AVRO Case

Andrea Bragantini andrea.aldo.bragantini@gmail.com

The following paper is intended to be followed along with the attached *AvroCaseBragantini.pptx* file for the solution of the given exercise. References made in this text to figures or tables have to be found in the .pptx file.

The work is carried out using **Python** environment, in the Spyder IDE. The choice of the software came from the desire to explore new machine learning libraries in Python to test its capabilities. I used to perform such analysis on the R platform, a very complete statistical software with lots of suited libraries. However, in the last year I switched to Python programming language for its versatility in many different kinds of problems. In particular, the deep learning library PyTorch, that I have been recently using in my last working experience.

The following sub-sections summarizes the workflow and how I approached the task. Since the very beginning my idea was to use a **Supervised Learning technique** to build up a first simple model to tackle the regression problem. The easiest model I could think of was a **Multivariate Linear Regression**. Later, I used this model to benchmark further more powerful machine learning model such as the **Regression Trees**. The natural improvement of regression trees is the **Random Forest Regression** that I have ultimately used. This last ML model is very popular among practitioners and it is considered to be one of the state-of-art and best-practice in treating data-sets with lots of categorical variables for regression problems. Questions are answered in order.

1. Question 1

It is asked to develop a data science model with the data in the CSV file to predict the required time to resolve an alert, based on its characteristics.

1.1 Exploratory Analysis

The first thing to do is to explore the available data-set. It is important to understand the distributions of both numerical and categorical variables and how our data-set is composed. Please have a look at the presentation file for distribution plots and boxplots of the different variables that have been omitted here for space reasons. Bar charts, highlighting the numerosity of different levels have been used for categorical variables, while classic histograms have been used for numerical variables.

It is worth to note how many of the numerical variables are "count-like" type. This means that the representation of their distribution often looks like a simple bar chart[1]. The data-set is composed by the following most important features:

- Categorical Variables
  - **"Status"** : defines the current state of the alert.
    The large majority of alerts present in the data-set are closed. Closed and resolved alerts can be considered finished tasks and posses a valid resolution_date variable. Open, patchAvailable and reopened are instead outstanding tasks waited to be treated.
  - **"Priority"** : defines the priority assigned to the alert.
    Intuitively this variable should be very descriptive in defining the amount of time that it takes for an alert to be solved. It is easy to think how major, blocker and critical issues will be processed quicker than minor or trivial issues. Let's see whether data reflects this intuition later.
  - **"Issue_type"** : defines the type of alert.
    The data-set is mainly composed by bugs, improvements and new features.
  - **"Resolution"** : defines the closing cause for that alert.
    As already explained this features takes on many levels but it can probably be easily modelled as "fixed" alerts (the vast majority) and "others" reasons.
- Numerical Variables
  - **"Vote_count"** : tracks the number of votes for that alert in the website thread. The distribution is highly skewed to the right. There are a lot of alerts with 0 votes and very few alerts with more than 3 votes. Presence of outliers.
  - **"Comment_count"** : tracks the number of comments for that alert in the website thread. The distribution is also positevely skewed. There are lots of alerts with few comments and very few alerts with lots of comments. Presence of outliers.
  - **"Watch_count"** : tracks the number of watchers for that alert in the website thread. The distribution is highly skewed to the right and it is similar to *vote-count*. There are a lot of alerts with few watchers and very few alerts with more than 5-6 watchers. Presence of outliers.
  - **"Description_length"** : tracks the length of the alert description in the website thread. The distribution is also skewed to the right. There are some outliers with very long descriptions.
  - **"Summary_length"** : tracks the length of the alert summary in the website thread. This is the only distribution that looks different from the others and might look a little Gaussian. Nonetheless, a Shapiro test yields a very low p-value for which there is statistical evidence to reject the null hypothesis of normality. (Shapiro-Wilk statistics=0.965, p=0.000)

1. The reader would soon understand looking at the plots on the attached presentation file

- DateTime Variables
  - **"Created"** : reports the creation date for that alert.
  - **"Updated"** : reports the date of last update for that alert.
  - **"Resolutiondate"** : reports the closing date for that alert.

1.2 Data Pre-processing

1.2.1 Splitting Training and Validation sets

Looking at the available data with a rough consideration, both opened alerts, which still needs to be solved, and closed alerts are present. To calculate the required time to solve an alert, we must look only at successfully closed alerts in the data-set. In particular, when dealing with a supervised learning approach, the models has to be trained with labelled data and therefore the total completion time has to be available for each observation in our training set.

It is therefore required to cut the current data-set in two parts. The first one, used to train the model, will be called *training set*. The second one, containing all the current opened alerts, will be used as *validation set* to make predictions for the alert duration.

Here I found out two different options. The first one is to look only at alerts whose statuses are either *Closed* or *Resolved*. The other was to look only at alerts for which it was possible to determine a date of resolution. This latter approach yields more observations in our training set and therefore I decided to use this one.

1.2.2 Target Variable: duration

Sticking to the exercise request, it is evident from the data how the total processing time for an alert to be finally resolved is missing. We therefore need to derive the target variable from available data. I decided to call it **duration**. Looking at all the alert that have a resolution date, it is possible to calculate the duration of finished alerts by subtracting the resolution date to the creation date.

$$duration[\text{timedelta64}] = resolution\_date[\text{datetime64}] - created[\text{datetime64}] \qquad (1)$$

I exploited the Python data-type datetime64 and timedelta64 that can easily be converted to float64 at occurrence.

Please have a look at the pptx file for the distribution plots of the target variable. The distribution is highly skewed to the right with a strong presence of many outliers. Almost all the observations are characterized by smaller duration (order of magnitudes of minutes, hours, days). Few alerts are characterized by very long resolution times (order of magnitudes of years).

1.2.3 Useless Predictors

Many features of the current data-set does not offer any useful information for the following inference study. These are the variables that I removed a priori.

- **project** : same value for all observations
- **updated** : updates are posterior to resolutiondate
- **resolutiondate** : already used for determining duration
- **created** : already used for determining duration
- **key** : unique for all observations
- **days_in_current_status** : difficult to be included in the model[2]

1.2.4 Reporter VS Assignee

There are in particular two categorical variables which have a huge number of levels. This is absolutely painful to input in regression models therefore a solution must be found to include those predictors. Following indications given in the exercise text, some consideration can be made. On average, relevant reporters assign the issues themselves to be solved. Also minor reporters generally assign themselves the issue that they have raised. Mr Cutting is by far the biggest contributor. He is the reporter of a large fraction of current alerts and, at the same time, he solves his personally raised alerts and reported alerts from other smaller contributors[3].

With a very rough assumption we can say that all the reporters assign the issues to themselves. This creates a sort of superimposition between this 2 categorical variables. I therefore prefer to consider only reporters as categorical variable because it is always present, even in new alerts to be forecasted. While sometimes happens that issues are not assigned, therefore the assignee field is empty.

Nevertheless, other considerations about the assignee categorical variable can be made. Many of the unresolved alerts remain like so for years, without being assigned. This pushes towards the belief that unassigned alert are going to take likely more time to be solved than assigned ones. On the other hand, many reporters assign the issue themselves to be solved and normally this yields a quite efficient process such that the issue is quickly resolved. This other consideration instead pushes back in the other direction. It is therefore difficult to determine whether the status of assigned / not assigned can actually have an effect of the target variable. On top of that, the largest part of all resolved issues has indeed been assigned and it is not possible with available data to trace back the alert when it was unassigned to count the time. This information is somehow "masked" in the available resolved issues and could only be exploited in the case some information regarding the "history" of the alert is available.

Because of all these explained implications I decided to remove the variable *Assignee* and keep only *Reporters* to represent the contributors in the regression model.

1.2.5 Status VS Resolution

Sort of similar considerations can be made for these other 2 categorical variables. The concept at the baseline here is always that we have focused on a supervised learning method.

2. Some considerations will be made in question 3 on a possible use of this variable.
3. Please, have a look at the bar charts in the pptx presentation

All the features that the model will understand once called to predict new observations are those that only appears also in the training set.

Ideally, a new alert to be processed by the model would have a "status" variable that can assume the following: *Open*, *PatchAvailable*, *Reopened*, but never *Closed* or *Resolved*. These last 2 classes belongs in fact to an "already resolved" alerts. The current training set considers all alerts that have been closed/resolved for which it is possible to determine the duration. There is so no chance to train the model on actually useful levels belonging to observations in the test set[4]. For this reason the variable "status" should not be included in the model. In fact, it determines somehow the split in training/test sets.

At the same time, "resolution" levels have values only for closed/resolved alerts. Therefore, new alerts to be processed cannot have these features. In fact, looking at current available data-set it shows always *NaN*.

The same consideration as above can be made here. If only we were able to have access to the "history" of the subsequent statuses of a certain alert, we could observe how much an alert remained in a specific status and from that try to infer something on the duration of the new "open" observation that finds itself in that specific status. These data are however not available.

1.2.6 Logarithmic Transform

Looking at numerical data distributions it is clear how a transformation is needed. "Count-like" variables are typically distributed as a **Poisson** distribution. Normality tests show that they are particularly skewed and the transformation might help in reducing their skewness and kurtosis. I decided to perform the frequently used logarithmic transform. This should variable distributions to something closer to a Gaussian distribution. Please have a look at the pptx file for the distribution of transformed variables and a comprehensive pair-plot showing relationships between those transformed predictors. The following variables have been transformed:

- *Vote_count*
- *Comment_count*
- *Watch_count*
- *Description_length*
- *Duration*

As you can see also the target variable has been transformed. Therefore, we are in presence of a so called **log-log transformation**.

4. Such as *Open*, *PatchAvailable* and *Reopened*

Data pre-processing is so completed. It is worth noting that almost no missing or *NaN* values are present in the dataset. Only few description lengths were missing and therefore have been replaced with zeros.

1.3 Bivariate Analysis

Let us now focus on the relationships between our target variable "duration" and its numerical or categorical predictors. The aim is to find any pattern in predictors that can somehow explain variations in the target variables[5]. Once again, the reader is invited to check out available charts in the attached presentation.

As regards categorical variables, after dropping useless predictors, the following remain: *Priority*, *Issue_type*, *Reporter*. The number of levels[6] is anyway still pretty large[7]. The number of categorical variables levels affects the interpretability of our models and therefore is good practice to try to reduce those levels with some "combine methods". The bivariate analysis helps in finding new reduced classes. Here is what I have done: the analysis is structured as follows:

- Target Variable VS Categorical Predictors with full levels
  - **Issue_type** : It has been observed that issue types as *Wish* and *Subtask* have on average higher resolution times, while *Bug*, *Improvement*, *NewFeatures*, *Task*, *Test* has in general lower resolution times.
  - **Priority** : This is a very difficult predictor to analyze. Intuitively it seems very descriptive but unfortunately data show the contrary. In both transformed and original, variability of priority levels seems not to impact duration outcome relevantly. Just note presence of many outliers in *Major* class as it is very numerous class. Even looking at the transformed target variable does not produce any interesting insights in the effect of the different levels of this predictors[8].
  - **Reporters** : I did not report a bivariate analysis with respect of full levels of this predictor because they are too many. I decided to consider only reporters which has more than 10 reports. This reduces to an almost 20-levels categorical variable. The bivariate analysis shows how some of those reporters can actually be effective in explain duration variability as the duration takes higher values for some of them.
- Target Variable VS Categorical Predictors with reduced levels
  - **Issue_type** : After had reduced the levels to the 2 groups identified above, the analysis perfectly reflects the different behaviours on the target variables. The levels reduction has been successful.
  - **Priority** : Since I did not find any clear pattern that could explain the variation of the target variable I decided to keep all the 5 levels.

5. Here we are looking at our new processed training set
6. or classes
7. 5 levels for *Priority*, 7 levels for *Issue_type*, 245 levels for *Reporter*
8. One may not how *Trivial* issues leads roughly to shorter resolution times

  - **Reporters** : holds what said above.

- Target Variable VS Numerical Predictors

  - **Vote_count**, **Comment_count**, **Watch_count**: These 3 predictors behave very similarly with respect to the target variables. Bivariate analysis charts are much more explanatory when observed with the transformed target variable, so let's focus on those. It can be seen how a regression line can be fitted on data showing a , albeit weak, linear relationships between those predictors and the target variable. In general, the more the counts, the longer the resolution time for that alert.
  - **Description_length** : This predictor unfortunately just show an undefined cloud of points which does sound to me as it is not much linked to the target variable.

1.4 One Hot Encoding

Now that we have tried to reduce the levels of our categorical variables and looked for possible explanations of the variability of our target variable with respect to its predictors, we can proceed to include those categorical variables in the regression problem. Categorical variables need to be encoded with dummy binary variables. Given n levels of a categorical variable, one needs $n-1$ dummy variables to encode it properly. Following the class reduction as explained above this is how I encoded the variables

- **Issue_type** : 1 dummy variable for (*Short*,*Long*)
- **Priority**: 4 dummy variables representing the 5 levels
- **Reporter**: 18 dummy variables representing the 19 levels

1.5 Models Design

1.5.1 Multivariate Linear Regression

It is always good practice to try to fit a linear regression to have a benchmark model for further investigations. I decided to start with a multivariate linear regression with one-hot encoded data. However, usual assumptions to fit a linear model are not fully respected. As we have seen, predictors are not exactly normally distributed. Some little collinearity is present between *watch_count* and *comment_count*, *vote_count*[9].

After the encoding procedure, the number of predictors has increased a lot and I wanted to perform a feature selections in order to reduce its complexity, its tendency to overfit and make it easy to interpret. Actually, I had some thoughts whether to perform the feature selections. All of my model were not much powerful and showed a general tendency to underfit. However I noticed that building up the linear model with selected features could decrease the R^2 on the training set with (very small) gains in the R^2 on the test set. I used

9. Please, have a look at the heatmap on the pptx

4 different algorithms to perform the features selection and the outcomes were quite similar.

I started considered all the features selected in each algorithm to build up a first complex, hopefully overfitting model, but the results were quite poor. An overview with the parameters and the performances for the fitted model is given in the slides. Trying to remove non-significant variables did not bring any improvement, in fact it worsened performances. In any case, changing the number of predictors did not result in any relevant improvement worth to be noticed in terms of model predicting capabilities, as the R^2 on the test set was always around a quite low 0.2-0.25.

1.5.2 Regression Tree

It's now time to explore more powerful ML algorithms. My choice has fallen into regression trees which are able to better handle lots of categorical data with respect to linear regression. Furthermore, they are much appreciated for their interpretability. One advantage above all: the analysis of features importance.

The function to measure the quality of the split is the default MSE[10]. Following a trial and error approach, I played around with the depth of the tree and the maximum number of leaf nodes, to find out a fairly small tree which returned the best performances in terms of R^2 on the test set. Other parameters have been left as default.

Results shows how the feature importance focuses on mostly numerical variables, with a small contribution from the issue types. The reporter "Massie" seems to be explanatory.

1.5.3 Random Forest Regression

The natural development of regression tree is the Random Forest Regression, which fits a number of decision trees on various sub-samples of the data-set and uses averaging to improve the predictive accuracy and control over-fitting. I considered a forest of 100 estimators. Even here, I performed a very rough parameter tuning with a trial and error approach. It's interesting to see how the random forest allows way deeper and larger trees to be built, with an increase in performance. The feature importance charts follows the base-line of the previous one, giving great relevance to numerical variables. Instead the random forest is more able to exploit small pieces of information in other binary variables left apart by both the linear regression and the single tree.

Table 1: Comparison of performances for the different fitted models

Results	Multi Lin Reg	Regression Tree	Random Forest
Mean Absolute Error	1.7439	1.6775	1.6722
Mean Squared Error	4.8953	4.7905	4.4933
Root Mean Squared Error	2.2125	2.1887	2.1197
R^2 training	0.336	0.445	0.600
R^2 test	0.245	0.261	0.307

10. Mean Squared Error

1.6 Conclusions

It was clear since the very beginning that the extensive presence of outliers in both predictors and target variables would have affect the quality of our models. I did not want to remove the outliers, as usual procedure might suggest, since they were just too many and part itself of the predicting efforts[11]. All the models fails indeed in predicting very large alert resolution times, these being cleary part of that great sample of outliers in our population. Nevertheless, they have discrete prediction capabilities for smaller resolution times. Linear regression is the weakest model and clearly underfits the data. Regression Tree techniques have better performances but still struggle to fully explain variability of target variable.

Testing the model on unlabelled data coming from the validation highlights the following. Predictions from the Random Forest, supposed to be the most accurate, show a tendency to give more optimistic outcomes for larger resolutions time. They are therefore smaller compared to to predictions of the single tree and the linear regression. On the other hand, when predicting cases with supposedly low resolution time, the random forest returns a slightly larger predictions. Linear regression predictions are the ones with highest variance and returns more extreme predictions. For this interesting capability, I wouldn't discard the linear model a priori, as it might turn out to be useful in detecting those outliers with very large resolution time that are so frequent in our data.

2. Question 2

Implement the proposed model using the data and extract the predicted resolution time for 3 interesting cases.

Here I decided to report 3 interesting cases from the predictions on the validation set with unlabelled data.

- **Obs 1** : status:*Open* ; priority:*Major* ; issue_type:*NewFeature*

Index	Multi Lin Reg	Regression Tree	Random Forest
321	6836 days 01:35:33	286 days 03:56:24	177 days 19:13:03

  Assignee and reporter are the same non-frequent contributor. It has a high number of votes, comments and watchers, sign of great interest from the community. The linear regression returns an "explosive" predictions but however also the tree methods predicts quite long resolution times. In fact, looking back at the original data-set, therefore an information not processed by my models, this alert have been in the open status for almost 2 years. This might sound as the issue is not really proceeding and might remain like so for much longer.

