PharmaSUG 2019 - Paper AD-052

Text Analysis of QC/Issue Tracker by Natural Language Processing (NLP) Tools

Huijuan Zhang, Columbia University, New York, NY
Todd Case, Vertex Pharmaceuticals, Boston, MA

ABSTRACT

Many Statistical Programming groups use QC and issue trackers, which typically include text that describe discrepancies or other notes documented during programming and QC, or 'data about the data'. Each text field is associated with one specific question or problem and/or manually classified into a category and a subcategory by programmers (either by free text or a drop-down box with pre-specified issues that are common, such as 'Specs', 'aCRF', 'SDTM', etc.). Our goal is to look at this text data using some Natural Language Processing (NLP) tools. Using NLP tools allows us an opportunity to be more objective about finding high-level (and even granular) themes about our data by using algorithms that parse out categories, typically either by pre-specified number of categories or by summarizing the most common words. Most importantly, using NLP we do not have to look at text line by line, but it rather provides us an idea about what this entire text is telling us about our processes and issues at a high-level, even checking whether the problems were classified correctly and classifying problems that were not classified previously (e.g., maybe a category was forgotten, it didn't fit into a pre-specified category, etc.). Such techniques provide unique insights into data about our data and the possibility to replace manual work, thus improving work efficiency.

INTRODUCTION

The purpose of this paper is to demonstrate how NLP tools can be used in a very practical way: to find trends in data that we would not otherwise be able to find using more traditional approaches. Yes, we can perform Proc Freq on our spreadsheets and determine what ‘categories’ (however ‘categories’ are defined: by big buckets such as ‘data’, ‘spec’ and ‘aCRF’ or as granular as ‘AE’, ‘PARAMCD in Spec’ or ‘Annotation Update’) but frequency counts often don’t get at the underlying problem, which can be found using the free text entered about the actual problem. For this project, we looked at almost 10,000 rows of text and found that there were more than 1000 different ways the issues were classified! Given the hundreds of thousands of text fields that were in the QC and Issue trackers we planned to (1) take an initial step to get a general idea of the ‘topics’ of the entire collection of problems and (2) train a classification model based on those problems and the corresponding pre-specified categories (labeled data) and apply this model to the problems without a pre-specified category (unlabeled data) to predict what entry a text field should be entered into.

The benefit of this analysis is that anyone, from the individual contributor who is responsible for entering data into the QC/Issue Tracker, to the VP who is looking for a transparent and clear idea of the text that, when presented using NLP tools, go much further than frequency counts or even a ‘we already know these are the main issues’ approach.

We can then, using NLP tools, create process that allows us to fix the general process problems and hopefully let us write fewer issues and code more programs!

EXPLORATORY DATA ANALYSIS

We combined QC and issue trackers into one spreadsheet, which contains three hierarchical user-defined categories (from general to specific): *origin_sheet*, *point_of_issue*, and *question*. This spreadsheet includes almost 10,000 records and more than 200,000 words.

MANUAL CATEGORY

The questions belong to 1250 points of issue, which is in column *point_of_issue*. These points of issue further belong to 31 original sheets, which is the column *origin_sheet*. However, only 4 categories are expected, i.e. Specs, TFL, aCRF and Raw. To reduce the number of categories, we first further categorize them manually according to their original sheet names. Here is how they are categorized.

- ***Specs***: adam, adam and adam, adam and specs, define adam, define sdtm, sdtm, sdtm and specs, adtm_adam;
- **SAP**:
  - ***TFL Shells***: figures, listings, tables, tfl, sap & mock she, sap and tfl, sap shells;
  - ***SAP Methods***: sap, sap methods;
- ***Raw***: raw, raw data;
- ***Guide***: reviewer guide;
- ***aCRF***: acrf;
- (Confusion/Mixed): ***general***, ***issues***, ***sheet1***, ***sheet2***, ***sdtm raw***, ***acrf and spec***, '***protocol 1', 'protocol 2', 'protocol 3', 'protocol 4'*** ('***Country' Reimbursement***).

