Final Exam

Ahmet Gogebakan

2019700057

1

(a) The reason of poor performance could be overfitting. In SVM, choosing the right kernel and determining the kernel's parameter are so important. Since SVM can easily have overfitting. In this question, RBF kernel was used and γ and C parameters are set. We know that these parameters effect overfitting especially γ. γ is all about the decision boundary. If we set the γ parameter higher, the hyperplane will match the traning data higher. This can cause overfitting. If we change the parameter $\gamma = 0{,}0001$ to observe the test accuracy, we can see the accuracy increases upto 92.75%. It can solve the overfitting problem. It can conclued γ was selected high and causes overfitting. I also looked for the training accuracy. It 100%. When traning accuracy is so high and test accuracy is low, we can conclude there is an overfitting in here.

(b) Since classifier is fixed, we need to look for the given data. If we look at the data, every input data has fifty features in it. I thouhgt that reducing dimensionality make our model learn easier. We are not allowed to change the classifier so we have to preprocess the data. I chose to use the PCA. Since it can reduce the overfitting possiblities by reducing the dimension of the features data. I implemented PCA with different number of principal components. After the implementation, I observed that accuracy increased upto 79.25%. In addition to that, Before PCA, the data need to be scaled since PCA uses variances. One feature can dominate the others. PCA with scaling performs better than without scaling. With the help of this, accuracy increases upto 97%. After having two principal component, I observed that variance can change slightly. We can say after two principal component, the rest of the data affects the modal very little. I said two principal component but if we select two principal component, our accuracy will decrease to 81,25%. Since accuracy is more important, I chose the number of principal component one and accuracy is 97%.

(c) The implementation of the question is shown in figure 1. First I loaded the given data. I use a StandartScaler. After that for every feature, I loop over the PCA with the number of principal components 1 to 50. I get the accuracy and the variances according to all the component's number. The results can be seen in figure 2. I also plotted the changes in variances. We can observe that after having two principal components, variance doesn't change significantly.

```python
import numpy as np
from sklearn import svm
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from matplotlib import rcParams

#Load the data
train_data = np.load("train_data1.npy")
train_label = np.load("train_label1.npy")
test_data = np.load("test_data1.npy")
test_label = np.load("test_label1.npy")

#Scale the data
scaler = StandardScaler()
train_data = scaler.fit_transform(train_data)

accuracy = []

#Looking all 50 features in PCA(n_components)
for i in range (1,50):
    pca = PCA(n_components=i)
    train_data_new = pca.fit_transform(train_data)
    test_data_new = pca.transform(test_data)

    #Our classifier without any change
    clf = svm.SVC(gamma=0.001, C=100)
    clf.fit(train_data_new, train_label)
    y_pred = clf.predict(test_data_new)
    correct_prediction = np.equal(y_pred, test_label)
    accuracy.append(np.mean(correct_prediction.astype(np.float32)))

#printing and plotting the results
print("Accuracy array : " ,accuracy)
print("Variances : " , pca.explained_variance_ratio_)

fig, ax = plt.subplots()
rcParams.update({'font.size': 22})
ax.plot( list(range(1,10)), pca.explained_variance_ratio_[0:9] )
plt.show()
```

Figure 1: Snapshot of the code

```
Accuracy array :  [0.97, 0.8125, 0.6725, 0.6525, 0.62, 0.5275, 0.525, 0.51, 0.5075, 0.505, 0.5125, 0.5075, 0.5025, 0.
4975, 0.4975, 0.5025, 0.4975, 0.505, 0.495, 0.495, 0.495, 0.495, 0.495, 0.4975, 0.495, 0.495, 0.495, 0.4975, 0.5, 0.
5, 0.5025, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505, 0.505,
0.505, 0.505, 0.505]
Variances :  [0.27489315 0.03580847 0.03228208 0.02945601 0.02372251 0.02317342
 0.02278601 0.02212703 0.02128808 0.02117751 0.02093833 0.02032178
 0.0198045  0.01926425 0.01888901 0.01864759 0.0182435  0.01807865
 0.01723025 0.01667632 0.0164132  0.01595554 0.01564376 0.01499879
 0.01455966 0.01450604 0.01417195 0.01367882 0.01294987 0.01241932
 0.01224347 0.01144961 0.01120154 0.01034536 0.00979224 0.00939563
 0.00923508 0.00894398 0.00863126 0.00840965 0.00801827 0.00711805
 0.00683981 0.00659982 0.0064621  0.00625771 0.005853   0.00567771
 0.0052675 ]
```

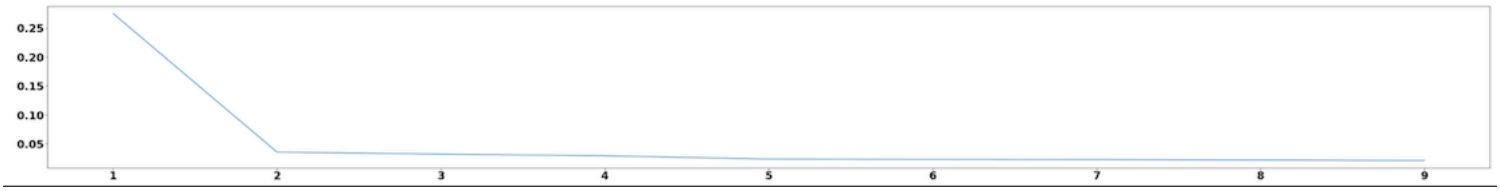

Figure 2: Accuracy and variances

2

(a) If we don't know the labels, it is an unsupervised learning. I chose the k-means algorithm to obtain the labels. Since this algorithm is simplest and finds the groups of the given data. We can get the same label in the same group easily. Next step is the choosing the right k value for the k-means. For finding the best k, I tried 5 different k values and look for their sum of squared distances to their closest cluster center. I printed the values. If we look at the values carefully, after 3, there is no slightly change. Therefore, I chose the k value 3. After selecting k, I got the labels (0,1,2) for each input.

```python
import numpy as np
from sklearn import svm
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

#Load the data
all_data = np.load("data2.npy")

#Look for which is suitable for k
k_mean_array = []
for means in range (1,6):
    k_mean_model = KMeans(means)
    k_mean_model.fit(all_data)
    k_mean_array.append(k_mean_model.inertia_)

print(k_mean_array)

#Chose the number of clusters
num_of_cluster = 3

#Set the labels
k_mean = KMeans(n_clusters=num_of_cluster)
k_mean.fit(all_data)
k_mean.cluster_centers_
labels=k_mean.labels_
```

```
[5194.485603786112, 2205.400186600814, 914.8020480609057, 795.9448024734747, 690.9651402330917]
```

Figure 3: Snapshot of the code

(b) I labeled the data with (0,1,2) for each input. I split the data into train and test which are 1600 train and 400 test data with the same as first question. And then use same classifier. I got the accuracy 99.5%.

```
import numpy as np
from sklearn import svm
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

#Load the data
all_data = np.load("data2.npy")

#Look for which is suitable for k
k_mean_array = []
for means in range (1,6):
    k_mean_model = KMeans(means)
    k_mean_model.fit(all_data)
    k_mean_array.append(k_mean_model.inertia_)

#print(k_mean_array)

#Chose the number of clusters
num_of_cluster = 3

#Set the labels
k_mean = KMeans(n_clusters=num_of_cluster)
k_mean.fit(all_data)
k_mean.cluster_centers_
labels=k_mean.labels_

#Split the data for the traning and test
train_data, test_data, train_label, test_label = train_test_split(
    all_data,labels,test_size=0.2,shuffle=False)

#SVM modal that we used in question(1)
clf = svm.SVC(gamma=0.001, C=100.)
clf.fit(train_data, train_label)
y_pred = clf.predict(test_data)
correct_prediction = np.equal(y_pred, test_label)
accuracy = np.mean(correct_prediction.astype(np.float32))

print(accuracy)
```

```
0.995
```

Figure 4: Accuracy and k-means

3

If we try to behave each attribute as the root node, we simply calculate and look for the information gains. Most information gain means lowest entropy. If our splitting completely homogeneous, this means entropy is zero and if it is equally divided, it means entropy one. We want less entropy. I think, insulin or glucose would have less entropy, probably have more information gain than others . Since I think they have more information than others about the health or diabetes. I think patient ID is not meaningful to learn a generalizable tree. It is not very relevant to classify patients as health or having diabetes. It would have less information gain and by this way, have more entropy.

4

If we look at these two approaches that handle the missing data, the means of the attribute that have missing data are same in both approaches. However, means of other attributes can change in these two approaches. Since one mean is calculation of 800 input, the other mean is calculation of 1000 input. Since the formula of covariance of two variables X and Y is ;

$$cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{y}) \tag{1}$$

where $\bar{x}$ and $\bar{y}$ are means.
Different means of attributes causes different covariance matrices. After finding the covariance matrix between all attributes, we can easily compute eigenvalues and eigenvectors from that matrix. Let say A be a square matrix (covariance matrix) v a vector and λ a scalar. We know that,

$$A \cdot v = \lambda \cdot v \tag{2}$$

λ is called eigenvalue and v is eigenvector of A. We can find eigenvalues of A by solving the characteristic equation,

$$det(A - \lambda I) = 0 \tag{3}$$

But in these two approaches , we have different covariance matrices because of the different values of means. We can find eigenvalues from the above equation and calculate eigenvectors easily.
To summarize it, these two approaches could have different mean values for the attributes that don't have any missing data. This would cause different covariance matrices. These different covariance matrices causes different eigenvalues and eigenvectors. Since eigenvectors are different, we can simply conclude that principal components are different in these two approaches.

