MSDS 422 Assignment 3
Husein Adenwala

Introduction

The Purpose of this study is to create a machine learning model that can predict passengers that would survive and passengers that would not survive on the Titanic. On April 15, 1912 the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew (70%).

Data preparation, exploration, visualization

The Titanic dataset is split into two datasets: train and test. The train dataset has 891 observation and 12 columns, and the test dataset has 418 observation and does not include the response variable (it therefore has 11 columns). The two datasets did have missing values for Age, cabin and Fare as seen in the table below:

Number of missing values for each column

```
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
```

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

Imputation

I will impute the Age by calculating the median age by grouping the data by Pclass and SibSp (number of siblings/spouse).

For the missing cabins, I will replace the missing cabin values by a new U class as there is not pattern for the missing cabin values. It is best to assign as new class U – unknown.

As there is only one missing Fare value, I will replace the value with the mean fare.

For the two missing Embarked values, I will replace the missing values with S as that is the most likely missing value. Both of the passengers in question are women who survived; as seen below, the majority of the women that survived embarked at port S (in the tables, 0 is female, 1 is male).

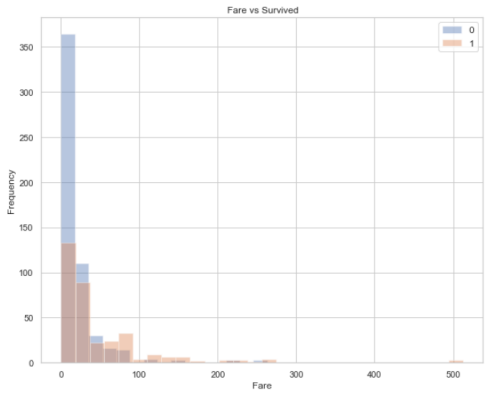
Sex	Embarked	Count
0	C	73
0	Q	36
0	S	203
1	C	95
1	Q	41
1	S	441

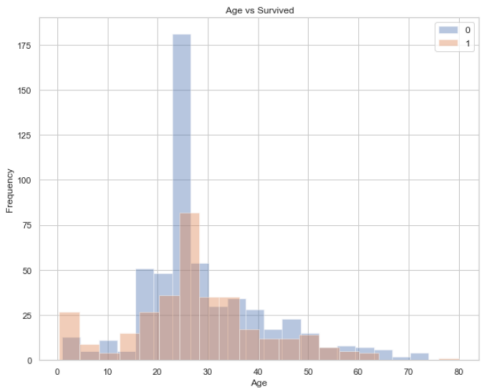
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
62	1	1	Icard, Miss. Amelie	0	38.0	0	0	113572	80.0	B	NaN
830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	0	62.0	0	0	113572	80.0	B	NaN

Features Selection

For features selection, I created histograms and cross tables to identify features which have value or categories which discriminate between survived and not survived.

The plot below for Fare and Age shows that passengers that paid a Fare between 0-50 were most likely to not survive and passengers between the age of 16-30 were most likely to not survive (in the graphs below, 0 is not survived and 1 is survived).

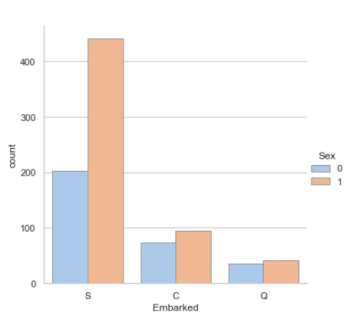


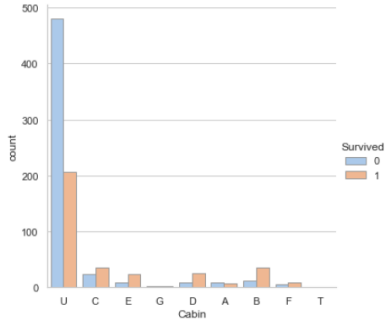


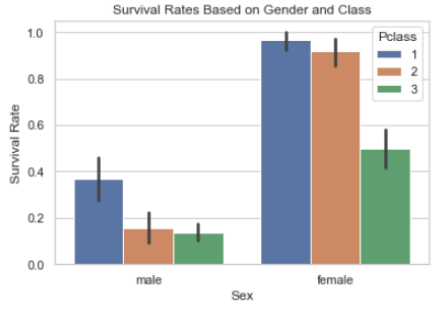
Women that embarked at S were more likely to survive and passengers with unknown cabins were most likely to not survive.

