I2A2 – Institut d'Intelligence Artificielle Appliquée
Artificial Intelligence Training Program
Machine Learning from disaster (Titanic)

DIEGO OLIVEIRA DE ARAÚJO

BRAZIL
2022

ABSTRACT

In an increasingly computerized world, data is becoming an extremely important component for organizations, regardless of size, sector or type. In fact, they are one of the greatest business assets, with products, services, intellectual property and people. However, more important than possessors is to know how to enlist them for the sake of the business. Therefore, having an assertive Data Analytics strategy is essential.

Machine is a data analysis method that automates the construction of analytical models. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. The iterative aspect of machine learning is important, when models are exposed to new data, they are able to adapt independently. They learn from computations to produce reliable, repeatable decisions and results. This is a science that is gaining new momentum.

Summary

1. Objective

Through the absorbed learning during the classes of the teachers (Celso Azevedo and Marcelo Piovan) to develop an algorithm at the Jupyter Notebook to preparing and exploratory data analysis using the Titanic disaster dataset.

2. Tools and libraries used

- Anaconda 4.9.2;
- Jupyter Notebook 6.1.4;
- Matplotlib 3.5.0;
- Numpy 1.12.1;
- Pandas 1.3.5;
- Python 3.8.5;
- Seaborn 0.9;
- Scikit-learn 1.0.1;

3. Files

- I2A2 – diego_oliveira_report.pdf;
- I2A2 – Challenge2.ipynb;
- requirements.txt;
- Train.xlsx;
- Test.xlsx;

All files can be acquired at the repository:
https://github.com/Diegeux/I2A2_diego_oliveira_TITANIC.git

4. Development

A virtual environment was set, the libraries were installed and used correctly, then a requirements file was created to download the development resources in other machines. However, to download it, it is necessary run the code: **python -m pip install -r requirements.txt.**

The following libraries were used:

Numpy: is the fundamental package for scientific computing in Python, it provides a multidimensional array object, various derived objects and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, basic linear algebra, basic statistical operations and much more

Pandas: is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

Matplolib: is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.

Seaborn: is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Scikit-learn: is a library in Python that provides many unsupervised and supervised learning algorithms. The functionality that it provides are: regression, classification, model selection and preprocessing.

Importing libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sb
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import cross_val_score
```

The received files were in .xlsx extension (Train.xlsx, Test.xlsx), they were loaded and converted to .csv extension to a better development of the algorithm.

Loading and converting the datasets (train and test)

Loading the data

```
In [2]: train_data = pd.read_excel("C:\\Users\\dell\\Desktop\\Treinamentos e Cursos\\I2A2\\I2A2 - Atividade 2\\Train.xlsx")
        test_data = pd.read_excel("C:\\Users\\dell\\Desktop\\Treinamentos e Cursos\\I2A2\\I2A2 - Atividade 2\\Test.xlsx")
```

Converting xlsx to csv

```
In [3]: train_data.to_csv('Train.csv', index=False)
        test_data.to_csv('Test.csv', index=False)

        train_data = pd.read_csv('Train.csv')
        test_data = pd.read_csv('Test.csv')
```

Train_data has 16 variables and 891 lines

```
In [6]: #Printing the quantity of variables and inputs
        print("The shape of the dataset is: {} variables/columns and {} inputs/lines".format(train_data.shape[1], train_data.shape[0]))

        The shape of the dataset is: 16 variables/columns and 891 inputs/lines
```

The dataset has different types of variables. They are: int64, object and float64.

Variable types

```
In [7]: display(train_data.dtypes)

        Kaggle_Pass_Id      int64
        pclass              int64
        survived            int64
        name               object
        sex                object
        age               float64
        sibsp               int64
        parch               int64
        ticket             object
        fare              float64
        cabin              object
        embarked           object
        boat               object
        body              float64
        home.dest          object
        orig_seq            int64
        dtype: object
```

The greatest missing data is the body variable, greater than 90%, and the least missing data is the embarked variable with 2%.

```
In [9]: #printing how much missing data is (percentual)
        null_percentual = (train_data.isnull().sum()/train_data.shape[0]).sort_values(ascending=False)
        null_percentual

        #A informação sobre a Cabin é a que possui o maior número de informações faltantes, com mais de 77%. Após, a coluna Age não possu

Out[9]: body             0.906846
        cabin            0.771044
        boat             0.629630
        home.dest        0.439955
        age              0.198653
        embarked         0.002245
        Kaggle_Pass_Id   0.000000
        pclass           0.000000
        survived         0.000000
        name             0.000000
        sex              0.000000
        sibsp            0.000000
        parch            0.000000
        ticket           0.000000
        fare             0.000000
        orig_seq         0.000000
        dtype: float64
```

Statistical distribution with count, mean, standard deviation, minimum, maximum and quartiles.

Statistical distribution

```
In [10]: train_data.describe()
```

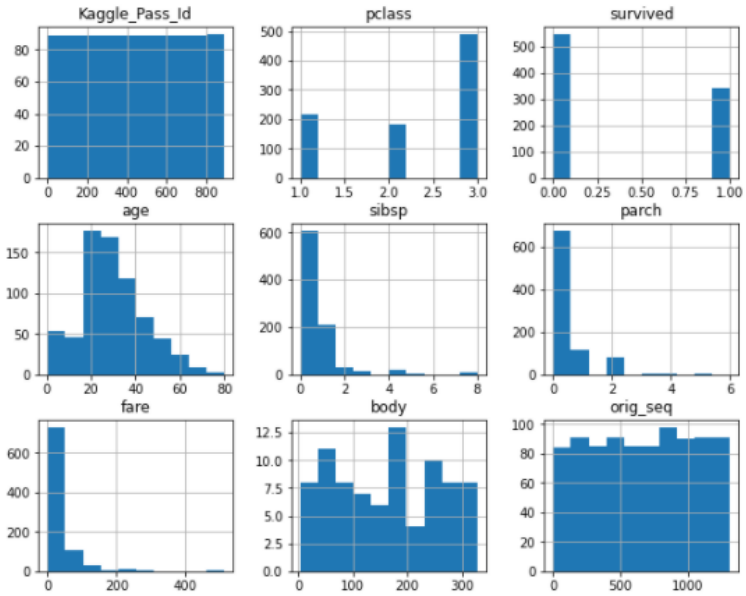
Out[10]:

	Kaggle_Pass_Id	pclass	survived	age	sibsp	parch	fare	body	orig_seq
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	83.000000	891.000000
mean	446.000000	2.308642	0.383838	29.699113	0.523008	0.381594	32.204208	162.843373	662.304153
std	257.353842	0.836071	0.486592	14.526507	1.102743	0.806057	49.693429	96.945356	378.282967
min	1.000000	1.000000	0.000000	0.416700	0.000000	0.000000	0.000000	4.000000	1.000000
25%	223.500000	2.000000	0.000000	20.125000	0.000000	0.000000	7.910400	73.500000	335.500000
50%	446.000000	3.000000	0.000000	28.000000	0.000000	0.000000	14.454200	169.000000	666.000000
75%	668.500000	3.000000	1.000000	38.000000	1.000000	0.000000	31.000000	252.000000	993.500000
max	891.000000	3.000000	1.000000	80.000000	8.000000	6.000000	512.329200	328.000000	1309.000000

Exploratory Data Analysis refers to the critical process of performing initial investigation on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

We can see the variables histogram, and we can realize that there were more dead than survived, we can also realize that there were more people between 20 and 30 years old.

```
In [11]: #Variables histogram
         train_data.hist(figsize=(10,8));
```



Between men and women, women are more likely to survive.

In [12]:
```
#Survival probability
train_data[['sex','survived']].groupby(['sex']).mean()
```

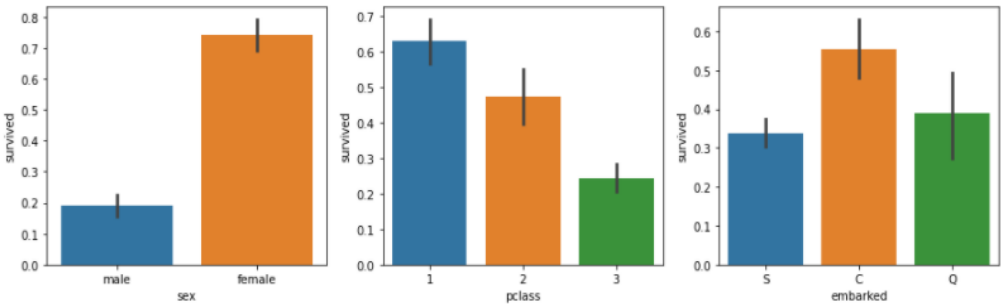
Out[12]:

sex	survived
female	0.742038
male	0.188908

We can analyze the quantity of people for sex, social class and embark place.

In [13]:
```
#Plotting barplot Survived x Sex, Pclass and  Embarked
fig, (axis1, axis2, axis3) = plt.subplots(1,3, figsize=(15,4))
sb.barplot(x='sex', y='survived', data=train_data, ax=axis1)
sb.barplot(x='pclass', y='survived', data=train_data, ax=axis2)
sb.barplot(x='embarked', y='survived', data=train_data, ax=axis3)
```

Out[13]: <AxesSubplot:xlabel='embarked', ylabel='survived'>

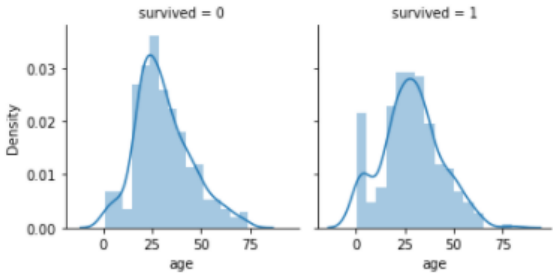


It is possible to see the influence age variable on probability of survival

In [14]:
```
#Influence of age on probability of survival
age_survived = sb.FacetGrid(train_data, col='survived')
age_survived.map(sb.distplot, 'age')
```

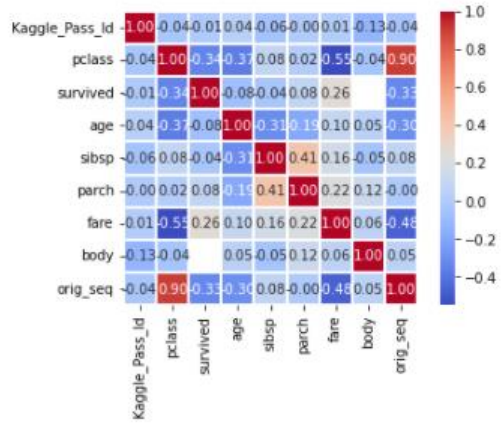
```
C:\Users\dell\anaconda3\envs\I2A2_diego_oliveira\lib\site-packag
a deprecated function and will be removed in a future version. P
unction with similar flexibility) or `histplot` (an axes-level f
  warnings.warn(msg, FutureWarning)
C:\Users\dell\anaconda3\envs\I2A2_diego_oliveira\lib\site-packag
a deprecated function and will be removed in a future version. P
unction with similar flexibility) or `histplot` (an axes-level f
  warnings.warn(msg, FutureWarning)
```

Out[14]: <seaborn.axisgrid.FacetGrid at 0x1d0c6e7a520>



Linear correlation between the variables. When the value is negative, the variables have opposite influences, while one grows the other falls. When the value is positive, the both grow.

```
In [16]: sb.heatmap(train_data.corr(), cmap='coolwarm', fmt='.2f', linewidths=0.1,
                    vmax=1.0, square=True, linecolor='white', annot=True);
```



It is also possible to analyze descriptive statistical for non-numerical data. For sex variable is just possible to have 2 values (male and female), male has more registers (almost 65%). The place where has more embarked people is in S – Southampton.

```
In [17]: #Descriptive statiscal for non-numerical data
         train_data.describe(include=['O'])
```

Out[17]:

	name	sex	ticket	cabin	embarked	boat	home.dest
count	891	891	891	204	889	330	499
unique	891	2	681	147	3	24	291
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S	15	New York, NY
freq	1	577	7	4	644	27	43

Train and test datasets were stored for a future use, the target was created, then the survived variable was dropped of the train dataset. The both of them were merged.

```
In [18]: train_x = train_data.shape[0]
         test_x = test_data.shape[0]

In [19]: target = train_data['survived']
         train_data.drop(['survived'], axis=1, inplace=True)

In [20]: merged = pd.concat(objs=[train_data,test_data], axis=0).reset_index(drop=True)

In [21]: print(merged.shape[0])
         print(merged.shape[1])

         1309
         15
```

The shape of new dataset.

```
In [25]: print("The shape of the new dataset is: {} variables/columns and {} inputs/lines".format(merged.shape[1],merged.shape[0]))

         The shape of the new dataset is: 15 variables/columns and 1309 inputs/lines
```

Unnecessary variables were dropped.

