MACHINE LEARNING BASED TICKET CLASSIFICATION

Aishwarya Sudhakar, Ariel Reches, Chris Watson, Kruti Chauhan, Nicholas Monath, Abe Handler;
University of Massachusetts – Amherst, Industry Partner: Pratt and Whitney

***Abstract*—All organizations have a ticketing system that collects service requests for a product or a service. Tickets from this system is then assigned to subject matter experts for resolution. As of today, mapping the ticket to an expert is the most challenging task that comes with the cost of time. The goal of this project is to apply machine learning models to perform the mappings. The main motivation behind applying machine learning techniques to this problem is to help save time and extend the models built for a specific dataset to work efficiently across different domains. Simple rule-based models cannot adapt to changes to the data as the system evolves over time. Machine learning approaches can scale out and understand semantic meanings that are difficult to be laid out as rules. In our solutions we have applied both supervised and unsupervised learning approaches to the problem. We handle the problem from different approaches and identify the most feasible one.**

***Index Terms*—Ticket classification, Feature Extraction, Word Embedding, Clustering, Bi-directional LSTM, FastText**

I. INTRODUCTION

Organizations have many mechanisms in place for resolving issues that arise in their service or product. A key part of this resolution is in identifying subject matter experts who can resolve this issue. In most organizations, identifying an expert for the issue is manual. This is a time-consuming process as it involves a gate-keeper who is aware of the existing system, the problems that may arise, the experts for the system and their availability. Alleviating all the problems that exists in this domain is not plausible, we look to machine learning solutions to help identify the most suited expert for an incoming issue. Although inherently this problem appears to be a direction application of a classification task, we identify unsupervised approaches to understand the underlying representation of the data.

II. OBJECTIVE

The primary objective of this project is to build a machine learning model that can identify the subject expert for an incoming issue. The system is aware of a set of available experts. Mapping an incoming ticket to one of these experts is interpreted as a classification task where we use experts as classes. However instead of confining ourselves to the supervised methods, we also perform unsupervised learning models to identify the underlying structure of the dataset.

III. METHODS

A. Feature Extraction

The main challenge of this project is extracting features from the existing array of assigned tickets. Tickets are plain-text representation of the issue it addresses. Extracting features from a free form text field is challenging and affects the scalability and the extensibility of the model into other domains. Our experiments use two major feature extraction techniques namely Bag of Words and the Term Frequency – Inverse Document Frequency methods.

Bag of Words is a vector-space model that uses the multiplicity of each word in the document to represent the document. The frequency of each word in the document is obtained and normalized. Thus, each document is expressed as a vector in the space.

The issue in a simple Bag of Words representation is that words with most frequencies are biased that those that are infrequent but however are important. Term Frequency-Inverse Document Frequency mitigates this problem and gives us a vector representation that highlights those words that uniquely identifies the document.

$$\mathrm{idf}(t) = log\frac{n_d}{\mathrm{df}(d,t)} + 1$$

Fig. 1. Calculating the TF-IDF for a specific term in the corpus

B. Supervised Learning

A supervised classification model is a one that is trained with labels. We use several out of box classifiers like Support Vector Machines (SVM), Random Forest, K-Nearest Neighbors Classifier, Multi-class logistic regression, Decision Tree Classifier and Naïve Bayes Classifier.

Apart from the basic classifiers we used a hierarchical

classifier which works well with unbalanced distribution of classes in the dataset. FastText is a text classification library from the Facebook AI Research Lab. It uses a hierarchical classifier to classify text into imbalanced classes. It uses low-dimensional matrix representation of word vectors using simple addition of word vectors. It is as efficient as deep learning models in terms of accuracy. The training time in fasttext is not high - dataset can be trained in seconds.

Bidirectional LSTM are currently the state of the art in text classification problems. The recurrent architecture allows learning over the sequence, rather than just word frequencies of the Bag of Words model. By disregarding the sequential nature of natural language, the BOW model cannot extract any meaning from the word order. Using BD-LSTM, we can retain and learn over this information.

C. Unsupervised Learning

Classical approaches to this problem works well, but to understand the problem better we applied some unsupervised techniques.

Principle Component Analysis is a traditional statistical procedure that reduces the linear correlation among the components. The representation of data from our feature extraction method was a very sparse matrix. We reduced this high dimensional representation of tickets and experts into their principle components down to a same dimensional subspace. With vectors representing each ticket and expert, a cosine similarity measure that identifies a unique expert closest to a ticket can give us the best expert to handle the ticket.

We can also use the reduced representation of the text to query the subspace and interpret the representation of the documents in the vector space models.

Another approach to this problem is that of clustering. Instead of clustering the tickets into big groups of subjects, we used each expert as a cluster and tried several clustering methods to cluster the tickets into that expert's cluster. Results from all methods are discussed in detail in the next section.

IV. EXPERIMENTS

A. Dataset

The dataset used for this project is the online Mozilla Firefox dataset. The features of the dataset are:

1. Bugs in Firefox Product
2. Bugs that were either in the closed or open state.
3. The dataset consists of 9946 bugs assigned and resolved.

The table below provides a comparison of the Mozilla data with the Pratt and Whitney Dataset. Approximately equivalent fields are shown on the same line.

TABLE I
Comparing Pratt and Whitney Dataset to Mozilla Dataset

Pratt and Whitney Dataset	Mozilla Dataset	Data Type
Task ID	Bug ID	Unique Id
Task Description	Bug Description	Free Text
Task Type	Product	Categorical
----------	**Component**	Categorical
Task Short Description	Keywords	Free Text (Short)
Task Status	Bug Status	Categorical
Analyst ID	Assignee	UniqueId

The distribution of the dataset revealed that most of the descriptions in the bug dataset are short. Below is the histogram of the bug data binned by the number of characters in the description.

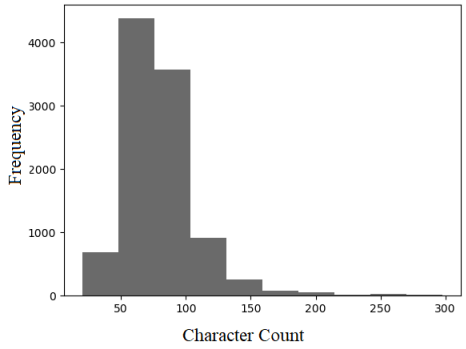

Fig. 2. Distribution of character count in the bug description field of the Mozilla

B. Data Preprocessing Pipeline

A basic data pre-processing pipelines was set up using python. Special characters and stop-words were removed. Bi-grams learned by the Gensim Phraser are joined back together. The following are the fields used from the dataset:

1. Product - Categorical Data
2. Component - Categorical Data
3. Keywords - Descriptive Data
4. Summary - Descriptive Data

The bugs assigned to "bugzilla" was removed from the dataset as these samples were bugs assigned to the system and not an assignee.

C. Supervised Learning

A combination of simple classifiers and vectorizers are used on the dataset using a grid search and 5-fold cross validation. Training Set: 5,885, Test Set: 4061

We used Bag of Words and TF-IDF Vectorizers. Classification –
1. SVM (Parameters – Kernel [rbf, poly], Degree [20,30,50], C [1, 10, 100, 1000])

2. Decision Forest (Parameters – Estimators [9,54], Max Depth [1, None], Min-Samples-Leaf [1, 10])
3. Logistic Regression - (Parameters – C [.0001, .01, 1, 100])
4. Decision Tree (Parameters – Max Depth [20, 100, None], Min-Samples-Leaf [1, 10])
5. K Nearest Neighbors Classifier (Parameters – Neighbors [1-20])

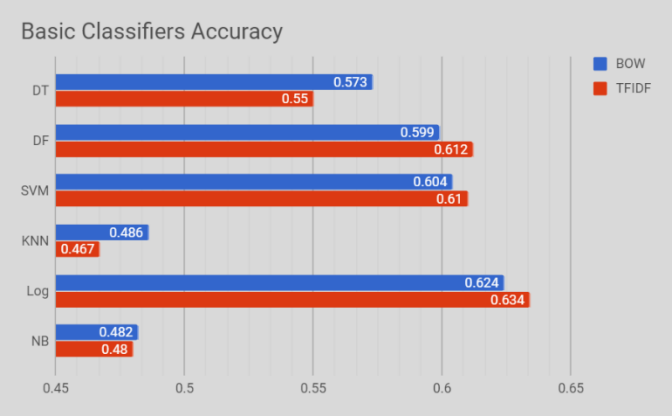

Fig. 3. Results from basic classifiers

From our CV search it was the case with Logistic Regression, SVM the most extreme parameter produced the best results.

To achieve interpretability on our models, accuracy of any model on the test set can be broken down by various bug features such as "Component" or "Assignee". P&W expressed interest in including this as part of the classification pipeline.

This allows for future work to understand the reasons a bug may be harder or easier to predict. However, we are still searching for a way to apply this type of information.

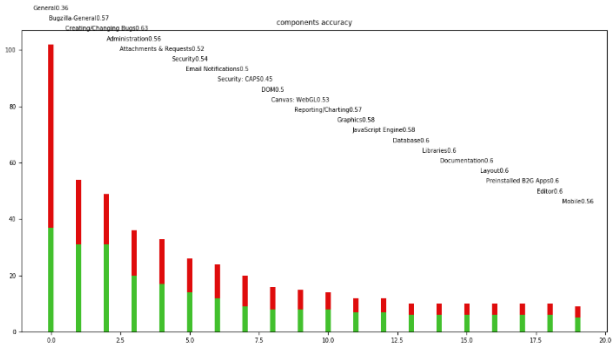

Fig. 4. Accuracy of Decision Tree Bug Classification by "Component" Field

As a next step to our experiment we used the FastText Classifier to train the Mozilla bug dataset and classify them to their assignees. Based off the understanding of word representation in the dataset.

We used fasttext classification for two datasets:

1. Using Bug Description Column as an input feature:
   Model Constant Parameters: Learning Rate 1.0, Epoch: 50, Bucket: 20000, Word Vector Dimension: 50, Loss: Hierarchical SoftMax
   The Results for Precision are:
   a. Word N Grams:2, Precision: 0.46 ± 0.01
   b. Word N Grams:5, Precision: 0.44 ± 0.01
   c. Word N Grams:7, Precision: 0.42 ± 0.01
2. Using Component + Keywords columns as input features:
   Model Constant Parameters: Learning Rate 1.0, Epoch: 50, Bucket: 20000, Word Vector Dimension: 50, Loss: Hierarchical SoftMax The Results for Precision are:
   a. Word N Grams:2, Precision: 0.44 ± 0.02
   b. Word N Grams:5, Precision: 0.52 ± 0.03
   c. Word N Grams:7, Precision: 0.47 ± 0.03

It can be observed that using component + keywords in the fast text classifier has better classification accuracy.

Our next approach was to use a deep learning model for classification. The model architecture used is as follows: (LSTM forward layer - LSTM backward layer - Attention layer- Dense layer). RMSprop was used to update the gradients, and binary cross entropy loss was taken into consideration. Currently word2vec is being used to obtain the vector inputs to the Bi-LSTM, and keras is being used to implement the model. mechanism to the model. Additionally fasttext could to be obtain word vectors, in order to input better representation of the data to the model.

This model was found to be highly sensitive to the amount of available training data. Initial experiments showed an accuracy around 55% when the data included all assignees with more than 10 bugs. If we raise the cutoff to assignees with more than 50 bugs, we can achieve an accuracy of 70% and a top-2 accuracy of 82%.

One clear problem with our dataset is that the class representation is not balanced. The model may be learning to bias its predictions toward the most highly represented classes. As we can see from the breakdown of accuracy by class, the most highly represented classes have the greatest accuracy. This suggests that that a bias is indeed present. To solve this problem, it may be useful to balance the representation of each class in the dataset. Doing so however would greatly reduce the number of bugs available for training.

Our experiments indicate that neural networks are very sensitive to the amount of available data, and the distribution of the classes in the data. Real datasets may only have a few examples for some assignees. For this reason, neural network learning models may not be the best approach to this problem. The ideal model should be able to handle variation in the amount and distribution of training data.

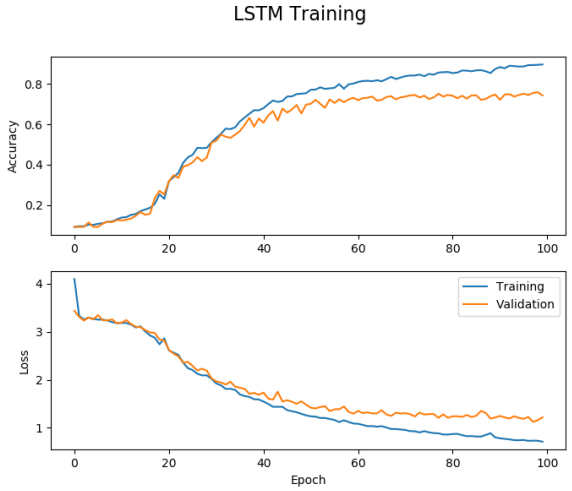

Fig. 5. Learning Curves

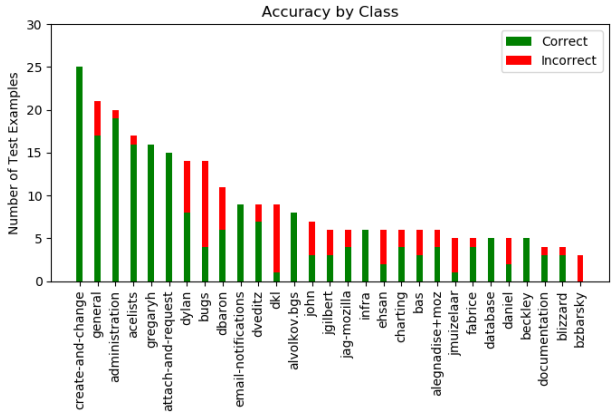

Fig. 6. Learning Curves

D. *Unsupervised Learning - PCA*

The dataset presented is of 9937 bugs with 11397 dimensions. It is found that the sparsity for this data is 99.99%. A dimensionality reduction might help us reduce the subspace of the data and identify the latent features of bugs.

SVD analysis on this dataset revealed that with 2000 principal components about 80% of variance could be maintained.

One of the approaches we tried was to represent the bugs by their latent features. In addition to this, the analysts could also be represented in terms of the bugs they resolved. So, a document-term matrix for analysts was constructed, where each document represents all the bugs solved by the analysts and the terms include all the words in the corpus. We had 1133 documents represented as 11397 features. This is again a sparse matrix with 99.9% sparsity. Dimensionality reduction on this would reveal the latent features of the analysts by the bugs they resolved.

If both these latent factors were projected onto the same subspace then by simple cosine similarity we could identify the analyst closely related to the bug. But however, since we have only 1133 analysts, the maximum reduction possible was only 1133 principal components which on the bug document-term matrix accounts to 65% of variance. In assigning the bug with the most closely represented analyst and in comparing it with the labels, we were able to achieve only 0.1%-0.8% accuracy. This indicates that with 1133 components we have loss of data and the bugs are not represented efficiently.

E. *Clustering*

Another unsupervised experiment that we on the Mozilla Dataset is clustering. Usually clusters are distinct subgroups of the data. However, to provide a reasonable interpretation for our model. We perceived each assignee to be a cluster on its own. We used two classical clustering methods: KMeans and Agglomerative clustering. We used the basic vectorizers to build the input for our unsupervised models. However, due to conjoined words our clusters were not pure.

We then used fasttext's text representation module to build vectors for each word. FastText has two text representation techniques: Skip-gram and Continuous Bag of Words. We tried experiments with both. We then constructed an input matrix with these words vector representation for each bug. This matrix was then given as input to the clustering algorithms.

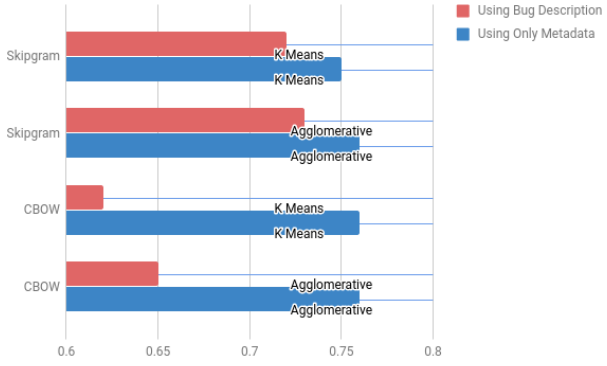

Fig. 7. Homogeneity of Clusters

We used the classification labels as the ground truth to measure the homogeneity of clusters. We got pretty good clusters compared to use the basic vectorizers. Notably, using metadata alone which is the categorical columns, the bag of words and the skip-gram representation gave almost similar results. However, Using the bug description where words were joined together, the bag of words did not do that well. Whereas the character n-gram gave better separated clusters.

Even though we got well separated clusters, we were unable to convert them to a deliverable.

Example:
Bugs: Firefox, Wi-Fi
True Assignee: changysin (24 bugs)
Pred Assignee: 10 different clusters
Pred Cluster for Changysin (13 bugs)
Bugs: firefox, core, thunderbird, tasks

A good cluster had bugs from different components in it. And we were unable to without loss of information map the clusters to the assignees.

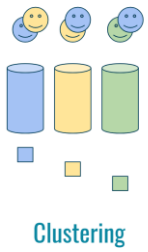

Fig. 8. Unable to map clusters with assignees

F. Querying Word Vectors

Motivation: To query the feature representation of the dataset to identify closely related bugs.

Features Used: Bug Description

A term-document matrix was created using the Bag-Of-Word Vectorizer used in our experiment with the Simple Classifiers. With the documents being the words in each bug's description column and terms being all the words in the corpus.

We reduced the dimensionality of the term-document matrix to 1000 components and represented the low rank matrix in a KD-Tree data structure. We used single and double query words to find the 10 closest neighbors by Euclidean distance. The results were semantically unrelated words which occurred within a small window of the query word in the original description. No perceivable reasoning could be made of the results of the query.
Term-Document Matrix: 17417 words - 9937 bugs
Sparsity: 99.9%

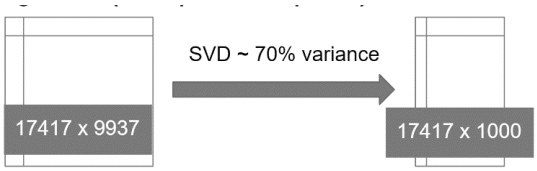

Fig. 9. Dimensionality Reduction using Bug Description

Example:
Query: security
Actual: javascriptvalidatelogin (1 bug), presenting (1 bug), awarded (1 bug)
Expected: process, sandboxing, enterprise, information
Actual Records: Enterprise Process Sandboxing (35 bugs), Enterprise: Information Security (20 bugs)

The reason for the poor results were attributed to the fact that words were conjoined together. And using character n-gram for vectorizing might improve performance.

Features Used: Component + Keywords

To avoid the sparsity and the poor representation of bugs, we created a model using the metadata of the bugs and analyze if these falls within a proximity on a reduced subspace. The experiment was to understand if the bugs are better represented using metadata rather than the text description. Like before, a term-document matrix using the bag-of-words vectorizer was created but the features we used to represent the bugs were text in the component and keywords column of the dataset.

The resulting term-document matrix was reduced to a lower dimension and represented in a KD-Tree structure. As the dimensions of this matrix compared to the previous one is small a higher reduction of just 50 features maintaining 70% of the variance in the data was possible. The querying of words gave us expected results.
Term-Document Matrix: 892 words - 9937 bugs
Sparsity: 99.9%

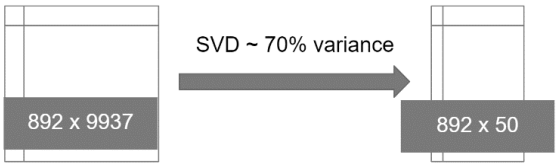

Fig. 10. Dimensionality Reduction using Component + Keywords

Query: security
Actual: process, sandboxing, enterprise, information, caps, risk
Expected: process, sandboxing, enterprise, information
Actual Records: Enterprise Process Sandboxing (35 bugs), Enterprise: Information Security (20 bugs)

And this leads us to believe that the representation of the bugs using only Component and Keyword columns are better than using the entire Bug Description

V. Deliverable

A key expectation of this project was to build a plug-and-play tool for all our models that can adapt to any dataset. Lots of effort has been put into building a configurable, user friendly, simple classifier pipeline P&W can use themselves on their own data, which P&W requested.

Using the tool, we can set data file that we would like to use and the specific columns of the data that needs to be used with the machine learning models.

Configurable options:

1. vectorizer: bag of words(bow), tfidf(tfidf)
2. classifier: decision tree(dt), knn(knn), svm(svm), decision-forest(df) naive-bayes(nb)
3. use_saved_data: (true or false) uses saved parsed csv data generated by csv_parser. true will save time if you previously ran learn_classifier or csv_parser on your data
4. test_split: *sets the train/test split*
5. save_model: (true or false) *saves model according to model_destination*
6. data_file: *path to parsed CSV of data*
7. model_destination: *saves model and held out test set on learned classifier*
8. visual_columns: *this is used to save the same held out test set for the visual data*

A visualization tool to view the results of the classifer was also built.

Configurable options:

1. model_train_data: the data the model you want to visualize was trained on. used to find the model
2. model_type: the model type of the model you want to visualize. of the form classifier_vectorizer
3. use_default_test_data (true or false): if true uses the held-out test set from when the classifier was initially learning otherwises uses the alternative_test_data_filename
4. alternative_test_data_filename: if you want to visualize your classifiers performance on a dataset other than the held-out test from where it was trained. the csv must have been parsed earlier by csv_parser. (learn_classifier uses csv_parser and saves parsed_csv)
5. columns: *columns of data to visualize. set this in the [DATA] section*
6. model_dir: *the directory the models are stored in*
7. alternate_test_data: *this is the path to the alternate_test_data*

Pratt and Whitney achieved 80% accuracy with our models.

VI. Related work

The baseline model to this project is a thesis paper [5] on Automatic Bug Triaging using Ticket Classification. It is an exact translation of the problem this project tries to solve. The thesis paper focuses on using supervised learning techniques using different datasets. They were able to achieve an accuracy of about 52%. Which is almost close to what we were able to achieve with the Mozilla dataset using various techniques.

The difference from the baseline paper to our models is that we have tried to access this problem from multiple directions. Using dimensionality reduction and studying the semantics of the word representation we identified that using a deep-learning approach such as LSTM would give us better representations of the text.

However, in translating model's cross domain and implementing the same model that we built for the Mozilla dataset on the Pratt and Whitney dataset the accuracy that the model achieved was close to 80%. This gives us the intuition that the representation of bugs in the Pratt and Whitney dataset is much better to the common Mozilla dataset.

The bi-directional LSTM approach to this problem is referred from the Deep Triage paper from IBM Research India [6]. The problem and dataset used by this research is identical to the problem that we try to solve.

VII. Conclusion and Future Work

From all the experiments above we learnt that ticket classification is more of a Natural Language Processing problem than it is a Machine Learning problem. The same models that performed at 60% accuracy on the Mozilla dataset gave about 80% accuracy with Pratt and Whitney's dataset which was more structured and well written. Unsupervised models like clustering and topic modelling did not give good results. Even feeding the clustering results into a classifier did not improve the accuracy.

For future work from a research standpoint, even though a deliverable is not feasible at the present instance, using clustering and grouping bugs into a cluster with multiple analysts can be tried.

The FastText vectorizer seems to provide good results with clustering over basic vectorizers, learning from this, we can use FastText vectorizer with our other classification models.

Matrix Factorization is another method to explore to understand the underlying structure of the bugs and the assignees.

From a deliverable standpoint, Pratt and Whitney would like to extend the tool into a GUI based tool that can predict an expert for an incoming ticket using the pre-trained model.

VIII. References

[1] Hughey, M. & Berry, M. Information Retrieval (2000) 2: 287.Improved Query Matching Using kd-Trees: A Latent Semantic Indexing Enhancement https://doi.org/10.1023/A:1009915010963

[2] Korenius T, Laurikkala J & Juhola M. On principal component analysis, cosine and Euclidean measures in information retrieval https://doi.org/10.1016/j.ins.2007.05.027

[3] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of Tricks for Efficient Text Classification https://arxiv.org/abs/1607.01759

[4] Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. Machine Learning: ECML-98 Lecture Notes in Computer Science, 137-142. doi:10.1007/bfb0026683

[5] https://is.muni.cz/th/374278/fi_m/thesis.pdf

[6] Senthil Mani, Anush Sankaran, Rahul Aralikatte, Deep Triage