DecisionTree_Titanic

December 19, 2020

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from pylab import rcParams
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn import metrics
from sklearn.metrics import classification_report
```

[2]:
```
data_url='https://raw.githubusercontent.com/BigDataGal/Python-for-Data-Science/
 ↪master/titanic-train.csv'
```

[3]:
```
#Fetching the data from URL
titanic=pd.read_csv(data_url)
titanic.
 ↪columns=['PassengerId','Survived','Pclass','Name','Sex','Age','SibSp','Parch','Ticket','Far
 ↪mbarked']
```

[4]:
```
titanic.head()
```

[4]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

[5]:
```
#Creating a new dataframe with required columns
data=titanic[['Pclass','Sex','Age','SibSp','Parch','Fare','Survived']]
```

[6]:
```
data.head()
```

[6]:

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

[7]:
```
#Getting dimentions of data frame
print(data.shape)
```

```
(891, 7)
```

[8]:
```
#Getting details of basic stats from the dataframe
data.describe()
```

[8]:

	Pclass	Age	SibSp	Parch	Fare	Survived
count	891.000000	714.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	29.699118	0.523008	0.381594	32.204208	0.383838
std	0.836071	14.526497	1.102743	0.806057	49.693429	0.486592
min	1.000000	0.420000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	20.125000	0.000000	0.000000	7.910400	0.000000
50%	3.000000	28.000000	0.000000	0.000000	14.454200	0.000000
75%	3.000000	38.000000	1.000000	0.000000	31.000000	1.000000
max	3.000000	80.000000	8.000000	6.000000	512.329200	1.000000

[9]:
```
#Converting sex columns to numeric by mapping male to 1 and female to 0
data.loc[:,'Sex']=data['Sex'].replace(['male','female'],[0,1])
```

```
C:\Users\garahul\Anaconda3\lib\site-packages\pandas\core\indexing.py:966:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

[10]:
```
#Check for NULL values in the dataframe
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Pclass    891 non-null    int64
 1   Sex       891 non-null    int64
 2   Age       714 non-null    float64
 3   SibSp     891 non-null    int64
 4   Parch     891 non-null    int64
 5   Fare      891 non-null    float64
 6   Survived  891 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

[11]:
```
table=pd.crosstab(data['Survived'],data['Sex'])
print(table)
```

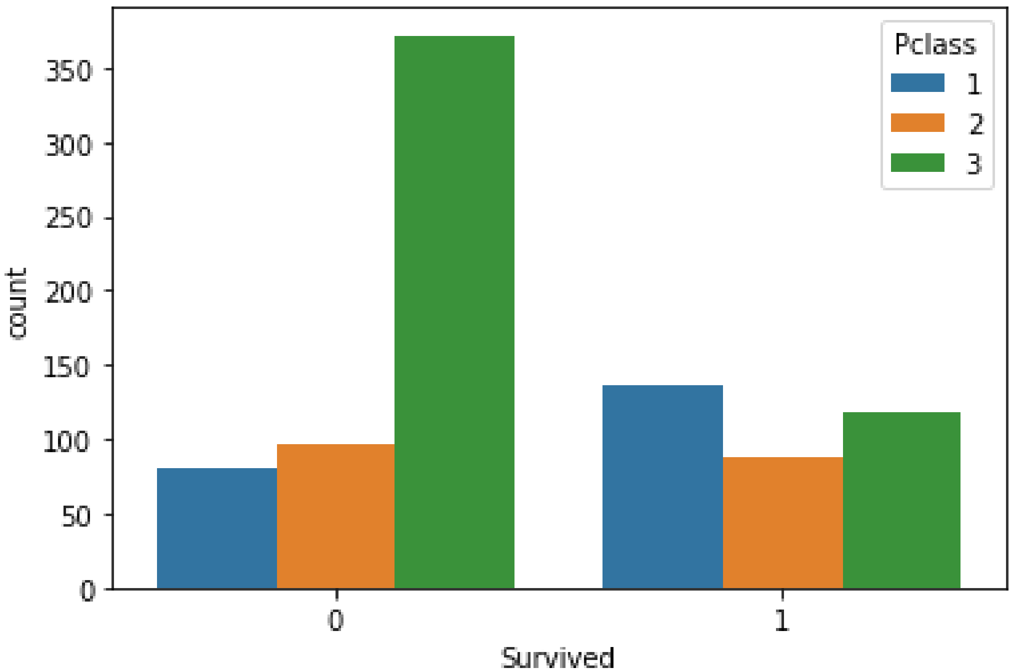
```
Sex         0    1
Survived
0         468   81
1         109  233
```

[12]:
```
data.Age.isnull().sum()
print('No of Null Values in Age Column {} and percentage {}'.format(data.Age.
 ↪isnull().sum(),round(data.Age.isnull().sum()/data.Age.count() * 100,2)))
```

```
No of Null Values in Age Column 177 and percentage 24.79
```

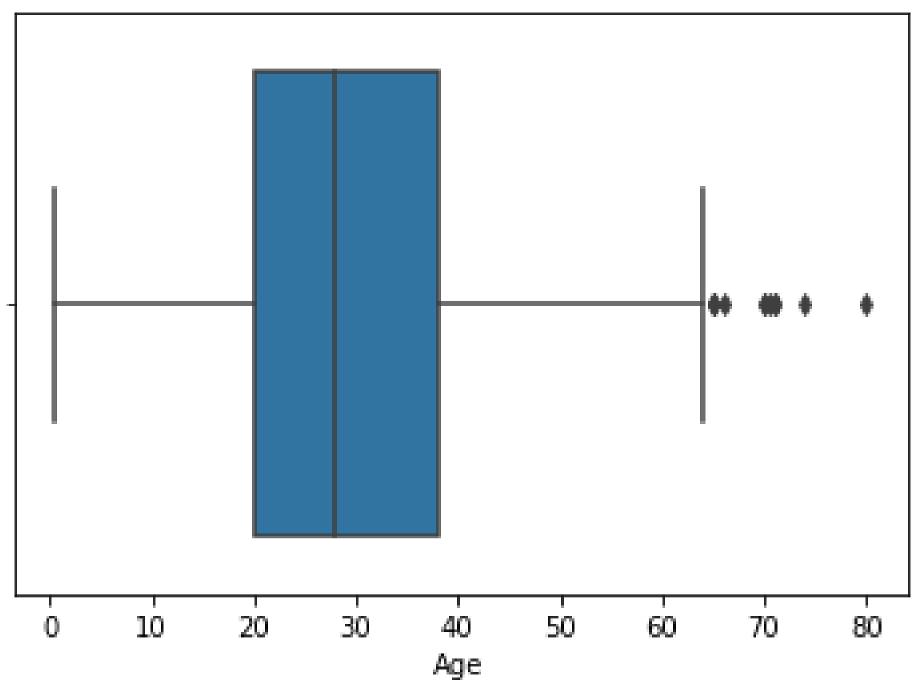
[13]:
```
#Checking the class wise distribution for Survived Feature
sns.countplot(x='Survived',hue='Pclass',data=data)
```

[13]: `<matplotlib.axes._subplots.AxesSubplot at 0x1aa7d643cd0>`



[14]:
```
sns.boxplot('Age',data=data)
```

[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1aa7ddf5850>



[15]:
```python
#Getting Mean age of By Sex
age_list=data.groupby('Sex')['Age'].mean().to_list()
male_age=age_list[0]
female_age=age_list[1]
print('Mean age of Male is {} and Female is {}'.
 ↪format(round(male_age,2),round(female_age,2)))
```

