
# Ahsanullah University of Science and Technology

**Department of Computer Science and Engineering**

**Course No.** : CSE4108
**Course Name** : Artificial Intelligence Lab

**Assignment No.** : 04

## Submitted By:

Name : Faiza Anan Noor
ID No. : 17 01 04 093
Session : Spring - 2020
Section : B (B2)

# Questions:

1) Implement Linear Regression without using Scikit-learn.

2) Implement Logistic Regression from scratch without using Scikit-learn. Run it against a dataset of choice (any dataset with over 1000 samples). Run the same algorithm with the help of Scikit learn. Compare your implementation with Scikit-learn's one.

3) Make a dataset by yourself which should have enough samples and attributes and write documentation of it. Do classification or regression on it. If you want to do a classification task, implement at least five models. If you want to do regression, similarly at least five models need to be implemented. For each model get at least three performance metric scores. Implementation of cross validation is a must. (Name your dataset as Dataset_StudentId and upload here: shorturl.at/zBFH8)

# Ans to Qno 1:

## Code in Python:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15.0,10.0)
#Reading Data
data=pd.read_csv('headbrain.csv')
print(data.shape)
data.head()

x=data['Head Size(cm^3)'].values
y=data['Brain Weight(grams)'].values
mean_x=np.mean(x)
mean_y=np.mean(y)
#Total number of values
m=len(x)
#using the formula to calculate b1 & b0
numer=0
```

```
denom=0
for i in range(m):
    numer+=(x[i]-mean_x)*(y[i]-mean_y)
    denom+=(x[i]-mean_x)**2
b1=numer/denom
b0=mean_y-(b1*mean_x)
#print coefficient
print(b1,b0)
#plotting values and regression line
max_x=np.max(x)+100
min_x=np.min(x)-100
#calculating x and y
x=np.linspace(min_x,max_x,1000)
y=b0+b1*x
#ploting line
plt.plot(x,y,color='blue',label='Regression line')
plt.scatter(x_test,y_test,color='red',label='Scatter plot')
plt.xlabel('Head size in cm3')
plt.ylabel('Brain weight in grams')
plt.show()
```

# Explanation:

**Regression:**

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent and independent variable

## Least Square Method – Finding the best fit line

Least squares is a statistical method used to determine the best fit line or the regression line by minimizing the sum of squares created by a mathematical function. The “square” here refers to squaring the distance between a data point and the regression line. The line with the minimum value of the sum of square is the best-fit regression line.

Regression Line, y = mx+c where,

y = Dependent Variable

x= Independent Variable ; c = y-Intercept

$$m = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

**Step 1: Importing Necessary Libraries**

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (15.0, 10.0)

# Reading Data
data = pd.read_csv('headbrain.csv')
print(data.shape)
data.head()


# Collecting X and Y
X = data['Head Size(cm^3)'].values
Y = data['Brain Weight(grams)'].values
# Importing Necessary Libraries
```

**Step 2: In order to find the value of m and c, you first need to calculate the mean of X and Y**

```
# Mean X and Y
mean_x = np.mean(X)
mean_y = np.mean(Y)

# Total number of values
n = len(X)

# Using the formula to calculate m and c
numer = 0
denom = 0
for i in range(n):
numer += (X[i] - mean_x) * (Y[i] - mean_y)
denom += (X[i] - mean_x) ** 2
m = numer / denom
c = mean_y - (m * mean_x)
# Print coefficients
print(m, c)
```

The value of m and c from above will be added to this equation

$$BrainWeight = c + m * HeadSize$$

**Step 3: Plotting Linear Regression Line**

Now that we have the equation of the line. So for each actual value of x, we will find the predicted values of y. Once we get the points we can plot them over and create the Linear Regression Line,

