STAT426 Final Writeup

Andrew Widjaja

PART 1: PCA

PCA is an unsupervised learning technique that aims to reduce the number of dimensions your data lives in by projecting it onto the axes that express the most variation in the data called the principle components. It turns out that the axes of greatest variation are the eigenvectors of the covariance matrix, hence in practice we use SVD to get the projected/embedded data and the principle components themselves. In the code we embedded the training data into 72 dimensions (where each dimension is a linear combination of the original 784 features). We used 72 dimensions because it took 72 principle components to capture more than 90% of the variance in our training data (where the variance is measured by the cumulative sum of the corresponding eigenvalues). I don't think our data should really live in 784 dimensions for the reasons stated below.

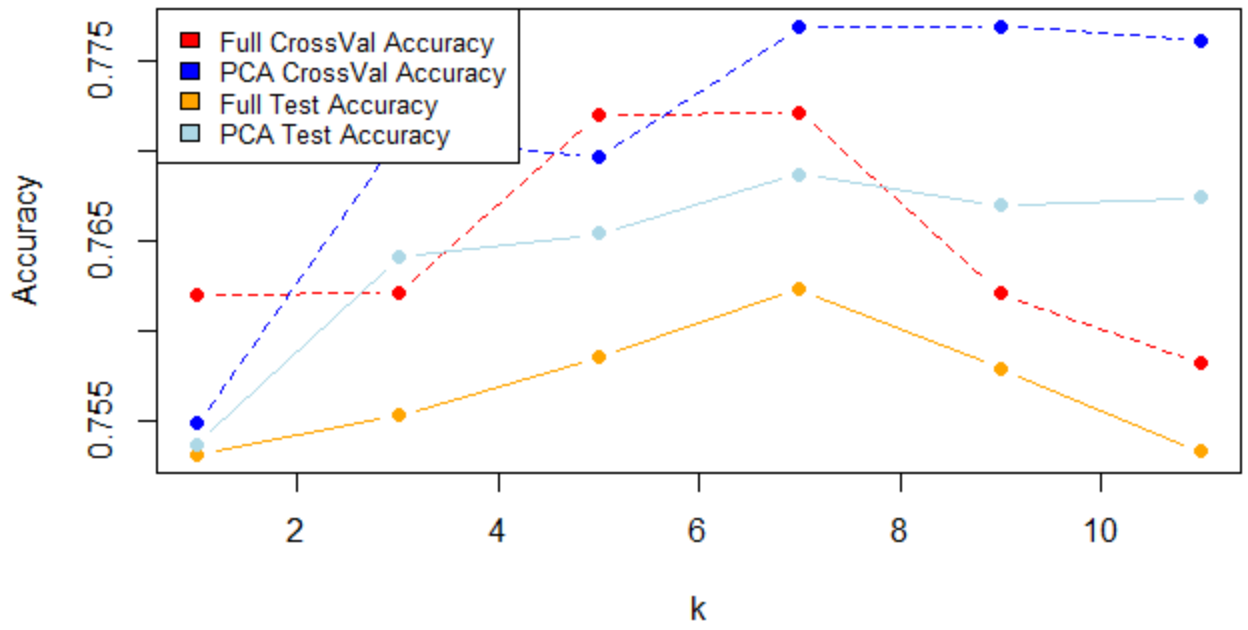
Projecting into a reduced feature space allows for much faster training (and testing for KNN) because we use smaller inputs which typically means you can estimate less parameters in your model. Also, it's nice to have n >> p since for some methods like logistic regression, you can't even get a unique solution when p > n, and for other methods like QDA as p increases the number of parameters you need to estimate also increases so you need more data to get better parameter estimates. However some methods like SVM work better in higher dimensions since there it can more easily find linear separability, but we can always just use the kernel trick to go back to higher dimensional space. Lastly, methods that use Euclidian distance, like KNN, suffer from high dimensional space since in higher dimensions uniformly distributed points are moved to the edge of the feature space (points get farther away) and points tend to become equally distant from each other, nullifying our ability to capture similarity through Euclidian distance.

Overall, I don't think we will sacrifice accuracy for speed because a lot of the pixels in each image don't signify anything, like the pixels at the corners which are just empty/blank space. Hence, by only considering the most important dimensions we remove a lot of the information conveyed by the "noisy" pixels. Removing noise hopefully increases test accuracy as it forces our models to generalize rather than memorize as they have less features to work with. For example, if you look at the data below, the loadings for the first and last pixel (top left and bottom right pixels, respectively) are closer to 0 than the loadings for the 392nd pixel. So the information stored at the corners mean less in the embedded data than the information stored in the middle of the image, essentially zeroing-out the noisy pixels while preserving the signal pixels.