- **Obs 2** : status:*PatchAvailable* ; priority:*Major* ; issue_type:*NewFeature*

Index	Multi Lin Reg	Regression Tree	Random Forest
120	711 days 13:59:55	77 days 00:48:05	73 days 19:42:33

11. Why so many alerts take such a long time? are we able to predict this behaviour?

This alert has not been assigned yet, which is normally a sign of longer times. There is however an available solution already: probably cutting[12] is taking care of it. Also this thread is pretty popular on the website as it has lots of comments, watchers and votes. The status suggests that it is going to be solved possibly soon. However, the issue seems stuck in the same status for almost 2 years without progressing. This behaviour again pushes the linear regression to return an "inflate" results while the trees method are more optimistic, predicting a more or less close resolution.

- **Obs 3** : status:*PatchAvailable* ; priority:*Major* ; issue_type:*Bug*

Index	Multi Lin Reg	Regression Tree	Random Forest
113	4 days 23:47:08	2 days 23:53:08	3 days 22:20:25

Assignee and reporter are the same frequent contributor[13], possible sign of a quick resolution of the issue. The model is in this status only since 5 days and there is already a Patch Available. Everything seems to indicate a quick resolution of the issue. In fact, all models return correctly quite low predicted resolution times."'

3. Question 3

Identify at least another information, available on the site of the community, that could be useful to improve the quality of the model.

During the inference analysis and the modelling of the supervised learning method I have been trying to understand how to best exploits the several given categorical variables. Soon I realized how it was not possible to use many of them. In fact, using a supervised learning approach, the model has to be trained with a training set containing the labelled target variables. Many of the available data on categorical variables were referred to observations that could **not** belong to the training dataset. One example for all, the variable *Status*. It can assume at least 6 different levels (*Open, InProgress, PatchAvailable, Resolved, Reopened, Closed*). As already explained, our training set contains only *Closed* and *Resolved* statuses. The rest of the information is completely lost. Training a supervised model on data that does not present those features implies that those cannot be "read" by the model even if present in new data.

Focusing my reasoning on the *Status* and the *Days_in_current_status* features, I wanted to find a way to integrate this wasted information into my supervised models. The idea was to find out a way to understand how much time the completed issues have remained in a determined previous status. In this way I could have quantified how much is the weight with respect to the overall resolution time, of being in a determined status. Doing so, I could have had a "number" with which to train my model such that, when looking at new non-labelled data, would have been able to read also intermediate statuses.

Looking at the website I understood how these status are in fact in sequential, one after the other. Their history is tracked in the **"Transitions"** field present in the dedicated

12. Mr Cutting is the most active contributor
13. Mr Tom White is one of the more active contributors

web page of the issue in analysis. For each issue is therefore possible to see how it came to completion (*Closed*) through the several different statuses, along with the time that the issue has spent in a determined status. That was exactly what I was looking for.

This information can be exploited in different ways. First, the model can better predict the total completion time, having the capability to compare the amount of time spent in the current intermediate status by the new issue, with the time that labelled issues have spent in the same status. We can do so by exploiting the *days_in_current_status* feature in new unlabelled issues. On the other hand, intuitively, the more transitions step the issue status does, the longer the total time completion. In fact, an issue can be reopened and processed again and this can lead to an increased time to resolve the issue.

It is clear how both test and training sets has to be manipulated to apply this logic. In particular, in the training set the *"Status"* variable should be encoded in as many variables as the its levels. Nevertheless, these encoded variables are not dummy binary variables but instead take a value. That value is the amount of time the issues spent in that status, which gives the name to the encoded variable. The amount of time spent in each previous status is given by the **"Transitions"** records. It takes 0 an all other cases.

On the other hand, non-labelled data in the testset should be also manipulated. Even here the *"Status"* variable should be encoded in as many variables as its levels. The variable *days_in_current_status* can be therefore be dropped and splitted in the new encoded variables to quantify the time of the current status.

4. Question 4 (optional)

All the Information on the website is included in the JSON file mentioned above. Extract from the file the information identified at the point 3 and insert it in the CSV.

I explained only theoretically all the passages in question 3 because I have not been able to find the additional needed information in the JSON file.

I have to admit it has been a challenge to try to manipulate the given JSON file. However, I have been able to explore it in almost all its parts and understand its structure. The given JSON file is composed by 1458 lines, each of them containing a big, nested dictionary. This corresponds exactly to the number of observations in my CSV dataset, representing all the analyzed AVRO alerts. It is therefore straightforward how each of those dictionary corresponds to a particular AVRO issue in the dataset. Looking at the *"_ID"* key of each dictionary, they appear to be also in the same order.

My attention has fallen from the beginning on the key dictionary *"Transitions"* available for each of the 1458 dictionaries. I was expecting to find the desired new data in that field. Unfortunately, that only contains an **empty list** for each dictionary.

Further exploring all the different sub-dictionaries for each alert did not bring any new valuable information.