AYADI Imen – BERGAOUI Khalil – HACHICHA Mohamed Amine – LETAIEF Zeineb

Assignment 2 : Titanic Kaggle Competition

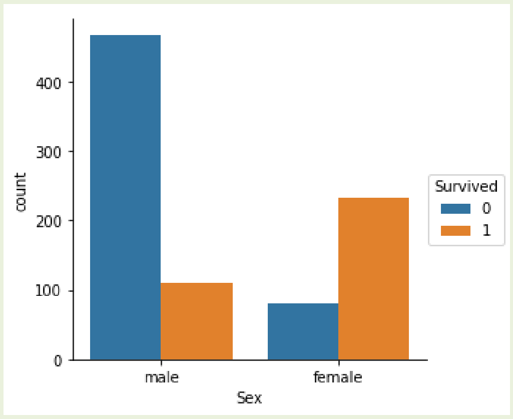
Team name : TeamCentrale

I. Feature engineering

1. Data exploration and visualisation :

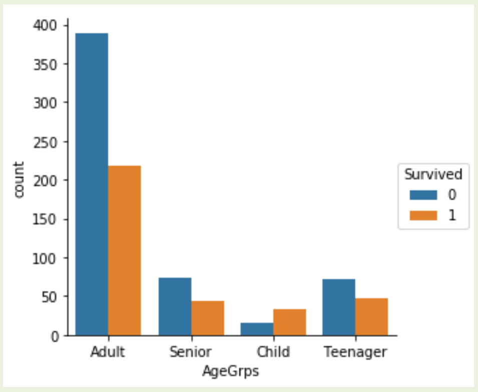
We started by taking a look at the available data in the training set. Most features are categorical (except for Age and Fare) that's why it is not possible to do a correlations plot.

Gender versus survival bar plot



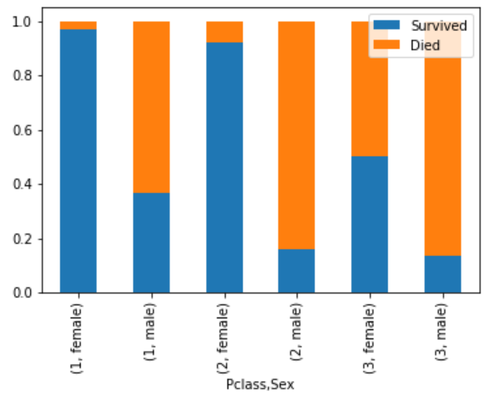
It's clear that females have a greater chance of survival than male individuals. Passenger's gender is therefore decisive.

Age category versus survival plot



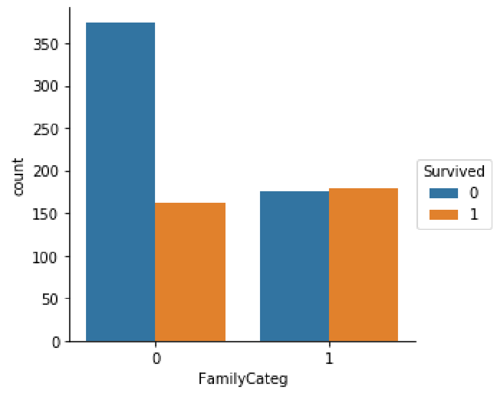
Children are more likely to survive while adults are most likely to not make it.

Ticket class and gender versus survival bar plot



It's clear that higher is the ticket's class, higher are the chances of survival for its' holder. In particular, almost all females from classes 1 and 2 survived the accident.

Family category versus survival



Category 0 represents individuals who came alone or with a big size family (4 members or more). Category 1 is for small size families. We note that the second one is more likely to survive.

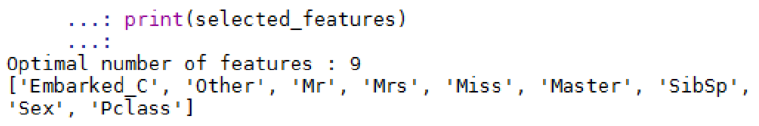
We did also note that passengers ***embarked*** in Cherbourg and Queenstown have greater chances to survive than those embarked in Southampton and people who paid the cheapest ***Fares*** are more likely to die. And finally, the majority of cabin numbers are unavailable.

2. Data preprocessing:

1. We replaced the Sex feature by a binary feature (0 for male and 1 for female) in order to be exploitable in classification algorithms.
2. We created a feature 'Title' that contains the title (Mr, Mrs, Miss, Master or Other) which can be extracted from the passenger's name.
3. We created 5 binary features: Mr, Master, Miss, Mrs and Other which contains 1 if the passengers belongs to the respective category. In fact we observed through visual plotting that some Titles are more likely to survive than others (especially non Mr) and is useful for Age filling (point 4).
4. Feature Age has around 20% of missing data in the training set. We chose to fill missing values with the mean of ages of other passengers who have the same title. For example, we fill a Master's missing age with the mean age of other Masters.
5. The cabin feature could be useful because it gives the distance from passenger's cabin to Deck. Unfortunately, 77% of data are missing so we chose to discard this feature.
6. Embarkment port has 2 missing values that we filled with 'S' (most frequent Port). We then splitted the 'Embarked' into 3 binary features: Embarked_Q, Embarked_S and Embarked_C.

3. *Features selection and dimensionality reduction:*

To select most important features we used the RFE method with logistic regression as a classifier. Out of the 15 features (both originally existing or created with preprocessing), the algorithm chooses 9 of them which are:



```
        ...: print(selected_features)
        ...:
Optimal number of features : 9
['Embarked_C', 'Other', 'Mr', 'Mrs', 'Miss', 'Master', 'SibSp',
'Sex', 'Pclass']
```

We see that RFE discards feature 'Age', maybe because passengers titles already give an idea about the age category. We choose later to add feature 'Age' to our learning models because we saw that it betters performance on both training, test, and invisible Kaggle sets.

The dimension of the dataset is now less than 10. When we tried to apply PCA on it to retain for example the 5 most important components, we noted that it doesn't affect accuracies on predicting the training set but it considerably increases our submission accuracy. That's why we decided to not use PCA.

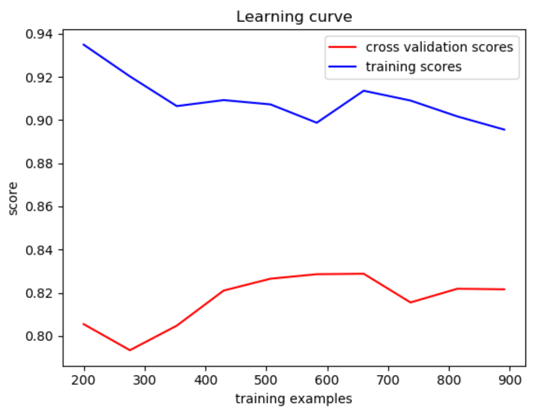
II. *Learning algorithms*

1- Random Forest:

We decided to try an ensemble training algorithm, in particular Random Forest algorithm. It is very popular amongst machine learning methods and generally provides good results.

Ensemble methods work by combining predictions of base estimators with a given learning algorithm, in order to improve results and generalizability.

In the case of random forest, base estimators are decision trees, it fits them on the data set (or various sub-samples of the data set).



Manual Forest model tuning : The Sklearn classifier we used has many parameters, with some of course

more important than others, first we conducted a few manual tries tweaking the parameters we found most interesting. For the parameter "n_estimators" as well as "max_depth" of the decision trees if they have values that are too big it can easily lead to overfitting so we start by setting a small value and see how the score varies when we gradually augment it.
To evaluate the model we use cross validation and compare with simple training.

Model tuning with GridSearchCV: We then try and tune the model's hyperparameters using sickit learn tools: RandomizedSearchCV and GridSearchCv, which both use kfold cross validation to train the model and find the best suited parameters. We start by defining a parameter grid with radom ranges. Bases on the first result provided by the RandomisedSearch we can refine our grid and use gridSearchCv

2- ***Decision Tree***

We used a DecisionTreeClassifier. Only "max_depth" parameter of this sklearn classifier was tuned.

Tree depth tuning: To determine the maximum depth parameter without overfitting our model, we used the test error as a reference. We tested the classifier for different values of the parameter "max_depth" and calculated the corresponding test error using Cross-Validation.

This gives us the value of the tree depth parameter which minimizes the test error and avoids overfitting (In our example max_depth =4)

3- ***Additional tested models with low performance:***

3.1 K-Nearest neighbors:

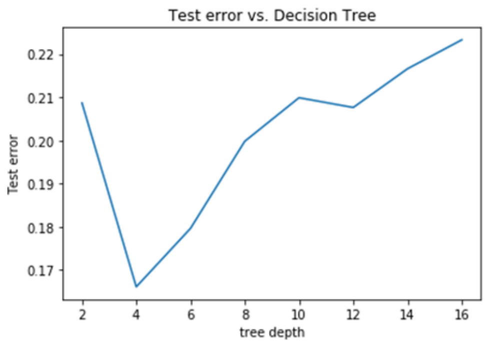
We used the KNeighborsClassifier. Only number of nearest neighbors parameter "k" of this sklearn classifier was tuned.
Nearest neighbors tuning: We used the same tuning method as in Decision Tree : We tested the classifier for different values of the parameter "k" and calculated the corresponding test error using Cross-Validation.

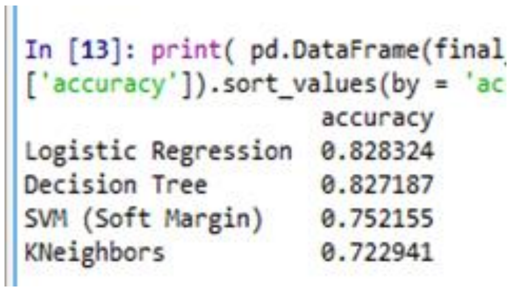
3.2 SVM (Soft Margin):

We used the SVC classifier. Only error penalizing parameter "C" of this sklearn.svm classifier was tuned. We used the same method as for kNN.

Note: We put the different learning algorithms and their tuning in the file codes.py. This file cannot be executed at once, but a block of coding lines for each algorithm that we used.

III. ***Evaluation on training set:***





```
In [13]: print( pd.DataFrame(final
['accuracy']).sort_values(by = 'ac
                        accuracy
Logistic Regression     0.828324
Decision Tree           0.827187
SVM (Soft Margin)       0.752155
KNeighbors              0.722941
```

Performance on Kaggle: For our submission we used Random Forest since it was the best performing algorithm on the training data. We got a score of 0.79904 so we had approximatively 80% of correct perdictions.