Good work as usual!
A+

PROJECT 3 - MACHINE LEARNING CLASSIFICATION

For this project, I decided on the two algorithms: convolutional neural networks and k-nearest neighbors with PCA. Both of these algorithms were performed on the MNIST data set. For each of the two algorithms, I varied their hyperparameters and listed down the effect that this had on the training and test time as well as the accuracy of the classifications. The code that I used were both from class: mnist_convnet.py and mnist_pca_knn.py, with alterations in the code. There were some functions that I did not understand and decided to comment them out and print results in the terminal to see what their functions were. I also used open cv documentation to better understand the principal component functions in the mnist_pca_knn.py code. Reading documentation and forums on stack overflow were of some help, but actually tinkering with the code in the terminal helped much more.

I often find this too!

CONVOLUTIONAL NEURAL NETWORKS VS K-NEAREST NEIGHBORS

I predicted that since the convolutional neural network has more parameters, it follows that it would take more time for learning to occur. Hence, I predicted that the convolutional neural could take more time training than the k-nearest neighbors algorithm, but would be more accurate as a result of having more parameters. By design, the convolutional neural network also takes into account the relationships between the spatial features of the digit images, and by also including a dropout layer to prevent overfitting, it follows that we would have a longer training time but a higher accuracy on the data set. However, I thought that this would depend on the number of principal components that I decide on. The accuracy of k-NN with PCA should also be very high since we are dealing with a data set where different handwritten versions of the same number would have data points very close to the points in the training set. It also does not require training time, hence I predicted that it would be faster than convolutional neural networks for a low number of principal components, but slower for a high number of principal components as it calculates all of the distances between the data points.

CONVOLUTIONAL NEURAL NETWORKS - Varied hyperparameters and predictions

I tried different numbers of kernels for each of the two convolution layers. The first one was with 64 in the first layer and 128 in the second, and then I doubled this, resulting in 128 in the first layer and 256 in the second layer. To test this, I kept the dropout rate at 0.5. I predicted that this would result in a slower training time, but I was not sure whether or not the proportionality between doubling the number of kernels per layer is linearly proportional or not to the increase in training time. The accuracy could also increase due to more parameters to accurately classify new information.

I also tried varying dropout values. For testing the effect of the dropout value on the classifier's performance, I kept the number of kernels constant at 32 in the first convolutional layer and 64 in the second. The dropout values that I put in were 0.1 and 0.5. I predicted that the lower dropout value of 0.1 would yield a lower accuracy on the test set due to significantly more overfitting based on the training data, because we are only zeroing out 10% of the data values going into a dense layer. The higher value of 0.5 for the dropout would result in a better accuracy on the test set due to the fact that it's introducing more perturbations into the data, hence the trained network will become more robust when it encounters different variations from what it has learned.

K-NEAREST NEIGHBORS - Varied hyperparameters and predictions

I decided to use k=5 and k=11 as the two different values with a fixed PCA value of 150. I also tried changing the number of principal components from 150 to 250 to see what effect this would have on the accuracy and time taken with a fixed value of k=5. I predicted that having a higher number of centroids would increase the accuracy due to being able to better handle noise in the data. I also thought that higher principal components will yield a lower accuracy as overfitting is more likely to occur. However, I know that this would take more time as we are considering three times more principal components.

RESULTS

The table below shows the effects of varying the number of kernels for the convolutional neural network with a fixed dropout value of 0.5, and varying the number of centroids from 5 to 11 with a fixed PCA value of 150.

	CNN - 64/128	CNN - 128/256	KNN - k = 5	KNN - k = 11
Training error	0.67%	0.43%	N/A	N/A
Testing accuracy	99.15%	99.26%	97.12%	96.83%
Training time	446.28 seconds	1183.83 seconds	0.32 seconds*	0.27 seconds*
Testing time	1.01 seconds	3.28 seconds	47.11 seconds	48.26 seconds

TABLE 1. *Varying number of kernels for convolutional neural networks with a fixed dropout rate of 0.5, and number of centroids from 5 to 11 with a fixed PCA value of 150. *This reflects only the time taken for PCA = 150 since kNN does not train on a data set, but rather computes statistical data.*

The results in the table above show that increasing the number of kernels did increase the accuracy as predicted due to the increase in the number of kernels. It also did take longer, and by the results it seems as if it is approximately linear in that doubling the number of kernels doubles the training time, but this data is not enough to validate this conclusion. The hypothesis that CNN would take longer was correct, but I did not realize that it would be this significant. The time gap between the two is much larger than expected. My hypothesis was supported since in all four experiments, the testing accuracy of the CNN outperformed KNN, and that CNN does take more time due to the fact that it has to train on a data set first, then undergo testing. Hence if time is not particularly a big concern, then using a convolutional neural network would provide a higher accuracy for classifying images.

I then varied the dropout value from 0.1 to 0.5 with fixed kernel numbers of 32 in the first layer and 64 in the second layer for the convolutional neural network. I also tested changing the principal components to see its effect on the classifier.

	CNN- drop 0.1	CNN - drop 0.5	KNN PCA 150	KNN PCA 250
Training error	0.79%	0.82%	N/A	N/A
Testing accuracy	99.00%	99.14%	97.12%	96.96%
Training time	221.27	219.46	0.32 seconds*	0.36 seconds*
Testing time	0.39 seconds	0.39 seconds	47.11 seconds	76.01 seconds

TABLE 2. *Varying dropout value from 0.1 to 0.5 with fixed kernel numbers of 32/64 in the respective layers, and number of principal components from 150 to 250 with a fixed centroid number k = 5. *This reflects only the time taken for PCA = 150 since kNN does not train on a data set, but rather computes statistical data.*

The results in the table above shows the effect of taking measures against overfitting. I saw that a higher dropout value would result in a higher testing accuracy, which supported my hypothesis as a higher dropout rate would result in less overfitting, making the network more robust to small perturbations in new test data, making it more accurate. I also saw that adding 100 more principal components to consider resulted in a lower accuracy for the testing set, and the results supported my hypothesis. This was perhaps because overfitting occurred as it started to take into account irrelevant data amongst the pixels.

✓

It is extremely important to validate the final classifier on a separate data set from the one it trained on because the entire purpose of learning is to then generalize and classify novel situations based on what it has learned. Hence, training on the entire data set without validating it on a separate data set means that you are running the risk that this classifier only does a good job on this particular set of data, and not novel data sets. If we vary our hyperparameters such that the training error is close to zero, then overfitting may occur where classification errors occur when we actually test the classifier with novel data due to the fact that it is not “exactly” the same as what it has seen from its training data set.

By observing the results in both of the tables above, the accuracy drops when we shift over from the training set to the testing set, which is expected as there could be some perturbations in the data that the classifier does not know how to handle. Another interesting observation in the data was that when I increased the dropout rate from 0.1 to 0.5, the testing accuracy increased, but the training accuracy decreased. This is perhaps because overfitting likely occurred with a dropout rate of 0.1 because we are not introducing that many perturbations from the training data. The classifier with the dropout rate of 0.5 is doing a better job of generalizing what it has learned rather than memorizing specific patterns in the data and applying that to the testing data sets.