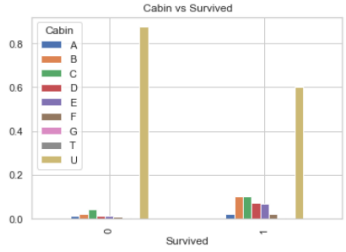
Embarked	C		Q		S	
Sex	0	1	0	1	0	1
Survived						
0	0.02	0.12	0.02	0.07	0.11	0.66
1	0.19	0.09	0.08	0.01	0.41	0.23

Embarked	C		Q		S	
Sex	female	male	female	male	female	male
Survived						
0	9	66	9	38	63	364
1	64	29	27	3	140	77

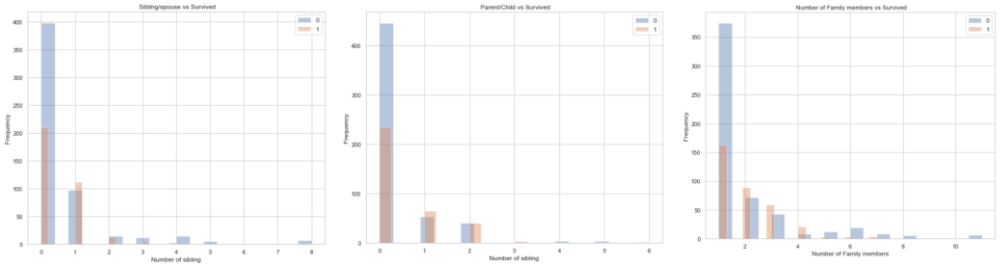








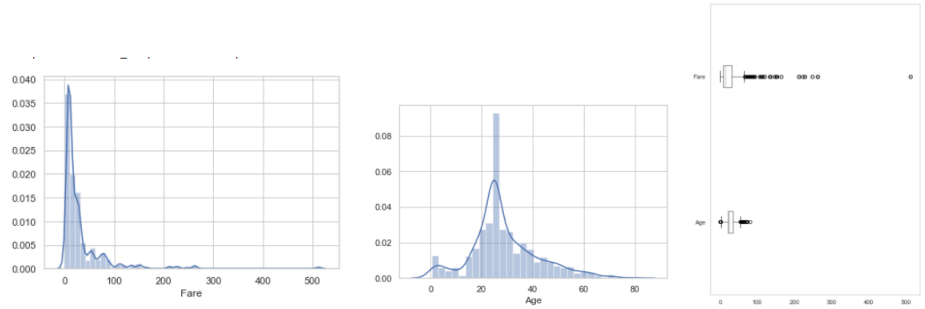
The people without parent/child or sibling/spouse were less likely to survive. I combined the two to create a new feature, family size, that I will use in the model in place of the two features (parent/child and sibling/spouse), that have mild collinearity.



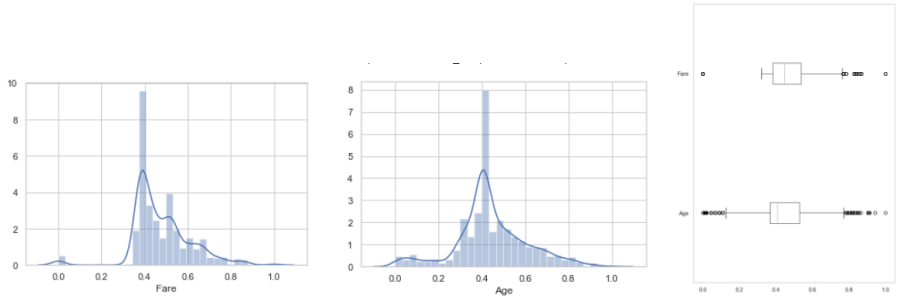
Transformation for SVM and KNN only

The continuous variable for Age and Fare is transformed so they are in the same scale and are more normally distributed. I have also transformed Pclass and Family_size so all variables are in the same scale.

Before Transformation



After Transformation



I have selected the following six features—Pclass, Sex, Age, Fare, Cabin, Embarked and Family size—for my predictive model.

