Linear Discriminant Analysis

Course Name: **Data Warehousing and Mining**
Course Code: **CO461**

Submitted By:
Aparna (15CO236), Indukala (15CO230)
Team ID: 43

Introduction:

Linear discriminant analysis (**LDA**) is a generalization of **Fisher's linear discriminant**, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects.

LDA is most commonly used as a dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting and also reduce computational costs.

The general LDA approach is very similar to Principal Component Analysis in that they both look for linear combinations of variables which best explain the data. However, in addition to finding the component axes that maximize the variance of our data (PCA), we are also interested in the axes that maximize the separation between multiple classes (LDA).

Basically, the goal of LDA is to project a feature space (a dataset n-dimensional samples) onto a smaller subspace k (where $k \leq n-1$) while maintaining the class-discriminatory information. Dimensionality reduction does not only help reducing computational costs for a given classification task, but it can also be helpful to avoid overfitting by minimizing the error in parameter estimation.

Principal Component Analysis vs Linear Discriminant Analysis:

Both Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are linear transformation techniques that are commonly used for dimensionality reduction. PCA can be described as an “unsupervised” algorithm, since it “ignores” class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset. In contrast to PCA, LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that that maximize the separation between multiple classes.

Although it might sound intuitive that LDA is superior to PCA for a multi-class classification task where the class labels are known, this might not always the case.

For example, comparisons between classification accuracies for image recognition after using PCA or LDA show that PCA tends to outperform LDA if the number of samples per class is relatively small. In practice, it is also not uncommon to use both LDA and PCA in combination, for example, PCA for dimensionality reduction followed by an LDA.

Background:

The original dichotomous discriminant analysis was developed by Fisher et al (1936). It is different from an ANOVA or MANOVA, which is used to predict one (ANOVA) or multiple (MANOVA) continuous dependent variables by one or more independent categorical variables. Discriminant function analysis is useful in determining whether a set of variables is effective in predicting category membership.

Principal Component Analysis (PCA) is an unsupervised learning method, which treats samples of different classes in the same way. Donato et al (1999) compared various methods of feature extraction for automatically recognizing facial expression, including PCA, LDA, Gabor wavelet, etc. Best performances were obtained using the Gabor wavelet presentation. But the computation and memory requirement of Gabor wavelet is very large and the dimension is very high. For further dimensionality reduction and good recognition performance, MartõÂnez et al (2001) adopted a two-phase framework PCA plus LDA for feature compression and selection in their facial expression recognition system.

Multi-class LDA (MLDA) was introduced by Li et al (2005). MLDA is a generalization of standard two-class LDA that can handle arbitrary number of classes. It provides an elegant way for classification using discriminant features.

When we conduct LDA/PCA learning over datasets in real-world applications, we often confront difficult situations where a complete set of training samples is not given in advance. A straightforward approach is that we can collect data whenever new data is presented and then construct a provisional system by batch learning over the data collected so far. However, it is obvious that such system only works under large memory and high computation expenses, because the system would need to maintain a huge memory to store the data either previously learned, or newly presented.

A solution to the above problem is described by Pang et al (2006) using one-pass incremental learning. In this learning scheme, a system must acquire knowledge with a single presentation of the training data and retain the knowledge acquired in the past without keeping a large number of training samples. We can observe that Incremental LDA (ILDA) updates the discriminant eigenspace gradually as the progress of incremental learning grows, and it can finally construct exactly the same discriminant eigenspace as that of Batch LDA.

A straightforward discriminant criterion function discussed in Tang et al (2006) is proved to be equal to Fisher's criterion function and analyzed from the viewpoint of the Laplacian of a graph.

Moreover, the LDA, which is termed Laplacian Linear Discriminant Analysis (LLDA), is linked to spectral decomposition of the Laplacian of a graph. At last, two algorithms are employed to maximize the criterion function.

While supervised feature selection has been explored extensively, there are only a few unsupervised methods that can be applied to exploratory data analysis. Niijima et al. (2006) addresses the problem of unsupervised feature selection. An algorithm is proposed which extends laplacian LDA to unsupervised case, and is particularly efficient when the feature size is much larger than the sample size. It performs even better than Fisher score for some of the data sets, despite the fact that LLDA-RFE is fully unsupervised.

