Project Title : TITANIC

Machine Learning for Disaster Analysis & Management

(Predicting the Titanic Survival Rate)

Prepared for : ICT Academy at IITK

Prepared by : Paras Laul (paraslaul7@gmail.com)

Kkishita Jain (kkishitajain23@gmail.com)

Aditya Singh

INTRODUCTION

The sinking of the Titanic is one of the most historic shipwrecks of all time. The tragedy killed thousands, 1502 out of 2224 passengers, and led many wondering what could have been done better. One of the most important reason is that there was not enough lifeboats, and although there was probably quite amount of luck involved, there were some groups of people that were more likely to survive than others. In this paper a data analytical study will be conducted with the passenger's data from the Titanic dataset to find out about this survival likelihood. For the data analytical approach, we apply the theory of machine learning and predicting the survival of passengers by applying Machine Learning models.

Requirements

1. Jupyter Notebook
2. The following libraries are required to run the code in Jupyter Notebook

- Pandas
- Numpy
- Seaborn
- Matplotlib
- Sklearn

Procedure

All the Lifecycle In A Data Science Projects¶¶

- Data Analysis
- Feature Engineering
- Feature Selection
- Model Building
- Model Deployment

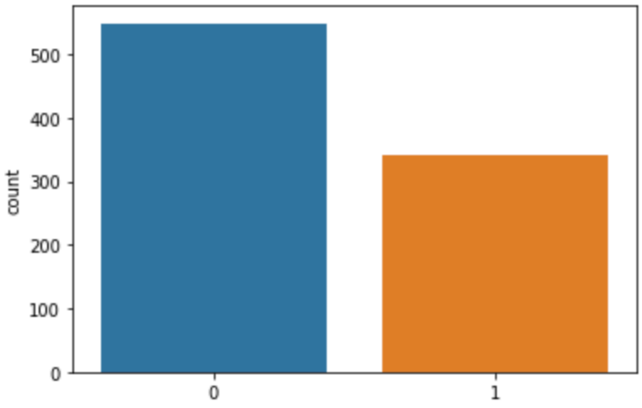
1. **Data Analysis**

In Data Analysis we will find the relationship of different features of titanic dataset with survived attribute and visualise these relationships using seaborn library.

- Find the count for passenger who survived. Here 0 stands for Survived and 1 stands for not Survived.

```
In [7]: df.Survived.value_counts() #unique values in target variable
Out[7]: 0    549
        1    342
        Name: Survived, dtype: int64
```

```
In [8]: sns.countplot(x='Survived',data=df)
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1cf899d0>
```



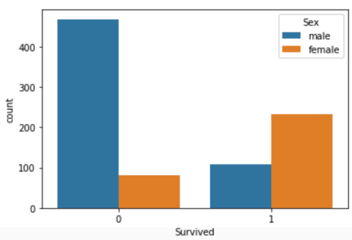
- Survived and Sex, Survived and PClass, Survived and Parch, Sex and Embarked

```
In [10]: df[df.Sex=='female'].Survived.value_counts()
Out[10]: 1    233
         0     81
         Name: Survived, dtype: int64
```

```
In [11]: df[['Sex','Survived']].groupby('Sex',as_index=False).mean()
Out[11]:
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

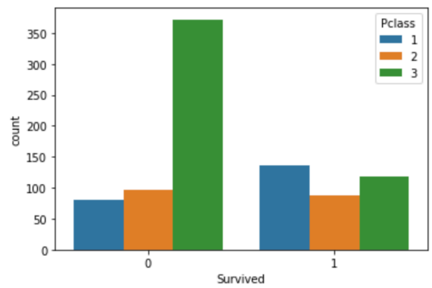
```
In [12]: sns.countplot(x='Survived',hue='Sex',data=df)
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d6fd950>
```




```
In [14]: df[['Pclass','Survived']].groupby('Pclass',as_index=False).mean()
Out[14]:
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

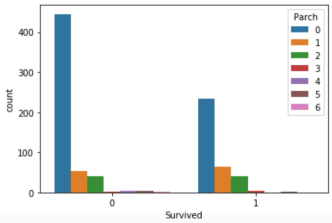
```
In [15]: sns.countplot(x='Survived',hue='Pclass',data=df)
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d788d50>
```