The categories in bold and italic type are the 13 manual categories. Adam and sdtm related categories belong to *Specs*. Figures, listings, tables, etc. belong to *TFL shells*. Sap and sap methods are categorized as *SAP methodology*. There is ambiguity in some categories, such as *general*, *issues*, *sheet1* and *sheet2*, which are not meaningful. Also some categories seem to belong to both *sdtm* and *raw*, but are supposed to be separate. These ambiguous categories are kept as their original categories. All "procotols's" are grouped as '*Country' Reimbursement* (based on what we know about the data). Figure 1 shows the structure of the dataset.

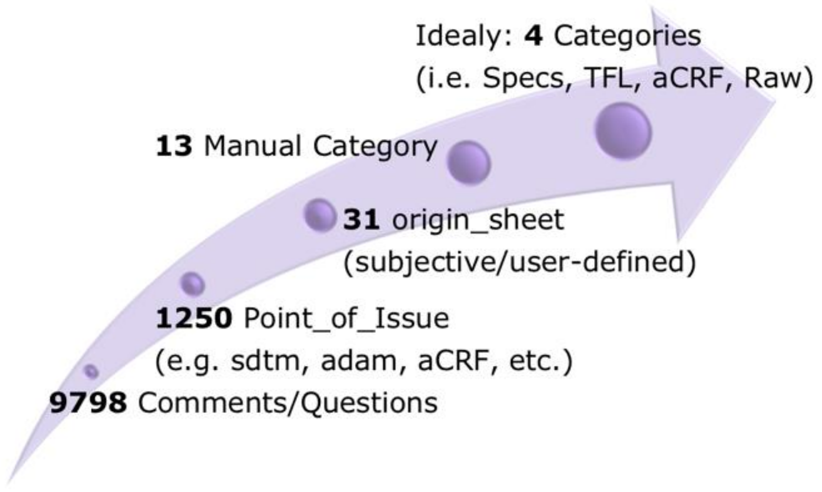

Figure 1. Dataset Structure

DESCRIPTIVE ANALYSIS

Figure 2 shows the data proportions of the 13 categories.

