Multi-class Multi-tag Classifier System for StackOverflow Questions

Team name: Dr. AI
Team Members:
Tong Xu, 10124271
Michael Lasby, 10037047
Yunying Zhang, 30001970

Summary of Contributions

Data Collection

Coding

Writeup

Notebook Links

Preprocessing

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5573750848688710/4394517619816310/3899583605155350/latest.html

Tag2- Combined

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5573750848688710/1695999107813846/1229053072156622/latest.html

Tag2- existing

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5573750848688710/1695999107813913/1229053072156622/latest.html

Tag3 - Original Data

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5573750848688710/1695999107813979/3899583605155350/latest.html

Tag3 - Combined Data

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5573750848688710/1695999107814056/3899583605155350/latest.html

Abstract

Context

StackOverflow website is the largest online community for developers to learn, share their programming knowledge. A tag is a word or phrase that describes the topic of the question. Tags are able to categorize questions into different categories, which makes it more convenient for users' search. In StackOverflow, one question can have multiple tags.

Objective

In our project, we used data from StackOverflow website and predicted the tags assigned to questions. The task is multi-class, multi-tag, which means a question can be assigned to different topics and can also have several tags. The results obtained by this classification scheme are discussed, by analysing certain score metrics of the classifier system.

Method

The dataset is pre-processed, and CountVectorizer and VectorAssember is used for feature engineering. The data is splitted into training and testing sets, and hyperparameter tuning is applied to tune models. The data is then fitted to each model for discussion of the results.

Result

The dataset contained two textual features, Body and Title, which were preprocessed using traditional preprocessing and “lazy” preprocessing. We selected three classifier models for our work: Logistic Regression, Naive Bayes, and Random Forest. We evaluated the performance of these three models using a sample of about 6000 questions. From these test runs, we found that Logistic Regression and Random Forest classifiers provided better performance compared to the Naive Bayes classifier. In addition, traditional preprocessing led to improved model performance. Hence, we proceeded with traditional processing and used Logistic Regression and Random Forest models.

Due to limited processing capacity of databricks community edition and with the permission of Dr. Uddin, we sampled about 100,000 questions from the original and combined (original and new data) dataset. The model was trained to predict 500 of the most common tags for a tag position. We achieved a weighted F1 score of 0.47, 0.25, and 0.20 respectively for tag 1, 2 and 3. Logistic Regression resulted in the best performance for tag 1 and 3, while random forest performed best on tag 2. It is worth noting that for random forest classifiers, additional model enhancement could be attained by increasing the tree depth and number of trees, but we were limited by the processing capacity of our local machines. Furthermore, our model performance was impeded by insufficient data, which again was caused by a limitation of the processing power. Stratified sampling was used to preserve the class proportions, however, the less frequent tags have insufficient data for model training. For instance, the least common tag in tag 1 only has 6 records in total for the training and test dataset.

Conclusion

We successfully trained a multi-class, multi-tag classification system with three different classification models. The best performing models achieved weighted F1 scores of 0.468, 0.247,0.201 for tag positions 1,2, and 3 respectively. We found that models performed best when a small amount of textual processing was completed. We also found that a large amount of useful data was encoded in textual components that would normally be discarded as part of a natural language processing pipeline. Based on our findings, it appears that tokens traditionally classified as “noise” may in fact be significant to models attempting to classify documents from a technical source.

Introduction

Our project was proposed to predict the tags assigned to a set of questions from the StackOverflow website, based on the findings of Gonzalez et al. which used the data provided by the website www.kaggle.com for their contest *Identify Keywords and Tags from Millions of Text Questions* [1]. A training set of more than 4.2 million questions was obtained, and we used stratified sampling to obtain 100,000 dataset per tag. The dataset was used to develop and validate the classification model including Logistic Regression and Random Forest. We re-used the original data and added new data in our analysis to compare the performance results of the models. The dataset was pre-processed and we used CountVectorizer and VectorAssembler for feature engineering.We trained the dataset using different models, and the workbooks reference above showed how our machine learning (ML) models can be used to summarize our dataset and data similar to our analyzed dataset.

a) Motivation

In machine learning, multiclass classification is the problem of classifying instances into one of three or more classes. Multi-class problems have a richer structure than binary classification problems. Compared with binary class classification, multi-class classification models have broader use in real life, as the examples can be assigned to more than two classes.[6]

b) Background

In this section, we presented the major concepts and techniques upon which this study was founded.

2.1 StackOverflow website

Stack Overflow is a question and answer site for professional and enthusiast programmers, and it is a broadly used community for programmers.