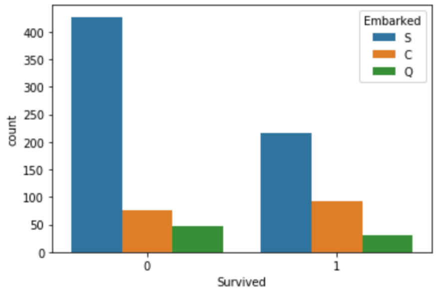
```
In [16]: df[['Parch','Survived']].groupby('Parch',as_index=False).mean()
Out[16]:
```

	Parch	Survived
0	0	0.343658
1	1	0.550847
2	2	0.500000
3	3	0.600000
4	4	0.000000
5	5	0.200000
6	6	0.000000

```
In [17]: sns.countplot(x='Survived',hue='Parch',data=df)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d82e450>
```




```
In [20]: sns.countplot(x='Survived',hue='Embarked',data=df)
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1c15f710>
```



2. **Data wrangling**

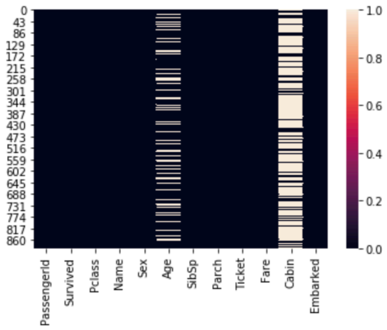
Processing the data and removing missing values in dataset by either substituting them with mean, mode, median or most frequent value.

```
In [30]: df.isnull().sum()
Out[30]: PassengerId      0
         Survived         0
         Pclass           0
         Name             0
         Sex              0
         Age            177
         SibSp            0
         Parch            0
         Ticket           0
         Fare             0
         Cabin          687
         Embarked         2
         dtype: int64
```

Age, Cabin and Embarked columns has 177,687 and 2 missing values respective

- Age has 177 missing values whereas Cabin and Embarked has 687 and 2 missing values respectively

```
In [31]: sns.heatmap(df.isnull()) #heatmap for missing values
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1d886550>
```



- Heat Map for missing values. Heat map shows that Cabin has maximum no. of missing values.

3. **Feature Engineering(Feature Scaling)**

We will be performing all the below steps in Feature Engineering

- Missing values
- Temporal variables
- Categorical variables: remove rare labels
- Standardise the values of the variables to the same range.

```
n [32]: #removing Cabin column
        df=df.drop(['Cabin'],axis=1)

n [33]: df.Embarked.value_counts()

ut[33]: S    644
        C    168
        Q     77
        Name: Embarked, dtype: int64

n [34]: df=df.fillna({'Age':df.Age.median(),
                      'Embarked':'S'}) #filling missing Age values with mean and missing Embarked val
```

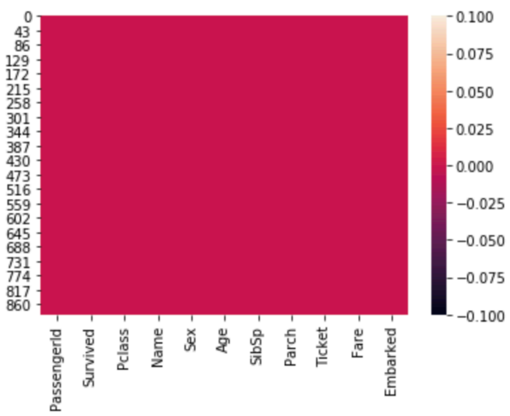
•Dropping the Cabin column with maximum no. of missing values.

•Replacing the missing values in Age column with median value of Age column.

- Replacing the missing values in Embarked column with most frequent values in Embarked column i.e ’S’.

```
In [36]: sns.heatmap(df.isnull()) #heatmap

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1da14dd0>
```



- Heat map after removing missing values.

- **Converting string values in categorical values**

Converting string values to categorical values

In [37]:
```python
df1 = pd.get_dummies(df[['Sex', 'Embarked']])
df1
```
Out[37]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	1	0	0	1
1	1	0	1	0	0
2	1	0	0	0	1
3	1	0	0	0	1
4	0	1	0	0	1
...	...	...	...	...	...
886	0	1	0	0	1
887	1	0	0	0	1
888	1	0	0	0	1
889	0	1	1	0	0
890	0	1	0	1	0

891 rows × 5 columns

- Converting string values of Sex and Embarked column to Categorical values and then concatenating old and new data to get new Dataframe.