However, it becomes a serious challenge to use LDA method in the microarray analysis settings. There are two major concerns here. First, the sample covariance matrix estimate is singular and cannot be inverted. Although we may use the generalized inverse instead, the estimate will be very unstable due to lack of observations. Second, high dimensionality makes direct matrix operation formidable, hence hindering the applicability of this method. Therefore, we will make some changes to the original LDA to overcome these problems. Guo et al (2007) introduced a slightly biased covariance estimate through which not only the singularity problem gets resolved but the sample covariance estimate also gets stabilized.

A challenging issue in applying PCA, LDA and RLDA is the estimation of model parameters, i.e., the dimension for intermediate PCA-subspace in PCA, LDA and the regularization parameter in RLDA. Using Generalized LDA (GLDA) proposed by Shuiwang et al (2008), the matrix computations involved in LDA-based algorithms can be simplified so that the cross-validation procedure for model selection can be performed efficiently.

The purpose of using SWLDA described by Siddiqi et al (2008) as a feature extraction technique is to extract the localized features from faces that the previous feature extraction techniques were limited in analyzing. SWLDA is easy to explain, has good predictive ability, and computationally, it is less expensive than other existing methods. Some limitations of the existing works, such as illumination change, do not affect the performance of the SWLDA. SWLDA only extracts a small set of features by employing forward and backward regression models.

A novel multitask learning framework for multiview action recognition allows for the sharing of discriminative SSM features among different views. Inspired by the mathematical connection between multivariate linear regression and LDA, Yan et al (2014) proposed a multitask multiclass LDA as a single optimization problem by choosing an appropriate class indicator matrix. The proposed multi-task LDA framework is novel, and is modeled as a single

optimization problem through the use of a class indicator matrix. The proposed approach is shown to be highly effective and achieves improved action recognition performance with respect to other classification methods based on SSM descriptors.

Linear discriminant analysis (LDA) is powerful for supervised data dimensionality reduction and has been applied successfully to many applications, including machine learning, data mining, and bioinformatics, which require to deal with high-dimensional data efficiently. However, in many real-world applications, such as customer shopping transaction analysis, mining web logs, and mining DNA sequences, data is incrementally received from various data sources presented as a data stream, and so data grows incrementally. To overcome this, a new incremental LDA algorithm called ILDA/QR is proposed by Chu et al (2015). The main features of ILDA/QR are easy handleability, efficient computational complexity, efficient space complexity.

He et al (2018) proposed a local pairwise linear discriminant analysis (LPLDA) algorithm. The aim of LPLDA is to maximize the covariance between target class samples and the associated confusable samples. This algorithm can be used as an effective backend in the field of speaker verification. For a specific target speaker, neighbouring non-target utterances (local confusable vectors) are much more important for learning a decision boundary, so emphasizing these instead of treating all examples equally should be beneficial.

Algorithm:

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors ($\mathbf{e}_1,\mathbf{e}_2,...\mathbf{e}_d$) and corresponding eigenvalues ($\lambda_1,\lambda_2,...\lambda_d$) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a $d \times k$ dimensional matrix W (where every column represents an eigenvector).
5. Use this $d \times k$ eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: $Y = X \times W$ (where X is a $n \times$

d-dimensional matrix representing the ***n*** samples, and ***Y*** are the transformed ***n*** × ***k***-dimensional samples in the new subspace).

Algorithm Explanation:

How do we select ***k*** if our goal is to reduce the dimensions of a ***d***-dimensional dataset by projecting it onto a ***k***-dimensional subspace (where ***k<d***)?

The k-dimensional feature space should represent the data well.

We will compute eigenvectors (the components) from our data set and collect them in a scatter-matrices.

Each of these eigenvectors is associated with an eigenvalue, which tells us about the "magnitude" of the eigenvectors.

If we would observe that all eigenvalues have a similar magnitude, then this may be a good indicator that our data is already projected on a "good" feature space.

And in the other scenario, if some of the eigenvalues are much much larger than others, we might be interested in keeping only those eigenvectors with the highest eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and we might consider dropping those for constructing the new feature subspace.

Example of LDA:

Consider the famous "Iris" dataset that has been deposited on the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/Iris).

The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

After preparing the data for LDA, we do the following:

1) Compute d-dimensional mean vector of the three different flower classes. Here, d=4.

