

Introduction

This kernel was inspired in part by the work of [SarahG](#)'s analysis that I thank very much for the quality of her analysis. This work represents a deeper analysis by playing on several parameters while using only logistic regression estimator. In a future work, I will discuss other techniques. I am open to any criticism and proposal. You do not hesitate to evaluate this analysis. The following kernel contains the steps enumerated below for assessing the Titanic survival dataset:

- 1. [Import data and python packages](#)
- 2. [Assess Data Quality & Missing Values](#)
 - 2.1. [Age - Missing Values](#)
 - 2.2. [Cabin - Missing Values](#)
 - 2.3. [Embarked - Missing Values](#)
 - 2.4. [Final Adjustments to Data](#)
 - 2.4.1 [Additional Variables](#)
- 3. [Exploratory Data Analysis](#)
- 4. [Logistic Regression and Results](#)
 - 4.1. [Feature selection](#)
 - 4.1.1. [Recursive feature elimination](#)
 - 4.1.2. [Feature ranking with recursive feature elimination and cross-validation](#)
 - 4.2. [Review of model evaluation procedures](#)
 - 4.2.1. [Model evaluation based on simple train/test split using `train_test_split\(\)`](#)
 - 4.2.2. [Model evaluation based on K-fold cross-validation using `cross_val_score\(\)`](#)
 - 4.2.3. [Model evaluation based on K-fold cross-validation using `cross_validate\(\)`](#)
 - 4.3. [GridSearchCV evaluating using multiple scorers simultaneously](#)
 - 4.4. [GridSearchCV evaluating using multiple scorers, RepeatedStratifiedKFold and pipeline for preprocessing simultaneously](#)

1. Import Data & Python Packages

```
1 import numpy as np
2 import pandas as pd
3
4 from sklearn import preprocessing
5 import matplotlib.pyplot as plt
6 plt.rc("font", size=14) #dictionary objects
7 import seaborn as sns
8 sns.set(style="white") #white background style for seaborn plots
9 sns.set(style="whitegrid", color_codes=True)
10
11 import warnings
12 warnings.simplefilter(action='ignore')
```

```
1 # Read CSV train data file into DataFrame
2 train_df = pd.read_csv("../input/titanic/train.csv")
3
4 # Read CSV test data file into DataFrame
5 test_df = pd.read_csv("../input/titanic/test.csv")
6
7 # preview train data
8 train_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0		1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1		2	1	1	Cumings, Mrs. John Bradley (Florence Briggs)	female	38.0	1	0	PC 17599	71.2833	C85

```
1 print("The number of samples into the train data is {}".format(train_df.shape[0]))
```

The number of samples into the train data is 891.

```
1 # preview test data
```

```
2 test_df.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S

```
1 print("The number of samples into the test data is {}".format(test_df.shape[0]))
```

The number of samples into the test data is 418.

Note: there is no target variable into test data (i.e. "Survival" column is missing), so the goal is to predict this target using different machine learning algorithms such as logistic regression.

2. Data Quality & Missing Value Assessment

```
1 # check missing values in train data
```

```
2 train_df.isnull().sum()
```

```
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
```

2.1. Age - Missing Values

```
1 # percent of missing "Age"
```

```
2 print('Percent of missing "Age" records is %.2f%%' % ((train_df['Age'].isnull().sum()/train_df.shape[0])*100))
```

Percent of missing "Age" records is 19.87%

~20% of entries for passenger age are missing. Let's see what the 'Age' variable looks like in general.

Since "Age" is (right) skewed, using the mean might give us biased results by filling in ages that are older than desired. To deal with this, we'll use the median to impute the missing values.

```
1 # mean age
```

```
2 print("The mean of "Age" is %.2f" % (train_df["Age"].mean(skipna=True)))
```

```
3 # median age
```

```
4 print("The median of "Age" is %.2f" % (train_df["Age"].median(skipna=True)))
```

The mean of "Age" is 29.70

The median of "Age" is 28.00

Any better ideas for "Age" imputation?

2.2. Cabin - Missing Values

```
1 # percent of missing "Cabin"
2 print('Percent of missing "Cabin" records is %.2f%%' % ((train_df['Cabin'].isnull().sum()/train_df.shape[0])*100))
```

Percent of missing "Cabin" records is 77.10%

77% of records are missing, which means that imputing information and using this variable for prediction is probably not wise. We'll ignore this variable in our model.

2.3. Embarked - Missing Values

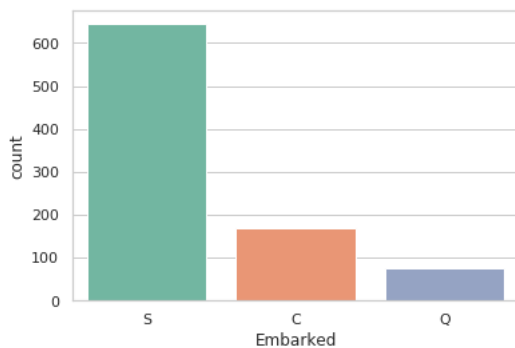
```
1 # percent of missing "Embarked"
2 print('Percent of missing "Embarked" records is %.2f%%' % ((train_df['Embarked'].isnull().sum()/train_df.shape[0])*100))
```

Percent of missing "Embarked" records is 0.22%

There are only 2 (0.22%) missing values for "Embarked", so we can just impute with the port where most people boarded.

```
1 print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):')
2 print(train_df['Embarked'].value_counts())
3 sns.countplot(x='Embarked', data=train_df, palette='Set2')
4 plt.show()
```

Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):
 S 644
 C 168
 Q 77
 Name: Embarked, dtype: int64



```
1 print('The most common boarding port of embarkation is %s.' % train_df['Embarked'].value_counts().idxmax())
```

The most common boarding port of embarkation is S.

