Apprendimento Statistico

Nenna Giulio

Data Analysis applied to a Marketing Dataset

1 Introduction

The main goal of this report is to apply different machine learning tecniques to a dataset that can be found on the UCI Machine Learning Repository. The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. Each entry represents a phone call to a client of whom several features are provided.

The responce variable is binary (yes or no) and indicates whether a phone call successfully convinced the client to purchase a financial product the bank is tryng to sell (bank term deposit). Among the 20 provided features, 11 are categorical and the remaining are numerical. An extensive description of each feature will now follow.

1.1 Attribute information

Table 1 provides an extensive overview about each attribute featured in the dataset.

Important notes about some of the attributes are the following:

- Attributes from `Contact` to `duration` are related to the last contact of the current campaign
- The attribute `duration` affers to the call duration and it's not known until the call is completed. It highly affects the prediction (e.g. if `duration=0` then the outcome will always be *not successful*), hence it will be deleted since the model is ment to be deployed with real world data.

Name	Description	Type	Possible values
Age	Age of each client	numeric	-
Job	Type of job	categorical	Admn, blue-collar, entrepreneur...
Marital	Marital Status	categorical	Divorced, married, single, unknown
Education	level of education	categorical	basic.4y, basic.6y, basic.9y, high-school,...
Default	whether a client has credit in default	Categorical	Yes, No, Unknown
Housing	whether a client has an housing loan	Categorical	Yes, No, Unknown
Loan	whether a client has a personal loan	Categorical	Yes, No, Unknown
Contact	contact communication type	Categorical	cellular, telephone
Month	last contact month of the yeah	Categorical	jan, feb, mar...
Day of the week	last contact day of the week	Categorical	mon, tue, ...
Duration	Last contact duration in seconds (see notes)	Numeric	-
Campaign	number of contacts performed during this campaign for that client	Numeric	-
pdays	number of days that passed by after the client was last contacted from a previous campaign	numeric	-
previous	number of contacts performed before this campaign for this client	numeric	-
poutcome	outcome of the previous marketing campaign	Categorical	Failure, nonexistent, success
emp.var.rate	employment variation rate - quarterly indicator	Numeric	-
cons.price.idx	consumer price index - monthly indicator	Numeric	-
cons.conf.idx	consumer conficence index - monthly indicator	Numeric	-
euribor3m	euribor 3 month rate - daily indicator	Numeric	-
nr.employed	Number of employees - quarterly indicator	Numeric	-
Responce variable: y	Has the client subscribed to a term deposit	Binary	Yes, No

Table 1: Extensive description for each attribute in the dataframe

2 Data Exploration and Preprocessing

We first import the dataset and encode each categorical feature using *one-hot encoding*. This means that numerical features will be left unchanged while for each categorical feature the process is the following:

1. Determine all the distinct values of that feature (categories)

2. For each category generate a new binary column

3. Assign values to the binary columns according to categories featred in each line.

feature	category 1	category 2	category 3
category 1	1	0	0
category 2	0	1	0
category 3	0	0	1
category 2	0	1	0

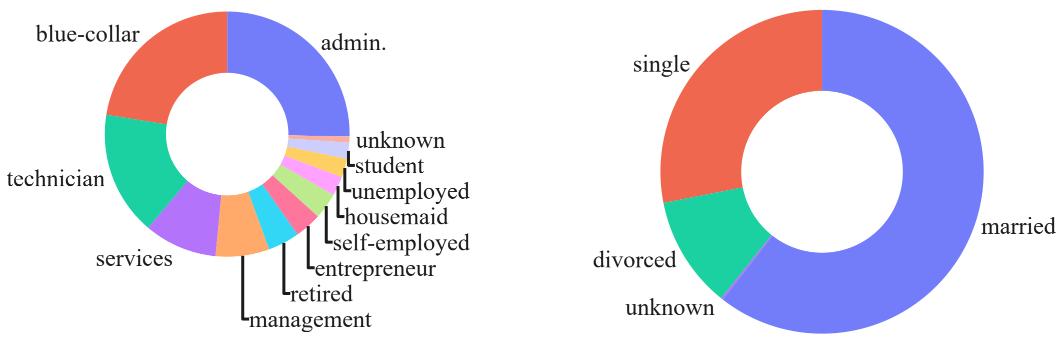
```
import pandas as pd

df = pd.read_csv('Data/bank/bank-additional-full.csv', sep=';')
display(df.head())
cat_col = df.dtypes=='O'
df_enc = pd.get_dummies(df.loc[:, cat_col], prefix=df.columns[cat_col])
df_enc = df_enc.join(df.loc[:, np.logical_not(cat_col)])

df_enc = df_enc.drop('y_no', axis=1) #deleting column since attribute "y" is binary
df_enc = df_enc.drop('duration', axis = 1) #drop the duration column (see attribute information)
```

Listing 1: Data encoding

2.1 Data Exploration

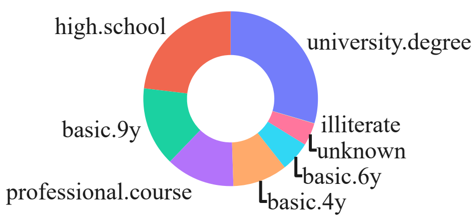
In order to gain useful insights about some of the attributes featured in the dataset we can now perfom some *data exploration*. This step of the analysis is qualitative by nature and consists in plotting graphs and distributions relative to each feature in the dataset.

There are 11 categorical features and the remaining are numerical. Categorical data will be explored by means of *pie charts* while numerical data will be explored by means of a *scatterplot matrix*

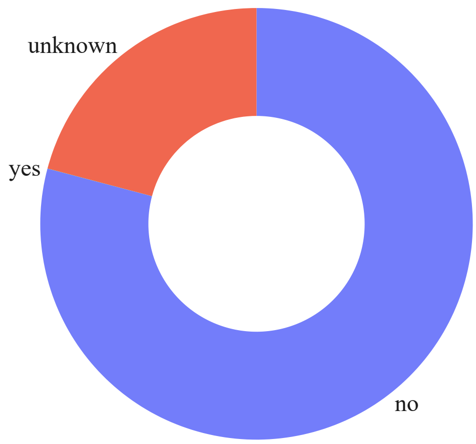


(a) Pieplot relative to `job`

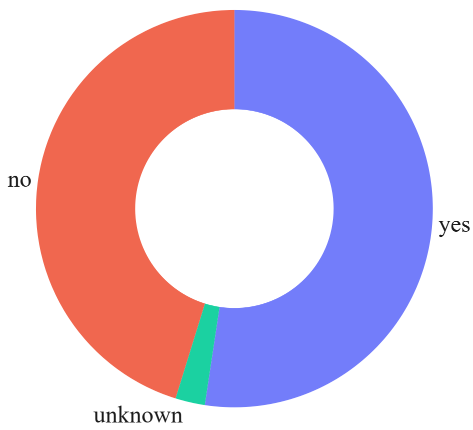
(b) Pieplot relative to `marital`



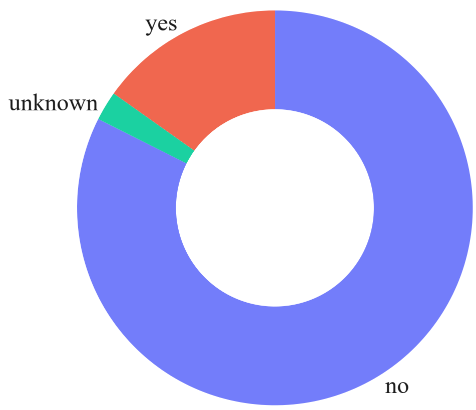
(c) Pieplot relative to `education`



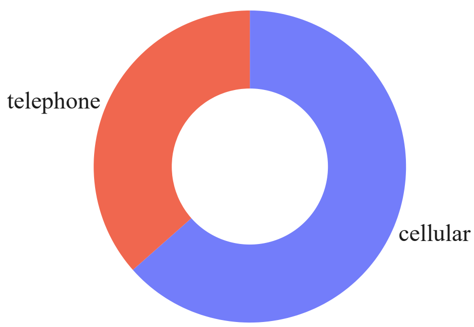
(d) Pieplot relative to `default`



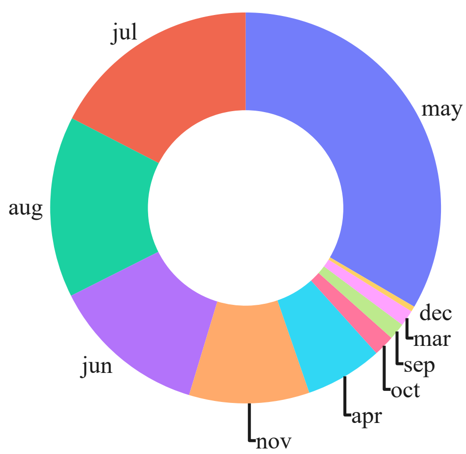
(e) Pieplot relative to `housing`



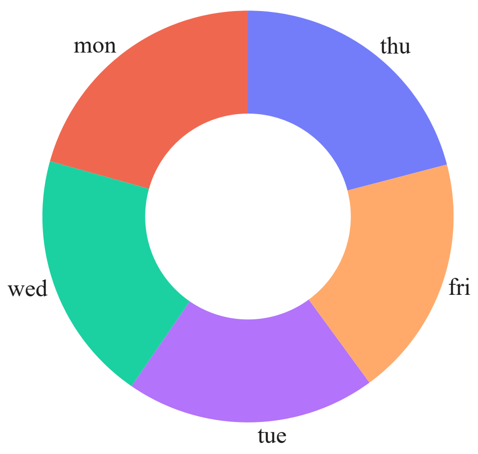
(f) Pieplot relative to `loan`



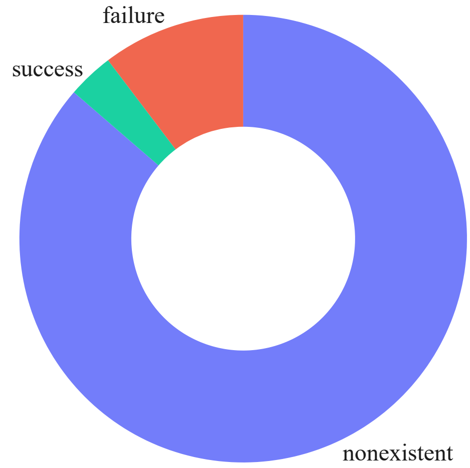
(g) Pieplot relative to `contact`



(h) Pieplot relative to `month`



(i) Pieplot relative to `day_of_the_week`



(j) Pieplot relative to `poutcome`

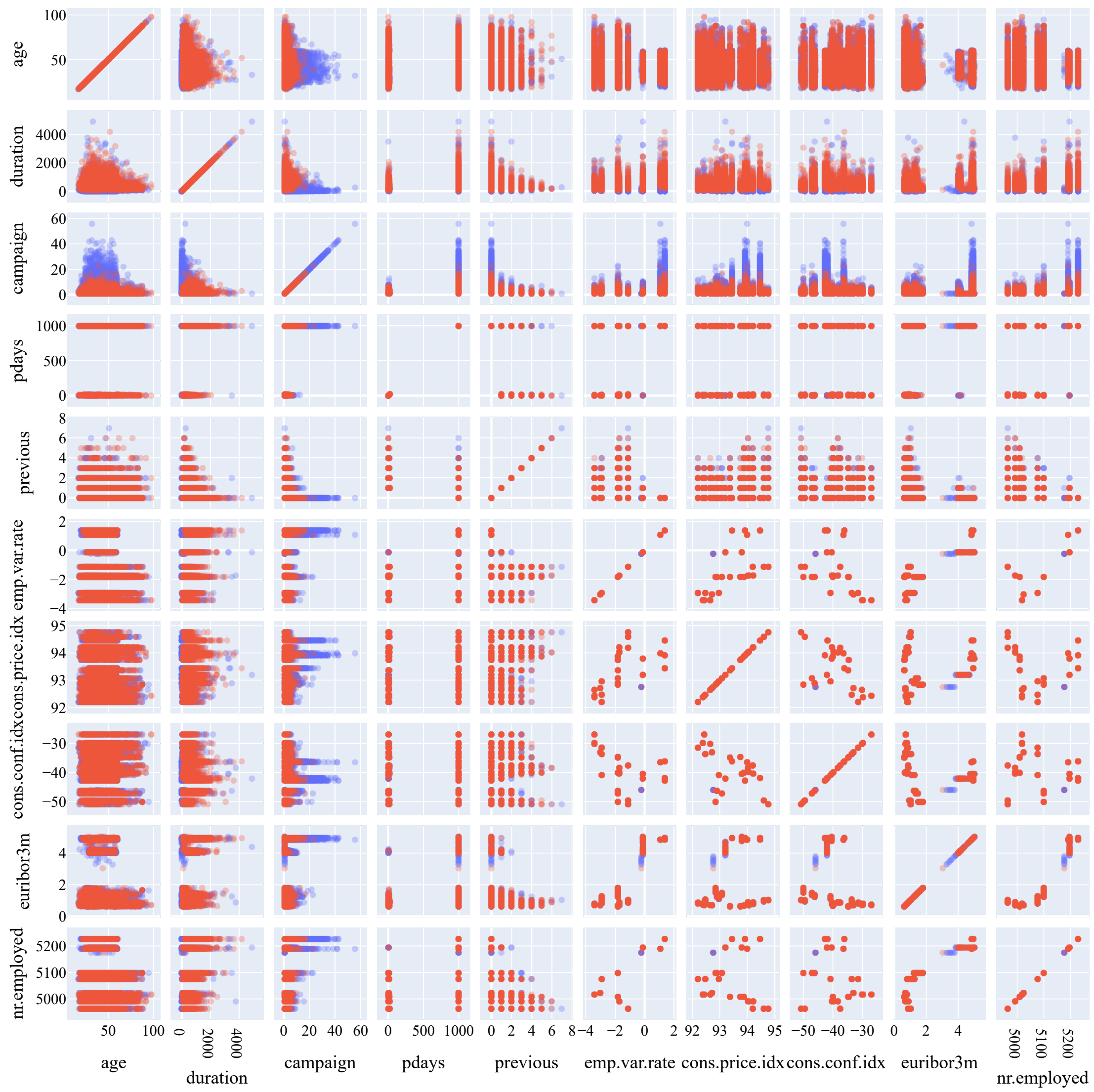


Figure 1: Scatterplot matrix of numerical features

From the scatterplot we don't see any particular correlation between numerical features. Yet, as for the coloring of the data, one can notice that the attribute campaign discriminates pretty well the outcome. As a matter of fact people that recieved a lot of calls from the call center never buy the product that the bank is trying to sell. This information alone could be of some value for reducing cost and better targeting calls.

2.2 Dataset splitting

After exploring data, the first thing to do is to perform a split of our dataset into training and test set.

```
    df_train, df_test = train_test_split(df_enc, test_size=0.1, random_state=42)
```

Listing 2: Data splitting

As it is the case with some binary classification datasets, our data is heavly biased towards one class of outcome. In our case the most common outcome for each call is, reasonably, unsuccesful call: meaning that the client that has been called have **not** purchased the product that the bank is trying to sell. In order to quantify the class unbalance we can use a simple pie chart:

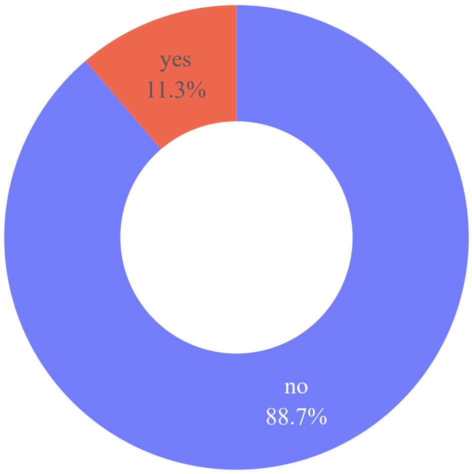
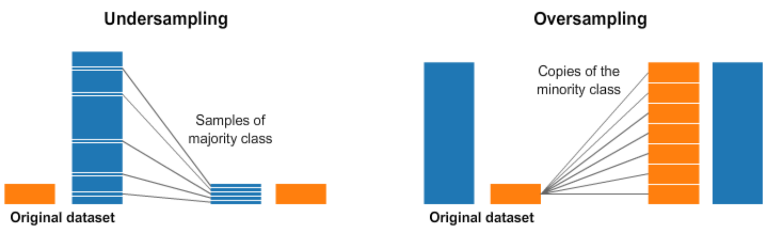


Figure 2: Responce variable class unbalance

As we can see $\sim 90\%$ of our data is biased towards the majority class (no). If we tried to feed the data as it is to any classification model the resulting classifier would seem to be "accurate" even if it really isn't.

As a simple example we could think of a classifier that classifies every data entry as "no" no matter of the features. This naive classifier, tested with our heavily biased data, would result to have $\sim 90\%$ accuracy, the same percentage of the majority class with respect to the whole dataset.

Various techniques can be used in order to solve this problem, among them there are **Oversampling** and **Undersampling**.



Visual exapmple of Oversampling and Undersampling

As the name suggests, oversampling takes place when the minority class is used to generate new bootstrapped data to balance classess. On the other hand undersampling consists in randomly sampling from the majority class enough samples such that the two classes are balanced. In our case, since we have enough data, we can opt for undersampling in order to balance classes.

```
response = df_train['y_yes']
feature_matrix = df_train.loc[:, df_train.columns!='y_yes']

#rebalancing classes (y_yes=1 is the minority class)--------
ratio = df_train['y_yes'].sum()/df_train['y_yes'].size
print("Original training dataset has", df_train.index.size, "Observations. \n", "
    Among them", response.sum(), "are positive and", response.size-response.sum(),
    "are negative \n Performing dataset rebalancing by undersampling... \n\n" )
df_train_false = df_train.loc[response==0, :]
df_train_false_resampled = df_train_false.sample(frac=ratio)

df_train_true = df_train.loc[response==1,:]

df_train = pd.concat([df_train_true, df_train_false_resampled])
ratio = df_train['y_yes'].sum()/df_train['y_yes'].size

response = df_train['y_yes']
feature_matrix = df_train.loc[:, df_enc.columns!='y_yes']

print("Training Dataset rebalancing performed. Dataset has now", df_train.index.
    size, "Observations. \n", "Among them", response.sum(), "are positive and",
    response.size-response.sum(), "are negative \n\n" )

X_train = df_train.loc[:, df_train.columns!='y_yes']
X_test = df_test.loc[:, df_test.columns!='y_yes']
y_train = df_train['y_yes']
y_test = df_test['y_yes']
```

Listing 3: Data rebalancing

2.3 Scaling the dataset

The encoded dataset is characterized by features with very different scale/range of values. In this case, if we apply PCA or any kind of training model on this data, some important features can be "covered" by other feature just for the unit measure disparity. This is why is a good practice to **rescale** the data. The most used scalers used are the **Min-max Scaler** and **Standard Scaler**

The **Min-Max Scaler** trasnform the data such that the values of each column are distributed between a minimum value m and a maximum value M. Let $X \in \mathbb{R}^{n \times p}$ be our feature matrix, with $X = [x_1, x_2, \dots, x_p]$. Let $X' \in \mathbb{R}$ be our scaled matrix with $X' = [x'_1, x'_2, \dots, x'_p]$. If $M_i = \max\{x_i\}$, $m_i = \min\{x_i\}$, $i \in \{1, \dots, p\}$ then

$$x'_i = \frac{x_i - m_i}{M_i - m_i} \quad i \in \{1, \dots, p\} \tag{2.1}$$

where all operations are intended to be element-wise.

On the other hand the **Standard-Scaler** makes the values of each feature in the data to have zero-mean and unit-variance. In particular:

$$x'_i = \frac{x_i - \bar{x_i}}{s_i} \quad i \in \{1, \dots, p\} \tag{2.2}$$

where $\bar{x_i}$ is the *sample mean* and s_i is the *sample variance* of the i-th column

```
#Performing scaling using min-max scaler
scaler_mm = MinMaxScaler()
scaler_mm.fit(X_train)

X_train_scaled_mm = scaler_mm.transform(X_train)
X_test_scaled_mm = scaler_mm.transform(X_test)

#Performing scaling using standard scaler
scaler_std = StandardScaler()
scaler_std.fit(X_train)

X_train_scaled_std = scaler_std.transform(X_train)
X_test_scaled_std = scaler_std.transform(X_test)
```

Listing 4: Data scaling