Fashion Mnist Data Classification Report

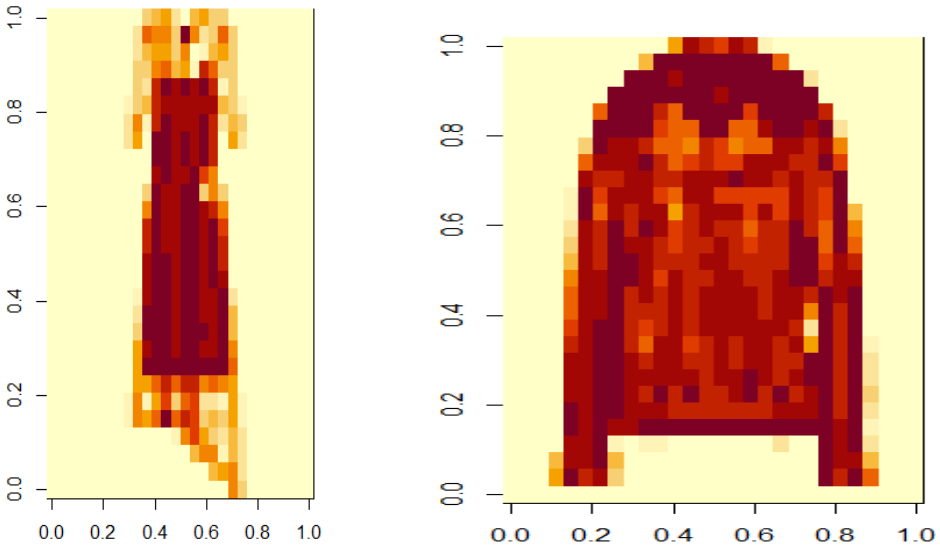
Hussein Abdulmohsin

Professor Vince Lyzinski

STAT426

Introduction

This report involves using various classification methods in R for predicting clothing item types in the Fashion MNist dataset. This dataset includes seventy thousand entries of low-resolution images, each of which would be difficult to classify with direct observation. It is split into a training set of sixty thousand entries, and a test set of ten thousand entries. The images are defined with 784 variables representing pixel information, and one variable representing article clothing type.



Two images from the Fashion Mnist dataset.

The following methods will be used: K-nearest neighbors, Linear and Radial Support Vector Machines, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Deep Neural Networks, and Multinomial Logistic Regression. Moreover, each of the above methods will also be trained using dimensionally reduced data via Principal Component

Analysis, and the difference in error rate and computation speed will be examined. It should be noted that I was unsuccessful with coding the Radial SVM and Multinomial Logistic Regression approaches.

Principal Component Analysis (PCA)

PCA uses Eigen Decomposition to capture as much signal as possible in a reduced-dimensional setting, and therefore has the potential to vastly improve computation. In this case, 90% of the signal was captured by 74 as opposed to the original dataset's 784 dimensions. Then, each of the following methods will be evaluated with an embedding of 75 dimensions, counting the dimension specifying article clothing type.

Tenfold Cross-Validation (CV)

A sample of 1200 datapoints were chosen at random from the training set, and 600 points were similarly chosen from the testing set. In order to derive a predictive model, a subset of these points must be used to fit the model, and another subset to test this fitted model. A split of 80%/20% may be used, but this does not fully utilize the available data for training. In this case, Tenfold Cross-Validation was used to overcome this problem. It divides the sample into 10 sets of 120 datapoints, uses 9 of them for training and 1 for testing, then repeats this process several times by switching the training set. With the exception of KNN models, 5 repetitions were used for training cross-validated models. 10 repetitions were used for KNN. This process is often used to tune model parameters and tends to work well for optimizing the bias-variance tradeoff

in practice. However, the tuned models are often positively biased and report an optimistic accuracy measure. This is because cross-validation utilizes the entire training set for tuning model parameters. Meaning the model has "already seen" the predictions it is making. Once the parameters are tuned, the model is then used to predict the classification of the *separate* 600-point sample from the testing set, and this gives a better estimate of the true error rate.

