Capstone Project Report

Nadima Dwihusna

January 19, 2021

1 Definition

1.1 Project Overview

The Titanic was widely considered as the “unsinkable” ship. However, during its voyage on April 15, 1912, the Titanic sank after hitting an iceberg. Because there were not enough lifeboats, 1502 out of 2224 passengers died. Using the passenger data (e.g. name, age, gender, socio-economic class, etc.) that boarded the Titanic during the voyage, there seems to be a pattern where some groups of people were more likely to survive than others. The main objective of the project is to use machine learning to create a model that predicts which passengers survived the Titanic shipwreck. This project is taken from the Kaggle Competition: “Titanic – Machine Learning from Disaster” (https://www.kaggle.com/c/titanic/overview/description). I am doing this project as this will be my first Kaggle competition and first time building a predictive machine learning model.

1.2 Problem Statement

As stated in the Kaggle Competition (https://www.kaggle.com/c/titanic/overview), the main goal of this project is to build a machine learning model that answers the following question: “What sorts of people were more likely to survive?”. The data of the passengers who boarded the Titanic in 1912 is provided and to be used to build the model that predicts if the passenger survives.

1.3 Evaluation Metrics

An accuracy score and confusion matrix of the classification results will be used as an evaluation metric to quantify the performance of the model.

- The accuracy score compares the total of true positives and true negatives divided by the total number of samples. In other words, this accuracy score indicates how much the machine learning model predicts the outcome of the passenger correctly to the total number of passengers.
- The confusion matrix visualizes the accuracy of the machine learning model by comparing the machine learning model predictions to the actual outcomes. The goal of the project is

to have a high number of true positives and true negatives where the actual outcomes matches with the machine learning predictions.

	1 (Predicted)	0 (Predicted)
1 (Actual)	True Positive	False Negative
0 (Actual)	False Positive	True Negative

Figure 1: Confusion matrix to visualize accuracy of machine learning models.

2 Analysis

2.1 Data Exploration

The training and testing data set can be found in the Kaggle Competition website (https://www.kaggle.com/c/titanic/data). The data is split into two groups:

- **training set (train.csv)**
- **test set (test.csv)**

The training set is used to build the machine learning models. This training set contains the outcomes (ground truth) for each passenger. The test set is used to see how well the model performs with new data. The test set does not have ground truth for each passenger as it is my job to predict the outcomes (passenger survive or not). Additionally, a **gender_submission.csv** file is provided. This file shows an example of what a submission file should look like. All the features in the dataset describes the passenger information as follow:

Feature Variable	Definition	Key
PassengerId (integer)	Passenger ID / count number	
Survived (integer)	Survival	0 = No, 1 = Yes
Pclass (integer)	Ticket class (socio-economic status or SES)	1 = 1st, 2 = 2nd, 3 = 3rd
Name (string)	Passenger name	
Sex (string)	Sex	

Age (integer)	Age in years (Age is fractional if <1. If the age is estimated, it is in the form of xx.5)	
SibSp (integer)	# of siblings / spouses aboard the Titanic	
Parch (integer)	# of parents / children aboard the Titanic	
Ticket (integer)	Ticket number	
Fare (float)	Passenger fare	
Cabin (string)	Cabin number	
Embarked (string)	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

After viewing the summary of the training and testing data, I decided that I do not need Passenger ID, Name, and Ticket. I assumed Name and Ticket are related to the Cabin class. Figure 2 represents the final features we are working with.

	Pclass	Age	SibSp	Parch	Fare	Survived
count	891.000000	714.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	29.699118	0.523008	0.381594	32.204208	0.383838
std	0.836071	14.526497	1.102743	0.806057	49.693429	0.486592
min	1.000000	0.420000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	20.125000	0.000000	0.000000	7.910400	0.000000
50%	3.000000	28.000000	0.000000	0.000000	14.454200	0.000000
75%	3.000000	38.000000	1.000000	0.000000	31.000000	1.000000
max	3.000000	80.000000	8.000000	6.000000	512.329200	1.000000

Figure 2: train.describe() is ran to show the summary of the training data.

In the preprocessing step, the null or missing values are detected in the training and testing data set. Majority of the missing data are in the Age and Cabin features. For the Age features, the

missing data are inputted with the median Age, and for the Cabin features, the missing data are inputted as Cabin “N”. Additionally, as part of preprocessing, all the features are converted to integers to be visualized in the correlation matrix and pairwise relationship plots.

2.2 Exploratory Visualization

To summarize the relevant features and characteristic of the dataset, various visualizations have been made. From Figure 3, based on the training data, we can see that only 38.59 % of the passengers survived the sinking of the Titanic. Figure 4 shows a correlation matrix and the pairwise relationship in the training data showing the similarity between the different features in the data set.

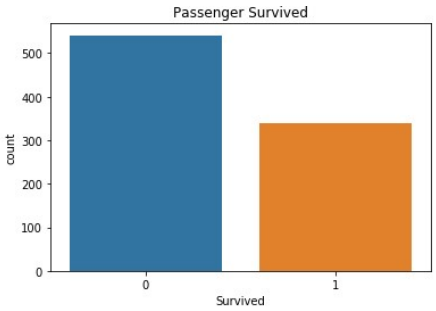


Figure 3: Outcome of passenger in the training data with 1 as survived and 0 as died.

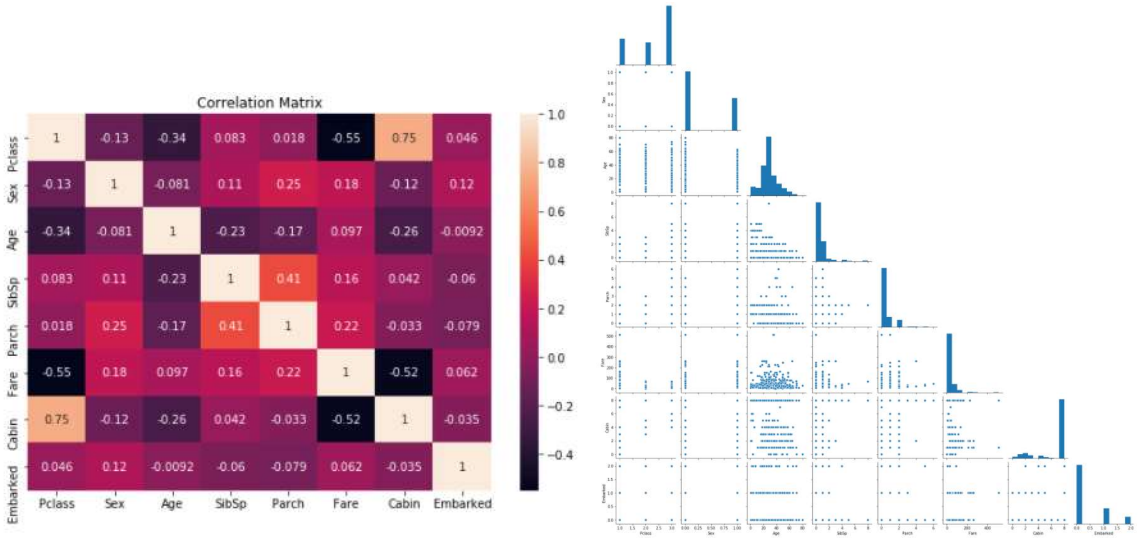


Figure 4: Correlation Matrix and Plot Pairwise Relationship of Features in training data. The Survived values serve as the label for the training in supervised machine learning.

As seen in Figures 5 and Figure 6, further visualizations have been made for each features and its probability of survival. Seen in these figures, females (74.75% survival probability) are more

likely to survive than males. The passengers with Pclass 1 and embarked from C (Cherbourg) are also more likely to survive. Also, seen from the age bell curves in Figure 6, young children and babies are more likely to survive.