For the categorical data – Embarked and Cabin—I will create N-1 dummy variables and the final features that will be used are as follows:

Pclass	Sex	Age	Fare	Cabin A	Cabin B	Cabin C	Cabin D	Cabin E	Cabin F	Cabin G	Cabin T	Embarked S	Embarked C	FamilySize

Review research design and modeling methods

For this analysis, I used a classification model because of the Binary nature of our response variable Survived. Specifically, I used Logistic Regression (LR), Support Vector Machines (SVM) and K-nearest neighbors (KNN). Each of these models have various characteristics that make them suitable for specific datasets.

SVM tries to find the best margin that separates the classes and this reduces the risk of error on the data, while logistic regression does not; instead, it can have different decision boundaries with different weights that are near the optimal point.
KNN is a non-parametric model, where LR is a parametric model. KNN is comparatively slower than LR. KNN supports non-linear solutions where LR supports only linear solutions.

LR is easier to implement, interpret, and very efficient to train for binary data. SVM is effective in high dimension space and KNN is very simple to implement and robust (e.g. classes don't have to be linearly separable).

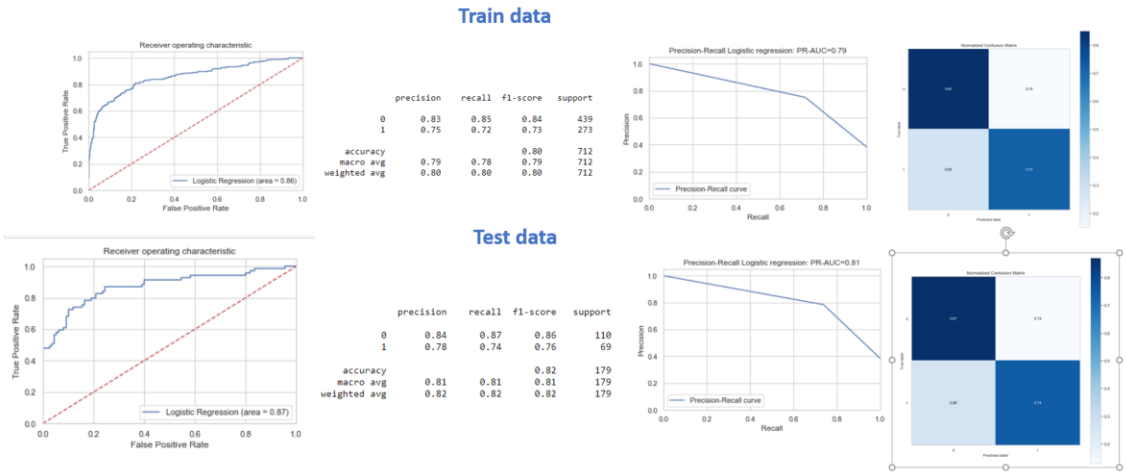
All three models seem to be appropriate for the titanic dataset.
To evaluate these models, I used only the selected 6 features (16 with dummy coded features) and did a 80/320 split on the dataset for Training and Testing respectively. I used the same test and training data on all the models to do a fair comparison.

I also used the k-fold cross validation (k=10) technique to calculate and compare the accuracy for each model. Additionally, I ran a comparison of accuracy score, AUC score, Precision, Recall and F1 score between train and test data as an evaluation metric.

Review results, evaluate models

Logistic Regression:

Logistic Regression performed well on Precision, Recall, Accuracy and f1-score and the performance between test and train data was very similar. With the imbalanced data, where there is a high number of 0 as compared to 1 for the response variable survived; the recall score is important. The Precision and Recall scores are similar and don't have a huge variation between test and train data. The False positive rate is lower compared to other models.



Below is the equation for the logistic regression:

```
Y_hat = 4.33 - 0.771*Pclass - 2.578*Sex -0.038*Age  + 0.003*Fare  + 0.595*
Cabin A + 0.255*Cabin B -0.074*Cabin C + 0.817*Cabin D + 1.571*Cabin E +0.
908*'Cabin F - 0.015*Cabin G - 0.12*Cabin T - 0.306* Embarked S + 0.018* E
mbarked C - 0.204* FamilySize
```

The intercept $= 4.33$ can be interpreted as follows when all other coefficients = 0 :

- exp(4.33) = 75.94, i.e. we estimate that the log odds of survival increases by 75.94 when sex is Female (0), cabin is U and Embark is Q

Pclass$= -0.771$ is discrete categorical variable :

- exp(-0.771) = 0.46, i.e we estimate that the log odds of survival decreases by 0.46 when pclass increase by one unit

Sex$= -2.578$ is a binary variable.

