# Chapter1

## 1-Can you tell the difference between machine learning and traditional

یادگیری ماشین (Machine Learning)یکی از زیر مجموعه های هوش مصنوعی می باشد که سیستم ها به کمک آن می توانند بطور اتوماتیک و بدون نیاز به هیچگونه برنامه نویسی صریحی یادگیری و پیشرفت کنند. تمرکز اصلی یادگیری ماشین بر توسعه برنامه های کامپیوتری می باشد که بتوانند به اطلاعات دسترسی یافته و از آن برای یادگیری خود بهره ببرند

از آنجایی که استفاده از نیروی انسانی هزینه بر است و همچنین دقت نیروی انسانی در یک فرآیند طولانی مدت پایین می آید از ماشین لرنینگ Machine learningاستفاده می شود زیرا تکرار پذیری بالاتری دارد به عبارتی یک کار را هزار بار با یک تکرار و یک دقت انجام میدهد. در مجموع می توان مزایای ماشین لرنینگ را به صورت زیر عنوان نمود:

۱ -به راحتی روند ها و الگو ها را شناسایی می کند
۲ -هیچ مداخله انسانی نیاز نیست (اتوماسیون)
۳ -بهبود مستمر دقت و کارایی در حین کار و کسب تجربه
۴ -مدیریت داده های چند بعدی و چند متغیره
۵- کاربرد های گسترده

در عین حال یادگیری ماشین معایبی هم دارد که به ترتیب زیر است:

۱- نیاز به مجموعه داده های گسترده
۲- نیاز به زمان و منابع کافی جهت یادگیری و پیشرفت
۳- نیاز به دقت در تفسیر نتایج
۴- حساسیت به خطای بالا

در برنامه نویسی سنتی، برنامه نویس برنامه را به طور کامل به کامپیوتر میداد و دقیقا به کامپیوتر می گفت که چه کار کند، در واقع فرآیند کاملا دستی بود، و برنامه نویس برنامه را خلق میکرد. پس می توان گفت در برنامه نویسی سنتی، فرآیند به شکل زیر بوده است:

**خروجی = برنامه + ورودی**

اما در یادگیری ماشین یا ماشین لرنینگ، فرآیند کاملا اتوماتیک است، به این شکل که ما دیگر به کامپیوتر برنامه ای نمیدهیم و در عوض ورودی و خروجی را می دهیم و برنامه را میگیریم؛ پس فرآیند ماشین لرنینگ به این صورت است:

**برنامه = ورودی + خروجی**

برای مثال، فرض کنید که میخواهیم ببینیم در سازمانمان مشتریان دچار ریزش می شوند یا خیر؟ ورودی می‌شود تراکنش های مشتریان و خروجی هم که دو حالت بیشتر ندارد یا بله یا خیر ورودی ما خواهد بود. در نهایت هم برنامه همان مدل پیشگویانه است.

در نتیجه خیلی راحت میبینید که میتوانیم با یک برنامه نویسی ساده از دیتاهای سازمان خود که هر روز هم تعدادشان بیشتر می شود طوری استفاده کنیم که در نهایت به دارایی مالی مان تبدیل شود.

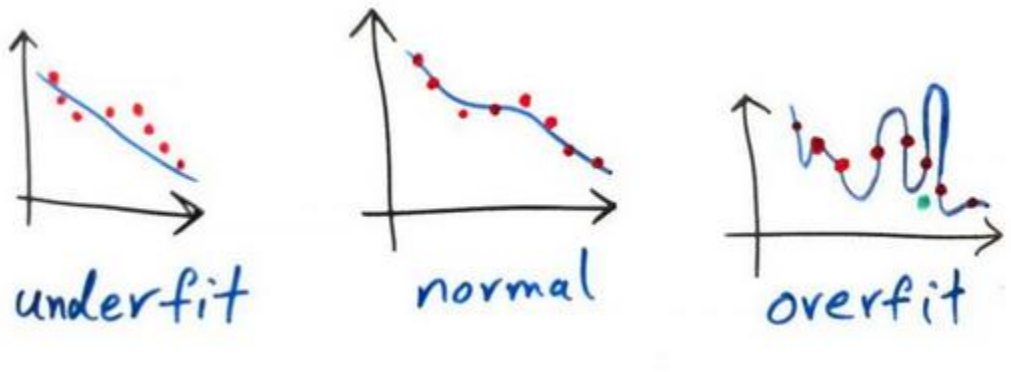
2-What's overfitting and how do we avoid it?