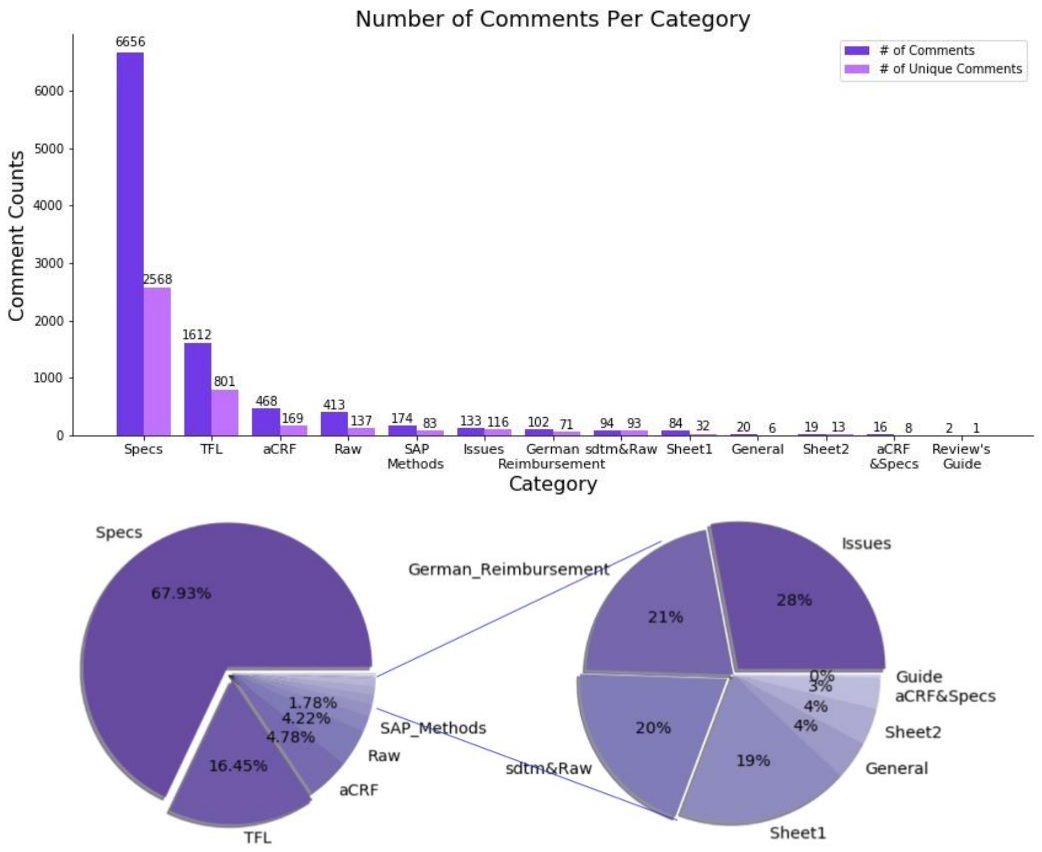

Figure 2. Data proportion per category

The vast majority (~68%) of the issues come from specs. There are a lot of duplicates - if we remove the duplicates, there are over 2000 unique issues about specs. Beyond *Specs*, there were three other categories that had relatively large percentages: *TFLs* (16%), *aCRF* (5%), *Raw* (4%), etc. The category *Guide* only has 2 questions and they're duplicates, so it's removed from the dataset in later analysis. Our later analysis is based on the remaining 12 categories.

METHODS AND RESULTS

TOPIC MODELING (LATENT DIRICHLET ALLOCATION)

Topic modeling is a tool in NLP, which is used to identify most common topics within a collection of unlabeled documents. Latent Dirichlet Allocation is a basic and the most widely used topic model. [1] Many other topic models are based on LDA and they relax some assumptions of it (e.g. Hierarchical Dirichlet process).

To build this model, the number of topics to be extracted needs to be decided first. For each topic, it gives us some words that are most important in this topic. Then, we can name the topic according to these words.

To perform the analysis, all records in *question* are converted to lower case, tokenized, stop words removed and lemmatized. Having observed that 'please' appears in most questions, we add 'please' into stop words to remove it from analysis.

Firstly, we assume that we only have *questions*, without category as the label and perform LDA to get a sense of data. But performance of LDA suffers from data imbalance, which is exactly what our data showed. To overcome this imbalance, we resample the data to obtain a balanced dataset. Then we train an LDA model on this dataset.

Before building a LDA model to resample for a balanced dataset we need to decide which categories to sample from first. We tried 3 combinations: (1) 3 topics: Specs, TFL, aCRF; (2) 3 topics: Specs, TFL, Raw; (3) 4 topics: Specs, TFL, aCRF, Raw. The training data is obtained by taking 3 topics (e.g., Specs, TFL, Raw) and then taking category Raw and dropping duplicates to get 138 records. Then, in order to make the training data balanced, we also need 138 records for Specs and TFL, separately. Since there are many duplicates, we sort questions by the numbers of duplicates and take the first 138 records that have most duplicates, intending that they are the most representative records within this category.

An interactive visualization can be created by pyLDAvis package (Figure 4). After taking a look at the results, choosing 3 topics (Specs, TFL and Raw) arguably makes the most sense (Figure 3), considering topic word meaning and word distribution in each topic. From top to bottom, these words can be interpreted as Specs, TFL and Raw.