- exp$(-2.578 - 1)$= 0.08 , i.e we estimate that the log odds of survival decreases by 0.08 When sex is Male (1)

Age = -0.038 is interpreted as follows when all other coefficients are = 0 :

- exp(-0.038) = 0.96 , i.e we estimate that the log odds of survival decreases by 0.96 with increase in Age

Fare = 0.003 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.003) = 1.003 , i.e we estimate that the log odds of survival increases by 1.003 with increase in Fare

Cabin A= 0.595 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.595) = 1.81 , i.e we estimate that the log odds of survival increases by 1.81 when cabin is A

Cabin B= 0.255 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.255) = 1.29 , i.e we estimate that the log odds of survival increases by 1.29 when cabin is B

Cabin C= -0.074 can be interpreted as follows when all other coefficients are = 0 :

- exp(-0.074) = 0.93 , i.e we estimate that the log odds of survival decreases by 0.93 when cabin is C

Cabin D= 0.817 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.871) = 2.38 , i.e we estimate that the log odds of survival increases by 2.38 when cabin is D

Cabin E= 1.571can be interpreted as follows when all other coefficients are = 0 :

- exp(1.571) = 4.81 , i.e we estimate that the log odds of survival increases by 4.81 when cabin is E

Cabin F = 0.908 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.908) = 2.48 , i.e we estimate that the log odds of survival increases by 2.48 when cabin is F

Cabin G= -0.015 can be interpreted as follows when all other coefficients are = 0 :

- exp(-0.015) = 0.98 , i.e we estimate that the log odds of survival decreases by 0.98 when cabin is G

Cabin T= -0.012 can be interpreted as follows when all other coefficients are = 0 :

- exp(-0.012) = 0.99 , i.e we estimate that the log odds of survival decreases by 0.99 when cabin is T

Embared S= 0.306 can be interpreted as follows when all other coefficients are = 0 :

- exp(0.306) = 1.35 , i.e we estimate that the log odds of survival increases by 1.35 when Embarked is S

Embared E= 0.018 can be interpreted as follows when all other coefficients are = 0 :

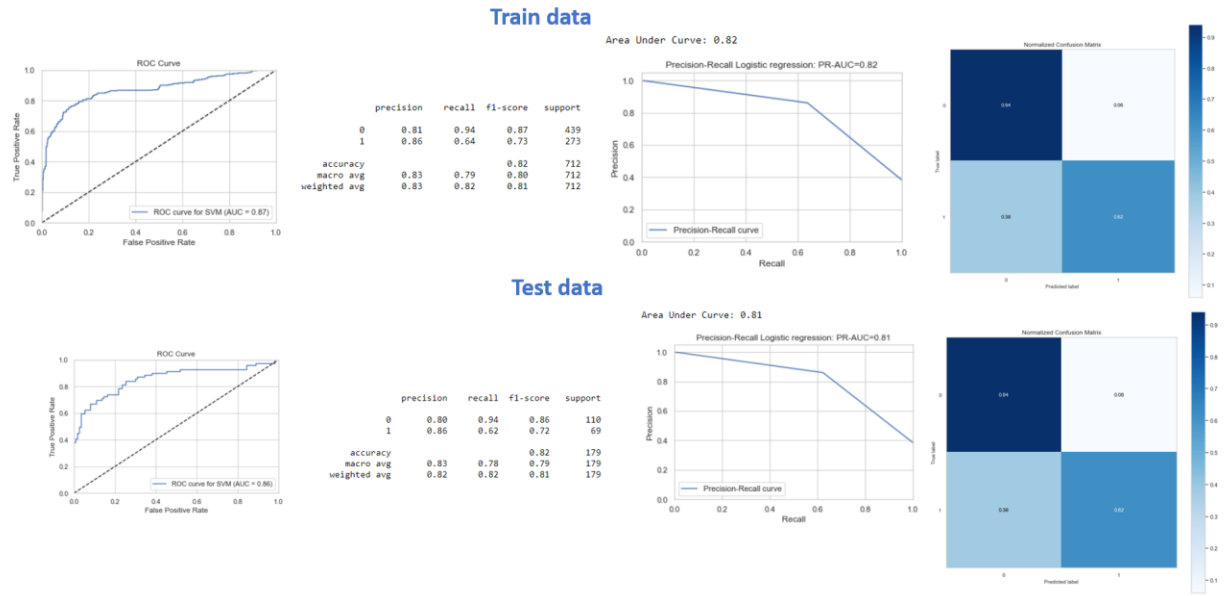
- exp(0.018) = 1.02 , i.e we estimate that the log odds of survival increases by 1.02 when Embarked is E