By far the most passengers boarded in Southampton, so we'll impute those 2 NaN's w/ "S".


2.4. Final Adjustments to Data (Train & Test)

Based on my assessment of the missing values in the dataset, I'll make the following changes to the data:

- If "Age" is missing for a given row, I'll impute with 28 (median age).
- If "Embarked" is missing for a given row, I'll impute with "S" (the most common boarding port).
- I'll ignore "Cabin" as a variable. There are too many missing values for imputation. Based on the information available, it appears that this value is associated with the passenger's class and fare paid.

```
1 train_data = train_df.copy()
2 train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
3 train_data["Embarked"].fillna(train_df["Embarked"].value_counts().idxmax(), inplace=True)
4 train_data.drop('Cabin', axis=1, inplace=True)
```

```
1 # check missing values in adjusted train data
2 train_data.isnull().sum()
3 train_data.head()
```

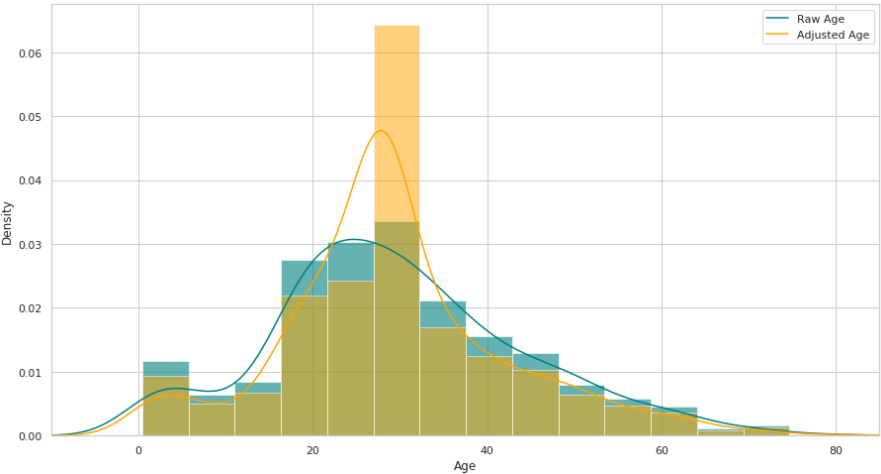


PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2833

```
1 # preview adjusted train data
2 train_data.head()
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2833

```
1 plt.figure(figsize=(15,8))
2 ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
3 train_df["Age"].plot(kind='density', color='teal')
4 ax = train_data["Age"].hist(bins=15, density=True, stacked=True, color='orange', alpha=0.5)
5 train_data["Age"].plot(kind='density', color='orange')
6 ax.legend(['Raw Age', 'Adjusted Age'])
7 ax.set(xlabel='Age')
8 plt.xlim(-10,85)
9 plt.show()
```



2.4.1. Additional Variables

According to the Kaggle data dictionary, both SibSp and Parch relate to traveling with family. For simplicity's sake (and to account for possible multicollinearity), I'll combine the effect of these variables into one categorical predictor: whether or not that individual was traveling alone.

```

1 ## Create categorical variable for traveling alone
2 train_data["TravelAlone"] = np.where((train_data["SibSp"] + train_data["Parch"]) > 0, 0, 1)
3 train_data.drop('SibSp', axis=1, inplace=True)
4 train_data.drop('Parch', axis=1, inplace=True)
5 train_data.head()

```

	PassengerId	Survived	Pclass	Name	Sex	Age	Ticket	Fare	Embarked	T
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	A/5 21171	7.2500	S	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs T. B.)	female	38.0	PC 17599	71.2833	C	

I'll also create categorical variables for Passenger Class ("Pclass"), Gender ("Sex"), and Port Embarked ("Embarked").

```

1 train_data.head()
2 #create categorical variables and drop some variables
3 # Convert categorical variable into dummy/indicator variables.
4 training = pd.get_dummies(train_data, columns=["Pclass", "Embarked", "Sex"])
5 training.drop('Sex_female', axis=1, inplace=True)
6 training.drop('PassengerId', axis=1, inplace=True)
7 training.drop('Name', axis=1, inplace=True)
8 training.drop('Ticket', axis=1, inplace=True)
9
10 final_train = training
11 final_train.head()

```

	Survived	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_S
0	0	22.0	7.2500	0	0	0	1	0	
1	1	38.0	71.2833	0	1	0	0	1	
2	1	26.0	7.9250	1	0	0	1	0	
3	1	35.0	53.1000	0	1	0	0	0	
4	0	35.0	8.0500	1	0	0	1	0	

Now, apply the same changes to the test data.

I will apply to same imputation for "Age" in the Test data as I did for my Training data (if missing, Age = 28).

I'll also remove the "Cabin" variable from the test data, as I've decided not to include it in my analysis.

There were no missing values in the "Embarked" port variable.

I'll add the dummy variables to finalize the test set.

Finally, I'll impute the 1 missing value for "Fare" with the median, 14.45.