Pixel #	PC1	PC2	PC3	PC4	PC5
1	1.404215e-06	-5.519884e-06	-1.815725e-05	1.111291e-05	-3.778442e-07
392	0.020818950	0.026605588	0.017801186	0.007457991	0.006465271
784	3.955042e-05	-2.528499e-04	-2.605008e-04	5.823787e-04	-1.051339e-03

Part II: KNN

KNN is a supervised learning technique used to predict the labels of our test data. We first provide the training data (both full and embedded) to the KNN model. We run 10-fold cross validation which gives us a lower variance estimate of the model's accuracy (when compared to a validation set approach since we average over the validation accuracy of 10 different models). Also, since we cannot use the test set to tune our model (as that would be cheating), and we don't have a validation set, cross validation is necessary. We choose the optimal k to be the one that gave the highest accuracy estimate. When we want to classify one of our test points, we provide the point to the tuned model, the model then finds the K nearest training points to our queried test point. Finally the model predicts the most common label found among those K training points.



We can see that the model trained on the PCA data performs slightly better in terms of test accuracy for all tested values of k. It achieves a max test accuracy of ~76.9% which is greater than the full model's max by ~0.7%. Furthermore, doing predictions on the test data across the different values of k for the full model took about 6.5 minutes, whereas predictions on the reduced model took about 4 seconds. This is because the full model needs to calculate the Euclidean distance in 784 dimensional space, whereas the reduced model calculates the distance in 72 dimensional space which is substantially less computations. So we can conclude that the PCA/reduced model is better than the full model.

We can also see that the accuracies returned by 10-fold cross validation over estimate the true accuracies on the test set, but by no more than 2%. So the cross validation estimates for model accuracies are within 2% of the actual test accuracies. Cross validation even showcases the fact that the PCA model outperforms the full data model for most k (except 1,5), and the PCA model achieves a greater maximum accuracy of 77.7%. Cross validation also found decent values for k because for the full model it indicates that k=7 is optimal, and in fact the true test accuracy for the full model is largest when k=7. However, cross validation indicates that k=9 is optimal for the PCA model, but the test accuracy is largest when k=7. Looking at the graph above, you can see that doing cross validation shows that k=7 and k=9 are about equally as good for the PCA model, and k=5 and k=7 are about equally as good for the full model. So at least cross validation recognizes that the optimal k (=7) is a good choice for both models. Perhaps we could do repeated cross validation to get even more robust results (since there is still some variation in our estimates due to

correlated models), but this would take long on the full model since it already takes 6.5 minutes just to run one 10-fold cross validation.

Part III: Other classifiers

Linear + Radial SVM

Linear SVM is a supervised classification technique that aims to find a hyperplane that linearly separates the classes in our feature space such that the margin is maximized. It is called linear SVM because we use the linear kernel which is just the regular dot product; this corresponds to the identity feature mapping. Typically SVMs solve binary classes, but since we have 10 classes we have to use the "one vs one" or "one vs rest" technique to classify based on votes from multiple SVM models. Since the data is often not linearly separable in our given feature space, we have a Cost hyperparameter that will allow some points to lie within the maximal margin or even be wrongly classified (there is a trade-off between the maximal margin and the cost; if we allow more errors, we can get a wider margin). In general a wider margin means a stronger ability to generalize, so decreasing cost can help with overfitting. We use cross validation to tune this hyperparameter so that the generalization accuracy is maximized.

Radial SVM is a supervised classification technique that aims to find a hyperplane that linearly separates the classes in a higher dimensional feature space such that the margin is maximized in that feature space; the resulting decision boundary is non-linear in the original feature space. It is called radial basis SVM because the kernel it uses is the radial basis function which could potentially result in a feature mapping onto an infinite dimensional space. Once again, we have a cost parameter in case linear separability cannot be achieved in the higher dimensional space, and to help control over/under-fitting. However, we have another hyperparameter gamma (=1/sigma^2), where gamma controls the "width" of the gaussian kernel. Larger values of gamma means a smaller width so the decision boundary will be more local (overfit) around points, whereas smaller values of gamma means a larger "width" which entails the opposite. Once again we use cross validation to tune these 2 hyperparameters.

