Saumya Rawat
201401110

Assignment 3 : SVM and Kernel Methods

<u>Problem 1:</u>

In cases that Fisher Discriminant Analysis cannot even recognize the structure in the data and therefore gives us no guarantee that discriminative features are preserved. The idea of nonlinear discriminant analysis is to transform the input data non-linearly, before standard linear discriminant analysis is applied.

Kernel Fisher Discriminant Analysis implicitly maps the input data into a high dimensional space H, where it performs regular Fisher Discriminant Analysis.

Given two sets of labeled data, C_1 and C_2 , let the class means m_1 and m_2 be

$$\mathbf{m}_i = \frac{1}{l_i}\sum_{n=1}^{l_i}\mathbf{x}_n^i,$$

where l_i – # of instances of class C_i . LDA maximises the distance between two classes and minimizes intra class variance. This can be formulated as

$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}},$$

where S_b – between class scatter and S_w - within class scatter. These are individually formulated as :

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$
$$\mathbf{S}_W = \sum_{i=1,2}\sum_{n=1}^{l_i}(\mathbf{x}_n^i - \mathbf{m}_i)(\mathbf{x}_n^i - \mathbf{m}_i)^T.$$

To extend LDA to non linear space, the data can be mapped to a new space F via Φ. We now need to maximize :

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \mathbf{S}_B^\phi \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W^\phi \mathbf{w}},$$

where

$$\mathbf{S}_B^\phi = (\mathbf{m}_2^\phi - \mathbf{m}_1^\phi)(\mathbf{m}_2^\phi - \mathbf{m}_1^\phi)^\top$$
$$\mathbf{S}_W^\phi = \sum_{i=1,2} \sum_{n=1}^{l_i} (\phi(\mathbf{x}_n^i) - \mathbf{m}_i^\phi)(\phi(\mathbf{x}_n^i) - \mathbf{m}_i^\phi)^\top,$$

$$\mathbf{m}_i^\phi = \frac{1}{l_i} \sum_{j=1}^{l_i} \phi(\mathbf{x}_j^i).$$

The kernel function can be given by

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}).$$

And w by,

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i \phi(\mathbf{x}_i).$$

Then note that

$$\mathbf{w}^\top \mathbf{m}_i^\phi = \frac{1}{l_i} \sum_{j=1}^{l} \sum_{k=1}^{l_i} \alpha_j k(\mathbf{x}_j, \mathbf{x}_k^i) = \alpha^\top \mathbf{M}_i,$$

where

$$(\mathbf{M}_i)_j = \frac{1}{l_i} \sum_{k=1}^{l_i} k(\mathbf{x}_j, \mathbf{x}_k^i).$$

The numerator of $J(\mathbf{w})$ can then be written as:

$$\begin{aligned}\mathbf{w}^\top \mathbf{S}_B^\phi \mathbf{w} &= \mathbf{w}^\top (\mathbf{m}_2^\phi - \mathbf{m}_1^\phi)(\mathbf{m}_2^\phi - \mathbf{m}_1^\phi)^\top \mathbf{w} \\ &= \alpha^\top \mathbf{M} \alpha,\end{aligned}$$

Also,

$$\begin{aligned}
\mathbf{w}^{\mathsf{T}}\mathbf{S}_W^{\phi}\mathbf{w} &= \left(\sum_{i=1}^{l}\alpha_i\phi^{\mathsf{T}}(\mathbf{x}_i)\right)\left(\sum_{j=1,2}\sum_{n=1}^{l_j}(\phi(\mathbf{x}_n^j)-\mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j)-\mathbf{m}_j^{\phi})^{\mathsf{T}}\right)\left(\sum_{k=1}^{l}\alpha_k\phi(\mathbf{x}_k)\right)\\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\alpha_i\phi^{\mathsf{T}}(\mathbf{x}_i)(\phi(\mathbf{x}_n^j)-\mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j)-\mathbf{m}_j^{\phi})^{\mathsf{T}}\alpha_k\phi(\mathbf{x}_k)\\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i k(\mathbf{x}_i,\mathbf{x}_n^j)-\frac{1}{l_j}\sum_{p=1}^{l_j}\alpha_i k(\mathbf{x}_i,\mathbf{x}_p^j)\right)\left(\alpha_k k(\mathbf{x}_k,\mathbf{x}_n^j)-\frac{1}{l_j}\sum_{q=1}^{l_j}\alpha_k k(\mathbf{x}_k,\mathbf{x}_q^j)\right)\\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i,\mathbf{x}_n^j)k(\mathbf{x}_k,\mathbf{x}_n^j)\right.\right.\\
&\quad \left.\left.-\frac{2\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i,\mathbf{x}_n^j)k(\mathbf{x}_k,\mathbf{x}_p^j)+\frac{\alpha_i\alpha_k}{l_j^2}\sum_{p=1}^{l_j}\sum_{q=1}^{l_j}k(\mathbf{x}_i,\mathbf{x}_p^j)k(\mathbf{x}_k,\mathbf{x}_q^j)\right)\right)\\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i,\mathbf{x}_n^j)k(\mathbf{x}_k,\mathbf{x}_n^j)-\frac{\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i,\mathbf{x}_n^j)k(\mathbf{x}_k,\mathbf{x}_p^j)\right)\right)\\
&= \sum_{j=1,2}\alpha^{\mathsf{T}}\mathbf{K}_j\mathbf{K}_j^{\mathsf{T}}\alpha-\alpha^{\mathsf{T}}\mathbf{K}_j\mathbf{1}_{l_j}\mathbf{K}_j^{\mathsf{T}}\alpha\\
&= \alpha^{\mathsf{T}}\mathbf{N}\alpha.
\end{aligned}$$