```
1 test_df.isnull().sum()
```

```

PassengerId    0
Pclass          0
Name            0
Sex             0
Age            86
SibSp           0
Parch           0
Ticket          0
Fare            1
Cabin          327
Embarked        0
dtype: int64

```

```

1 test_data = test_df.copy()
2 test_data["Age"].fillna(test_data["Age"].median(skipna=True), inplace=True) #inplace=True -> overwrite the existing dataframe
3 test_data["Fare"].fillna(test_data["Fare"].median(skipna=True), inplace=True)
4 test_data.drop('Cabin', axis=1, inplace=True)
5
6 test_data["TravelAlone"] = np.where((test_data["SibSp"] + test_data["Parch"]) > 0, 0, 1)
7
8 test_data.drop('SibSp', axis=1, inplace=True)

```

```

9 test_data.drop('Parch', axis=1, inplace=True)
10
11 testing = pd.get_dummies(test_data, columns=["Pclass", "Embarked", "Sex"])
12 testing.drop('Sex_female', axis=1, inplace=True)
13 testing.drop('PassengerId', axis=1, inplace=True)
14 testing.drop('Name', axis=1, inplace=True)
15 testing.drop('Ticket', axis=1, inplace=True)
16
17 final_test = testing
18 final_test.head()

```

	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	E
0	34.5	7.8292	1	0	0	1	0	1	
1	47.0	7.0000	0	0	0	1	0	0	
2	62.0	9.6875	1	0	1	0	0	1	
3	27.0	8.6625	1	0	0	1	0	0	
4	22.0	12.2875	0	0	0	1	0	0	

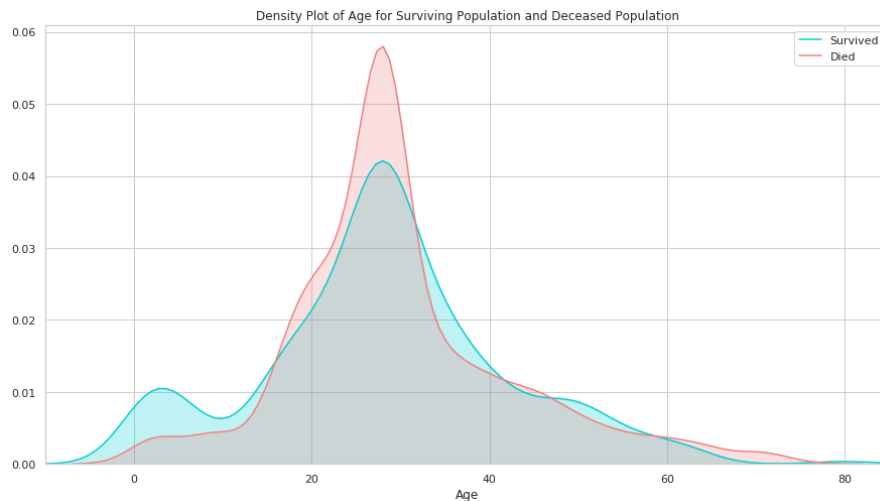
3. Exploratory Data Analysis

3.1. Exploration of Age

```

1 plt.figure(figsize=(15,8))
2 ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturquoise", shade=True)
3 sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral", shade=True)
4 plt.legend(['Survived', 'Died'])
5 plt.title('Density Plot of Age for Surviving Population and Deceased Population')
6 ax.set(xlabel='Age')
7 plt.xlim(-10,85)
8 plt.show()

```

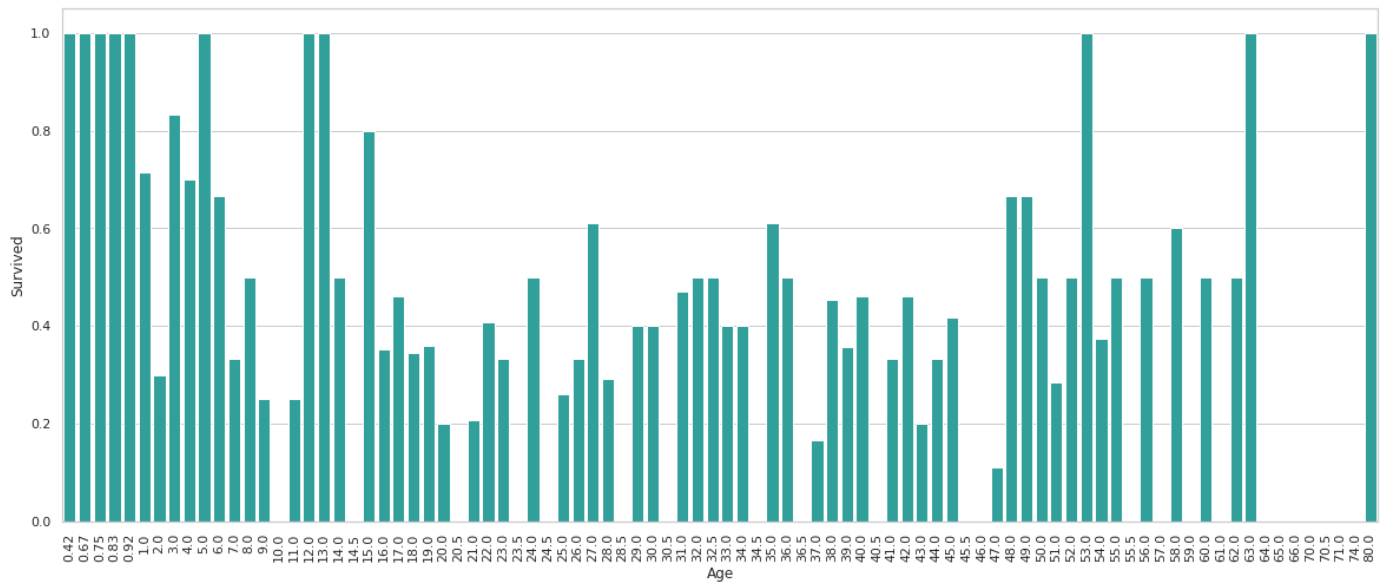


The age distribution for survivors and deceased is actually very similar. One notable difference is that, of the survivors, a larger proportion were children. The passengers evidently made an attempt to save children by giving them a place on the life rafts.