In [38]:
```python
df = df[['Sex', 'Pclass', 'Fare','Embarked', 'Survived','Age']]
df
```
Out[38]:

	Sex	Pclass	Fare	Embarked	Survived	Age
0	male	3	7.2500	S	0	22.0
1	female	1	71.2833	C	1	38.0
2	female	3	7.9250	S	1	26.0
3	female	1	53.1000	S	1	35.0
4	male	3	8.0500	S	0	35.0
...	...	...	...	...	...	...
886	male	2	13.0000	S	0	27.0
887	female	1	30.0000	S	1	19.0
888	female	3	23.4500	S	0	28.0
889	male	1	30.0000	C	1	26.0
890	male	3	7.7500	Q	0	32.0

891 rows × 6 columns

In [40]:
```python
df2.drop(['Sex', 'Embarked'], axis=1, inplace=True)
df2
```
Out[40]:

	Pclass	Fare	Survived	Age	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	3	7.2500	0	22.0	0	1	0	0	1
1	1	71.2833	1	38.0	1	0	1	0	0
2	3	7.9250	1	26.0	1	0	0	0	1
3	1	53.1000	1	35.0	1	0	0	0	1
4	3	8.0500	0	35.0	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...
886	2	13.0000	0	27.0	0	1	0	0	1
887	1	30.0000	1	19.0	1	0	0	0	1
888	3	23.4500	0	28.0	1	0	0	0	1
889	1	30.0000	1	26.0	0	1	1	0	0
890	3	7.7500	0	32.0	0	1	0	1	0

891 rows × 9 columns

In [41]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2['Pclass'] = scaler.fit_transform(df2[['Pclass']])
df2['Fare'] = scaler.fit_transform(df2[['Fare']])
df2['Age'] = scaler.fit_transform(df2[['Age']])
df2
```
Out[41]:

	Pclass	Fare	Survived	Age	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1.0	0.014151	0	0.271174	0	1	0	0	1
1	0.0	0.139136	1	0.472229	1	0	1	0	0
2	1.0	0.015469	1	0.321438	1	0	0	0	1
3	0.0	0.103644	1	0.434531	1	0	0	0	1
4	1.0	0.015713	0	0.434531	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...
886	0.5	0.025374	0	0.334004	0	1	0	0	1
887	0.0	0.058556	1	0.233476	1	0	0	0	1
888	1.0	0.045771	0	0.346569	1	0	0	0	1
889	0.0	0.058556	1	0.321438	0	1	1	0	0
890	1.0	0.015127	0	0.396833	0	1	0	1	0

891 rows × 9 columns

- Scaling the values in Age , PClass and Fare columns between 0 and 1 using Min max scaler.

4. **Feature Selection**

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

Applying some feature selection methods to extract important features.

- K Best
- Model for Logistic Regression
- Recursive Feature Selection

Select K Best

Select features according to the k highest scores.

In [42]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X = df2.drop("Survived",axis=1)
y = df2["Survived"]

mdlsel = SelectKBest(chi2, k=5)
mdlsel.fit(X,y)
ix = mdlsel.get_support()
data2 = pd.DataFrame(mdlsel.transform(X), columns = X.columns.values[ix]) #
data2.head(n=5)
```

Out[42]:

	Pclass	Fare	Sex_female	Sex_male	Embarked_C
0	1.0	0.014151	0.0	1.0	0.0
1	0.0	0.139136	1.0	0.0	1.0
2	1.0	0.015469	1.0	0.0	0.0
3	0.0	0.103644	1.0	0.0	0.0
4	1.0	0.015713	0.0	1.0	0.0

Select From Model for Logistic Regression

In [43]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

X = df2.drop("Survived",axis=1)
y = df2["Survived"]