این دو مفهوم از مفاهیمِ اساسی است که در بحثِ طبقه‌بندیِ داده‌ها مورد بحث قرار می‌گیرند .

در دنیای الگوریتم‌ها Overfit شدن به معنای این است که الگوریتم فقط داده‌هایی که در مجموعه آموزشی (train set)یاد گرفته است را می‌تواند به درستی پیش‌بینی کند ولی اگر داده‌ای کمی از مجموعه‌ی آموزشی فاصله داشته باشد، الگوریتمی که Overfit شده باشد، نمی‌تواند به درستی پاسخی برای این داده‌های جدید پیدا کند و آن‌ها را با اشتباهِ زیادی طبقه‌بندی می کند.

Underfitشدن نیز زمانی رخ می دهد که الگوریتم یک مدلِ خیلی کلی از مجموعه آموزشی به دست می‌آورد. یعنی حتی اگر خودِ داده‌های مجموعه‌ی آموزشی را نیز به این الگوریتم بدهیم، این الگوریتم خطایی قابل توجه خواهد داشت.

فرض کنید نقطه‌ها در شکل زیر نمونه سوالاتی هستند که استاد برای آمادگی در امتحان همراه با پاسخِ آن‌ها به ما داده است. سوال در محور افقی داده می‌شود و پاسخ در محور عمودی است. به این معنی که به شما X داده می‌شود و شما باید از روی عددِ اینX ، عددِ Y (عدد روی محور عمودی) را تشخیص دهید .مثلاً اگر مختصاتِ **عدد نخست سمت چپ در تصویر زیر (۶ ,۱)** باشد، به این معنی است که اگر عدد **۱** را به این الگوریتم بدهیم، الگوریتم عدد **۶** را برگرداند. پس با این حساب اگر به الگوریتم عدد **۱,۱** را دادیم، این الگوریتم (که یادگیری را قبلاً از روی داده‌ها انجام داده است) احتمالاً باید عددی نزدیک به **۶** را برگرداند. این ها در واقع همان مجموعه آموزشی ما هستند:




خط آبیِ موجود، در واقع یادگیریِ مدل طبقه‌بندی است (به صورت دقیق‌تر در این‌جا رگرسیون داریم). همان‌طور که می‌بینید در سمت چپ، خطی که الگوریتمِ طبقه‌بندیْ یادگرفته است از تمامی داده‌ها به مقدار قابل توجهی فاصله دارد .**یعنی در این شکل (سمت چپ) underfitting** رخ داده است. این در حالی است که در شکل سمت راست، **overfitting رخ داده** .توجه کنید که در شکلِ سمت راست، اگر یک نقطه جدید (مثلا یک سوال جدید در امتحان) داده شود (نقطه سبز رنگ داده شده) الگوریتم خطای بسیار زیادی دارد. یعنی مقدار Y ی که برمیگرداند بسیار با مقدار واقعی فاصله دارد چون الگوریتم خیلی نتوانسته است که یادگیری را عمومی سازی کند و نسبت به مقادیر جدید خطای بالایی نشان می‌دهد .شکل وسط نیز یک خط معقول و درست برای یک طبقه بند را نشان می دهد که overfitیا underfit نشده است.

3- Name two feature engineering approaches:

Answer: **Deep neural networks** are often able to derive features **automatically**.
We will briefly look at several techniques such as polynomial features, power transformations, and binning.

4- Name two ways to combine multiple models.

Answer: prefer to have our models cooperate with the following schemes:
• Voting and averaging
• Bagging
• Boosting
• Stacking

# Chapter2

1- As mentioned earlier, we extracted user-movie relationships only from the movie rating data where most ratings are unknown. Can you also utilize data from the files movies.dat and users.dat?