   Mean Vector class 1: [5.006 3.418 1.464 0.244]

   Mean Vector class 2: [5.936 2.77   4.26   1.326]

   Mean Vector class 3: [6.588 2.974 5.552 2.026]

2) Compute within-class and the between-class scatter matrix.

   The within class matrix S_w is computed by the following equation:

$$S_w = \sum_{i=1}^{c} S_i$$

   where

$$S_i = \sum_{x \in D_i}^{n} (x - m_i)(x - m_i)^T$$ (scatter matrix for every class)

   and m_i is the mean vector

$$m_i = \frac{1}{n_i} \sum_{x \in D_i}^{n} x_k$$

Within-class Scatter Matrix:

[[38.95 13.683 24.614 5.6556]

[13.683 17.035 8.12 4.9132]

[24.614 8.12 27.22 6.2536]

[5.655 4.9132 6.2536 6.1756]]

The between-class scatter matrix S_b is computed by the following equation:

$$S_b = \sum_{i=1}^{c} N_i \, (m_i - m)(m_i - m)^T$$

where m is the overall mean and m_i and N_i are the sample mean and sizes of the respective classes.

Between-class Scatter Matrix:

[[63.2121 -19.534 165.1647 71.3631]

[-19.534 10.9776 -56.0552 -22.4924]

[165.1647 -56.0552 436.6437 186.9081]

[71.3631 -22.4924 186.9081 80.6041]]

3) Solving the generalized eigenvalue problem for the matrix $S^{-1}{}_W S_B$ to obtain linear discriminants and sorting eigenvectors in decreasing order of eigenvalues.

32.2719577997

0.27756686384

5.71450476746e-15

5.71450476746e-15

Calculating “explained variance” as percentage:

eigenvalue 1: 99.15%
eigenvalue 2: 0.85%
eigenvalue 3: 0.00%
eigenvalue 4: 0.00%

The first eigenpair is by far the most informative one, and we won’t lose much information if we would form a 1D-feature spaced based on this eigenpair.

Choose 2 most informative eigenpairs to get 2-dimensional feature subspace.

Compute matrix W and use it to transform samples onto new subspace via equation Y = X * W

Matrix W:

[[-0.2049 -0.009]
 [-0.3871 -0.589]
 [0.5465 0.2543]
 [0.7138 -0.767]]

Implementation using Tool:

We have used Python to implement LDA. The first program involves implementation of our own algorithm. We compare the results of our algorithm and the in-built function present in Python - *sklearn.discriminantanalysis.LinearDiscriminantAnalysis.* The following is the code used:

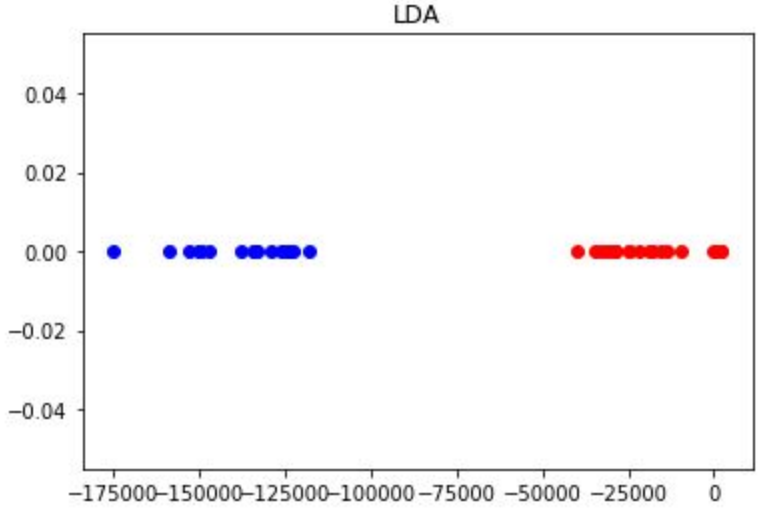
Program 1's code:

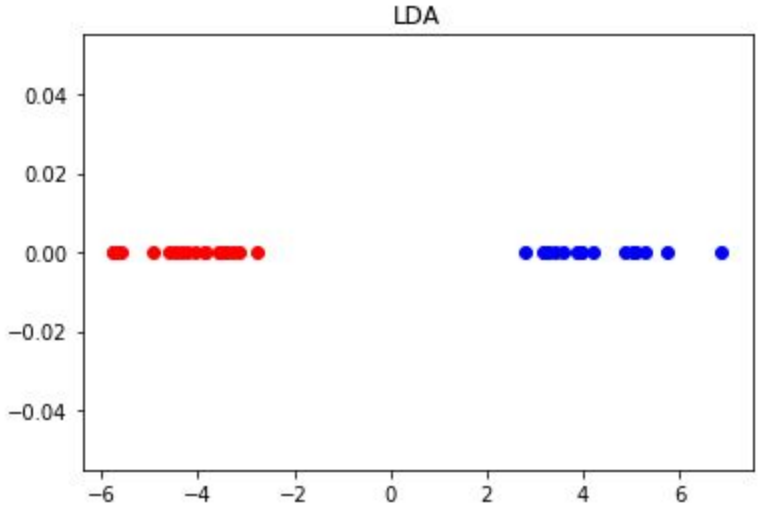
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy import linalg as lg
get_ipython().magic(u'matplotlib inline')


df                                                                                =
pd.read_csv('C:/Users/induapps/Desktop/SCLC_study_output_filtered_2.csv',header
=None)

df1 = df.drop(df.index[0])

df2 = df1.drop(df.columns[0], axis=1)

df3 = df2

df3_1 = df2.values[0:20,:]
df3_2 = df2.values[20:, : ]

m_1 = df3_1.mean(axis = 0)
m_2 = df3_2.mean(axis = 0)
mean_all = df2.mean(axis = 0)

mean_1 = m_1.reshape(1,19)
mean_1 = np.repeat(mean_1,20,axis = 0)

mean_2 = m_2.reshape(1,19)
mean_2 = np.repeat(mean_2,20,axis = 0)

within_class_scatter = np.zeros((19,19))
wcs_1 = np.zeros((19,19))
wcs_1 = np.matmul((np.transpose(df3_1 - mean_1 )), (df3_1 - mean_1))

wcs_2 = np.zeros((19,19))
wcs_2 = np.matmul((np.transpose(df3_2 - mean_2 )), (df3_2 - mean_2))

within_class_scatter = np.add(wcs_1,wcs_2)

bcs_1 = np.multiply(len(df3_1),np.outer((m_1 - mean_all),(m_1 - mean_all)))
bcs_2 = np.multiply(len(df3_2),np.outer((m_2 - mean_all),(m_2 - mean_all)))

between_class_scatter = np.add(bcs_1,bcs_2)
```

```
e_val,                                  e_vector                                  =
np.linalg.eig(np.dot(lg.inv(within_class_scatter),between_class_scatter))
for e in range (len(e_val)):
    e_scatter = e_vector[:,e].reshape(19,1)

    print(e_val[e].real)

print(between_class_scatter)


eig_pairs    =    [(np.abs(e_val[i]).real,    e_vector[:,i].real)    for    i    in
range(len(e_val))]


eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)



print('Eigenvalues in decreasing order:\n')
for i in eig_pairs:
    print(i[0])

W= eig_pairs[0][1].reshape(19,1)

W

lda_project = np.dot(df2,W)

lda_project


# In[177]:

#plot
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_title('LDA')
ax.plot(lda_project[0:20],    np.zeros(20),    linestyle='None',    marker='o',
color='blue', label='NSCLC')
ax.plot(lda_project[20:40],    np.zeros(20),    linestyle='None',    marker='o',
color='red', label='SCLC')
fig.show()


# In[185]:

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
y1_ = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]

# LDA

sklearn_lda = LDA(n_components=1)
X_lda_sklearn = sklearn_lda.fit_transform(df2, y1_)

X_lda_sklearn= -X_lda_sklearn

print(X_lda_sklearn)

#plot
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.set_title('LDA')
ax.plot(X_lda_sklearn[0:20],    np.zeros(20),    linestyle='None',    marker='o',
color='blue', label='NSCLC')
ax.plot(X_lda_sklearn[20:40],    np.zeros(20),    linestyle='None',    marker='o',
color='red', label='SCLC')
fig.show()
```

Results of Program 1: The output of our own LDA algorithm and in-built LDA function of Python respectively:


LDA
0.04
0.02
0.00
−0.02
−0.04
−175000
−150000
−125000
−100000
−75000
−50000
−25000
0


LDA
0.04
0.02
0.00
−0.02
−0.04
−6
−4
−2
0
2
4
6