5

(a) According to the given data;
$X = \begin{bmatrix} 1 & 0 \\ 3 & 0 \end{bmatrix}$, $Y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, if we solve the second equation which is $y_i(w^T x_i + b) \geq 1$ for every data point, we will get two equations;

$$-1 \cdot (w_1 \cdot 1 + w_2 \cdot 0 + b \geq 1) = -1 \cdot (w_1 + b) \geq 1 \tag{4}$$

$$1 \cdot (w_1 \cdot 3 + w_2 \cdot 0 + b \geq 1) = 3 \cdot w_1 + b \geq 1 \tag{5}$$

If we solve the equation for w_1 sum of two equation (1) and (2) , we get $2w_1 \geq 2$ and simply $w_1 \geq 1$. Or if we solve the equation for b, multiple first (1) equation by 3 and sum equations again (1) and (2). We get $b \leq -2$. w_2 doesn't depend on any equation. Because of the first constrain, we have to minimize the term $\frac{1}{2}w^T w = \frac{1}{2}(w_1^2 + w_2^2)$. Therefore, we simply say the value of $w_2 = 0$ and $w_1 = 1$. To sum up all these; our optimal solution for w and b becomes;
$w^* = \begin{bmatrix} w_1 = 1 \\ w_2 = 0 \end{bmatrix}$, $b^* = -2$

(b) Since we have two data, for finding α;

$$-\frac{1}{2}\sum_{n=1}^{2}\sum_{m=1}^{2} y_n y_m \alpha_n \alpha_m x_n^T x_m + \sum_{n=1}^{2} \alpha_n \tag{6}$$

We need to open this equation according to the inputs and labels;

$$-\frac{1}{2}\sum_{n=1}^{2}(y_n y_1 \alpha_n \alpha_1 x_n^T x_1 + y_n y_2 \alpha_n \alpha_2 x_n^T x_2) + \sum_{n=1}^{2} \alpha_n \tag{7}$$

$$-\frac{1}{2}(y_1 y_1 \alpha_1 \alpha_1 x_1^T x_1 + y_1 y_2 \alpha_1 \alpha_2 x_1^T x_2 + y_2 y_1 \alpha_2 \alpha_1 x_2^T x_1 + y_2 y_2 \alpha_2 \alpha_2 x_2^T x_2) + \alpha_1 + \alpha_2 \tag{8}$$

We know the values $y_1 = -1$ and $y_2 = 1$, we need to find the necessary values.
$x_1^T x_1 = [1, 0] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$, $x_1^T x_2 = [1, 0] \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 3$, $x_2^T x_1 = [3, 0] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 3$,
$x_2^T x_2 = [3, 0] \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 9$, $y_1 y_1 = 1$, $y_1 y_2 = 1$, $y_2 y_1 = -1$
If we replace that values into the equation, we can get;

$$-\frac{1}{2}(\alpha_1^2 - 3\alpha_1\alpha_2 - 3\alpha_2\alpha_1 + 9\alpha_2^2) + \alpha_1 + \alpha_2 = -\frac{1}{2}(\alpha_1^2 - 6\alpha_1\alpha_2 + 9\alpha_2^2) + \alpha_1 + \alpha_2 \tag{9}$$

We know also, $\sum_{n=1}^{2} y_n \alpha_n = 0$. This means $y_1\alpha_1 + y_2\alpha_2 = 0$. Since $y_1 = -1$ and $y_2 = 1$, we can conclude $\alpha_1 = \alpha_2$. If we replace this knowledge in equation 6.

$$-\frac{1}{2}(\alpha_1^2 - 6\alpha_1^2 + 9\alpha_1^2) + 2\alpha_1 = -\frac{1}{2}4\alpha_1^2 + 2\alpha_1. \tag{10}$$

If we take the derivate of this term with respect to α_1, $2 - 4\alpha_1 = 0$. We get the results of $\alpha_1 = \alpha_2 = \frac{1}{2}$. $\alpha = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$

(c) For finding w^*, we use the equation of ;

$$w^* = \sum_{n=1}^{2} \alpha_n y_n x_n = \alpha_1 y_1 x_1 + \alpha_2 y_2 x_2 = \frac{1}{2} \cdot (-1) \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{2} \cdot 1 \cdot \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} + \frac{3}{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (11)$$

which is the same result that we found in part (a).