پاسخ: بله می‌توان از این دو فایل نیز استفاده نمود.

2- Practice makes perfect—another great project to deepen your understanding could be heart disease classification. The dataset can be downloaded directly at https://www.kaggle.com/ronitf/heart-disease-uci, or from the original page at https://archive.ics.uci.edu/ml/datasets/Heart+Disease

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will
list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
['heart.csv']
```

# Read Data

```
# We are reading our data
df = pd.read_csv("../input/heart.csv")
# First 5 rows of our data
df.head()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | 1 |

Data contains;

- age - age in years
- sex - (1 = male; 0 = female)
- cp - chest pain type

- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg - resting electrocardiographic results
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest
- slope - the slope of the peak exercise ST segment
- ca - number of major vessels (0-3) colored by flourosopy
- thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
- target - have disease or not (1=yes, 0=no)

# Data Exploration

```
df.target.value_counts()
1    165
0    138
Name: target, dtype: int64
sns.countplot(x="target", data=df, palette="bwr")
plt.show()


countNoDisease = len(df[df.target == 0])
countHaveDisease = len(df[df.target == 1])
print("Percentage of Patients Haven't Heart Disease:
{:.2f}%".format((countNoDisease / (len(df.target))*100)))
print("Percentage of Patients Have Heart Disease:
{:.2f}%".format((countHaveDisease / (len(df.target))*100)))
Percentage of Patients Haven't Heart Disease: 45.54%
Percentage of Patients Have Heart Disease: 54.46%
sns.countplot(x='sex', data=df, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()


countFemale = len(df[df.sex == 0])
countMale = len(df[df.sex == 1])
print("Percentage of Female Patients: {:.2f}%".format((countFemale /
(len(df.sex))*100)))
print("Percentage of Male Patients: {:.2f}%".format((countMale /
(len(df.sex))*100)))
Percentage of Female Patients: 31.68%
Percentage of Male Patients: 68.32%
df.groupby('target').mean()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| target | | | | | | | | | | | | | |
| 0 | 56.601449 | 0.826087 | 0.478261 | 134.398551 | 251.086957 | 0.159420 | 0.449275 | 139.101449 | 0.550725 | 1.585507 | 1.166667 | 1.166667 | 2.543478 |
| 1 | 52.496970 | 0.563636 | 1.375758 | 129.303030 | 242.230303 | 0.139394 | 0.593939 | 158.466667 | 0.139394 | 0.583030 | 1.593939 | 0.363636 | 2.121212 |

```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```

```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#1CA53B'
,'#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()
```

```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```

```
pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#DAF7A
6','#FF5733' ])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```

```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#FFC300'
,'#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
```

```
plt.show()


pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA',
'#AA1190' ])
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```

## Creating Dummy Variables

Since 'cp', 'thal' and 'slope' are categorical variables we'll turn them into dummy variables.

```
a = pd.get_dummies(df['cp'], prefix = "cp")
b = pd.get_dummies(df['thal'], prefix = "thal")
c = pd.get_dummies(df['slope'], prefix = "slope")
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | ... | cp_1 | cp_2 | cp_3 | thal_0 | thal_1 | thal_2 | thal_3 | slope_0 | slope_1 | slope_2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | ... | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | ... | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 25 columns

```
df = df.drop(columns = ['cp', 'thal', 'slope'])
df.head()
```

| | age | sex | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | ca | ... | cp_1 | cp_2 | cp_3 | thal_0 | thal_1 | thal_2 | thal_3 | slope_0 | slope_1 | slope_2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 1 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | ... | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 37 | 1 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | ... | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 41 | 0 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 56 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 4 | 57 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

5 rows × 22 columns

# Creating Model for Logistic Regression

We can use sklearn library or we can write functions ourselves. Let's them both. Firstly we will write our functions after that we'll use sklearn library to calculate score.

```
y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

Normalize Data

```
# Normalize
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

We will split our data. 80% of our data will be train data and 20% of it will be test data.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size =
0.2,random_state=0)
#transpose matrices
x_train = x_train.T
y_train = y_train.T
```