```
Mean age of Male is 30.73 and Female is 27.92
```

[16]:
```python
data.loc[(data['Sex']==0) & (data['Age'].isna()),'Age']=round(female_age)
data.loc[(data['Sex']==1) & (data['Age'].isna()),'Age']=round(male_age)
print(data.isna().sum())
```

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

C:\Users\garahul\Anaconda3\lib\site-packages\pandas\core\indexing.py:966:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

[17]:
```python
X=data.drop('Survived',axis=1)
Y=data['Survived']
```

[18]:
```python
X
```

[18]:

	Pclass	Sex	Age	SibSp	Parch	Fare
0	3	0	22.0	1	0	7.2500
1	1	1	38.0	1	0	71.2833
2	3	1	26.0	0	0	7.9250
3	1	1	35.0	1	0	53.1000
4	3	0	35.0	0	0	8.0500
..	…	…	…	…	…	…
886	2	0	27.0	0	0	13.0000
887	1	1	19.0	0	0	30.0000
888	3	1	31.0	1	2	23.4500

```
889        1     0   26.0       0       0   30.0000
890        3     0   32.0       0       0    7.7500

[891 rows x 6 columns]
```

[19]:
```
Y
```

[19]:
```
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
```

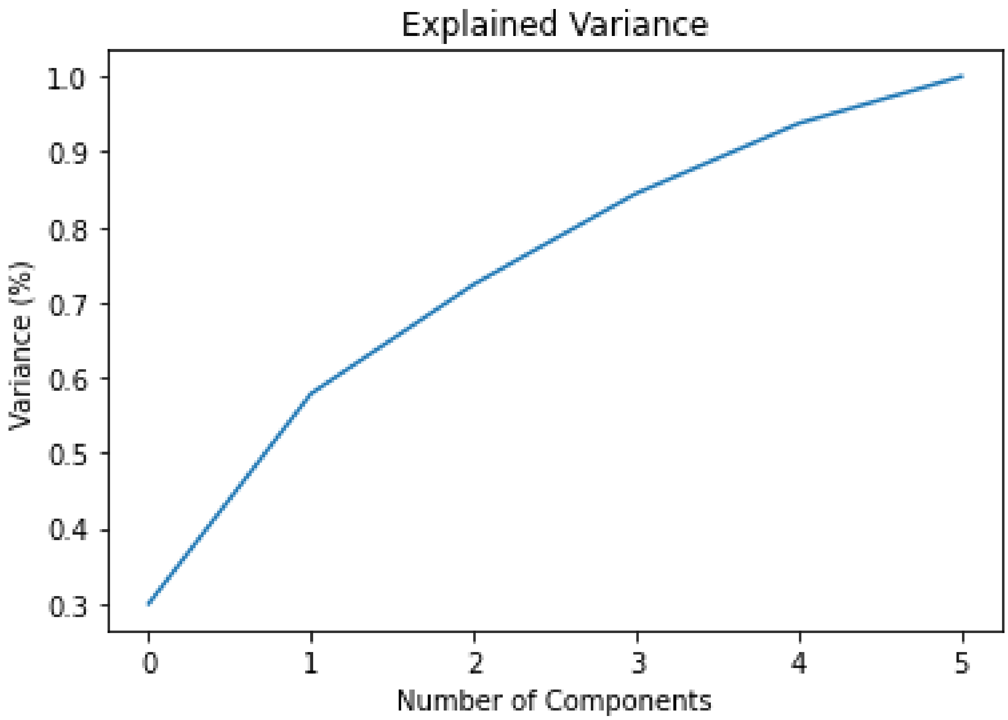
[20]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
 ↪roc_auc_score
```

[21]:
```
scalar = StandardScaler()
X_scaled = scalar.fit_transform(X)
```

[22]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scaled,Y,test_size = 0.20,␣
 ↪random_state= 355)
```

[23]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
pca = PCA()
principalComponents = pca.fit_transform(X_scaled)
print(pca.explained_variance_ratio_)
print(np.cumsum(pca.explained_variance_ratio_))
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Explained Variance')
plt.show()
```

```
[0.29936847 0.27917601 0.14521662 0.12085065 0.09317706 0.06221118]
[0.29936847 0.57854448 0.7237611  0.84461175 0.93778882 1.        ]
```



No need for PCA in this case as no major imapct by reducing no of columns

[24]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
clf = DecisionTreeClassifier()
clf.fit(x_train,y_train)
clf.score(x_test,y_test)
```

[24]: 0.7430167597765364

[25]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
 ↪roc_auc_score
# we are tuning three hyperparameters right now, we are passing the different␣
 ↪values for both parameters
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,32,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'splitter' : ['best', 'random']
}
```

[26]:
```
grid_search = GridSearchCV(estimator=clf,
                     param_grid=grid_param,
                     cv=5,
```

```
                    n_jobs =-1)
```

[27]:
```
grid_search.fit(x_train,y_train)
```

[27]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})
```

[28]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'criterion': 'gini', 'max_depth': 31, 'min_samples_leaf': 3,
'min_samples_split': 9, 'splitter': 'random'}
```

[29]:
```
grid_search.best_score_
```

[29]:
```
0.8272037821333595
```

[30]:
```
clf = DecisionTreeClassifier(criterion = 'entropy', max_depth =13,␣
 ↪min_samples_leaf= 6, min_samples_split= 4, splitter ='best')
clf.fit(x_train,y_train)
```

[30]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=13, min_samples_leaf=6,
                       min_samples_split=4)
```

[31]:
```
clf.score(x_test,y_test)
```

[31]:
```
0.7988826815642458
```

[32]:
```
y_pred=clf.predict(x_test)
```

[33]:
```
print(list(y_test))
```

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

[34]:
```
print(list(y_pred))
```

```
[1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
```

```
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[35]:
```python
confusion_matrix(y_test, y_pred)
```

[35]:
```
array([[98, 21],
       [15, 45]], dtype=int64)
```

[36]:
```python
tn,fp,fn,tp=confusion_matrix(y_test, y_pred).ravel()
print('True Negative: ',tn)
print('False Positive: ',fp)
print('False Negative: ',fn)
print('True Positive: ',tp)
```

```
True Negative:  98
False Positive:  21
False Negative:  15
True Positive:  45
```

[37]:
```python
print('Accuracy Score: ',accuracy_score(y_test, y_pred))
```

```
Accuracy Score:  0.7988826815642458
```

[38]:
```python
#Working on getting Precission and Recall Value
precision=tp/(tp+fp)
print('Precision: ', round(precision,2))
```

```
Precision:  0.68
```

[39]:
```python
recall=tp/tp+fn
print('Recall Value:', round(recall,2))
```

```
Recall Value: 16.0
```

[40]:
```python
f1_score=2*precision*recall/(precision + recall)
print('F1 Score: ',f1_score)
```

```
F1 Score:  1.307901907356948
```

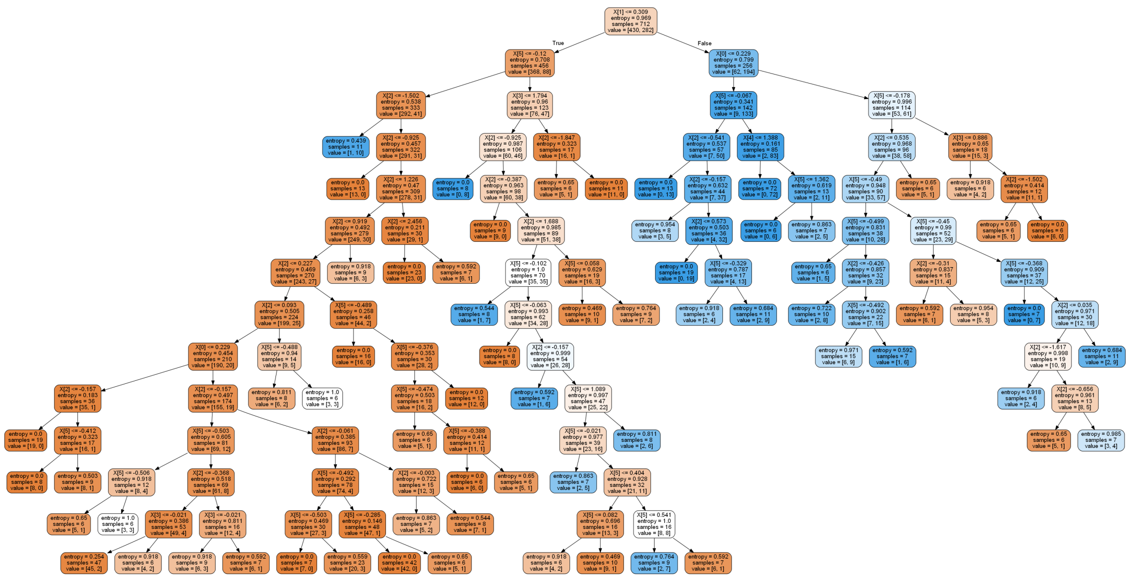
[41]:
```python
from sklearn.metrics import f1_score
score = f1_score(y_test, y_pred, average='binary')
print(score)
```

```
0.7142857142857143
```

[42]:
```python
import pydotplus
from IPython.display import Image
```

[43]:
```python
feature_name=list(X.columns)
class_name = list(y_train.unique())
# create a dot_file which stores the tree structure
dot_data = export_graphviz(clf,rounded = True,filled = True)
# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png("titanic_tree.png")
# Show graph
Image(graph.create_png())
```

[43]:

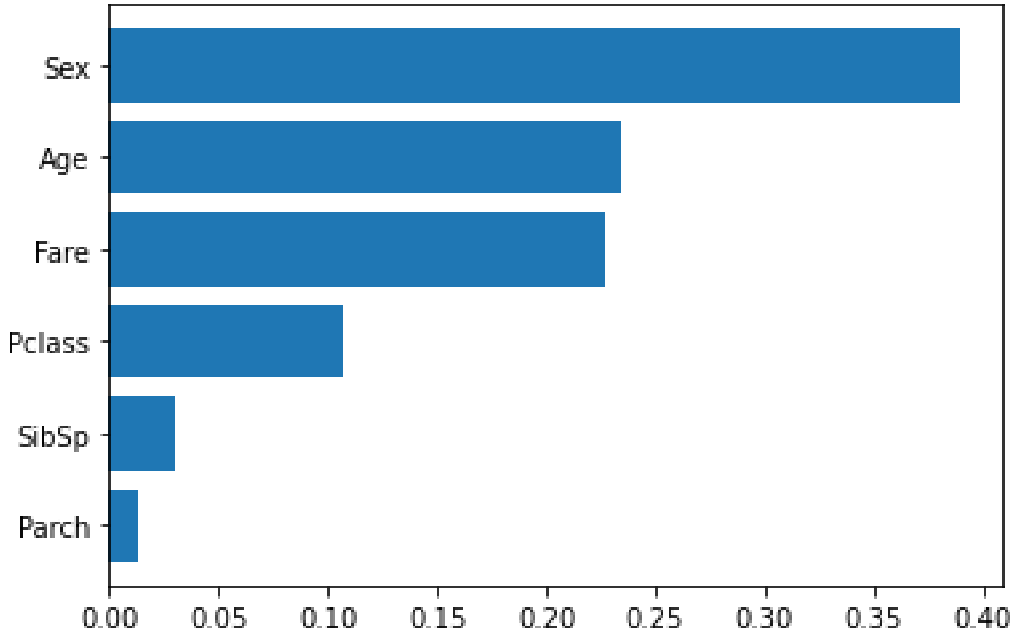


[44]:
```python
clf.feature_importances_
```

[44]:
```
array([0.10683133, 0.38917158, 0.23362812, 0.03038756, 0.01345519,
       0.22652622])
```

[45]:
```python
#Checking the importance of each feature and
pd.Series(clf.feature_importances_,X.columns).sort_values(ascending=True).plot.
 ↪barh(width=0.8)
```

[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1aa7e6fe1f0>
```


Sex
Age
Fare
Pclass
SibSp
Parch
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

```
[ ]:
```