# Linear Model
linmdl = LogisticRegression()
linmdl.fit(X,y)
mdl = SelectFromModel(linmdl,prefit=True)
ix = mdl.get_support()
data3 = pd.DataFrame(mdl.transform(X), columns = X.columns.values[ix])
data3.head(n=5)
```

Out[43]:

	Pclass	Age	Sex_female	Sex_male
0	1.0	0.271174	0.0	1.0
1	0.0	0.472229	1.0	0.0
2	1.0	0.321438	1.0	0.0
3	0.0	0.434531	1.0	0.0
4	1.0	0.434531	0.0	1.0

Recursive Feature Selection

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features.

In [44]:
```
#last feature selection
from sklearn.feature_selection import RFE

mdl = RFE(linmdl,n_features_to_select=5)
mdl.fit(X,y)
ix = mdl.get_support()

data4 = pd.DataFrame(mdl.transform(X), columns = X.columns.values[ix])
data4.head(n=5)
```

Out[44]:

	Pclass	Age	Sex_female	Sex_male	Embarked_S
0	1.0	0.271174	0.0	1.0	1.0
1	0.0	0.472229	1.0	0.0	0.0
2	1.0	0.321438	1.0	0.0	1.0
3	0.0	0.434531	1.0	0.0	1.0
4	1.0	0.434531	0.0	1.0	1.0

```
In [66]: from sklearn.model_selection import train_test_split
```

```
In [67]: X=df2.drop(['Survived'],axis=1)
         y=df2['Survived']
         X
```

Out[67]:

	Pclass	Fare	Age	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1.0	0.014151	0.271174	0	1	0	0	1
1	0.0	0.139136	0.472229	1	0	1	0	0
2	1.0	0.015469	0.321438	1	0	0	0	1
3	0.0	0.103644	0.434531	1	0	0	0	1
4	1.0	0.015713	0.434531	0	1	0	0	1
...	...	...	...	...	...	...	...	...
886	0.5	0.025374	0.334004	0	1	0	0	1
887	0.0	0.058556	0.233476	1	0	0	0	1
888	1.0	0.045771	0.346569	1	0	0	0	1
889	0.0	0.058556	0.321438	0	1	1	0	0
890	1.0	0.015127	0.396833	0	1	0	1	0

891 rows × 8 columns

- Splitting the data into training and test.
- Declaring X as input variable and y as target variable.

```
In [69]: y

Out[69]: 0      0
         1      1
         2      1
         3      1
         4      0
               ..
         886    0
         887    1
         888    0
         889    1
         890    0
         Name: Survived, Length: 891, dtype: int64
```

```
In [48]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

5. **Model Building**

In model building we apply different machine learning algorithms and select models according to accuracy score.

MODEL BUILDING

Logistic Regression

In [120]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
from sklearn.linear_model import LogisticRegression
lg=LogisticRegression()
lg.fit(X_train,y_train)
y_pred=lg.predict(X_test)
```

In [121]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred) #accuracy score for Logistic Regression
```

Out[121]: 0.7877094972067039

Decision Tree

In [257]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
dc=DecisionTreeClassifier()
dc.fit(X_train,y_train)
y_p=dc.predict(X_test)
accuracy_score(y_test,y_p)
```

Out[257]: 0.8379888268156425

Random Forest

In [242]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
rf.fit(X_train,y_train)
y_pd=dc.predict(X_test)
accuracy_score(y_test,y_pd)
```

Out[242]: 0.8938547486033519

Support Vector Machine

In [258]:
```
from sklearn.svm import SVC
sv=SVC()
sv.fit(X_train,y_train)
y_pp=sv.predict(X_test)
accuracy_score(y_pp,y_test)
```

Out[258]: 0.8212290502793296

6. **Hyperparameter Tuning**

Tuning of Hyperparameters are important in order to achieve high accuracy of model. Using GridsearchCV from sklearn best hyperparameters are achieved.

Hyperparameter Tuning

In [167]:
```
# Create first pipeline for base without reducing features.
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
pipe = Pipeline([('classifier' , RandomForestClassifier())])
# pipe = Pipeline([('classifier', RandomForestClassifier())])

# Create param grid.

param_grid = [
    {'classifier' : [LogisticRegression()],
     'classifier__penalty' : ['l1', 'l2'],
    'classifier__C' : np.logspace(-4, 4, 20),
    'classifier__solver' : ['liblinear']},
    {'classifier' : [RandomForestClassifier()],
    'classifier__n_estimators' : range(1, 100),
    'classifier__max_depth': range(5,15),

    'classifier__criterion':['entropy', 'gini']}
]

# Create grid search object

clf = GridSearchCV(pipe, param_grid = param_grid, cv = 5, verbose=True,n_jobs=-1)

# Fit on data

best_clf = clf.fit(X_train, y_train)
clf.best_params_
```

- Hyperparameter tuning for RandomForest.