```
Eigenvalues in decreasing order:

18.478039912606995
1.0648453936830176e-13
4.0053245559703925e-14
3.294426122855085e-14
3.294426122855085e-14
3.225430232112015e-14
3.225430232112015e-14
4.506902250240742e-15
4.506902250240742e-15
4.371847763764651e-15
4.371847763764651e-15
3.59117216127921e-15
3.552713678800501e-15
1.3186276910140236e-15
1.3186276910140236e-15
1.1646113806290988e-15
1.1266260194432053e-15
6.65955980248242e-16
4.725394553934102e-17
[[ 3.25296264]
 [ 6.87395904]
 [ 4.22018965]
 [ 5.07509696]
 [ 3.93463394]
 [ 5.72516785]
 [ 3.8613171 ]
 [ 3.95272386]
 [ 5.28479124]
 [ 3.26522652]
 [ 5.08498545]
 [ 3.57189064]
 [ 5.05511495]
 [ 3.15378176]
 [ 4.86686796]
 [ 2.81120157]
 [ 3.93543558]
 [ 3.39771836]
 [ 3.25628819]
 [ 3.21596383]
 [-4.49065733]
 [-4.6115194 ]
 [-3.45215203]
 [-2.75643608]
 [-3.83408221]
 [-3.54607243]
 [-3.25235288]
 [-3.40306303]
 [-4.04197759]
```

The above implementation of linear discriminant analysis performs dimensionality reduction on the given dataset : Small cell lung cancer. The above graph shows the results of our LDA algorithm. It is similar to that obtained using in-built function provided by Python.

The **second program** implemented by us compares the performance of LDA + SVM and PCA + SVM :

Dataset	Data Type	Attribute Characteristics	Number of features
Arcene	non-sparse	Real	10000
Madelon	non-sparse	Real	500

Program 2's code and results are as follows:

Kernel PCA with SVM classification using RBF Kernel:

```
import pandas as pd
import numpy as np

from warnings import filterwarnings
from sklearn import svm
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn.metrics import accuracy_score

# Disable warnings from being printed
filterwarnings('ignore')
```

```
# Get the train and validation data for Arcene dataset
train = pd.read_csv("arcene_train.data.txt", header=None, sep=" ",
usecols=range(10000))
train_labels = pd.read_csv("arcene_train.labels.txt", header=None)
valid = pd.read_csv("arcene_valid.data.txt", header=None, sep=" ",
usecols=range(10000))
valid_labels = pd.read_csv("arcene_valid.labels.txt", header=None)

# Get the train and validation data for Madelon dataset
#train = pd.read_csv("madelon_train.data.txt", header=None, sep=" ",
usecols=range(500))
#train_labels = pd.read_csv("madelon_train.labels.txt", header=None)
#valid = pd.read_csv("madelon_valid.data.txt", header=None, sep=" ",
usecols=range(500))
#valid_labels = pd.read_csv("madelon_valid.labels.txt", header=None)
```

```
def KPCA(X, k, gamma):
    # Calculating the squared Euclidean distances for every pair of points
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Converting the pairwise distances into a symmetric MxM matrix.
    mat_sq_dists = squareform(sq_dists)

    # Computing the MxM RBF kernel matrix.

    K = exp(-gamma * mat_sq_dists)

    # Normalizing the symmetric NxN kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K_norm = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

    # Obtaining eigenvalues in ascending order with corresponding
    # eigenvectors from the symmetric matrix.
    eigvals, eigvecs = eigh(K_norm)

    # Obtaining i eigenvectors (alphas) that corresponds to i highest eigenvalues
```

```
    (lambdas)
    alphas = np.column_stack((eigvecs[:,-i] for i in range(1,k+1)))
    lambdas = [eigvals[-i] for i in range(1,k+1)]

    return alphas, lambdas
```

```
def project(valid, X, k, gamma, alphas, lambdas):
    projected_data = np.zeros((valid.shape[0], k))
    X_arr = np.array(train)
    valid_arr = np.array(valid)
    for i in range(valid_arr.shape[0]):
        cur_dist = np.array([np.sum( (valid_arr[i]-x) ** 2) for x in X_arr])
        cur_k = np.exp(-gamma * cur_dist)
        projected_data[i, :] = cur_k.dot(alphas / lambdas)
    return projected_data
```

