HW1

February 7, 2016

1 Homework 1 | Linear Clasification and Subspaces

Course: CS7616
Name: Sumant Hanumante
GT ID: shanumante3

1.1 Plotting eigenvetors for PCA/LDA

- Both PCA and LDA use eigendecomposition to figure out the direction of projection for the given data.
- PCA computes the eigenvectors of the covariance matrix and the data is then rotated so as to get a diagonal covariance matrix. We may also drop the eigenvectors with low eigenvalues to reduce data dimentionality. PCA does not care about the classes and minimizes the reconstruction error.
- LDA uses the eigenvector corresponding to the largest eigenvalue of inv(Sw) * Sb [reference] as the direction of projection. LDA tries to project data on a plane that maximizes separation and generally would not have good reconstruction characteristics.

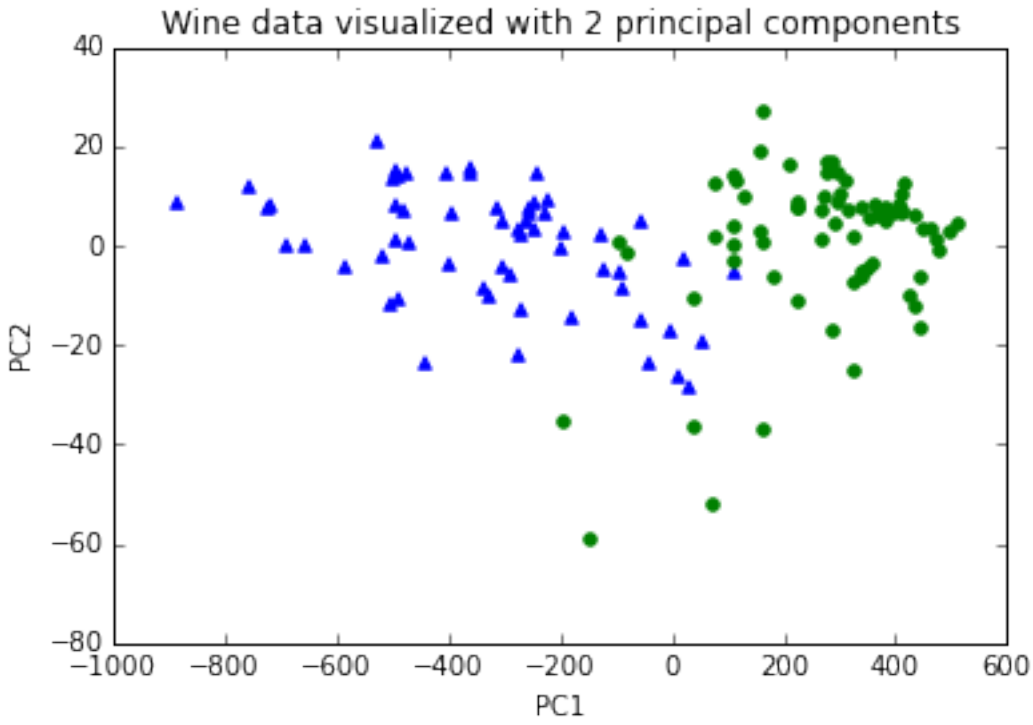
1.1.1 Wine dataset

- For wine dataset, the eigenvectors are being shown for the entire dataset since computing eigenvalues for just 10 test samples wouldn't make much sense. In case LDA/PCA is used later on during classification, we use **only train data** and not the entire set as is done here.

PCA

```
1st Eigenval :  124313.073445
Eigenvec:  [ -1.90615566e-03   2.68379014e-04  -2.56407459e-04   4.11826686e-03
  -1.86561171e-02  -7.83110101e-04  -1.14208106e-03   1.02136703e-04
  -4.20937723e-04  -3.57515943e-03  -5.58270451e-05  -3.08914143e-04
  -9.99808098e-01]


13th Eigenval :  0.0061339899981
Eigenvec:  [ -5.62139324e-03  -1.16925805e-02  -1.38330530e-01  -1.58640158e-03
   8.21297180e-04   5.70618175e-02  -1.96501389e-02   9.82314109e-01
   1.95062440e-02  -2.53841717e-03  -5.44626521e-02   9.35780237e-02
   7.42889379e-05]
```


Wine data visualized with 2 principal components
PC2
PC1
40
20
0
−20
−40
−60
−80
−1000
−800
−600
−400
−200
0
200
400
600

