I would also say this was not a large difference in accuracy between the two data types.

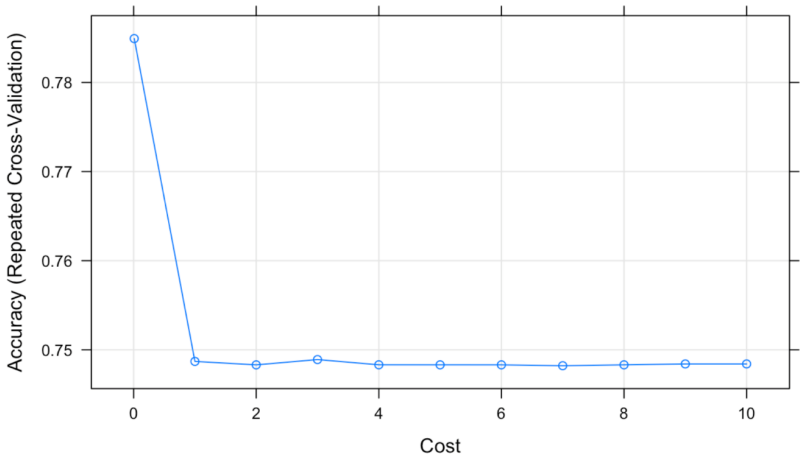
In general, Cross Validation works to estimate how well a data set fits a model that was trained by data that is independent of the data that it is being compared to. In this case, the predicted accuracies are estimating how well our KNN algorithms will predict the article of clothing based on the number of folds and dimensionality it was trained by.

In terms of the optimal values, the values were the same in the 784 dimensionality but varied slightly in the reduced dimensionality.

Part 3: SUPPORT VECTOR MACHINES

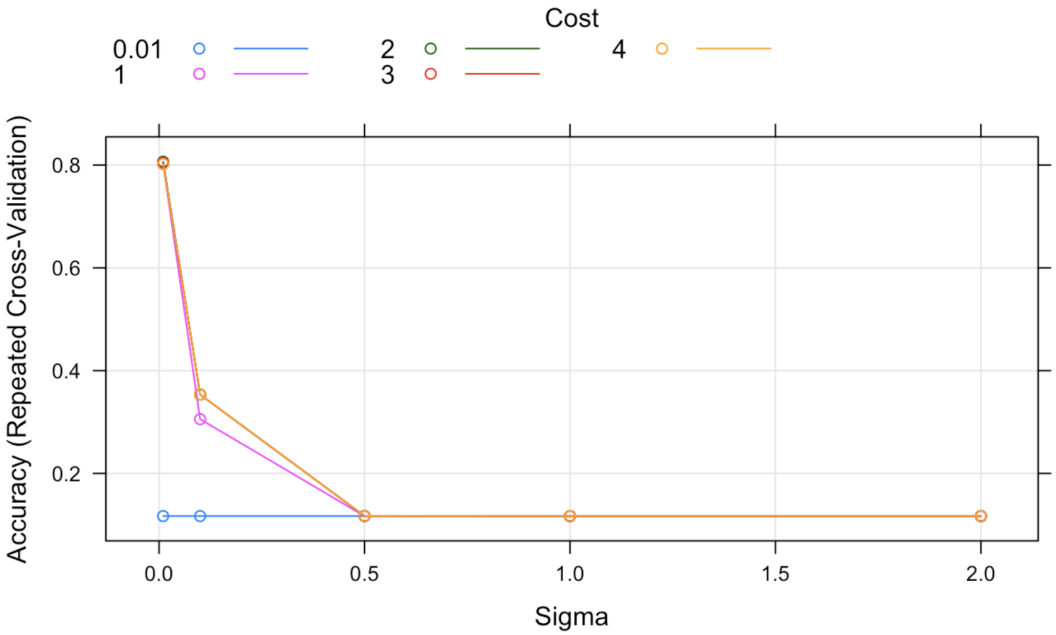
Support Vector Machine is a machine learning algorithm that looks to divide space between two sets of data points to make it so that the gap between the line and types of data points is as large as possible. The pros of Support Vector Machines is that they work well in high dimensionality - which is important in this case - because they can control the feature space such that they can divide the points by a hyperplane after changing the dimensionality to its preferred liking. Our two SVMs are a linear and radial kernel, and the good thing about the radial kernel is that it focuses on points closest to the dividing hyperplane, which eliminates the worries of having an outlier negatively affect our data.

We first ran a linear SVM. we defined an empty list of size 10 named C and labeled the first element to be 0.01. This was done because if we had used zero, R would not have recognized a support vector and would have kept looking for one indefinitely.



This was the plot of the linear SVM. our optimal accuracy was found at Cost = 0.01, and this value was ~0.785. In terms of how well the linear SVM did, it yielded an accuracy of 0.796, which is quite high in comparison to the accuracies we have been seeing up until this point.

For the radial SVM, 5 models needed to be fit for the 5-fold cross validation with 10 repeats. These values were $sigma = \{0.01, 0.10, 0.5, 1.0, 2.0\}$.



