# Lab Assignment 9
# Chaudhary Hamdan
# 1905387
# Date: 30-03-2022

From dataset ‘social ad’:
Calculate performance metric
1. Accuracy
2. Sensitivity/Recall
3. Specificity
4. F-score
5. Precision

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

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('Social_Network_Ads.csv')
df.head()
```

| | Age | EstimatedSalary | Purchased |
|---|---|---|---|
| 0 | 19 | 19000 | 0 |
| 1 | 35 | 20000 | 0 |
| 2 | 26 | 43000 | 0 |
| 3 | 27 | 57000 | 0 |
| 4 | 19 | 76000 | 0 |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              400 non-null    int64
 1   EstimatedSalary  400 non-null    int64
 2   Purchased        400 non-null    int64
dtypes: int64(3)
memory usage: 9.5 KB
```

```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

| | Age | EstimatedSalary | Purchased |
|---|---|---|---|
| Age | 1.000000 | 0.155238 | 0.622454 |
| EstimatedSalary | 0.155238 | 1.000000 | 0.362083 |
| Purchased | 0.622454 | 0.362083 | 1.000000 |

```
x_train, x_test, y_train, y_test = train_test_split(df.drop(columns =
['Purchased']), df['Purchased'], test_size = 0.2)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

```
((320, 2), (320,), (80, 2), (80,))
```

```
algos = []
accuracy = []
recall = []
precision = []
f1Score = []
specificity = []
```

Use Algorithm
1. k-NN

```
algo = "K Nearest Neighbour"
model = KNeighborsClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
K Nearest Neighbour
[[45  7]
 [12 16]]


Accuracy: 76.25
Recall: 57.14285714285714
Precision: 69.56521739130434
F score: 62.745098039215684
Specificity: 0.8653846153846154
```

2. Logistic regression

```
algo = "Logistic Regression"
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
Logistic Regression
[[52  0]
 [28  0]]


Accuracy: 65.0
Recall: 0.0
Precision: 0.0
F score: 0.0
Specificity: 1.0
```

3. SVM

```
algo = "SVM"
model = SVC(kernel='rbf')
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
SVM
[[51  1]
 [17 11]]


Accuracy: 77.5
Recall: 39.285714285714285
Precision: 91.66666666666666
F score: 55.00000000000001
Specificity: 0.9807692307692307
```

4. Decision Tree

```
algo = "Decision Tree"
model = DecisionTreeClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
Decision Tree
[[46  6]
 [ 4 24]]


Accuracy: 87.5
Recall: 85.71428571428571
Precision: 80.0
F score: 82.75862068965519
Specificity: 0.8846153846153846
```

5. Naive Bayes

```
algo = "Naive Bayes"
model = GaussianNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
Naive Bayes
[[49  3]
 [ 5 23]]


Accuracy: 90.0
Recall: 82.14285714285714
Precision: 88.46153846153845
F score: 85.18518518518519
Specificity: 0.9423076923076923
```

6. Adaboost

```
algo = "Adaboost"
model = AdaBoostClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
Adaboost
[[48  4]
 [ 2 26]]


Accuracy: 92.5
Recall: 92.85714285714286
Precision: 86.66666666666667
F score: 89.65517241379311
Specificity: 0.9230769230769231
```

7. Gradient Boost

```
algo = "GradientBoost"
model = GradientBoostingClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
GradientBoost
[[48  4]
 [ 1 27]]


Accuracy: 93.75
Recall: 96.42857142857143
Precision: 87.09677419354838
F score: 91.52542372881356
Specificity: 0.9230769230769231
```

8. Random Forest

```
algo = "Random forest"
model = RandomForestClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(algo)
print(confusion_matrix(y_test, y_pred), '\n\n')
acc = accuracy_score(y_test, y_pred) * 100
print('Accuracy:', acc)
rec = recall_score(y_test, y_pred) * 100
print('Recall:', rec)
pre = precision_score(y_test, y_pred) * 100
print('Precision:', pre)
f1s = f1_score(y_test, y_pred) * 100
print('F score:', f1s)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
spc = tn / (tn+fp)
print('Specificity:', spc)

algos.append(algo)
accuracy.append(acc)
recall.append(rec)
precision.append(pre)
f1Score.append(f1s)
specificity.append(spc)
```

```
Random forest
[[49  3]
 [ 1 27]]


Accuracy: 95.0
Recall: 96.42857142857143
Precision: 90.0
F score: 93.10344827586206
Specificity: 0.9423076923076923
```

Plot a bar graph and compare the accuracy obtained in each case.

```
mx = 0
for i in range(len(algos)):
   print(algos[i], ':    ', accuracy[i],',  ', recall[i],',  ', precision[i],',  ',
f1Score[i])
   if accuracy[i] > accuracy[mx]:
      mx = I
```

```
Logistic Regression :      65.0 ,   0.0 ,   0.0 ,   0.0
K Nearest Neighbour :      76.25 ,   57.14285714285714 ,   69.56521739130434 ,   62.745098039215684
Decision Tree :      87.5 ,   85.71428571428571 ,   80.0 ,   82.75862068965519
Naive Bayes :      90.0 ,   82.14285714285714 ,   88.46153846153845 ,   85.18518518518519
SVM :      77.5 ,   39.285714285714285 ,   91.66666666666666 ,   55.00000000000001
Adaboost :      92.5 ,   92.85714285714286 ,   86.66666666666667 ,   89.65517241379311
GradientBoost :      93.75 ,   96.42857142857143 ,   87.09677419354838 ,   91.52542372881356
Random forest :      95.0 ,   96.42857142857143 ,   90.0 ,   93.10344827586206
```

```
print('Maximum Accuracy : ', accuracy[i], 'of', algos[i], 'algorithm.')
```

```
Maximum Accuracy :  95.0 of Random forest algorithm.
```

```
plt.bar(algos, accuracy)
plt.xticks(rotation = 45)
plt.show()
```