AWS Machine Learning Engineer Nanodegree - capstone project

I. Definition

I.I. Project Overview

RMS Titanic was designed to be the more luxurious and safest ship built in 20th century. On the night of April 20, the Titanic hit an iceberg and sink in the middle on its journey. Unfortunately, due to the low number of rescue boats, more than a half of the passengers have died. The survive number was only 722 of 2224 in total.

The project proposal is to build a predictor model that recieves, as input, passenger information (like name, age, gender, socio-economic and class), makes the text preprocessor, guesses if this fictitious passenger would survive or not in Titanic tragedy and return it as a HTTP response. As an experiment, supervised and unsupervised machine learning algorithms will be used to build and improve the model. To go further and receive theses passenger information, an endpoint will be develop using Python Frameworks in order to demonstrate another away to create endpoints, instead of those shown during the Nanodegree Program using AWS.

The main idea of this project is to put into practice all the machine learning and software engineer knowledge learned during the Machine Learning Engineer Nanodegree Program and join it into my developed skills as a Software Developer.

I.II. Problem Statement

The problem is Kaggle challenge and can be access here (https://www.kaggle.com/c/titanic). Based on the passenger data, the challenge is to build a predictive model that answers the question:

> “what sorts of people were more likely to survive?”

In others words. The idea is to use the provided dataset, which contains all informations about the passenger aboard Titanic in 1912, and build a machine learning model that predicts if the passenger would survive, based on new data received.

To build a great predictor model, supervised machine learning classification algorithms will be used, like K-Nearest Neighbors (KNN), Naive Bayes, Random Forest and Support Vector Machines (SVM). Due to the labeled dataset provided, which indicates if the passenger survived or not.

I.III. Metrics

For the model evaluation, the follow metrics are used to measure how good the model is.

- **Accuracy Score**: value that indicates how many predicts the model guessed that the passenger would survived and guessed right, comparing to the total data sent. In other words, is the total number of True Positive and True Negatives divided by the total number of samples.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All Samples}}$$

- **Confusion Matrix**: matrix that indicates how many True Negatives, False Positives, False Negatives and True Positives. Our goal is to increase the number of True Negatives and True Positives, which show that the model more guessing right than wrong.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

II. Analysis

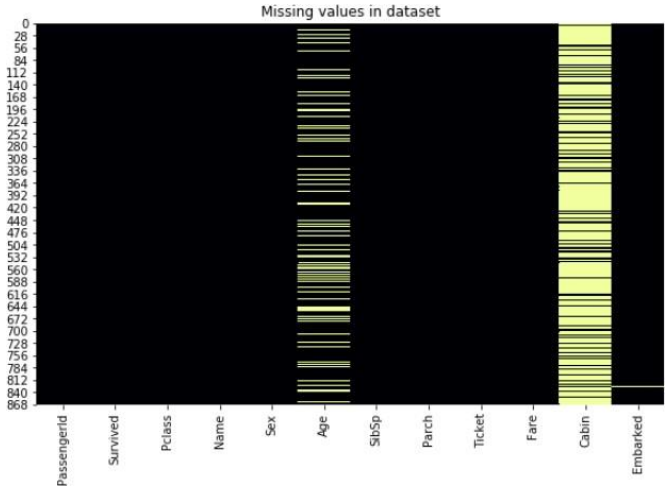
II.I Data Explation

II.I.I Dataset

The dataset brings informations about passenger onboard on RMS Titanic that have survived or not on the night of tragedy. Each row contains unique passengers with different information about them, from name and parents onboard to the amount paid on the ticket.

	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

Over the 890 rows in the dataset, 866 cells have missing values and the most of those values are in the Cabin column. Due to the quantity of null values in that column, it can't be used as features to the model. In other hand, features like Age, which also have null values, but they can be filled out with mean or median. The follow heatmap plot shows that in more details.



Whites lines indicates how many values are missing in each column. It's clear that Cabin column has almost all of its values as empty. While Age column has less than half and Embarked column only one or two data.

II.I.II Data Statistics

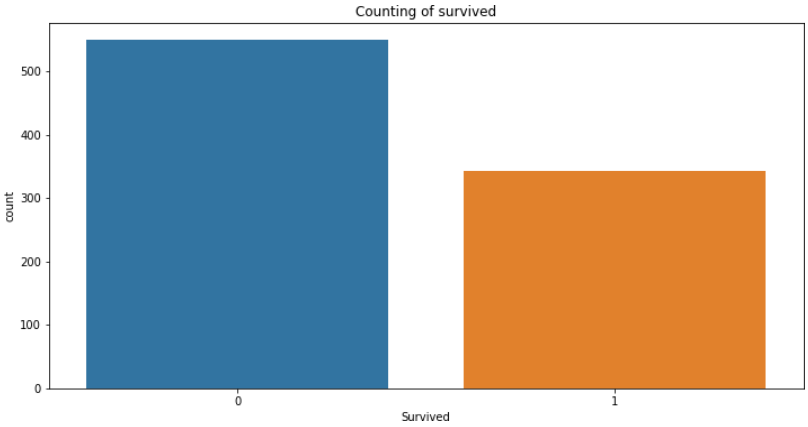
With Pandas, a Python Library, we can easy extract statistics information in numerical columns in a dataset.

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

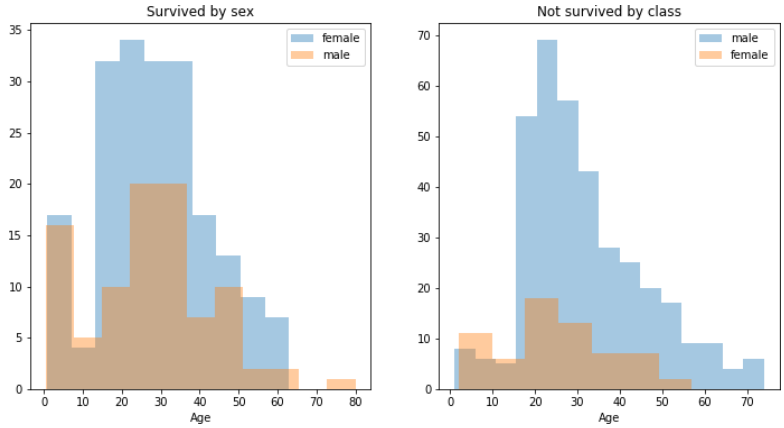
We can see useful information in the table above. It's noticed that the mean of all ages are equal to 29.69 and the missing values in Age column can be filled with it. Another interesting information is the Survived mean value, only 38.38% of the passenger have survived. It's indicates that we have an imbalanced dataset and have to balance it in data preprocessing step.

II.II Exploratory Visualization

It was said before that the dataset is imbalanced and there are more not survived passenger. The next plot show this counting the number survived and not survived passengers. 0 indicates not survived and 1 indicates survived on the X axis.

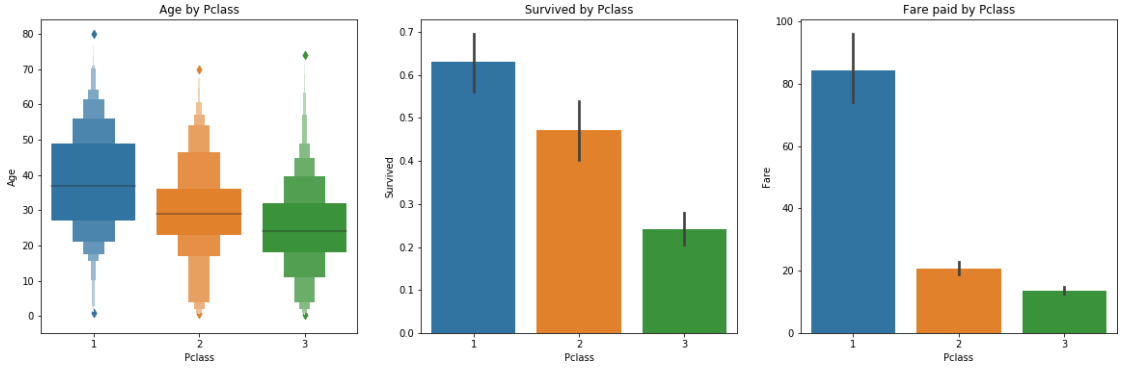


We can go further and see how many person by sex have survived based on their age.



The next plot brings a lot of information about the dataset. The first figure (Age by Pclass) shows the mean, median, min, max and quarters values of the age by passenger class. This kind of plot is useful to use to fill missing values in the Age column. To not drop all null data in Age column, we can fill it with mean age per class, without spoil the entire dataset.

Other information we can extract of figure is which class had more survived passenger. The Survived by Pclass plot shows that the class with the greter number of death was the Pclass 3. This kind of information indicates that, when the rescue boats arrived after the crashed, they prioritize the more fancy class (Pclass number 1). To reiterate this information, the Fare Paid by Pclass plot show that Pclass 1 had the most expensive fare and, consequently, Pclass 3 the cheapest.



II.I Algorithms and Techniques

Scikit-Learn (https://scikit-learn.org/) is a Python open source machine learning library that provides a bunch of data preprocessing algorithms, statistical estimator and metrics score functions. This library will be used as base of all steps of this report (from preprocessing to predictions).

For make prediction of Titanic dataset, it will apply four different algorithms: K-Nearest Neighbors (KNN), Naive Bayes, Random Forest and Support Vector Machines (SVM). The model that provides the best result, it will be deployed lately in custom endpoint to make prediction via HTTP request.

Since our dataset has more than two dimension (more than two features/columns), SVM will form my benchmark. SVM is a great benchmark because it's a common algorithm that provides good results and , in a multidimensional data, it can easily make a regression line. The goal is to reach more than 70% of accuracy and recall in SVM classifier.

KNN and Naive Bayes algorithm is only to make comparison. The first one makes predictions based on the Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance) between points in a Cartesian plane (those points is the dataset features/columns). The second one, also makes predictions based on points in a plane, but instead of use Euclidean distance, it guesses using probability.

The last algorithm used is Random Forest Classifier, Titanic dataset state-of-the-art classifier. Sklearn library defines this algorithm in a great way:

> A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

II.II Benchmark

As mentioned before, SVM it will be our benchmark since it performance well with multidimensional data and can easily reach more than 70% of accuracy. Then, apply other classifier algorithm to get better model and improve its hyperparameters.

III. Methodology

III.I Data Preprocessing

First, let's remove columns data we considered useless. Like PassengerId that only contains passenger index and Cabin, because, it was said before, it has a lot of missing data and do not worth fill it with some data.

```
>>> dataset.drop(['Cabin'], axis=1, inplace=True)
>>> dataset.drop(['PassengerId', 'Name'], axis=1, inplace=True)
>>> dataset.drop(dataset['Embarked'].isna(), axis=0, inplace=True)   # only
 2 data are missing
```

On Exploratory Visualization step, we saw that Age was a column that also has missing values. But, differently of Cabin column, we can fill it with data. Considering the Age By Class plot, we are going to fill those missing age data with the mean age value of their passenger class.

```
>>> dataset.loc[dataset[(dataset['Pclass'] == 1) & (dataset['Age'].isna
())].index, 'Age'] = 38
>>> dataset.loc[dataset[(dataset['Pclass'] == 2) & (dataset['Age'].isna
())].index, 'Age'] = 29
>>> dataset.loc[dataset[(dataset['Pclass'] == 3) & (dataset['Age'].isna
())].index, 'Age'] = 24
```

Also, we can turn the remain string values to number, by mapping them. The Sex column, which has "male" and "female" values in it, it was turn to 1 and 0, respectively. Embarked column also has string values that can be mapped. "S", "C" and "Q" values was changed to 0, 1 and 2, respectively.

```
>>> gender = {'female': 0, 'male': 1}
>>> dataset['Sex'] = dataset['Sex'].map(gender)

>>> embarked = {"S": 0, "C": 1, "Q": 2}
>>> dataset['Embarked'] = dataset['Embarked'].map(embarked)
```

Ticket column has string type and could also turn into int, but it has 680 unique different kind of tickets in dataset. We can also discard this feature.

```
count          889
unique         680
top       CA. 2343
freq             7
Name: Ticket, dtype: object
```

```
>>> dataset.drop(['Ticket'], axis=1, inplace=True)
```

Finally, we can also make more features based on others. We can simplify SibSp and Parch columns in a single column called Relatives, by summing them. Then, we add another column that indicates with this passenger was alone, by verifying the values in the new Relatives column.

```
>>> dataset['Relatives'] = dataset['SibSp'] + dataset['Parch']
>>> dataset.loc[dataset['Relatives'] > 0, 'Alone'] = 0
>>> dataset.loc[dataset['Relatives'] == 0, 'Alone'] = 1
>>> dataset['Alone'] = dataset['Alone'].astype(int)
>>> dataset.drop(['SibSp', 'Parch'], axis=1, inplace=True)
```

With all those changes, the dataset leave from:

	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

To:

	Survived	Pclass	Sex	Age	Fare	Embarked	Relatives	Alone
0	0	3	1	22	7.2500	0	1	0
1	1	1	0	38	71.2833	1	1	0
2	1	3	0	26	7.9250	0	0	1
3	1	1	0	35	53.1000	0	1	0
4	0	3	1	35	8.0500	0	0	1

Note: All the preprocessing steps above was applied in train and test datasets.

III.II Train Test Split

Before put this data in a model, we have to split into a train and test dataset. 85% of the dataset it will be used for training the model, and 25% remaining to evaluation.

```
>>> from sklearn.model_selection import train_test_split

>>> X = dataset.drop('Survived', axis=1)
>>> Y = dataset.Survived
>>> x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2
5, random_state=101)
```

III.III Implementation

III.III.I Benchmark Implementation

As mention before, our benchmark model is a SVM classifier. The predictor will be performed with Sklearn estimators.

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.preprocessing import MinMaxScaler
>>> from sklearn.svm import SVC
```

For the pipeline of the model is used the Sklearn.Pipeline, which joins Sklearn estimator into a unique object. The first step of the pipeline is the Sklearn.MinMaxScaler, which turns each feature of the data into values between 0 and 1 based on the max and the min values of each column. The last step is the own SVM classifier, which receives the normalized data and start the fit process.

```
>>> steps = [('scaller', MinMaxScaler()),
             ('classifier', SVC())]
>>> benchmark_predictor = Pipeline(steps)
>>> benchmark_predictor.fit(x_train, y_train)
```

The accuracy of SVM classifier was **79.82%**, more than expected (70%). But, if we take a look into the confusion matrix, we got a high value of False Positives, which indicates that the model predicts that 37 person survived, but actually they didn't. This number is more than the half of True Positive values.

	Predict 0	Predict 1
Actual 0	128	37
Actual 1	8	50

III.III.II Others Implementation

To compare the benchmark model with the others one (KNN, Naive Bayes and Random Forest), the same pipeline was used, but the "classifier" step was change to the corresponded classifier.

After all training process, the result is shown in the next table.

	Accuracy
Naive Bayes	0.6995515695067265
SVM	0.7982062780269058
KNN	0.8071748878923767
Random Forest	0.8295964125560538

It's noticed that the Random Forest Classifier model performance pretty well when comparing to the others. Consequently, it will be our main classifier model.

III.IV Refinement

Comparing the four classifiers, the one with best accuracy was Random Forest, with 82.95% of accuracy. Since it is the best model, let's try to improve its hyperparameters with Grid Search Algorithm.

Sklearn provides GridSearchCV class. This algorithm consists in create a "grid" with all possibles hyperparameters passed (many to many) and get the model with each possibilities. The best model will be the one we are going to use to deploy in Flask Endpoint lately.

```
>>> from sklearn.model_selection import GridSearchCV

>>> steps = [('scaller', MinMaxScaler()),
             ('classifier', RandomForestClassifier())]

>>> param_grid = {
    'classifier__n_estimators': [100, 300, 500, 700, 1000],
    'classifier__max_depth': [None, 1, 2, 3],
    'classifier__criterion': ['gini', 'entropy']
    }

>>> pipeline = Pipeline(steps)
>>> search = GridSearchCV(pipeline, param_grid, n_jobs=-1, scoring='recall'
)
>>> search.fit(x_train, y_train)
```

After training all possibles models and see the best param, we realize that the best model was the almost the same model with default param we trained previously. Consequently, we got the same accuracy score of the last Random Forest Classifier: **82.95%**.

Comparing the Random Forest Model with the SVM Benchmark model, we see that we got a better result. The new model decrease the number of False Positives and increase the number of True Positives.

	Predict 0	Predict 1
Actual 0	119	22
Actual 1	17	65

IV. Deployment

To simulate AWS Deploy Endpoint, as it was saw during the Nanodegree Program, I developed an endpoint using Python Flask and Python Flask-Restful APIs. It's a simple web application which contains only one endpoint (/model) and only one HTTP method (POST).

In AWS endpoint we only need to use two method to deploy the model and make predictions:

```
>>> predictor = estimator.deploy(...)   # to deploy
>>> predictor.predict(...)   # to make predictions
```

But, with this custom endpoint, we have to create almost the full HTTP request body from scratch:

```
>>> data='{"Pclass": 3, "Sex":1, "Age": 22, "Fare": 7.2500, "Embarked": 0,
 "Relatives": 1,"Alone": 0}'
>>> curl http://localhost:5000/model --header "Content-Type: application/js
on" --request POST --data "${data}"

<<< {"Survived": 0}
```

The idea here is not to deploy the model into WEB that every one could make prediction, but to show another different way to make prediction via HTTP requests.

V. Results

V.I Model Evaluation and Validation

First, we develop a SVM classifier to predict if a passenger in Titanic disaster would survived or not. This model was our benchmark. After compraring the SVM classifier to other tree classifiers, we found that Random Forest Classifier was the best model even though we used the default paramenters. After Grid Search algorithm, we found that the best params was:

paramter	orignal	best
criterion	gini	entropy
max_depth	None	None
n_estimators	100	500

V.II Justification

With this change of classifier, the accuracy score increase from **79.82%** to **82.95%**. Comparing the Confusion Matrix, we also see an improvement.

SVM	Predict 0	Predict 1
Actual 0	128	37
Actual 1	8	50

R. Forest	Predict 0	Predict 1
Actual 0	119	22
Actual 1	17	65

Even though the number of the True Positives decreased in **7%**, with the Random Forest Classifier, we increase the number of True Negatives in **30%** which indicates that the model is classifying better if a passenger really survived. Other information is great to take a look is the False Positives values: Random Forest Classifier decrease in **68.18%**, which means that we are making less mistakes guessing that a passenger survived when he actually died.