```

1 plt.figure(figsize=(20,8))
2 avg_survival_byage = final_train[["Age", "Survived"]].groupby(['Age'], as_index=False).mean()
3 g = sns.barplot(x='Age', y='Survived', data=avg_survival_byage, color="LightSeaGreen")
4 g.set_xticklabels(avg_survival_byage["Age"], rotation=90)
5 plt.show()

```



Considering the survival rate of passengers under 16, I'll also include another categorical variable in my dataset: "Minor"

```

1 final_train["IsMinor"] = np.where(final_train["Age"] <= 16, 1, 0)
2
3 final_test["IsMinor"] = np.where(final_test["Age"] <= 16, 1, 0)

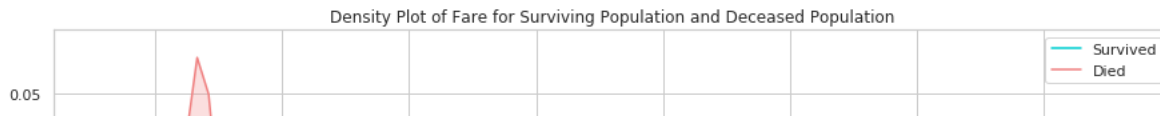
```

3.2. Exploration of Fare

```

1 plt.figure(figsize=(15,8))
2 ax = sns.kdeplot(final_train["Fare"], final_train.Survived == 1, color="darkturquoise", shade=True)
3 sns.kdeplot(final_train["Fare"], final_train.Survived == 0, color="lightcoral", shade=True)
4 plt.legend(['Survived', 'Died'])
5 plt.title('Density Plot of Fare for Surviving Population and Deceased Population')
6 ax.set(xlabel='Fare')
7 plt.xlim(-20,200)
8 plt.show()

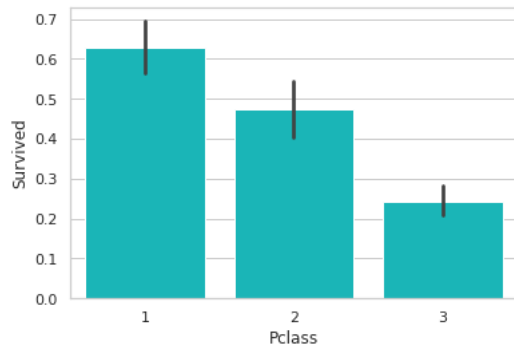
```



As the distributions are clearly different for the fares of survivors vs. deceased, it's likely that this would be a significant predictor in our final model. Passengers who paid lower fare appear to have been less likely to survive. This is probably strongly correlated with Passenger Class, which we'll look at next.

3.3. Exploration of Passenger Class

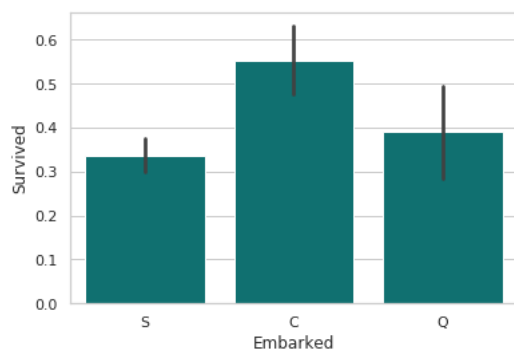
```
1 sns.barplot('Pclass', 'Survived', data=train_df, color="darkturquoise")
2 plt.show()
```



Unsurprisingly, being a first class passenger was safest.

3.4. Exploration of Embarked Port

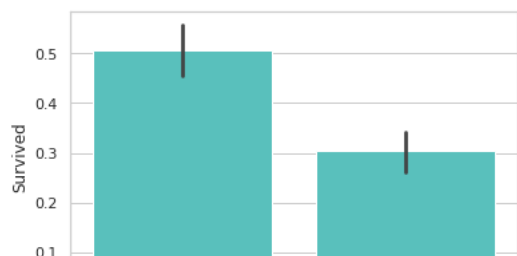
```
1 sns.barplot('Embarked', 'Survived', data=train_df, color="teal")
2 plt.show()
```



Passengers who boarded in Cherbourg, France, appear to have the highest survival rate. Passengers who boarded in Southampton were marginally less likely to survive than those who boarded in Queenstown. This is probably related to passenger class, or maybe even the order of room assignments (e.g. maybe earlier passengers were more likely to have rooms closer to deck).

3.5. Exploration of Traveling Alone vs. With Family

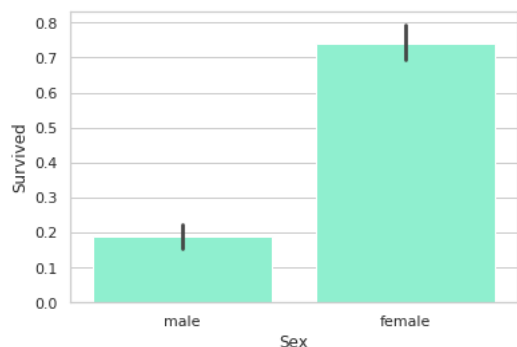
```
1 sns.barplot('TravelAlone', 'Survived', data=final_train, color="mediumturquoise")
2 plt.show()
```

Individuals traveling without family were more likely to die in the disaster than those with family aboard. Given the era, it's likely that individuals traveling alone were likely male.

3.6. Exploration of Gender Variable