```
x_test = x_test.T
y_test = y_test.T
```

Let's say weight = 0.01 and bias = 0.0

```
#initialize
def initialize(dimension):

    weight = np.full((dimension,1),0.01)
    bias = 0.0
    return weight,bias
```

Sigmoid Function

```
def sigmoid(z):

    y_head = 1/(1+ np.exp(-z))
    return y_head
```

Forward and Backward Propagation

Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Gradient Descent

**Gradient Descent**

Remember that the general form of gradient descent is:

$$\begin{aligned} &Repeat\ \{ \\ &\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ &\} \end{aligned}$$

We can work out the derivative part using calculus to get:

$$\begin{aligned} &Repeat\ \{ \\ &\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\ &\} \end{aligned}$$

By the way in formulas;

- h0(x^i)= y_head
- y^i = y_train
- x^i = x_train

```
def forwardBackward(weight,bias,x_train,y_train):
    # Forward

    y_head = sigmoid(np.dot(weight.T,x_train) + bias)
    loss = -(y_train*np.log(y_head) + (1-y_train)*np.log(1-y_head))
    cost = np.sum(loss) / x_train.shape[1]

    # Backward
    derivative_weight = np.dot(x_train,((y_head-y_train).T))/x_train.shape[1]
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]
    gradients = {"Derivative Weight" : derivative_weight, "Derivative Bias" :
derivative_bias}

    return cost,gradients
def update(weight,bias,x_train,y_train,learningRate,iteration) :
    costList = []
    index = []

    #for each iteration, update weight and bias values
    for i in range(iteration):
```

```
        cost,gradients = forwardBackward(weight,bias,x_train,y_train)
        weight = weight - learningRate * gradients["Derivative Weight"]
        bias = bias - learningRate * gradients["Derivative Bias"]

        costList.append(cost)
        index.append(i)

    parameters = {"weight": weight,"bias": bias}

    print("iteration:",iteration)
    print("cost:",cost)

    plt.plot(index,costList)
    plt.xlabel("Number of Iteration")
    plt.ylabel("Cost")
    plt.show()

    return parameters, gradients
def predict(weight,bias,x_test):
    z = np.dot(weight.T,x_test) + bias
    y_head = sigmoid(z)

    y_prediction = np.zeros((1,x_test.shape[1]))

    for i in range(y_head.shape[1]):
        if y_head[0,i] <= 0.5:
            y_prediction[0,i] = 0
        else:
            y_prediction[0,i] = 1
    return y_prediction
def
logistic_regression(x_train,y_train,x_test,y_test,learningRate,iteration):
    dimension = x_train.shape[0]
    weight,bias = initialize(dimension)

    parameters, gradients =
update(weight,bias,x_train,y_train,learningRate,iteration)

    y_prediction = predict(parameters["weight"],parameters["bias"],x_test)

    print("Manuel Test Accuracy: {:.2f}%".format((100 -
np.mean(np.abs(y_prediction - y_test))*100)))
logistic_regression(x_train,y_train,x_test,y_test,1,100)
iteration: 100
cost: 0.3597736123664534


Manuel Test Accuracy: 86.89%
```

# Manuel Test Accuracy is **86.89%**</font>

Let's find out sklearn's score.

Sklearn Logistic Regression

