logistic Regression

Data set: Titanic Data Set from Kaggle. Goal: trying to predict a classification- survival or deceased.

Note: Data is a "semi-cleaned" version of the titanic data set. It has non values.

Import Libraries

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Load the dada set

In [30]:

```
train_org = pd.read_csv('Data/titanic_train.csv')
train_org.head(5)
```

Out[30]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
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	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

In [90]:

```
# data statistic
print(train_org.shape)
print('total nun values= ',train_org.isnull().sum().sum())
print(train_org.isnull().sum())
```

```
(891, 12)
total nun values=  866
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
```

In [100]:

```
# object columns
s = (train_org.dtypes == 'object')
#print (s)
col_object = list(s[s].index)
print(col_object)
```

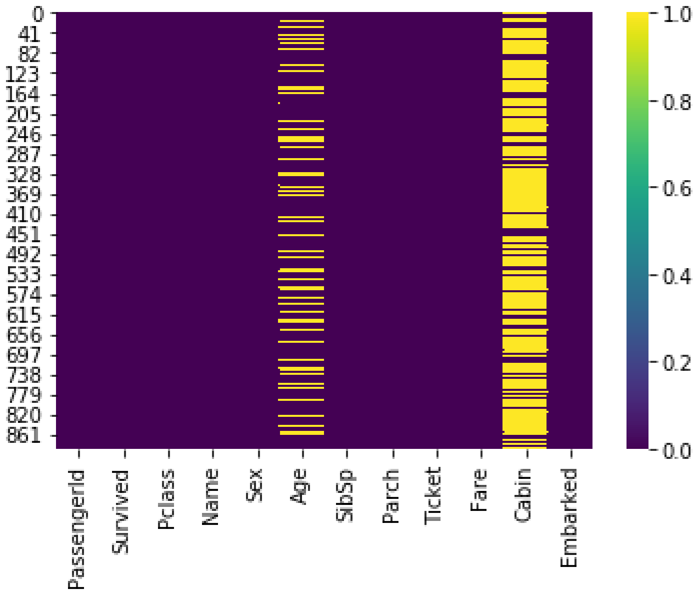
```
['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']
```

In [34]:

```
sns.heatmap(train_org.isnull(), cmap='viridis')
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x210860c1f98>
```



Preprocessing on Data

In Cabin colum, around 77% of data are missing --> drop the column In Age column, around 20% of the data are missing --> impute ...

Categorical column like Sex must be converted into numerical values

Column Ticket and Name will also be dropped, (no useful features)

In [19]:
```
# make a copy from original data
train_clean = train_org.copy()
```

Drop un-necessary columns

In [22]:
```
print(train_clean.shape)
train_clean.columns
```

```
(891, 12)
```

Out[22]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [24]:
```
train_clean.drop(columns=['Name', 'Ticket', 'Cabin'], inplace=True)
```

In [25]:
```
print(train_clean.shape)
train_clean.columns
```

```
(891, 9)
```

Out[25]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Fare', 'Embarked'],
      dtype='object')
```

Handle missing Data in Age column

Filling in the mean age of all the passengers (imputation).

Or here, fill them by the average age of the passenger in class.

In [31]:
```
print('Average age of all pasenger = %d' %(train_clean['Age'].dropna().mean
()))
```

```
Average age of all pasenger = 29
```

In [45]:

```
train_clean.corrwith(train_clean['Age'])
```

Out[45]:

```
PassengerId    0.036847
Survived      -0.077221
Pclass        -0.369226
Age            1.000000
SibSp         -0.308247
Parch         -0.189119
Fare           0.096067
dtype: float64
```

In [52]:

```
train_clean.groupby('Pclass')['Age'].mean().reset_index()
```

Out[52]:

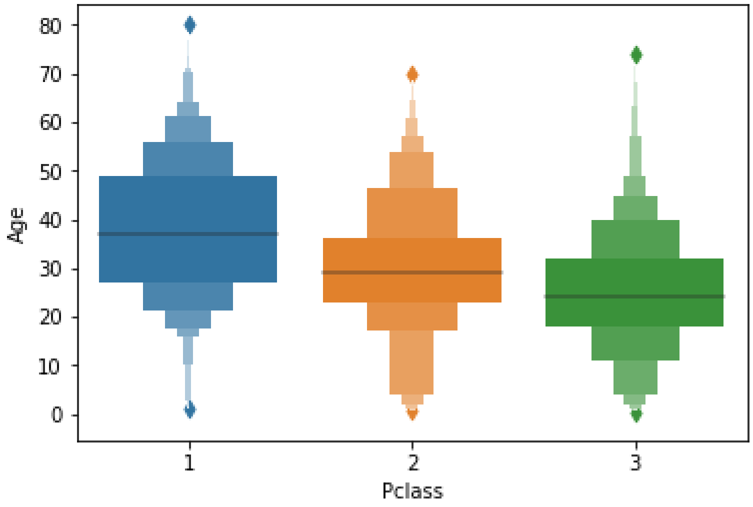
	Pclass	Age
0	1	38.233441
1	2	29.877630
2	3	25.140620

In [44]:

```
sns.boxenplot(x='Pclass', y='Age', data=train_clean)
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2108c4f2d68>
```



In [53]:

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 38

        elif Pclass == 2:
            return 30

        else:
            return 25

    else:
        return Age
```

In [55]:

```
train_clean['Age'] = train_clean[['Age', 'Pclass']].apply(impute_age, axis=1)
```

In [56]:

```
train_clean['Age'].isnull().sum()
```

Out[56]:

```
0
```

Handle numerical column

column Sex, Embarked

In [59]:

```
sex = pd.get_dummies(train_clean['Sex'], drop_first=True)
```

In [63]:

```
train_clean['Embarked'].value_counts()
```

Out[63]:

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [115]:

```
embarked = pd.get_dummies(train_clean['Embarked'], drop_first=True)
```

In [116]:

```
train_clean.drop(columns=['Sex', 'Embarked'], inplace=True, axis=1)
```

In [117]:

```
print(sex.shape)
print(embarked.shape)
print(train_clean.shape)
```

```
(891, 1)
(891, 2)
(891, 7)
```

In [118]:

```
train = pd.concat([train_clean, sex, embarked], axis=1)
```

In [119]:

```
train.shape
```

Out[119]:

```
(891, 10)
```

In [120]:

```
train.columns
```

Out[120]:

```
Index(['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare',
       'male', 'Q', 'S'],
      dtype='object')
```

In [124]:

```
(train.dtypes == 'Object').any()
```

Out[124]:

```
False
```

Logistic Regression Model

In [125]:

```
X = train.drop(columns=['Survived'], axis=1)
y = train['Survived']
```

In [126]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

In [127]:

```
from sklearn.linear_model import LogisticRegression
```

In [132]:

```
def get_acc (max_iter):
    log_m = LogisticRegression(max_iter=max_iter)
    log_m.fit(X_train, y_train)
    log_m.predict(X_test)
    acc_train = log_m.score(X_train, y_train)
    acc_test = log_m.score(X_test, y_test)
    return[acc_train, acc_test]
```

In [135]:

```
import warnings; warnings.simplefilter('ignore')
```

In [136]:

```
get_acc(100)
```

Out[136]:

```
[0.7929373996789727, 0.835820895522388]
```

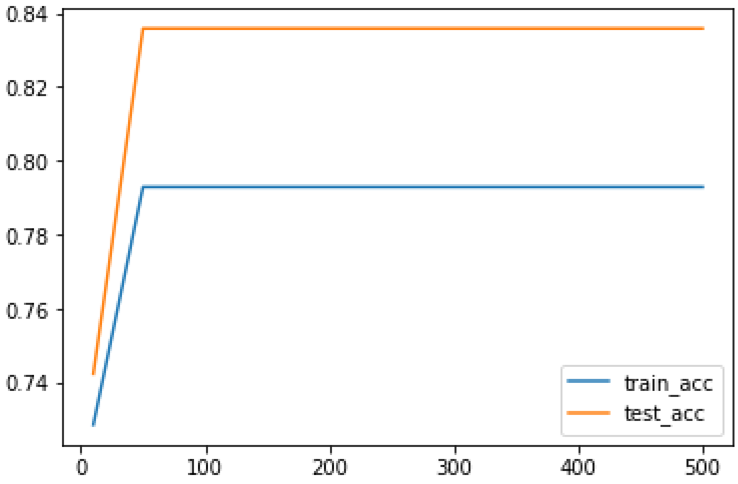
In [143]:

```
train_acc = []
test_acc = []
iteration = [10, 50, 100, 200, 500]

for i in iteration:
    res = get_acc(i)
    train_acc.append(res[0])
    test_acc.append(res[1])
```

```
In [144]: plt.plot(iteration, train_acc)
          plt.plot(iteration, test_acc)
          plt.legend(['train_acc', 'test_acc'])

Out[144]: <matplotlib.legend.Legend at 0x2108faa4400>
```



```
In [ ]:
```

Random Forest model

```
In [146]: from sklearn.ensemble import RandomForestClassifier
```

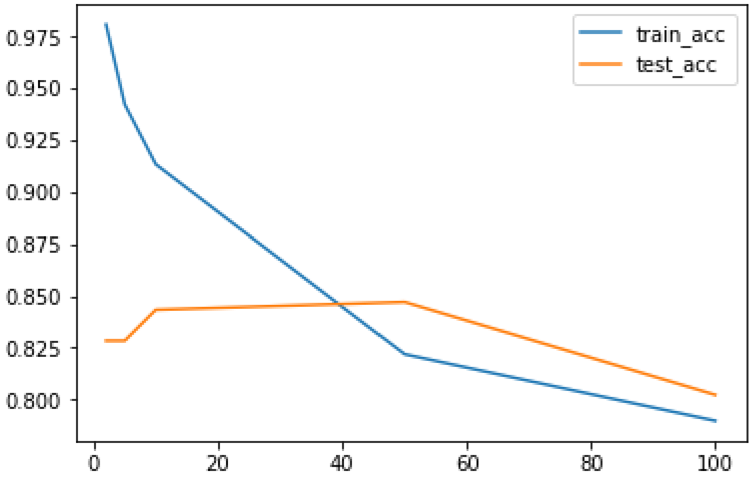
```
In [149]: def get_acc_rfc (min_samples_split):
              mdl = RandomForestClassifier(min_samples_split=min_samples_split)
              mdl.fit(X_train, y_train)
              #log_m.predict(X_test)
              acc_train = mdl.score(X_train, y_train)
              acc_test = mdl.score(X_test, y_test)
              return[acc_train, acc_test]
```

```
In [150]: train_acc = []
          test_acc = []
          min_sample = [2, 5, 10, 50, 100]

          for i in min_sample:
              res = get_acc_rfc(i)
              train_acc.append(res[0])
              test_acc.append(res[1])
```

```
In [152]: plt.plot(min_sample, train_acc)
          plt.plot(min_sample, test_acc)
          plt.legend(['train_acc', 'test_acc'])

Out[152]: <matplotlib.legend.Legend at 0x21094ab0eb8>
```



```
In [153]: get_acc_rfc(min_samples_split=35)

Out[153]: [0.8491171749598716, 0.8432835820895522]
```