```
1 sns.barplot('Sex', 'Survived', data=train_df, color="aquamarine")
2 plt.show()
```



This is a very obvious difference. Clearly being female greatly increased your chances of survival.

4. Logistic Regression and Results

```
1 from sklearn.linear_model import LogisticRegression
2
3 X = final_train.loc[:, final_train.columns != 'Survived']
4 print(X.shape)
5 print("X = ", X.head())
6
7 y = final_train['Survived']
8 print("y = ", y.head())
9
10 # Build a logreg and compute the feature importances
11 model = LogisticRegression()
12 model.fit(X,y)
13
14 print(final_test.head())
15 final_test['Survived'] = model.predict(final_test)
16 final_test['PassengerId'] = test_df['PassengerId']
17
18 submission = final_test[['PassengerId','Survived']]
19
20 submission.to_csv("submission.csv", index=False)
21
22 submission.tail()
```

```
(891, 11)
X =
  Age  Fare  TravelAlone  ...  Embarked_S  Sex_male  IsMinor
0  22.0  7.2500         0  ...           1         1         0
1  38.0  71.2833         0  ...           0         0         0
2  26.0  7.9250         1  ...           1         0         0
3  35.0  53.1000         0  ...           1         0         0
4  35.0  8.0500         1  ...           1         1         0
```

```
[5 rows x 11 columns]
```

```
y = 0 0
```

```
1 1
```

```
2 1
```

```
3 1
```

```
4 0
```

```
Name: Survived, dtype: int64
```

```

  Age  Fare  TravelAlone  ...  Embarked_S  Sex_male  IsMinor
0  34.5  7.8292         1  ...           0         1         0
1  47.0  7.0000         0  ...           1         0         0
2  62.0  9.6875         1  ...           0         1         0
3  27.0  8.6625         1  ...           1         1         0
4  22.0  12.2875         0  ...           1         0         0
```

```
[5 rows x 11 columns]
```

```

PassengerId  Survived
-----

```

----- Week6 ends here. -----

4.1. Feature selection

4.1.1. Recursive feature elimination

Given an external estimator that assigns weights to features, recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a `coef_` attribute or through a `feature_importances_` attribute. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

References:

http://scikit-learn.org/stable/modules/feature_selection.html

```

1 from sklearn.linear_model import LogisticRegression
2 from sklearn.feature_selection import RFE
3
4 #cols = ["Age", "Fare", "TravelAlone", "Pclass_1", "Pclass_2", "Embarked_C", "Embarked_S", "Sex_male", "IsMinor"]
5 #X = final_train[cols]
6 #y = final_train["Survived"]
7 # Build a logreg and compute the feature importances
8 model = LogisticRegression()
9 # create the RFE model and select 8 attributes
10 rfe = RFE(model, 8)
11 rfe = rfe.fit(X, y)
12 # summarize the selection of the attributes
13 print('Selected features: %s' % list(X.columns[rfe.support_]))

Selected features: ['TravelAlone', 'Pclass_1', 'Pclass_2', 'Pclass_3', 'Embarked_C', 'Embarked_Q', 'Sex_male', 'IsMinor']
```

4.1.2. Feature ranking with recursive feature elimination and cross-validation

RFECV performs RFE in a cross-validation loop to find the optimal number or the best number of features. Hereafter a recursive feature elimination applied on logistic regression with automatic tuning of the number of features selected with cross-validation.

```

1 from sklearn.feature_selection import RFECV
2 # Create the RFE object and compute a cross-validated score.
3 # The "accuracy" scoring is proportional to the number of correct classifications
4 rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=5, scoring='accuracy')
5 rfecv.fit(X, y)
6
7 print("Optimal number of features: %d" % rfecv.n_features_)
8 print('Selected features: %s' % list(X.columns[rfecv.support_]))
9
10 # Plot number of features VS. cross-validation scores
11 plt.figure(figsize=(10,6))
12 plt.xlabel("Number of features selected")
```

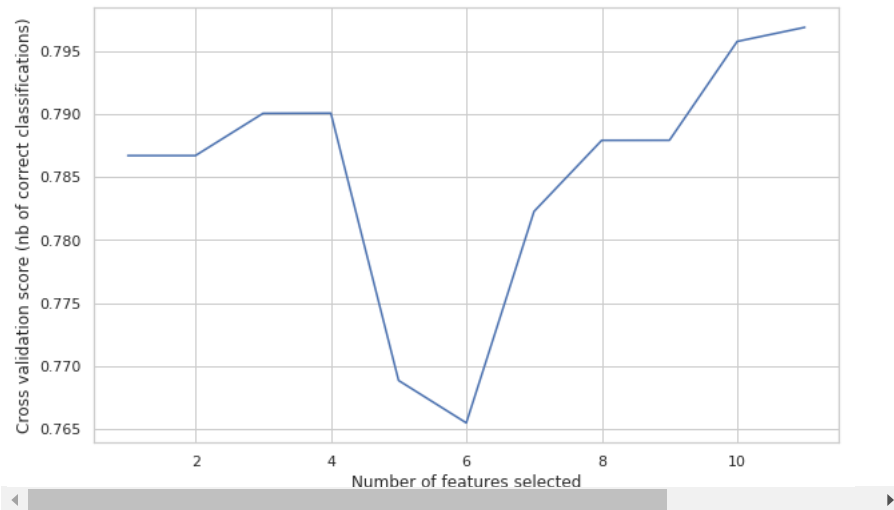
```

