**README/REPORT**

This ReadMe document will describe in detail the following:

1. Requirements to execute the program
2. Contents of the zip
3. Steps to execute the program
4. Detailed description of the major functions used and created in the program
5. Output of the program
6. Output Analysis and Questions in the Tasks
7. Section for any additional information and answers to questions in the assignment

**REQUIREMENTS**

1. Python 3.4 (This version will work with the cvxopt wheel attached with the code, for any other version of Python, a different one will be required)
2. PIL library (For image manipulations)
3. Numpy
4. Matplotlib
5. CVXOPT (For SVM)

Steps to install requirements:

* *$* sudo pip install –U numpy
* *$* sudo pip install –U matplotlib
* *$* sudo pip install –U Pillow
* $ sudo pip install –U cvxopt-1.1.7+openblas-cp34-none-win\_amd64.whl
* $ sudo pip install –U numpy-1.11.2+mkl-cp34-cp34m-win\_amd64.whl

**CONTENTS OF THE ZIP**

The zip contains a data directory which contains all images used for training and testing the algorithm. A ReadMe report file which describes how to use the program and a python executable file called script.py. cvxopt-1.1.7+openblas-cp34-none-win\_amd64.whl and numpy-1.11.2+mkl-cp34-cp34m-win\_amd64.whl which are used for quadratic programming (as mentioned above, these will work with Python 3.4 only). All the plot images for each of the task, the names of the images are self-explanatory of the task to which they belong.

**EXECUTION**

Once the dependencies are downloaded, and the contents of the zip are uncompressed, the program can be executed from the folder where script.py exists by using the command:

*$* python script.py

The time required for each step to be executed is mentioned in the output section.

**DESCRIPTION OF script.py**

It contains the following functions:

* read\_pgm

This function uses the Python Image Library (PIL) to read the pgm file. The image matrix is then converted into a single array so it can be used as a single data point for the algorithm. Since each image is 92x112 in size, each array for an image is of length 10304.

* resize\_and\_read\_image

This function, like read\_pgm also uses PIL to read pgm files and convert them into a single array, but before converting it into an array it resizes the images to the given length and breadth values.

* get\_data\_list

This function makes a python dictionary for the entire data, where the key is class name (s1 to s40) and the value is the list of images in each class. It returns a variable which holds the entire dataset for the algorithm.

* readAllImages

This function is used to read the entire data and create training and testing data for each fold. It creates and returns an array for the training set, testing set, class list in the sequence of the training set and a class list in the sequence of the testing set. These class lists are useful during the prediction phase of the algorithms applied on this data.

* KNN: Following are the functions implemented for KNN strategy.
  + getNeighbors

It takes in the training data, a test instance and K as input. According to the tasks, K=1. This algorithm gets the nearest neighbors to the test instance calculated using Euclidean distance.

* + getResponse

It takes all the neighbors list as input and provides the neighbor with the highest Euclidean distance as output.

* + euclideanDistance

It calculates and returns the Euclidean distance between 2 instances that are provided to it.

* + getAccuracy

It takes as input the test set with classes as the last column and the predictions made for each image in the set and calculates the accuracy of the predictions made using the KNN algorithm.

* pca

This function uses Principle Component Analysis to calculate new features for the images. These new features can be used to reduce the number of features for the image being used. The first step performed in this function is centerizing the images, which requires the mean of the features to be subtracted from each value in the input data. The function uses this centerized data to do further calculate covariance and then eigenvalues and eigenvectors using the numpy linalg.eig function. Once we receive these values we plot a graph of the eigenvalues and chose only the dimensions greater than 0. Very small values are considered as zero. The eigenvectors are then sorted with respect to the eigenvalues and the vector is then used to reduce the original feature vector into a new, reduced feature vector. The function returns this new feature vector and the principle component selected. The function also takes as input windowTitle so save the plot images with that name and dimensions. The dimensions parameter can be used for a function that wants to specify the reduced number of dimensions and not decide according to the plot.

* Lda

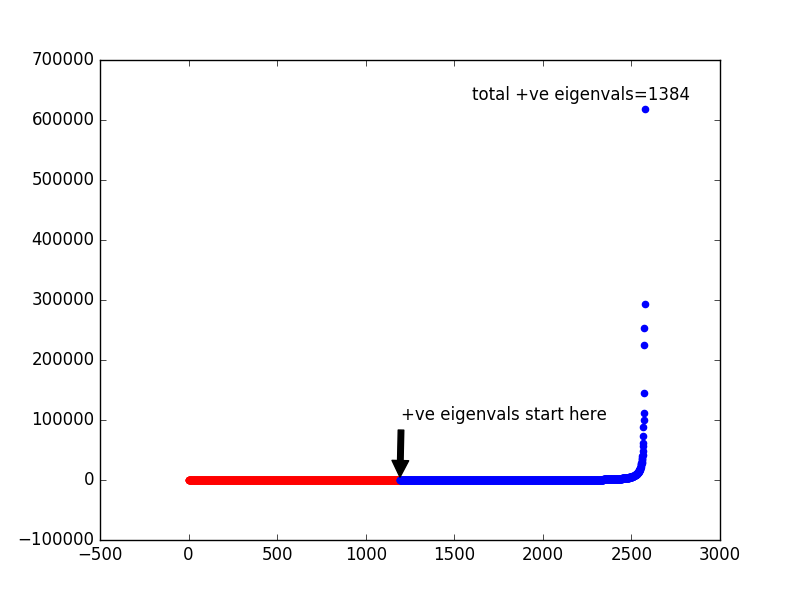
This function uses Linear Discriminant Analysis to calculate new features for the images. These new features can be used to reduce the number of features for the image being used. The first step performed in this function is centerizing the images, which requires the mean of the features to be subtracted from each value in the input data. LDA uses classes to create dimensions for the data. This requires to create mean vectors for each dimension for each class. These mean vectors are then used to calculate the scatter matrices. The other component needed is the covariance matrix of the data. The dot product of the inverse of the scatter matrix and covariance matrix the eigenvalues and eigenvectors are calculated. Once we receive these values we plot a graph of the eigenvalues and chose only the dimensions greater than 0. Very small values are considered as zero. The eigenvectors are then sorted with respect to the eigenvalues and the vector is then used to reduce the original feature vector into a new, reduced feature vector. The function returns this new feature vector and the linear discriminants selected.

* get\_principle\_comp, get\_linear\_analysis\_vector

These two functions perform similar tasks for PCA and LDA feature reduction techniques. They both sort the eigenvalues in decreasing order and compute the respective eigenvectors. These reduced vectors are returned, which are then used to compute the reduced feature set.

* plot\_eigenvalues

This function takes as input the eigenvalues and window title. The window title is used to store the image of the plot and eigenvalues plotted and used to calculate all dimensions that are greater than zero and can be used for reducing the dimensions. Matplotlib scatter plot is used to display the various eigenvalues. It returns the number if dimensions that can be used to create the reduced feature vector by using pca or lda techniques. Below is an example of a plot and its explanation:



As we can see in the example above, a scatter graph is plotted for the eigenvalues. The red plots indicate all the negative eigenvalues and the blue plots indicate all the positive values. The arrow(🡪) indicates the point from which positive eigenvalues begin. Total positive eigenvalues are also displayed in the graph, this value is used as the number of dimensions that will be used for the new feature vector.

* knn\_with\_pca, knn\_with\_pca\_resized\_image

These 2 functions perform KNN with K as 1. These functions take in data for execute 5 folds for cross validation. For each set of training data created for a fold, PCA is applied on it. The principle component obtained is then used to reduce the test data and KNN is used to predict the classes for the test set. The accuracy is calculated. This process is repeated for all five folds and the final accuracy is then calculated as an average of the accuracy for each fold. For knn\_with\_pca\_resized\_image , the only additional steps performed is resizing the image to 56x46 as given in task 2, the rest of the steps for folds and pca calculations are same.

* knn\_with\_lda, knn\_with\_pca\_lda

These 2 functions perform KNN with K as 1. These functions take in data for execute 5 folds for cross validation. For each set of training data created for a fold, LDA is applied on it. Except for knn\_with\_pca\_lda, where PCA is applied on the data first and then LDA is applied on the reduced data set. The linear component obtained is then used to reduce the test data and KNN is used to predict the classes for the test set. The accuracy is calculated. This process is repeated for all five folds and the final accuracy is then calculated as an average of the accuracy for each fold for both the functions.

* SVM: Following are the functions implemented for SVM strategy.
  + train

This function is used for training the data set it receives. The one-vs-rest strategy is used to train data where each class data is used to train against all the other classes present. Hence, multiple loops are used. It takes input as described above and uses it for calculating the values of alpha for each class using quadratic programming available through cvxopt library of python. A description of its input is given in a section below. Since all values of alpha are very low, the lower ones are ignored and only the bigger ones are used as support vectors. These values are then used to calculate b and w. The w array and b for each class will then be used to predict the class of testing files.

* + predict

This function is used to predict the class of the testing image. For each image, 40 values are calculated for the 40 classes, and the list of these values is returned.

* + svm and svm\_with\_pca

This function performs the training and testing of data.