Random Forest

Random forest is a supervised technique that uses bootstrapping to build many deep trees and then averages the predictions of those trees when testing. In other words, we take the average of many low bias, high variance models (deep trees) in order to get a low bias, lower variance model overall. However, the mean of many uncorrelated models has a lower variance than the mean of many correlated models, so random forests aim to reduce the correlation between trees by only considering m (= sqrt(p) = 9 for our classification case) of the predictors at each split, and greedily selects the best one each time. This prevents any one predictor from dominating the trees, so the trees in the random forest will look different (trees wont split on the same variable at each interior node since we give different predictors a chance each time). We have to do this because bootstrapped data on average contains only 2/3 of the original data, so the bootstrapped sets are correlated.

LDA + QDA

LDA is a supervised classification technique that classifies based on the greatest posterior probability

P(Y=k|X=x) over all k classes. It estimates the posterior probability through Bayes' theorem which states that the posterior = prior * likelihood / evidence. We estimate the prior through counting the number of observations of class k divided by the total number of observations. We estimate the likelihood by assuming X|Y=k is distributed normally with class specific mean, but shared covariance (which is basically pooled variance), both of which we estimate from the data. If the purpose is purely classification (as in our case) we simply check to see if the log of the ratio of posterior probabilities are greater than 0, so we can ignore estimating the evidence since they cancel out in the ratio. In fact, you can purely look at the difference between two classes' linear discriminant functions (which are linear in x); if k's discriminant function output is greater than j's, then we prefer label k. Hence we obtain a linear decision boundary.

QDA is a supervised classification technique that classifies based on the greatest posterior probability P(Y=k|X=x) over all k classes. It is similar to LDA except it estimates the likelihood as being distributed normally with class specific mean AND class specific covariance. This makes the quadratic discriminant functions end up being non-linear in x, so the resulting decision boundary is non-linear.

Comparison

Classifier	Train + Test Time (Sec)	Test Set Accuracy
KNN	0.78	0.7654
SVM-Linear	0.34	0.7678
SVM-Radial	5.42	0.8044
Random Forest	1.97	0.7842
LDA	0.06	0.7829
QDA	0.39	0.6619

The table above records the time it took to train and test the final tuned classifiers on the PCA data of 1000 samples (time is measured as user + system time). Note that the time does not take into account cross validation time as not all methods used cross validation. Also we record train + test time because KNN doesn't spend time training since training consists of just storing the data. The final accuracy obtained on the test set is also recorded.

We can see that the runtime of linear SVM (.34 seconds) is much lower than that of the radial SVM (5.42 seconds) by a factor of ~16. However, the radial SVM achieves a 4% better test accuracy at 80%. KNN is slightly slower than linear SVM at .78 seconds while also achieving a similar test accuracy, so we can conclude that KNN is overall worse than the linear SVM. The random forest model is somewhere in between the linear SVM and radial SVM models in terms of performance. That's because the RF model achieves a test accuracy of 78% which is better than linear SVM's but worse than radial SVM's. Also, it has a runtime of 2 seconds which is faster than radial SVM but slower than linear SVM. Furthermore, LDA achieves a much better test accuracy of 78% when compared to QDA's 66%. It is also much faster as it took only 0.06 seconds to train and test whereas it took QDA 0.39 seconds. So QDA is overall worse than LDA. LDA is also much faster than RF while achieving a similar test set accuracy. The only model that you might consider over LDA is the radial SVM model which has a 2% better accuracy, but also takes ~90 times longer to train and test.

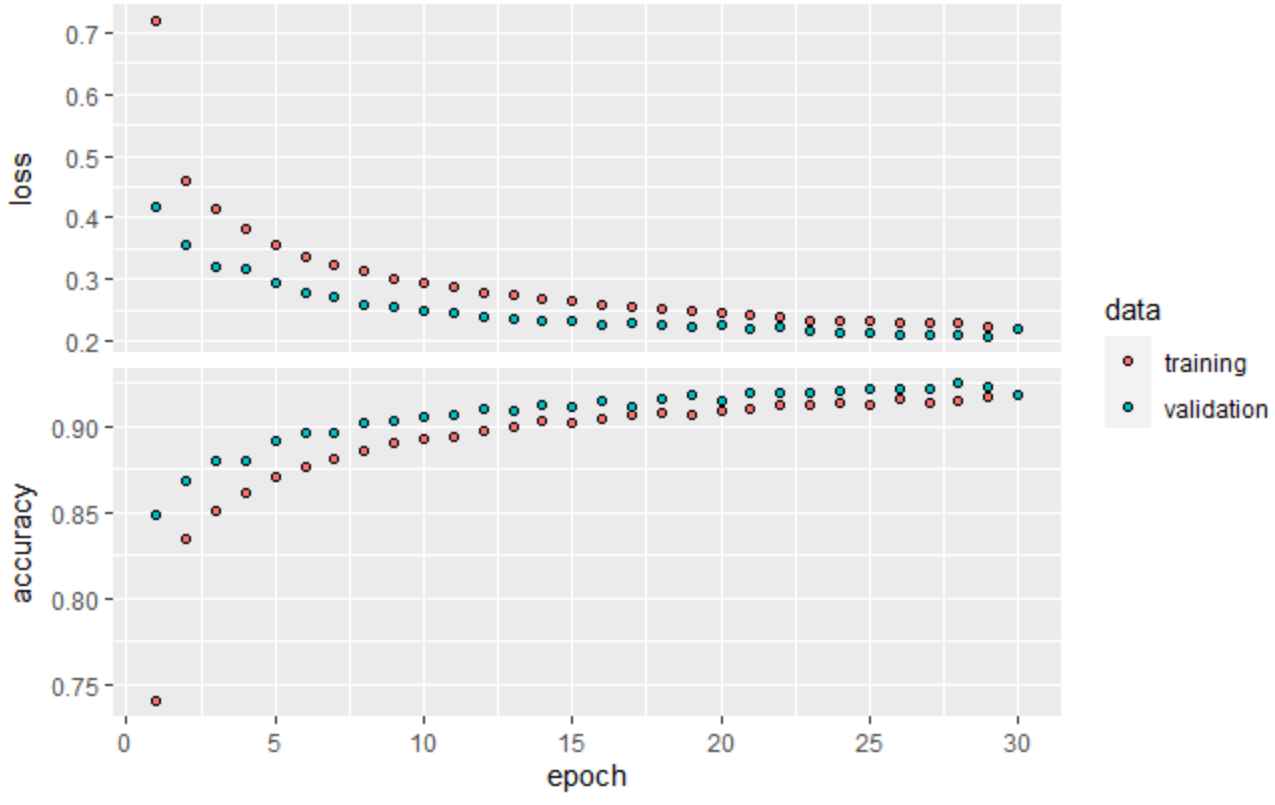
Hence for the best time performance I would pick LDA, but for the best accuracy I would pick radial SVM. The choice depends on if the user thinks a slight increase in accuracy is worth spending much longer on training and testing (and even more time must be spent to tune the gamma and cost hyperparameters for

the RBF SVM model, whereas the LDA model achieved the results without any tuning). Also, LDA is a simpler model (higher bias, lower variance) since it only produces linear decision boundaries, whereas the radial SVM model produces non-linear decision boundaries in the provided feature space. So ultimately, the LDA model is probably the better choice because it is a simpler model that still achieves relatively good test accuracies whilst being the fastest method by far and requires no tuning; it is the most efficient.

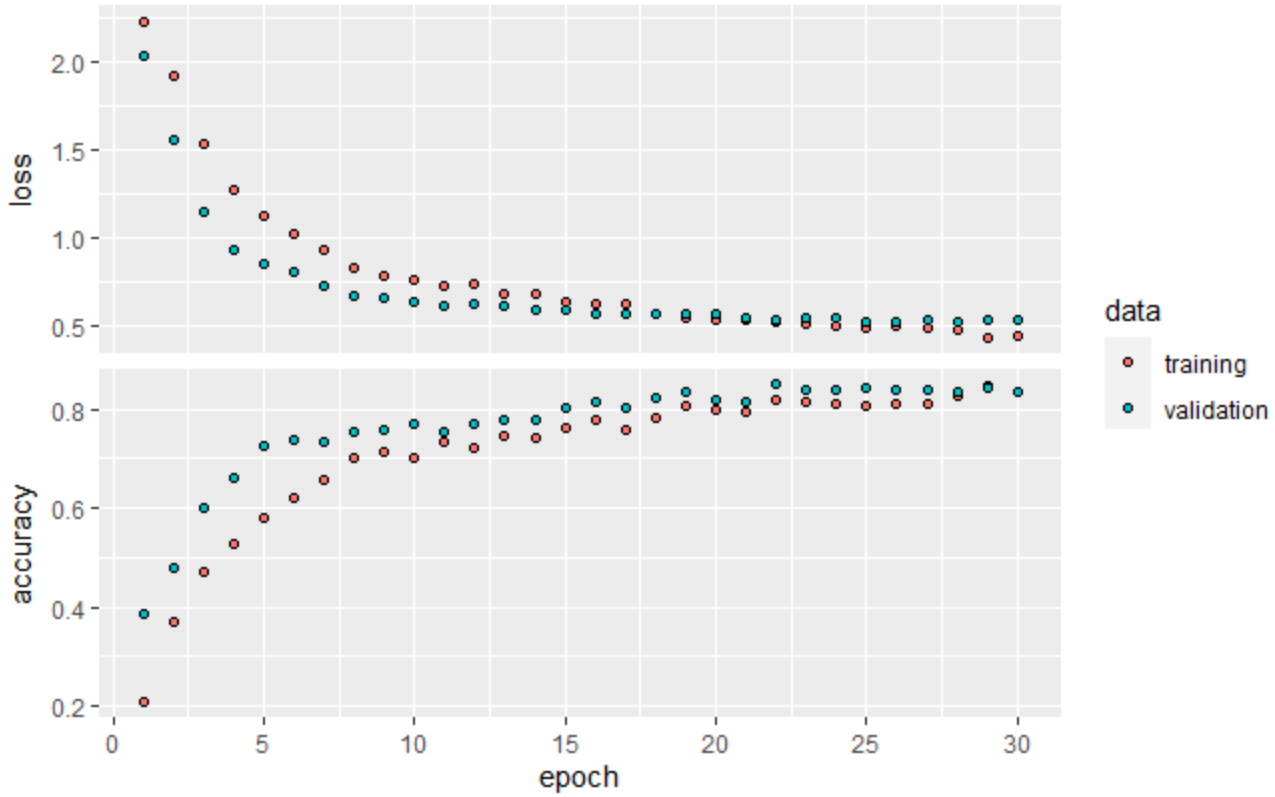
Part IV: Deep Learning

In deep learning we typically take a linear combination of non-linear functions (controlled by the width of a hidden layer) and we compose non-linear functions (controlled by the depth of the model) in order to model some complex relationship. In our case, we use the softmax activation function in the output layer to predict the 10 posterior probabilities (which is why we one-hot encode the truth labels). We assign the label corresponding to the largest posterior probability. Since we are dealing with images we use a special neural network architecture called a Convolutional Neural Network (CNN). CNN's learn features in our image starting from simple features like horizontal edges to more abstract features like a zipper. It achieves this by learning the weights for kernels. The first half of the CNN uses these kernels (and pooling) typically to reduce the dimension but increase the number of channels of our input. CNN's take advantage of the spatial information stored in the images since we don't flatten our images when using a CNN. At the end of the convolution/pooling layers, we flatten the array and feed the resulting vector into a regular neural network.

Neural networks must learn tons of parameters. Our model has about 207000 parameters, whereas we only have 60000 training points. Hence, neural networks are low bias, high variance models that are prone to overfitting. To reduce overfitting we use dropout regularization which gives each hidden node an equal chance to learn by zeroing out the signals of random hidden nodes during training; one node cannot dominate the others by learning more. We also provide tons of data to reduce the variance. We use batches of size 128 to speed up gradient descent and we use Adam's optimizer to converge faster and escape local minima.



Above is the train-validation performance of the CNN using all 60000 training points. Decrease in loss and increase in accuracy steadily converges across epochs (1 epoch contains about 60000 obs/128 batch size = ~469 gradient descent updates per epoch).



Above is the train-validation performance of the CNN using only 1000 training points. Decrease in loss and increase in accuracy isn't as steady across epochs because an epoch only looks at 1000 observations (1000 obs/128 batch size = ~8 gradient descent updates per epoch).

Classifier	Train Time (Sec)	Test Set Accuracy
CNN w/n=60,000	2494.22	0.9178
CNN w/n=1,000	48.56	0.8051

The test accuracy of the model that uses all 60000 training observations is 91.8% which is substantially better than all the PCA models we looked at before (at least an ~11% increase). However, using 1000 training observations only, like the PCA models, we achieve an accuracy of 80.5% which is not even 1% better than the radial SVM. Here we train on all 784 features instead of the PCA data, hopefully to gain a higher test accuracy by not possibly losing any of the discriminating features.

You can see that both models took very long to train, about 2500 seconds for the full model and 50 seconds for the reduced sample model. Hence, I think if you want to use CNNs for their great test accuracies, you should go all out and spend the extra time needed for training. If you're conservative with how many observations you provide (like in the 1000 observations case) your test accuracy will be about as good as the classifiers we looked at before while being a way more complex model. It also takes much much longer to train and tune (though in this case I didn't tune, I just used the architecture provided in the MNIST pdf) compared to the previous classifiers. Hence, CNNs are a good choice if you have a lot of time and data.