13 plt.ylabel("Cross validation score (nb of correct classifications)")
14 plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
15 plt.show()

```

Optimal number of features: 11

Selected features: ['Age', 'Fare', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Pclass_3', 'Embarked_C', 'Embarked_Q', ' ']

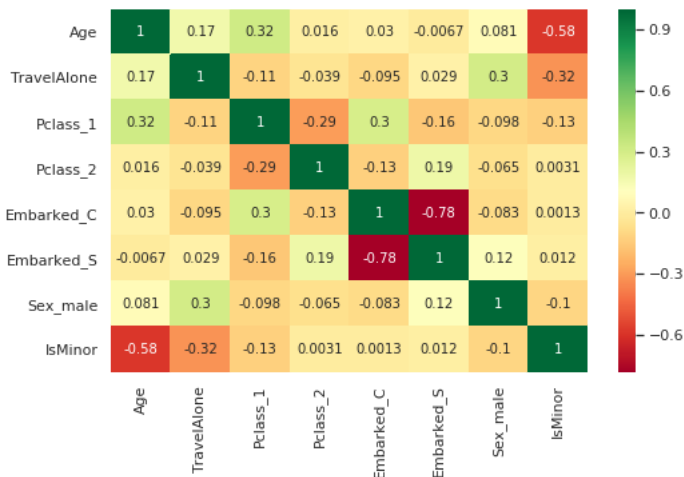


As we see, eight variables were kept.

```

1 Selected_features = ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C',
2                     'Embarked_S', 'Sex_male', 'IsMinor']
3 X = final_train[Selected_features]
4
5 plt.subplots(figsize=(8, 5))
6 sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
7 plt.show()

```



4.2. Review of model evaluation procedures

Motivation: Need a way to choose between machine learning models

- Goal is to estimate likely performance of a model on out-of-sample data

Initial idea: Train and test on the same data

- But, maximizing training accuracy rewards overly complex models which overfit the training data

Alternative idea: Train/test split

- Split the dataset into two pieces, so that the model can be trained and tested on different data
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance
- Problem with train/test split

- It provides a high variance estimate since changing which observations happen to be in the testing set can significantly change testing accuracy
- Testing accuracy can change a lot depending on a which observation happen to be in the testing set

Reference:

<http://www.ritchieng.com/machine-learning-cross-validation/>

4.2.1. Model evaluation based on simple train/test split using `train_test_split()` function

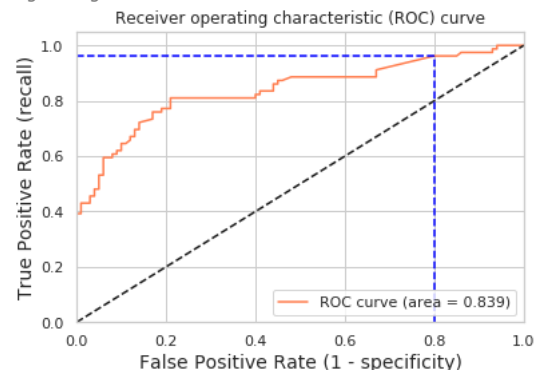
```
1 from sklearn.model_selection import train_test_split, cross_val_score
2 from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
3 from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss
4
5 # create X (features) and y (response)
6 X = final_train[Selected_features]
7 y = final_train['Survived']
8
9 # use train/test split with different random_state values
10 # we can change the random_state values that changes the accuracy scores
11 # the scores change a lot, this is why testing scores is a high-variance estimate
12 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
13
14 # check classification scores of logistic regression
15 logreg = LogisticRegression()
16 logreg.fit(X_train, y_train)
17 y_pred = logreg.predict(X_test)
18 y_pred_proba = logreg.predict_proba(X_test)[:, 1]
19 [fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
20 print('Train/Test split results:')
21 print(logreg.__class__.__name__+" accuracy is %2.3f" % accuracy_score(y_test, y_pred))
22 print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_proba))
23 print(logreg.__class__.__name__+" auc is %2.3f" % auc(fpr, tpr))
24
25 idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95
26
27 plt.figure()
28 plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
29 plt.plot([0, 1], [0, 1], 'k--')
30 plt.plot([0, fpr[idx]], [tpr[idx], tpr[idx]], 'k--', color='blue')
31 plt.plot([fpr[idx], fpr[idx]], [0, tpr[idx]], 'k--', color='blue')
32 plt.xlim([0.0, 1.0])
33 plt.ylim([0.0, 1.05])
34 plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
35 plt.ylabel('True Positive Rate (recall)', fontsize=14)
36 plt.title('Receiver operating characteristic (ROC) curve')
37 plt.legend(loc="lower right")
38 plt.show()
39
40 print("Using a threshold of %0.3f" % thr[idx] + "guarantees a sensitivity of %0.3f" % tpr[idx] +
41       "and a specificity of %0.3f" % (1-fpr[idx]) +
42       ", i.e. a false positive rate of %0.2f%%." % (np.array(fpr[idx])*100))
```

Train/Test split results:

LogisticRegression accuracy is 0.782

LogisticRegression log_loss is 0.504

LogisticRegression auc is 0.839



4.2.2. Model evaluation based on K-fold cross-validation using `cross_val_score()` function