```
gamma = 1e-10
ks = [10, 100]

for k in ks:
    alphas, lambdas = KPCA(train, k, gamma)
    projected_valid = project(valid, train, k, gamma, alphas, lambdas)
    projected_train = project(train, train, k, gamma, alphas, lambdas)
    clf = svm.SVC(kernel="rbf", max_iter=1000000)
    clf.fit(projected_train, train_labels)
    results = clf.predict(projected_valid)
    print("For k=", k, ",", "Accuracy=", accuracy_score(valid_labels, results))
```

Results on Arcene dataset:

K	Accuracy
10	0.56

100	0.56

Results on Madelon dataset:

K	Accuracy
10	0.551666666667
100	0.593333333333

Kernel PCA with SVM classification using Linear Kernel:

```
import pandas as pd
import numpy as np

from warnings import filterwarnings
from sklearn import svm
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn.metrics import accuracy_score

# Disable warnings from being printed
filterwarnings('ignore')
```

```
# Get the train and validation data for Arcene dataset
# train = pd.read_csv("arcene_train.data.txt", header=None, sep=" ",
usecols=range(10000))
# train_labels = pd.read_csv("arcene_train.labels.txt", header=None)
# valid = pd.read_csv("arcene_valid.data.txt", header=None, sep=" ",
usecols=range(10000))
# valid_labels = pd.read_csv("arcene_valid.labels.txt", header=None)

# Get the train and validation data for Madelon dataset
train = pd.read_csv("madelon_train.data.txt", header=None, sep=" ",
usecols=range(500))
train_labels = pd.read_csv("madelon_train.labels.txt", header=None)
valid = pd.read_csv("madelon_valid.data.txt", header=None, sep=" ",
usecols=range(500))
valid_labels = pd.read_csv("madelon_valid.labels.txt", header=None)
```

```
def KPCA(X, k, gamma):
    # Calculating the squared Euclidean distances for every pair of points
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Converting the pairwise distances into a symmetric MxM matrix.
    mat_sq_dists = squareform(sq_dists)

    # For linear kernel
    K = X.dot(X.T)

    # Normalizing the symmetric NxN kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K_norm = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

    # Obtaining eigenvalues in ascending order with corresponding
    # eigenvectors from the symmetric matrix.
    eigvals, eigvecs = eigh(K_norm)

    # Obtaining i eigenvectors (alphas) that corresponds to i highest eigenvalues
(lambdas).
```

```
    alphas = np.column_stack((eigvecs[:,-i] for i in range(1,k+1)))
    lambdas = [eigvals[-i] for i in range(1,k+1)]

    return alphas, lambdas
```

```
def project(valid, X, k, gamma, alphas, lambdas):
    projected_data = np.zeros((valid.shape[0], k))
    X_arr = np.array(train)
    valid_arr = np.array(valid)
    for i in range(valid_arr.shape[0]):
        cur_k = np.array([np.sum( (valid_arr[i]-x) ** 2) for x in X_arr])
        projected_data[i, :] = cur_k.dot(alphas / lambdas)
    return projected_data
```

```
gamma = 1e-10
ks = [10, 100]

for k in ks:
    alphas, lambdas = KPCA(train, k, gamma)
    projected_valid = project(valid, train, k, gamma, alphas, lambdas)
    projected_train = project(train, train, k, gamma, alphas, lambdas)
    clf = svm.SVC(kernel="linear", max_iter=1000000)
    clf.fit(projected_train, train_labels)
    results = clf.predict(projected_valid)
    print("For k=", k, ",", "Accuracy=", accuracy_score(valid_labels, results))
```

Results on Arcene dataset:

K	Accuracy
10	0.64

100	0.53

Results on Madelon dataset:

K	Accuracy
10	0.581666666667
100	0.595

Kernel LDA with SVM classification using RBF Kernel:

```
import pandas as pd
import numpy as np

from warnings import filterwarnings
from sklearn import svm
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn.metrics import accuracy_score