In [168]:
```
clf.best_score_,clf.best_estimator_
```

Out[168]:
```
(0.8356544863587118,
 Pipeline(memory=None,
          steps=[('classifier',
                  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                         class_weight=None, criterion='entropy',
                                         max_depth=7, max_features='auto',
                                         max_leaf_nodes=None, max_samples=None,
                                         min_impurity_decrease=0.0,
                                         min_impurity_split=None,
                                         min_samples_leaf=1, min_samples_split=2,
                                         min_weight_fraction_leaf=0.0,
                                         n_estimators=14, n_jobs=None,
                                         oob_score=False, random_state=None,
                                         verbose=0, warm_start=False))],
          verbose=False))
```

In [169]:
```
y_pred=clf.predict(X_test)
```

In [170]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[170]:
```
0.8491620111731844
```

Hyperparameter tuning of Decission Tree

In [363]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2,3,4,5],'criterion':
grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), params, verbose=1, cv
grid_search_cv.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 14112 candidates, totalling 42336 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 42336 out of 42336 | elapsed:  2.1min finished
```

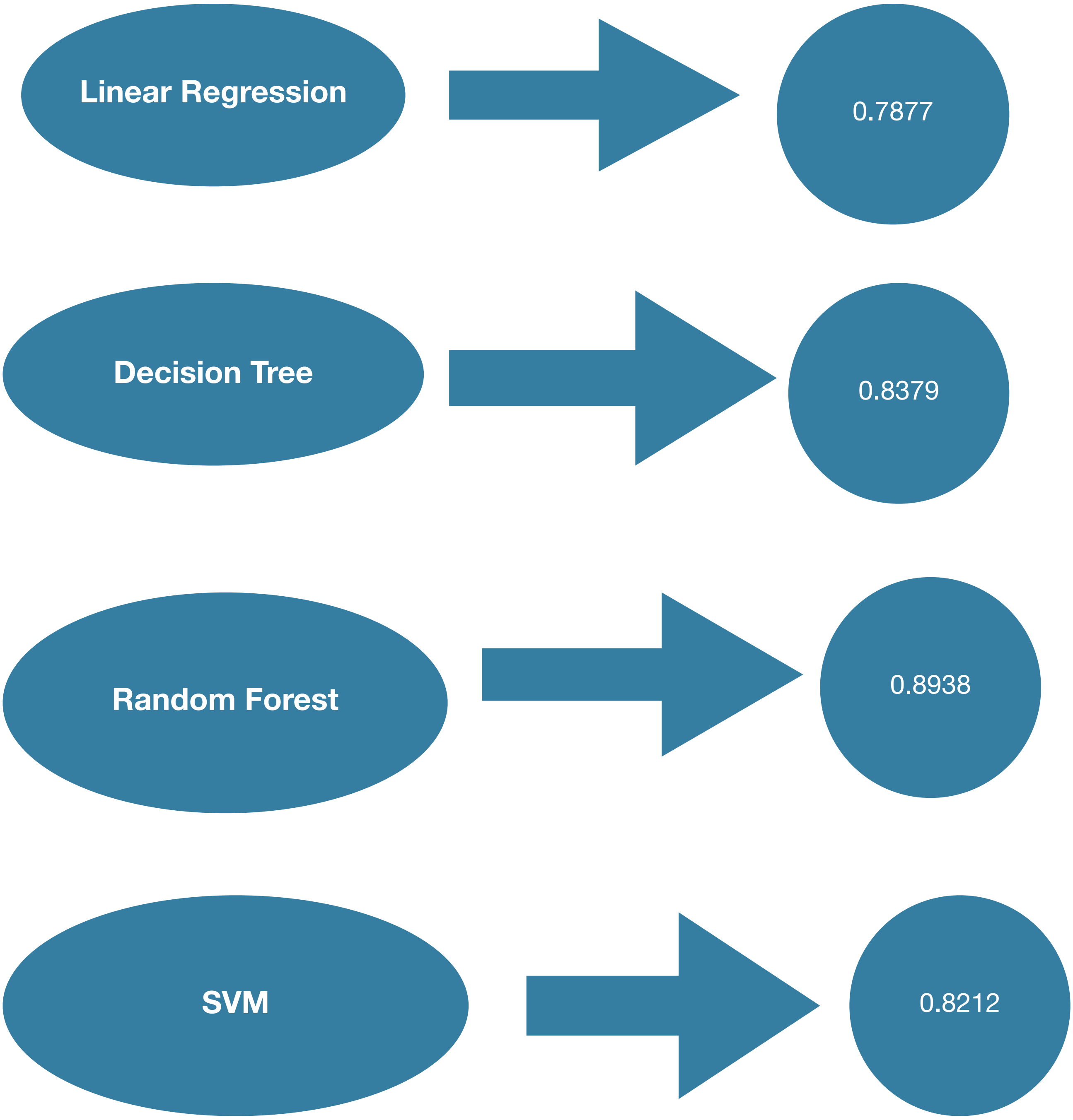
Out[363]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              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='deprecated',
                                              random_state=42,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 30,
                                       40, 50, 70, 90, 120, 150],
                         'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                            13, 14, 15, 16, 17, 18, 19, 20, 21,
                                            22, 23, 24, 25, 26, 27, 28, 29, 30,
                                            31, ...],
```