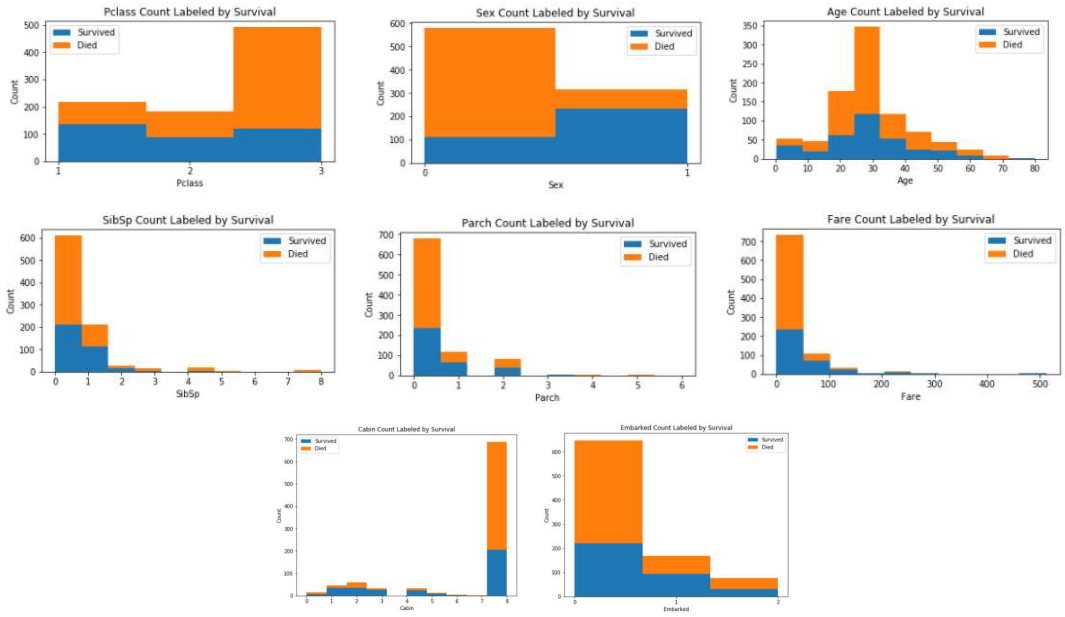


Figure 5: Counts of different features labeled by survival in the training data.

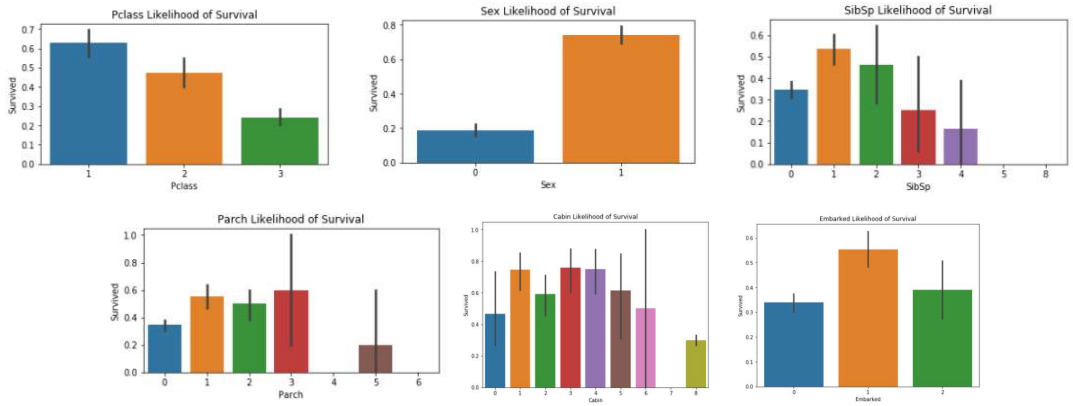


Figure 6: Features versus the probability of survival in the training data.

2.3 Algorithms and Techniques

Pandas and NumPy libraries were used for data processing and the machine learning model are built using scikit-learn estimators. The quality is measured using the performance metrics or loss function. The Scikit-Learn python open source machine learning library is used for all the

processing, modelling, and prediction steps. These are the four machine learning algorithms trained and optimized to predict the passengers' outcomes in the Titanic:

1. Support Vector Classifier (SVC)
2. Decision Tree
3. Random Forest
4. K-Nearest Neighbors (KNN).

I chose SVC because it is known to be a good algorithm for classification or regression problems. SVC transforms the data and finds the optimal boundary between the outputs. I chose decision tree and random forest because these works well in classification problem. It would also be interesting to compare the outcomes since the random forest combines the output of individual decision trees to generate the final output. Lastly, KNN is used since it is an intuitive and simple classification algorithm I have used before in other projects.

2.4 Benchmark

For benchmarking, I decided to compare the four machine learning classifiers and evaluate the mean accuracy of each by a cross validation process. For benchmark, the default “standard” hyperparameters that relates to the domain, problem statement, and solution. Afterwards, the models are optimized with different hyperparameters. These standard benchmark models are used to objectively compare and evaluate the results with the optimized models. This will help identify improve the final model performance.

3 Methodology

3.1 Data Preprocessing

After visualizing and getting a better understanding of the data and its features, the training and testing data is further processed. As stated in previous section, I filled in any missing data specifically in Age and Cabin features. Seen in Figure 7, there are 256 missing values for Age for the whole training and testing dataset. Since age seems to be an important factor to determining the passenger outcomes, I decided to fill in the missing Age values with median age. For the missing Cabin features, I decided to add an extra cabin ‘N’ for passengers with unassigned cabin.

```
train = train.fillna(np.nan)
train.info()
train.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 881 entries, 0 to 880
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  881 non-null    int64
 1   Survived     881 non-null    int64
 2   Pclass       881 non-null    int64
 3   Name         881 non-null    object
 4   Sex          881 non-null    object
 5   Age          711 non-null    float64
 6   SibSp        881 non-null    int64
 7   Parch        881 non-null    int64
 8   Ticket       881 non-null    object
 9   Fare         881 non-null    float64
 10  Cabin        201 non-null    object
 11  Embarked     881 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 82.7+ KB
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            170
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          680
Embarked         0
dtype: int64
```

```
test = test.fillna(np.nan)
test.info()
test.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         418 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

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

Figure 7: Null and missing values in the training (left) and testing data set (right). In the training data, there are 170 missing values for Age and 680 missing values for Cabin. In the testing data, there are 86 missing values for Age and 327 missing values for Cabin.

After eliminating outliers and computing missing values, all the features in the training and testing data are converted to integers as seen in Figure 8 below. The Cabin features are represented by integer 0 to 8, Embarked features are represented by integer 0 to 2, and Sex as 0 for male and 1 for female. Afterwards, further visualizations such as Correlation Matrix and Plot Pairwise Relationship can be made using the integer feature values as seen previously in Figure 4.

	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	3	0	34.5	0	0	7.8292	8	2
1	3	1	47.0	1	0	7.0000	8	0
2	2	0	62.0	0	0	9.6875	8	2
3	3	0	27.0	0	0	8.6625	8	0
4	3	1	22.0	1	1	12.2875	8	0

Figure 8: train.head() is run to show the first few rows of the training data.

3.2 Feature Selection

In addition to visualizing the features, I also applied Feature Importance using Extra Tree Classifier (https://machinelearningmastery.com/feature-selection-machine-learning-python/). From the Extra Tree Classifier, we can determine which feature are most important. The higher the

feature_importance_ number, the more important the features are. The Age and Cabin features are identified as the most important features.

```
###################################
# Feature Selection Through ExtraTreesCkassufer
###################################
from sklearn.ensemble import ExtraTreesClassifier
```

```
array = train.values
X = array[:,0:7]
Y = array[:,7]
```

```
# feature extrations
model = ExtraTreesClassifier(n_estimators = 10)
model.fit(X, Y)
```

```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

```
print(model.feature_importances_)
```

```
[0.05618038 0.02220112 0.23949922 0.04987457 0.04551744 0.52657955
 0.0601477 ]
```

```
train.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	3	0	22.0	1	0	7.2500	8	0
1	1	1	38.0	1	0	71.2833	2	1
2	3	1	26.0	0	0	7.9250	8	0
3	1	1	35.0	1	0	53.1000	2	0
4	3	0	35.0	0	0	8.0500	8	0

3.3 Standardization

After preprocessing the data, I decided to condition or standardize the training and testing data using scikit-learn Standard Scaler class (https://scikit-learn.org/stable/modules/g https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html). This ensures that the features are normally distributed. From past machine learning experience, this step significantly improves the model performance.

```
# normalized training and testing data
scaler = StandardScaler().fit(train)
```

```
scaled_train = scaler.transform(train)
```

```
scaled_test = scaler.transform(test)
```

3.4 Train Test Split

The training data are then split 90% for training and 10% for validation. The cross validation will help us optimize and refine the model. I used the scikit-learn to train_test_split the data (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html). The test.csv testing data will be used for the end model evaluation.

```
x_train, x_cv, y_train, y_cv = train_test_split(scaled_train, label, test_size=0.10, random_state=77)
```

3.5 Implementation

As stated previously, for benchmarking, I compared the four different machine learning algorithms (SVC, Decision Tree, Random Forest, and KNN) and evaluate the mean accuracy. Figure 9 shows accuracy of the four benchmark algorithms using standard hyperparameters. These benchmarks will be compared to the optimized models to see how much improvements have been made.

	Accuracy
Support Vector	0.877778
Decision Tree	0.800000
Random Forest	0.800000
K-Nearest Neigbors	0.800000

Figure 9: Accuracy of initial “benchmark” models with standard hyperparameters.

Then, each model hyperparameters are optimized using GridSearch CV class. Each algorithm creates a grid with all the possible hyperparameters and get model with each possibility. The optimized models should have a higher accuracy score than the benchmark models. The following represents the GridSearch CV applied to each model:

1. Support Vector Machine

```
#################################
# Support Vector classifier
#################################
svc_predictor = SVC() #Pipeline([('scaller', MinMaxScaler()), ('clf', SVC())])

svc_param_grid ={"kernel": ['linear'],
                 "gamma": [ 0.001, 0.01, 0.1, 1, 10, 'scale'],
                 "C": [1, 10, 100, 1000]}

svc_search = GridSearchCV(svc_predictor, svc_param_grid, n_jobs = -1, cv = 3, scoring='accuracy', verbose = 1)

svc_search.fit(x_train, y_train)
svc_estimator = svc_search.best_estimator_
print(svc_estimator)
svc_score = svc_search.best_score_
print(svc_score)

Fitting 3 folds for each of 24 candidates, totalling 72 fits

[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent workers.

SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
0.7852684144818975

[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  6.0min finished
```

	Prediction 0	Prediction 1
Actual 0	55	8
Actual 1	9	18

2. Decision Tree

```
###################################
# Decision Tree classifier
###################################

dtc_predictor = DecisionTreeClassifier() # Pipeline([('scaller', MinMaxScaler()), ('clf', DecisionTreeClassifier())])
```

```
dtc_param_grid ={"criterion" : ["gini", "entropy"],
                 "max_depth" : [2,4,6,8,10,12]}
```

```
dtc_search = GridSearchCV(dtc_predictor, dtc_param_grid, n_jobs = 2, cv = 3,  scoring='accuracy', verbose = 1)
```

```
dtc_search.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  36 out of  36 | elapsed:    1.9s finished
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=2,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

```
dtc_estimator = dtc_search.best_estimator_
print(dtc_estimator)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=8, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

	Prediction 0	Prediction 1
Actual 0	57	7
Actual 1	7	19

3. Random Forest

```
###############################
# Random Forest classifier
###############################

rfc_predictor = RandomForestClassifier() #Pipeline([('scaller', MinMaxScaler()), ('clf', RandomForestClassifier())])
```

```
rfc_param_grid = {"max_depth": [None, 1, 2, 3],
                  "max_features": [1, 3, 10],
                  "min_samples_split": [2, 3, 10],
                  "min_samples_leaf": [1, 3, 10],
                  "bootstrap": [False],
                  "n_estimators" :[100, 300, 500, 700, 1000],
                  "criterion": ["gini", "entropy"]}
```

```
rfc_search = GridSearchCV(rfc_predictor, rfc_param_grid, n_jobs = 2, cv = 3, scoring='accuracy', verbose = 1)
```

```
rfc_search.fit(x_train, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,...
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=2,
             param_grid={'bootstrap': [False], 'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 1, 2, 3],
                         'max_features': [1, 3, 10],
                         'min_samples_leaf': [1, 3, 10],
                         'min_samples_split': [2, 3, 10],
                         'n_estimators': [100, 300, 500, 700, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

```
rfc_estimator = rfc_search.best_estimator_
print(rfc_estimator)
```

```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features=3,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=700,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
rfc_score = rfc_search.best_score_
print(rfc_score)
```

```
0.8164794007490638
```

	Prediction 0	Prediction 1
Actual 0	58	8
Actual 1	6	18

4. K-Nearest Neighbors

```
################################
# K-Nearest Neighbors classifier
################################

knn_predictor = KNeighborsClassifier() #Pipeline([('scaller', MinMaxScaler()), ('clf', KNeighborsClassifier())])
```

```
knn_param_grid = {"n_neighbors": [1,2,3,5,11,19],
                  "weights": ['uniform', 'distance'],
                  "metric": ['euclidean','manhattan']}
```

```
knn_search = GridSearchCV(knn_predictor, knn_param_grid, n_jobs = 2, cv = 3, scoring='accuracy', verbose = 1)
```

```
knn_search.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  72 out of  72 | elapsed:    0.8s finished
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=2,
             param_grid={'metric': ['euclidean', 'manhattan'],
                         'n_neighbors': [1, 2, 3, 5, 11, 19],
                         'weights': ['uniform', 'distance']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

```
knn_estimator = knn_search.best_estimator_
print(knn_estimator)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='manhattan',
                     metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                     weights='uniform')
```

```
knn_score = knn_search.best_score_
print(knn_score)
```

```
0.8064918851435706
```

	Prediction 0	Prediction 1
Actual 0	58	7
Actual 1	6	19

There are various challenges in this project. First, there is a challenge in optimizing the hyperparameters and using the GridSearchCV class. The correct combination of hyperparameters must be selected to optimize the machine learning model. Second, there is a challenge in feature selection. Other than optimizing the model hyperparameters, another way to improve the accuracy score is to select the correct features to be used during training. Feature selection allows the machine learning algorithm to train faster as it reduces the complexity of a model making it easier to interpret. Third, I learned that the accuracy significantly improved through standardizing the training and testing data.

3.6 Refinement

From standardizing the data, optimizing the hyperparameters, and applying the necessary features, I was able to refine and improve the performance accuracy of my machine learning model. Figure 10 summarizes the validation accuracy of the optimized machine learning models.

	Accuracy
Support Vector	0.877778
Decision Tree	0.800000
Random Forest	0.800000
K-Nearest Neigbors	0.800000

	Accuracy
Support Vector	0.785268
Decision Tree	0.803995
Random Forest	0.816479
K-Nearest Neigbors	0.806492

Figure 10: Accuracy of initial “benchmark” ML models (*left*) vs accuracy of optimized machine learning models (*right*) with standard hyperparameters.

4 Results

4.1 Model Evaluation and Validation

In the end, the solution is a machine learning model that predicts if the passenger would survive based on the given passenger information and features. The model was tested to predict which passengers survive the Titanic. Comparing the four initial benchmark models with the optimized models, all the algorithm training accuracy performance improved except for Support Vector Classifier. The Random Forest seemed to perform with the highest accuracy, thus I decided to use the optimized Random Forest Classifier as my final machine learning predictor. The following shows the optimized Random Forest Classifier hyperparameters:

```
rfc_estimator = rfc_search.best_estimator_
print(rfc_estimator)
```

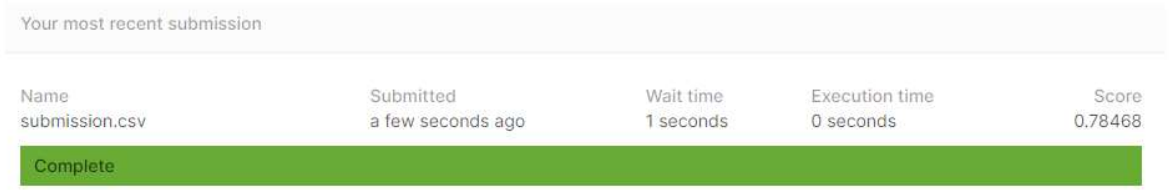
```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features=3,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=700,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

The Random Forest Classifier is used to predict the outcome of the Titanic passengers in the testing data set. The final testing accuracy of this Random Forest Classifier is **78.46 %.**

```
test_Survived = pd.Series(rfc_estimator.predict(scaled_test), name="Survived")
results = pd.concat([test_Survived],axis=1)

results.to_csv("submission.csv",index=False)
# I added the PassengerID first column manually
```

Final Testing Accuracy Score for RFC_Estimator = 78.46 percent



4.2 Justification

In the end, the results (models with optimized hyperparameters) were compared to the initial benchmark results (models with standard hyperparameters). The model performance to predict the passenger outcomes in the Titanic significantly improved after optimizing the hyperparameters through GridSearchCV and feature selection.

The results from the Random Forest classifier with an accuracy of 78.46 % is justifiable. As seen in Figure 13 of the training accuracy validation score, for the optimized Random Forest model, there are a total of 76 True Negatives and True Positives which are much higher when compared to other models and its initial benchmark model. This means that the optimized Random Forest classifier made better prediction of which passenger survives and died in the Titanic.

	Prediction 0	Prediction 1
Actual 0	58	8
Actual 1	6	18

Figure 13: Random Forest classifier with training validation accuracy of 81.67 %.