Kaggle Competition: Titanic

João Leal

Explaining decisions

Well, the first thing observed was that some data were missing in the train dataset and some features had qualitative data (they were not numerical values).

```
1 train.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	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
1 # humm... there are missing data. How many are there actually?
2 train.isna().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
```

These problems would have to be solved, but firstly I decided to remove the following features based on the assumptions:

- 'PassengerID' does not affect the outcoming (if the person survived or not);

- 'Name' could affect the outcoming if it was an "important" (rich) person. However, this is already indirectly included in the 'Pclass' feature! Thus, I can discard this column;
- 'Ticket' is equivalent to 'PassengerID'. Thus, I could discard it as well.

Thus, those features were dropped and the new train dataset has become the following:

```
1 train = train.drop(['PassengerId','Name', 'Ticket'], axis=1)
```

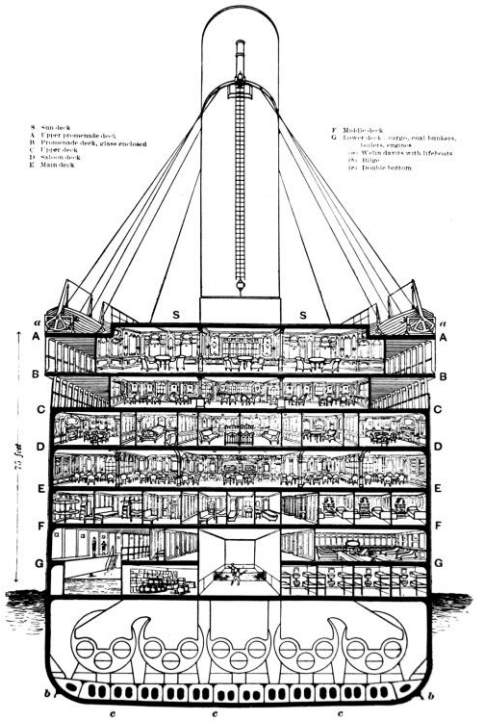
```
1 train.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S

To solve the missing data in the features ‘Cabin’, ‘Embarked’ and ‘Age’, a different approach for each one was used.

Missing data in ‘Cabin’

This is a trick one, actually. There are a lot of missing data, which could justify the discard of this feature. However, based on some internet figures of the Titanic, we see that the letter in the cabin’s code is related to which floor it was located.



Source: [First-class facilities of the Titanic - Wikipedia]

Since the Titanic has drown, I presume the people in the superior cabins would have reached the top floor faster than the others from below. Perhaps their survival chance would have increased with this…

Thus, first of all I removed the numbers in the cabin's code, keeping only the letter (where it was NaN before was kept the same). Maybe the numbers could have given some position information about the cabins, but I dropped it in understanding the letters would be more important.

After, I grouped then in three quantitative categories: 1 for cabins 'A', 'B' and 'C' (top); 2 for cabins 'D' and 'E' (middle); 3 for 'F', 'G' and 'T' (bottom). I actually don't know which floor 'T' stands for, so I've put it in the bottom group. Anyway, all this division was arbitrary, although based on the Titanic image.

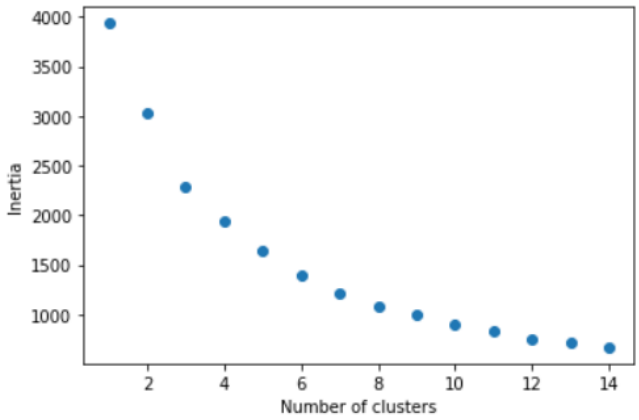
The next step was to treat the missing values. Since there was an economic factor in the cabins (cabins in the top are more expensive, while in the bottom they are cheaper), I believe an estimative to the person's cabin could be obtained from the others features (especially 'PClass' and 'Fare'). The idea, then, was to cluster the data in groups that are more similar and, with that, use the

most common cabin value in the cluster to replace the missing data of a person that the model predicts to belong to this same group.

