Acadgild machine learning Assiagnment 4

In [1]:

```
# Predicting Survival in the Titanic Data Set:

# We will be using a decision tree to make predictions about the Titanic data set from Kagg
# information on the Titanic passengers and can be used to predict whether a passenger surv

# Loading Data and modules
# import numpy as np
# import pandas as pd
# import seaborn as sb
# import matplotlib.pyplot as plt
# import sklearn
# from pandas import Series, DataFrame
# from pylab import rcParams
# from sklearn import preprocessing
# from sklearn.linear_model import LogisticRegression
# from sklearn.cross_validation import train_test_split
# from sklearn import metrics
# from sklearn.metrics import classification_report
# Url= https://raw.githubusercontent.com/BigDataGal/Python-for-Data-Science/master/titanic-
# titanic = pd.read_csv(url)
# titanic.columns = ['PassengerId','Survived','Pclass','Name','Sex','Age','SibSp','Parch','

# You use only Pclass, Sex, Age, SibSp (Siblings aboard), Parch (Parents/children aboard),
# passenger survived.
```

Load Libraries

In [1]:

```
# Core Libraries - Data manipulation and analysis
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Core Libraries - Machine Learning, Preprocessing and generating Performance Metrics
import sklearn
from sklearn import preprocessing
from sklearn import metrics

# Importing Classifiers - Modelling
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

## Importing train_test_split,cross_val_score,GridSearchCV,KFold - Validation and Optimizat
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit

# Importing Graphing and Visualization tools
import pydotplus
from IPython.display import Image
```

Load Data

In [2]:

```
# Loading the data into the dataframe
url= 'https://raw.githubusercontent.com/BigDataGal/Python-for-Data-Science/master/titanic-t
titanic = pd.read_csv(url)
```

In [3]:

```
titanic.head()
```

Out[3]:

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

Understand Dataset and Data

In [4]:

```
print(titanic.columns)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [5]:

```
print(titanic.shape)
```

```
(891, 12)
```

In [6]:

```
titanic.head()
```

Out[6]:

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

In [8]:

```
titanic.tail()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	Na
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	Na
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C1
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	Na

In [7]:

```
print(titanic.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

There are null values in the dataset which need to be removed or imputed

In [8]:

```
titanic.get_dtype_counts()
```

Out[8]:

```
float64    2
int64      5
object     5
dtype: int64
```

Data Cleaning

Find rows containing null values or zeros(that don't belong in the dataset) and then either impute or remove them

Checking for columns containing null values

In [9]:

```
titanic.columns.values
```

Out[9]:

```
array(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype=object)
```

The question points out: You use only Pclass, Sex, Age, SibSp (Siblings aboard), Parch (Parents/children aboard), and Fare to predict whether a passenger survived.

Therefore, removing other columns

In [10]:

```
titanic.drop(axis =1, columns= ["PassengerId","Name","Ticket","Cabin","Embarked"], inplace
titanic.head()
```

Out[10]:

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

All columns in the dataframe have non-null values except the Age

In [11]:

```
titanic.isna().sum()
```

Out[11]:

```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
dtype: int64
```

In [12]:

```
# Imputing the null values in Age column with the column's mean
titanic['Age'].fillna((titanic['Age'].mean()), inplace=True)
```

In [13]:

```
# Checking if all the values have been imputed
titanic.isna().sum()
```

Out[13]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
dtype: int64
```

In [14]:

```
titanic.shape
```

Out[14]:

```
(891, 7)
```

In [17]:

```
# Checking for rows with all values = 0, to remove or impute
titanic.loc[(titanic==0).all(axis=1)].shape
```

Out[17]:

```
(0, 7)
```

In [16]:

```
# Checking for rows with any values < 0, to remove or impute
titanic.loc[(titanic<0).any(axis=1)].shape
```

Out[16]:

```
(891, 7)
```

In [17]:

```
# Selecting categorical columns to feature engineer
cat_cols = titanic.select_dtypes(include='object').columns.values
cat_cols
```