```
accuracies = {}
```

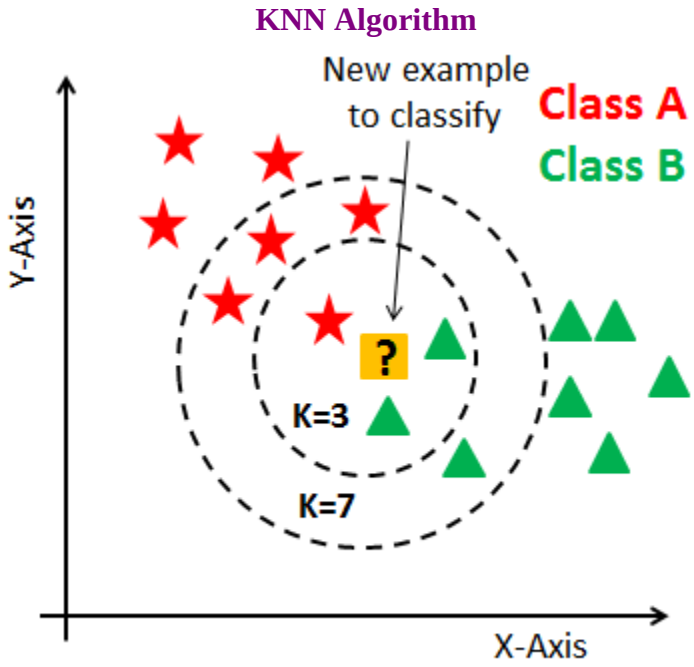
```
lr = LogisticRegression()
lr.fit(x_train.T,y_train.T)
acc = lr.score(x_test.T,y_test.T)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))
Test Accuracy 86.89%
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

1. ## Our model works with **86.89%** accuracy.</font>

# K-Nearest Neighbour (KNN) Classification

Let's see what will be score if we use KNN algorithm.




```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train.T, y_train.T)
prediction = knn.predict(x_test.T)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test.T, y_test.T)*100))
2 NN Score: 77.05%
# try ro find best k value
```

```
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
    scoreList.append(knn2.score(x_test.T, y_test.T))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))


Maximum KNN Score is 88.52%
```

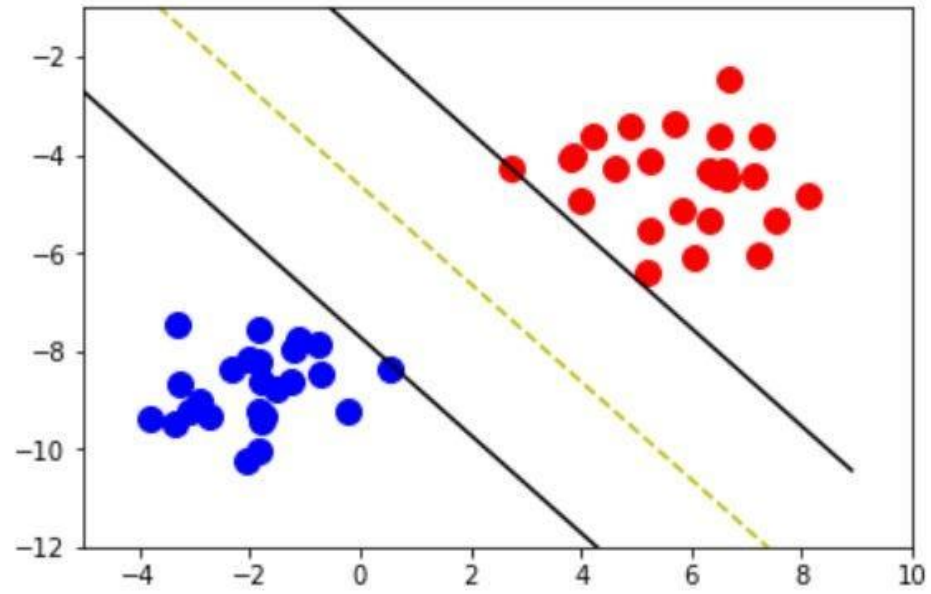
As you can see above if we define k as 3-7-8 we will reach maximum score.