K-Nearest Neighbors (KNN)

The K-Nearest Neighbors model classifies datapoints by comparing them to the classification of their closest K neighbors. Cross-validation was used here to find that the optimal number of neighbors for prediction in this case is 7. The processing time for this model using the PCA (74-dimension) data was 0.3s, and 1.9s for the full (784 dimension) data. Moreover, the expected accuracy of prediction here is 79.2% using PCA, and 73.8% without, meaning an error rate of 26.2% and 20.8% respectively. This is highly overestimating the true accuracy of this model. In fact, the error rate on the testing sample was found to be 78.7% using PCA, and 78.8% without. The error rate here was found using the proportion of correct observations. For clarity, I will be using the error rate computed using the test data for the remaining models. Note here that the reduced-dimension data provided slightly better prediction.

Linear Support Vector Machines (SVMs)

The Linear SVMs model finds the hyperplane that optimally separates classes within the data. This linear hyperplane is a line in the two-dimensional case. If there is a clear

separation of the data based on class, it does this by maximizing the distance between the points closest to the separating hyperplane, known as support vector machines. In practice, however, a clear separation is unlikely to ever be the case. In what is called the "soft case," the Linear SVMs model allows for some misclassification while optimizing to lower an objective function that describes the distance of support vector machines from the hyperplane. There is an additional penalty for misclassified points. In this case, the processing time was 58s with PCA, 607s without. Error rate was found using proportion of correct observations. It is 23% with PCA, 21% without. Despite the 2% improvement, note that the processing time is almost tenfold without PCA.

Radial Support Vector Machines (SVMs)

The Radial SVM model is similar to the linear version but allows for classification of data that is not linearly separable. It does this by using what is known as a kernel function. Kernel functions map features into a higher-dimensional space—called feature space—in which the relation may be modeled as linear. The resulting "hyperplane" is non-linear in the lower-dimensional setting. I was not able to tune the Radial SVM model.

Linear Discriminant Analysis (LDA)

In Linear Discriminant Analysis, the model is assumed to be normal with the same covariance for each class. The prior probabilities are estimated as the proportion of points belonging to a class relative to the total number of points. The model is fitted over posterior probabilities and assigns each point to the maximum likelihood class by comparing the classes pairwise. The processing time for this model was 0.05s with PCA,

3.39s without. In this case, two of the variables of the full dimensional data could not be processed with LDA, and I was therefore forced to omit two of the 784 variables. The error rate was found by averaging over the number of incorrect predictions. It is 21.7% with PCA, and 35.5% without PCA. Interesting to note here that using PCA resulted in a lower error rate, perhaps because much of the variation was captured by the two variables lost in the case without PCA.

Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis is similar to Linear Discriminant Analysis but with each class having its own variance. This requires more computation and is therefore prone to incompatibilities when using the full-dimensional dataset. I was not able to run QDA without PCA as one or several of the 784 variables had too little variance for the model to be correctly tuned. Without PCA, the processing time was 0.02s, and the error rate was 26.3%. This is interestingly lower than the error rate with LDA using the PCA data. This could be explained by less variance and a more optimal bias-variance tradeoff.