Family Size= -0.204 can be interpreted as follows when all other coefficients are = 0 :

- exp(-0.204) = 0.815 , i.e we estimate that the log odds of survival decreases by 0.815 when family size increases by 1 unit

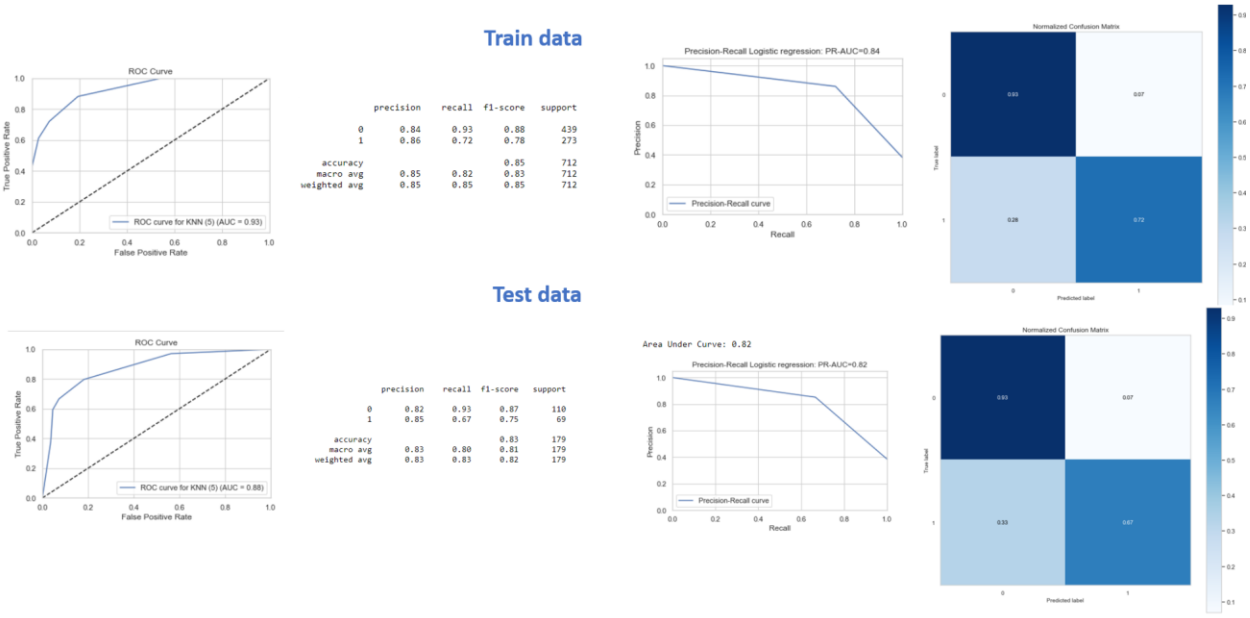
SVM (Transformed Features)

SVM also performed well on Precision, Recall (slightly lower), Accuracy and F1-score and the performance between test and train data was very similar. Like with the logistic regression, with the imbalanced data, where there is a high number of 0 as compared to 1 for the response variable survived; the recall score is important. SV has lower recall score compared to precession and has relatively higher False Positives on both test and train data than the Logistic Regression.



KNN with transformed features

KNN performed best on Precision, Recall, Accuracy and F1-score and the performance between test and train data was very similar. Like the other models, with the imbalanced data, where there is a high number of 0 as compared to 1 for the response variable survives, the precision recall curve and recall score is important. KNN has a lower recall score compared to precision and has slightly more false negatives on the test data as compared to logistic regression.



Summary of three models with training and test data

This summary shows that SVM performed highest in accuracy and AUC, followed by KNN and finally Logistic regression but all three models are close in performance. The difference between the test and train scores as shown above and the scores shown below is because the summary below takes the average of a few Accuracy methods.

