Laboratory Work 2

ML & Data Mining

Q1. Watch the video on Support Vector Machines: https://www.youtube.com/watch?v=efR1C6CvhmE

Q2. Use the Social_Network_Ads.csv file to build an SVM model. The dataset contains data about the age and salary of customers of a company that produces cars. Recently the company released a new SUV and organized a marketing campaign. The dependent variable (Purchased) is a categorical variable that takes the value of 1 if a customer purchased the new SUV after the campaign and zero otherwise. Navigate scikit learn library's API here: https://scikitlearn. org/stable/modules/classes.html to find the description of the SVC classifier in the svm module of the library. Use the SVC classifier to predict whether one will purchase the new SUV or not. Follow the following structure in your approach:

1. Import libraries

In [2]:
```
from sklearn import svm
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
```

2. Import dataset

In [3]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

3. Split the dataset into Training and Test groups (use 20-80 split, i.e. 20% of data will be used for the Test group and 80 for training).

In [4]:
```
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [5]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

4. Perform feature scaling. (do not scale y – remember y=0/1 so it needs no scaling).

In [6]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

5. Train the SVC classifier (for the classifier specify random_state=0 to ensure that everybody gets the same results. Also choose the 'linear' kernel -- you must specify this choice, otherwise the default kernel is 'rbf').

In [7]:
```
clf = svm.SVC()
clf.fit(x_train, y_train)
```

Out[7]:
```
SVC()
```

6. Make predictions and compare the results by displaying the predicted values of y next to the test values of y in a two-dimensional array.

In [8]:
```
y_pred = clf.predict(x_test)
y_pred
```

Out[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1], dtype=int64)
```

7. Create and print the Confusion Matrix and the accuracy score of the model and interpret the two.

In [9]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
[[55  3]
 [ 1 21]]
Accuracy: 95.00%
```

8. Use the K-fold cross-validation method (use K=10; in python - cv=10) to ensure that the accuracy score calculated only on one test set was not just a lucky occurrence. Print the best accuracy score and the standard deviation of the scores computed. Comment on the result, compare to the score computed when using only one test set in (7).

In [10]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = clf, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 90.00 %
Standard Deviation: 4.80 %
```

We can observe that the mean accuracy of the model calculated for 10 smaller test data sets is smaller with 5% than the accuracy of the model calculated on only one data set. This means that there was a little bit of lucky occurence for the model.

9. Implement the grid search method to tune the following two hyper parameters:

C - is a parameter of regularization. One can control the possibility of overfitting by specifying different values of C. Try the following values for C = 0.25, 0.5, 0.75, 1.

Kernel – is the kernel used to manipulate the data. Try the following two kernels: 'linear', 'rbf'.

In [13]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear']},
              {'C': [0.25, 0.5, 0.75, 1], 'kernel': ['rbf']}]
grid_search = GridSearchCV(estimator = clf,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search.fit(x_train, y_train)
```

Out[13]:
```
GridSearchCV(cv=10, estimator=SVC(), n_jobs=-1,
             param_grid=[{'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear']},
                         {'C': [0.25, 0.5, 0.75, 1], 'kernel': ['rbf']}],
             scoring='accuracy')
```

10. Print the accuracy score of the best performing model found by the grid search. Print the parameters of the best model.

In [14]:
```
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)
```

```
Best Accuracy: 90.00 %
Best Parameters: {'C': 1, 'kernel': 'rbf'}
```

We can observe that the best parameters for the best accuracy are: "C" = 1, "kernel" = "rbf". The accuracy is the same as the mean accuracy for k-fold validation when calculating the accuracies of the model in 10 iterations for 10 smaller data sets.