Entrée [1]:

```
# Badr TADJER | Naoufal ARRADI | Leo TRAN
# M1-LS1-APP
```

Predict survival on the Titanic

In this Lab, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy

Dataset

The dataset contains 891 observations of 12 variables:

- **PassengerId**: Unique ID for each passenger
- **Survived**: Survival (0 = No; 1 = Yes)
- **Pclass**: Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- **Name**: Name
- **Sex**: Sex
- **Age**: Age
- **Sibsp**: Number of Siblings/Spouses Aboard
- **Parch**: Number of Parents/Children Aboard
- **Ticket**: Ticket Number
- **Fare**: Passenger Fare
- **Cabin**: Cabin
- **Embarked** Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Entrée [2]:

```
# imports
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
```

Entrée [3]:

```
titanic = pd.read_csv("titanic.csv" )
titanic.drop('Cabin', axis=1, inplace=True) # Drop this column because it contains a
titanic["Age"].fillna(titanic["Age"].median(),inplace=True)
titanic["Embarked"].fillna("S", inplace = True)
print ('survival rate =', titanic.Survived.mean())
```

```
survival rate = 0.3838383838383838
```

Model training

Entrée [4]:

```
# Some of the columns don't have predictive power, so let's specify which ones are i
predictors = ["Pclass", "Sex", "Age", 'SibSp' ,'Parch', "Fare", "Embarked"]
# We need now to convert text columns in predictors to numerical ones
for col in predictors: # Loop through all columns in predictors
    if titanic[col].dtype == 'object':  # check if column's type is object (text)
        titanic[col] = pd.Categorical(titanic[col]).codes  # convert text to numeric

titanic.head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Emb
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	

Entrée [5]:

```
# Split the data into a training set and a testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(titanic[predictors], titanic['Su

from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=1)
clf.fit(X_train, y_train)
train_score = clf.score(X_train, y_train)
print ('train accuracy =', clf.score(X_train, y_train))

from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, titanic[predictors], titanic["Survived"], scoring = 'a
print('cross validation accuracy =', scores.mean())
```

```
train accuracy = 0.8073836276083467
cross validation accuracy = 0.7957428214731586
```

Decision Trees

Let's start with one single tree

Entrée [6]:

```
# import from: http://scikit-learn.org/stable/modules/generated/sklearn.tree.Decisic
# your code here
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
clf_dt = DecisionTreeClassifier(random_state = 1)

# your code here
clf_dt.fit(X_train, y_train)

print ('train accuracy =', clf_dt.score(X_train, y_train))
print ('test accuracy =', clf_dt.score(X_test, y_test))
```

```
train accuracy = 0.9887640449438202
test accuracy = 0.7574626865671642
```

Predictions are obtained in the same way of Logistic Regression

Entrée [7]:

```
y_pred = clf_dt.predict(X_test)
print (y_pred)
```

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

Entrée [8]:

```
y_prob = clf_dt.predict_proba(X_test)
print (y_prob)
```

```
[[0.   1.  ]
 [1.   0.  ]
 [0.   1.  ]
 [0.   1.  ]
 [0.   1.  ]
 [1.   0.  ]
 [1.   0.  ]
 [0.   1.  ]
 [1.   0.  ]
 [0.   1.  ]
 [1.   0.  ]
 [1.   0.  ]
 [1.   0.  ]
 [1.   0.  ]
 [0.25 0.75]
 [1.   0.  ]
 [1.   0.  ]
 [1.   0.  ]
 [1.   0.  ]
```

Let's play around with some of the decision tree's parameters

Entrée [9]:

```
# check the sklearn documentation and change the folowing parametrs: max_depth, min_
clf_dt = DecisionTreeClassifier(random_state=1, max_depth=8, min_samples_split=20, m
# your code here
clf_dt.fit(X_train, y_train)
print ('train accuracy =', clf_dt.score(X_train, y_train))

# Cross validation
scores_dt = cross_val_score(clf_dt, titanic[predictors], titanic["Survived"], scorin
print('cross validation accuracy =', scores_dt.mean())
```

```
train accuracy = 0.8828250401284109
cross validation accuracy = 0.813709120582512
```

Plot the decision tree

Set the max_depth parameter in the previous classifier to 3 and leave all the other ones to default values.

Entrée [10]:

```
from sklearn import tree
tree.export_graphviz(clf_dt, out_file='tree.dot')
# As a reminder, these are the predicting features in order
print (dict(zip(range(len(predictors)),predictors)))
```

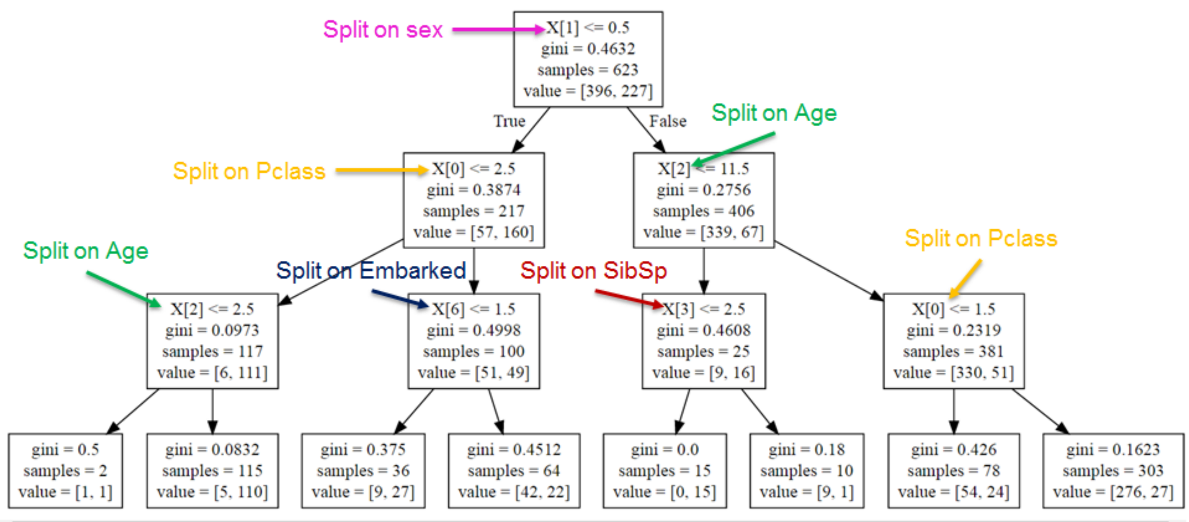
```
{0: 'Pclass', 1: 'Sex', 2: 'Age', 3: 'SibSp', 4: 'Parch', 5: 'Fare',
6: 'Embarked'}
```

The image should look like the following

Entrée [11]:

```
from IPython.display import Image
Image("DT.png")
```

Out[11]:



Predict the survival of a female, Pclass 1 or 2, above age 2.5

Entrée [12]:

```
passenger1=np.array([1, 0, 30, 0, 0, 0, 0]).reshape(1, -1)
print ('proba =', clf_dt.predict_proba(passenger1))
print ('class =', clf_dt.predict(passenger1))
```

```
proba = [[0. 1.]]
class = [1]
```

Predict the survival of a male, above age 11.5, Pclass 2 or 3

Entrée [13]:

```
passenger2=np.array([3, 1, 40, 0, 0, 0, 0]).reshape(1, -1)
print ('proba =', clf_dt.predict_proba(passenger2))
print ('class =', clf_dt.predict(passenger2))
```

```
proba = [[1. 0.]]
class = [0]
```

By looking at this decision tree, you can get a sense the relative importance between features. let's see which are the most important ones using the attribute: **feature_importances_**

Entrée [14]:

```
feat_imp = pd.DataFrame(clf_dt.feature_importances_, predictors, columns=['Importanc
feat_imp.sort_values('Importance', ascending=False)
```

Out[14]:

	Importance
Sex	0.507956
Pclass	0.156924
Age	0.137721
Fare	0.072380
Embarked	0.057184
SibSp	0.056693
Parch	0.011141

As expected, **Parch** and **Fare** are the least important ones because they were not used for splitting, while **Sex** is the most important one since it was used first for splitting.

Random Forest

A [Random Forest](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomF (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomF from sklearn.ensemble import RandomForestClassifier) is an ensemble of decision trees (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)

Entrée [15]:

```
# import from: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Ran
from sklearn.ensemble import RandomForestClassifier
# your code here
clf_rf = RandomForestClassifier(random_state = 1)  # by default, 10 trees are used
clf_rf.fit(X_train, y_train)
print ('train accuracy =', clf_rf.score(X_train, y_train))

# Cross validation
scores_rf = cross_val_score(clf_rf, titanic[predictors], titanic["Survived"], scorin
print('cross validation accuracy =', scores_rf.mean())
```

```
train accuracy = 0.9887640449438202
cross validation accuracy = 0.808122528403741
```

In the same way, you can print the feature importance of all the trees

Entrée [16]:

```
# your code here
feature_imp = pd.DataFrame(clf_rf.feature_importances_, predictors, columns=['Import
feature_imp.sort_values('Importance', ascending = False)
```

Out[16]:

	Importance
Fare	0.260215
Sex	0.260039
Age	0.252220
Pclass	0.088561
SibSp	0.053251
Parch	0.045476
Embarked	0.040238

Random forest, like decision trees have a lot of parameters to tune. Usually, performance does not change linearly with parameters. Let's take as an example, the accuracy as a function of number of trees (**n_estimators**)

Entrée [17]:

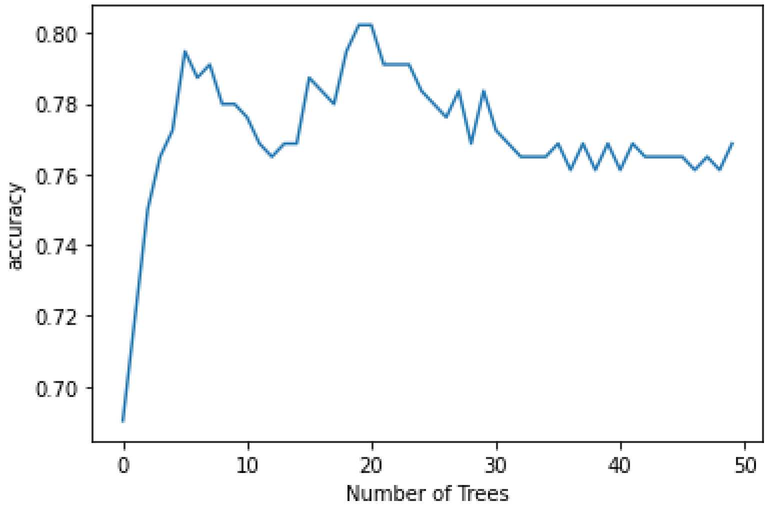
```
%matplotlib inline
import matplotlib.pyplot as plt

trees=range(50)
accuracy=np.zeros(50)
for idx in range(len(trees)):
    clf_rf=RandomForestClassifier(random_state=1, n_estimators=idx + 1)
    clf_rf.fit(X_train,y_train)
    accuracy[idx]=clf_rf.score(X_test, y_test)

plt.plot(trees, accuracy)
plt.ylabel('accuracy')
plt.xlabel('Number of Trees')
```

Out[17]:

```
Text(0.5, 0, 'Number of Trees')
```



In the following, try to tune manually the following parameters: **min_samples_leaf, min_samples_split, max_depth, n_estimators** in order to increase cross validation accuracy.

Entrée [18]:

```
clf_rf = RandomForestClassifier(random_state = 1, min_samples_leaf = 1, min_samples_
clf_rf.fit(X_train, y_train)
print('train accuracy =', clf_rf.score(X_train, y_train))

# Cross validation
scores_rf = cross_val_score(clf_rf, titanic[predictors], titanic["Survived"], scorin
print('cross validation accuracy = ', scores_rf.mean())
```

```
train accuracy = 0.9293739967897271
cross validation accuracy =  0.8305379448873266
```

This might be a difficult job to do manually. In other way is to search automatically the best combination of different ranges for these parameters. This is done using **Grid Search**

Grid Search

Entrée [19]:

```
# Documentation: http://scikit-learn.org/stable/modules/generated/sklearn.model_sele
from sklearn.model_selection import GridSearchCV
# your code here
params = {'min_samples_leaf':list(range(1,5)),'min_samples_split':list(range(2,10,2)
          'n_estimators':list(range(10,50,10))}
clf_rf2 = RandomForestClassifier(random_state=1)
clf_gs = GridSearchCV(clf_rf2, params, scoring = 'accuracy',cv=5)
clf_gs.fit(titanic[predictors], titanic["Survived"])
```

Out[19]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'min_samples_leaf': [1, 2, 3, 4],
                         'min_samples_split': [2, 4, 6, 8],
                         'n_estimators': [10, 20, 30, 40]},
             scoring='accuracy')
```

Print the best score

Entrée [20]:

```
# your code here
print('Best score : ', clf_gs.best_score_)
```

```
Best score :  0.8327976900382902
```

Print the best parameters

Entrée [21]:

```
# your code here
print('Best score : ', clf_gs.best_params_)
```

```
Best score :  {'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estim
ators': 30}
```

Let's use these best parameters and check whether they achieve really the above cv accuracy

Entrée [22]:

```
clf_rf3 = RandomForestClassifier(random_state=1, min_samples_leaf = 3, min_samples_s
clf_rf3.fit(X_train, y_train)
print ('train accuracy =', clf_rf3.score(X_train, y_train))

scores_rf3 = cross_val_score(clf_rf3, titanic[predictors], titanic["Survived"], scor
print('cross validation accuracy =',scores_rf3.mean())
```

```
train accuracy = 0.9036918138041734
cross validation accuracy = 0.8327976900382902
```

As you can see, grid search allows you to find the best model parameters to improve your accuracy. Now, we can see the most important features of this last classifier

Entrée [23]:

```
feat_imp = pd.DataFrame(clf_rf3.feature_importances_, predictors, columns=['Importar
feat_imp.sort_values('Importance', ascending=False)
```

Out[23]:

	Importance
Sex	0.368993
Fare	0.215793
Age	0.169397
Pclass	0.112599
SibSp	0.056356
Parch	0.042660
Embarked	0.034202