Training data

	Classifier	Accuracy	AUC
0	Logistic Regression	0.796405	0.857281
1	Support Vector Machine	0.804826	0.927391
2	K-Nearest Neighbours	0.793588	0.869830

Test data

Classifier	Accuracy	AUC
Logistic Regression	0.770476	0.871542
Support Vector Machine	0.793333	0.875889
K-Nearest Neighbours	0.798889	0.856258

K-fold cross validation score

Like the test and train performance metrics, KNN performs better than the two models in K-fold comparison but the logistic regression performs slightly better than all others in the Test data at predicting True positives (survived) compared to other models. 61% of people did not survive, so it is easier to predict not survived (0) compared to survived (1). In the full dataset, there should be around 70% who did not survive.

```
Method    accuracy score
Logistic    0.792409
SVM         0.801348
KNN         0.803670
dtype: float64
   Logistic       SVM       KNN
0  0.755556  0.800000  0.733333
1  0.808989  0.764045  0.775281
2  0.764045  0.752809  0.797753
3  0.842697  0.820225  0.786517
4  0.752809  0.842697  0.865169
5  0.808989  0.820225  0.820225
6  0.764045  0.786517  0.820225
7  0.808989  0.752809  0.797753
8  0.842697  0.865169  0.831461
9  0.775281  0.808989  0.808989
```

Implementation and programming

Using Python Pandas package, I loaded the Titanic Train and test csv file to the Data Frame object. Exploratory data analysis and feature selection was done utilizing the function in the Pandas Numpy, seaborn and matplotlib packages. Features were evaluated using SelectKBest and Feature_importance_ methods from ExtraTreesClassifier package in scikit-learn.

Imputation was done by using pandas and Numpy packages.

I then created a Numpy array (y) for response variable Survived and removed it from the Titanic Data Frame.

Then I created a copy for features that will be used for logistic regression.

I then used the box-cox function from the scipy package to do the variable transformation on the non-categorical variable Data Frame and then used the Minmaxscalar() from the scikit-learn package to scale all the values in the Data Frame.

At this point, I created an 80/20 train and test split of the two sets of Numpy arrays (X: response variable, with and without transformed features) and Numpy array (Y: features), then used the Logistic Regression function in scikit learn package to create a logistic regression model (using the non-transformed data) and then evaluated accuracy, ROC, AUC, F1 Score, Precision for train and test data using scikit learn method and manually created functions.
I used the Support Vector Machines from scikit learn to create the SVM model and then evaluated accuracy, ROC, AUC , F1 Score, Precision for train and test data using scikit learn method and manually created functions.
I then used the K-Nearest Neighbors function from scikit learn to create the KNN model and then evaluated accuracy, ROC, AUC, F1 Score, Precision for train and test data using scikit learn method and manually created functions.
I did hyperparameter tuning for the best model (KNN), and for all models and slightly improved the model in training but made it slightly worse on test data (I did not end up using it in any of the models).

Finally, I did K-Fold cross-validation (k=10) using the K-Fold package in scikit-learn. To do this, I created a numpy array of all the data for all the variables, including the transformed and scaled features and the original response variable, and calculated the mean Accuracy of the 10 folds for comparison.

It is important to note that logistic regression was not sensitive to the transformation and performed very similarly for transformed and not transformed variables. For K-fold, I used data with transformed features for all models but performing train and test logistic regression and to derive the regression equation, I used the non-transformed data.

Also, KNN and SVM were sensitive to transformation.

Exposition, problem description, and management recommendations

After evaluating all three models, I would recommend the KNN model. All models are very similar in performance, but based on the following analysis, I feel that K-Nearest Neighbor is a best fit for this dataset.
Accuracy, AUC, Recall, Precision and F1 score are similar for all the models. The difference between accuracy and AUC for Train and test data does not show any significant overfitting and, in fact, for KNN, it performs better in test. However, K-fold validation (k=10) is a better technique to evaluate the model fit with a small dataset. Based on the accuracy score calculated by cross validation, I found that accuracy score was highest for KNN, followed by SVM and then logistic regression. All of them are within one percent difference.
The logistic regression does well when there is clear linear separation and SVM does well when we have high dimensional data. KNN is a non-parametric algorithm, which means that it does not make any assumptions on the underlying data distribution. KNN is very simple to implement and robust, e.g. classes don't have to be linearly separable but does not perform well on high dimensional data.

The titanic data set does not have high dimension and is not a large data set. There is some linear separation but there overlap too and this make KNN most suitable model for the titanic data set.