* + - This function also executes in 5 folds, so the training and testing data set is calculated for each fold.
    - For svm\_pca, the PCA is function is called to reduce the feature set and then continue with creating support vectors for the new feature set.
    - After dividing the data, it calls the train function to get values of w and b for each class. This is stored in a variable.
    - The next step is prediction; each testing image calls the predict function, which provides the yi value for each class.
    - The class which has the maximum of these values is considered as the predicted class.
    - The actual and predicted classes are compared and accuracy is then calculated.
* main

The main function of the script, reads data, and calls the various functions in the following order:

Knn\_with\_pca, knn\_with\_pca\_resized\_image, knn\_with\_lda, knn\_with\_pca\_lda, svm, svm\_with\_pca

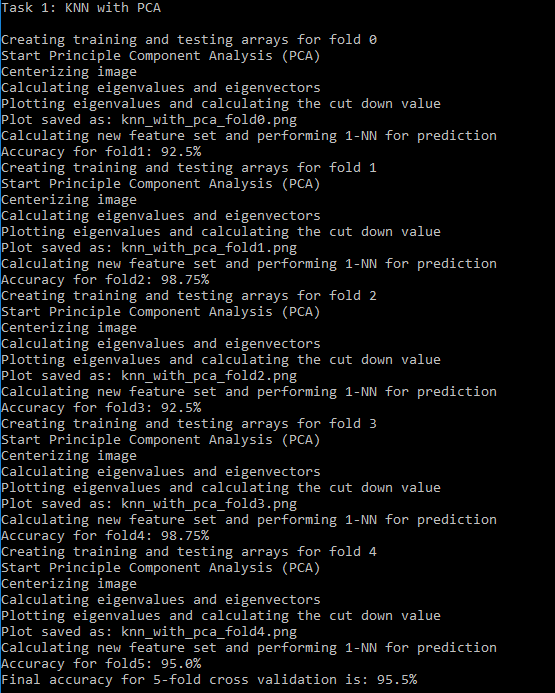
The description for each task and details are printed on the console as shown in the output section.

**OUTPUT**

The output of the entire program is too big to display in a single screen, hence below is a description of the output of each task as given in the assignment.

**Task 1: Do** **classification using KNN (1NN in this project)(Execution Time: 30-35 mins)**

Below is the command line output for task 1:



The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 1: KNN with PCA

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 1.1)

Start Principle Component Analysis (PCA) --🡪 Indicates beginning of PCA process (Task 1.2)

Centerizing image --🡪 Indicates task centerizing the images (Task 1.2.1)

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value --🡪 Plotting eigenvalues (Task 1.2.2)

Plot saved as: knn\_with\_pca\_fold0.png --🡪 Saving the plot above in an image

Calculating new feature set and performing 1-NN for prediction --🡪 Apply KNN (Task 1)

Accuracy for fold1: 92.5% --🡪 Accuracy for fold0

Creating training and testing arrays for fold 1

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_fold1.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold2: 98.75%

Creating training and testing arrays for fold 2

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_fold2.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold3: 92.5%

Creating training and testing arrays for fold 3

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_fold3.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold4: 98.75%

Creating training and testing arrays for fold 4

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

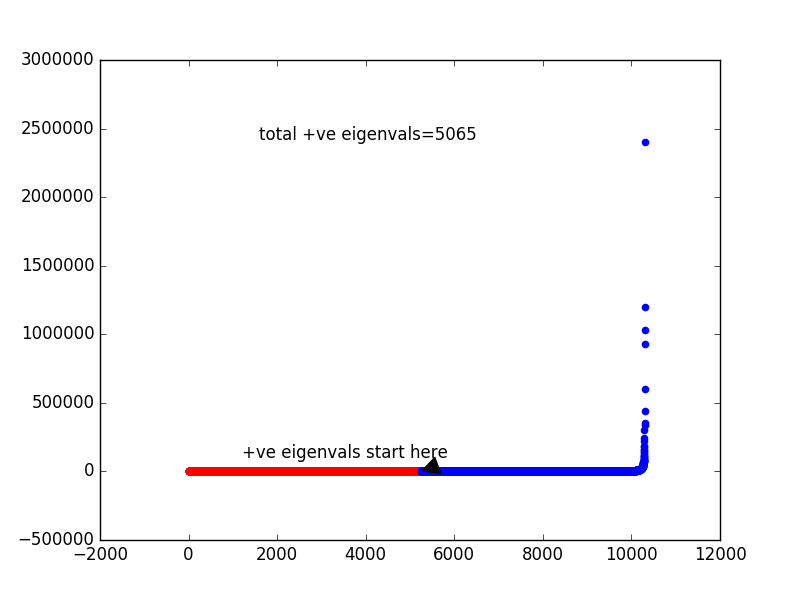
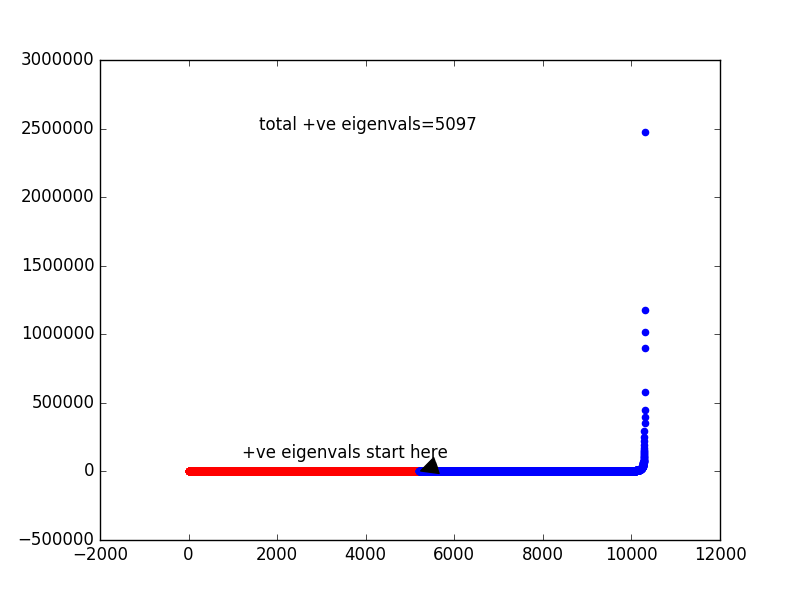
Plot saved as: knn\_with\_pca\_fold4.png

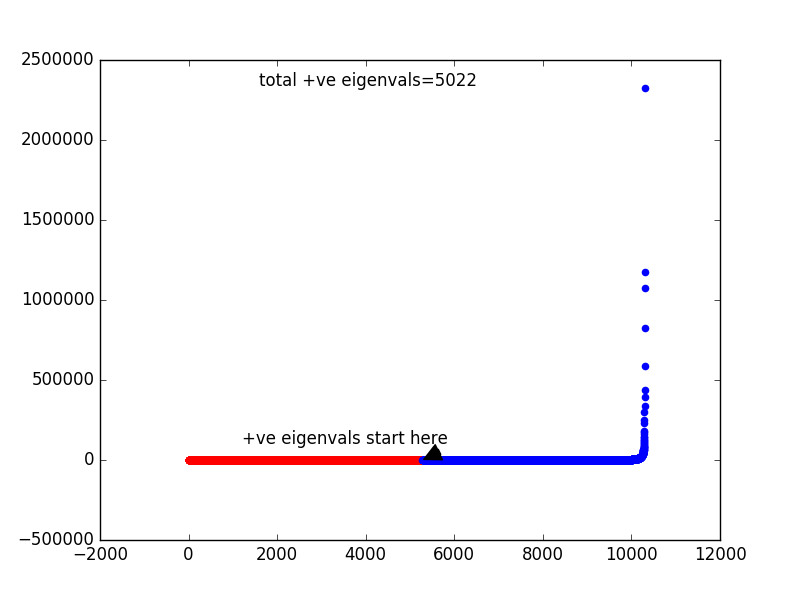
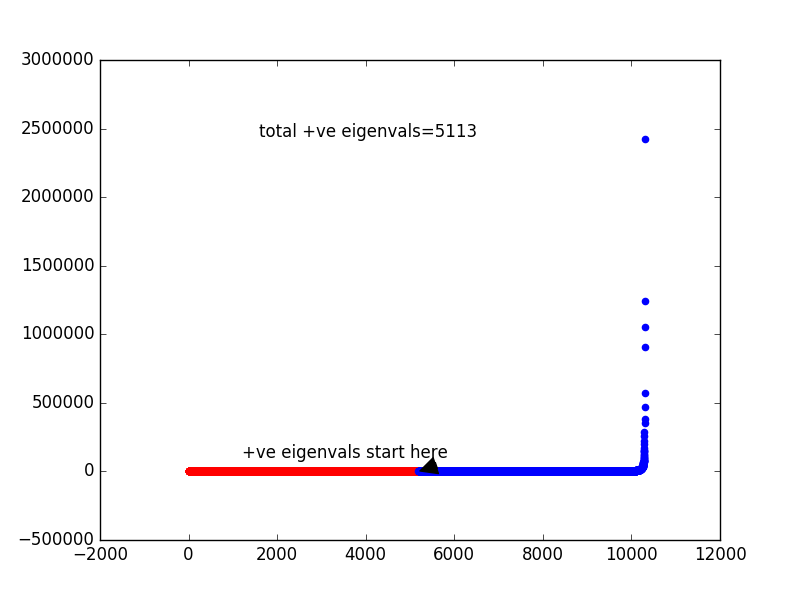
Calculating new feature set and performing 1-NN for prediction

Accuracy for fold5: 95.0%

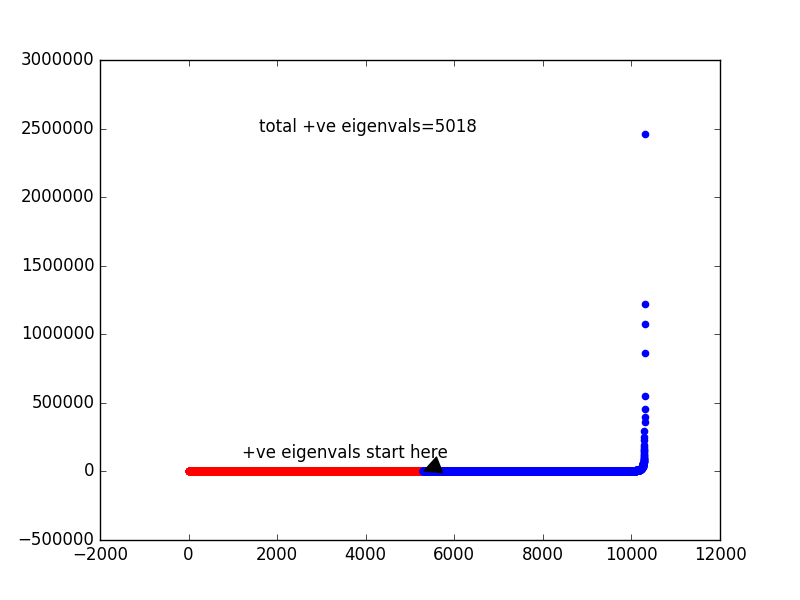
**Final accuracy for 5-fold cross validation is: 95.5%** --🡪 Final Accuracy for all folds

Images given below show all the plots for this task. These images are also included in the zip.



Fold 0 Fold1 

Fold 2 Fold 3



Fold 4

**Task 2: Resize images from 112 x 92 to 56 x 46 and repeat Task 1, compare the new results to the results using un-resized images (Execution Time: 15-20 mins)**

Below is the command line output for task 2:

****

The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 2: KNN with PCA using resized images

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 2.1)

Resizing images --🡪 Indicates resizing images for this task (Task 2)

Start Principle Component Analysis (PCA) --🡪 Indicates beginning of PCA process (Task 2.2)

Centerizing image --🡪 Indicates task centerizing the images (Task 2.2.1)

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value --🡪 Plotting eigenvalues (Task 2.2.2)

Plot saved as: knn\_with\_pca\_resized\_image\_fold0.png --🡪 Saving the plot above in an image

Calculating new feature set and performing 1-NN for prediction --🡪 Apply KNN (Task 2)

Accuracy for fold 1: 92.5% --🡪 Accuracy for fold0

Creating training and testing arrays for fold 1

Resizing images

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_resized\_image\_fold1.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold 2: 98.75%

Creating training and testing arrays for fold 2

Resizing images

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_resized\_image\_fold2.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold 3: 95.0%

Creating training and testing arrays for fold 3

Resizing images

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca\_resized\_image\_fold3.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold 4: 98.75%

Creating training and testing arrays for fold 4

Resizing images

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

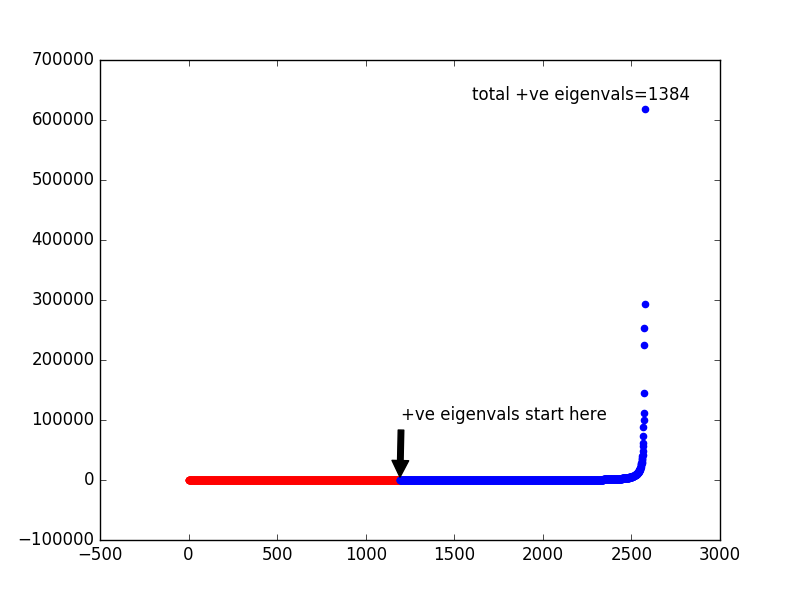
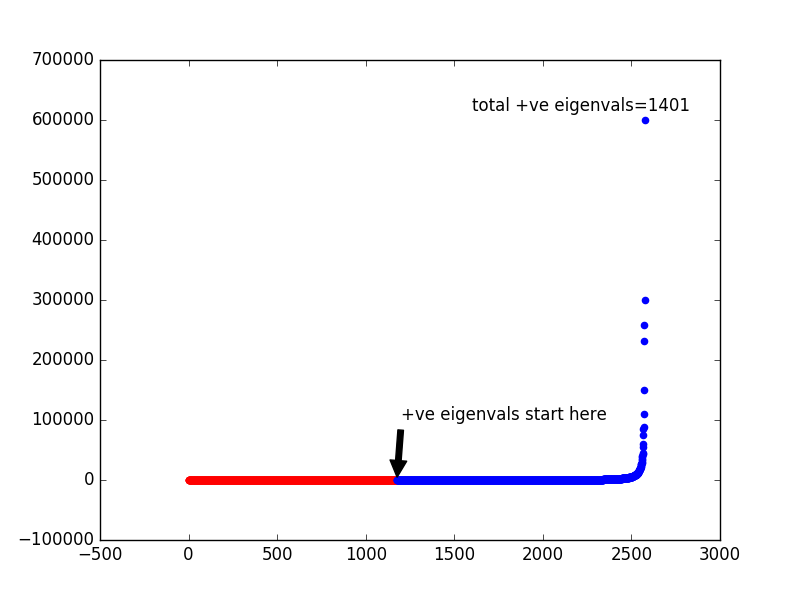
Plot saved as: knn\_with**\_pca\_resized**\_image\_fold4.png

Calculating new feature set and performing 1-NN for prediction

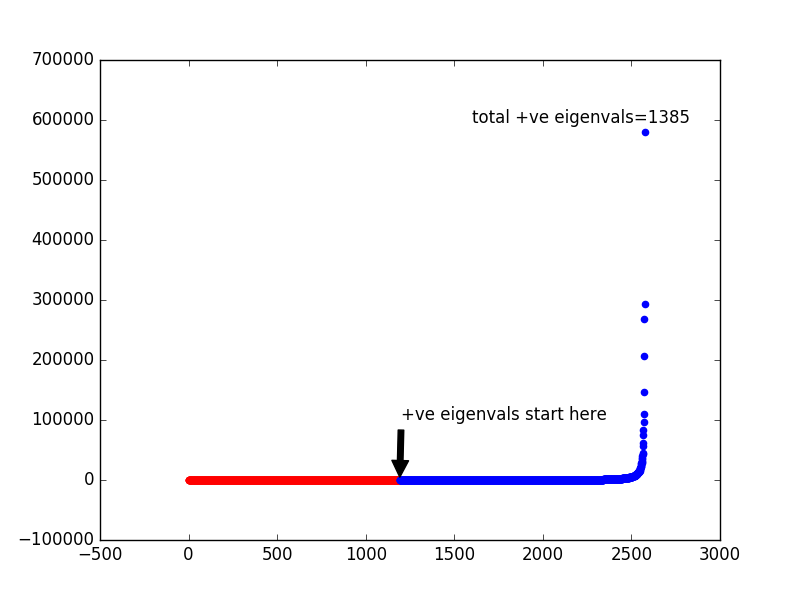
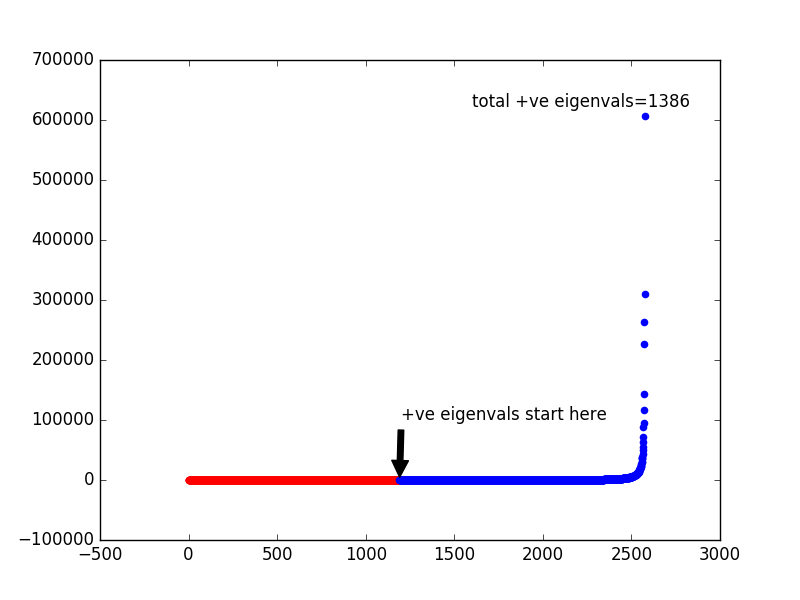
Accuracy for fold 5: 95.0%

**Final accuracy for 5-fold cross validation is: 96.0%** --🡪 Final Accuracy for all folds

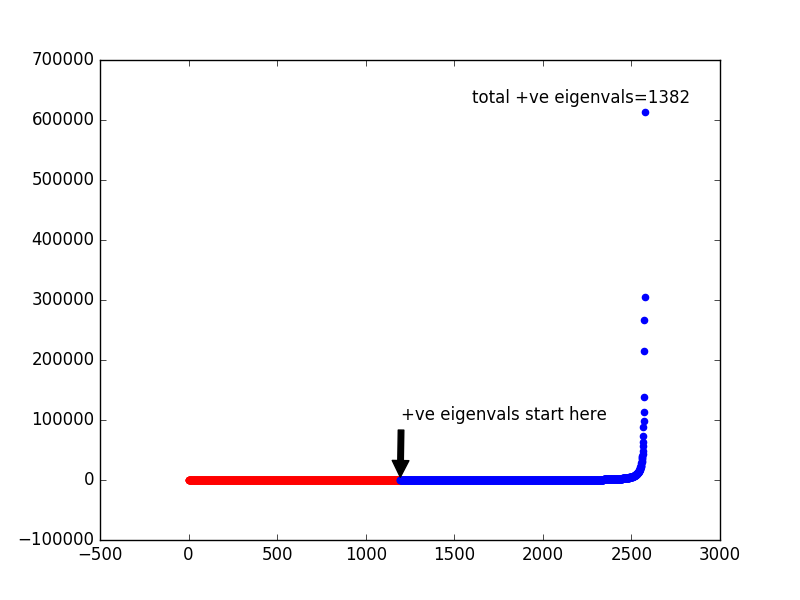
Images given below show all the plots for this task. These images are also included in the zip.

** **

Fold 0 Fold 1

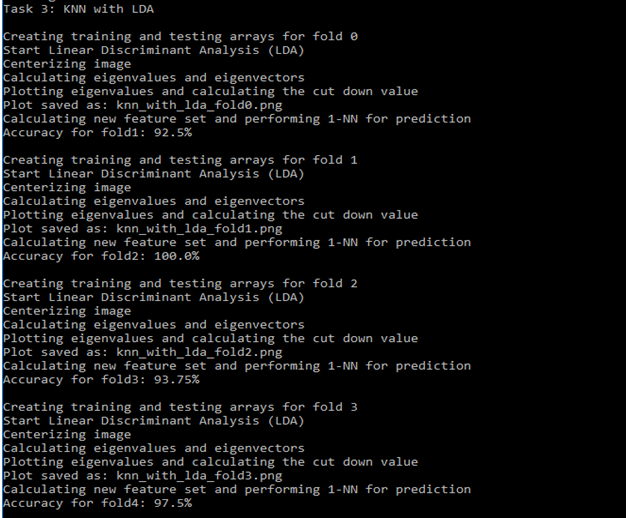
Fold 2 Fold 3

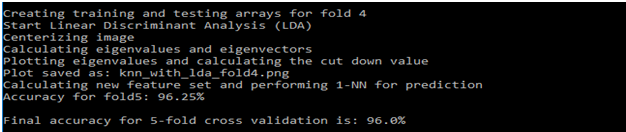


Fold 4

**Task 3: Apply LDA to replace PCA for dimensionality reduction and repeat Task 1 (Execution Time: 60-65 mins)**

Below is the command line output for task 3:

****



The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 3: KNN with LDA

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 3.1)

Start Linear Discriminant Analysis (LDA) ) --🡪 Indicates beginning of LDA process (Task 3.2)

Centerizing image --🡪 Indicates task centerizing the images (Task 3.2.1)

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value --🡪 Plotting eigenvalues (Task 3.2.2)

Plot saved as: knn\_with\_lda\_fold0.png --🡪 Saving the plot above in an image

Calculating new feature set and performing 1-NN for prediction --🡪 Apply KNN (Task 3)

Accuracy for fold1: 92.5% --🡪 Accuracy for fold0

Creating training and testing arrays for fold 1

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_lda\_fold1.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold2: 100.0%

Creating training and testing arrays for fold 2

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_lda\_fold2.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold3: 93.75%

Creating training and testing arrays for fold 3

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_lda\_fold3.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold4: 97.5%

Creating training and testing arrays for fold 4

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

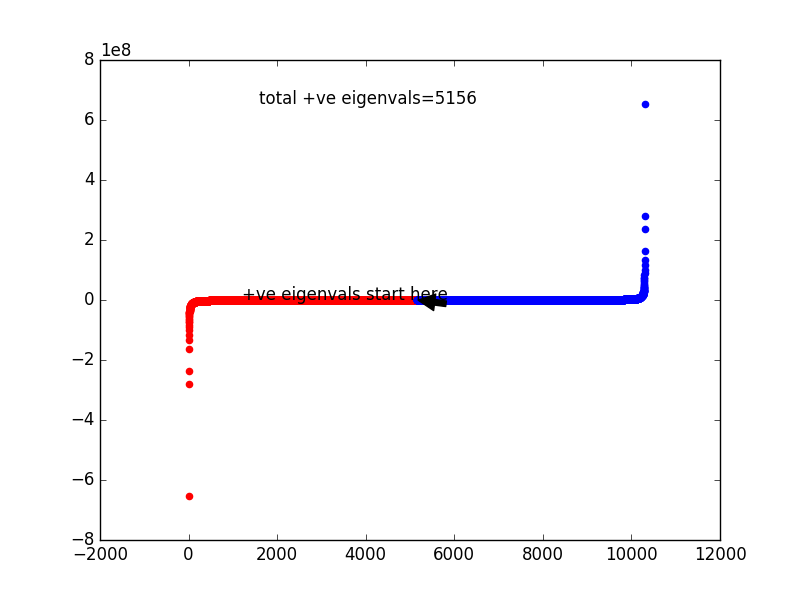
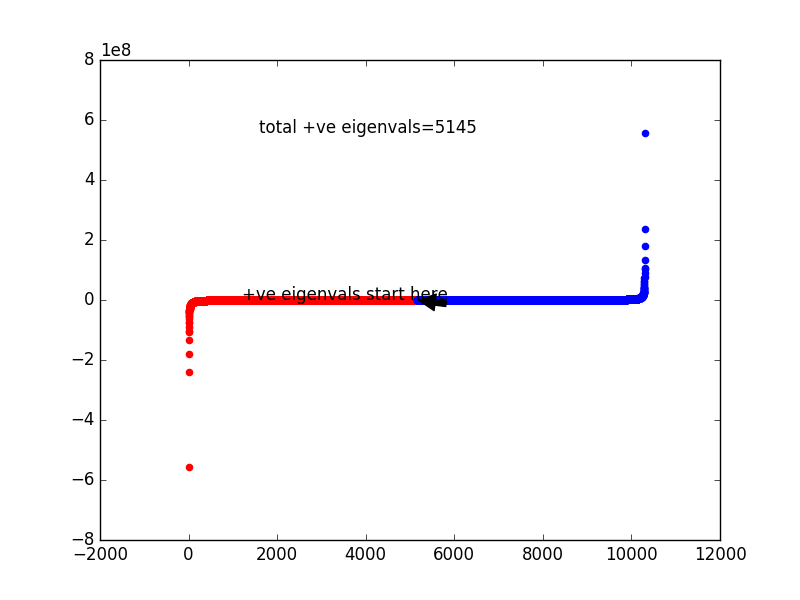
Plot saved as: knn\_with\_lda\_fold4.png

Calculating new feature set and performing 1-NN for prediction

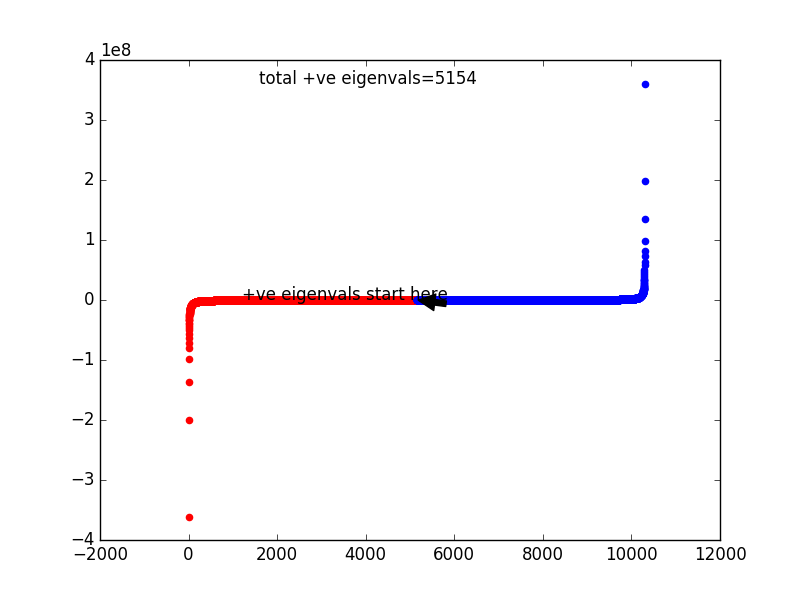
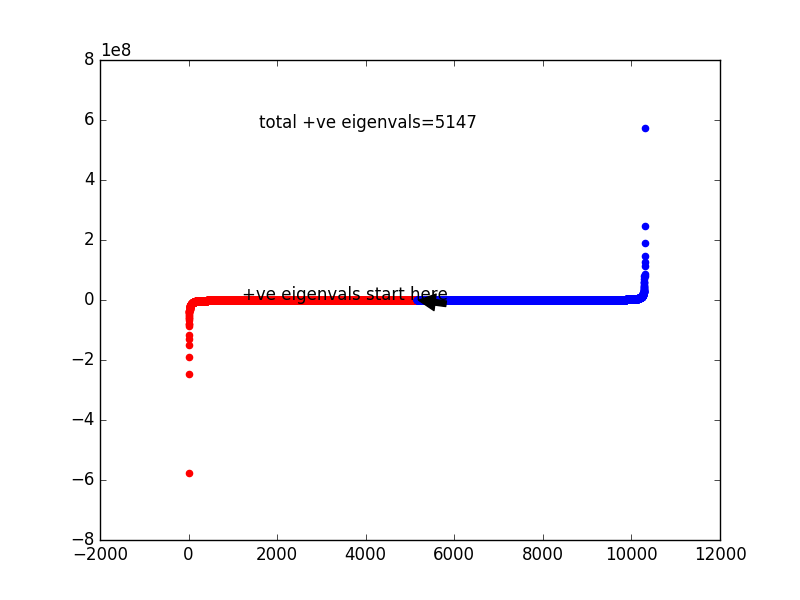
Accuracy for fold5: 96.25%