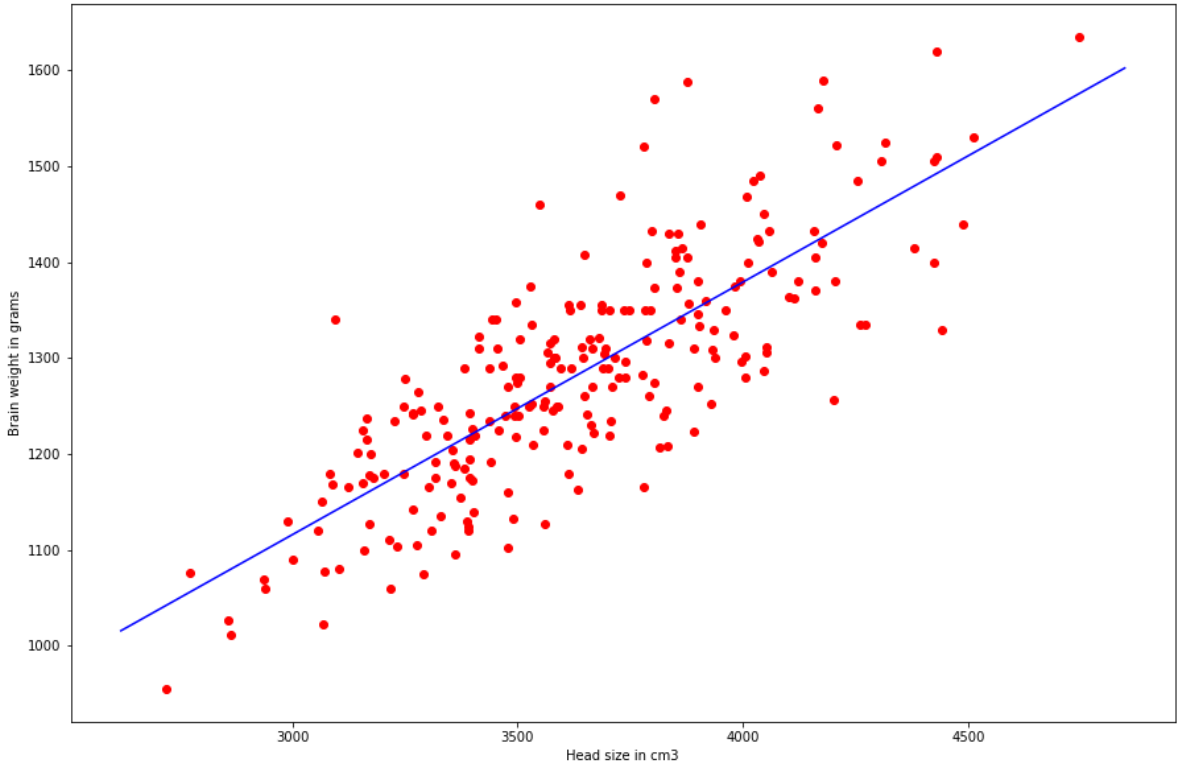
```
# Plotting Values and Regression Line
max_x = np.max(X) + 100
min_x = np.min(X) - 100
# Calculating line values x and y
x = np.linspace(min_x, max_x, 1000)
y = c + m * x

# Ploting Line
plt.plot(x, y, color='#52b920', label='Regression Line')
# Ploting Scatter Points
plt.scatter(X, Y, c='#ef4423', label='Scatter Plot')

plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```

# Output:

```
(237, 4)
0.26342933948939945 325.57342104944223
```



# Ans to Qno 2:

## Code in Python:

```
#!/usr/bin/env python
# coding: utf-8

# # Logistic regression with and without SKLEARN

# In[13]:


# Importing libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split


# to compare our model's accuracy with sklearn model
from sklearn.linear_model import LogisticRegression
# Logistic Regression
class LogitRegression() :
    def __init__( self, learning_rate, iterations ) :
        self.learning_rate = learning_rate
        self.iterations = iterations

    # Function for model training
    def fit( self, X, Y ) :
        # no_of_training_examples, no_of_features
        self.m, self.n = X.shape

        # weight initialization
        self.W = np.zeros( self.n )
        #print(self.W)

        self.b = 0
        self.X = X
        self.Y = Y

        # gradient descent learning
```

```
        for i in range( self.iterations ) :
            self.update_weights()
        return self

    # Helper function to update weights in gradient descent

    def update_weights( self ) :
        A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )

        tmp = ( A - self.Y.T )
        #print(self.Y)

        tmp = np.reshape( tmp, self.m )

        dW = np.dot( self.X.T, tmp ) / self.m
        db = np.sum( tmp ) / self.m

        # update weights
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db

        return self


    def predict( self, X ) :
        Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )
        Y = np.where( Z > 0.5, 1, 0 )
       # print(Y)
        return Y



# In[14]:



df = pd.read_csv( "framingham.csv" )
X = df.iloc[:,:-1].values
Y = df.iloc[:,-1:].values
X_train, X_test, Y_train, Y_test = train_test_split(
X, Y, test_size = 0.3, random_state = 12 )

df.head()
```

# In[15]:

```
model = LogitRegression( learning_rate = 0.01, iterations = 100)

model.fit( X_train, Y_train )
model1 = LogisticRegression()
model1.fit( X_train, Y_train)

# Prediction on test set
Y_pred = model.predict( X_test )
Y_pred1 = model1.predict( X_test )

import sklearn.metrics as metrics


print( "Accuracy on test set by our model   :  ",metrics.accuracy_score(Y_test, Y_pred))
print( "Accuracy on test set by sklearn model   :  ",metrics.accuracy_score(Y_test, Y_pred1))
```

# In[ ]:

# In[ ]:

# In[ ]:

# In[ ]:

## Explanation:

Logistic Regression is a supervised learning algorithm that is used when the target variable is categorical. Hypothetical function h(x) of linear regression predicts unbounded values. But in the case of Logistic Regression, where the target variable is categorical we have to strict the range of predicted values. Consider a classification problem, where we need to classify whether an email is a spam or not. So, the hypothetical function of linear regression could not be used here to predict as it predicts unbound values, but we have to predict either 0 or 1.

To do, so we apply the sigmoid activation function on the hypothetical function of linear regression. So the resultant hypothetical function for logistic regression is given below :

**h( x ) = sigmoid( wx + b )**

**Here, w is the weight vector.**
**x is the feature vector.**
**b is the bias.**

**sigmoid( z ) = 1 / ( 1 + e( - z ) )**

The cost function of linear regression ( or mean square error ) can't be used in logistic regression because it is a non-convex function of weights. Optimizing algorithms like i.e gradient descent only converge convex function into a global minimum

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \text{(simultaneously update } \theta_j \text{ for all } j).$$

With each step of gradient descent, your parameters $\theta_j$ come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

**Comparsion between sklearn implementation and implementation without sklearn:**

Sklearn implementation:
For the implementation using sklearn, we imported the built-in module for the implementation of Logistic Regression in this way:
from sklearn.linear_model import LogisticRegression

Then we instantiated it and then fit our model and then made predictions.

```
model1 = LogisticRegression()
model1.fit( X_train, Y_train)
Y_pred1 = model1.predict( X_test )
```

Implementation from scratch:
To do, so we apply the sigmoid activation function on the hypothetical function of linear regression.

```
A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )
```

The cost function of linear regression ( or mean square error ) can't be used in logistic regression because it is a non-convex function of weights. Optimizing algorithms like i.e gradient descent only converge convex function into a global minimum
For this purpose we created our own class called LogitRegression where the constructor or def __init__ takes in learning rate and iterations. Then we try to update our weights each time and also try to minimize the cost using gradient descent. Below are code fragments to update weights and make predictions.

```
def update_weights( self ) :
        A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) )
        tmp = ( A - self.Y.T )
        #print(self.Y)
        tmp = np.reshape( tmp, self.m )

        dW = np.dot( self.X.T, tmp ) / self.m
        db = np.sum( tmp ) / self.m

        # update weights
        self.W = self.W - self.learning_rate * dW
        self.b = self.b - self.learning_rate * db
        return self
    def predict( self, X ) :
        Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )
        Y = np.where( Z > 0.5, 1, 0 )
       # print(Y)
        return Y
```

# Output:

## Some samples from the dataset:

:

| | male | age | education | currentSmoker | cigsPerDay | BPMeds | prevalentStroke | prevalentHyp | diabetes | totChol | sysBP | diaBP | BMI | heartRate | glucose | Te |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 39 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 195 | 106.0 | 70.0 | 26.97 | 80 | 77 | |
| **1** | 0 | 46 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 250 | 121.0 | 81.0 | 28.73 | 95 | 76 | |
| **2** | 1 | 48 | 1 | 1 | 20 | 0 | 0 | 0 | 0 | 245 | 127.5 | 80.0 | 25.34 | 75 | 70 | |
| **3** | 0 | 61 | 3 | 1 | 30 | 0 | 0 | 1 | 0 | 225 | 150.0 | 95.0 | 28.58 | 65 | 103 | |
| **4** | 0 | 46 | 3 | 1 | 23 | 0 | 0 | 0 | 0 | 285 | 130.0 | 84.0 | 23.10 | 85 | 85 | |

## Performance metrics from the sklearn model and our model:

```
print( "Accuracy on test set by our model   :  ",metrics.accuracy_score(Y_test, Y_pred))
print( "Accuracy on test set by sklearn model   :  ",metrics.accuracy_score(Y_test, Y_pred1))
```

```
Accuracy on test set by our model   :   0.8494623655913979
Accuracy on test set by sklearn model   :   0.8476702508960573
```

# Ans to Qno 3:

## Code in Python:

```
#!/usr/bin/env python
# coding: utf-8

# # Classification model 1 (KNN)

# In[1]:

import numpy as np
import pandas as pd

#Loading the PlayTennis data
df = pd.read_csv("depression.csv")

df


# In[2]:


from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

df['physical condition'] = Le.fit_transform(df['physical condition'])
df['mental condition'] = Le.fit_transform(df['mental condition'])
df['marital status'] = Le.fit_transform(df['marital status'])
df['unemployment'] = Le.fit_transform(df['unemployment'])
df['depression'] = Le.fit_transform(df['depression'])
df


# In[3]:


y = df['depression']
X = df.drop(['depression'],axis=1)
```

# In[4]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

# In[5]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

# In[6]:

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

# In[7]:

```
import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))
```

# In[8]:

```
#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
import numpy as np
#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)
```

```
#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

# In[ ]:

# # Decision Tree

# In[23]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

# In[24]:

```
y_pred = classifier.predict(X_test)
```

# In[25]:

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

# In[26]:

```
import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))


# In[27]:


#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
import numpy as np

cvx = DecisionTreeClassifier()
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))


# # Naive Bayes

# In[14]:


y = df['depression']
X = df.drop(['depression'],axis=1)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()
```

```
#Train the model using the training sets
gnb.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gnb.predict(X_test)


# In[15]:


from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))

import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))

#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
import numpy as np

cvx = GaussianNB()
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))


# In[ ]:




# # SVM

# In[16]:
```

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)


# In[17]:


from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))


# In[18]:


import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))


# In[19]:


from sklearn.model_selection import cross_val_score
import numpy as np

cvx =svm.SVC(kernel='linear')
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

# # Random Forest

# In[20]:

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

# In[21]:

```
import sklearn.metrics as metrics

print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
```

# In[22]:

```
from sklearn.model_selection import cross_val_score
import numpy as np

cvx =RandomForestClassifier(n_estimators=100)
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

# In[ ]:

# Explanation:

### Step 1: Importing dependencies for opening dataset matrix manipulation.

```
import numpy as np
import pandas as pd

#Loading the PlayTennis data
df = pd.read_csv("depression.csv")

df
```

### Step 2: Separating the feature vectors and target vectors

```
y = df['depression']
X = df.drop(['depression'],axis=1)
```

### Step 3:Dividing the datset into training and testing set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

### Step4: Implementation of classification models

### Implementation of KNN:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

### Implementation of Decision Tree:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

**Implementation of Naive Bayes:**

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gnb.predict(X_test)
```

**Implementation of SVM:**

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

**Implementation of Random Forest:**

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

**Step 5: Finding out Performance metrics and cross validation:**

## Finding out confusion matrix:

**Basic concept:**

*Confusion Matrix*

- a performance measurement for machine learning classification problem
- output can be two or more classes
- a table with **four** different combinations of **predicted** and **actual** values

| | | Actual Values: +ve (1) | Actual Values: -ve (0) |
|---|---|---|---|
| Predicted Values | +ve (1) | **TP** | **FP** |
| | -ve (0) | **FN** | **TN** |

**Implementation in our code:**

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

**Finding out Accuracy,f1 score, precision , recall:**

**Basic Concept:**

**Accuracy:**

Defined as the number of correct predictions made as a ratio of all predictions made

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

**Precision**:
Defined as the number of correct documents returned by our ML model.
Precision is a measure that tells us what proportion of patients that we diagnosed as having deprssion, actually had depression.

$$Precision = \frac{TP}{TP + FP}$$

**F1 score:**

- Harmonic mean of precision and recall
- F1 score is the weighted average of the precision and recall

- difficult to compare two models with low precision and high recall or vice versa.
- to make them comparable, we use F-Score.
- F-score helps to measure Recall and Precision at the same time.

$$F - measure = \frac{2 * precision * recall}{precision + recall}$$

**Recall:**
Defined as the number of positives returned by our ML model
Ex. - a measure that tells us what proportion of patients that actually had depression was diagnosed by the algorithm as having depression.

$$Recall = \frac{TP}{TP + FN}$$

**Implementation in our code**:

```
import sklearn.metrics as metrics
print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred,
labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))
```

**Doing cross validation:**

**Basic Idea:**
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation
Implementation in our code:

The **goal of cross-validation** is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem)

**Implementation in code:**

```
from sklearn.model_selection import cross_val_score
import numpy as np

cvx =svm.SVC(kernel='linear')
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

# Output:

### 17 rows from our dataset:

| | physical condition | mental condition | marital status | unemployment | depression |
|---|---|---|---|---|---|
| 0 | normal | normal | married | yes | no |
| 1 | normal | normal | married | no | no |
| 2 | fatigue | normal | married | yes | yes |
| 3 | heart disease | guilt | married | yes | yes |
| 4 | heart disease | anxiety | unmarried | yes | yes |
| 5 | heart disease | anxiety | unmarried | no | no |
| 6 | fatigue | anxiety | unmarried | no | yes |
| 7 | normal | guilt | married | yes | no |
| 8 | normal | anxiety | unmarried | yes | yes |
| 9 | heart disease | guilt | unmarried | yes | yes |
| 10 | normal | guilt | unmarried | no | yes |
| 11 | fatigue | guilt | married | no | yes |
| 12 | fatigue | normal | unmarried | yes | yes |
| 13 | heart disease | guilt | married | no | no |
| 14 | fatigue | guilt | unmarried | yes | yes |
| 15 | heart disease | guilt | unmarried | yes | no |
| 16 | normal | normal | married | no | no |
| 17 | normal | guilt | unmarried | no | no |

**Our dataset converted to numeric form:**

| | physical condition | mental condition | marital status | unemployment | depression |
|---|---|---|---|---|---|
| **0** | 2 | 2 | 0 | 1 | 0 |
| **1** | 2 | 2 | 0 | 0 | 0 |
| **2** | 0 | 2 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 | 1 |
| **4** | 1 | 0 | 1 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 1 |
| 7 | 2 | 1 | 0 | 1 | 0 |
| 8 | 2 | 0 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 | 1 |
| **10** | 2 | 1 | 1 | 0 | 1 |
| **11** | 0 | 1 | 0 | 0 | 1 |
| **12** | 0 | 2 | 1 | 1 | 1 |
| **13** | 1 | 1 | 0 | 0 | 0 |
| **14** | 0 | 1 | 1 | 1 | 1 |
| **15** | 1 | 1 | 1 | 1 | 0 |
| **16** | 2 | 2 | 0 | 0 | 0 |
| **17** | 2 | 1 | 1 | 0 | 0 |

## Performance metrics and Cross validation on KNN:

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[1 1]
 [1 1]]
```

```
import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))
```

```
f1 score is  0.5
precision score is  0.5
accuracy is  0.5
recall score is  0.5
```

```
#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
import numpy as np
#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)
#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
[0.75       0.75       0.25       0.66666667 1.        ]
cv_scores mean:0.6833333333333333
```

## Performance Metrics and Cross Validation on Decision Tree:

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[1 2]
 [0 1]]
```

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[1 2]
 [0 1]]
```

```
import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))
```

```
f1 score is  0.5
precision score is  0.3333333333333333
accuracy is  0.5
recall score is  1.0
```

```
#CROSS VALIDATION
from sklearn.model_selection import cross_val_score
import numpy as np

cvx = DecisionTreeClassifier()
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
[0.75       0.5        0.5        0.66666667 0.66666667]
cv_scores mean:0.6166666666666666
```

## Performance metrics and Cross Validation on Naive Bayes:

```
[[1 2]
 [0 1]]
f1 score is  0.5
precision score is  0.3333333333333333
accuracy is  0.5
recall score is  1.0
[0.75       0.75       0.25       0.66666667 1.         ]
cv_scores mean:0.6833333333333333
```

## Performance metrics and cross validation on SVM:

```
from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[2 1]
 [0 1]]
```

```
import sklearn.metrics as metrics

print('f1 score is ',metrics.f1_score(y_test, y_pred, labels=np.unique(y_pred)))
print('precision score is ',metrics.precision_score(y_test, y_pred, labels=np.unique(y_pred)))
print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
print('recall score is ',metrics.recall_score(y_test, y_pred))
```

```
f1 score is  0.6666666666666666
precision score is  0.5
accuracy is  0.75
recall score is  1.0
```

```
from sklearn.model_selection import cross_val_score
import numpy as np

cvx =svm.SVC(kernel='linear')
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
[0.75       0.75       0.5        0.66666667 1.         ]
cv_scores mean:0.7333333333333333
```

**Performance metrics and Cross Validation on Random Forest:**

```
import sklearn.metrics as metrics

print('accuracy is ',metrics.accuracy_score(y_test, y_pred))
```

```
accuracy is  0.75
```

```
from sklearn.model_selection import cross_val_score
import numpy as np

cvx =RandomForestClassifier(n_estimators=100)
#train model with cv of 5
cv_scores = cross_val_score(cvx, X, y, cv=5)
#print each cv score (accuracy) and average them
print(cv_scores)
print("cv_scores mean:{}".format(np.mean(cv_scores)))
```

```
[0.75       0.75       0.25       0.66666667 0.66666667]
cv_scores mean:0.6166666666666666
```

**Dataset Informtation:**

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | physical co | mental co | marital sta | unemploy | depressio |
| 2 | normal | normal | married | yes | no |
| 3 | normal | normal | married | no | no |
| 4 | fatigue | normal | married | yes | yes |
| 5 | heart dise | guilt | married | yes | yes |
| 6 | heart dise | anxiety | unmarriec | yes | yes |
| 7 | heart dise | anxiety | unmarriec | no | no |
| 8 | fatigue | anxiety | unmarriec | no | yes |
| 9 | normal | guilt | married | yes | no |
| 10 | normal | anxiety | unmarriec | yes | yes |
| 11 | heart dise | guilt | unmarriec | yes | yes |
| 12 | normal | guilt | unmarriec | no | yes |
| 13 | fatigue | guilt | married | no | yes |
| 14 | fatigue | normal | unmarriec | yes | yes |
| 15 | heart dise | guilt | married | no | no |
| 16 | fatigue | guilt | unmarriec | yes | yes |
| 17 | heart dise | guilt | unmarriec | yes | no |
| 18 | normal | normal | unmarriec | no | no |
| 19 | normal | guilt | unmarriec | no | no |
| 20 | normal | normal | married | no | no |
| 21 | fatigue | normal | married | no | no |

**Sample number**: 20
**Feature number:** 4
**Feature description:**
The dataset consists of features called physical condition, mental condition, marital status, unemployment
**Class Label:**
The class label will come out either yes or no.
**Class Description:**
Here, the class label is either yes or no it was used in determining output based on the feature vectors running different kinds of classification algorithms.