PROJECT 3: CLASSIFICATION

Objective
The goal of this project is to analyse the Titanic dataset to predict whether a given passenger survived.

Description of data
For more details on the dataset and the process of cleaning the data, see Project 1. Here is a summary. The data initially had 891 entries and 13 features. The target feature is Survived. In cleaning the data, we did the following:

1. Dropped PassengerId, Name, Ticket, Fare, Cabin
2. Encoded Pclass, Embarked, Sex as ordinal numeric variables, masking missing values of Embarked
3. Imputed to missing Age values the mean according to Sex

```
df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	2	1	22.0	1	0	7.2500	2
1	1	0	0	38.0	1	0	71.2833	0
2	1	2	0	26.0	0	0	7.9250	2
3	1	0	0	35.0	1	0	53.1000	2
4	0	2	1	35.0	0	0	8.0500	2

Classification models
We will explore the use of classification models on this data set. We use a train-test split with 20% test size, and use a standard scaler on the features.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
s = StandardScaler()
X_train = s.fit_transform(X_train)
X_test = s.transform(X_test)
```

We will consider three kinds of classification models: decision tree, random forest, and booster. For each classifier, we use F1 score to tune a hyperparameter.

First, we tune the max_depth for decision tree:

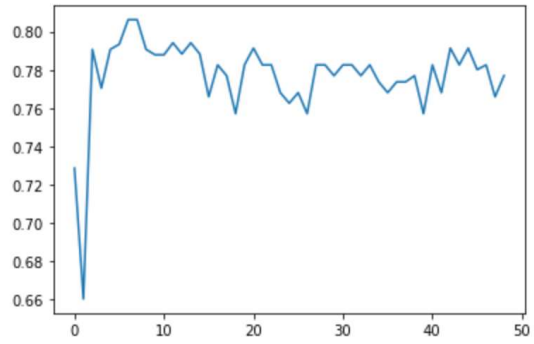
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score

scores = []

for i in range(1,50):
    model = DecisionTreeClassifier(max_depth=i)
    model.fit(X_train,y_train)
    scores.append(f1_score(y_test,model.predict(X_test)))

plt.plot(scores)
```

```
[<matplotlib.lines.Line2D at 0x25ad3edacd0>]
```



The best F1 score on the validation set is obtained when max_depth = 7, so we use this hyperparameter for our tree classifier.

```
treemodel = DecisionTreeClassifier(max_depth = 7)
treemodel.fit(X_train,y_train)
```

```
DecisionTreeClassifier(max_depth=7)
```

Next, we tune the n_estimators for Random Forest:

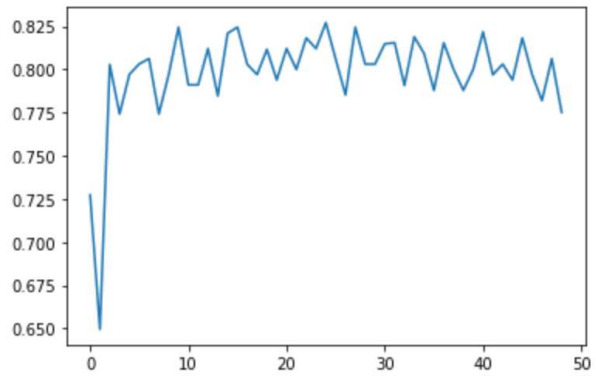
```
from sklearn.ensemble import RandomForestClassifier

scores = []

for i in range(1,50):
    model = RandomForestClassifier(n_estimators=i)
    model.fit(X_train,y_train)
    scores.append(f1_score(y_test,model.predict(X_test)))

plt.plot(scores)
```

```
[<matplotlib.lines.Line2D at 0x25ad1eb5d90>]
```



The best F1 score on the validation set is obtained when n_estimators = 24, so we use this hyperparameter for our random forest classifier.

```
rfmodel = RandomForestClassifier(n_estimators=24)
rfmodel.fit(X_train,y_train)
```

```
RandomForestClassifier(n_estimators=24)
```

Then, we tune the max_depth for Gradient Boosting classifier:

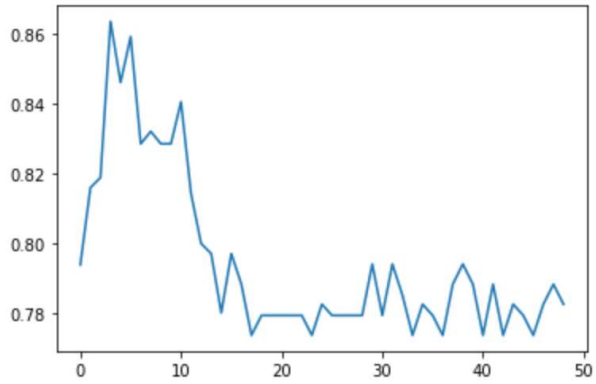
```
from sklearn.ensemble import GradientBoostingClassifier

scores = []

for i in range(1,50):
    model = GradientBoostingClassifier(max_depth=i)
    model.fit(X_train,y_train)
    scores.append(f1_score(y_test,model.predict(X_test)))

plt.plot(scores)
```

```
[<matplotlib.lines.Line2D at 0x25ad3ed7460>]
```



The best F1 score on the validation set is obtained when max_depth = 4, so we use this hyperparameter for our gradient boosting classifier.

```
gbmodel = GradientBoostingClassifier(max_depth=4)
gbmodel.fit(X_train,y_train)
```

```
GradientBoostingClassifier(max_depth=4)
```

All classifiers were trained on the same training set and validated against the same testing set.

Evaluation

To evaluate the optimal model of each kind, we compare their f1_scores, accuracy, and area under ROC curve

```
from sklearn.metrics import accuracy_score, roc_auc_score

classifiers = ['Decision tree', 'Random forest', 'Gradient boosting']
f1 = [f1_score(y_test,treemodel.predict(X_test)),
      f1_score(y_test,rfmodel.predict(X_test)),
      f1_score(y_test,gbmodel.predict(X_test))]
acc = [accuracy_score(y_test,treemodel.predict(X_test)),
       accuracy_score(y_test,rfmodel.predict(X_test)),
       accuracy_score(y_test,gbmodel.predict(X_test))]
roc = [roc_auc_score(y_test,treemodel.predict(X_test)),
       roc_auc_score(y_test,rfmodel.predict(X_test)),
       roc_auc_score(y_test,gbmodel.predict(X_test))]

scores = pd.DataFrame({'F1 score': f1, 'Accuracy': acc, 'Area under ROC': roc}, index=classifiers)
scores
```

	F1 score	Accuracy	Area under ROC
Decision tree	0.815385	0.865922	0.849165
Random forest	0.796992	0.849162	0.835652
Gradient boosting	0.848485	0.888268	0.875729

We observe that the gradient boosting classifier does best on all counts.