

**Helpdesk Support Ticket   
Auto-Classifier**

**Project Report**

*July 2022*

**Derek Powell**

**Business Problem:**

In the world of technical support, identifying patterns and trends within the issues reported by users is critical for understanding solutioning gaps and making plans towards addressing those gaps. While some patterns and trends are easy to see, especially at a micro level, it can become problematic to identify quality patterns at a macro level.

For this reason, most ticketing (or CRM) software used by support desks includes selections within the ticket for agents to categorize the nature of the issue reported by the user. However, this presents a set of key challenges:

* As the complexity of the solution grows, there arises more and more categories of interest, and there comes more and more hierarchical selections, and the complexity of simply categorizing for agents the issue grows.
* As the complexity of categorizing the issue grows, so too does the cost of training help desk agents as well as likelihood of incorrect selections.

For large companies with complex solutions, categorizing every ticket accurately presents a very high training cost and even still then a high probability of incorrect categorization.

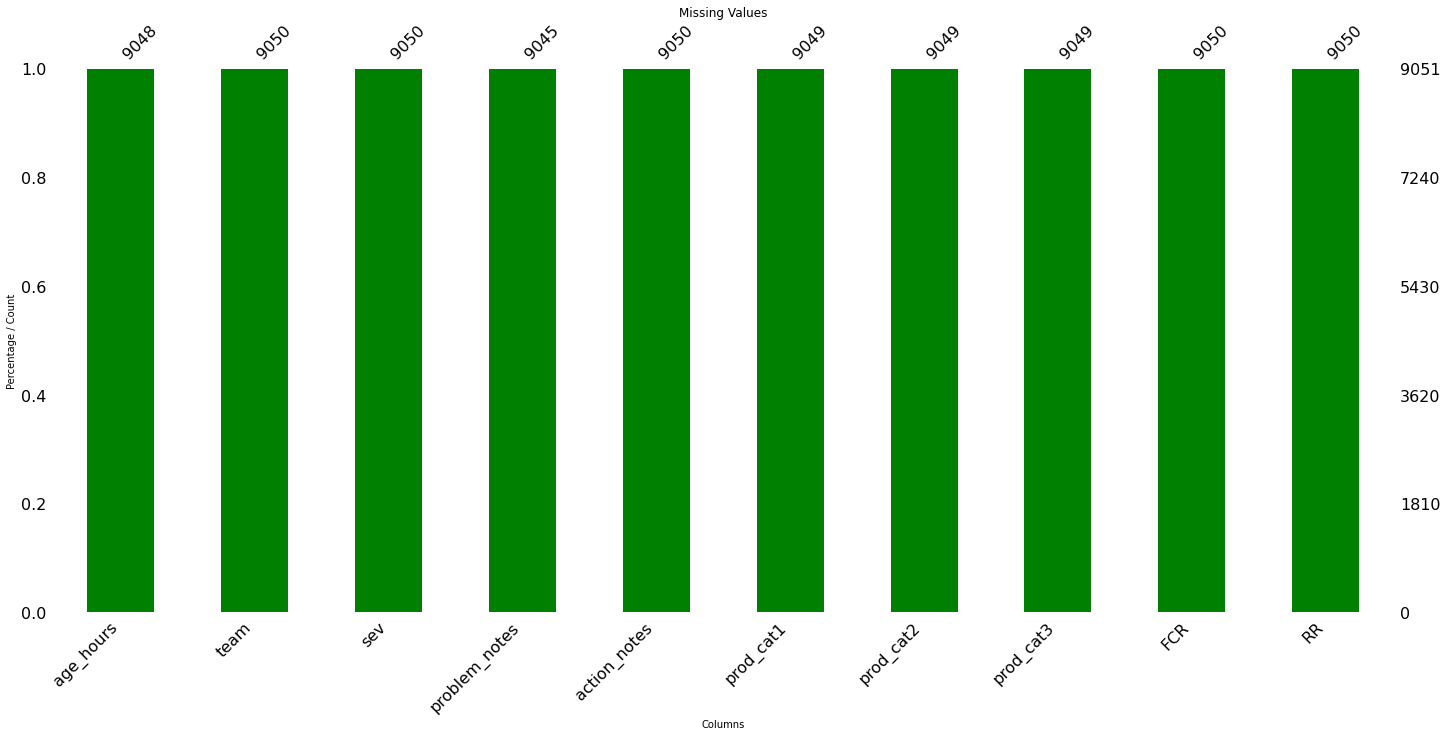
This project seeks to create a machine learning model which can evaluate features within a help desk ticket, including user commentary, and automatically (and accurately) classify the issue. This would greatly improve the value of this data in the hands of Product Management, Development, and Support organizations and would greatly reduce training costs.