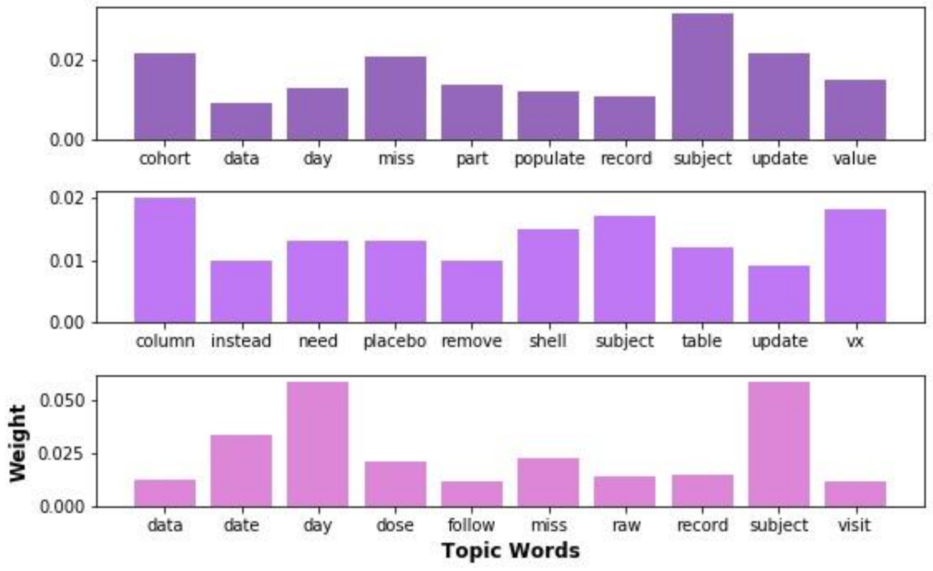

Figure 3. Topic modeling results

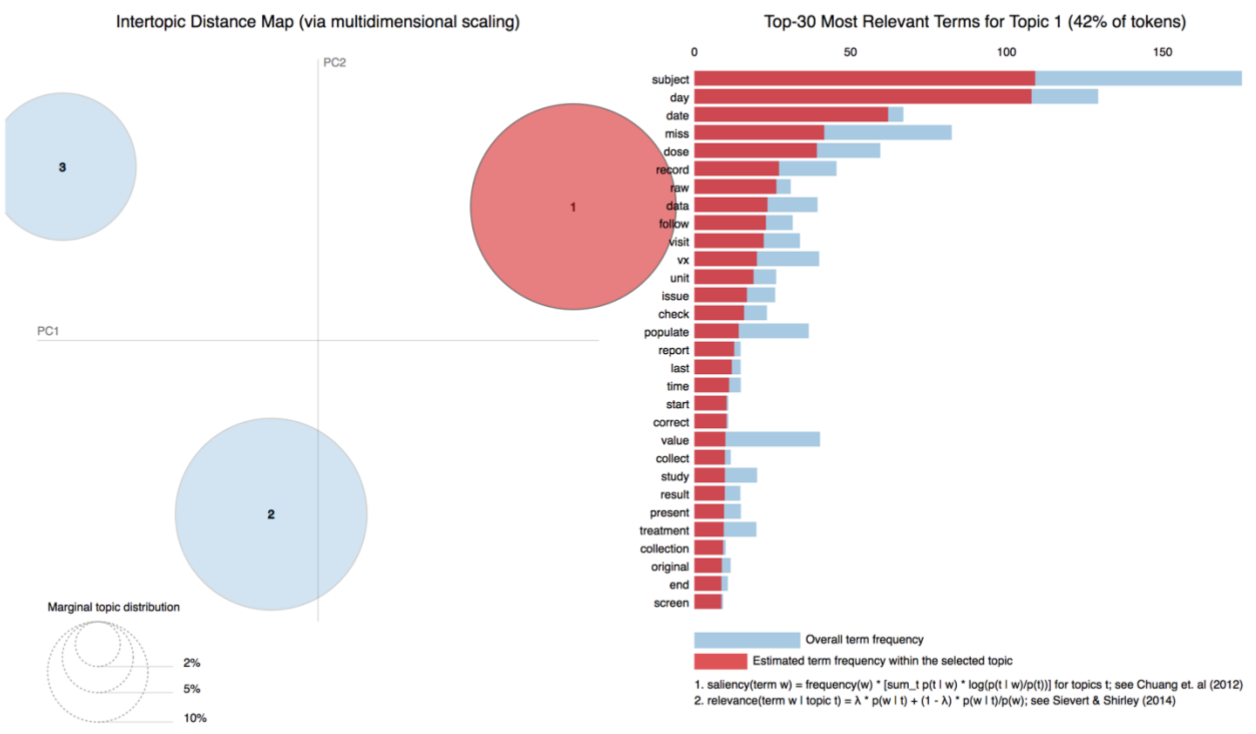

Figure 4. An interactive visualization by pyLDAvis

One problem here is that due to the data imbalance, the overall sample size is small. So the model cannot capture the features very well. Another possible problem is that the words used in each category are similar to some extent, which may also cause performance issues of the model.

To sum up, we used the topic modeling approach here to get a rough sense about how all the questions characterize the data. Further analysis is needed to provide more insights.

DOCUMENT SIMILARITY

Previously we looked at unlabeled documents. In order to pre-train a model we sample training data from several categories, still "pretending" to not know document labels when building a model. Here we'll take labels into consideration.

Intuitively, we start at calculating word frequencies, to see if there're some similarities between those categories (Figure 5).

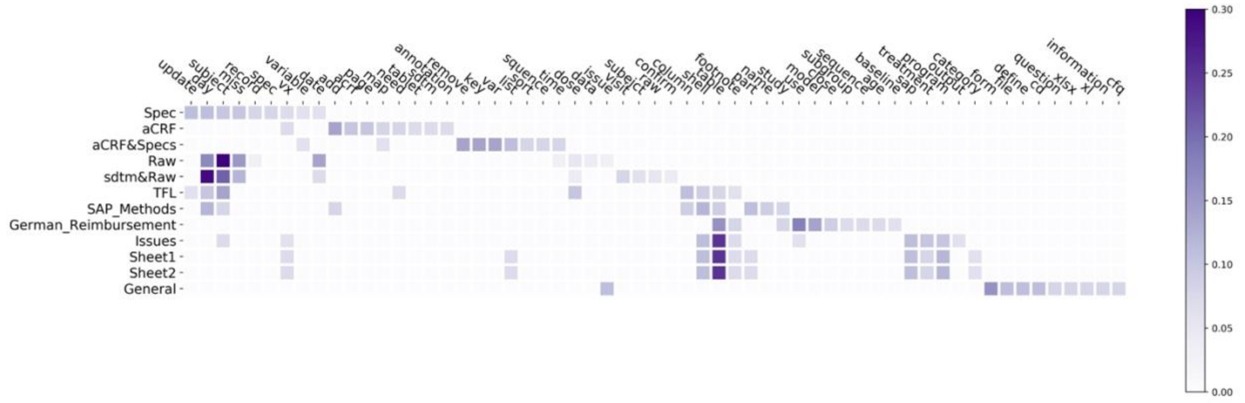

Figure 5. Ten most frequent words within each category ('please' removed)

Here are the 12 categories. We recalculate word counts within each category and obtain the ten most frequent words to calculate percentages within this category. The word ‘please’ appears so frequently and is not meaningful, thus it is taken out from the plots.

In terms of word frequency, category *aCRF&Specs* is not similar to either Specs or aCRF. *Sdtm&Raw* is roughly similar to Raw e.g., words ‘day’, ‘subject’ and ‘miss’. *TFL* and *SAP Methods* are similar because they are both about the SAP. Categories *issues*, *sheet1* and *sheet2* are similar, and we may suppose they are all about one particular category.

We further explore by document similarity, which not only considers word frequency, but also word location within a document (Figure 6).

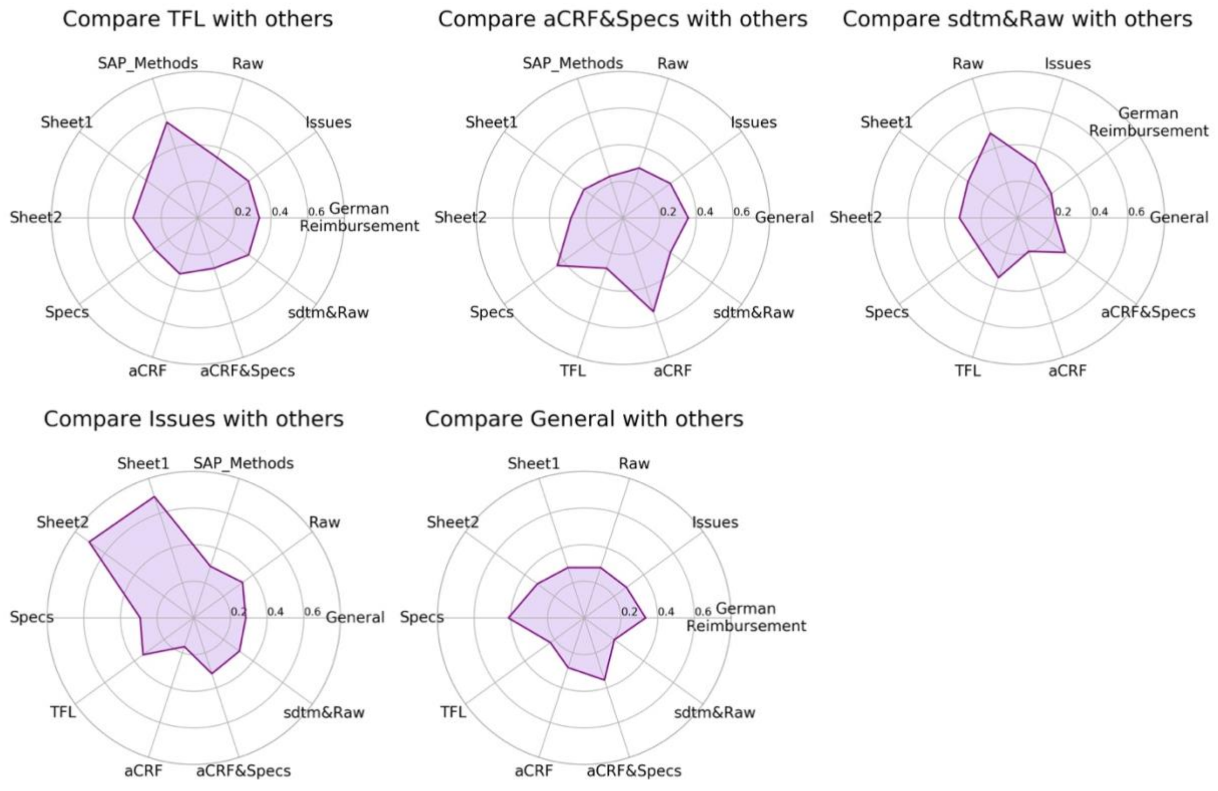

Figure 6. Compare similarities between categories

TFL is most similar to *SAP Methods*, *sdtm&Raw* is most similar to *Raw*, and *issues*, *sheet1* and *sheet2* are similar, which are consistent with what we see from the heat map. *aCRF&Specs* are similar to *aCRF* and *Specs*, which is reasonable. *General* is probably a mix of several categories, of which *Specs* takes the largest part (consistent with what we have discussed about regarding data proportion).

Up to now, we know some properties at the category level. Although we guess that some categories may have relationships to each other, without any context we still have no idea what they are talking about, e.g. *issues*, *sheet1*, *sheet2*. Thus, we will move from the category level to the document level.

Back to our dataset, we’ve considered over 9000 questions and 13 manual categories. More information is stored in variable *point_of_issue*. As we see, although some categories are ambiguous, in some records, they still contain information in *point_of_issue* (Table 1).

Manual Category	Question	point_of_issue
issues	Both columns for PES and SS population were requested…	adsl

Table 1. An example of where the category is ambiguous but with information in point_of_issue.

Ideally, *point_of_issues* can be considered as subgroups of the manual categories, that is, each point of issue belongs to, and only belongs to, one manual category (of course this is not always the case). We can learn from their convention and use this convention to map *point_of_issue* to *Manual Category*. When there are clear categories for *point_of_issue* and *Manual Category*, we can get one-to-one or multiple-to-one mapping; for records whose *Manual Category* is ambiguous, applying the mapping rules obtained above to *point_of_issue* and get a *Mapped Category*; one-to-multiple mapping will appear as NaN. Thus, all information comes from the dataset. For this example, points of issue ae, adsl, vs, etc. belong to *Specs*. Then map *point_of_issue* adsl to *Specs* (Table 2).

Manual Category	**Question**	**point_of_issue**	**Mapped Category**
issues	Both columns for PES and SS population were requested…	adsl	Specs

Table 2. An example of mapping from *point_of_issue* to Manual Category

(information stored in Mapped Category).

Not every *point_of_issue* belongs to only one *Manual Category*. Some *point_of_issue* can belong to multiple categories. For example, we find that *point_of_issue* '*vs'* belongs to *Raw*, *Specs*, *aCRF* and/or *sdtm&raw*. And there are cases in which both columns are ambiguous, e.g. *point_of_issue* general belongs to *General*.

Finally, we use another approach to make a prediction to those documents with ambiguous categories. Text classification is widely used nowadays in such areas as spam filtering, which you must have experience when receiving emails, and language identification, if you open the Google translation website, you can enter some text and its language can be detected as English or whatever, etc. Similarly, here we're trying to assign a pre-defined label to a document, so that these documents can be categorized. In order to capture text features, we use a distributed memory model (PV-DM) to convert paragraphs to paragraph vectors D. [2] And we use tree boosting in XGBoost implementation to make multiclass classifications. This paragraph vectors D obtained in Gensim Doc2Vec implementation are used as predictors in tree boosting. And, lastly, we use a 5-fold cross validation to build a model. This model is applied to predict unseen paragraphs (i.e. questions).

In more detail, the training data is obtained by following steps. The dataset contains 9798 records, and there are 9793 records after removing records where *question* is NA. After removing duplicates in *question* we get 4098 records. Removing *Review's Guide* category we get 4097 records. There's a study pointing out that doc2vec performs particularly strongly over longer documents. [3] A further filter of questions with more than 80 characters leads us to 2760 records finally.

This classification is created as follows.

1. Divide 2760 records into labeled data (whose Manual Category is Specs, TFL, aCRF or Raw) and unlabeled data (other groups). Shuffle labeled data and apply Doc2Vec to obtain paragraph vectors. Divide these vectors into 80% as training data for cross validation and 20% as test data. Use model obtained in Doc2Vec to infer vectors for unlabeled data and apply model obtained by xgboost to it for classification.
2. Divide 2760 records into labeled data (whose Manual Category is Specs, TFL, aCRF or Raw) and unlabeled data (other groups). Resample labeled data to make it balanced. Shuffle this balanced dataset and then apply Doc2Vec. Then get vector matrix and add noise to it. Divide into 80% and 20% for xgboost. Infer vectors of unlabeled data and apply model by xgboost for classification.
3. Shuffle the 2760 records and apply Doc2Vec. Divide these 2760 vectors to labeled data and unlabeled data. Further divide labeled data to 80% and 20%, and then apply xgboost. Apply model obtained by xgboost to unlabeled data vectors for classification.

Finally, we choose the third approach. Firstly, since we are not going to use the model obtained by Doc2Vec for classification, it's not necessary to train model on labeled data and then infer vectors for unlabeled data. If doing so, the vectors cannot capture the features of unlabeled data and this step will result in lower performance. Instead, we use Doc2Vec to get paragraph vectors for both labeled and unlabeled data, without any inferring steps. Secondly, since there's no inferring step, there's no need to make labeled data balanced. That's why we adopted the third approach at last.

In the third approach, accuracy on training data is 1 and accuracy on test data is 0.9663. Then apply this model on unlabeled data and output to an excel file. Model performance on unlabeled data needs to be further discussed.

CONCLUSION

The data suggests we should pay most attention to issues about specifications, TFL, aCRF and Raw data, of which specifications take the largest part (~ 68%). A classification rule needs to be improved as the category names are more than expected and cause confusion sometimes. A suggestion is that making a dropdown (include a 'catch all' category so new classifications are not created) when people are making classification selections, which will ensure there will be no extra categories. A pre-trained model can be applied to classify comments to provide a 'target' when checking manually.

REFERENCES

[1] David M. Blei, Andrew Y. Ng and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003) 993-1022.

[2] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)* 1188–1196, Beijing, China.

[3] Jey Han Lau and Timothy Baldwin. An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. *Proceedings of the 1st Workshop on Representation Learning for NLP* 78-86, Berlin, Germany.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Huijuan Zhang
E-mail: hz2510@columbia.edu

Name: Todd Case
Enterprise: Vertex Pharmaceuticals
Address: 50 Northern Avenue
City, State ZIP: Boston, MA 02210
Work Phone: 617 961 7907
E-mail: todd_case@vrtx.com