**Final accuracy for 5-fold cross validation is: 96.0%** --🡪 Final Accuracy for all folds

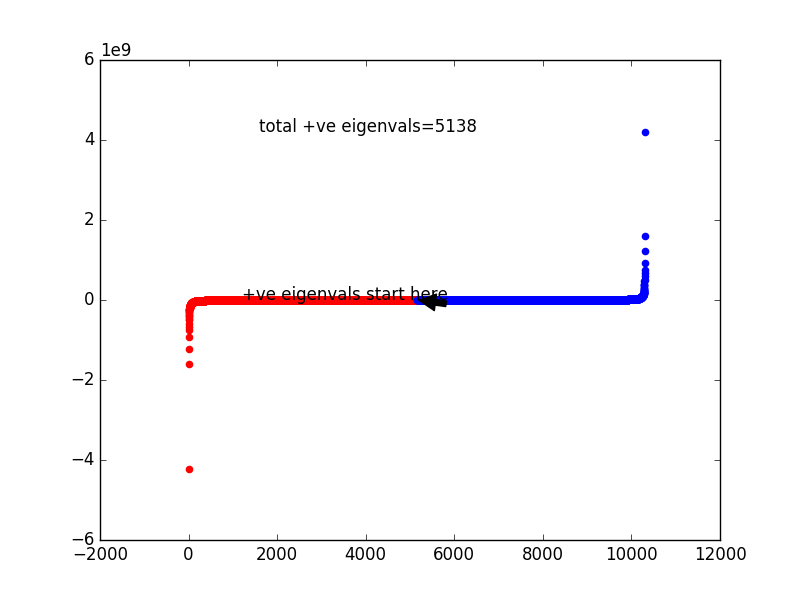
Images given below show all the plots for this task. These images are also included in the zip.

** **

Fold 0 Fold 1

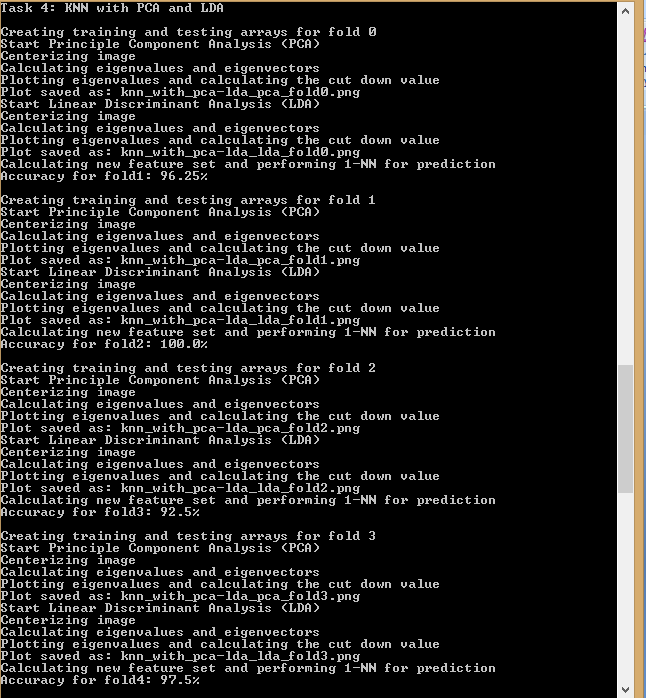
Fold 2 Fold 3

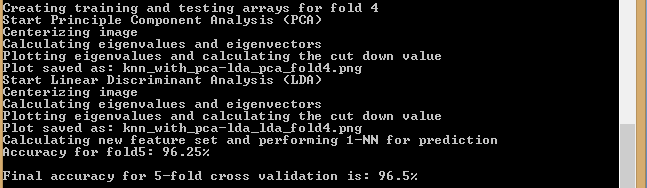


Fold 4

**Task 4: Repeat Task 3, but run PCA first to reduce the image dimensionality to the number of training data, then using LDA to reduce the image dimensionality (Execution Time: 25-30 mins)**

Below is the command line output for task 4:

****

****

The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 4: KNN with PCA and LDA

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 4.1)

Start Principle Component Analysis (PCA) --🡪 Indicates beginning of PCA process (Task 4.2)

Centerizing image --🡪 Indicates task centerizing the images (Task 4.2.1)

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value --🡪 Plotting eigenvalues (Task 4.2.2)

Plot saved as: knn\_with\_pca-lda\_pca\_fold0.png --🡪 Saving the plot above in an image

Start Linear Discriminant Analysis (LDA) --🡪 Indicates beginning of LDA process (Task 4.2)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value --🡪 Plotting eigenvalues (Task 4.2.2)

Plot saved as: knn\_with\_pca-lda\_lda\_fold0.png --🡪 Saving the plot above in an image

Calculating new feature set and performing 1-NN for prediction --🡪 Apply KNN (Task 4)

Accuracy for fold1: 96.25% --🡪 Accuracy for fold1

Creating training and testing arrays for fold 1

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_pca\_fold1.png

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_lda\_fold1.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold2: 100.0%

Creating training and testing arrays for fold 2

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_pca\_fold2.png

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_lda\_fold2.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold3: 92.5%

Creating training and testing arrays for fold 3

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_pca\_fold3.png

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_lda\_fold3.png

Calculating new feature set and performing 1-NN for prediction

Accuracy for fold4: 97.5%

Creating training and testing arrays for fold 4

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: knn\_with\_pca-lda\_pca\_fold4.png

Start Linear Discriminant Analysis (LDA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

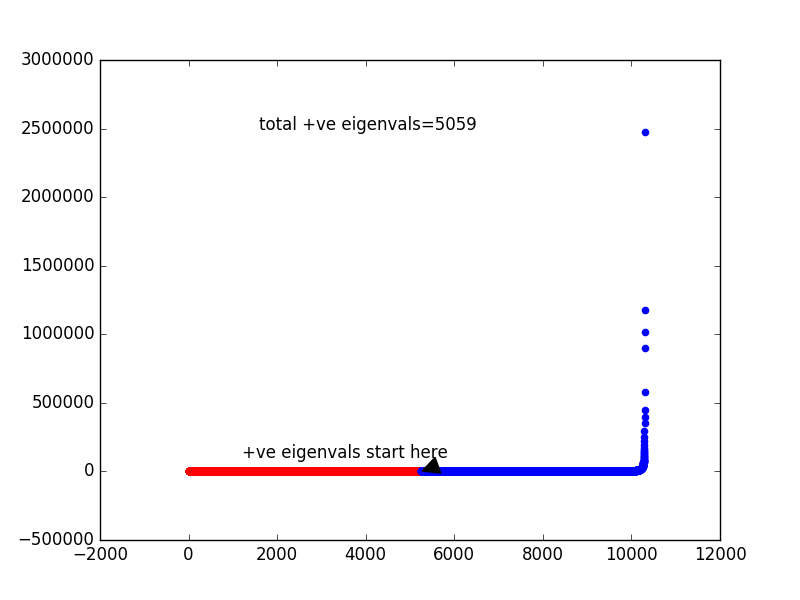
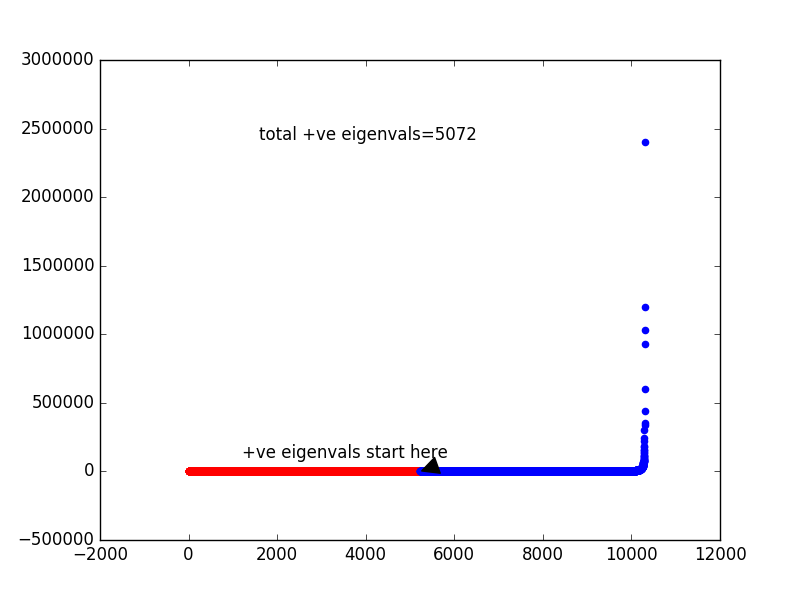
Plot saved as: knn\_with\_pca-lda\_lda\_fold4.png