# Disable warnings from being printed
filterwarnings('ignore')
```

```
# Get the train and validation data
#train = pd.read_csv("arcene_train.data.txt", header=None, sep=" ",
usecols=range(10000))
#train_labels = pd.read_csv("arcene_train.labels.txt", header=None)
#valid = pd.read_csv("arcene_valid.data.txt", header=None, sep=" ",
usecols=range(10000))
#valid_labels = pd.read_csv("arcene_valid.labels.txt", header=None)

train = pd.read_csv("madelon_train.data.txt", header=None, sep=" ",
usecols=range(500))
train_labels = pd.read_csv("madelon_train.labels.txt", header=None)
valid = pd.read_csv("madelon_valid.data.txt", header=None, sep=" ",
usecols=range(500))
valid_labels = pd.read_csv("madelon_valid.labels.txt", header=None)
```

```
def KLDA(X, X_labels, gamma, lmb):
    # Calculating the squared Euclidean distances for every pair of points
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Converting the pairwise distances into a symmetric MxM matrix.
    mat_sq_dists = squareform(sq_dists)

    # Computing the MxM RBF kernel matrix.

    # For RBF kernel
    K = exp(-gamma * mat_sq_dists)

    Karr = np.array(K, dtype=np.float)
    yarr = np.array(X_labels, dtype=np.int)

    labels = np.unique(yarr)
    n = yarr.shape[0]

    idx1 = np.where(yarr==labels[0])[0]
    idx2 = np.where(yarr==labels[1])[0]
    n1 = idx1.shape[0]
    n2 = idx2.shape[0]

    K1, K2 = Karr[:, idx1], Karr[:, idx2]

    N1 = np.dot(np.dot(K1, np.eye(n1) - (1 / float(n1))), K1.T)
    N2 = np.dot(np.dot(K2, np.eye(n2) - (1 / float(n2))), K2.T)
    N = N1 + N2 + np.diag(np.repeat(lmb, n))

    M1 = np.sum(K1, axis=1) / float(n1)
    M2 = np.sum(K2, axis=1) / float(n2)
    M = M1 - M2

    coeff = np.linalg.solve(N, M).reshape(-1, 1)

    return coeff
```

```
def project(data, X, coeff, gamma):
    projected_data = np.zeros((data.shape[0], 1))
    X_arr = np.array(X)
    data_arr = np.array(data)
    for i in range(data_arr.shape[0]):
        cur_dist = np.array([np.sum((data_arr[i]-x)**2) for x in X_arr])
        cur_k = np.exp(-gamma * cur_dist)
        projected_data[i, :] = cur_k.dot(coeff)
    return projected_data
```

```
lmb = 1e-3
gamma = 1e-10
coeff = KLDA(train, train_labels, gamma, lmb)
projected_valid = project(valid, train, coeff, gamma)
projected_train = project(train, train, coeff, gamma)
clf = svm.SVC(kernel="rbf", max_iter=1000000)
clf.fit(projected_train, train_labels)
results = clf.predict(projected_valid)
print(accuracy_score(valid_labels, results))
```

Results on Arcene dataset:

Accuracy	0.56

Results on Madelon dataset:

Accuracy	0.508333333333

Kernel LDA with SVM classification using Linear Kernel:

```
import pandas as pd
import numpy as np

from warnings import filterwarnings
from sklearn import svm
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from sklearn.metrics import accuracy_score

# Disable warnings from being printed
filterwarnings('ignore')
```

```
# Get the train and validation data
#train = pd.read_csv("arcene_train.data.txt", header=None, sep=" ", usecols=range(10000))
#train_labels = pd.read_csv("arcene_train.labels.txt", header=None)
#valid = pd.read_csv("arcene_valid.data.txt", header=None, sep=" ", usecols=range(10000))
#valid_labels = pd.read_csv("arcene_valid.labels.txt", header=None)

train = pd.read_csv("madelon_train.data.txt", header=None, sep=" ", usecols=range(500))
train_labels = pd.read_csv("madelon_train.labels.txt", header=None)
valid = pd.read_csv("madelon_valid.data.txt", header=None, sep=" ", usecols=range(500))
valid_labels = pd.read_csv("madelon_valid.labels.txt", header=None)
```

```
def KLDA(X, X_labels, lmb):
    # Calculating the squared Euclidean distances for every pair of points
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Converting the pairwise distances into a symmetric MxM matrix.
    mat_sq_dists = squareform(sq_dists)

    # For linear kernel
    K = X.dot(X.T)

    Karr = np.array(K, dtype=np.float)
    yarr = np.array(X_labels, dtype=np.int)

    labels = np.unique(yarr)
    n = yarr.shape[0]

    idx1 = np.where(yarr==labels[0])[0]
    idx2 = np.where(yarr==labels[1])[0]
    n1 = idx1.shape[0]
    n2 = idx2.shape[0]