```
In [26]: #Dropping unnecessary variables
         merged.drop(['Kaggle_Pass_Id', 'name', 'ticket', 'cabin', 'boat', 'body', 'home.dest', 'orig_seq'], axis=1, inplace=True)
```

The missing data were filled with median for age and fare variables and mode for embarked variables. Then, we can see that there is no missing data.

```
In [28]: (merged.isnull().sum()/merged.shape[0]).sort_values(ascending=True)

Out[28]: pclass      0.000000
         sex         0.000000
         sibsp       0.000000
         parch       0.000000
         fare        0.000764
         embarked    0.001528
         age         0.200917
         dtype: float64

In [29]: #age - median
         age_median = merged['age'].median()
         merged['age'].fillna(age_median, inplace=True)

         #fare - median
         fare_median = merged['fare'].median()
         merged['fare'].fillna(fare_median, inplace=True)

         #embarked - mode
         embarked_top = merged['embarked'].value_counts()[0]
         merged['embarked'].fillna(embarked_top, inplace=True)

In [31]: #printing age and fare median, and embarked mode
         print("{}, {}, {}".format(age_median, fare_median, embarked_top))

         28.0, 14.4542, 914

In [32]: merged.isnull().sum()

Out[32]: pclass      0
         sex         0
         age         0
         sibsp       0
         parch       0
         fare        0
         embarked    0
         dtype: int64
```

The variables were prepared for the model, the sex variable was converted for 0 – male and 1 – female. And the values C, Q, S of the embarked variable were done dummy.

```
In [33]: #Converting Sex in 0 and 1
         merged['sex'] = merged['sex'].map({'male':0, 'female':1})

In [34]: #Dummies variables for "Embarked"
         embarked_dummies = pd.get_dummies(merged['embarked'], prefix='embarked')
         merged = pd.concat([merged, embarked_dummies], axis=1)
         merged.drop('embarked', axis=1, inplace=True)

In [35]: merged
```

Out[35]:

	pclass	sex	age	sibsp	parch	fare	embarked_914	embarked_C	embarked_Q	embarked_S
0	3	0	22.0	1	0	7.2500	0	0	0	1
1	1	1	38.0	1	0	71.2833	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	0	1
4	3	0	35.0	0	0	8.0500	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
1304	3	0	28.0	0	0	8.0500	0	0	0	1
1305	1	1	39.0	0	0	108.9000	0	1	0	0
1306	3	0	38.5	0	0	7.2500	0	0	0	1
1307	3	0	28.0	0	0	8.0500	0	0	0	1
1308	3	0	28.0	1	1	22.3583	0	1	0	0

1309 rows × 10 columns

The train and test dataset were recovered and put in a same file to improve the development.

```
In [36]: train = merged.iloc[:train_x]
         test = merged.iloc[train_x:]
```

```
In [37]: train
```

Out[37]:

	pclass	sex	age	sibsp	parch	fare	embarked_914	embarked_C	embarked_Q	embarked_S
0	3	0	22.0	1	0	7.2500	0	0	0	1
1	1	1	38.0	1	0	71.2833	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	0	1
4	3	0	35.0	0	0	8.0500	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
886	2	0	27.0	0	0	13.0000	0	0	0	1
887	1	1	19.0	0	0	30.0000	0	0	0	1
888	3	1	28.0	1	2	23.4500	0	0	0	1
889	1	0	26.0	0	0	30.0000	0	1	0	0
890	3	0	32.0	0	0	7.7500	0	0	1	0

891 rows × 10 columns

```
In [38]: test
```

Out[38]:

	pclass	sex	age	sibsp	parch	fare	embarked_914	embarked_C	embarked_Q	embarked_S
891	3	0	34.5	0	0	7.8292	0	0	1	0
892	3	1	47.0	1	0	7.0000	0	0	0	1
893	2	0	62.0	0	0	9.6875	0	0	1	0
894	3	0	27.0	0	0	8.6625	0	0	0	1
895	3	1	22.0	1	1	12.2875	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...
1304	3	0	28.0	0	0	8.0500	0	0	0	1
1305	1	1	39.0	0	0	108.9000	0	1	0	0
1306	3	0	38.5	0	0	7.2500	0	0	0	1
1307	3	0	28.0	0	0	8.0500	0	0	0	1
1308	3	0	28.0	1	1	22.3583	0	1	0	0

418 rows × 10 columns

```
In [39]: target
```

```
Out[39]: 0      0
         1      1
         2      1
         3      1
         4      0
               ..
         886    0
         887    1
         888    0
         889    1
         890    0
         Name: survived, Length: 891, dtype: int64
```

Two Machine Learning models were used to predict – Logistic Regression and Decision Tree. The last one presented the best accuracy.

```
In [40]: #Creating a Logistic Regression Model
         lr_model = LogisticRegression(solver='liblinear')
         lr_model.fit(train, target)

         #Checking model accuracy
         acc_lr_model = round(lr_model.score(train, target) * 100, 2)
         print("The Logistic Regression model accuracy is: {}".format(acc_lr_model))
```

```
The Logistic Regression model accuracy is: 80.13
```

```
In [41]: ##Creating a Decision Tree Model
         tree_model = DecisionTreeClassifier(max_depth=3)
         tree_model.fit(train, target)

         #Checking model accuracy
         acc_tree_model = round(tree_model.score(train, target) * 100, 2)
         print("The Decision Tree model accuracy is: {}".format(acc_tree_model))
```

```
The Decision Tree model accuracy is: 82.72
```

```
In [43]: #Comparative between Logistic Regression model accuracy and Decision Tree model accuracy
         models = [[acc_lr_model,acc_tree_model]]
         new = pd.DataFrame(models, columns=['Logistic Regression Model','Tree Decision Model'])
         new.rename(columns={'Logistic Regression Model':'Logistic Regression Model','Tree Decision Model':'Tree Decision Model'}, inplace
         new.index = ['Accuracy']
         new
```

Out[43]:

	Logistic Regression Model	Tree Decision Model
Accuracy	80.13	82.72

It was also done cross validation of the models to check out the generalization. The best model is still Decision Tree

In [44]:
```
#Models cross validation
scores_lr_model = cross_val_score(lr_model,train.values,target,cv=5)
scores_tree_model = cross_val_score(tree_model,train.values,target,cv=5)
cross_lr = np.mean(scores_lr_model)
cross_tree = np.mean(scores_tree_model)
print(cross_lr)
print(cross_tree)
```

```
0.7878852551628899
0.8091959073504487
```

In [45]:
```
#Comparative between Logistic Regression cross validation and Decision Tree cross validation
cross_compare = [[cross_lr,cross_tree]]
df_cross_compare = pd.DataFrame(cross_compare,columns=["1","2"])
df_cross_compare.rename(columns={"1":"Logistic Regression","2":"Decision Tree"}, inplace=True)
df_cross_compare.index = ["Cross Validation"]
df_cross_compare
```

Out[45]:

	Logistic Regression	Decision Tree
Cross Validation	0.787885	0.809196

Two people's data were tested on the Decision Tree prediction model.

In [46]:
```
#Declaring variables values for different people
person_1 = np.array([3, 0, 29, 3, 3, 22.25, 0, 0, 1, 0]).reshape((1, -1))
person_2 = np.array([2, 1, 30, 1, 1, 32.2, 0, 0, 0, 1]).reshape((1, -1))

if tree_model.predict(person_1) == 0:
    result_person_1 = "Died"
else:
    result_person_1 = "Survived"
if tree_model.predict(person_2) == 0:
    result_person_2 = "Died"
else:
    result_person_2 = "Survived"

#Checking out
print("Person_1 will be:\t{}".format(result_person_1))
print("Person_2 will be:\t{}".format(result_person_2))
```

```
Person_1 will be:       Died
Person_2 will be:       Survived
```

5. Conclusion

It is important to note that the lifecycle of a data science project is quite simple to understand, it is necessary to define the problem, obtain, explore and prepare the data, so that later we can model and evaluate the data. The time to define the problem is necessary to take the time to really understand what to do and how to do it, but the most important step is the exploration of the data (EDA), many questions and hypotheses can be answered at this stage. The data science project is not static, it is very iterative, so we will be able to go back and forth in many stages of development.

Bibliography

https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15

https://santodigital.com.br/afinal-qual-e-a-importancia-dos-dados-na-estrategia-das-empresas/

https://www.sas.com/pt_br/insights/analytics/machine-learning.html

https://academic.oup.com/mnras/article/449/2/1275/1074732?login=true

https://www.codecademy.com/article/scikit-learn

https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/

https://seaborn.pydata.org/

https://numpy.org/doc/stable/user/whatisnumpy.html

https://pandas.pydata.org/