▾ 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)

```
import os
from google.colab import drive
drive.mount('/content/drive', force_remount=False)
```

```
    Mounted at /content/drive
```

```
#cd drive/MyDrive/Machine\ Learning\ Labs/Week4
```

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

```python
titanic = pd.read_csv("titanic.csv" )
titanic.drop('Cabin', axis=1, inplace=True) # Drop this column because it contai
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

```python
# Some of the columns don't have predictive power, so let's specify which ones a
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 num

titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282
				Futrelle, Mrs. Jacques					

```
# 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

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=
print('cross validation accuracy =', scores.mean())
```

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

Decision Trees

Let's start with one single tree

```
# import from: http://scikit-learn.org/stable/modules/generated/sklearn.tree.Dec
from sklearn.tree import DecisionTreeClassifier

clf_dt = DecisionTreeClassifier(random_state=1)
clf_dt.fit(X_train, y_train)

train_score = clf_dt.score(X_train, y_train)
test_score = clf_dt.score(X_test, y_test)

print ('train accuracy =', train_score)
print ('test accuracy =', test_score)
```

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

Predictions are obtained in the same way of Logistic Regression

```
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 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 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]
```

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

```
[0.5  0.5 ]
[0.   1.  ]
[1.   0.  ]
[0.8  0.2 ]
[0.   1.  ]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[0.   1.  ]
[0.   1.  ]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[0.   1.  ]
[0.   1.  ]
[1.   0.  ]
[1.   0.  ]
[0.   1.  ]
[0.   1.  ]
[0.   1.  ]
[1.   0.  ]
[0.   1.  ]
[1.   0.  ]
[0.   1.  ]
[0.5  0.5 ]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[0.25 0.75]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[1.   0.  ]
[0.   1.  ]
[1.   0.  ]
[0.   1.  ]
```

```
     [0.   1.  ]
     [0.   1.  ]
     [1.   0.  ]
     [0.   1.  ]
     [1.   0.  ]
     [1.   0.  ]
     [0.   1.  ]
     [0.   1.  ]
     [0.   1.  ]
     [0.   1.  ]
     [0.   1.  ]
     [0.   1.  ]
     [1.   0.  ]
     [1.   0.  ]
     [0.   1.  ]
     [1.   0.  ]
     [0.25 0.75]
     [1.   0.  ]
     [1.   0.  ]
     [0.5  0.5 ]
     [1.   0.  ]
     [0.   1.  ]]
```

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

```
# check the sklearn documentation and change the following parametrs: max_depth,
clf_dt = DecisionTreeClassifier(random_state=1, max_depth = 3, min_samples_split
clf_dt.fit(X_train, y_train)



train_score = clf_dt.score(X_train, y_train)
print ('train accuracy =', train_score)



# Cross validation
from sklearn.model_selection import cross_val_score

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

```
    train accuracy = 0.8571428571428571
    cross validation accuracy = 0.8091959073504487
```

▾ Plot the decision tree

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

```
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: 'Er
```

The image should look like the following

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


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

```
passenger1=np.array([1, 0, 2.5, 0, 0, 0, 0]).reshape(1, -1)
print ('proba =', clf_dt.predict_proba(passenger1))
print ('class =', clf_dt.predict(passenger1))
```

```
proba = [[0.5 0.5]]
class = [0]
```

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

```
passenger2=np.array([2, 0, 11.5, 0, 0, 0, 0]).reshape(1, -1)
print ('proba =', clf_dt.predict_proba(passenger2))
print ('class =', clf_dt.predict(passenger2))
```

```
proba = [[0.04347826 0.95652174]]
class = [1]
```

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_**

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

	Importance
Sex	0.611786
Pclass	0.189001
Age	0.084796
SibSp	0.064193
Embarked	0.050225
Parch	0.000000
Fare	0.000000

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.RandomForestClassifier from sklearn.ensemble import RandomForestClassifier) is an ensemble of decision trees

```
# import from: http://scikit-learn.org/stable/modules/generated/sklearn.ensemb
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(random_state=1)  # by default, 10 trees are us

clf_rf.fit(X_train, y_train)
train_score = clf_rf.score(X_train, y_train)
#test_score = clf_rf.score(X_test, y_test)

print ('train accuracy =', train_score)

# Cross validation
from sklearn.model_selection import cross_val_score

#scores_rf = # your code here
#print('cross validation accuracy =', score_rf)
scores_rf = cross_val_score(clf_rf, titanic[predictors], titanic["Survived"],
print('cross validation accuracy =', scores)
```

```
    train accuracy = 0.9887640449438202
    cross validation accuracy = [0.78888889 0.79775281 0.76404494 0.82022472 0
     0.78651685 0.78651685 0.80898876 0.84269663]
```

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

```
#·your·code·here
```

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**)

```
%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')
```

Text(0.5, 0, 'Number of Trees')
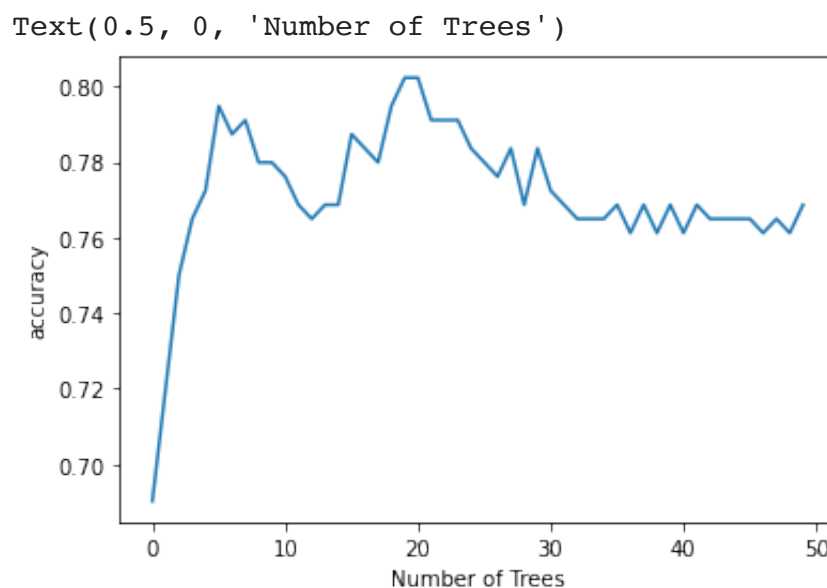
In the following, try to tune manually the following parameters: **min_samples_leaf, min_samples_split, max_depth, n_estimators** in order to increase cross validation accuracy.

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

# Cross validation
from sklearn.model_selection import cross_val_score
scores_rf = cross_val_score(clf_rf, titanic[predictors], titanic["Survived"], so
print('cross validation accuracy =', scores_rf.mean())
```

```
    train accuracy = 0.8764044943820225
    cross validation accuracy = 0.8260498399347185
```

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

```
# Documentation: http://scikit-learn.org/stable/modules/generated/sklearn.model_
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
# your code here
params = {'min_samples_leaf':list(range(1,5)),'min_samples_split':list(range(2,1
          '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"])
```

```
    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

```
print(clf_gs.best_score_)
```

```
    0.8327976900382902
```

Print the best parameters

```
print(clf_gs.best_params_)
```

```
    {'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 30}
```

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

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

clf_rf3 = RandomForestClassifier(random_state=1)
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"],
print('cross validation accuracy =',scores_rf3.mean())
```

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

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

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

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

✓ 0 s   terminée à 00:03   ● ✕