Predicting labels for Github issues

Harsh Agrawal
North Carolina State University
hagrawa2@ncsu.edu

Siddu Madhure Jayanna
North Carolina State University
smadhur@ncsu.edu

ABSTRACT

This paper discusses a possibility of using machine learning to label issues with one of the default label provided by Github. Every issue opened on github has text associated with it and we believe we can extract some knowledge from that text and use it to classify the issue. This project proposes the idea of taking the text from the issues, extracting features from it using simple text processing and vectorization techniques and applying classification models on the processed data. We present the results of using classification techniques such as clustering, svm, Decision Trees, Neural networks on unstructured text data extracted from the title and body of Github issues. We also dive deeper into the concepts of using clustering to improve performance of traditional models. Initial experimental results demonstrate that this approach can produce fairly accurate results.

KEYWORDS

github, issue labelling, clustering, nlp, neural networks

ACM Reference Format:
Harsh Agrawal and Siddu Madhure Jayanna. 2019. Predicting labels for Github issues. In *ASE '19: Automated Software Engineering, Fall, 2019, Raleigh, NC*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the rise in popularity of git and open source projects, Github has recently shown meteoric rise. Many companies and groups are now using Github to host their projects so people can collaborate and work towards a common goal. Large number of people, now acting as collaborators in such projects, are generating uncountable issues daily. It has become a tedious task for the maintainers to sort through the issues daily and prioritize them efficiently.

ASE '19, Fall, 2019, Raleigh, NC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

To create a model which can predict what the issue contains, it is paramount that we first understand the features of the issue. Every issue opened on Github has a title and a body. Title is usually short between 8-20 words and summarizes the issue. Body describes the issue in detail and can contain questions, requests or sometimes code where the bug is present. We believe we can extract text from these 2 part and use it to train our model.

There are numerous amount of open-source software repositories with issues opened and closed over time. We can use these issues to learn how their labels are dependent on their title or the text contained in their body. Usually a user titles or describes an issue using natural language and a lot of information can be obtained by understanding the text and the keywords that uniquely categorize an issue. We can determine if an issue is referring to a bug, is a request for a new feature or just a random question that user has on the project.

Github provides 3 labels which can be used to categorize the issues. In this project, we explore automated machine lerning methods to classify an issue in one of these categories. We will see in later sections why we are using only 3 out of these 10 labels to train and get predictions from our model.

Natural Language processing is core to understanding of issue title and body of an issue. We are using ktext preprocessor to generate vectors, which can then be used in the various classification models. Pro-processing involves cleaning the data, tokenizing the text and then creating a vocabulary, which is then used to generate the vectors. This is being using 'ktext' text processor written by the developers at Github, specifically for tasks involving data extracted from git commits and issues.

2 RESEARCH METHODOLOGY

Data:

The data is collected by mining of open-source software projects on Github. GH-Archive or Github Arvhive is a collection of tremendous amount of data from Github (created by ingesting over 20 API calls provided by Github) stored in Google BigTable that can be retrieved using Google Big Query which is SQL interface. We extracted all the issues that have the following properties :

(1) Have a label
(2) Opened between 2017 and 2019
(3) With title length between 3 to 50 words

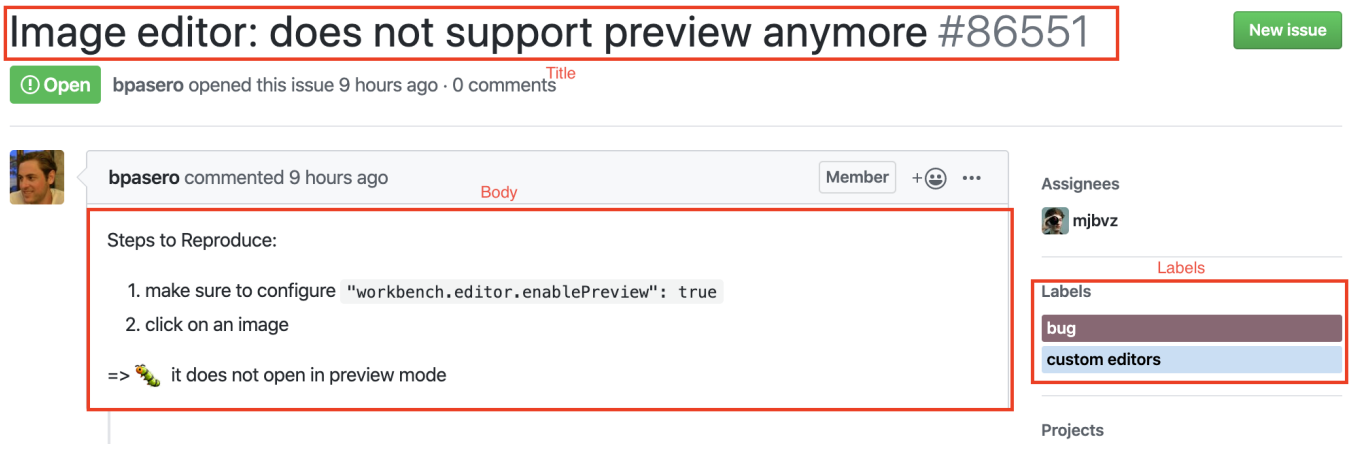

Figure 1: Typical example of an issue opened by a user in a Github repository

(4) With body length between 6 to 1000 words
(5) Are not duplicate of any issue in the same repository (label = duplicate)

The data is in JSON format and contains the following fields:

(1) url
(2) repo
(3) title
(4) body
(5) labels

We ended up with over 2.1 million rows of data. Original distribution of classes in the data is shown in figure 2.

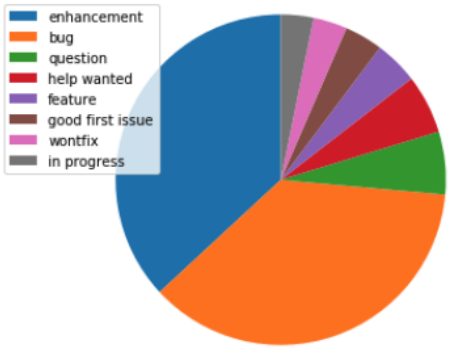

Figure 2: Initial distribution of classes

Praeto chart of all the issues and their labels is shown in figure 3.

On performing exploratory data analysis we found that most of the issues fall under one of the three category: Bug, Enhancement(feature request) or Question. There are many labels which can be clubbed together for the purpose of classification eg. 'feature', 'enhancement' and 'feature_request'.

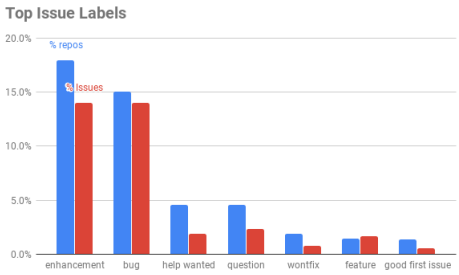

Figure 3: Praeto chart of all the labels

We also found that there are labels such as 'good first issue', 'wontfix', 'duplicate' which are not, in any way, related to the text of the issue and are arbitary.

To simplify our experiment, we decided to ignore all such labels and group all the closely related labels under one label. In the end, we ended up with our data being divided under just 3 labels mentioned above. We divided the resulting data using stratified sampling into two sets containing 80% and 20% data respectively for training and testing purpose.

Text Pre-processing:

Before we could feed our data into any of the algorithms, there was a need to pre-process it. As mentioned earlier we used 'ktext' pre-processor to process the data and generate vectors. 'ktext' performs the common pre-processing steps accociated with machine learning / deep learning such as:

(1) Cleaning: Data in the body of an issue sometimes contains a lot of code, which is of no interest to us, as it is not a feature that plays a important significance in the outcome of the classification. Removing the code, phone numbers, emails, mentions from the data, helps reduce noise.

(2) Tokenization: Taking a saw string, eg. "This is a bug", we can tokenize it to an array of words such as ["this", "is", "a", "bug"]
(3) Creating Vocabulary: Mapping each unique token in our corpus to an integer value, so it can be later replaced with this value to generate the vectors. This is usually store in a dictionary. We used the top 5000 most common words used in the issues to generate our vocabulary. We then went ahead and replaced all the words with their integer value from the vocabulary or '0' if the word was absent.
(4) Truncation and padding: To maintain a constant dimensionality , we padded the smaller length vectors with zeros and truncated all the longer vectors. Once this was done we ended up with vectors with 110 dimensions.

A couple of important thing to note in these pre-processing step is that we did not remove stop words from our data and that we maintained the sequence of words in the original order. This was done because we had planned on using Sequence-To-Class classification model which learns from the sequence in which the words appear in the data.

t-SNE Visualization:

Now that we have pre-processed our data, we wanted to visualize it and see if there any clusters forming with clear boundaries. We stumbled upon t-SNE or T-distributed stochastic neighbor embedding, which is a machine learning algorithm for visualization. It is a nonlinear dimensionality reduction technique well-suited for embedding high dimensional data for visualization in a low-dimensional space of tow or three dimensions. Specifically, it models each high-dimensional object by a two or three dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are models by distant points with high probability.

On applying t-SNE visualization on our data, we ended up with the distribution shown in figure 4. From the distribution, we can clearly see that, although there is no clearly defined boundaries between the classes, all the bugs tend to lie on the bottom while the feature are mostly clustered at the top of the figure, with questions distributed evenly among the two.

Hierarchical Clustering:

Hierarchical clustering, also knows as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other

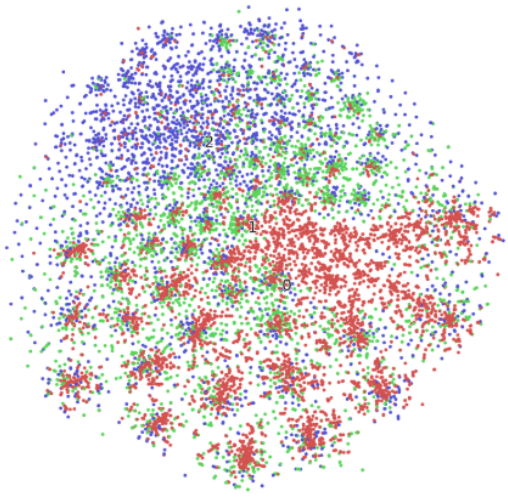

Figure 4: t-SNE visualization of classes

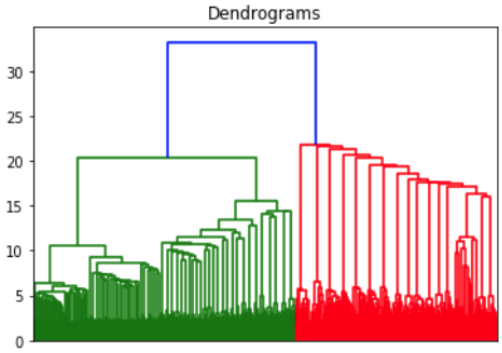

Figure 5: Hierarchical clustering of classes

On applying hierarchical clustering on the vectorized data, we got the results shown in figure 5. Here also, we can see that the data is majorly divided into 2 main clusters with questions cluster merged with the cluster containing bugs.

SVM:

Support Vector Machines or SVMs are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.SVM creates a maximum margin hyperplane in a high dimensional space which can be used to separate different classes of data. It uses a fixed number of data points called Support Vectors near the margin of the boundary to create the hyperplane. The equation of the plane is represented as follows:

$$w^T x + w_0 = 0$$

$$w^T x + w_0 >= 0 \quad for\ all\ i,\ such\ that \quad y_i = +1$$

$$w^T x + w_0 >= 0 \quad for\ all\ i,\ such\ that \quad y_i = -1$$

$$Together, \quad y_i(w^T x + w_0) >= 0$$

The points on the margins (support vectors), satisfy the equation:

$$y_i(w^T x + w_0) = 1$$

$$Width\ of\ margin: \quad 2/||w||_{L2}$$

The problem is the maximize this margin as well as satisfy the constraints:

$$y_i(w^T x + w_0) >= 0$$

Instead of solving multiple constraints SVM incorporates the constraints into the optimization by using a Lagrangian function given by the following equation:

$$J(w, w_0, \alpha) = ||w||^2/2 - \sum_{i=1}^{n} \alpha_i \left[y_i(w^T x + w_0) - 1 \right], \alpha_i \geq 0$$

Since not all data sets are linearly separable ,the original feature space can be mapped to a high dimension in which the data points are separable, the Kernel trick allows us to do just this with no extra cost.
Radial Basis Kernel/ Gaussian equation:

$$K(x, x^{'}) = exp\left[-1/2||x - x^{'}||^2 \right]$$

This is a type of kernel which we can use to map the features to a higher dimensional space.

$$maximize_\alpha \sum_i \alpha_i - 1/2 \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$K(x_i, x_j) = \phi(x_i).\phi(x_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$c \geq \alpha_i \geq 0$$

Where ϕ is a feature in high dimensional space, alpha are the Lagrangian coefficients.

We chose SVM because of its above mentioned capabilities and its kernel tricks. Also, because SVM enables classification of linearly unseparable data.

Decision Tree:

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

We used decision tree because it is useful with categorical attributes, is easy to construct, is able to formulate rules for classification and provides relatively high accuracy

Neural Networks:

A deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers between the input and output layers. The model is an assembly of inter-connected nodes and weighted links and the output node sums up each of its input value according to the weights of its links. It then passes this summation through an activation function and reports the output to the next connected node.The neural network adjusts its weights in order to predict the outputs correctly for the given training data.

For Deep Learning strategy, we decided to go with Sequence classification which is predictive modeling problem where you have some sequence of inputs over space or time and the task is to predict a category for the sequence. What makes this problem difficult is that sequences can vary in length, be comprised of very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols. We took care of all these problems in our pre-processing steps by limiting the number of words in the vocabulary and also truncating/padding our data so we have all the sequences of same length.

Sequence classification can be done using LSTM model. LSTM stands for long short term memory and is an artificial recurrent neural network architecture. Unlike standard feedforward neural networks, LSTM has feedback connections.

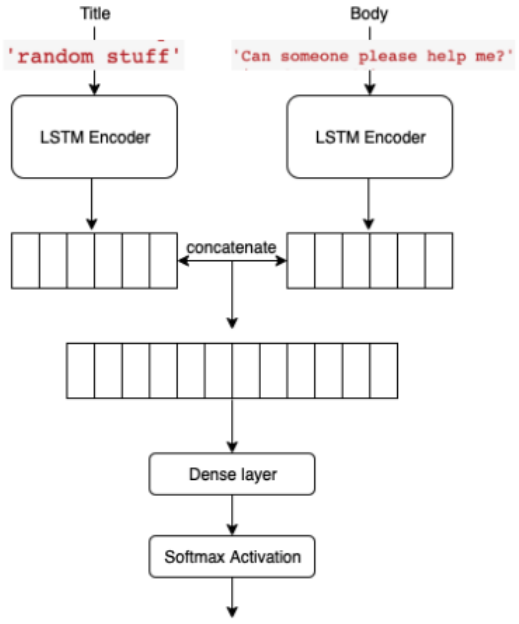

Figure 6: Neural network architecture

As shown in the above figure, We ended up using a very simple architecture for our project because of the huge size

of data and limitations of resources in training a very deep neural network. Our model takes two inputs: the issue title and body. Some of the parameters used during the training of the model are specified below:

(1) Optimizer: Adam
(2) Learning Rate: 0.001
(3) Loss: Sparse categorical cross entropy
(4) Metrics: Accuracy (optimize for acc)
(5) Activation function: softmax

3 IMPLEMENTATION STRATEGY

Initial Approach:

The initial approach was to train 3 different classification models (DNN, SVM and Decision Tree) and evaluate their performace.

Extended Approach:

The final approach was to experiment and see if there is any scope to improve the performance of classification models using unsupervised techniques like K-Means clustering. Also, since our data set had 100 plus dimensions, this seemed like a perfect candidate to experiment with dimensionality reduction techniques like PCA. Following are the experiments we carried out:

- **Approach 1:** Using clustering alone to build a new 1-D feature set and use that to train the classification model
- **Approach 2:** Using PCA to transform the data to a lower dimensional space and use that new dataset to train the classification model
- **Approach 3:** Using the results of clustering as a new feature(column) and use it with the existing columns to train classification model

4 EVALUATION STRATEGY

We tried to keep the evaluation strategy simple as the aim of this project is to predict a label for the issue by looking at the its title and body. Therefor, we will just be using accuracy and precision to gauge the performance of our models. This model is n-class classifier with n being 3. Hence, we can define metrics based on accuracy by the use of confusion matrix. The definition of these metrics are below:

- Accuracy: Defined as the proportion of true results, both true positives and true negatives, among the total number of cases examined.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: Defined as the number of true labels as a fraction of total truly predicted labels. it says, number of true positives of all the positive predicted labels. Basically it indicates of all positives that was predicted how many of them were actually positive. For precision score, the higher the better.

$$Precision = \frac{TP}{TP + FP}$$

5 HYPER PARAMETER TUNING STRATEGY

GridSearchCV Approach: This approach lets user combine an estimator with a grid search preamble to tune hyperparameters. The method picks the optimal parameter from the grid search and uses it with the estimator selected by the user.

Sample set of hyper parameters used for Decision Tree:

- Criterion: ['gini', 'entropy']
- MinSamplesSplit: range (150,450,30)
- MaxDepth: range (4,55,3)

Sample set of hyper parameters used for SVM:

- Kernel: ['rbf', 'linear']
- Gamma: [1e-3, 1e-4]
- C: [1, 10, 100]

6 RESULTS: INITIAL APPROACH

Decision Tree:

Test accuracy of 59.04 and train accuracy of 63.21

	precision	recall	f1-score
0	0.59	0.64	0.61
1	0.60	0.67	0.63
2	0.40	0.01	0.03

Figure 7: Results - Baseline Decision Tree

Hyperparameters picked for the best model:

(1) Best score for training data: 0.59
(2) Best max depth: 37
(3) Best criterion: Gini
(4) Best min samples split: 420

Confusion Matrix:

Decision Tree

	B	E/F	Q
B	14930	7455	29
E/F	7602	15018	67
Q	2605	2188	106

SVM:

SVM gave a test accuracy of 54.31.

	precision	recall	f1-score
0	0.54	0.54	0.54
1	0.55	0.66	0.60
2	0.50	0.00	0.00

Figure 8: Results - Baseline SVM

Hyperparameters picked for the best model:

(1) Best score for training data: 0.54
(2) Best C: 1
(3) Best kernel: rbf
(4) Best gamma: auto-deprecated

Neural Network:

This base model yielded the best result with an accuracy of over 0.8 for bugs and features and an accuracy of around 0.42 for question. We can attribute this to imbalance in the training data and non-consistent vectors for questions. See Figure 9.

Summary:

- Clearly, LSTM Neural Network outperformed the other 2 models
- Performance of Decision tree model was better than SVM model

7 RESULTS: EXTENDED APPROACH

Approach 1 Results: Decision Tree

Train and test accuracy of 51 percent (Fig 10)

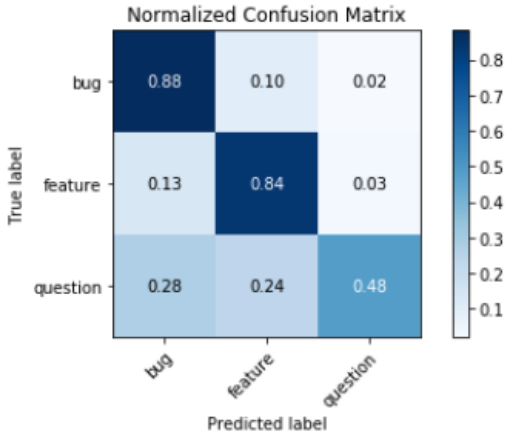

Figure 9: Neural network confusion matrix

	precision	recall	f1-score
0	0.50	0.53	0.51
1	0.53	0.61	0.57
2	0.00	0.00	0.00

Figure 10: Result - Extended Decision Tree

Summary:

- This was a great idea to reduce the computation power as we could replace 100 plus dimensions data with just one column and use that to train Decision Tree classifier
- Poor model: The data we have is a vectorized text of 100 plus dimensions and one feature can't just capture all the variances in the data set
- Clustering techniques alone can't drive the classification models to yield good results

Approach 2 Results: Decision Tree

Train accuracy of 58.32 and test accuracy of 53.07 (Fig 11)

	precision	recall	f1-score
0	0.52	0.57	0.54
1	0.54	0.61	0.57
2	0.13	0.00	0.00

Figure 11: Results - Extended Decision Tree 2

Hyperparameters picked for the best model:

(1) Best score for training data: 0.53
(2) Best max depth: 16
(3) Best criterion: Gini
(4) Best min samples split: 420

Confusion Matrix:

Decision Tree

	B	E/F	Q
B	62948	48275	29
E/F	44339	69694	108
Q	13707	10745	39

Summary:

- PCA constructs orthogonal - mutually uncorrelated - linear combinations that (successively) explains as much captures as much variance as possible
- Close enough results: We reduced our dimensions from 120 to 70 and 95 separately and analyzed their results. The performance of both PCA models were almost same and 5-10 percent less efficient compared to the original one

Approach 3 Results: Decision Tree

Test accuracy of 61.01 and train accuracy of 65.10 (Fig 12)

	precision	recall	f1-score
0	0.60	0.67	0.64
1	0.61	0.68	0.65
2	0.49	0.03	0.06

Figure 12: Results - Extended Decision Tree 3

Confusion Matrix:

Decision Tree

	B	E/F	Q
B	14800	7502	67
E/F	7510	15159	103
Q	2582	2175	155

ROC curves:

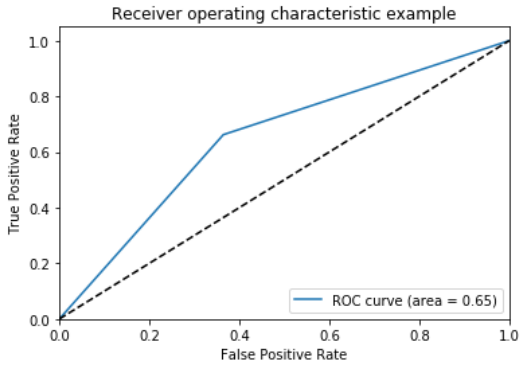

Figure 13: Class - BUG

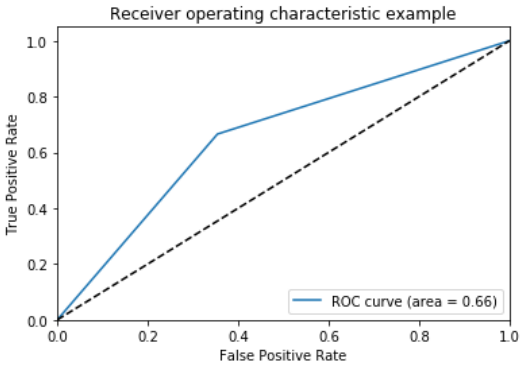

Figure 14: Class - Feature

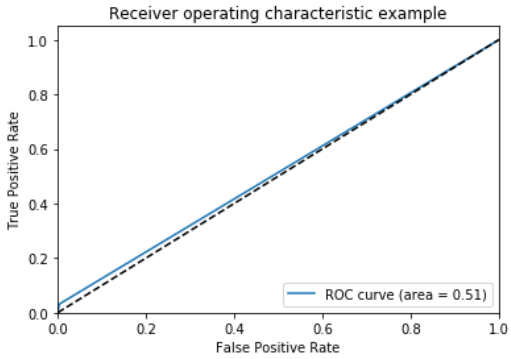

Figure 15: Class - Question

Hyperparameters picked for the best model:

(1) Best score for training data: 0.61
(2) Best max depth: 25
(3) Best criterion: Gini
(4) Best min samples split: 420

Precision-Recall curves:

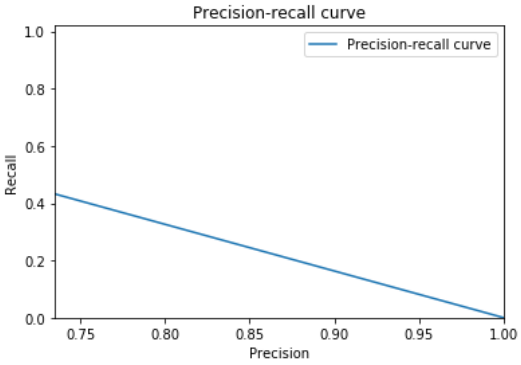

Figure 16: Class - BUG

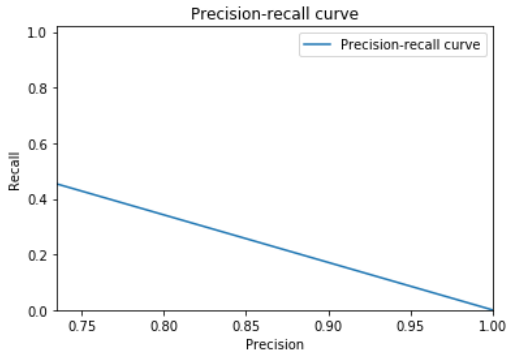

Figure 17: Class - Feature

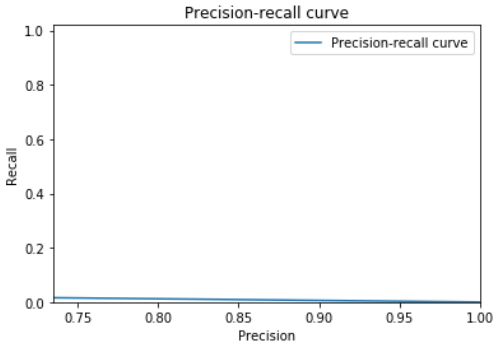

Figure 18: Class - Question

Summary:

- Slightly improved performance: Clustering apart from being an unsupervised machine learning can also be used to create clusters as features to improve classification models
- Although it was just 1 new feature added to the existing 120 features(dimensions), this 1 strong feature column generated using k-means clustering (trained on the actual data) could improve the overall performance by a small margin
- In general, this strategy can significantly improve the performance of a plain classifier if the data we are dealing with has few features (less than 5). Ideally, this strategy is not very suitable for classification tasks that deal with NLP (high-dimensional data)

8 KEY TAKEAWAYS

- Clustering apart from being an unsupervised machine learning can also be used to create clusters as features to improve classification models.
- Inconsistency of data plays a big role in model generation
- Large quantities of data cannot not guarantee a good model
- Selecting the right type of data and hyper parameters is important

REFERENCES

[1] https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
[2] https://towardsdatascience.com/how-to-create-data-products-that-are-magical-using-sequence-to-sequence-models-703f86a231f8
[3] https://www.gharchive.org/
[4] https://towardsdatascience.com/plotting-text-and-image-vectors-using-t-sne-d0e43e55d89
[5] https://www.kaggle.com/pranathichunduru/svm-for-multiclass-classification/notebook
[6] https://towardsdatascience.com/kmeans-clustering-for-classification-74b992405d0a