EMRAH SARIBOZ

HW-5 Report

Accuracy and run-time analysis

Model	Dataset	Run-time	Accuracy	Precision	Recall	F1
PCA + DecisionTree	Iris	0.08 seconds	0.90	0.90	0.90	0.90
PCA + DecisionTree	MNIST	45 seconds	0.73	0.737	0.732	0.734
KPCA + DecisionTree	Iris	0.10 seconds	0.87	0.87	0.87	0.87
KPCA + DecisionTree	MNIST	55 seconds	0.75	0.74	0.74	0.74
LDA + DecisionTree	Digit	0.02 seconds	0.96	0.96	0.965	0.958
LDA + DecisionTree	MNIST	10 seconds	0.85	0.8598	0.8548	0.8557

*The accuracy, precision, recall and f1 scores are the mean value of k-fold cross validation (k = 5)

*In the report, I use PRF which stands for Precision, Recall and F1

- **Data Preprocessing**
  - I have used train_test_split method to use 10 percent of the original dataset and stratify is True.

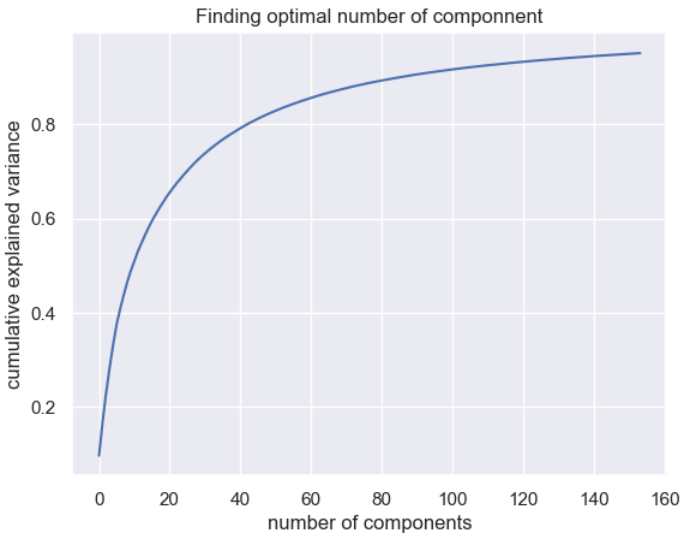
Analysis

- The main purpose of the PCA is to reduce the dimensionality of the huge dataset. In this context, I applied PCA, kernel PCA and LDA. I used decision tree classifier to fit reduced dataset.

PCA

- **Parameter Tuning**
  - n_**component**
    - n_component is the number of components we want to reduce from the original dimension. I found out that if you pass a float number (say 0.95), it means “find the number of components that preserves that 0.95 variance of the original dimension space”
    - If you pass integer, it is the # of component you want to find. I passed 0.95 to keep the original variance as much as possible. When I change this value to 0.9, which means preserve the 90 percent of the original dataset, the values fluctuated only by +/- 0.01.

- However, 0.5 variance give me interestingly better accuracy (0.76) and better PRF (0.76, 0.76, 0.76). I am not quite sure why it is increased. I was expecting it to decrease. I believe the dataset is not informative enough.
- I used this floating number of the test purposes. I found the optimal k for the PCA component with the following plot.



- Above plot shows that cumulative variance reaches its 95% when we use k = 140.

Kernel PCA

- **Parameter Tuning**
  - **Kernel:**
    - The kernel function it will use to apply PCA. I was received 0.74 mean accuracy for the Mnist dataset but I got 0.76 and PRF 0.77 when I use poly.
  - **Degree:**
    - I changed this value from 3 to 4 and 5 ;however, I didn't get any change in my accuracy and RPF scores.

Linear Discriminant Analysis:

- **Solver:**
  - It uses singular value decomposition by default.
  - I changed this value to eigen value decomposition, and it required another parameter known as shrinkage.
- **Shrinkage:**
  - This value requires when we use EVD rather than SVD. Setting this equal to auto (Ledoit-Wolf lemma), my accuracy dropped dramatically from 0.85 to 0.71.
  - PRF dropped from (0.8598, 0.8548, 0.8557) to (0.725, 0.722, 0.719) respectively.

Parameter tuning for Decision Tree:

- **Max Depth**
  - If nothing is specified, it will not stop until all nodes are expanded. I used this value to save my model from the overfitting
  - I used GridSearch to test which parameters will give me best accuracy. The grid search returned max_depth = 9

- **min_samples_split**
  - I used GridSearch to test which parameters will give me best accuracy. The grid search returned , min_samples_split = 4. The best accuracy I got from this is 0.72 using both max_depth and min_samples_split values.
- **Criteria**
  - Two criteria used in Decision Tree are Gini index and Entropy. I got all the above result using the Gini index. Although this criterion didn't make any improvement over the accuracy, it increased the run time especially on the MNIST dataset. It requires more mathematical operations compare to Gini.

Problem

- My model suffers from overfitting issue. In other words, the training accuracy is high compared to testing accuracy in all techniques (pca, lda, kpca). I used parameter tuning techniques to reduce it; however, there is still issue.

PCA on MNIST dataset

Mean of the accuracy for testing dataset : 0.7362857142857143

Mean of the accuracy for training dataset : 0.851678571428571

PCA on iris dataset

Mean of the recall for testing dataset : 0.8866666666666665

Mean of the recall for training dataset: 0.9783333333333333

KPCA on MNIST dataset

Mean of the accuracy for testing dataset : 0.728

Mean of the accuracy for training dataset : 0.8380714285714286

KPCA on iris dataset

Mean of the accuracy for testing dataset : 0.8800000000000001

Mean of the accuracy for training dataset : 0.9650000000000001

LDA on MNIST dataset [I am not sure about this one]

Mean of the accuracy for testing dataset : 0.8425714285714285

Mean of the accuracy for training dataset : 0.8853214285714286

LDA on iris dataset[I am not sure about this one]

Mean of the accuracy for testing dataset : 0.9533333333333334

Mean of the accuracy for training dataset : 0.9983333333333334

Comparison

- **For the following results, I used Decision tree with following arguments**

- DecisionTree(max_depth = 9, min_samples_split =3). The reason I choose this is I applied GridSearch (commented in the myPCA, myKernelPCA and myLDA classes).

- LDA has proved how fast it is compared to PCA and kPCA in my analysis. It took 30+ seconds for the PCA and kPCA while it took only 10 second for the same dataset with same decision tree algorithm.
- Accuray:
    - LDA has won the game in the accuracy category. I have used the same number of components for LDA, PCA and kPCA and I received almost identical result using the PCA and kPCA. Their accuracy only fluctuated 0.1 +/- in accuracy context using different DecisionTree parameter.
- PRF:
    - When I split the dataset using train_test_split to use 10% of the original dataset, I set stratify option TRUE to preserve original class distribution. In this sense, no matter what the kind of calculation (MICRO, AVERAGE, WEIGHTED, MICRO) for the precision, recall and f1 scores, they were almost same. At first, I thought there is something wrong with my implementation. However, after a little bit

research, I found out that since the distribution of the classes are the same for the dataset, it is possible to have these results.

Observations

I wrote my observations and couple of reasons why I believe my model suffering from the overfitting issue, below.

- Using only 10% of the dataset wasn't informative. My model always suffered from overfitting issue no matter what I tried.
- We got asked to apply PCA after standardization of the dataset; however, since we standardize the whole dataset (rather than fitting StandardScaler on training and transforming the test dataset using the mean and variance of the training) we probably have a data leakage problem. We leaked the information of the testing dataset to the training dataset.