In [364]:
```
grid_search_cv.best_score_
```

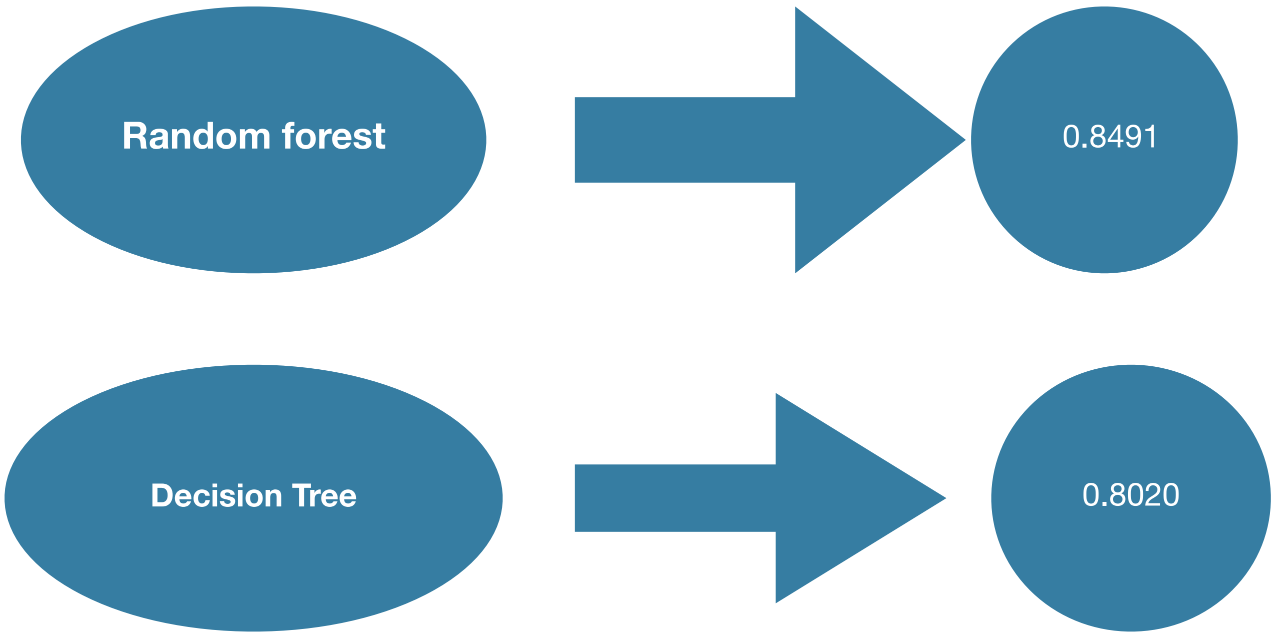
Out[364]:
```
0.8020068787008473
```

RESULTS

Accuracy scores of different Machine Learning algorithms on Titanic dataset.



RESULTS after HyperParameter Tuning



Source code

For source code visit : https://github.com/Paraslaul/Titanic (paraslaul7@gmail.com)

https://github.com/kkishitajain/Titanic (kkishitajain23@gmail.com)