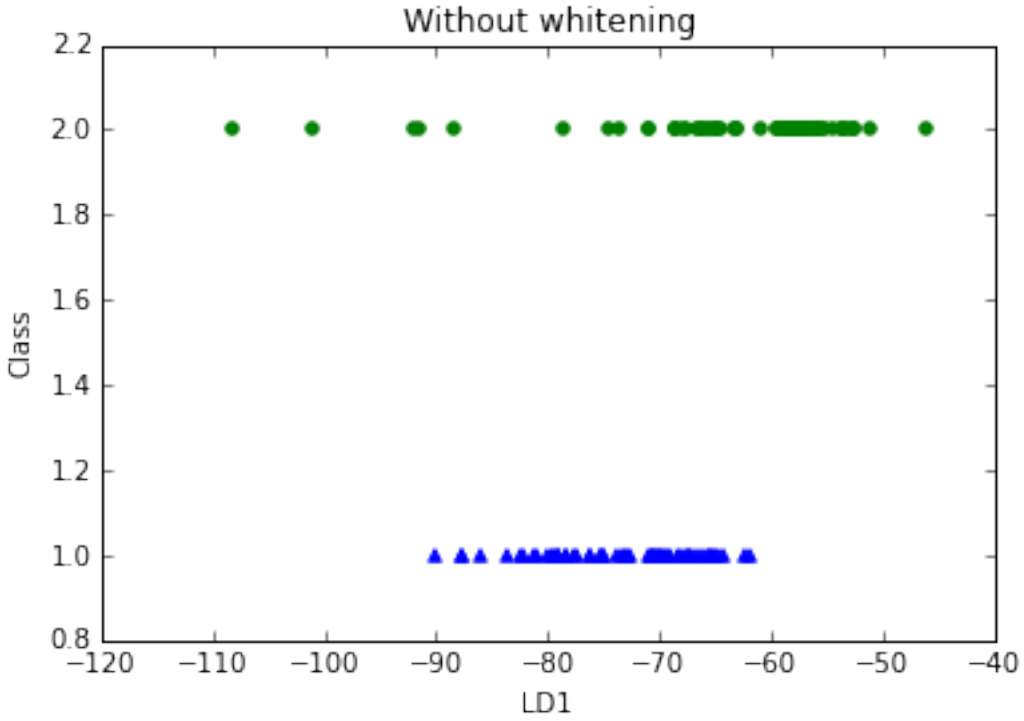
LDA

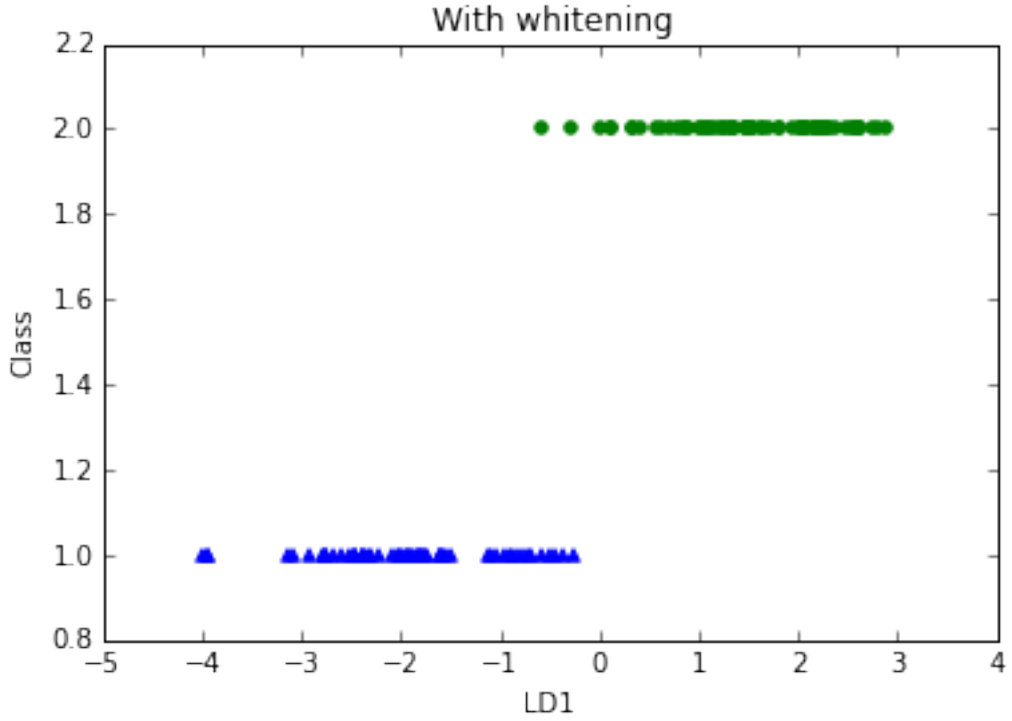
```
1st Eigenval :  18.5261341576
Eigenvec:  [-0.10252817 -0.00719884 -0.03366967  0.18977244 -0.6612435   0.2025607
 -0.23774331  0.23323152  0.13104315 -0.11850553  0.13323114 -0.5604677
 -0.00241784]


13th Eigenval :  -17.9204800319
Eigenvec:  [ 0.07401133  0.00263669 -0.00346238 -0.17458011  0.67706755  0.27899188
 -0.16271673  0.2345019   0.15638762  0.05952962  0.13670542 -0.54736453
 -0.0017877 ]
```

- Note that LDA gives negative eigenvalues. This is because the Sw-1 * Sb is not guaranteed to be a real-symmetric matrix.
- PCA on the other hand always gives positive eigenvalues since the covariance matrix is real symmetric.
- On the next page, we visualize the effect of whitening on the eigenvectors produced by LDA. We observe that whitening creates a good separation between classes as compared to non-whitened data.

Visualizing effect of whitening on data split after LDA

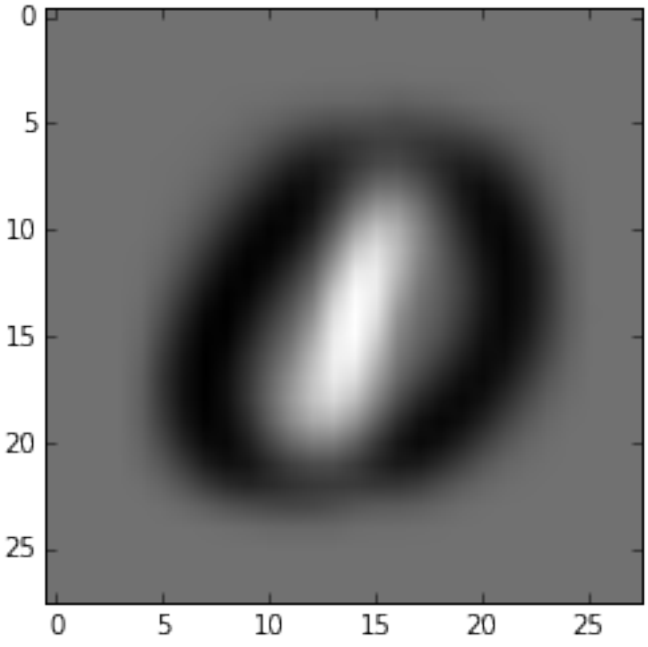




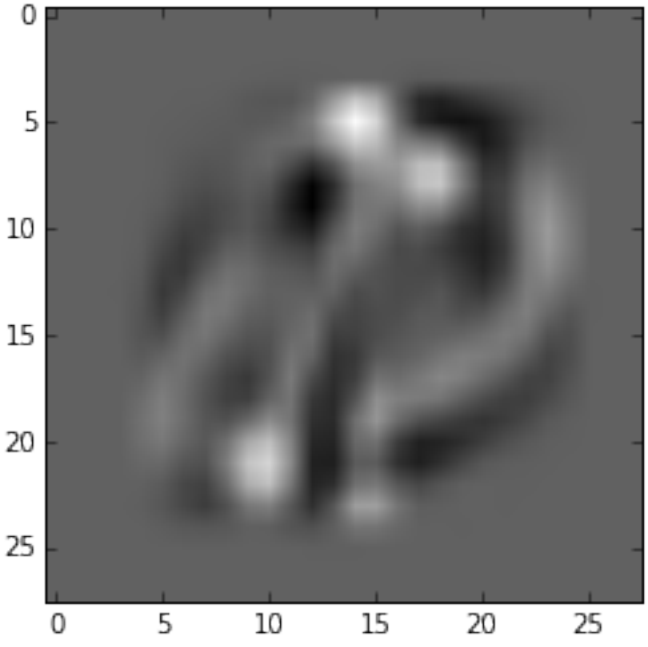
1.1.2 MNIST

PCA

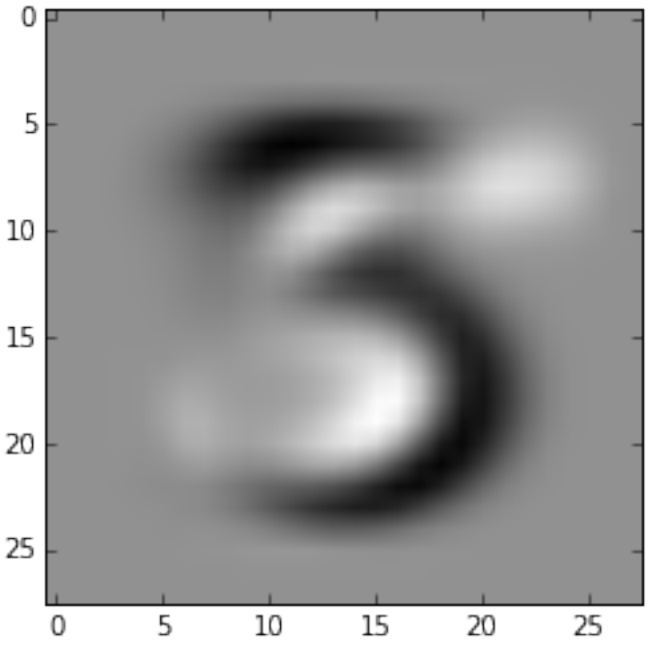
```
1st Eigenval :  3.987009616  1st eigenvector:
```



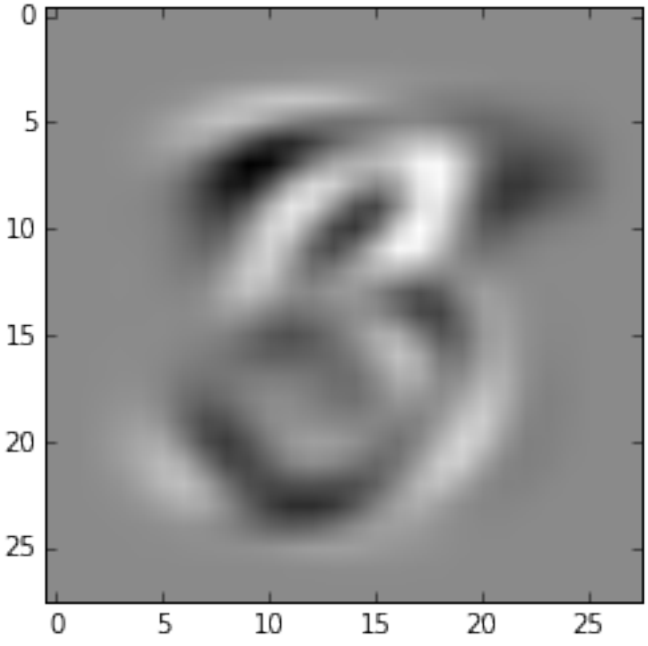

```
20th Eigenval :  0.59474162583  20th eigenvector:
```



```
1st Eigenval :  2.42917456171  1st Eigenvec
```




```
20th Eigenval :  0.71846340603  20th Eigenvec
```

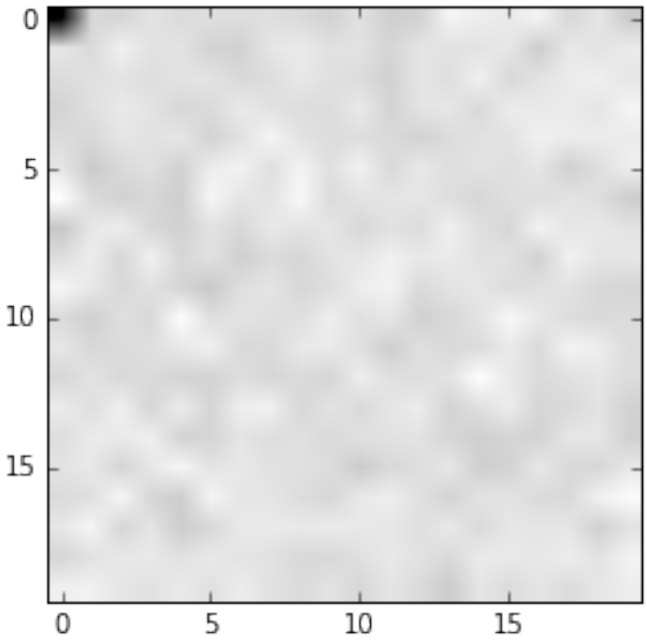


LDA

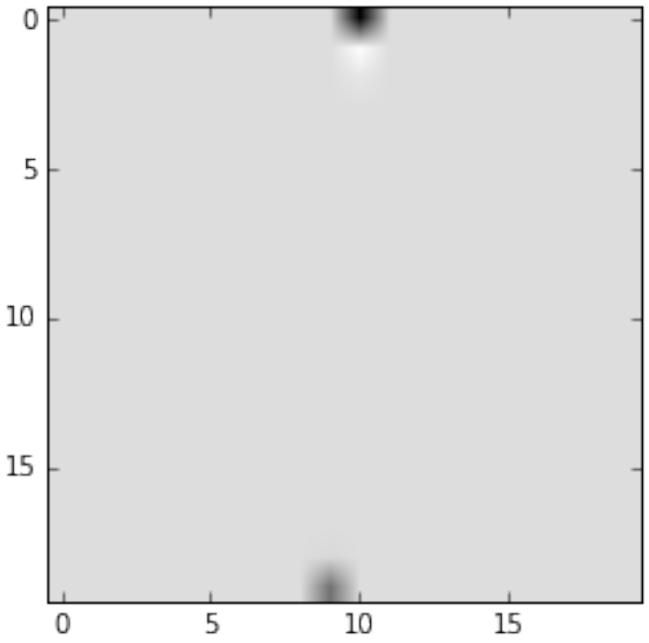
- For LDA, both the scatter matrices turn out to be illconditioned (I think this happens because there are multiple pixels in the dataset which are always 0 thus giving singular matrices).
- One way around this issue is to remove all 0-ONLY rows from the scatter matrices.
- However, given that we already have PCA, it makes more sense to project the data onto the principal basis set and then proceed with LDA (as has been done below).
- The eigenvectors that we would end up visualizing for LDA would thus be the "PCA-conditioned" versions and it would be incorrect to draw any conclusions from the eigenvectors alone.
- **From here onwards, wherever we use LDA on MNIST data, we first transform the data using PCA (the number of components are found empirically) and then perform further analysis.**

```
MNIST 01

1st Eigenval :  142.23946395  Eigenvec:
```

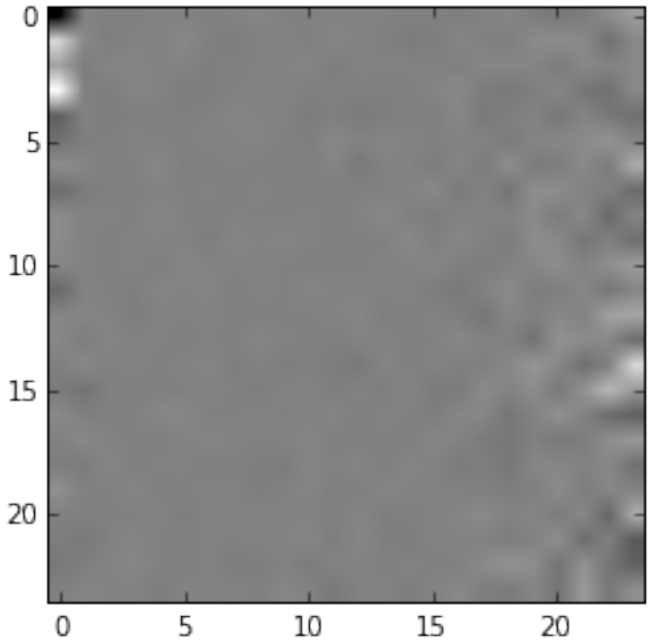



```
20th Eigenval :  -1.53140124801e-08  Eigenvec:
```

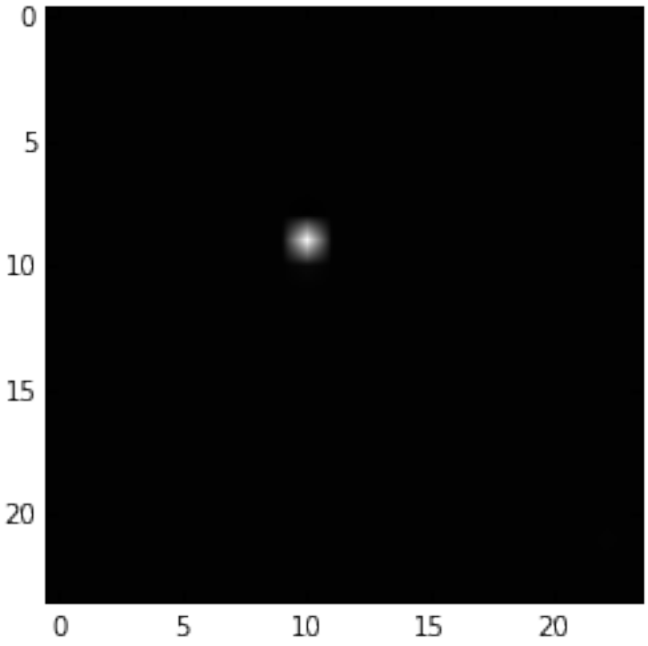


MNIST 35

1st Eigenval : 42.2552762118 Eigenvec:



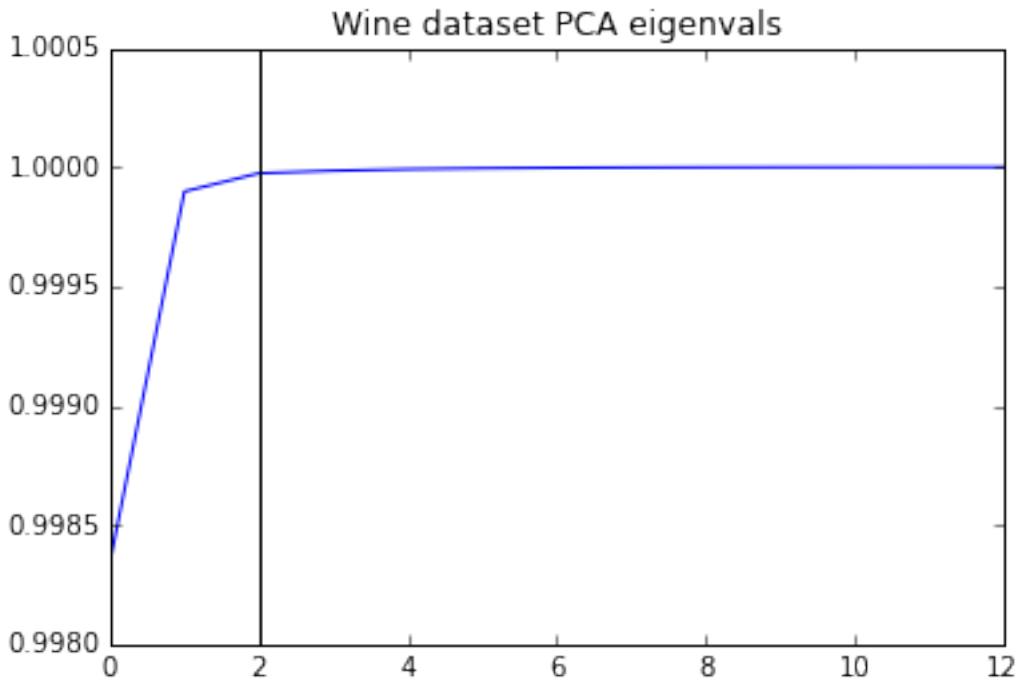
```
20th Eigenval :  -5.70216598569e-10  Eigenvec:
```

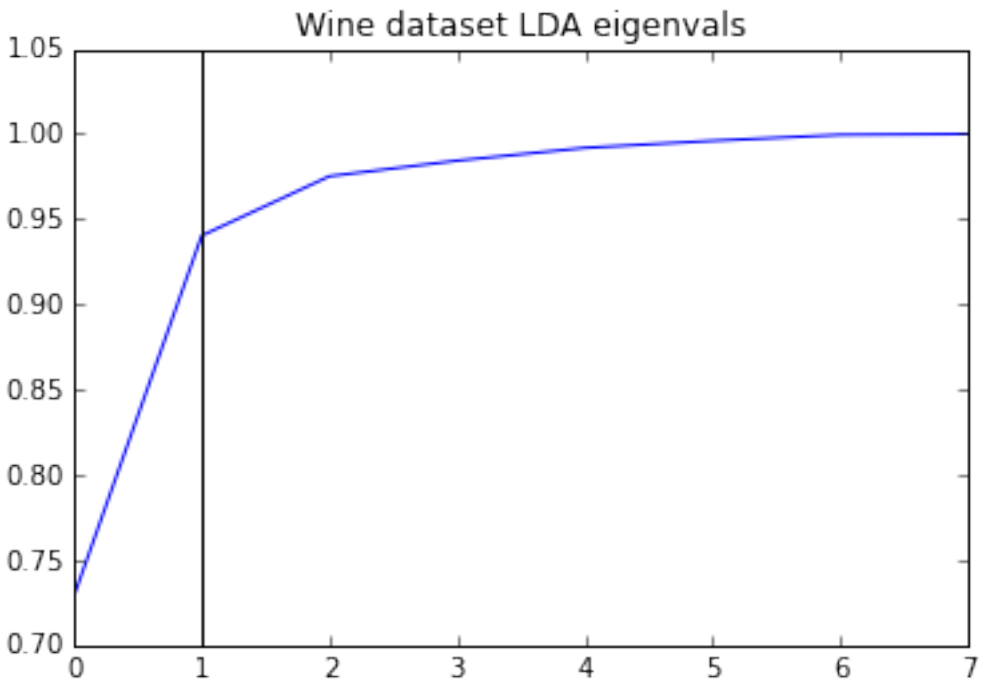


1.2 Plotting cumulative sum of eigenvalues for PCA/LDA

- Here we plot the cumulative sum of eigenvalues divided by the total sum. This plot gives us a measure of variance (information) retained if we used the set of corresponding eigenvectors for transforming the data.
- For LDA it doesn't really make sense to choose multiple eigenvectors for the binary classification case since the rank of the scatter matrices turns out to be at most C-1. To maximize the Fisher discriminant, we would always end up using the largest eigenvector, which has already been plotted/shown earlier.

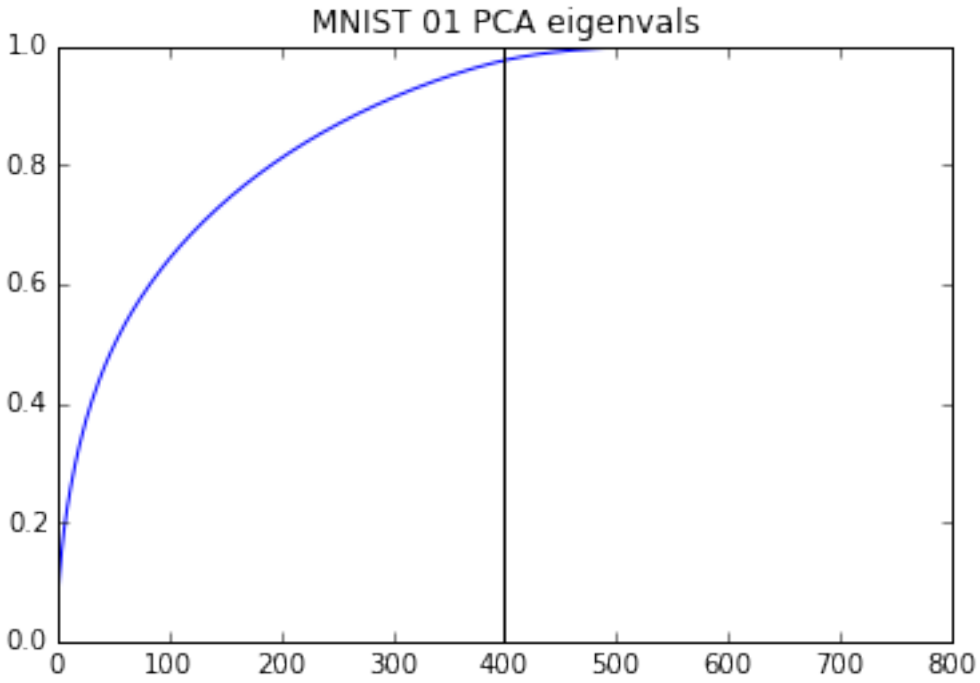
1.2.1 Wine dataset

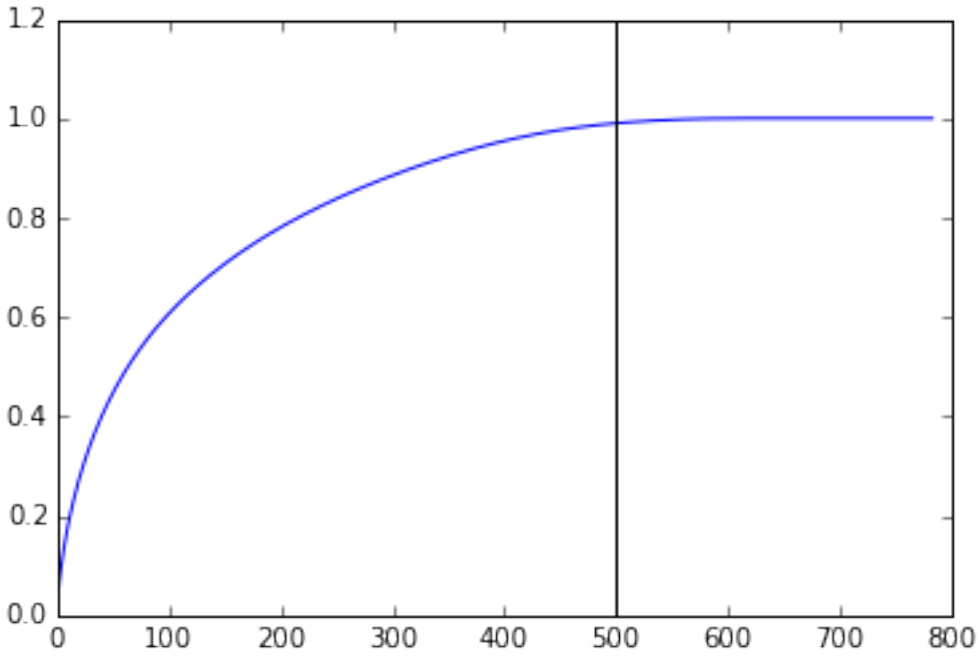



Wine dataset LDA eigenvals
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
1
2
3
4
5
6
7

1.2.2 MNIST dataset

- For the MNIST dataset, we choose just 1 eigenvector in LDA for binary classification case. Thus, I haven't plotted the cumulative sum for LDA's eigenvectors in this case (there is also the pesky issue of complex and negative eigenvalues to deal with which would completely throw off this particular analysis).





1.2.3 Choosing the number of eigenvectors to use for projection

- The above plots give us an idea about the information gained by increasing the number of components. I have taken a conservative approach and choosen the number of components such that the information ratio is > 0.98. Based on the particular use case and looking at memory restrictions, we may increase or decrease this number. One can also look at the reconstruction error as a function of number of components to make this choice (as has been done in the next section). The final numbers I chose are :
- Wine : 2
- MNIST 01 : 400
- MNIST 35 : 500
- Note that we require more components in case of MNIST 35. This is because the data is not as well-separated as 0-1 case (which can also be seen from the visualization of the eigenvectors done in the first section).
- Note that this choice of number of eigenvectors is with respect to minimizing the reconstruction error. It is not necessarily true that using more components would give better classification accuracy Reference. For the classification case, choosing number of components should be done using cross-validation.

1.3 Reconstruction of data

- The reconstruction error metric used is L2 penalty (ie sum of L2 distance between actual and projected data over all samples)
- For LDA, I do not visualize the eigenvectors since they have undergone a PCA transformation beforehand and thus visualizing their reconstruction them would not provide anything resembling the actual data.
- Also note that for LDA, we are only looking for #components in the range 1-10. This is because LDA is used to project the data using C-1 eigenvectors (where C is the number of classes). Thus, we explore what the reconstrution error for LDA would be in a typical setting, unlike PCA where a large number of components can potentially be chosen.

1.3.1 Wine data

PCA

```
Wine reconstruction Error with PCA:
 Num components :  1 Error :  253.7849289
 Num components :  2 Error :  93.3624276457
 Num components :  3 Error :  57.6618884473
 Num components :  4 Error :  44.8448371426
 Num components :  5 Error :  35.4960459854
 Num components :  6 Error :  26.3298469758
 Num components :  7 Error :  20.1990503026
 Num components :  8 Error :  14.7600456611
 Num components :  9 Error :  9.70943946065
 Num components :  10 Error :  6.99687495045
 Num components :  11 Error :  3.33914221828
 Num components :  12 Error :  1.23400616751
 Num components :  13 Error :  8.1846306092e-12
```

LDA

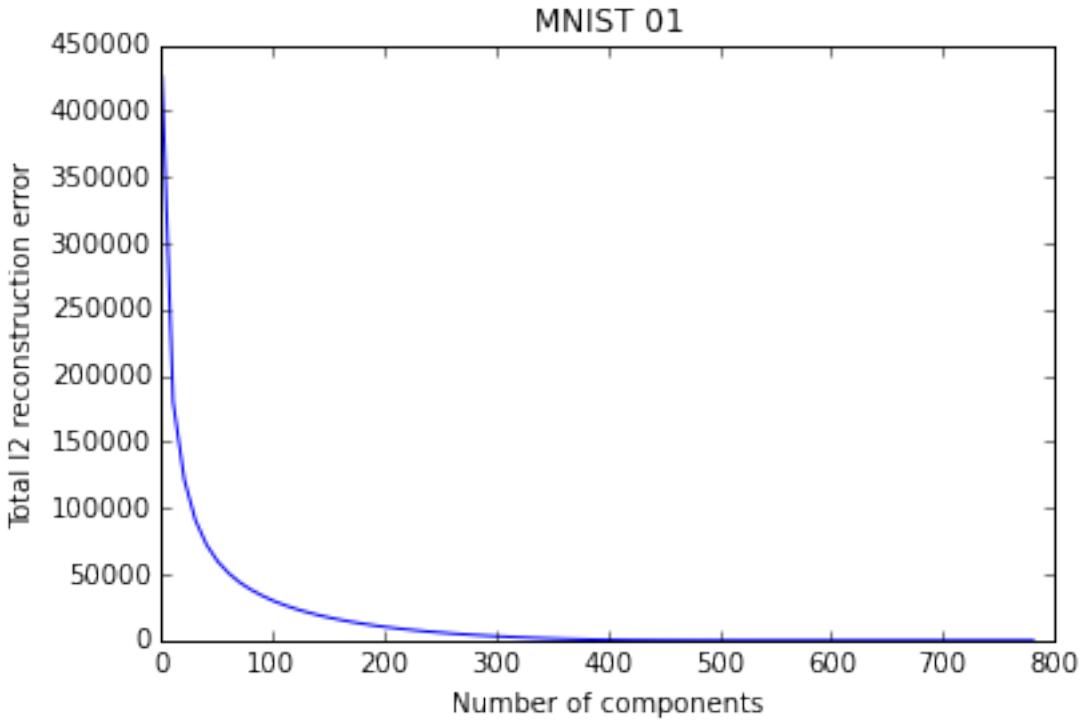
```
Wine reconstruction error with LDA:
 Num components :  1 Error :  123.099633106
```

```
Num components :  2 Error :  117.622210382
Num components :  3 Error :  115.21519895
Num components :  4 Error :  109.023179974
Num components :  5 Error :  99.4662365371
Num components :  6 Error :  92.8304548634
Num components :  7 Error :  80.888154249
Num components :  8 Error :  70.7776421766
Num components :  9 Error :  60.7989032208
```

1.3.2 MNIST Data

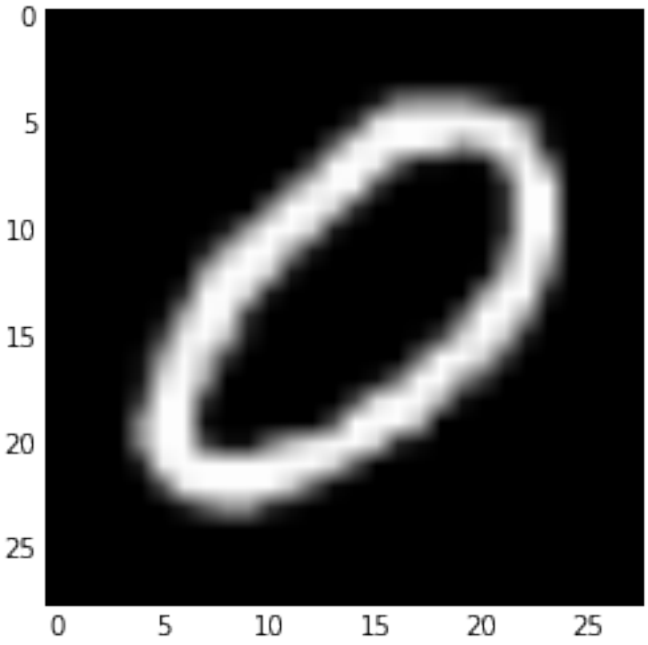
PCA

```
Num components :  200 Error :  2233.98074858
Num components :  250 Error :  1696.23935503
Num components :  300 Error :  1192.71386573
Num components :  350 Error :  726.916658745
Num components :  400 Error :  325.007732688
Num components :  450 Error :  123.801653138
Num components :  500 Error :  40.5839340224
Num components :  550 Error :  6.06403272373
Num components :  600 Error :  4.1739389276e-10
Num components :  650 Error :  4.17468308365e-10
```

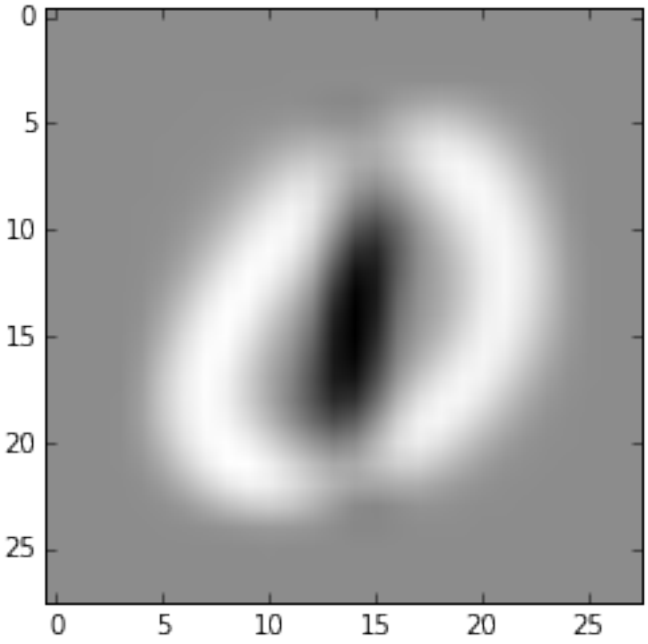


Visualizing sample reconstruction for different basis set sizes

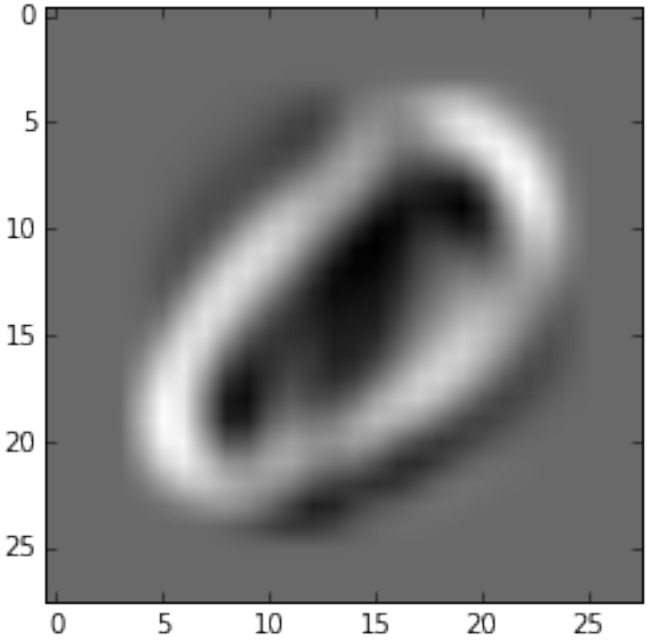
`Original`



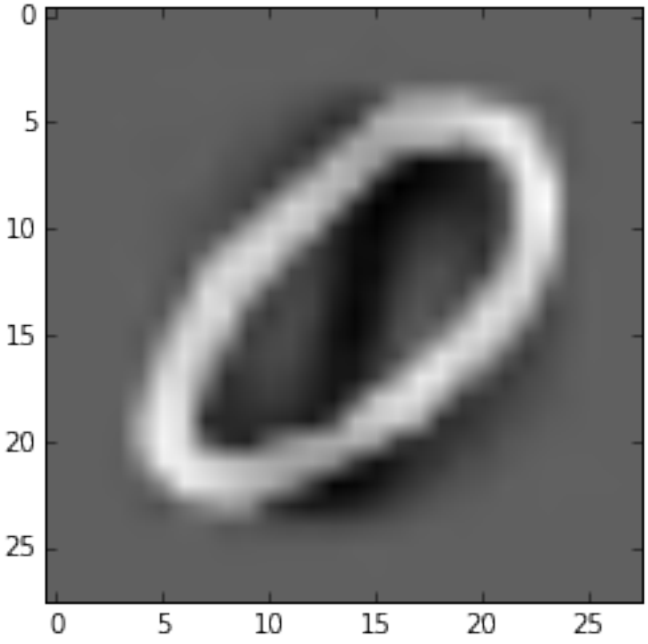
`Num components  2`



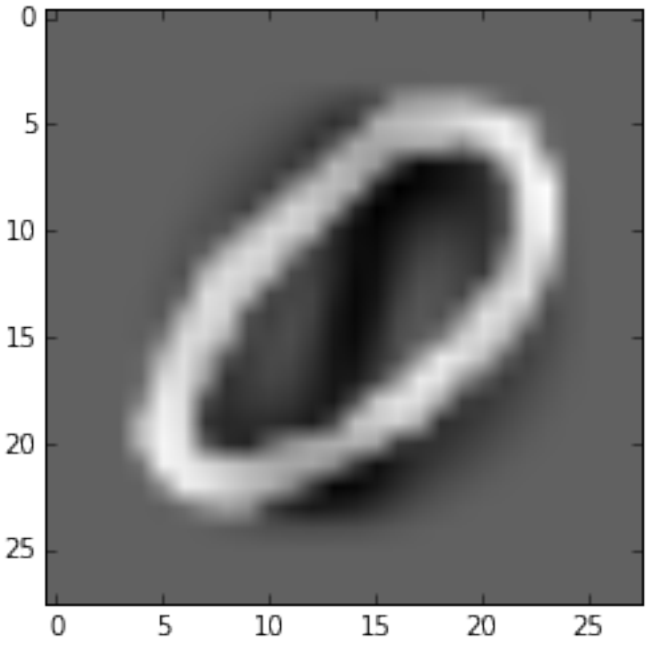
```
Num components  20
```




```
Num components  300
```




```
Num components  500
```



From the figures above, we can see that the reconstructed image becomes closer and closer to the actual image as the number of components is increased

```
MNIST 35
 Num components :  100 Error :  4749.91503189
 Num components :  150 Error :  3675.55470072
 Num components :  200 Error :  2926.22862092
 Num components :  250 Error :  2326.10026359
 Num components :  300 Error :  1778.92446935
 Num components :  350 Error :  1218.85688
 Num components :  400 Error :  719.739713755
 Num components :  450 Error :  363.683544754
 Num components :  500 Error :  151.36234246
 Num components :  550 Error :  45.7028887677
 Num components :  600 Error :  3.19721787119
 Num components :  650 Error :  4.51173404278e-10
```

- As was predicted from the cumulative sum plot of eigenvectors, around 400 eigenvectors are sufficient in 0/1 case and around 500 are sufficient for 3/5 case to get low reconsruction errors.

LDA

```
MNIST 01
 Num components :  1 Error :  9258.15741857
 Num components :  2 Error :  9242.54773475
 Num components :  3 Error :  9239.18361073
 Num components :  4 Error :  9233.41970484
 Num components :  5 Error :  9224.51890997
 Num components :  6 Error :  9216.93713907
 Num components :  7 Error :  9209.94729334
 Num components :  8 Error :  9198.74633095
 Num components :  9 Error :  9188.85521397
```

```
MNIST 35
 Num components :  1 Error :  10111.7830472
 Num components :  2 Error :  10119.2560972
 Num components :  3 Error :  10121.9021136
 Num components :  4 Error :  10111.4377677
 Num components :  5 Error :  10120.2111267
 Num components :  6 Error :  10118.9066285
 Num components :  7 Error :  10118.7340455
 Num components :  8 Error :  10113.965258
 Num components :  9 Error :  10112.7566427
```

- We can see that the reconstructoin error is large for LDA.

1.4 Training and Testing

Wine dataset

1.4.1 Confusion matrices

```
>>>>>>>>>>>>>>>>>>>>>>>> WINE DATA <<<<<<<<<<<<<<<<<<<<<<<<

------------------------ Without whitening ------------------------

>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<
******Test set size 5******
Classifier :  lda
Predicted  1.0  2.0  __all__
Actual
1.0         53    5       58
2.0          4   63       67
__all__     57   68      125

Accuracy :  0.928

Classifier :  nb
Predicted  1.0  2.0  __all__
Actual
1.0          0   58       58
2.0          0   67       67
__all__      0  125      125

Accuracy :  0.536


>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<
******Test set size 50******
Classifier :  lda
Predicted  1.0  2.0  __all__
Actual
1.0         36    3       39
2.0          4   44       48
__all__     40   47       87

Accuracy :  0.919540229885

Classifier :  nb
Predicted  1.0  2.0  __all__
Actual
1.0         39    0       39
2.0          1   47       48
__all__     40   47       87

Accuracy :  0.988505747126


********************************************************************************
```

```
------------------------ With whitening ------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
******Test set size 5******
Classifier :  lda
Predicted  1.0  2.0  __all__
Actual
1.0         52    6       58
2.0         24   43       67
__all__     76   49      125

Accuracy :  0.76

Classifier :  nb
Predicted  1.0  2.0  __all__
Actual
1.0          0   58       58
2.0          0   67       67
__all__      0  125      125

Accuracy :  0.536


>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
******Test set size 50******
Classifier :  lda
Predicted  1.0  2.0  __all__
Actual
1.0         38    1       39
2.0          4   44       48
__all__     42   45       87

Accuracy :  0.942528735632

Classifier :  nb
Predicted  1.0  2.0  __all__
Actual
1.0         39    0       39
2.0          1   47       48
__all__     40   47       87

Accuracy :  0.988505747126


********************************************************************************
```