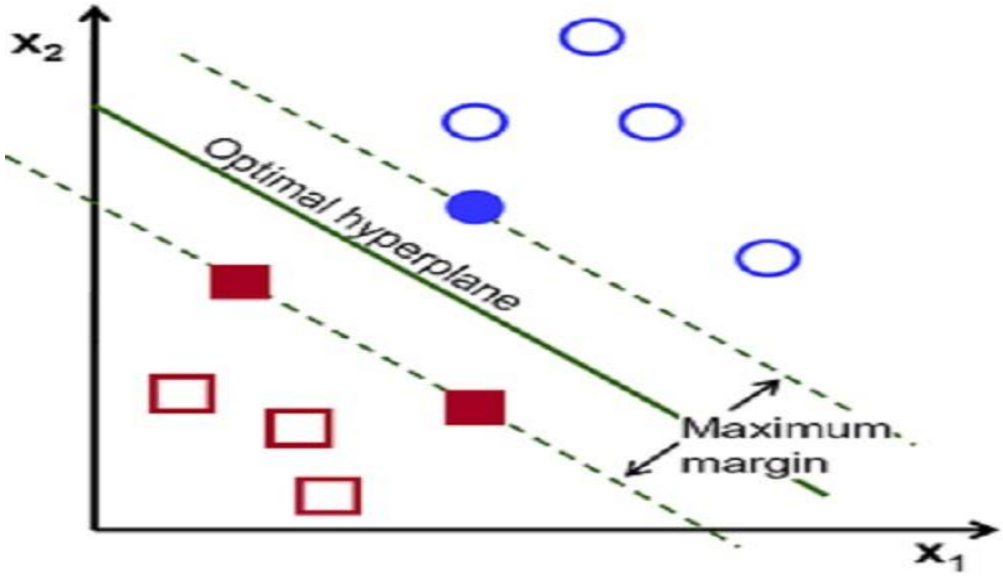
# KNN Model's Accuracy is 88.52%</font>

# Support Vector Machine (SVM) Algorithm

Now we will use SVM algorithm.

**Support Vector Machine Algorithm**



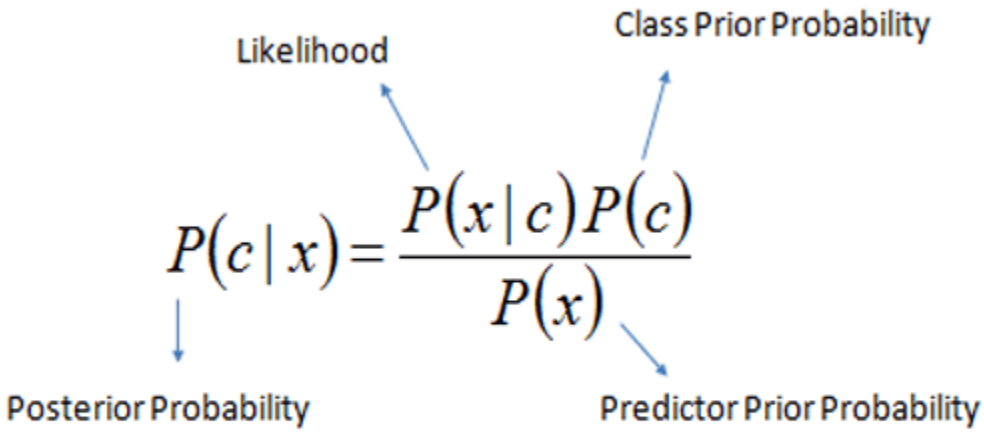

```
from sklearn.svm import SVC
svm = SVC(random_state = 1)
svm.fit(x_train.T, y_train.T)

acc = svm.score(x_test.T,y_test.T)*100
accuracies['SVM'] = acc
print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
Test Accuracy of SVM Algorithm: 86.89%
/opt/conda/lib/python3.6/site-packages/sklearn/svm/base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale'
in version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

# Test Accuracy of SVM Algorithm is 86.89%</font>

# Naive Bayes Algorithm

**Naive Bayes Algorithm**




$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

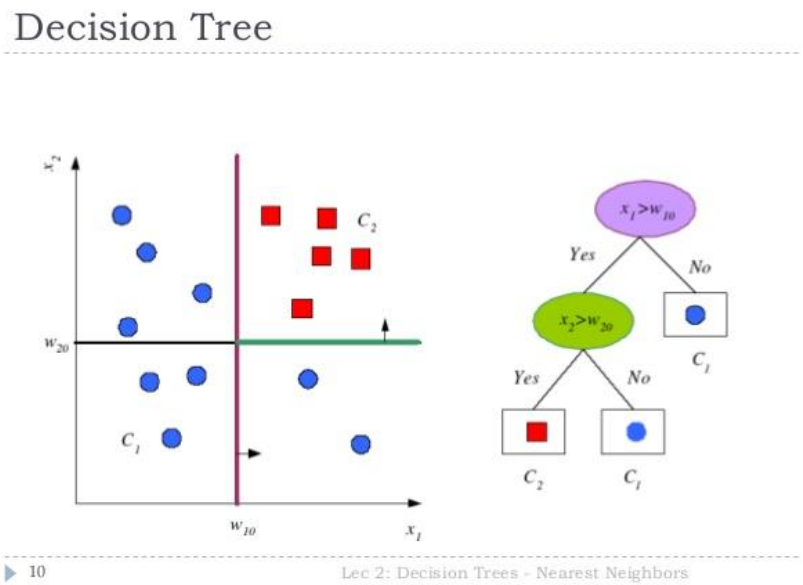
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train.T, y_train.T)

acc = nb.score(x_test.T,y_test.T)*100
accuracies['Naive Bayes'] = acc
print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
Accuracy of Naive Bayes: 86.89%
```

