SMAI Assignment 3
Rehas Mehar Kaur Sachdeva
201401102

I. Non linear Kernal Fisher's LDA derivation:

Kernel Fisher discriminant analysis (KFD) is a kernelized version of linear discriminant analysis. Using the kernel trick, LDA is implicitly performed in a new feature space, which allows non-linear mappings to be learned.

Linear Discriminant Analysis:

Intuitively, the idea of LDA is to find a projection where class separation is maximized. Given two sets of labeled data, C_1 and C_2 , define the class means m_1 and m_2 to be:

$$m_i = \frac{1}{l_i} \sum_{n=1}^{n=l_i} x_n^i$$

where l_i is the number of examples of class C_i . The goal of linear discriminant analysis is to give a large separation of the class means while also keeping the in-class variance small. This is formulated as maximizing:

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

where S_B is the between-class covariance matrix and S_W is the total within-class covariance matrix:

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

$$S_W = \sum_{i=1,2} \sum_{n=1}^{n=l_i} (x_n^i - m_i)(x_n^i - m_i)^T$$

Differentiating $J(w)$ with respect to w , setting equal to zero, and rearranging gives:

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

Since we only care about the direction of w and $S_B w$ has the same direction as $(m_2 - m_1)$, $S_B w$ can be replaced by $(m_2 - m_1)$ and we can drop the scalars $w^T S_B w$ and $w^T S_W w$ to give:

$$w \propto S_W^{-1}(m_2 - m_1)$$

Kernel trick with LDA:

To extend LDA to non-linear mappings, the data can be mapped to a new feature space, F , via some function φ . In this new feature space, the function that needs to be maximized is:

$$J(w)=\frac{w^T S_B^{\varphi} w}{w^T S_W^{\varphi} w}$$

where

$$S_B^{\varphi}=(m_2^{\varphi}-m_1^{\varphi})(m_2^{\varphi}-m_1^{\varphi})^T$$

$$S_W^{\varphi}=\sum_{i=1,2}\sum_{n=1}^{n=l_i}(\varphi(x_n^i)-m_i^{\varphi})(\varphi(x_n^i)-m_i^{\varphi})^T$$

$$\text{and}\quad m_i^{\varphi}=\frac{1}{l_i}\sum_{n=1}^{n=l_i}\varphi(x_n^i)$$

Further, note that $w\in F$. Explicitly computing the mappings $\varphi(x_n^i)$ and then performing LDA can be computationally expensive, and in many cases intractable. For example, F may be infinitely dimensional. Thus, rather than explicitly mapping the data to F the data can be implicitly embedded by rewriting the algorithm in terms of dot products and using the kernel trick in which the dot product in the new feature space is replaced by a kernel function:

$$k(x,y)=\varphi(x)\cdot\varphi(y)$$

LDA can be reformulated in terms of dot products by first noting that w will have an expansion of the form:

$$w=\sum_{i=1}^{l}\alpha_i\varphi(x_i)$$

where $l=l_1+l_2$.

Then note that,

$$w^T m_i^{\varphi}=\frac{1}{l_i}\sum_{j=1}^{l}\sum_{k=1}^{l_i}\alpha_j k(x_j,x_k^i)=\alpha^T M_i$$

where,

$$(M_i)_j=\frac{1}{l_i}\sum_{k=1}^{l_i}k(x_j,x_k^i)$$

The numerator of $J(w)$ can then be written as:

$$w^T S_B^{\varphi} w=w^T(m_2^{\varphi}-m_1^{\varphi})(m_2^{\varphi}-m_1^{\varphi})^T w=\alpha^T M\alpha$$

where

$$M=(M_2-M_1)(M_2-M_1)^T$$

Similarly, the denominator can be written as:

$$w^T S_W^{\varphi} w=\alpha^T N\alpha$$

where

$$N=\sum_{j=1,2} K_j(I-1_{l_j})K_j^T$$

with the n^{th} , m^{th} component of K_j defined as $k(x_n,x_m^j)$, I is the identity matrix, and 1_{l_j} the matrix with all entries equal to $\frac{1}{l_j}$. This identity can be derived as follows:

$$w^T S_W^{\varphi} w=\left(\sum_{i=1}^{l}\alpha_i\varphi^T(x_i)\right)\left(\sum_{j=1,2}\sum_{n=1}^{n=l_j}(\varphi(x_n^j)-m_j^{\varphi})(\varphi(x_n^j)-m_j^{\varphi})^T\right)\left(\sum_{k=1}^{l}\alpha_k\varphi(x_k)\right)$$

$$=\sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\alpha_i\varphi^T(x_i)(\varphi(x_n^j)-m_j^{\varphi})(\varphi(x_n^j)-m_j^{\varphi})^T\alpha_k\varphi(x_k)$$

$$=\sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i k(x_i,x_n^j)-\frac{1}{l_j}\sum_{p=1}^{l_j}\alpha_i k(x_i,x_p^j)\right)\left(\alpha_k k(x_k,x_n^j)-\frac{1}{l_j}\sum_{q=1}^{l_j}\alpha_k k(x_k,x_q^j)\right)$$

$$=\sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(x_i,x_n^j)k(x_k,x_n^j)-\frac{2\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(x_i,x_n^j)k(x_k,x_p^j)+\frac{\alpha_i\alpha_k}{l_j^2}\sum_{p=1}^{l_j}\sum_{q=1}^{l_j}k(x_i,x_p^j)k(x_k,x_q^j)\right)\right)$$

$$=\sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(x_i,x_n^j)k(x_k,x_n^j)-\frac{\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(x_i,x_n^j)k(x_k,x_p^j)\right)\right)$$

$$=\sum_{j=1,2}\alpha^T K_j K_j^T\alpha-\alpha^T K_j 1_{l_j} K_j^T\alpha$$

$$=\alpha^T N\alpha$$

With these equations for the numerator and denominator of $J(w)$, the equation for J can be rewritten as:

$$J(\alpha)=\frac{\alpha^T M\alpha}{\alpha^T N\alpha}$$

Then, differentiating and setting equal to zero gives:

$$(\alpha^T M\alpha)N\alpha=(\alpha^T N\alpha)M\alpha$$

Since only the direction of w and hence the direction of α, matters, the above can be solved for α as:

$$\alpha = N^{-1}(M_2 - M_1)$$

Note that in practice, N is usually singular and so a multiple of the identity is added to it,

$$N_\varepsilon = N + \varepsilon I$$

Given the solution for α, the projection of a new data point is given by:

$$y(x) = w \cdot \varphi(x) = \sum_{i=1}^{l} \alpha_i k(x_i, x)$$

II. Datasets Used:

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

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