**Dataset(s):**

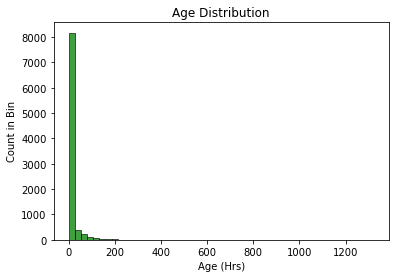
Within this project, I will be leveraging data found from my current employer’s ticketing software, seeking to take a recent, and well-rounded, sample of cases. The data is laden with features, representing each field found within a ticket, including discrete variables, continuous variables, and date/time objects. I was able to find everything I needed within the same data warehouse and data export.

**Data Wrangling & EDA:**

The dataset I used contained a lot of feature variables which I did not need nor planned to use for my project. I dropped all of these variables as an initial step, and then simplified the column names for the remaining feature variables. Then I checked the dtypes of the remaining features and (using domain knowledge) corrected the dtype of several features, as well as the type of each cell within certain features. For missing values, I found very few. Again using domain knowledge, I imputed the median for some continuous variables and then dropped rows for categorical variables I could not impute.

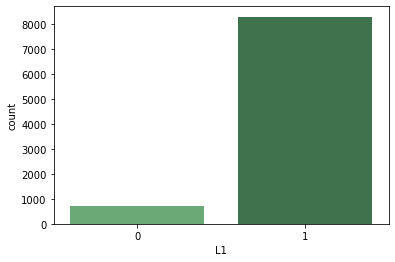


In checking for outliers, I found several in my age\_hours column which were several STDs away from the mean and represented <0.003% of the data, so I replaced these with the median of the feature.



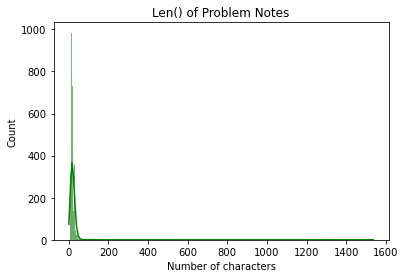
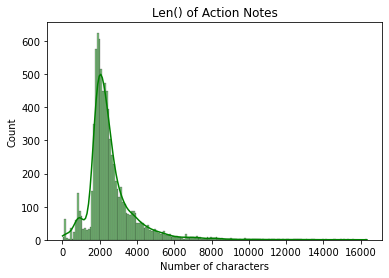
**Feature Engineering:**

I created several new features to support the analysis and potential performance of the model. One such feature was evaluating the ‘Team’ feature, indicating which team of agents within the org handled the resolution of the ticket. From my domain knowledge, I know these team selections generally are broken into two categories: junior (Lv1) and senior (Lv2) analysts. Using str.split I was able to identify and break out into a new feature a binary value indicating 1 for L1 and 0 for L2



My target variable was intended to be a single prediction but was defined across 3 features (Prod\_cat1,2,3) so I had to concatenate these into one new feature and then drop the individual features. Since the total number of unique classes created as a result of this was well over 100, I dropped any rows which had a categorization which didn’t appear at least 25 times in the dataset. This left me with a “more-agreeable” 64 classes (but still far from ideal).

Lastly, I examined the length of my two text features and created new features for the Len() of each cell.

** **

**Pre-processing:**

Using get\_dummies(), I encoded several categorical variables into their components. Then, I needed to clean and process the text features, as I anticipated this contributing the highest value toward the classifier training. Again, my text was broken into two features which I knew to be similar and directly related, so I combined them into one feature and dropped the individual features. I then cleaned the text by removing special characters, dead space, and dropping case to lower(). Later in vectorization, stop words were also removed. I then ran a TF-IDF vectorizer on my remaining text features, including n-grams of 2 and 3 words. This gave me well over 17K vocabulary features, even excluding any word or word combinations that occurred less than 10 times in the corpus. I then split the data into the X and y components and split those into a 75/25 training/test split. Lastly, I fit a PowerTransformer onto X\_train and then used it to transform X\_train and X\_test.

**Modeling:**

In the modeling phase, I ran 4 classifier models with 3 sets of input data each. The input data was the combined X dataset, the TF-IDF vocabulary features only, and the X dataset without the TF-IDF vocabulary. In this scenario, my Recall is the target scoring metric.

* **Random Forest** – this model had fairly mediocre results with out of the box parameters, with a macro avg 60% Recall on the combined dataset, and performed fairly the same with just TF-IDF features, but performed far worse without the TF-IDF features.
* **Gradient Booster** – this model failed to converge on hours of training with the combined dataset and the TF-IDF-only inputs. I lack the reasonable computing power needed to complete the training, so I abandoned those models. The Gradient Booster did train on the non-vocabulary X input, and had a mildly better performance than the non-TFIDF version of Random Forest, but both models scored too low to be usable.
* **SVM.LinearSVC** – This model had the best all around performance, regardless what X input was used. It gave the best performance using the TFIDF-only X input, giving a 68% macro avg on Recall. It is ultimately the model variation that I selected as the final model.
* **Logistic Regression** – This model failed to converge in 300 iterations on the full X input, had similar performance as Random Forest on the TFIDF-only X input. I did not run it on the non-TFIDF X input as the other models performed very poorly with this X input.

With LinearSVC as the final model, I ran a HalvingRandomSearch hyper param optimization on the C and Loss arguments, yet found that the default parameters yielded the best performance, so the model tuning was stopped there.

Conclusions**:**

In the goal of taking helpdesk ticket data and classifying it into the available labels for categorization, this project was not as successful as I had hoped. Overall success was impacted by the huge number of classes in the prediction variable. This mean that even the large ~10,000 row dataset was not able to allow for sufficient model training on some unrepresented classes, and cross fold validation would only serve to exacerbate the problem.

The freeform text fields (Problem Notes, Action Notes) appear to hold the most value in pattern identification, and if further pre-processing was done to clean up the content of them, their value may be recognized even further. Even still, the scores were not too bad for about half of the classes, and it is likely that performance can be improved even further via implementation of the steps listed in Future Scope section, below.

**Business** Insight:

This model could be used in its current form to make specific classifications on huge amounts of helpdesk ticket data, which could serve to lower training costs and improve data accuracy. Improved data accuracy could be used by Product teams to better understand bugs and gaps in their software and make improvements on behalf of the users.

Future Scope:

* Larger dataset to give more rows for each class in the target variable
* Try more n-grams, such as 3 or even 4, to provide more richness to NLP
* Reduce target classes to fewer selections to improve scores for remaining classes
* Try a Neural net model
* Better text pre-processing of Action Notes, which are cluttered with irrelevant words, e.g. email headers, agent IDs, date/times, etc
* Build a set of sequential classifiers, one for Prod\_cat1, then, for each available Prod\_cat1 selection, build another to predict that class' Prod\_cat2, then repeat for Prod\_cat3. This would require well over 100 classifier models, which could be trained in an iterative loop