# Accuracy of Naive Bayes: 86.89%</font>

# Decision Tree Algorithm

**Decision Tree Algorithm**




```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train.T, y_train.T)

acc = dtc.score(x_test.T, y_test.T)*100
accuracies['Decision Tree'] = acc
print("Decision Tree Test Accuracy {:.2f}%".format(acc))
Decision Tree Test Accuracy 80.33%
```

# Test Accuracy of Decision Tree Algorithm: 78.69%</font>

# Random Forest Classification

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train.T, y_train.T)

acc = rf.score(x_test.T,y_test.T)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
Random Forest Algorithm Accuracy Score : 88.52%
```

# Test Accuracy of Random Forest: 88.52%</font>

# Comparing Models

```
colors = ["purple", "green", "orange", "magenta","#CFC60E","#0FBBAE"]
```

```
sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()),
palette=colors)
plt.show()
```

Our models work fine but best of them are KNN and Random Forest with 88.52% of accuracy. Let's look their confusion matrixes.

# Confusion Matrix

```
# Predicted values
y_head_lr = lr.predict(x_test.T)
knn3 = KNeighborsClassifier(n_neighbors = 3)
knn3.fit(x_train.T, y_train.T)
y_head_knn = knn3.predict(x_test.T)
y_head_svm = svm.predict(x_test.T)
y_head_nb = nb.predict(x_test.T)
y_head_dtc = dtc.predict(x_test.T)
y_head_rf = rf.predict(x_test.T)
from sklearn.metrics import confusion_matrix

cm_lr = confusion_matrix(y_test,y_head_lr)
cm_knn = confusion_matrix(y_test,y_head_knn)
cm_svm = confusion_matrix(y_test,y_head_svm)
cm_nb = confusion_matrix(y_test,y_head_nb)
cm_dtc = confusion_matrix(y_test,y_head_dtc)
cm_rf = confusion_matrix(y_test,y_head_rf)
plt.figure(figsize=(24,12))

plt.suptitle("Confusion Matrixes",fontsize=24)
plt.subplots_adjust(wspace = 0.4, hspace= 0.4)

plt.subplot(2,3,1)
plt.title("Logistic Regression Confusion Matrix")
sns.heatmap(cm_lr,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.subplot(2,3,2)
plt.title("K Nearest Neighbors Confusion Matrix")
sns.heatmap(cm_knn,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.subplot(2,3,3)
plt.title("Support Vector Machine Confusion Matrix")
sns.heatmap(cm_svm,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.subplot(2,3,4)
```

```
plt.title("Naive Bayes Confusion Matrix")
sns.heatmap(cm_nb,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.subplot(2,3,5)
plt.title("Decision Tree Classifier Confusion Matrix")
sns.heatmap(cm_dtc,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.subplot(2,3,6)
plt.title("Random Forest Confusion Matrix")
sns.heatmap(cm_rf,annot=True,cmap="Blues",fmt="d",cbar=False,
annot_kws={"size": 24})

plt.show()
```

3- Don't forget to fine-tune the model you obtained from Exercise 2 using the techniques you learned in this chapter. What is the best AUC it achieves?

KNN Model's Accuracy is 88.52%