Calculating new feature set and performing 1-NN for prediction

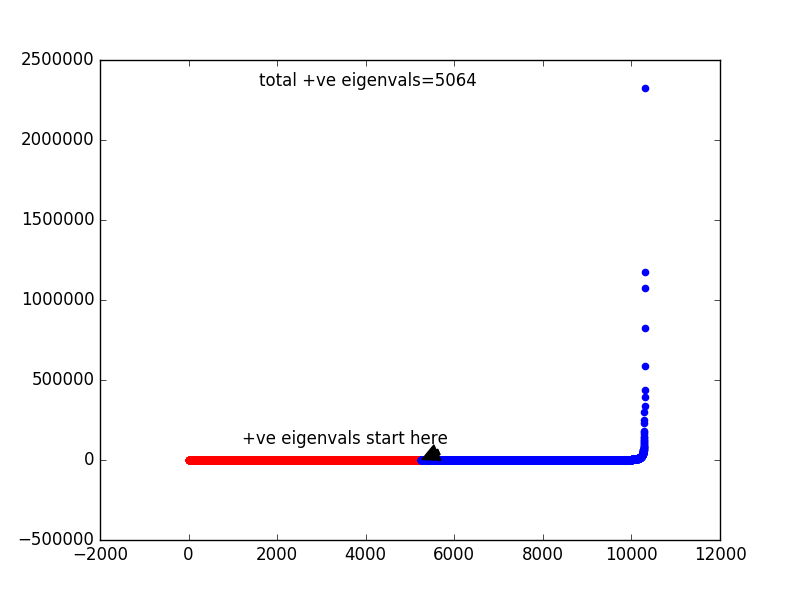
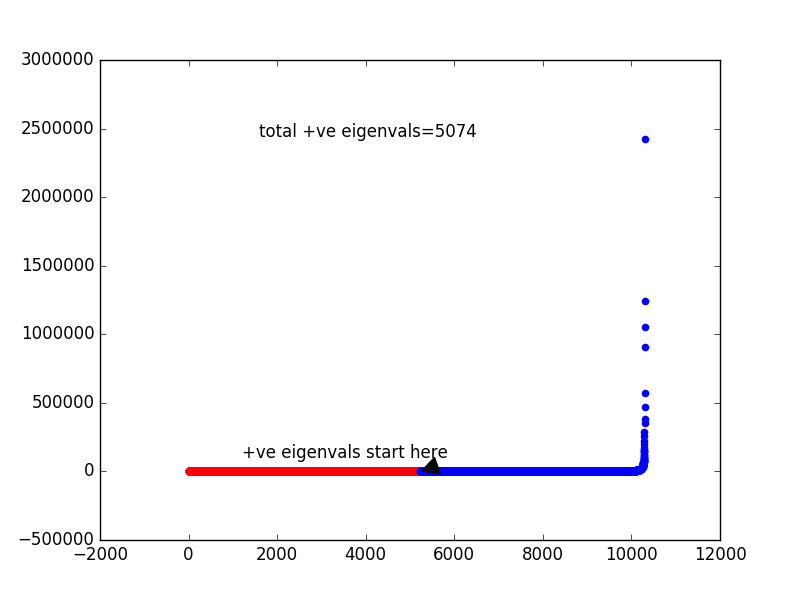
Accuracy for fold5: 96.25%

**Final accuracy for 5-fold cross validation is: 96.5%** --🡪 Final Accuracy for all folds

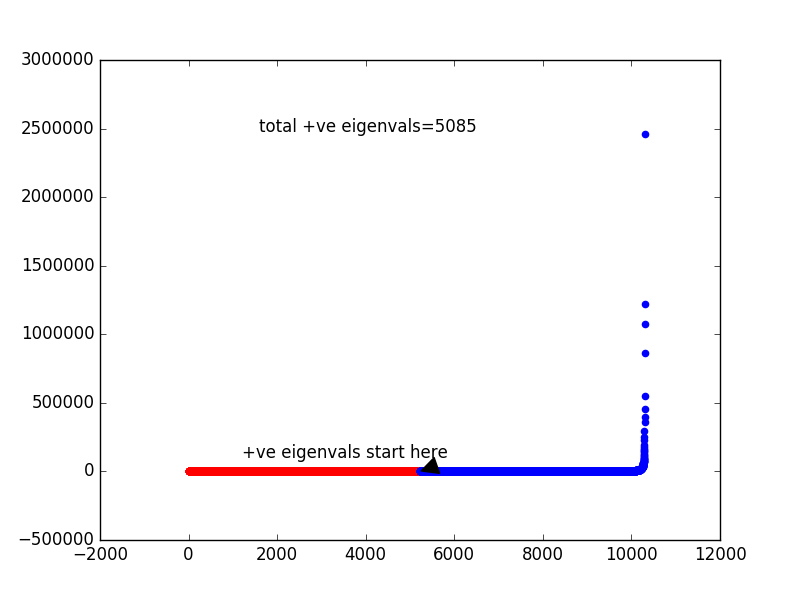
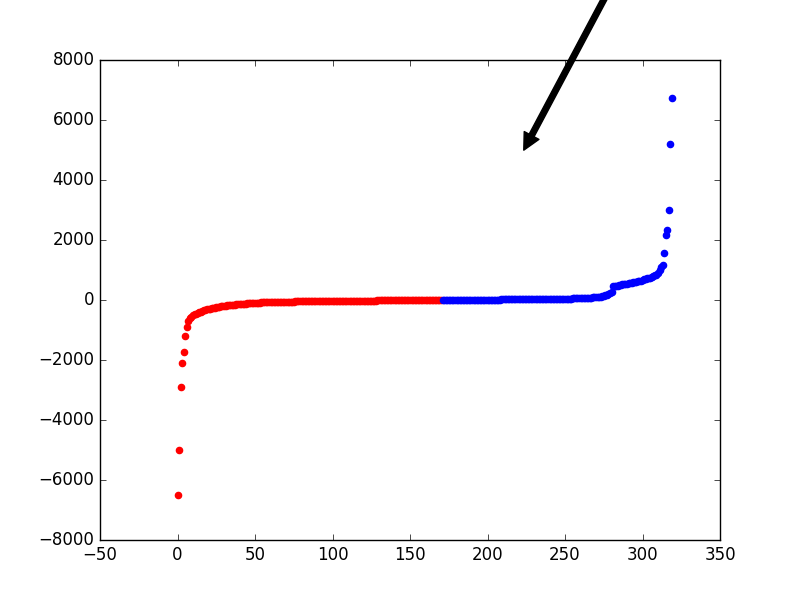
Images given below show all the plots for this task. These images are also included in the zip.

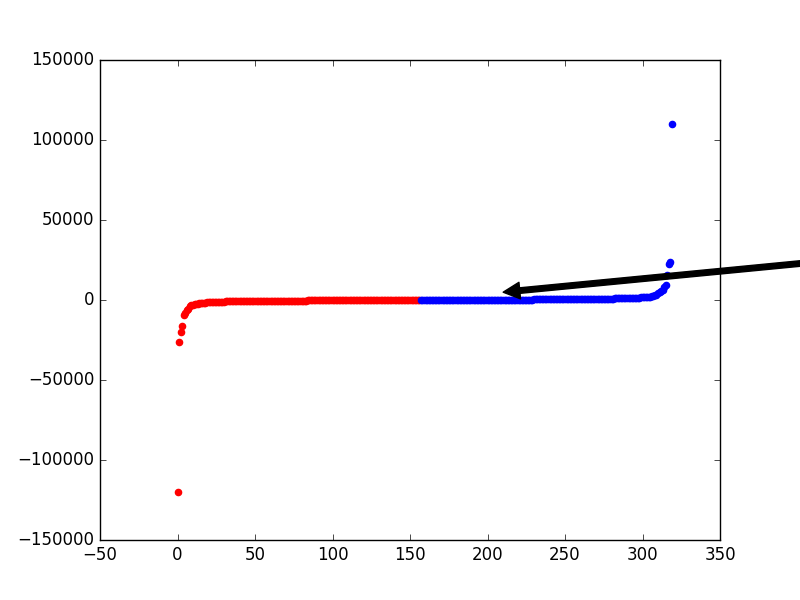
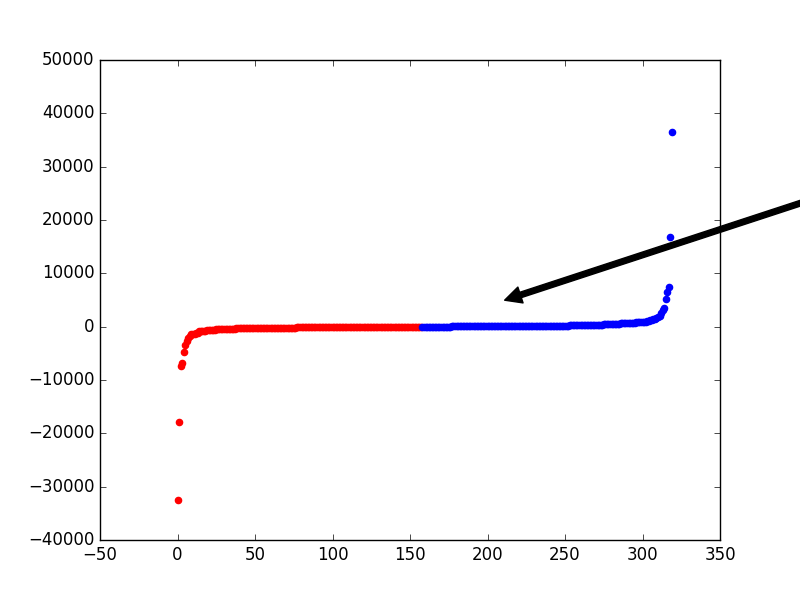
PCA-Fold 0 PCA-Fold 1

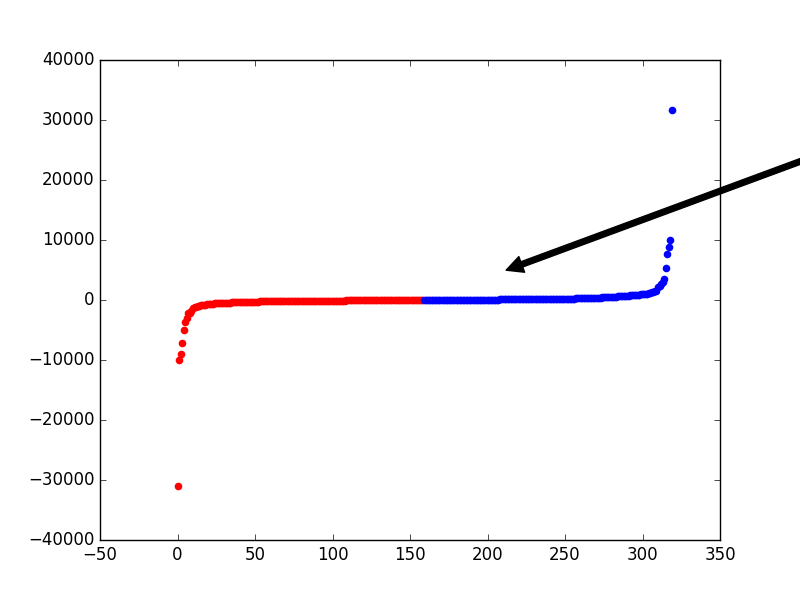
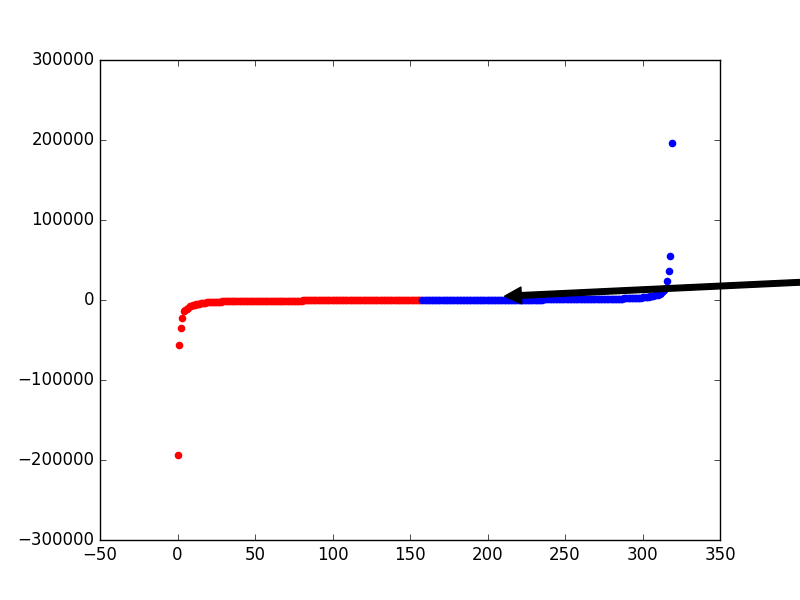
PCA-Fold 2 PCA-Fold 3

PCA-Fold 4 LDA-Fold 0

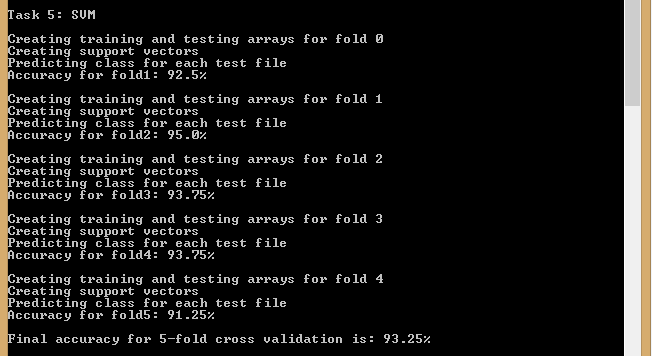
LDA-Fold 1 LDA-Fold 2

LDA-Fold 3 LDA-Fold 4

**Task 5: Implement kernel SVM to do classification with 5-fold cross validation (Execution Time: 7-12 mins)**

Below is the command line output for task 5:

****

The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 5: SVM

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 5.1)

Creating support vectors --🡪 Indicates creating support vectors using one-vs-rest for all classes

Predicting class for each test file --🡪 Indicates using the support vectors for predicting class

Accuracy for fold1: 92.5% --🡪 Accuracy for fold1

Creating training and testing arrays for fold 1

Creating support vectors

Predicting class for each test file

Accuracy for fold2: 95.0%

Creating training and testing arrays for fold 2

Creating support vectors

Predicting class for each test file

Accuracy for fold3: 93.75%

Creating training and testing arrays for fold 3

Creating support vectors

Predicting class for each test file

Accuracy for fold4: 93.75%

Creating training and testing arrays for fold 4

Creating support vectors

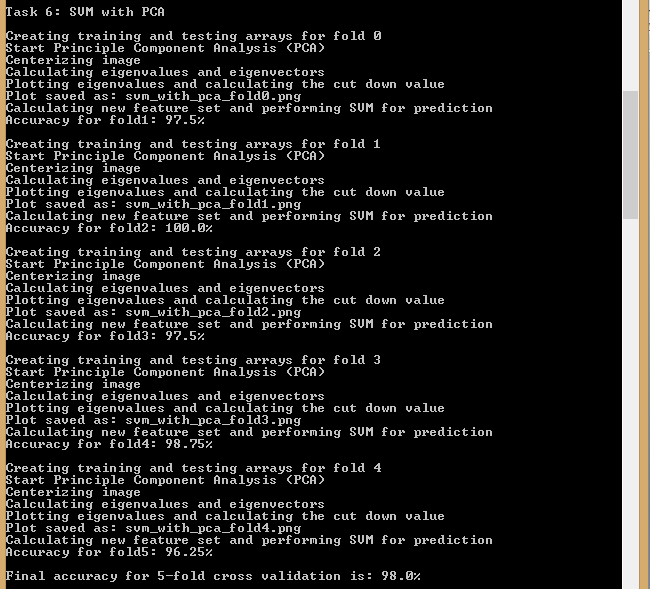
Predicting class for each test file

Accuracy for fold5: 91.25%

**Final accuracy for 5-fold cross validation is: 93.25%** --🡪 Final Accuracy for all folds

**Task 6: Repeat Task 5, but run PCA to reduce the image dimensionality first (Execution Time: 30-35 mins)**

Below is the command line output for task 6:

****

The output is also given as a text below for easier analysis. The description is given in red for the first fold and can be extrapolated similarly for the other folds as well.

Task 6: SVM with PCA

Creating training and testing arrays for fold 0 --🡪 Indicates creating data for fold0 (Task 6.1)

Start Principle Component Analysis (PCA) --🡪 Indicates beginning of PCA process (Task 6.2)

Centerizing image --🡪 Indicates task centerizing the images (Task 6.2.1)

Calculating eigenvalues and eigenvectors --🡪 Plotting eigenvalues (Task 6.2.2)

Plotting eigenvalues and calculating the cut down value

Plot saved as: svm\_with\_pca\_fold0.png --🡪 Saving the plot above in an image

Calculating new feature set and performing SVM for prediction --🡪 Predicting class

Accuracy for fold1: 97.5% --🡪 Accuracy for fold1

Creating training and testing arrays for fold 1

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: svm\_with\_pca\_fold1.png

Calculating new feature set and performing SVM for prediction

Accuracy for fold2: 100.0%

Creating training and testing arrays for fold 2

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: svm\_with\_pca\_fold2.png

Calculating new feature set and performing SVM for prediction

Accuracy for fold3: 97.5%

Creating training and testing arrays for fold 3

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

Plot saved as: svm\_with\_pca\_fold3.png

Calculating new feature set and performing SVM for prediction

Accuracy for fold4: 98.75%

Creating training and testing arrays for fold 4

Start Principle Component Analysis (PCA)

Centerizing image

Calculating eigenvalues and eigenvectors

Plotting eigenvalues and calculating the cut down value

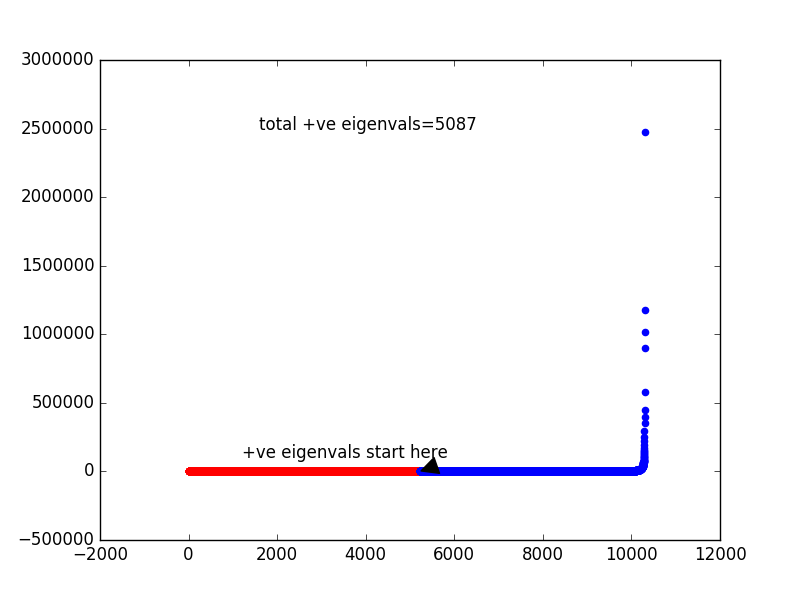
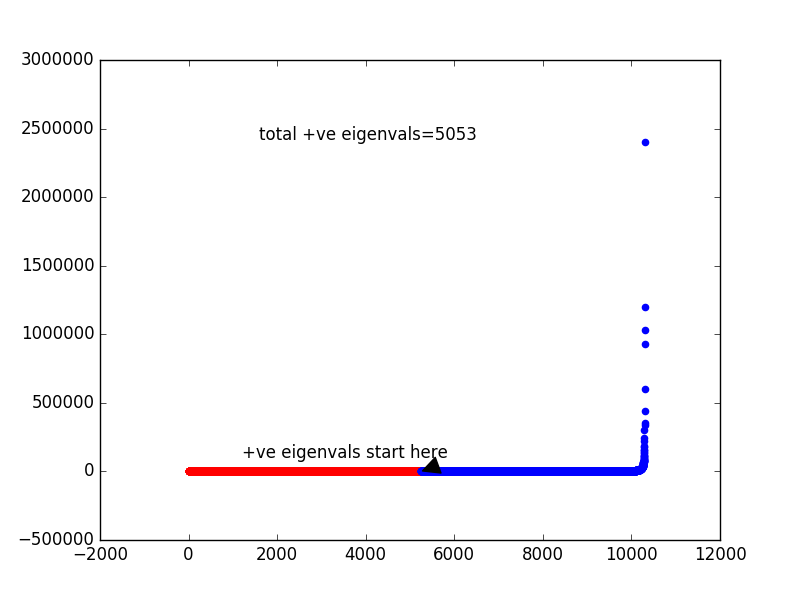
Plot saved as: svm\_with\_pca\_fold4.png

Calculating new feature set and performing SVM for prediction

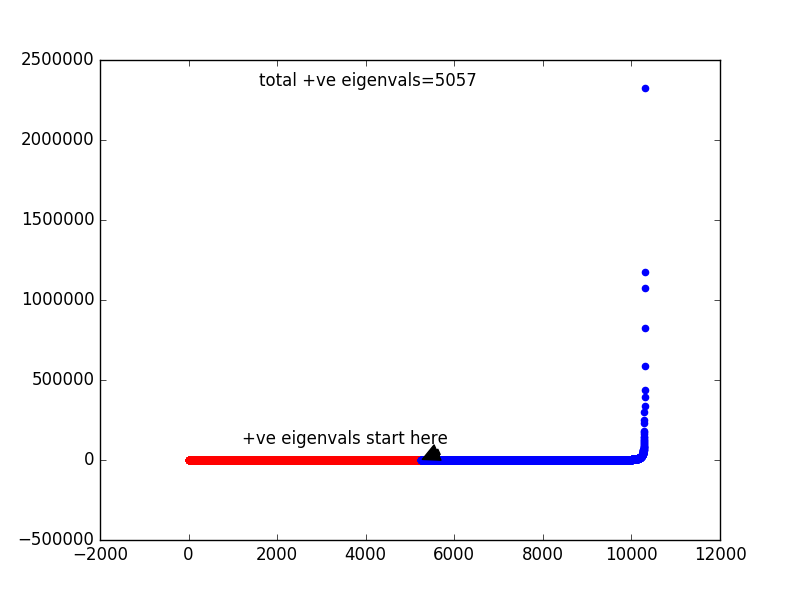
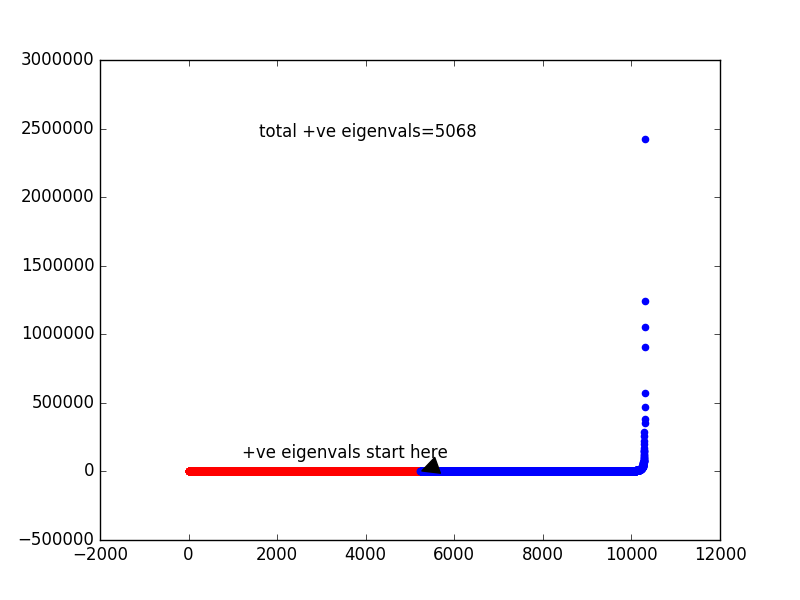
Accuracy for fold5: 96.25%

**Final accuracy for 5-fold cross validation is: 98.0%** --🡪 Final Accuracy for all folds

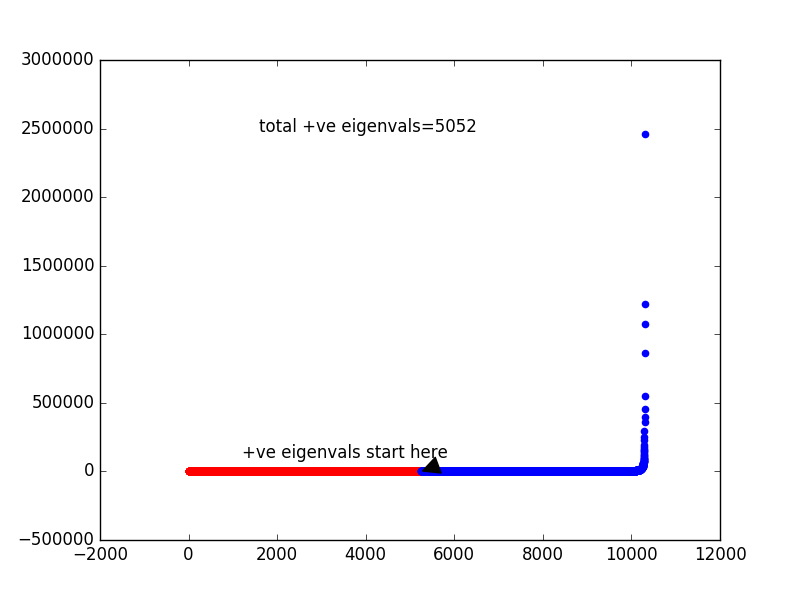
Images given below show all the plots for this task. These images are also included in the zip.

** **

Fold 0 Fold 1

Fold 2 Fold 3



Fold 4

**OUTPUT ANALYSIS AND QUESTIONS IN THE TASKS**

The accuracy for all the tasks is provided again here for easier analysis:

* Task 1: 95.5%
* Task 2: 96.0%
* Task 3: 96.0%
* Task 4: 96.5%
* Task 5: 93.25%
* Task 6: 98.0%

It can be seen from the above accuracies that using PCA followed by LDA provides the best results for KNN algorithm, in terms of a combination of accuracy and time taken for execution.

Task 2 question: Compare results

The images for task 2 are resized to almost half the size of the original image. As it can be seen from the accuracies above, the current resize factor has not affected the accuracy much, which means that the recognition algorithm still works as expected. But if the time of execution is considered, there is a big change, since task 1 takes around 35 minutes to execute while task 2 takes about half the time, which means it boosts the execution speed of the algorithm. The image size if further reduced might affect the accuracy of the algorithm, but with the current value the difference is very less, such that it does not affect the algorithm accuracy much.

Task 6 question: is KNN or SVM more sensitive to the high dimensionality of data?

Considering only time of execution as factor, KNN is more sensitive to high dimensional data, since a reduction in the number of dimensions significantly reduces the rate of execution, while SVM is works faster even when the dimensionality is high of reduced using PCA. Considering accuracy as a factor, PCA works well on reducing the image dimensionality, hence reduction in dimensions does not affect the accuracy a lot, while for SVM the results improved by using dimension reduction, so we can say that for high dimensional data SVM works better when reduction techniques are applied.

**ADDITIONAL INFORMATION**

**CVXOPT**

The cvxopt.solvers.qp function is used to solve quadratic programming. Its format is as follows:

cvxopt.solvers.qp(P, q, G, h, A, b)

The following values of the imputs are used for our SVM problem:

b is a matrix with a single value 0

A is a matrix of size 1x200 with the value of y. [200 is the number of samples in training set and y is the list of classes these samples]

h is a matrix of size 1x200 of all zeroes

G is a matrix of size 200x200, with all values as 1, except the diagonal values as -1

q is a matrix of size 1x200, with all values as -1

P is a matrix which contains the product of the dot products of x and x transpose and y and y transpose. [x is the input data set and y is the list of all classes for the dataset]