    K1, K2 = Karr[:, idx1], Karr[:, idx2]

    N1 = np.dot(np.dot(K1, np.eye(n1) - (1 / float(n1))), K1.T)
    N2 = np.dot(np.dot(K2, np.eye(n2) - (1 / float(n2))), K2.T)
    N = N1 + N2 + np.diag(np.repeat(lmb, n))

    M1 = np.sum(K1, axis=1) / float(n1)
    M2 = np.sum(K2, axis=1) / float(n2)
    M = M1 - M2

    coeff = np.linalg.solve(N, M).reshape(-1, 1)

    return coeff
```

```
def project(data, X, coeff):
    projected_data = np.zeros((data.shape[0], 1))
    X_arr = np.array(X)
    data_arr = np.array(data)
    for i in range(data_arr.shape[0]):
        cur_k = np.array([np.sum((data_arr[i]-x)**2) for x in X_arr])
        projected_data[i, :] = cur_k.dot(coeff)
    return projected_data
```

```
lmb = 1e-3
coeff = KLDA(train, train_labels, lmb)
projected_valid = project(valid, train, coeff)
projected_train = project(train, train, coeff)
clf = svm.SVC(kernel="linear", max_iter=1000000)
clf.fit(projected_train, train_labels)
results = clf.predict(projected_valid)
print(accuracy_score(valid_labels, results))
```

Results on Arcene dataset:

Accuracy	0.72

Results on Madelon dataset:

Accuracy	0.496666666667

References:

1. Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems", *Annals of Eugenics*.
2. G. Donato, M. S. Bartlett, J. C. Hager, “Classifying Facial Actions”, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 21, 1999, pp. 974-989.
3. Martinez, A. M.; Kak, A. C. (2001). "PCA versus LDA", *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

4. Shaoning Pang , S. Ozawa and N. Kasabov, “Incremental linear discriminant analysis for classification of data streams”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* ,Volume: 35 , Issue: 5 , Oct. 2005.
5. Li, T., Zhu, S. & Ogihara, “Using discriminant analysis for multi-class classification: an experimental investigation”, *M. Knowl Inf Syst*, 2006, 10: 453.
6. Tang H., Fang T., Shi P.F.,”Laplacian linear discriminant analysis”,*Pattern Recognition*, 39 (1) (2006), pp. 136-139.Yaqian Guo, Trevor Hastie, Robert Tibshirani, “Regularized linear discriminant analysis and its application in microarrays”, *Biostatistics*, Volume 8, Issue 1, 1 January 2007, Pages 86–100.
7. Yaqian Guo, Trevor Hastie, Robert Tibshirani, “Regularized linear discriminant analysis and its application in microarrays”, *Biostatistics*, Volume 8, Issue 1, 1 January 2007, Pages 86–100.
8. S Ji, J Ye. "Generalized linear discriminant analysis: A Unified Framework and Efficient Model Selection," *IEEE Transactions on Neural Networks*, vol. 19, no. 10, 2008, 1768-1782.
9. Niijima, S., Okuno, Y., “Laplacian Linear Discriminant Analysis Approach to Unsupervised Feature Selection”, *Computational Biology and Bioinformatics*, 2009, 605-614.
10. Y. Yan, E. Ricci, R. Subramanian, G. Liu, N. Sebe, "Multitask linear discriminant analysis for view invariant action recognition", *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5599-5611, Dec. 2014.
11. D. Chu, L.-Z. Liao, M. K. P. Ng, X. Wang, "Incremental linear discriminant analysis: A fast algorithm and comparisons", *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2716-2735, Nov. 2015.
12. M.H. Siddiqi, R. Ali, A.M. Khan et al., "Human facial expression recognition using stepwise linear discriminant analysis and hidden conditional random fields", *IEEE Trans. Image Process.*, vol. 24, no. 4, pp. 1386-1398, 2015.
13. Liang He, Xianhong Chen, Can Xu, Jia Liu and Michael T. Johnson, “Local Pairwise Linear Discriminant Analysis for Speaker Verification”, *IEEE Signal Processing Letters*, Volume: 25 , Issue: 10 , Oct. 2018.