MNIST

- Note here that the number of principal components chosen is lower as compared to what the plots in section 2 showed. This is because there is no direct relation between lowering reconstruction error and the classification accuracy i.e. feature subset selection is independent of the variance information content and smaller number of components seemed to work better empirically.

```
>>>>>>>>>>>>>>>>>>>>>>>>> MNIST 01 Data <<<<<<<<<<<<<<<<<<<<<<<<<
```

```
------------------------ Without whitening ------------------------
Classifier :  lda
Predicted  0.0   1.0  __all__
Actual
0.0        970    10      980
1.0          0  1135     1135
__all__    970  1145     2115

Accuracy :  0.995271867612
```

```
*******************************************************************************
Classifier :  nb
Predicted  0.0   1.0  __all__
Actual
0.0        979     1      980
1.0         11  1124     1135
__all__    990  1125     2115

Accuracy :  0.994326241135
```

```
*******************************************************************************

------------------------ With whitening ------------------------
Classifier :  lda
Predicted  0.0   1.0  __all__
Actual
0.0        972     8      980
1.0          0  1135     1135
__all__    972  1143     2115

Accuracy :  0.99621749409
```

```
*******************************************************************************
Classifier :  nb
Predicted  0.0   1.0  __all__
Actual
0.0        979     1      980
1.0         11  1124     1135
__all__    990  1125     2115

Accuracy :  0.994326241135
```

```
*******************************************************************************

>>>>>>>>>>>>>>>>>>>>>>>>> MNIST 35 Data <<<<<<<<<<<<<<<<<<<<<<<<<
```

```
------------------------ Without whitening ------------------------
Classifier :  lda
Predicted   3.0  5.0  __all__
Actual
3.0         894  116     1010
5.0         132  760      892
__all__    1026  876     1902

Accuracy :  0.869610935857

********************************************************************************
Classifier :  nb
Predicted  3.0  5.0  __all__
Actual
3.0         914   96     1010
5.0          80  812      892
__all__     994  908     1902

Accuracy :  0.907465825447

********************************************************************************

------------------------ With whitening ------------------------
Classifier :  lda
Predicted  3.0  5.0  __all__
Actual
3.0         912   98     1010
5.0          72  820      892
__all__     984  918     1902

Accuracy :  0.910620399579

********************************************************************************
Classifier :  nb
Predicted  3.0  5.0  __all__
Actual
3.0         914   96     1010
5.0          80  812      892
__all__     994  908     1902

Accuracy :  0.907465825447

********************************************************************************
```

1.5 Accuracy Table

In [ ]:

DataSet	LDA	Naive_Bayes
Wine_test5	92.8%	53.6%
Wine_test50	91.9%	98.8%
Wine_test5 Whitened	76%	53.6%
Wine_test50 Whitened	94.2%	98.8%
MNIST01	99.52%	99.43%
MNIST01 Whitened	99.62%	99.43%
MNIST35	86.96%	90.74%
MNIST35 Whitened	91.06%	90.74%

1.5.1 K-fold Crss validation on wine dataset

In [17]:
```
# k-fold crossvalidation code
# I've used a helper function available in scikit which will generate the indices
# for k-fold CV.
from sklearn.cross_validation import KFold
def run_k_fold_cv(dataX, dataY, n_folds, classifier_type):
    classifier_types = ['lda', 'nb']
    assert classifier_type in classifier_types
    kf = KFold(dataX.shape[0], n_folds=n_folds, shuffle=True)
    accuracies = []
    for train_index, test_index in kf:
        trainX = dataX[train_index]
        trainY = dataY[train_index]
        testX = dataX[test_index]
        testY = dataY[test_index]
        if classifier_type == 'nb':
            nbc = NaiveBayesClassifier()
            nbc.add_data(trainX, trainY)
            predY = nbc.predict(testX)
        elif classifier_type == 'lda':
            lda = LDA(trainX, trainY)
            predY = lda.predict(testX)
        accuracy = np.sum([testY == predY]) / float(testY.shape[0])
        accuracies.append(accuracy)
    return accuracies
```

```
********************************************************************************

Classifier :  lda
Mean accuracy :  0.923076923077

********************************************************************************

Classifier :  nb
Mean accuracy :  0.976923076923
```

1.6 Results (Description and Analysis)

Methodology

- LDA : In order to use LDA to reduce dimentionality, I've used the Fisher discriminant function (and its extension for multiclass case). To use LDA as a classifier, I look at the log-ratio of probabilites and choose the class with the highest probabiliy.
- Naive Bayes : In, Naive Bayes I've inferred the class priors from the training set. This is not a good approach in general since data collection might be skewed, but the impact is not significant for the given datasets. Naive bayes sometimes also makes a simplifying assumption of the variance being same accross classes (ie there is a common source of noise). I have ignored this assumption and used per-class covariance while computing the gaussian probability.

Analysis

- Whitening the data (0 mean and unit variance) has a significant impact on the classification accuracy and it is an important pre-processing step. (Look at the whitened vs non-whitened plot for LDA shown in the first section for an illustration of this impact)
- In case of Naive Bayes, performing PCA on the data beforehand helps improve accuracy. One way this can be explained is that NB assumes independent features, and PCA transforms the data so as to give diagonal covariance matrix, thus satisfying a necessary condition for independence.
- Both the classifiers achieve ˜99% accuracy on the MNIST 01 dataset while it is ˜90% for MNIST 35. It thus seems harder to classify latter dataset. This may be because 3 and 5 are not as well-separable as 0 and 1. This fact is also evident from by plotting the first eigenvector (as done in section 2) which shows a better distinction for the 0/1 case.
- For the case of training wine data with just 5 samples, it is evident that we would get a very poor classifier. The NB classifier in this case always ends up predicting the same class. However, LDA is still able to achieve ˜86% accuracy which is surprising. Maybe it latches on to a particularly distinctive feature and is able to learn that from just the 5 samples that it sees.

Multiclass Classifiers

- For using LDA as a multiclass classifier, the data is first sphered (as suggested in Elements of Statistical Learning) and then the separating hyperplanes are computed for each pair of classes.
- With NB, the same assumptions about feature independence need to be adhered to for multiclass case. The predicted class is givenn by one that gives highest probability conditioned on Y (assuming equal priors).
- In terms of code, the same code should work for multiclass classifier. There are no assumptions made about the number of classes while coding the classifiers. In the multiclass case, it becomes even more important to whiten the data before running it through a classifier.

1.7 Extra Credit

In [ ]:
```
# Since the code does not worry about the number of classes, I tried to run my LDA
# and NB implementations on the entire MNIST dataset and obtained the following
# results :
```

```
Classifier :  lda
Predicted  0.0   1.0  3.0  5.0  __all__
Actual
0.0        901     0    7   72      980
1.0          0  1102    4   29     1135
3.0          9    30  875   96     1010
5.0         21    63  109  699      892
__all__    931  1195  995  896     4017

Accuracy :  0.890465521533

*****************************************************************************************

Classifier :  nb
Predicted  0.0   1.0  3.0  5.0  __all__
Actual
0.0        913     0    8   59      980
1.0          0  1095   24   16     1135
3.0          9    13  860  128     1010
5.0         30     5   82  775      892
__all__    952  1113  974  978     4017

Accuracy :  0.906895693303

*****************************************************************************************
```