Out[17]:

```
array(['Sex'], dtype=object)
```

In [18]:

```
# Encoding the Sex columns values into 0 and 1 and creating a new column with those values
titanic['Sex_Encoded'] = titanic['Sex'].replace({'female':0, 'male': 1})
```

In [19]:

```
# Dropping the Sex column
titanic.drop("Sex",axis =1, inplace = True)
```

In [20]:

```
titanic.head()
```

Out[20]:

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

Creating Input Vector and Output

In [21]:

```
X = titanic.drop("Survived", axis = 1)
```

In [22]:

```
Y = titanic.Survived
```

In [23]:

```
X.head()
```

Out[23]:

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

In [24]:

```
Y.head()
```

Out[24]:

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

Train Test Split

In [25]:

```
x_train,x_test,y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state =100)
```

Fitting the models and evaluating performance metrics

In [26]:

```
lr =  LogisticRegression()
lr.fit(x_train, y_train)
y_test_pred= lr.predict(x_test)
```

In [29]:

```
print("Logistic Regression Classifier - Base",
      "\n\t Accuracy:", metrics.accuracy_score(y_test, y_test_pred),
      "\n\t Precision:", metrics.precision_score(y_test, y_test_pred),
      "\n\t Recall:", metrics.recall_score(y_test, y_test_pred),
      "\n\t Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_test_pred),
      "\n\t Classification Report:",  metrics.classification_report(y_test, y_test_pred),"\
```

```
Logistic Regression Classifier - Base
         Accuracy: 0.7932960893854749
         Precision: 0.796875
         Recall: 0.68
         Confusion Matrix:
 [[91 13]
 [24 51]]
         Classification Report:              precision    recall  f1-score
support

          0       0.79      0.88      0.83       104
          1       0.80      0.68      0.73        75

avg / total       0.79      0.79      0.79       179
```

In [30]:

```
%%time
cart =  DecisionTreeClassifier()
cart.fit(x_train, y_train)
y_test_pred= cart.predict(x_test)
```

```
Wall time: 5.01 ms
```

In [31]:

```
print("Decision Tree Classifier - Base",
      "\n\t Accuracy:", metrics.accuracy_score(y_test, y_test_pred),
      "\n\t Precision:", metrics.precision_score(y_test, y_test_pred),
      "\n\t Recall:", metrics.recall_score(y_test, y_test_pred),
      "\n\t Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_test_pred),
      "\n\t Classification Report:\n",  metrics.classification_report(y_test, y_test_pred),
```

```
Decision Tree Classifier - Base
         Accuracy: 0.7653631284916201
         Precision: 0.72
         Recall: 0.72
         Confusion Matrix:
 [[83 21]
 [21 54]]
         Classification Report:
              precision    recall  f1-score   support

          0       0.80      0.80      0.80       104
          1       0.72      0.72      0.72        75

avg / total       0.77      0.77      0.77       179
```

In [33]:

```
cart.get_params
```

Out[33]:

```
<bound method BaseEstimator.get_params of DecisionTreeClassifier(class_weigh
t=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')>
```

Hyper parameter Optimization

In [35]:

```
# Initializing the classifier to optimize,
# Setting CV split and tree hyper-parameters for using in GridSearchCV optimization

cart_classifier =  DecisionTreeClassifier()

CV = ShuffleSplit(test_size=0.20, random_state=100)

param_grid = {
              'criterion':['gini','entropy'],
              'max_depth': [2, 3, 4, 5, 6, 7, 8, 9],
              'max_features':[2,3,4,5,6],
              'max_leaf_nodes': [2, 3, 4, 6, 9],
              'min_samples_leaf':[ 2, 3, 5, 7],
              'min_samples_split':[2, 3, 5],
              'random_state' : [10]
            }
```

In [36]:

```
rscv_grid = GridSearchCV(cart_classifier, param_grid=param_grid, verbose=1)
```