```
kmeans_internal_distance = np.array([])
n = 15
```

```
# using KMeans
for num_clusters in range(1,n):
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(auxiliar_train)
    kmeans_internal_distance = np.append(kmeans_internal_distance, kmeans.inertia_)
```

```
# Let's check the internal distances as a function of the number of clusters.
fig = plt.figure()
plt.scatter(range(1,n), kmeans_internal_distance)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



K-means was the chosen model. It was used a subset of the train dataset without the features with missing data (auxiliar_train). At this point, ‘Embarked’ was already fixed.

```
auxiliar_train = train.drop(['Survived', 'Age', 'CabinPosition'], axis=1)
```

```
auxiliar_train.head()
```

	Pclass	Sex	SibSp	Parch	Fare	Embarked
0	3	0	1	0	-0.502445	2.0
1	1	1	1	0	0.786845	0.0
2	3	1	0	0	-0.488854	2.0
3	1	1	1	0	0.420730	2.0
4	3	0	0	0	-0.486337	2.0

The row indexes of the instances in each cluster were stored in five different arrays. With that, the train data was separated and the most common occurrence of the ‘Cabin’ in each cluster was obtained.

```
# With five clusters, I'm going to create five arrays to store the instance row number in each case.
cluster_0 = np.array([], dtype=int)
cluster_1 = np.array([], dtype=int)
cluster_2 = np.array([], dtype=int)
cluster_3 = np.array([], dtype=int)
cluster_4 = np.array([],dtype=int)
```

```
for indx, cluster_label in enumerate(kmeans.labels_):
    if cluster_label == 0:
        cluster_0 = np.append(cluster_0, indx)
    elif cluster_label == 1:
        cluster_1 = np.append(cluster_1, indx)
    elif cluster_label == 2:
        cluster_2 = np.append(cluster_2, indx)
    elif cluster_label == 3:
        cluster_3 = np.append(cluster_3, indx)
    else:
        cluster_4 = np.append(cluster_4, indx)
```

First of all, I will replace the 'Age'. The idea is to get the average and the standard deviation from the instances in each cluster and use it to replace the missing data with random samples from a Normal distribution.

```
cluster_0_train_data = train.iloc[cluster_0,:]
cluster_1_train_data = train.iloc[cluster_1,:]
cluster_2_train_data = train.iloc[cluster_2,:]
cluster_3_train_data = train.iloc[cluster_3,:]
cluster_4_train_data = train.iloc[cluster_4,:]
```

```
# Replacing np.nan for the most common ocurrence in the cluster

clusters = [cluster_0, cluster_1, cluster_2, cluster_3, cluster_4]
cabin_cluster_value = []

for cluster in clusters:

    top_counting = 0
    middle_counting = 0
    down_counting = 0

    for row in cluster[train.iloc[cluster,8].isna() == False]:
        # Find the most common ocurrence
        if int(train.iloc[row,8]) == 1:
            top_counting += 1
        elif int(train.iloc[row,8]) == 2:
            middle_counting += 1
        else:
            down_counting += 1

    # Replace the missing values
    for row in cluster[train.iloc[cluster,8].isna() == True]:
        # armax return the index of the highest value in an array. Since the categorical values are 1, 2 and 3,
        # it is possible to use this function with a well chosen list.
        train.iloc[row,8] = np.argmax([top_counting, middle_counting, down_counting]) + 1

    cabin_cluster_value.append(np.argmax([top_counting, middle_counting, down_counting]) + 1)
```

Missing data in ‘Age’

To replace the missing data in the ‘Age’ feature, I used the same approach of clusters. However, in contrast to using the most common value, the people age is not a few categories. The idea, then, was to use a random value extracted from a gaussian distribution with mean and standard deviation obtained from the clusters. The goal was to keep a similar age distribution in the clusters while the missing data were replaced.

```
# Replacing np.nan for a random.gauss(average, standard Deviation)
for row in cluster_0[train.iloc[cluster_0,3].isna() == True]:
    train.iloc[row,3] = random.gauss(cluster_0_train_data['Age'].mean(), cluster_0_train_data['Age'].std())
```

```
for row in cluster_1[train.iloc[cluster_1,3].isna() == True]:
    train.iloc[row,3] = random.gauss(cluster_1_train_data['Age'].mean(), cluster_1_train_data['Age'].std())
```

```
for row in cluster_2[train.iloc[cluster_2,3].isna() == True]:
    train.iloc[row,3] = random.gauss(cluster_2_train_data['Age'].mean(), cluster_2_train_data['Age'].std())
```

```
for row in cluster_3[train.iloc[cluster_3,3].isna() == True]:
    train.iloc[row,3] = random.gauss(cluster_3_train_data['Age'].mean(), cluster_3_train_data['Age'].std())
```

```
for row in cluster_4[train.iloc[cluster_4,3].isna() == True]:
    train.iloc[row,3] = random.gauss(cluster_4_train_data['Age'].mean(), cluster_4_train_data['Age'].std())
```

Missing data in ‘Embarked’

Since only two data values were missing in the ‘Embarked’ feature (out of 891), I replaced them with the categorical value closest to the median.

```
1 train['Embarked'].replace([np.nan], round(train['Embarked'].median()), inplace=True)
```

Standardizing the data

Two features have values way higher than the others: ‘Age’ and ‘Fare’. If not standardized, these columns will receive a higher weight when the models are being trained then it would be correct. Thus, the StandardScaler is used.

```
std_scaler = StandardScaler()
std_scaler.fit(pd.DataFrame(train['Fare'], columns=['Fare']))
train['Fare'] = std_scaler.transform(pd.DataFrame(train['Fare'], columns=['Fare']))
```

```
std_scaler_age = StandardScaler()
std_scaler_age.fit(pd.DataFrame(train['Age'], columns=['Age']))
train['Age'] = std_scaler_age.transform(pd.DataFrame(train['Age'], columns=['Age']))
```

The test data is also standardized, but it will use the same scaler fitted for the training data.

With all that done, the train set is ready to be used.

```
1 train.isna().sum()
```

```
Survived         0
Pclass           0
Sex              0
Age              0
SibSp            0
Parch            0
Fare             0
Embarked         0
CabinPosition    0
dtype: int64
```

```
1 train.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	CabinPosition
0	0	3	0	-0.505183	1	0	-0.502445	2.0	3.0
1	1	1	1	0.615591	1	0	0.786845	0.0	1.0
2	1	3	1	-0.224989	0	0	-0.488854	2.0	3.0
3	1	1	1	0.405446	1	0	0.420730	2.0	1.0
4	0	3	0	0.405446	0	0	-0.486337	2.0	3.0

Training and validating some models

There is not a perfect model for every dataset. Thus, the idea was to train some models and verify how well they can perform over the titanic data. At the end, the output will weigh the response of each model based on their performance to compute a final vote for the classification.

The models used were SVM, Random Forest, Logistic Regression, AdaBoost, KNN, Gaussian Naive Bayes and Bernoulli Naive Bayes.

```
Support Vector Machine
Accuracy:  0.8226711560044894
F1 score:  0.8226711560044894
Precision score:  0.8226711560044894
Recall score:  0.8226711560044894
Confusion Matrix
[[489  60]
 [ 98 244]]


Random Forest
Accuracy:  0.7890011223344556
F1 score:  0.7890011223344556
Precision score:  0.7890011223344556
Recall score:  0.7890011223344556
Confusion Matrix
[[528  21]
 [167 175]]

Logistic Regression
Accuracy:  0.7946127946127947
F1 score:  0.7946127946127947
Precision score:  0.7946127946127947
Recall score:  0.7946127946127947
Confusion Matrix
[[478  71]
 [112 230]]

AdaBoost
Accuracy:  0.8035914702581369
F1 score:  0.8035914702581368
Precision score:  0.8035914702581369
Recall score:  0.8035914702581369
Confusion Matrix
[[466  83]
 [ 92 250]]
```

```
K Nearest Neighbors
Accuracy:  0.7586980920314254
F1 score:  0.7586980920314254
Precision score:  0.7586980920314254
Recall score:  0.7586980920314254
Confusion Matrix
[[498  51]
 [164 178]]

Gaussian Naive Bayes
Accuracy:  0.7676767676767676
F1 score:  0.7676767676767676
Precision score:  0.7676767676767676
Recall score:  0.7676767676767676
Confusion Matrix
[[454  95]
 [112 230]]
------------------------------
Bernoulli Naive Bayes
Accuracy:  0.7710437710437711
F1 score:  0.7710437710437711
Precision score:  0.7710437710437711
Recall score:  0.7710437710437711
Confusion Matrix
[[442 107]
 [ 97 245]]
```

The order of best results is (in ascending order): KNN, GaussianNB, BernoulliNB, Random Forest, AdaBoost, Logistic Regression and SVM. Their weights were given in a trial-and-error method, based on their results.

```
# weights = [KNN, GaussianNB, BernoulliNB, Random Forest, ADA, Logistic Regression, SVM]
weights = [0.8, 0.8, 0.8, 1, 1.5, 2 ,2.5]
```

The vote system in the ensemble method was quite simple: for each method whose output was of one classification (say 1), a variable summed their weights. The same is done for the other classification (0). In the end, the program

compares which variable is greater – the one representing the output 1 or the one representing the output 0 – and returns which classification had the highest weight sum. This will be the output.

```
y_true = np.array([])
y_predicted = np.array([])
models = []

kfold = KFold(n_splits=5, shuffle=True)

for train_index, validation_index in kfold.split(train):
    # Start separating the target from the features
    X_train, X_validation = train.values[train_index], train.values[validation_index]

    y_train = X_train[:, 0]
    X_train = X_train[:, 1:]
    y_validation = X_validation[:,0]
    X_validation = X_validation[:,1:]

    models = []

    models.append(KNeighborsClassifier(n_neighbors=2, metric='euclidean', algorithm='kd_tree'))
    #knn.fit(X_train, y_train)
    models.append(GaussianNB())
    #gnb.fit(X_train, y_train)
    models.append(BernoulliNB())
    #bnb.fit(X_train, y_train)
    models.append(RandomForestClassifier(n_estimators=100,max_depth=2, random_state=None))
    #random_forest.fit(X_train, y_train)
    models.append(AdaBoostClassifier(n_estimators=100, random_state=None))
    #ada.fit(X_train, y_train)
    models.append(LogisticRegression(penalty='l2', C=0.1))
    #log_reg.fit(X_train, y_train)
    models.append(SVC(gamma='auto', kernel='rbf', random_state=None))
    #svc.fit(X_train, y_train)
```

```
    for m in models:
        m.fit(X_train, y_train)

    for indx, features in enumerate(X_validation):
        y_true = np.append(y_true, y_validation[indx])

        weight_for_not_survived = 0 # survived = 0
        weight_for_survived = 0 # survived = 1

        for i, model in enumerate(models):
            prediction = int(model.predict([features])[0])
            if prediction == 1:
                weight_for_survived += weights[i]
            else:
                weight_for_not_survived += weights[i]

        if weight_for_survived > weight_for_not_survived:
            y_predicted = np.append(y_predicted, 1)
        else:
            y_predicted = np.append(y_predicted, 0)

print('Ensemble model')
print('Accuracy: ', accuracy_score(y_true, y_predicted))
print('F1 score: ',f1_score(y_true, y_predicted, average='micro'))
print('Precision score: ',precision_score(y_true, y_predicted, average='micro'))
print('Recall score: ', recall_score(y_true, y_predicted, average='micro'))
print('Confusion Matrix')
print(confusion_matrix(y_true, y_predicted))
```

The metrics result for this ensemble method was the following:

```
Ensemble model
Accuracy:  0.8058361391694725
F1 score:  0.8058361391694725
Precision score:  0.8058361391694725
Recall score:  0.8058361391694725
Confusion Matrix
[[479  70]
 [103 239]]
```

The ensemble model does not score better than the SVM model. However, the reason for keeping using it will become clear with the score in Kaggle.

Test set

With the models trained, it was time to use the test set. Since it does not have a ‘Survived’ feature, it is not possible the evaluate how well the models predict the target.

```
1 test = pd.read_csv('test.csv')
```

```
1 test.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
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```
1 test.isna().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
```

Different from the train set, in the test set there is one instance which the ‘Fare’ feature does not contain any value.

The idea to solve this problem was to replace the missing data with the average from the passengers in the same ‘PClass’. The lower the ‘PClass’ value, more expensive the ‘Fare’ will be.

```
# I'm going to replace the missing data in 'Fare' with the average of people in the same 'Pclass'
test[test['Fare'].isna() == True]
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
152	1044	3	Storey, Mr. Thomas	male	60.5	0	0	3701	NaN	NaN	S

```
p_class_index = np.array([])
```

```
for indx, p_class in enumerate(test['Pclass']):
    if p_class == test.iloc[152, 1]:
        p_class_index = np.append(p_class_index, indx)
```

```
test.iloc[152,8] = test.iloc[p_class_index, 8].mean()
```

```
test.isna().sum()
```

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

After that, the same procedures to preprocess the train set were used in the test set – the same kmeans and standard scalers fitted for the train set were used to predict and scale the test instances, respectively. The result was a test set ready to be used in the trained models.

```
1 test.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	CabinPosition
0	3	0	0.331874	0	0	-0.490783	1	3.0
1	3	1	1.190494	1	0	-0.507479	2	3.0
2	2	0	2.220838	0	0	-0.453367	1	1.0
3	3	0	-0.183298	0	0	-0.474005	2	3.0
4	3	1	-0.526746	1	1	-0.401017	2	3.0

Results

As it was said before, the test set does not have the ‘Survived’ feature to score the models. To evaluate the output then it is necessary to send it to the Kaggle site in a format of the type:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1

Where the ‘Survived’ column must contain the model predictions for the test set.

Using only the SVM model, the prediction received the following score:

But with the ensemble model, the result was a little better:

This show us that although the SVM model had a better score in the train set, this is not necessarily true for the unseen data of the test set.

In general, the result score is far from the best, but it is far from the worst either. In the Kaggle Titanic competition rank, this score stays between the positions 2586 and 3251 out of 14658 at the data when this document is being written (02/05/2022).

Since I didn't search for any tutorial in this competition, I am still proud of this result.