Once plotted, you can see that 0.01 remained flat for the duration of the graph while 3 and 1 experienced a sharp decline before flattening along with 1. When searching for the optimal model, it was found that the final values used for the model were sigma = 0.01 and C = 2. In terms of the performance of the radial SVM, this yielded the highest one yet with 0.828. I believe that the fact that the radial SVM only accounted for the most local points of the data set helped it achieve this high of an accuracy.

Part 4: LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis is a linear classifier that works especially well when there are only two classes that need to be divided. It also deals with dimensionality reduction, and works by creating a line such that the line does its best at dividing the two classes apart as best as it can. Here, we used LDA to try to classify between our 10 classes, and while LDA isn't as good at multiple class divisions, you will see that it does a decent job in this case.

In terms of performance, it had a reported accuracy of 0.791. This was actually better than K-NN classification accuracy, but it did not perform as well as the SVMS did.

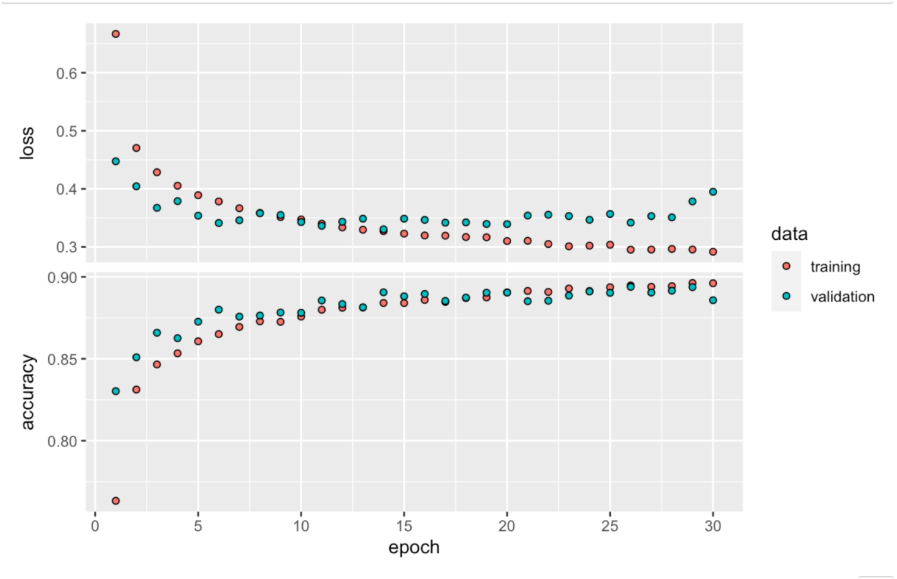
	y.test									
lda.class	0	1	2	3	4	5	6	7	8	9
0	76	0	1	3	0	0	19	0	2	0
1	0	95	0	0	0	0	0	0	0	0
2	2	1	65	1	9	0	9	0	0	0
3	6	8	1	80	7	0	8	0	1	0
4	0	1	18	1	63	0	8	0	1	0
5	0	0	0	0	0	87	0	17	4	6
6	5	1	13	4	16	0	55	0	3	1
7	0	0	0	0	0	5	0	91	1	5
8	0	0	1	1	1	2	1	0	94	1
9	0	0	0	0	0	5	0	9	0	85

Here is a matrix of the lda model predictions versus the actual classifications. The cases where the row and column number match ((0,0),(1,1), etc.) are the cases where it predicted correctly, and any value outside of that that is greater than 0 was an error.

Overall, The linear and radial SVMs achieved the best estimated testing accuracy based on CV *and* the best testing accuracy. I think they agree because since the SVMs work best in high dimensionality in comparison to the other classifiers, it was clear that they were going to do the best in terms of accuracy since our data set was in high dimensions.

Part 5: DEEP LEARNING:

To start, a Neural network is a function that takes an input vector **x** and builds a nonlinear function to return the response Y (This definition was taken from my lecture notes). Using tensorflow, a deep neural network was implemented to try to classify the data set, and the results of this were quite astounding.



From the graph above, you can see that at an ideal epoch value (~15) we were able to maximize accuracy while minimizing loss. The accuracy of the DNN was far superior in comparison to any other classifier, it had one of 0.88. I believe the reason for this outstanding performance is because between the input and hidden layer, we were able to create a model that covered and accounted for a lot of the variability of the data set, therefore making better predictions. To improve it, I would play with the batch_size and epochs variables to see what would yield even better results for me.

When I reran the code with the reduced data points, it returned an accuracy of 0.79. This means it did not yield a better accuracy result in a reduced dimensionality setting. I believe this is because the Net had less features to work with, so in most cases where classifiers overfit in higher dimensionality, this one actually does better at predicting in higher dimensionality since it can handle it better with its layers.

References

1. Lecture Slides by Vince Lyzinski
2. An Introduction to Statistical Learning (second edition)
3. https://github.com/zalandoresearch/fashion-mnist
4. https://www.kaggle.com/datasets/zalando-research/fashionmnist
5. https://medium.com/apprentice-journal/pca-application-in-machine-learning-4827c07a61db
6. https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb