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

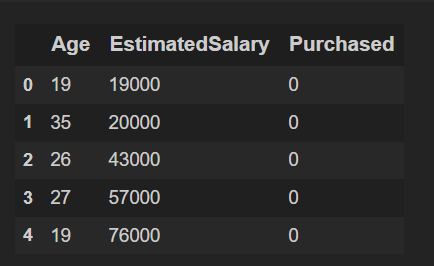
from sklearn.metrics import f1\_score

import warnings

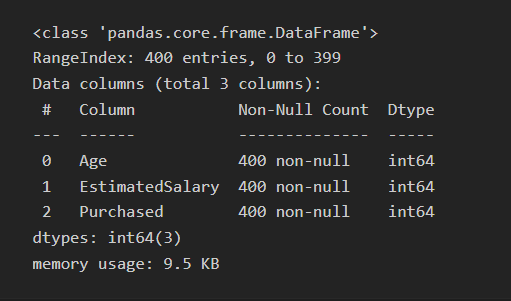
warnings.filterwarnings('ignore')

df = pd.read\_csv('Social\_Network\_Ads.csv')

df.head()



df.info()



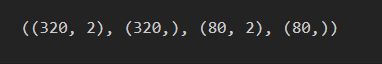
corr = df.corr()

corr.style.background\_gradient(cmap='coolwarm')



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



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)

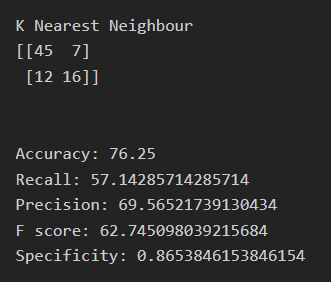
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

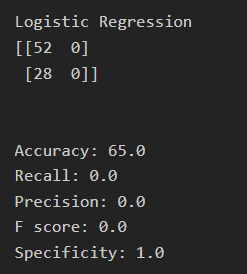
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

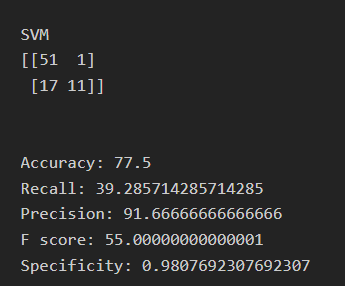
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

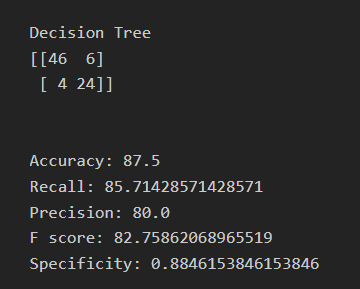
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

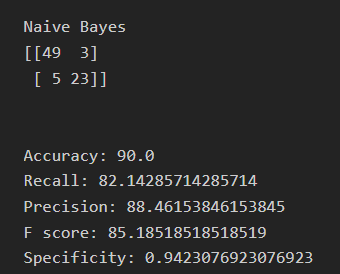
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

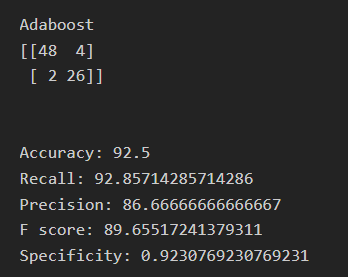
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

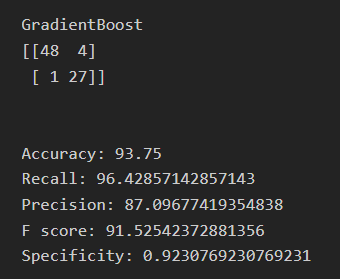
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



1. 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)

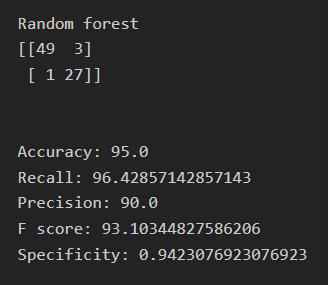
accuracy.append(acc)

recall.append(rec)

precision.append(pre)

f1Score.append(f1s)

specificity.append(spc)



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

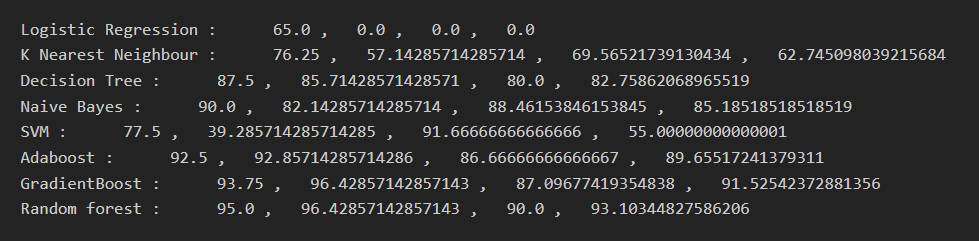
mx = 0

for i in range(len(algos)):

print(algos[i], ': ', accuracy[i],', ', recall[i],', ', precision[i],', ', f1Score[i])

if accuracy[i] > accuracy[mx]:

mx = I



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



plt.bar(algos, accuracy)

plt.xticks(rotation = 45)

plt.show()