```

1 # 10-fold cross-validation logistic regression
2 logreg = LogisticRegression()
3 # Use cross_val_score function
4 # We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
5 # cv=10 for 10 folds
6 # scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - although they are many
7 scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
8 scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
9 scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')
10 print('K-fold cross-validation results:')
11 print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
12 print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
13 print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())

K-fold cross-validation results:
LogisticRegression average accuracy is 0.802
LogisticRegression average log_loss is 0.454
LogisticRegression average auc is 0.850

```

4.2.3. Model evaluation based on K-fold cross-validation using `cross_validate()` function

```

1 from sklearn.model_selection import cross_validate
2
3 scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}
4
5 modelCV = LogisticRegression()
6
7 results = cross_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),
8                          return_train_score=False)
9
10 print('K-fold cross-validation results:')
11 for sc in range(len(scoring)):
12     print(modelCV.__class__.__name__+" average %s: %2.3f (+/-%2.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scoring.values())[sc]].mean()))
13     if list(scoring.values())[sc]=='neg_log_loss':
14         else results['test_%s' % list(scoring.values())[sc]].mean(),
15     results['test_%s' % list(scoring.values())[sc]].std())

K-fold cross-validation results:
LogisticRegression average accuracy: 0.802 (+/-0.025)
LogisticRegression average log_loss: 0.454 (+/-0.034)
LogisticRegression average auc: 0.850 (+/-0.025)

```

What happens when we add the feature "Fare"?

```

1 cols = ["Age", "Fare", "TravelAlone", "Pclass_1", "Pclass_2", "Embarked_C", "Embarked_S", "Sex_male", "IsMinor"]
2 X = final_train[cols]
3
4 scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}
5
6 modelCV = LogisticRegression()
7
8 results = cross_validate(modelCV, final_train[cols], y, cv=10, scoring=list(scoring.values()),
9                          return_train_score=False)
10
11 print('K-fold cross-validation results:')
12 for sc in range(len(scoring)):
13     print(modelCV.__class__.__name__+" average %s: %2.3f (+/-%2.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scoring.values())[sc]].mean()))
14     if list(scoring.values())[sc]=='neg_log_loss':
15         else results['test_%s' % list(scoring.values())[sc]].mean(),
16     results['test_%s' % list(scoring.values())[sc]].std())

K-fold cross-validation results:
LogisticRegression average accuracy: 0.796 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.034)
LogisticRegression average auc: 0.849 (+/-0.025)

```

We notice that the model is slightly deteriorated. The "Fare" variable does not carry any useful information. Its presence is just a noise for the logistic regression model.

4.3. GridSearchCV evaluating using multiple scorers simultaneously

```

1 from sklearn.model_selection import GridSearchCV
2
3 X = final_train[Selected_features]
4
5 param_grid = {'C': np.arange(1e-05, 3, 0.1)}
6 scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
7
8 gs = GridSearchCV(LogisticRegression(), return_train_score=True,
9                   param_grid=param_grid, scoring=scoring, cv=10, refit='Accuracy')
10
11 gs.fit(X, y)
12 results = gs.cv_results_
13
14 print('='*20)
15 print("best params: " + str(gs.best_estimator_))
16 print("best params: " + str(gs.best_params_))
17 print('best score:', gs.best_score_)
18 print('='*20)
19
20 plt.figure(figsize=(10, 10))
21 plt.title("GridSearchCV evaluating using multiple scorers simultaneously", fontsize=16)
22
23 plt.xlabel("Inverse of regularization strength: C")
24 plt.ylabel("Score")
25 plt.grid()
26
27 ax = plt.axes()
28 ax.set_xlim(0, param_grid['C'].max())
29 ax.set_ylim(0.35, 0.95)
30
31 # Get the regular numpy array from the MaskedArray
32 X_axis = np.array(results['param_C'].data, dtype=float)
33
34 for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
35     for sample, style in (('train', '--'), ('test', '-')):
36         sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%s' % (sample, scorer)]
37         sample_score_std = results['std_%s_%s' % (sample, scorer)]
38         ax.fill_between(X_axis, sample_score_mean - sample_score_std,
39                        sample_score_mean + sample_score_std,
40                        alpha=0.1 if sample == 'test' else 0, color=color)
41         ax.plot(X_axis, sample_score_mean, style, color=color,
42                alpha=1 if sample == 'test' else 0.7,
43                label="%s (%s)" % (scorer, sample))
44
45     best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
46     best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' % scorer][best_index]
47
48     # Plot a dotted vertical line at the best score for that scorer marked by x
49     ax.plot([X_axis[best_index], ] * 2, [0, best_score],
50            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)
51
52     # Annotate the best score for that scorer
53     ax.annotate("%0.2f" % best_score,
54               (X_axis[best_index], best_score + 0.005))
55
56 plt.legend(loc="best")
57 plt.grid('off')
58 plt.show()

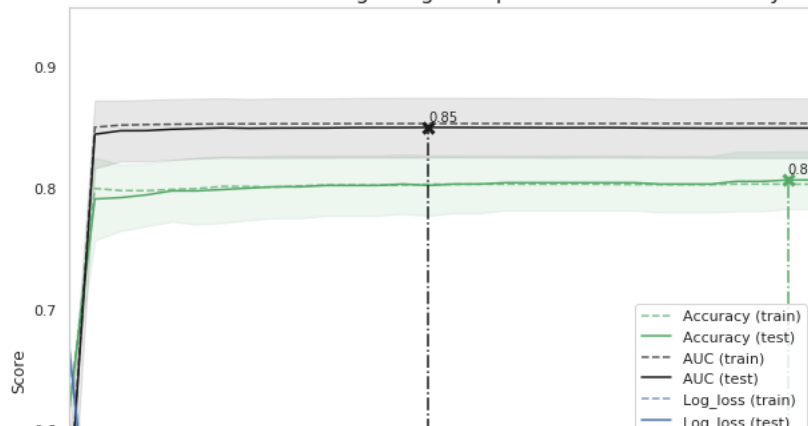
```

```

=====
best params: LogisticRegression(C=2.8000100000000003, class_weight=None, dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False)
best params: {'C': 2.8000100000000003}
best score: 0.8069584736251403
=====

```

GridSearchCV evaluating using multiple scorers simultaneously



4.4. GridSearchCV evaluating using multiple scorers, RepeatedStratifiedKFold and pipeline for preprocessing simultaneously

We can apply many tasks together for more in-depth evaluation like gridsearch using cross-validation based on k-folds repeated many times, that can be scaled or no with respect to many scorers and tuning on parameter for a given estimator!

```

1 from sklearn.preprocessing import StandardScaler
2 from sklearn.model_selection import RepeatedStratifiedKFold
3 from sklearn.pipeline import Pipeline
4
5 #Define simple model
6 #####
7 C = np.arange(1e-05, 5.5, 0.1)
8 scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
9 log_reg = LogisticRegression()
10
11 #Simple pre-processing estimators
12 #####
13 std_scale = StandardScaler(with_mean=False, with_std=False)
14 #std_scale = StandardScaler()
15
16 #Defining the CV method: Using the Repeated Stratified K Fold
17 #####
18
19 n_folds=5
20 n_repeats=5
21
22 rskfold = RepeatedStratifiedKFold(n_splits=n_folds, n_repeats=n_repeats, random_state=2)
23
24 #Creating simple pipeline and defining the gridsearch
25 #####
26
27 log_clf_pipe = Pipeline(steps=[('scale', std_scale), ('clf', log_reg)])
28
29 log_clf = GridSearchCV(estimator=log_clf_pipe, cv=rskfold,
30     scoring=scoring, return_train_score=True,
31     param_grid=dict(clf__C=C), refit='Accuracy')
32
33 log_clf.fit(X, y)
34 results = log_clf.cv_results_
35
36 print('='*20)
37 print("best params: " + str(log_clf.best_estimator_))
38 print("best params: " + str(log_clf.best_params_))
39 print("best score:", log_clf.best_score_)
40 print('='*20)
41
42 plt.figure(figsize=(10, 10))

```

```

43 plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)
44
45 plt.xlabel("Inverse of regularization strength: C")
46 plt.ylabel("Score")
47 plt.grid()
48
49 ax = plt.axes()
50 ax.set_xlim(0, C.max())
51 ax.set_ylim(0.35, 0.95)
52
53 # Get the regular numpy array from the MaskedArray
54 X_axis = np.array(results['param_clf__C'].data, dtype=float)
55
56 for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
57     for sample, style in (('train', '--'), ('test', '-')):
58         sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%s' % (sample, scorer)]
59         sample_score_std = results['std_%s_%s' % (sample, scorer)]
60         ax.fill_between(X_axis, sample_score_mean - sample_score_std,
61                        sample_score_mean + sample_score_std,
62                        alpha=0.1 if sample == 'test' else 0, color=color)
63         ax.plot(X_axis, sample_score_mean, style, color=color,
64                alpha=1 if sample == 'test' else 0.7,
65                label="%s (%s)" % (scorer, sample))
66
67     best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
68     best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' % scorer][best_index]
69
70     # Plot a dotted vertical line at the best score for that scorer marked by x
71     ax.plot([X_axis[best_index], ] * 2, [0, best_score],
72            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)
73
74     # Annotate the best score for that scorer
75     ax.annotate("%0.2f" % best_score,
76               (X_axis[best_index], best_score + 0.005))
77
78 plt.legend(loc="best")
79 plt.grid('off')
80 plt.show()

```



```

=====
best params: Pipeline(memory=None,
  steps=[('scale' StandardScaler(copy=True with mean=False with std=False)) ('clf' LogisticRegression)]
final_test['Survived'] = log_clf.predict(final_test[Selected_features]) final_test['PassengerId'] = test_df['PassengerId']
submission = final_test[['PassengerId','Survived']]
submission.to_csv("submission.csv", index=False)
submission.tail()

```