Hence we can write J(w) as :

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

Differentiating and setting to 0,

$$\alpha=\mathbf{N}^{-1}(\mathbf{M}_2-\mathbf{M}_1).$$

Since N *can* be singular,

$$\mathbf{N}_{\epsilon}=\mathbf{N}+\epsilon\mathbf{I}.$$

Hence the new set of data points can be given as ,

$$y(\mathbf{x}) = (\mathbf{w} \cdot \phi(\mathbf{x})) = \sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}).$$

Problem 2:

Datasets used : Validation sets of UCI Arcene Dataset, UCI Dexter Dataset

UCI Dexter Dataset :
The task of DEXTER is to filter texts about "corporate acquisitions". This is a two-class classification problem with sparse continuous input variables. There are 10001 attributes in total

	Positive ex.	Negative ex.
Training set	150	150
Validation set	150	150
Test set	1000	1000
All	1300	1300

a) Kernel PCA
Kernel PCA is a nonlinear generalization of PCA. To get nonlinear forms of PCA, we simpy choose a nonlinear kernel. Kernel PCA has the main advantage that it is simple dot product linear algenra and no nonlinear optimization is involved. Compared to principal curves, kernel PCA is so far harder to interpret in input space however, at least for polynomial kernels, it has a very clear interpretation in terms of higher order features.

Psuedo Code :

```
1. Compute Kernel
      _dists = pdist(X, 'wminkowski')
      sym_dists = squareform(_dists)
      K = exp(-gamma * sym_dists)

2. Center Kernel since data has to be standardizied
      kern_cent = KernelCenterer()
      K = kern_cent.fit_transform(K)
```

3. Compute Eigen Values and Eigen Vectors of K
```
 eig_vals, eig_vecs = np.linalg.eig(K)
 eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
```

4. Sort the (eigenvalue, eigenvector) tuples from high to low and choose top k eigen vectors
```
eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)
vec = np.array([ eig_pairs[i][1] for i in range(k)])
vec = vec.T # to make eigen vector matrix
```

Results :

1. Arcene Dataset
k = 100
rbf kernel
    1.1 Linear SVM : 55.56 % accuracy
    1.2 RBF SVM : 52.22 % accuracy

k = 10
rbf kernel
    1.1 Linear SVM : 54.34 % accuracy
    1.2 RBF SVM : 51.56 % accuracy

2. Dexter Dataset
k = 100
rbf kernel
    1.1 Linear SVM : 49.85 % accuracy
    1.2 RBF SVM : 49.74 % accuracy

k = 10
rbf kernel
    1.1 Linear SVM : 49.72 % accuracy
    1.2 RBF SVM : 50.03 % accuracy

b) *Kernel LDA*

Pseudo Code :

```
def k_lda(X,y):
    class_indices = []
    class_indices.append(np.where(y==-1)) #at pos 0
    class_indices.append(np.where(y==1))  #at pos 1

    n = X.shape[0]
    d = X.shape[1]

    n1 = X[class_indices[0]].shape[0]
    n2 = X[class_indices[1]].shape[0]
    K = X.dot(X.T)

1. Calculate Kernel matrix K1 n1xn1
    K1 = mlpy.kernel_gaussian(X, X[class_indices[0]], sigma=15)

    K2 = mlpy.kernel_gaussian(X, X[class_indices[1]], sigma=15)



  2. Calculate means
    2.1 Calculate mean matrix M1
    M1=np.zeros((n,1))
    class_no = 0
    for i,xi in enumerate(X):
            M1[i] = np.sum([[xi.dot(xj.T)] for xj in X[class_indices[class_no]]])

    2.2 Calculate mean matrix M2
    M2=np.zeros((n,1))
    class_no = 1
    for i,xi in enumerate(X):
            M2[i] = np.sum([[xi.dot(xj.T)] for xj in X[class_indices[class_no]]])

    3. Calculate between class scatter matrix
    M = (M2 - M1).dot((M2 - M1).T)

    4. Calculate within class scatter matrix
    N1 = (K1.dot(np.identity(n1) - (np.ones(n1)*n1))).dot(K1.T)
    N2 = (K2.dot(np.identity(n2) - (np.ones(n2)*n2))).dot(K2.T)
    N = N1 + N2;

    eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(N).dot(M))

    5.  Make a list of (eigenvalue, eigenvector) tuples
    eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

    6.  Sort the (eigenvalue, eigenvector) tuples from high to low
    eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)

    7.  Construct KxD eigenvector matrix W
    W = eig_pairs[0][1]
    ldaX = []
    for j in range(len(X)):
        ldaX.append( sum([W[i] * mlpy.kernel_gaussian(X[j], X[i], sigma=15) for i in range(n) ]))
    return ldaX
```

Results :

1. Arcene Dataset
k = 100
rbf kernel

    1.1 Linear SVM : 80.56 % accuracy
    1.2 RBF SVM : 82.22 % accuracy

k = 10
rbf kernel

    1.1 Linear SVM : 75.94 % accuracy
    1.2 RBF SVM : 77.56 % accuracy

2. Dexter Dataset
k = 100
rbf kernel

    1.1 Linear SVM : 69.06 % accuracy
    1.2 RBF SVM : 89.74 % accuracy

k = 10
rbf kernel

1.1 Linear SVM : 79.72 % accuracy
1.2 RBF SVM : 80.03 % accuracy

Observations :
1. Variation of γ in RBF kernel doesn't affect classification performance of SVM.
2. SVM with rbf kernel gives better results than linear SVM for these particular datasets.