In [38]:

```
rscv_grid.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 4800 candidates, totalling 14400 fits

[Parallel(n_jobs=1)]: Done 14400 out of 14400 | elapsed:   45.1s finished
```

Out[38]:

```
GridSearchCV(cv=None, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [2, 3, 4,
5, 6, 7, 8, 9], 'max_features': [2, 3, 4, 5, 6], 'max_leaf_nodes': [2, 3, 4,
6, 9], 'min_samples_leaf': [2, 3, 5, 7], 'min_samples_split': [2, 3, 5], 'ra
ndom_state': [10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [39]:

```
# Showing the best hyper-parameters for the decision tree
rscv_grid.best_params_
```

Out[39]:

```
{'criterion': 'entropy',
 'max_depth': 2,
 'max_features': 6,
 'max_leaf_nodes': 9,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'random_state': 10}
```

In [40]:

```
# Using the best estimator created from the above hyper-parameters listed in the params_gri
model = rscv_grid.best_estimator_
model.fit(x_train, y_train)
```

Out[40]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=2,
            max_features=6, max_leaf_nodes=9, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=2,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=10, splitter='best')
```

In [41]:

```
y_pred_test = model.predict(x_test)
```

In [42]:

```
print("Decision Tree Classifier - Best Estimator",
      "\n\t Accuracy:", metrics.accuracy_score(y_test, y_pred_test),
      "\n\t Precision:", metrics.precision_score(y_test, y_pred_test),
      "\n\t Recall:", metrics.recall_score(y_test, y_pred_test),
      "\n\t Confusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred_test),
      "\n\t Classification Report:\n",  metrics.classification_report(y_test, y_pred_test),
```

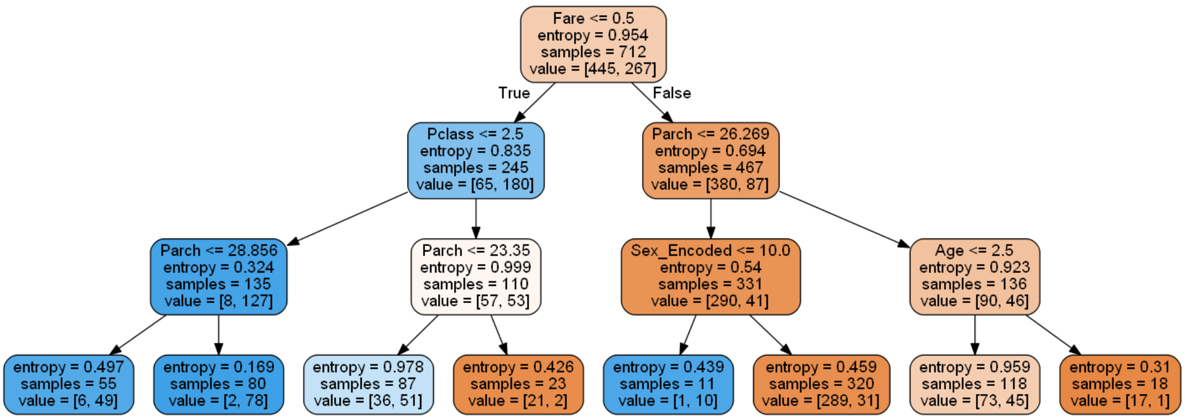
```
Decision Tree Classifier - Best Estimator
         Accuracy: 0.8100558659217877
         Precision: 0.8059701492537313
         Recall: 0.72
         Confusion Matrix:
 [[91 13]
 [21 54]]
         Classification Report:
              precision    recall  f1-score   support

          0       0.81      0.88      0.84       104
          1       0.81      0.72      0.76        75

avg / total       0.81      0.81      0.81       179
```

In [43]:

```
dot_data = tree.export_graphviz(rscv_grid.best_estimator_, out_file=None, filled=True, rour
                                feature_names=['Pclass', 'Sex_Encoded', 'Age', 'SibSp', 'Pa
graph = pydotplus.graph_from_dot_data(dot_data)
display(Image(graph.create_png()))
```



In [ ]: