Titanic

May 23, 2019

1 Hwk 3 - Titanic

Name: Troy Zhongyi Zhang
Netid: zhongyiz@uchicago.edu

1.1 Data Pre-processing

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        df1 = pd.read_csv("train.csv", header = None)
        titanic = df1.drop([0, 3,8,9,10],axis=1)
```

```
In [2]: #titanic
        #titanic.iloc[1,3] -> 0 -> str
        #titanic.iloc[6,3] -> Nan -> float
```

```
In [3]: #titanic = titanic.dropna(axis=0)
        #df['Embarked'] = df['Embarked'].fillna('S')
        #df['family'] = df['sbisp']+df['farch']
        #df = df.drop(['sibsp','parch'],axis=1)
        #test['Fare']=test['Fare'].fillna(test['Fare'].mean())
```

```
In [4]: #drop titles
        titanic = titanic.drop([0])
```

```
In [5]: #change age column data type from string to float
        titanic[5]=titanic[5].replace('',np.nan).astype(float)
```

```
In [6]: #test
        #type(titanic.iloc[1,3])
```

```
In [7]: from statistics import mean
        titanic[5] = titanic[5].fillna((titanic[5].mean()))
```

```
In [8]: print(titanic[5].mean())
```

```
29.699117647058763
```

In [9]:
```
#titanic
```

In [10]:
```
titanic.shape
```

Out[10]: (891, 7)

In [11]:
```
titanic = titanic.dropna(axis=0)
```

In [12]:
```
#check
#print(titanic.iloc[45:50])
```

In [13]:
```
titanic.shape
```

Out[13]: (889, 7)

In [14]:
```
titanic['3'] = pd.factorize(titanic[4])[0]
```

In [15]:
```
titanic['8'] = pd.factorize(titanic[11])[0]
titanic = titanic.drop([11],axis=1)
titanic = titanic.drop([4],axis=1)
#print (titanic)
```

In [16]:
```
tita = titanic[[1,2,'3',5,6,7,'8']]
```

In [17]:
```
#tita
```

In [18]:
```
tita.columns = ['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked']
```

In [19]:
```
tita
```

Out[19]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
1	0	3	0	22.000000	1	0	0
2	1	1	1	38.000000	1	0	1
3	1	3	1	26.000000	0	0	0
4	1	1	1	35.000000	1	0	0
5	0	3	0	35.000000	0	0	0
6	0	3	0	29.699118	0	0	2
7	0	1	0	54.000000	0	0	0
8	0	3	0	2.000000	3	1	0
9	1	3	1	27.000000	0	2	0
10	1	2	1	14.000000	1	0	1
11	1	3	1	4.000000	1	1	0
12	1	1	1	58.000000	0	0	0
13	0	3	0	20.000000	0	0	0
14	0	3	0	39.000000	1	5	0
15	0	3	1	14.000000	0	0	0
16	1	2	1	55.000000	0	0	0
17	0	3	0	2.000000	4	1	2
18	1	2	0	29.699118	0	0	0
19	0	3	1	31.000000	1	0	0

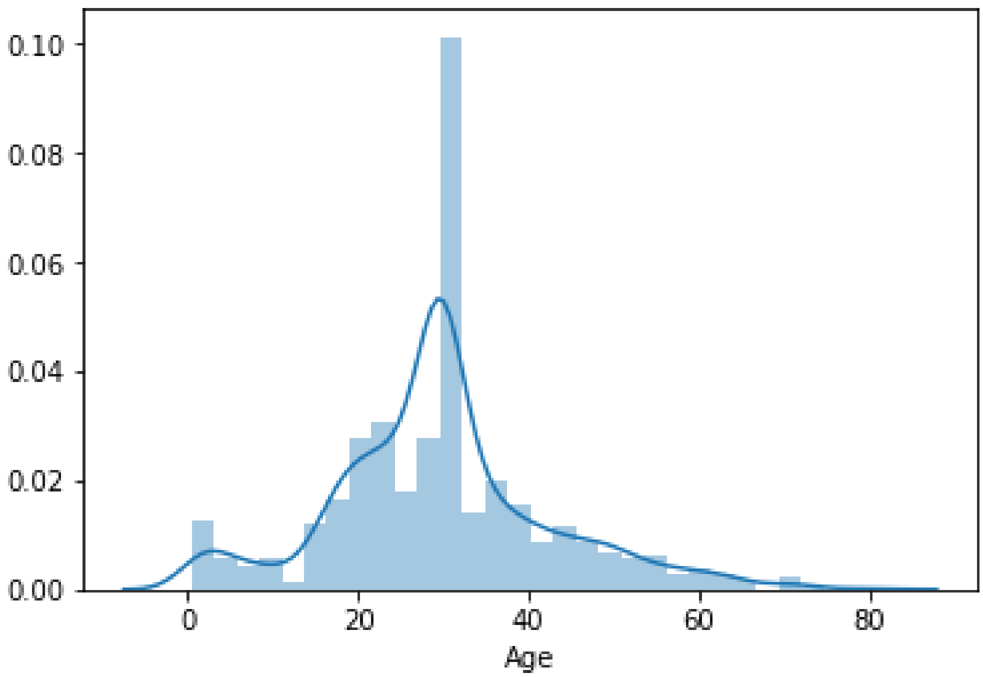
```
20          1        3    1  29.699118       0       0           1
21          0        2    0  35.000000       0       0           0
22          1        2    0  34.000000       0       0           0
23          1        3    1  15.000000       0       0           2
24          1        1    0  28.000000       0       0           0
25          0        3    1   8.000000       3       1           0
26          1        3    1  38.000000       1       5           0
27          0        3    0  29.699118       0       0           1
28          0        1    0  19.000000       3       2           0
29          1        3    1  29.699118       0       0           2
30          0        3    0  29.699118       0       0           0
..        ...      ...  ...        ...     ...     ...         ...
862         0        2    0  21.000000       1       0           0
863         1        1    1  48.000000       0       0           0
864         0        3    1  29.699118       8       2           0
865         0        2    0  24.000000       0       0           0
866         1        2    1  42.000000       0       0           0
867         1        2    1  27.000000       1       0           1
868         0        1    0  31.000000       0       0           0
869         0        3    0  29.699118       0       0           0
870         1        3    0   4.000000       1       1           0
871         0        3    0  26.000000       0       0           0
872         1        1    1  47.000000       1       1           0
873         0        1    0  33.000000       0       0           0
874         0        3    0  47.000000       0       0           0
875         1        2    1  28.000000       1       0           1
876         1        3    1  15.000000       0       0           1
877         0        3    0  20.000000       0       0           0
878         0        3    0  19.000000       0       0           0
879         0        3    0  29.699118       0       0           0
880         1        1    1  56.000000       0       1           1
881         1        2    1  25.000000       0       1           0
882         0        3    0  33.000000       0       0           0
883         0        3    1  22.000000       0       0           0
884         0        2    0  28.000000       0       0           0
885         0        3    0  25.000000       0       0           0
886         0        3    1  39.000000       0       5           2
887         0        2    0  27.000000       0       0           0
888         1        1    1  19.000000       0       0           0
889         0        3    1  29.699118       1       2           0
890         1        1    0  26.000000       0       0           1
891         0        3    0  32.000000       0       0           2

[889 rows x 7 columns]
```

In [20]:
```python
import seaborn as sns
sns.distplot(tita['Age'])
```

```
/Users/zhongyizhang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarn
```

```
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23ed68d0>



1.2 Cleaning holdout_test

In [21]:
```
df2 = pd.read_csv("holdout_test.csv", header = None)
test = df2.drop([1, 3,8,9,10],axis=1)
```

In [22]:
```
test = test.drop([0])
test[5]=test[5].replace('',np.nan).astype(float)
test[5] = test[5].fillna((test[5].mean()))
#test = test.dropna(axis=0)
test['3'] = pd.factorize(test[4])[0]
test['8'] = pd.factorize(test[11])[0]
test = test.drop([11],axis=1)
test = test.drop([4],axis=1)
#print (titanic)
test = test[[0,2,'3',5,6,7,'8']]
test.columns = ['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked']
test
```

Out[22]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
1	NaN	3	0	34.50000	0	0	0

2	NaN	3	1	47.00000	1	0	1
3	NaN	2	0	62.00000	0	0	0
4	NaN	3	0	27.00000	0	0	1
5	NaN	3	1	22.00000	1	1	1
6	NaN	3	0	14.00000	0	0	1
7	NaN	3	1	30.00000	0	0	0
8	NaN	2	0	26.00000	1	1	1
9	NaN	3	1	18.00000	0	0	2
10	NaN	3	0	21.00000	2	0	1
11	NaN	3	0	30.27259	0	0	1
12	NaN	1	0	46.00000	0	0	1
13	NaN	1	1	23.00000	1	0	1
14	NaN	2	0	63.00000	1	0	1
15	NaN	1	1	47.00000	1	0	1
16	NaN	2	1	24.00000	1	0	2
17	NaN	2	0	35.00000	0	0	0
18	NaN	3	0	21.00000	0	0	2
19	NaN	3	1	27.00000	1	0	1
20	NaN	3	1	45.00000	0	0	2
21	NaN	1	0	55.00000	1	0	2
22	NaN	3	0	9.00000	0	1	1
23	NaN	1	1	30.27259	0	0	1
24	NaN	1	0	21.00000	0	1	2
25	NaN	1	1	48.00000	1	3	2
26	NaN	3	0	50.00000	1	0	1
27	NaN	1	1	22.00000	0	1	2
28	NaN	3	0	22.50000	0	0	2
29	NaN	1	0	41.00000	0	0	1
30	NaN	3	0	30.27259	2	0	2
..	...	...	...	...	...	...	...
389	NaN	3	0	21.00000	0	0	0
390	NaN	3	0	6.00000	3	1	1
391	NaN	1	0	23.00000	0	0	1
392	NaN	1	1	51.00000	0	1	1
393	NaN	3	0	13.00000	0	2	1
394	NaN	2	0	47.00000	0	0	1
395	NaN	3	0	29.00000	3	1	1
396	NaN	1	1	18.00000	1	0	1
397	NaN	3	0	24.00000	0	0	0
398	NaN	1	1	48.00000	1	1	2
399	NaN	3	0	22.00000	0	0	1
400	NaN	3	0	31.00000	0	0	0
401	NaN	1	1	30.00000	0	0	1
402	NaN	2	0	38.00000	1	0	1
403	NaN	1	1	22.00000	0	1	2
404	NaN	1	0	17.00000	0	0	1
405	NaN	1	0	43.00000	1	0	2
406	NaN	2	0	20.00000	0	0	2

```
407      NaN      2    0   23.00000     1      0          1
408      NaN      1    0   50.00000     1      1          2
409      NaN      3    1   30.27259     0      0          0
410      NaN      3    1    3.00000     1      1          1
411      NaN      3    1   30.27259     0      0          0
412      NaN      1    1   37.00000     1      0          0
413      NaN      3    1   28.00000     0      0          1
414      NaN      3    0   30.27259     0      0          1
415      NaN      1    1   39.00000     0      0          2
416      NaN      3    0   38.50000     0      0          1
417      NaN      3    0   30.27259     0      0          1
418      NaN      3    0   30.27259     1      1          2

[418 rows x 7 columns]
```

1.3 Modeling

1.3.1 Decision Tree

```
In [23]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import accuracy_score
         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import VotingClassifier

         from sklearn import linear_model
         from sklearn.linear_model import LogisticRegression
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.naive_bayes import GaussianNB
         from sklearn.svm import SVC, LinearSVC
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import AdaBoostClassifier
         from sklearn.ensemble import GradientBoostingClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.linear_model import SGDClassifier
         from sklearn.linear_model import Perceptron

         from sklearn.metrics import mean_squared_error as MSE
```

```
/Users/zhongyizhang/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:2
  from numpy.core.umath_tests import inner1d
```

```
In [24]: y=tita.iloc[:,0]
         x=tita.iloc[:,1:7]
```

```
In [25]: import matplotlib.pyplot as plt
         from sklearn.metrics import classification_report
         from sklearn.model_selection import cross_val_score
```

```
import sklearn.model_selection as cv
from sklearn.tree import DecisionTreeRegressor

(x_train, x_test, y_train, y_test) = cv.train_test_split(x, y, test_size=.20)
# Instantiate dt
dt = DecisionTreeRegressor(max_depth=4,
                           min_samples_leaf=0.11,
                           random_state=3)

# Fit dt to the training set
dt.fit(x_train, y_train)
```

Out[25]:
```
DecisionTreeRegressor(criterion='mse', max_depth=4, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=0.11,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=3, splitter='best')
```

In [26]:
```
from sklearn.metrics import mean_squared_error as MSE

# Compute y_pred
y_pred_dt = dt.predict(x_test)

# Compute mse_dt
mse_dt = MSE(y_test, y_pred_dt)

# Compute rmse_dt
rmse_dt = mse_dt**(1/2)

# Print rmse_dt
print("Test set RMSE of dt: {:.2f}".format(rmse_dt))
```

```
Test set RMSE of dt: 0.39
```

In [27]:
```
from sklearn.model_selection import cross_val_score
# Compute the array containing the 10-folds CV MSEs
MSE_CV_scores = - cross_val_score(dt, x_train, y_train, cv=10,
                                  scoring='neg_mean_squared_error',
                                  n_jobs=-1)

# Compute the 10-folds CV RMSE
RMSE_CV = (MSE_CV_scores.mean())**(1/2)

# Print RMSE_CV
print('CV RMSE: {:.2f}'.format(RMSE_CV))
```

```
CV RMSE: 0.38
```

In [28]:
```
# Import mean_squared_error from sklearn.metrics as MSE
from sklearn.metrics import mean_squared_error as MSE

# Fit dt to the training set
dt.fit(x_train, y_train)

# Predict the labels of the training set
y_pred_train = dt.predict(x_train)

# Evaluate the training set RMSE of dt
RMSE_train = (MSE(y_train, y_pred_train))**(1/2)

# Print RMSE_train
print('Train RMSE: {:.2f}'.format(RMSE_train))
```

```
Train RMSE: 0.37
```

In [29]:
```
# Import train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
# Set SEED for reproducibility
SEED = 1

# Split the data into 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=

# Instantiate a DecisionTreeRegressor dt
dt = DecisionTreeRegressor(max_depth=4, min_samples_leaf=0.26, random_state=SEED)


# Compute the array containing the 10-folds CV MSEs
MSE_CV_scores = - cross_val_score(dt, X_train, y_train, cv=10,
                                  scoring='neg_mean_squared_error',
                                  n_jobs=-1)

# Compute the 10-folds CV RMSE
RMSE_CV = (MSE_CV_scores.mean())**(1/2)

# Print RMSE_CV
print('CV RMSE: {:.2f}'.format(RMSE_CV))
```

```
CV RMSE: 0.42
```

In [30]:
```
# Import mean_squared_error from sklearn.metrics as MSE
from sklearn.metrics import mean_squared_error as MSE

# Fit dt to the training set
```

```
    dt.fit(X_train, y_train)

    # Predict the labels of the training set
    y_pred_train = dt.predict(X_train)

    # Evaluate the training set RMSE of dt
    RMSE_train = (MSE(y_train, y_pred_train))**(1/2)

    # Print RMSE_train
    print('Train RMSE: {:.2f}'.format(RMSE_train))
```

```
Train RMSE: 0.42
```

```
In [31]: # Import train_test_split from sklearn.model_selection
         from sklearn.model_selection import train_test_split

         # Set SEED for reproducibility
         SEED = 1

         # Split the data into 70% train and 30% test
         X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=

         # Instantiate a DecisionTreeRegressor dt
         dt = DecisionTreeRegressor(max_depth= 3 , min_samples_leaf= 0.26 , random_state=SEED)
```

```
In [32]: decision_tree = DecisionTreeClassifier()
         decision_tree_model = decision_tree.fit(X_train, y_train)
         decision_tree_Y_pred = decision_tree.predict(X_test)

         acc_decision_tree = round(decision_tree.score(X_train, y_train) * 100, 2)
         acc_decision_tree_test = round(decision_tree.score(X_test, y_test) * 100, 2)
         print('Decision Tree training set accuracy score:',acc_decision_tree,'%')
         print('Decision Tree testing set accuracy score:',acc_decision_tree_test,'%')
```

```
Decision Tree training set accuracy score: 94.05 %
Decision Tree testing set accuracy score: 79.78 %
```

1.3.2 Random Forest

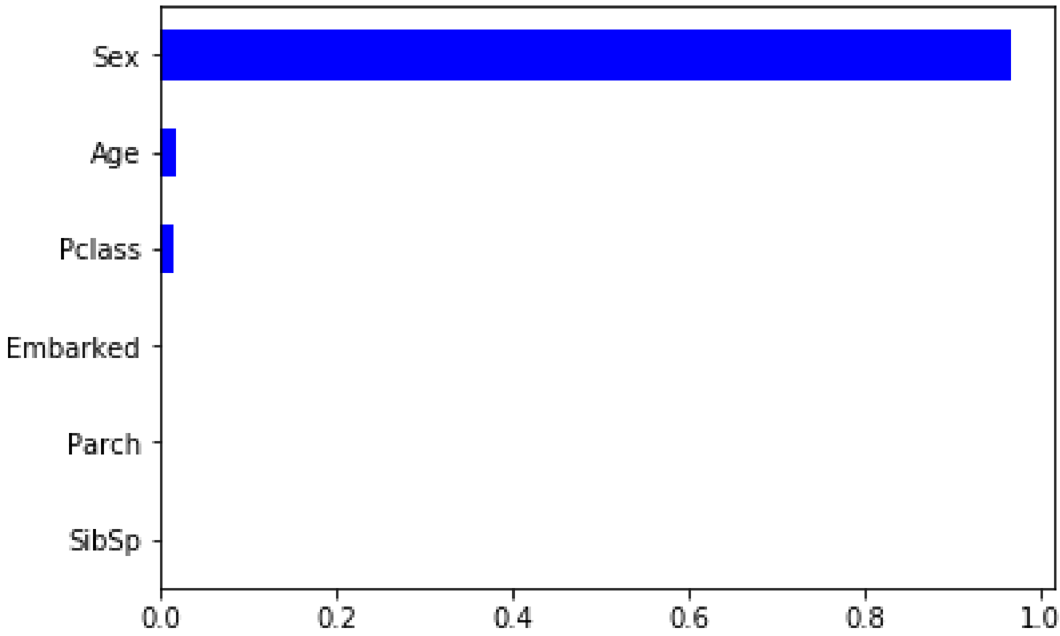
```
In [33]: # Basic imports
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import mean_squared_error as MSE

         y=tita.iloc[:,0]
         X=tita.iloc[:,1:7]
         X_train, X_test, y_train, y_test = \
```

```
train_test_split(X, y,
test_size=0.3,
random_state=SEED)
```

In [34]:
```
# Instantiate a random forests regressor 'rf' 400 estimators
rf = RandomForestRegressor(n_estimators=400,
min_samples_leaf=0.16,
random_state=SEED)
# Fit 'rf' to the training set
rf.fit(X_train, y_train)
# Predict the test set labels 'y_pred'
y_pred = rf.predict(X_test)
y_pred_train=rf.predict(X_train)
# Evaluate the test set RMSE
rmse_test = MSE(y_test, y_pred)**(1/2)
rmse_train = MSE(y_train, y_pred_train)**(1/2)
# Print the test set RMSE
print('Test set RMSE of rf: {:.4f}'.format(rmse_test))
print('Train set RMSE of rf: {:.4f}'.format(rmse_train))
```

```
Test set RMSE of rf: 0.3843
Train set RMSE of rf: 0.4154
```

In [35]:
```
# Create a pd.Series of features importances
importances_rf = pd.Series(rf.feature_importances_,
index = X.columns)
# Sort importances_rf
sorted_importances_rf = importances_rf.sort_values()
# Make a horizontal bar plot
sorted_importances_rf.plot(kind='barh', color='blue')
plt.show()
```



In [36]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100,
min_samples_leaf=0.12,
random_state=SEED)
rf.fit(X_train, y_train)
y_pred = rf.predict_proba(X_test)[:,1]
```

In [37]:
```
y_test=y_test.astype(float)
y_pred_rf=y_pred.astype(float)
```

In [38]:
```
#y_pred_proba_rf = rf.predict_proba(X_test)[:,1]
# Import roc_auc_score
from sklearn.metrics import roc_auc_score

# Evaluate test-set roc_auc_score
rf_roc_auc = roc_auc_score(y_test, y_pred_rf)

# Print roc_auc_score
print('ROC AUC score: {:.4f}'.format(rf_roc_auc))
```

```
ROC AUC score: 0.8786
```

In [39]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=
```

In [40]:
```
random_forest = RandomForestClassifier(n_estimators=400,
min_samples_leaf=0.011,
```

```
                random_state=SEED)
random_forest_model = random_forest.fit(X_train, y_train)

random_Y_pred = random_forest.predict(X_test)

random_forest.score(X_train, y_train)
acc_random_forest = round(random_forest.score(X_train, y_train) * 100, 2)

acc_random_forest_test = round(random_forest.score(X_test, y_test) * 100, 2)

print('Random Forest training set accuracy score:',acc_random_forest,'%')
print('Random Forest testing set accuracy score:',acc_random_forest_test,'%')
```

```
Random Forest training set accuracy score: 84.08 %
Random Forest testing set accuracy score: 85.02 %
```

1.3.3 AdaBoost

```
In [41]: dt = DecisionTreeClassifier(max_depth=5, random_state=1)

         # Instantiate ada
         ada = AdaBoostClassifier(base_estimator=dt, n_estimators=140, random_state=1)

In [42]: ada.fit(X_train, y_train)

         # Compute the probabilities of obtaining the positive class
         y_pred = ada.predict_proba(X_test)[:,1]

In [43]: y_test=y_test.astype(float)
         y_pred_ada=y_pred.astype(float)

In [44]: # Import roc_jauc_score
         from sklearn.metrics import roc_auc_score

         # Evaluate test-set roc_auc_score
         ada_roc_auc = roc_auc_score(y_test, y_pred_ada)

         # Print roc_auc_score
         print('ROC AUC score: {:.4f}'.format(ada_roc_auc))
```

```
ROC AUC score: 0.8563
```

```
In [45]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state

In [46]: decision_tree = DecisionTreeClassifier()
         ada = AdaBoostClassifier(base_estimator=decision_tree, n_estimators=180, random_state
         ada_model = ada.fit(X_train, y_train)
```

```
ada_Y_pred = ada.predict(X_test)

acc_ada = round(ada.score(X_train, y_train) * 100, 2)
acc_ada_test = round(ada.score(X_test, y_test) * 100, 2)

print('Random Forest training set accuracy score:',acc_ada,'%')
print('Random Forest testing set accuracy score:',acc_ada_test,'%')
```

```
Random Forest training set accuracy score: 94.05 %
Random Forest testing set accuracy score: 80.9 %
```

1.3.4 SVC

```
In [47]: import sklearn as skl
         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.datasets import make_classification
         from sklearn import preprocessing
         from sklearn import svm
         from sklearn.model_selection import GridSearchCV
         from sklearn.pipeline import Pipeline
         from sklearn import metrics
         from tempfile import mkdtemp
         from shutil import rmtree
         from sklearn.externals.joblib import Memory

         svc = SVC(kernel='linear', C=1)
         svc.fit(X_train, y_train)
         y_pred = svc.predict(X_test)
         acc_svc = round(svc.score(X_train, y_train) * 100, 2)
         print('SVC training set accuracy score:',acc_svc,'%')

         acc_svc_test = round(svc.score(X_test, y_test) * 100, 2)
         print('SVC testing set accuracy score:',acc_svc_test,'%')

         #X_test = X_test.astype(str)
         #y_test = y_test.astype(str)
         #y_pred_svc = y_pred.astype(float)
         #svc_roc_auc = roc_auc_score(y_test, y_pred_svc)

         #print('ROC AUC score: {:.4f}'.format(svc_roc_auc))
```

```
SVC training set accuracy score: 77.17 %
SVC testing set accuracy score: 82.02 %
```

1.3.5 Stochastic Gradient Descent

```
In [48]: sgd = SGDClassifier(max_iter=5, tol=None)
         sgd_model = sgd.fit(X_train, y_train)
         sgd_Y_pred = sgd.predict(X_test)
         y_pred = sgd_Y_pred
         sgd.score(X_train, y_train)

         acc_sgd = round(sgd.score(X_train, y_train) * 100, 2)
         print('SGD training set accuracy score:',acc_sgd,'%')

         acc_sgd_test = round(sgd.score(X_test, y_test) * 100, 2)
         print('SGD testing set accuracy score:',acc_sgd_test,'%')

         #y_test = y_test.astype(float)
         #y_pred = y_pred.astype(float)
         #sgd_roc_auc = roc_auc_score(y_test, y_pred)

         #print('ROC AUC score: {:.4f}'.format(sgd_roc_auc))
```

```
SGD training set accuracy score: 74.6 %
SGD testing set accuracy score: 76.78 %
```

1.3.6 Logistic Regression

```
In [49]: logreg = LogisticRegression(solver='lbfgs')
         log_model = logreg.fit(X_train, y_train)

         log_y_pred = logreg.predict(X_test)

         acc_log = round(logreg.score(X_train, y_train) * 100, 2)
         acc_log_test = round(logreg.score(X_test, y_test) * 100, 2)
         print('Logistic regression training set accuracy score:',acc_log,'%')
         print('Logistic regression testing set accuracy score:',acc_log_test,'%')


         #y_test = y_test.astype(float)
         #y_pred = log_y_pred.astype(float)
         #lr_roc_auc = roc_auc_score(y_test, y_pred)
         #print('ROC AUC score: {:.4f}'.format(lr_roc_auc))
```

```
Logistic regression training set accuracy score: 78.62 %
Logistic regression testing set accuracy score: 83.9 %
```

1.3.7 K-nearest Neighbors:

```
In [50]: knn = KNeighborsClassifier(n_neighbors = 3)
         knn_model = knn.fit(X_train, y_train)
```

```
knn_y_pred = knn.predict(X_test)

acc_knn = round(knn.score(X_train, y_train) * 100, 2)
print('K-nearest Neighbors training set accuracy score:',acc_knn,'%')
acc_knn_test = round(knn.score(X_test, y_test) * 100, 2)
print('K-nearest Neighbors testing set accuracy score:',acc_knn_test,'%')

#y_test = y_test.astype(float)
#y_pred = knn_y_pred.astype(float)
#knn_roc_auc = roc_auc_score(y_test, y_pred)
#print('ROC AUC score: {:.4f}'.format(knn_roc_auc))
```

```
K-nearest Neighbors training set accuracy score: 86.98 %
K-nearest Neighbors testing set accuracy score: 76.4 %
```

1.3.8 Gaussian Naive Bayes

In [51]:
```
gaussian = GaussianNB()
gaussian_model = gaussian.fit(X_train, y_train)

gaussian_y_pred = gaussian.predict(X_test)

acc_gaussian = round(gaussian.score(X_train, y_train) * 100, 2)
print('Gaussian Naive Bayes training set accuracy score:',acc_gaussian,'%')
acc_gaussian_test = round(gaussian.score(X_test, y_test) * 100, 2)
print('Gaussian Naive Bayes testing set accuracy score:',acc_gaussian_test,'%')

#y_test = y_test.astype(float)
#y_pred = gaussian_y_pred.astype(float)
#gaussian_roc_auc = roc_auc_score(y_test, y_pred)
#testscore = round(gaussian.score(X_test, y_test) * 100, 2)
#print(testscore)
#print('ROC AUC score: {:.4f}'.format(gaussian_roc_auc))
```

```
Gaussian Naive Bayes training set accuracy score: 78.3 %
Gaussian Naive Bayes testing set accuracy score: 81.27 %
```

1.3.9 Summary accuracy for each model

In [52]:
```
print('Accuracy Scores:', '\n',
'Logistic Regression:', round(acc_log_test,2), '%', '\n',
'Decision Tree:', round(acc_decision_tree_test,2), '%', '\n',
'Random Forest:', round(acc_random_forest_test,2), '%','\n',
'Ada boost:', round(acc_ada_test, 2), '%','\n',
'SVC:', round(acc_svc_test,2), '%','\n',
'Stochastic Gradient Descent:', round(acc_sgd_test,2), '%','\n',
```

```
        'K-nearest Neighbors:', round(acc_knn_test,2), '%','\n',
        'Gaussian Naive Bayes:', round(acc_gaussian_test,2), '%' )
```

```
Accuracy Scores:
 Logistic Regression: 83.9 %
 Decision Tree: 79.78 %
 Random Forest: 85.02 %
 Ada boost: 80.9 %
 SVC: 82.02 %
 Stochastic Gradient Descent: 76.78 %
 K-nearest Neighbors: 76.4 %
 Gaussian Naive Bayes: 81.27 %
```

The random forest gave me the highest accuracy score, which is over 85%. I will choose the random forest for my predictions. In logistic regression, SVC, Stochastic Gradient Descent, Gaussian Naive Bayes, and random forest models, the testing set accuracy score is higher than the training set accuracy score. In K-nearest neighbors, AdaBoost, and decision tree, the testing accuracy score is lower than the training accuracy score. If the CV RMSE is greater than training RMSE, my model will be overfitting, If CV error almost equal to my training error but they both greater than desired error, my model will be underfitting. With 80% and 20% training and testing set division, when Cross Validation = 10, my CV RMSE is 0.38, my Test set RMSE is 0.39, and Train set RMSE is 0.37. They are good, but I can make my model fits better. I tried 70% and 30% training and testing set division. In this case, my both CV RMSE and Train RMSE are 0.42. They are equal, which means my model is not overfitting. Finally, I choose the 0.7 and 0.3 division for my dataset in order to modeling work.

```
In [53]: test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418 entries, 1 to 418
Data columns (total 7 columns):
Survived    0 non-null object
Pclass      418 non-null object
Sex         418 non-null int64
Age         418 non-null float64
SibSp       418 non-null object
Parch       418 non-null object
Embarked    418 non-null int64
dtypes: float64(1), int64(2), object(4)
memory usage: 26.1+ KB
```

```
In [54]: test = test.loc[:,test.columns != "Survived"]
         pred_test = random_forest.predict(test)
         print(pred_test)
```

```
['0' '1' '0' '0' '1' '0' '0' '0' '1' '0' '0' '0' '1' '0' '1' '1' '0' '0'
 '1' '1' '0' '1' '1' '0' '1' '0' '1' '0' '0' '0' '0' '0' '1' '1' '0' '0'
```

```
 '1' '1' '0' '0' '0' '0' '0' '1' '1' '0' '0' '0' '1' '1' '1' '0' '1' '1'
 '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '1' '0' '0' '0' '1' '0' '0'
 '1' '0' '1' '0' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '1' '0' '1'
 '1' '0' '1' '0' '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '1' '0' '0' '0'
 '0' '0' '0' '0' '1' '0' '1' '0' '0' '1' '0' '1' '1' '0' '1' '0' '0' '1'
 '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '0' '0'
 '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '0' '0' '1' '1' '0' '1' '0' '1'
 '1' '0' '0' '1' '0' '0' '1' '1' '0' '0' '0' '0' '0' '1' '1' '0' '1' '1'
 '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '0' '0' '1' '0' '1' '0' '1' '1'
 '0' '1' '0' '1' '0' '1' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0'
 '0' '0' '1' '0' '1' '0' '1' '0' '1' '1' '0' '0' '0' '0' '0' '1' '0' '0'
 '0' '0' '0' '0' '1' '1' '1' '1' '0' '0' '0' '1' '1' '0' '1' '1' '1' '0'
 '1' '0' '0' '0' '0' '0' '1' '0' '0' '0' '1' '1' '0' '0' '0' '0' '1' '0'
 '0' '0' '1' '0' '0' '1' '0' '0' '0' '0' '1' '1' '0' '1' '1' '0' '0' '1'
 '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '1'
 '0' '1' '0' '1' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0'
 '1' '0' '1' '0' '0' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0'
 '0' '1' '0' '1' '0' '1' '0' '1' '1' '0' '0' '0' '1' '0' '1' '0' '0' '1'
 '0' '1' '1' '0' '1' '0' '0' '1' '1' '0' '0' '1' '0' '0' '1' '1' '1' '0'
 '0' '0' '0' '0' '1' '1' '0' '1' '0' '0' '0' '0' '0' '1' '0' '0' '0' '1'
 '0' '1' '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '0' '1' '0' '1' '1' '0'
 '1' '0' '0' '0']
```

```
In [55]: len(pred_test)
```

```
Out[55]: 418
```

```
In [56]: df3 = pd.read_csv("holdout_test.csv", header = None)
         df3 = df3.drop([0])
         df3[0] = pred_test
         df3.columns = ['Survived','PassengerId','Survived','Pclass','Name','Sex','Age',
                        'SibSp','Parch','Ticke','Cabin','Embarked']
```

```
In [58]: df3.to_csv('/Users/zhongyizhang/Desktop/Homework3_finalize/RandomForest_titanic_holdou
```