Appendix

Feature Section function

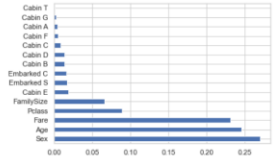
```
X = training_df.iloc[:,1:]  #independent columns
y = training_df.iloc[:,0:1]    #target column i.e price range
from sklearn.ensemble import ExtraTreesClassifier

#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=6)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
        Specs       Score
1         Sex  92.702447
0      Pclass  29.572786
5    Cabin B  25.875581
13  Embarked C  20.464401
7    Cabin D  19.489646
8    Cabin E  18.140638
14  FamilySize  13.506718
6    Cabin C  10.936730
12  Embarked S   5.489205
3        Fare   3.438646
```

```
X = training_df.iloc[:,1:]  #independent columns
y = training_df.iloc[:,0:1]    #target column i.e price range
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(16).plot(kind='barh')
plt.show()
```

```
[0.089061   0.26993286 0.24550091 0.23090742 0.00403771 0.01381264
 0.00866556 0.01313802 0.01853907 0.00521516 0.00224535 0.00050555
 0.01694718 0.01574625 0.06574532]
```



Kfold cross validation

```
# Seed value for random number generators to obtain reproducible results
RANDOM_SEED = 1

# The model input data outside of the modeling method calls
names = ['Logistic', 'SVM', 'KNN']

# Specify the set of regression models being evaluated (we set normalize=False because we have standardized above)
regressors = [LogisticRegression(max_iter=75),
              SVC(kernel="rbf"),
              KNeighborsClassifier(n_neighbors=3, leaf_size=1)
              ]
```

```
model_data= training_df.values
```

```
# Establish number of cross folds employed for cross-validation
N_FOLDS = 10

# Setup numpy array for storing results
cv_results = np.zeros((N_FOLDS, len(names)))
cv_r2 =np.zeros((N_FOLDS, len(names)))
# Initiate splitting process
kf = KFold(n_splits = N_FOLDS, shuffle=False, random_state = RANDOM_SEED)

# Check the splitting process by looking at fold observation counts
index_for_fold = 0  # Fold count initialized
for train_index, test_index in kf.split(model_data):
    print('\nFold index:', index_for_fold, '----------------------------------------------------------------------------------------')

# The structure of modeling data for this study has the response variable coming first and explanatory variables later
# so 1:model_data.shape[1] slices for explanatory variables and 0 is the index for the response variable
    x_train = model_data[train_index, 1:model_data.shape[1]]
    x_test = model_data[test_index, 1:model_data.shape[1]]
    y_train = model_data[train_index, 0]
    y_test = model_data[test_index, 0]

    index_for_method = 0  # Method count initialized
    #index_r = 0
    for name, reg_model in zip(names, regressors):
        reg_model.fit(x_train, y_train)  # Fit on the train set for this fold

        # Evaluate on the test set for this fold
        y_test_predict = reg_model.predict(x_test)
        fold_method_result = accuracy_score(y_test, y_test_predict)
        #R_squared = reg_model.score(X_test,y_test)
        cv_results[index_for_fold, index_for_method] = fold_method_result

        #cv_r2[index_r, index_for_method]= R_squared
        index_for_method += 1
        #index_r += 1

    index_for_fold += 1

cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names
cv_r2df= pd.DataFrame(cv_r2)
cv_r2df.columns = names
print('\n----------------------------------------------------------------------------------------')
print('Average results from ', N_FOLDS, '-fold cross-validation\n',      'in standardized units (mean 0, standard deviation
print(cv_results_df.mean())
print(cv_results_df)
```

```
preds = np.where(log_reg.predict_proba(x_train)[:,1] > 0.6, 1, 0)
```

```
print(classification_report(y_train, preds))
```

```
              precision    recall  f1-score   support

         0.0       0.80      0.93      0.86       493
         1.0       0.86      0.63      0.73       309

    accuracy                           0.82       802
   macro avg       0.83      0.78      0.80       802
weighted avg       0.82      0.82      0.81       802
```

Hyperparameter tuning with Grid search

```
#List Hyperparameters that we want to tune.
leaf_size = list(range(1,50))
n_neighbors = list(range(1,30))
p=[1,2,3]
#Convert to dictionary
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
#Create new KNN object
knn_2 = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn_2, hyperparameters, cv=10)
#Fit the model
best_model = clf.fit(x_train,y_train)
#Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

```
Best leaf_size: 1
Best p: 2
Best n_neighbors: 3
```