RUTGERS
Rutgers Business School
Newark and New Brunswick

Shipwreck Survivor Prediction Analysis Competition

Zhaofeng Shang

181006463

1. Business Understanding

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

In this challenge, the job is to complete the analysis of what sorts of people were likely to survive. It is necessary to apply the tools of machine learning to predict which passengers survived the tragedy.

2. Data Gathering

Here are all the attributes in this dataset, Sibling = brother, sister, stepbrother, stepsister
Spouse = husband, wife (mistresses and fiancés were ignored).

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Visualization

```
# Pclass with Survival
S_0=data_train.Pclass[data_train.Survived==0].value_counts()# survival arranged by Pclass
S_1=data_train.Pclass[data_train.Survived==1].value_counts()
df=pd.DataFrame({'Survival':S_1,'None':S_0})
df.plot(kind='bar',stacked=True)
plt.title('Pclass with Survival')
plt.xlabel('Pclass')
plt.ylabel('number')
plt.legend()
```

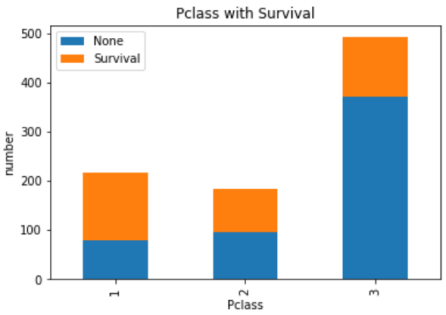
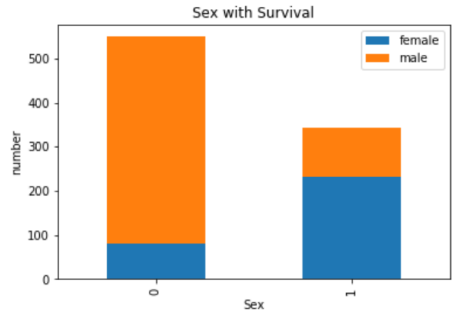


Figure above shows the relationship between “Pclass” and “Survival”, With the reduction of the class level, the number of survivors decreased.

```
#  Sex with Survival
S_m=data_train.Survived[data_train.Sex=='male'].value_counts()
Survived_cabin = data_train.Survived[pd.notnull(data_train.Cabin)].value_counts()
S_f=data_train.Survived[data_train.Sex=='female'].value_counts()
df=DataFrame({'male':S_m,'female':S_f})
print(df)
df.plot(kind='bar',stacked=True)
plt.title('Sex with Survival')
plt.ylabel('number')
plt.xlabel('Sex')

```

```
   female  male
0      81   468
1     233   109
```

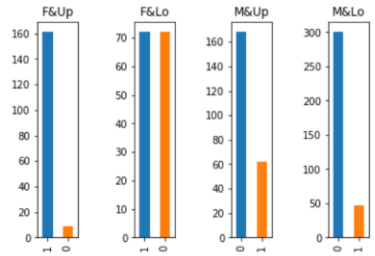
```
Text(0.5,0,'Sex')
```



The figure above shows the relationship between “Sex” and “Survival”, “0” in the figure represent not survive and “1” is in other side. Figure shows that the survival rate of female is much higher than male.

```
# Combine Sex and Pclass with Survival

plt.subplot2grid((1,7),(0,0))
data_train.Survived[data_train.Sex == 'female'][data_train.Pclass != 3].value_counts().plot('bar')
plt.title('F&Up')

plt.subplot2grid((1,7),(0,2))
data_train.Survived[data_train.Sex == 'female'][data_train.Pclass == 3].value_counts().plot('bar')
plt.title('F&Lo')

plt.subplot2grid((1,7),(0,4))
data_train.Survived[data_train.Sex == 'male'][data_train.Pclass !=3].value_counts().plot('bar')
plt.title('M&Up')

plt.subplot2grid((1,7),(0,6))
data_train.Survived[data_train.Sex == 'male'][data_train.Pclass == 3].value_counts().plot('bar')
plt.title('M&Lo')
```

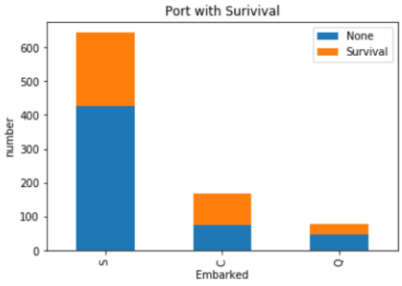
```
Text(0.5,1,'M&Lo')
```



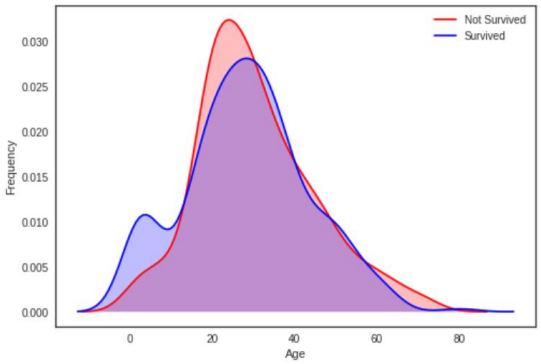
The figure above shows the relationship between “Survival” and the combination of “Sex” and “Pclass”. There are four different combination, all show that higher class level and female will get higher survival rate.

```python
# Embarked with Survival
Survived_0 = data_train.Embarked[data_train.Survived == 0].value_counts()
Survived_1 = data_train.Embarked[data_train.Survived == 1].value_counts()
df=pd.DataFrame({'Survival':Survived_1,'None':Survived_0})
print(df)
df.plot(kind='bar', stacked=True)
plt.title("Port with Surivival")
plt.xlabel('Embarked')
plt.ylabel('number')

```

```
   None  Survival
S   427       217
C    75        93
Q    47        30
```



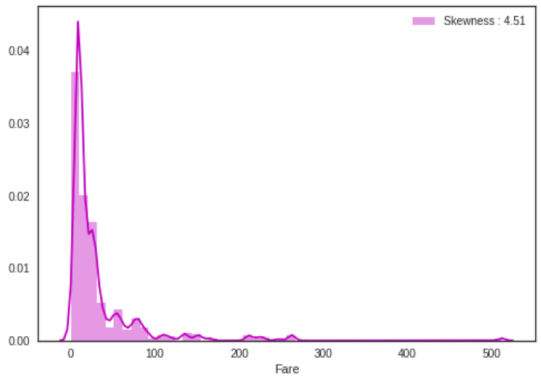
The figure above shows the relationship between “Port” and “Survival”. According to the ratio, the Embarked “C” has higher survival rate.




```
# Explore Age distibution
g = sns.kdeplot(train["Age"][(train["Survived"] == 0) & (train["Age"].notnull())], color="Red", shade = True)
g = sns.kdeplot(train["Age"][(train["Survived"] == 1) & (train["Age"].notnull())], ax =g, color="Blue", shade= True)
g.set_xlabel("Age")
g.set_ylabel("Frequency")
g = g.legend(["Not Survived","Survived"])
```

When we overlap the two graphs, it clearly shows a peak between 0-5, which represent the babies and young children. People during 20-30 has highest death rate, which represent the young people. People during 30-40 has high survival rate, which represent those matured people. And, the old person during 60-80 has low survival rate.

```
g = sns.distplot(dataset["Fare"], color="m", label="Skewness : %.2f"%(dataset["Fare"].skew()))
g = g.legend(loc="best")
```



The figure above shows that distribution of “Fare” attribute, High skewed Fare distribution can lead to overweight values in the model, even if it is scaled. In this case, it is better to transform it with the log function to reduce this skew.

Statistical Tests

```
1 train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
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(4), object(5)
memory usage: 59.2+ KB
```

The table above shows the basic information about the whole dataset, attribute “Age”, “Cabin” and “Embarked” has missing values, the next work is to finish data preprocessing.

3. Data Preprocessing

3.1 Correcting

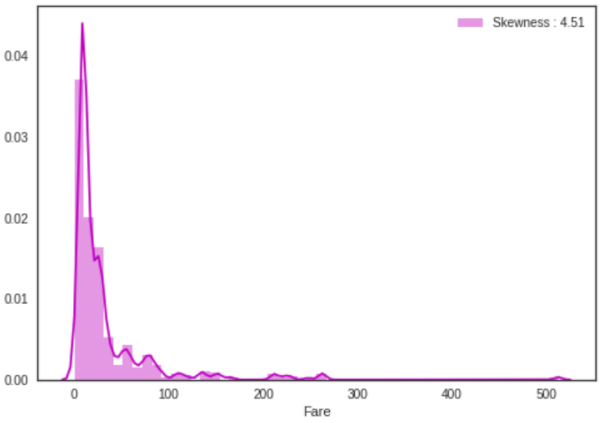
3.1.1 Outliers

Handling with outliers in the dataset is essentially important thing to improve the accuracy of prediction model. **Tukey–Kramer method** is a single-step multiple comparison procedure and statistical test. This method defines an interquartile range comprised between the 1st and 3rd quartile of the distribution values (IQR). An outlier is a row that have a feature value outside the (IQR +- an outlier step).

Here, using Tukey-Kramer method to handle the outliers in all numerical variables as bellow:

```
from collections import Counter
def detect_outliers(df,n,features):

    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[col],75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers
# detect outliers from Age, SibSp , Parch and Fare
Outliers_to_drop = detect_outliers(train,2,["Age","SibSp","Parch","Fare"])
```

3.1.2 Standardization

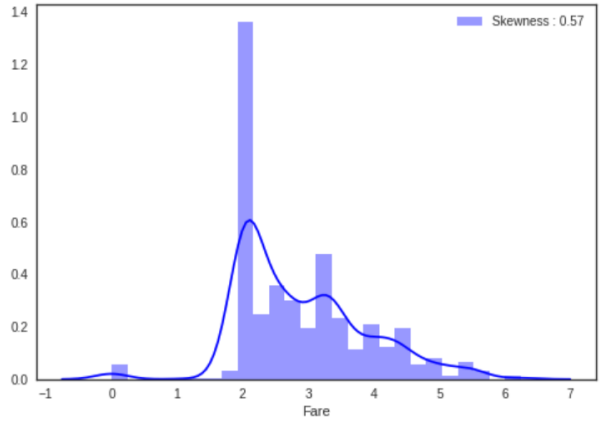


According to the “Fare” distribution above, it is very skewed, which will lead to overweight values in the model. In this case, it’s better to transform it with the log function to reduce the skew.

```
# Apply log to Fare to reduce skewness distribution
dataset["Fare"] = dataset["Fare"].map(lambda i: np.log(i) if i > 0 else 0)
```

```
graph = sns.distplot(dataset['Fare'], color='b', label='sKewness: %.2f' %(dataset['Fare'].skew()))
```

After applying the log function, the distribution of Fare as follow



:

3.2 Completing

3.2.1 Missing values

```
1 dataset.isnull().sum()
```

```
Age            256
Cabin          1007
Embarked          2
Fare              0
Name              0
Parch             0
PassengerId       0
Pclass            0
Sex               0
SibSp             0
Survived        418
Ticket            0
dtype: int64
```

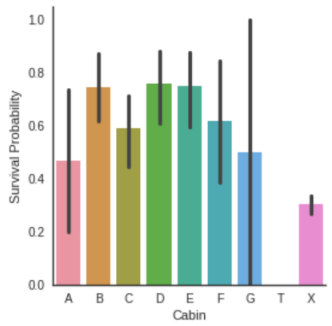
“Embarked” attribute:

```
def fill_missing_embarked(data):
    freq_port = data['Embarked'].mode()[0]
    data['Embarked'] = data['Embarked'].fillna(freq_port)
    data['Embarked'] = data['Embarked'].map({'S': 0, 'Q': 1, 'C': 2}).astype(int)
    return data

dataset = fill_missing_embarked(dataset)
```

According to the information table above, the number of the “Embarked” missing value is two. Here used the mode() function to fill the missing value, and used map method to change categorical value into the numerical value.

“Cabin” attribute:

First method:




```
def cabin_extract(data):  #  1-L  2-M  3-H
    # classify Cabin by fare
    data['Cabin'] = data['Cabin'].fillna('X')
    data['Cabin'] = data['Cabin'].apply(lambda x: str(x)[0])
    data['Cabin'] = data['Cabin'].replace(['A', 'D', 'E', 'T'], int(1))
    data['Cabin'] = data['Cabin'].replace(['B', 'C'], int(2))
    data['Cabin'] = data['Cabin'].replace(['F', 'G'], int(3))
    data['Cabin'] = data['Cabin'].replace(['X'], int(0))
   # data['Cabin'] = data['Cabin'].map({'X': 0, 'L': 1, 'M': 2, 'H': 3}).astype(int)
    return data
```

“Cabin” attribute is comprised by letter and number. Created “cabin_extract” function to fill null data and get the letter of the “Cabin” attribute, grouped all values into four intervals.

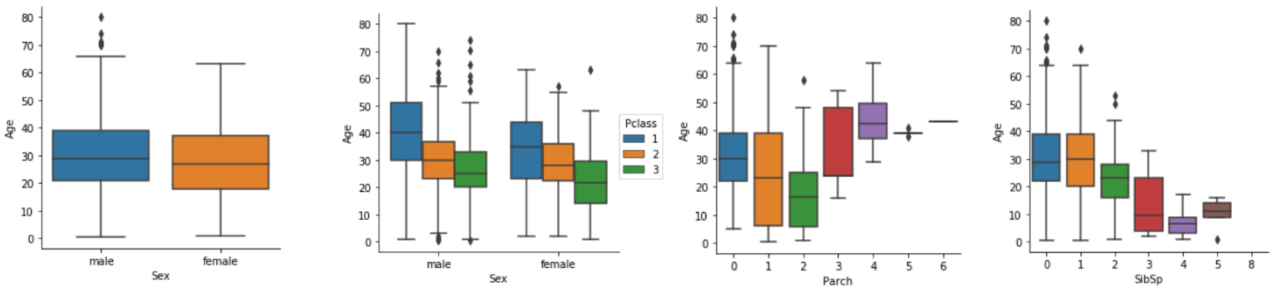
Second method:

Rather than changing the attribute “Cabin” as above, it may be better to delete the whole column because of too much missing values.

```
dataset.drop(labels='Cabin',axis=1,inplace=True)
```

“Age” attribute:

```
g = sns.factorplot(y="Age",x="Sex",data=dataset,kind="box")
g = sns.factorplot(y="Age",x="Sex",hue="Pclass", data=dataset,kind="box")
g = sns.factorplot(y="Age",x="Parch", data=dataset,kind="box")
g = sns.factorplot(y="Age",x="SibSp", data=dataset,kind="box")
```



The figure above shows the relationship between “Age” and several related attributes. Attribute “Sex” has less impact on the “Age”. However, 1st class passengers are older than 2nd class passengers who are also older than 3rd class passengers. Moreover, the more a passenger has parents/children the older he is and the more a passenger has siblings/spouses the younger he is.

Based on the correlation graph of “Pclass”, “Parch”, “SibSp” and “Age”, the strategy to fill missing age is to find the median of similar rows according to those attributes which have own “Age” value, if the finding result is empty, then fill those missing age with the new feature “Title” which will be described in

the following content.

```
def missing_age(data):
    data_not = data.loc[train_data.Age.notnull(),:]

    age_null = list(data[data['Age'].isnull()].index)
    print(len(age_null))
    for i in age_null:
        age_pred = data_not['Age'][((data_not['SibSp']==data.iloc[i]['SibSp'])&(data_not['Parch'] == data.iloc[i]["Parch"]) &
                                    (data_not['Pclass'] == data["Pclass"].iloc[i]))].median()
        if np.isnan(age_pred):
            data['Age'].iloc[i] = data_not['Age'][data_not['Title']==data['Title'].iloc[i]].median()
        else:
            data['Age'].iloc[i] = age_pred
    return data
```

3.3 Converting

3.31 Dummy Variable

```
def dummy(data, columns):  # temporarily just for Embarked
    for column in columns:
        if column not in data.columns:
            continue
        dummy_data = pd.get_dummies(data[column], drop_first=False,prefix='Embarked')

        data = pd.concat([data, dummy_data], axis=1)

        data = data.drop(column, axis=1)
    return data
```

The attribute “Embarked” has three categorical values S, Q and C. The function above is to change the categorical values into numerical values by using Dummy.

3.32 Factorization

Age:

```
def age_extract(data):

    data.loc[data['Age'] <= 22, 'Age'] = 1
    data.loc[(data['Age'] <= 26) & (data['Age'] > 22), 'Age'] = 2
    data.loc[(data['Age'] <=37 ) & (data['Age'] > 26), 'Age'] = 3
    data.loc[data['Age'] > 37, 'Age'] = 4
    data['Age'] = data['Age'].astype(int)
    return data
```

After filling all missing values, the function above changed the “Age” value into four intervals according to the quartile value. This kind of computation largely decrease the complexity.

Pclass:

```
# the pclass has three different values
```

```
dataset = pd.get_dummies(dataset, columns = ["Pclass"],prefix="Pc")
```

With the same method, the pclass attribute also has three values, we apply the dummy method to deal with this attribute.

Sex:

```
# treat with Sex attribute
def SexByPclass(data):
    data['Sex'] = data['Sex'].map({'female': 1, 'male':0})
    return data
dataset = SexByPclass(dataset)
```

According to the attribute Sex, there are two categories, male and female, here use the map function to handle this.

3.4 Creating

3.4.1 Attribute : Fare_Stage

```
# Here, deal with the Fare, combine the train and test dataset of mean to discretized values
def fare_stage(data, mean_fare):
    data.loc[data['Fare'] > mean_fare[1], 'FareStage'] = 1
    data.loc[(data['Fare'] > mean_fare[2]) & (data['Fare'] <= mean_fare[1]), 'FareStage'] = 2
    data.loc[(data['Fare'] > mean_fare[3]) & (data['Fare'] <= mean_fare[2]), 'FareStage'] = 3
    data.loc[data['Fare'] <=mean_fare[3],'FareStage'] = 4
    return data
combine = pd.concat([train_data.drop('Survived',axis=1), test_data])
mean_fare = combine.groupby('Pclass')['Fare'].mean().astype(int)
mean_fare.plot()
print(mean_fare)
train_data = fare_stage(train_data, mean_fare)
test_data = fare_stage(test_data, mean_fare)
```

3.4.2 Attribute : Title

It is not enough to build model only based on existed attributes, the function above created a new attribute “FareStage” to separate “Fare” into four intervals according to the quartile value.

```
def name_extract(data):
    # extract Title from name
    data['Title'] = data.Name.str.extract('([A-Za-z]+)\.', expand=False)# A-Za-z and end with a mark
    # delete rare title
    data['Title'] = data['Title'].replace(['Lady', 'Countess', 'Capt', 'Col',
        'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    data['Title'] = data['Title'].replace(['Mlle', 'Ms'], 'Miss')
    data['Title'] = data['Title'].replace('Mme', 'Mrs')

    title_map = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
    data['Title'] = data['Title'].map(title_map).astype(int)

    return data.drop('Name', axis=1)
```

The function above extracted title from “Name” attribute by regular expression to create new attribute “Title”.

3.43 Attribute : FamilySize & Alone

```
def family_info(data):
    data['FamliySize'] = data['SibSp'] + data['Parch'] + 1 #plus himself
    data['Alone'] = data['Alone'] = (data['SibSp'] == 0) & (data['Parch'] == 0)
    data['Alone'] = data['Alone'].astype(int)
    return data
```

Based on the number of “SibSP” and “Parch”, the function above created new attributes “FamilySize” and “Alone”.

3.44 Attribute: Ticket

```
Ticket = []
for i in list(dataset.Ticket):
    if not i.isdigit() :
        Ticket.append(i.replace(".","").replace("/","").strip().split(' ')[0]) #Take prefix
    else:
        Ticket.append("X")
```

```
dataset['Ticket']=Ticket
dataset = pd.get_dummies(dataset, columns = ["Ticket"], prefix="T")
```

The tickets show that sharing the same prefixes could be booked for cabins placed together. It could therefore lead to the actual placement of the cabins within the ship. In this way, tickets with same prefixes may have a similar class and survival. So that extracting all the tickets' prefix and replacing the numerical values with "X" seem to be a better method.

4. Modeling

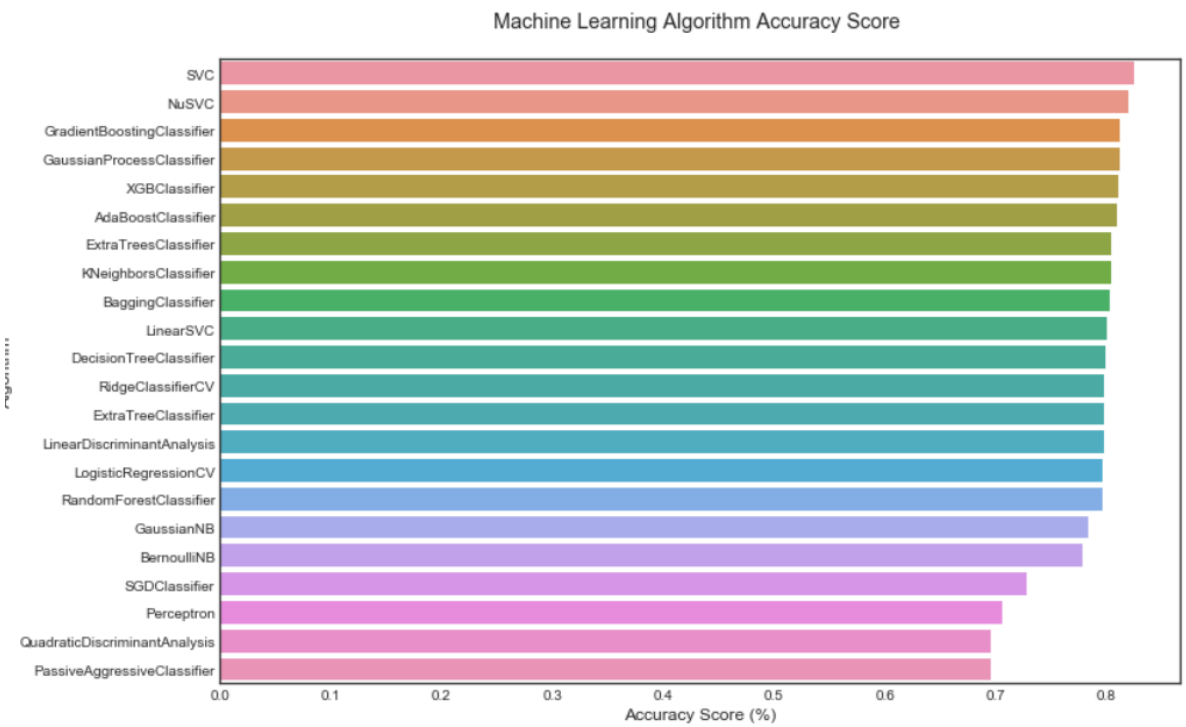
4.1 Model Selection

After finishing the data preparation part, the next job is to build the model. In the following part, I will compare twenty-two model's performance and select some better classifiers to build the final model.

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
14	SVC	{'C': 1.0, 'cache_size': 200, 'class_weight': ...	0.835417	0.825283	0.03824	0.0355513
15	NuSVC	{'cache_size': 200, 'class_weight': None, 'coe...	0.828598	0.819623	0.0350029	0.047182
3	GradientBoostingClassifier	{'criterion': 'friedman_mse', 'init': None, 'l...	0.87178	0.812075	0.0374784	0.0836143
5	GaussianProcessClassifier	{'copy_X_train': True, 'kernel': None, 'max_it...	0.870455	0.811698	0.030566	0.225155
21	XGBClassifier	{'base_score': 0.5, 'booster': 'gbtree', 'cols...	0.858523	0.810566	0.0452264	0.0401805
0	AdaBoostClassifier	{'algorithm': 'SAMME.R', 'base_estimator': Non...	0.828788	0.809811	0.025415	0.0682071
2	ExtraTreesClassifier	{'bootstrap': False, 'class_weight': None, 'cr...	0.897917	0.804528	0.0302923	0.0145634
13	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.850379	0.803774	0.0501191	0.00250652
1	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.893182	0.803019	0.0394768	0.0171609

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
16	LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True,...	0.813258	0.8	0.0447134	0.0357076
17	DecisionTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.897917	0.799623	0.0445555	0.00180109
8	RidgeClassifierCV	{'alphas': (0.1, 1.0, 10.0), 'class_weight': N...	0.810038	0.798113	0.0412082	0.00271029
18	ExtraTreeClassifier	{'class_weight': None, 'criterion': 'gini', 'm...	0.897917	0.798113	0.054821	0.00130076
19	LinearDiscriminantAnalysis	{'n_components': None, 'priors': None, 'shrink...	0.80928	0.798113	0.0358888	0.00251029
6	LogisticRegressionCV	{'Cs': 10, 'class_weight': None, 'cv': None, '...	0.813068	0.796604	0.0412703	0.0954502
4	RandomForestClassifier	{'bootstrap': True, 'class_weight': None, 'cri...	0.89375	0.796604	0.0245167	0.0169049
12	GaussianNB	{'priors': None}	0.793371	0.783019	0.0386401	0.00171077
11	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.786742	0.778491	0.0320399	0.0019078
9	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.727083	0.727925	0.326323	0.00188544
10	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'eta0'...	0.710227	0.70566	0.23752	0.00210829
20	QuadraticDiscriminantAnalysis	{'priors': None, 'reg_param': 0.0, 'store_cova...	0.716856	0.695849	0.145875	0.00230389
7	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.694508	0.695094	0.355724	0.00169442

The table above shows twenty-two classifiers' performance, the "'MLA Test Accuracy 3*STD" attribute is the classifier's test score multiply triple its standard deviation value, which will show how worst the model can be.



The graph above could clearly represent the performance of different kinds of models. Based on the information above, the final prediction model_set as below:

```
vote_set = [
    #Ensemble Methods: http://scikit-learn.org/stable/modules/ensemble.html
    ('ada', ensemble.AdaBoostClassifier()),
    ('bc', ensemble.BaggingClassifier()),
    ('etc',ensemble.ExtraTreesClassifier()),
    ('gbc', ensemble.GradientBoostingClassifier()),
    ('rfc', ensemble.RandomForestClassifier()),

    #GLM: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    ('lr', linear_model.LogisticRegressionCV()),

    #Navies Bayes: http://scikit-learn.org/stable/modules/naive_bayes.html
    ('bnb', naive_bayes.BernoulliNB()),
    ('gnb', naive_bayes.GaussianNB()),

    #Nearest Neighbor: http://scikit-learn.org/stable/modules/neighbors.html
    ('knn', neighbors.KNeighborsClassifier()),

    #SVM: http://scikit-learn.org/stable/modules/svm.html
    ('svc', svm.SVC(probability=True)),

    #xgboost: http://xgboost.readthedocs.io/en/latest/model.html
    ('xgb', XGBClassifier())

]
```

4.2 Hyper-Parameter Tuning

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. Typical examples include C, kernel and gamma for Support Vector Classifier, alpha for Lasso, etc. GridSearchCV exhaustively considers all parameter combinations, in this way, the parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid. So that It is possible and recommended to search the hyper-parameter space for the best cross validation score by GridSearchCV method.

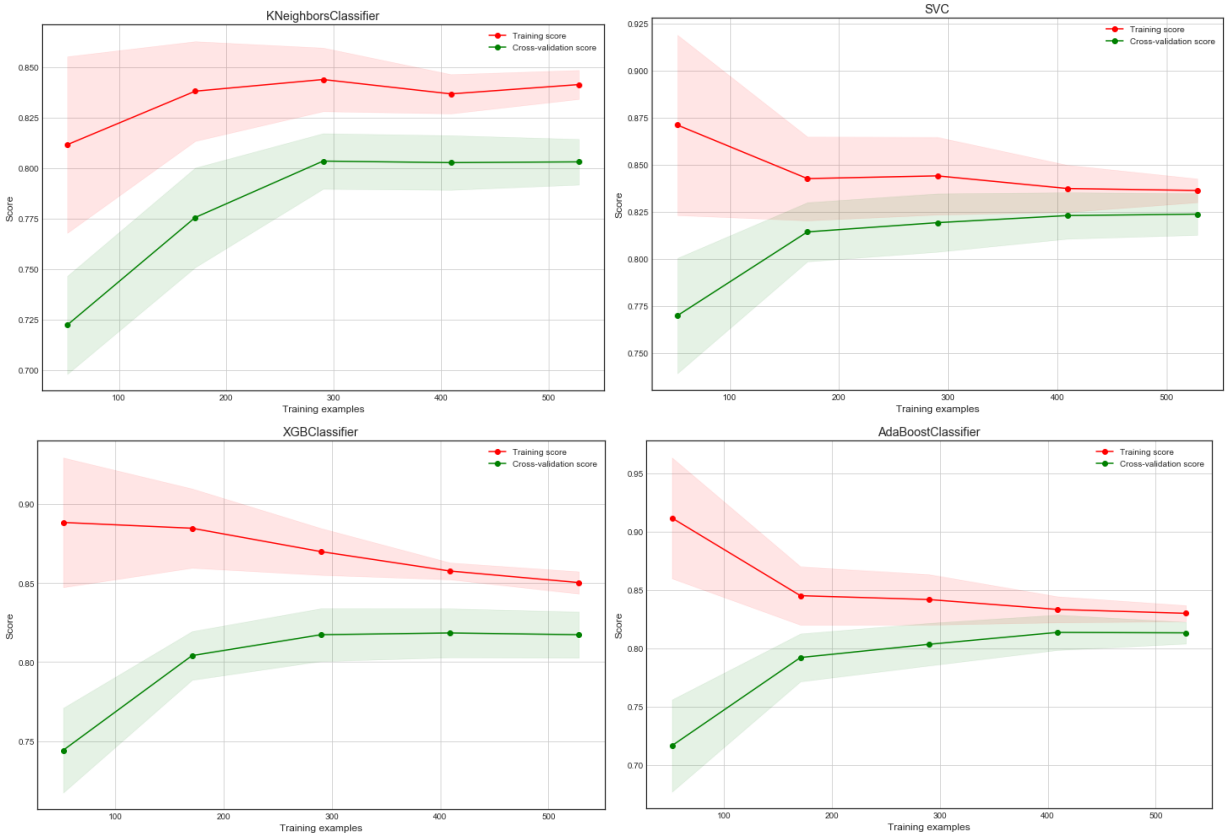
```
grid_n_estimator = [10, 50, 100, 300]
grid_ratio = [.1, .25, .5, .75, 1.0]
grid_learn = [.01, .03, .05, .1, .25]
grid_max_depth = [2, 4, 6, 8, 10, None]
grid_min_samples = [5, 10, .03, .05, .10]
grid_criterion = ['gini', 'entropy']
grid_bool = [True, False]
grid_seed = [0]


grid_param = [
            [{
            #AdaBoostClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
            'n_estimators': grid_n_estimator, #default=50
            'learning_rate': grid_learn, #default=1
            #'algorithm': ['SAMME', 'SAMME.R'], #default='SAMME.R
            'random_state': grid_seed
            }],


            [{
            #BaggingClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sk
            'n_estimators': grid_n_estimator, #default=10
            'max_samples': grid_ratio, #default=1.0
            'random_state': grid_seed
             }],
```

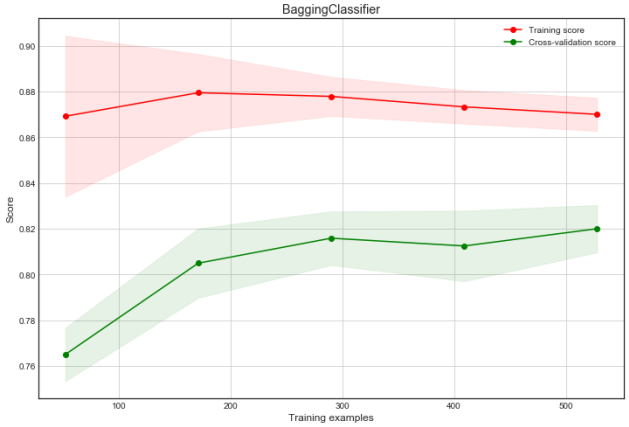
```
 [{
 #ExtraTreesClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.ht
 'n_estimators': grid_n_estimator, #default=10
 'criterion': grid_criterion, #default="gini"
 'max_depth': grid_max_depth, #default=None
 'random_state': grid_seed
  }],


 [{
 #GradientBoostingClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoosting
 #'loss': ['deviance', 'exponential'], #default='deviance'
 'learning_rate': [.05], #default=0.1 -- 12/31/17 set to reduce runtime -- The best parameter for GradientBoostin
 'n_estimators': [300], #default=100 -- 12/31/17 set to reduce runtime -- The best parameter for GradientBoosting
 #'criterion': ['friedman_mse', 'mse', 'mae'], #default="friedman_mse"
 'max_depth': grid_max_depth, #default=3
 'random_state': grid_seed
  }],


 [{
 #RandomForestClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifi
 'n_estimators': grid_n_estimator, #default=10
 'criterion': grid_criterion, #default="gini"
 'max_depth': grid_max_depth, #default=None
 'oob_score': [True], #default=False -- 12/31/17 set to reduce runtime -- The best parameter for RandomForestClas
 'random_state': grid_seed
  }],
```

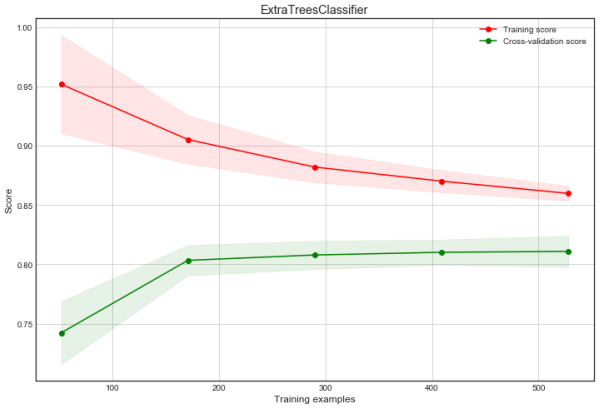
```
for clf, param in zip(vote_set, grid_param):
    best_search = model_selection.GridSearchCV(estimator=clf[1], param_grid= param , cv = cv_split, scoring = 'roc_auc')
    best_search.fit(X_train, Y_train)
#     gsRFC = GridSearchCV(RFC,param_grid = rf_param_grid, cv=kfold, scoring="accuracy", n_jobs= 4, verbose = 1)
#     gsRFC.fit(X_train,Y_train)
#     RFC_best = gsRFC.best_estimator_
#     this is a way to apply the best parameter
    best_param = best_search.best_params_
    print(" the best param for {} is {} ".format(clf[1].__class__.__name__, best_param))
    print("**********************************************")
    # this is another way to apply
    clf[1].set_params(**best_param)
```

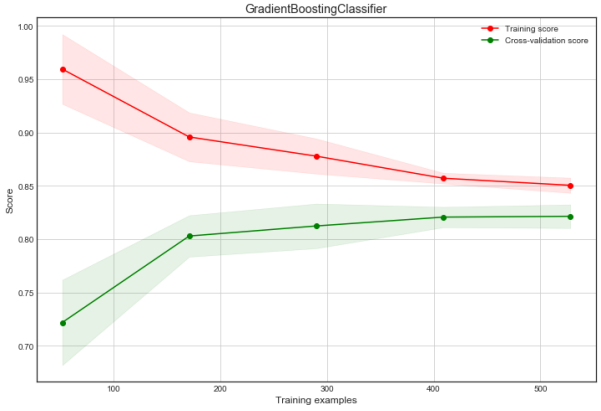
4.3 Model Performance

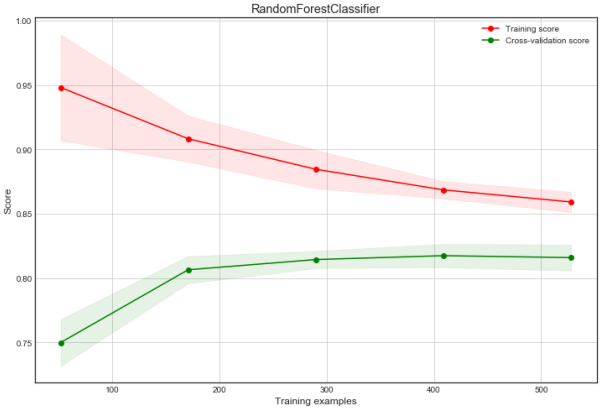
The graph below is the Learning Curve method of every classifiers. Learning curve shows the validation and training score of an estimator for varying numbers of training samples. It is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error. If both the validation score and the training score converge to a value that is too low with increasing size of the training set, we will not benefit much from more training data.

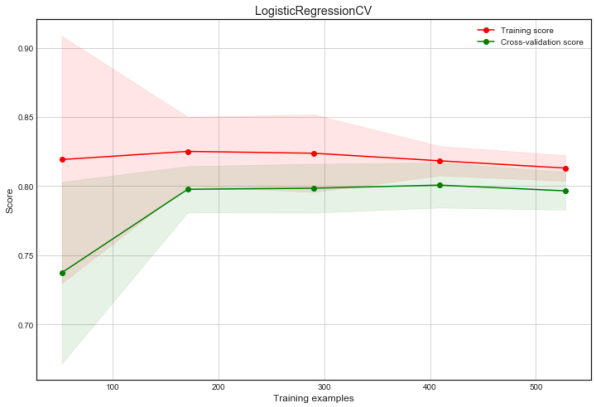


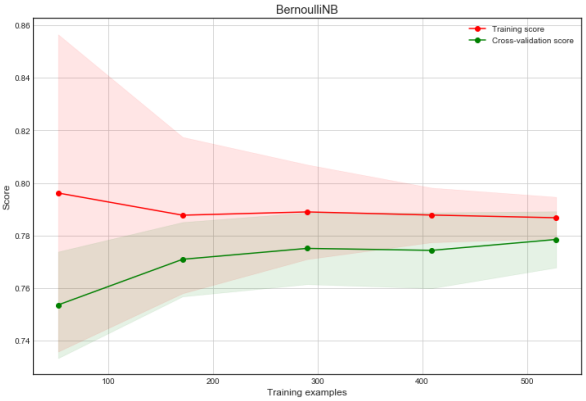

BaggingClassifier
Training score
Cross-validation score
Score
Training examples

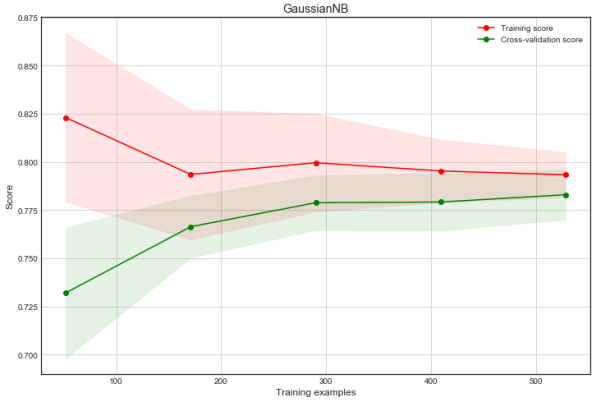

ExtraTreesClassifier
Training score
Cross-validation score
Score
Training examples


GradientBoostingClassifier
Training score
Cross-validation score
Score
Training examples


RandomForestClassifier
Training score
Cross-validation score
Score
Training examples


LogisticRegressionCV
Training score
Cross-validation score
Score
Training examples


BernoulliNB
Training score
Cross-validation score
Score
Training examples


GaussianNB
Training score
Cross-validation score
Score
Training examples

According to Random Forest Classifier, SVC, AdaBoost Classifier, LogisticRegressionCV, BernoulliNB and GaussianNB Classifier, those classifiers converged at some point, which represent the model will gain less benefit from increasing samples. The BenoulliNB and GaussianNB converged at the low score in the graph, which represent the high bias error, show that those model may suffer the underfitting problem.The ExtraTreesClassifier, BaggingClassifier and ExtraTreesClassifier doesn't converge to a score, and the training score always better than the testing score, which represent those classifiers suffer the high variance problem, this phenomenon shows that those classifiers have overfitting problem.

4.4 Ensemble Classifier

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Here, the VotingClassifier combines all eleven classifiers together by voting to do the prediction.

```
# compare the hard and soft voting method.

grid_hard = ensemble.VotingClassifier(estimators = vote_set , voting = 'hard')
grid_hard_cv = model_selection.cross_validate(grid_hard, X_train,Y_train, cv  = cv_split)
grid_hard.fit(X_train,Y_train)

print("Hard Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_hard_cv['train_score'].mean()*100)
print("Hard Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_hard_cv['test_score'].mean()*100))
print("Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_hard_cv['test_score'].std()*100*3
print('-'*10)

#Soft Vote or weighted probabilities w/Tuned Hyperparameters
grid_soft = ensemble.VotingClassifier(estimators = vote_set , voting = 'soft')
grid_soft_cv = model_selection.cross_validate(grid_soft, X_train,Y_train, cv  = cv_split)
grid_soft.fit(X_train,Y_train)

print("Soft Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_soft_cv['train_score'].mean()*100)
print("Soft Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_soft_cv['test_score'].mean()*100))
print("Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_soft_cv['test_score'].std()*100*3
print('-'*10)
```

There are two different methods to do the voting, soft method and hard method. If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.

```
Hard Voting w/Tuned Hyperparameters Training w/bin score mean: 84.56
Hard Voting w/Tuned Hyperparameters Test w/bin score mean: 82.23
Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 3.10
----------
Soft Voting w/Tuned Hyperparameters Training w/bin score mean: 84.19
Soft Voting w/Tuned Hyperparameters Test w/bin score mean: 81.77
Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 2.63
----------
```

After comparing two method, the hard voting is better.

4.5 Prediction

The final part is to do the prediction by the ensemble classifier, the result is as follow:

```
prediction = grid_hard.predict(test)
```

```
output = pd.DataFrame({'PassengerId': data_val['PassengerId'], "Survived": prediction})
output.to_csv('prediction2.csv', index=False)
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	1
5	897	0
6	898	1
7	899	0
8	900	1
9	901	0
10	902	0

5. Conclusion

Based on the analysis information above, the attribute “Sex”, “Pclass” and “Title” have more impact on the final target, female, upper class level and big family member have better survived chance. The attribute “Embarked”, “Age” have few impact to the survival rate, which means those attributes are not determinant element towards final survival rate.