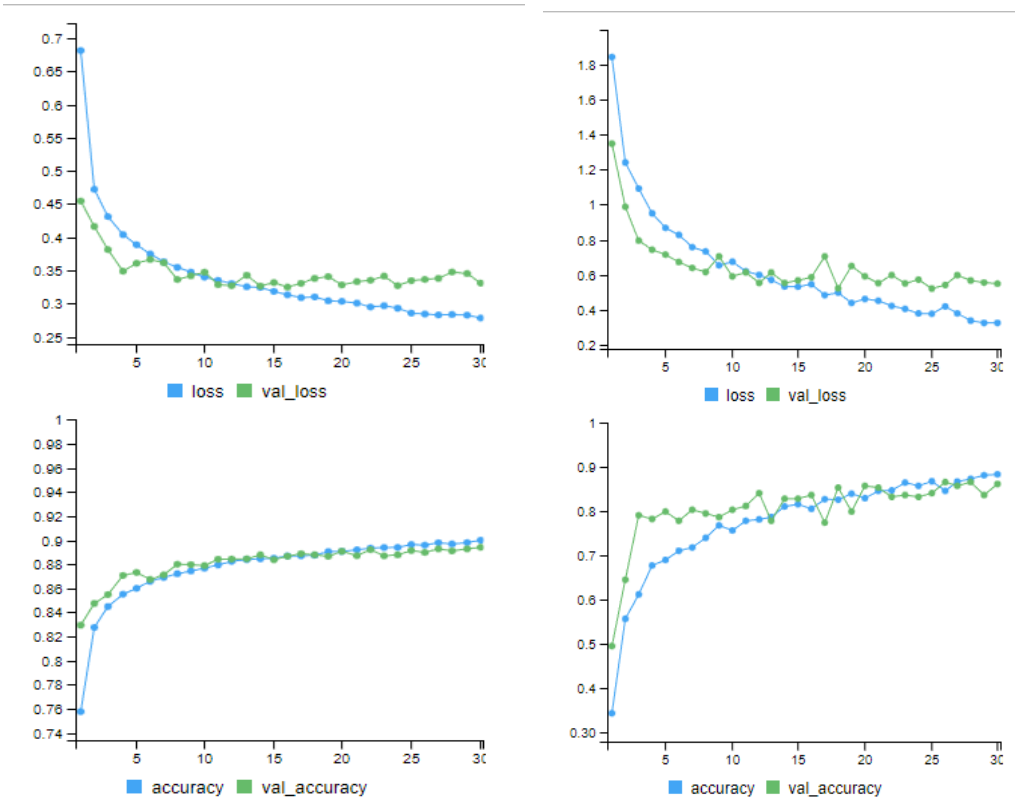
Random Forests (RFs)

Random Forests average many iterations of decision tress optimized over bootstrapped samples of the data. This model also randomly chooses a subset of parameters used to make each split, which allows for less correlated decision trees. This optimizes the bias-variance tradeoff by making it virtually impossible that all the averaged trees are at some level split using the same parameter, and therefore accounts for a wider range of possibilities—meaning more variance. The processing time for this model was 4.4s with

PCA , 23s without PCA. The error rate was found by averaging over the number of incorrect predictions. It is 20.5% with PCA, 19.7%, without PCA.

Of the above approaches, Random Forests has the most consistent results. Both trials with and without PCA have low error rates and processing time.

Deep Neural Nets (DNNs)



Test error rate versus data used. Left: without PCA Right: with PCA

Deep Neural Nets involve alternating layers of linear and nonlinear combinations of inputs, each layer having the output of the previous layer as its input. They tend to overfit small datasets—but given a sufficiently large dataset tend to drastically

outperform other models. The architecture of a deep neural net is more of an art than an exact science. A relatively straightforward set of layers was used here: starting with a layer of 256 units, then a layer of 128 units, then a layer of 10 units, each utilizing the ReLu as their nonlinear function. The processing time was 8.5s with PCA, 134s without. The error rate was 19.4 with PCA, and 11.5% without.

Multinomial Logistic Regression

Multinomial Logistic Regression is used to calculate the probability of a data point belonging to a particular class. It is similar to linear regression but modifies that model using exponentials and proportion to avoid such probabilities having negative values or values greater than 1. I was not able to correctly code the model for this report.

Of all the above models, Deep Neural Nets had the greatest accuracy. This is because DNNs are designed to leverage large amounts of data to fits models with very low bias. The large amount of data ensures that much of the variability in the population is captured by the dataset. One issue with DNNs is that they have low interpretability due to the complex interplay between linear and nonlinear processing. For this reason, it is difficult to come up with a precise method for optimizing their architecture. In general, optimizing neural nets requires a lot of trial and error.