It features questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and each question has tags assigned that shows the categories of questions.[2]

2.2 Tags for StackOverflow questions

A tag is a word or phrase that describes the topic of the question. Tags are a means of connecting experts with questions they will be able to answer by sorting questions into

specific, well-defined categories. Tags can also be used to help an user to identify questions that are interesting or relevant. [3]

2.3 F1 score and weighted F1 score

In statistical analysis of binary classification, the **F-score** or **F-measure** is a measure of a model's performance when dealing with unbalanced data. The $\mathbf{F_1}$ score is the harmonic mean of the precision and recall. [4]

For multiclass classification problems, a weighted F1 score can be found which weights each class' F1 score by the number of times that class appears in the corpus. The weighted F1 score is calculated as:

Weighted F-measure $$F_w(\beta) = \frac{1}{N} \sum_{\ell \in L} F(\beta, \ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$$

Since we are performing multiclass classification for our project, weighted F1 score is used to report in this paper.

2.4 Machine Learning Classification

Classification is the process of predicting the class of given data points, it is a type of supervised learning in Machine Learning. Classes are sometimes called targets, labels or categories. Classification predicts the output variable (y), from mapping input variables (x). [5]

c) Result Summary

1. How was the new data labeled/collected?

The dataset was downloaded from the 2013 Kaggle competition Facebook Recruiting III - Keyword Extraction. The data set consisted 6,034,195 stack overflow questions but contained many duplicates. After removing duplicates,there were 4,206,314 questions remaining in the training dataset. Additionally we used **Query Stack Overflow from Stack Exchange Data Explorer** to extract 11,298 new stackoverflow questions that were posted after 2013.

2. How does the newly added data compare with the original data?

The newly added data was compared against the original dataset by evaluating the top ten tag occurrences for all tag positions as well as cosine similarity of tags for each position. Expectedly, the top 10 tags for the two dataset are very similar and are related to popular programming language topics. The cosine similarity was high for tag 1 (0.86325) but decreased as we moved towards the end tag positions and the tag became more specific. Tag 5 only has a cosine similarity of 0.12326.

3. How was the data preprocessed?

The dataset was filtered based on the top 500 tags for each tag position. We explored two preprocessing methods, a traditional preprocessing and a "lazy" preprocessing [1], but found better model performance with traditional preprocessing, which included the conversion to

lowercase letters, the removal of html, urls, and stop words, and the application of word stemmer. After preprocessing, we compared the original and combined datasets based on the top ten labels (tags) and tokens in the body and title features. The labels and tokens in the body and title were very similar between the original and combined dataset, since the original dataset was almost 2000 times the size of the new data. Furthermore, we found that the title text contained more useful tokens because the top tokens in the body feature contained a lot of numbers.

4. How do the models perform on the original data vs the new + original data?
Since our original dataset was very large (4,206,314 questions), the number of new data (11,298 questions) was relatively small compared to the existing data. In general, original data had similar performance compared to the combined dataset. The original dataset had slightly better performance for tag position 1 and 3, and the combined dataset had slightly better performance for tag position 2.

5. How does the performance of the models change based on the choice of hyperparameters?
Each model was tuned for each tag position and every version of the preprocessing pipeline. The results of these tuning runs is summarized below in the results section. In general, we found that the model performance is strongly dependent on the choice of hyperparameters. We found that a properly tuned model had much better weighted F1 scores than a naive or improperly tuned model. The best hyperparameters varied slightly between tag positions, but several trends were identified.

6. How are the misclassifications of the best performing model distributed?
In general, our models performed best on common tags (those which occur in a high frequency in the input dataset) compared to uncommon tags. Due to the number of classes considered, it was not possible to identify the “worst” performing tags since many classes would not appear in a given test set.

Results

1. How was the new data labeled/collected?

Approach

The original training data set was downloaded from the 2013 Kaggle competition Facebook Recruiting III - Keyword Extraction. The data set consisted of 6,034,195 stack overflow questions but contained many duplicates. After removing the duplicates, 4,206,314 questions remained in the training dataset. The data has three features: ID, Title, and Body.

The target column, Tags, showed the tags assigned to each question as space-delimited strings. Each question can have up to five tags assigned to it.

Additionally,we used **Query Stack Overflow from Stack Exchange Data Explorer** to extract 11,298 new stackoverflow questions that were posted after 2013.

Result

The new data were filtered to remove questions that did not have a title, body, or tag, or had a score less than 1. After filtering, 2,105 new questions were added to the combined dataset which contained 4,208,419 questions.

2. How does the newly added data compare with the original data?

Approach

For each tag position, we compared the top tags for the original and new data. We also used cosine similarity to assess the tag similarity between the original and new data.

Results

We counted the tag occurrences for the original and new data. The top 10 tags for the original data are C#, java, php, javascript, android, jquery, C++, python, iphone, and asp.net. The top 10 tags for the new data are javascript, java, python, C#, android, php, html, C++, ios, and jquery. As shown, the top 10 tags for the original and new data are very similar. The changes to the order of the top tags can be attributed to changing popularity of the programming languages.

Tag position 1, which was the most general tag, demonstrated high cosine similarity. The cosine similarity decreased as we moved toward the bottom tag positions the tag became more specific. The cosine similarity for the 5 tag positions are reported below:

Cosine Similarity for tag position 1 is 0.86325
Cosine Similarity for tag position 2 is 0.74038
Cosine Similarity for tag position 3 is 0.57295
Cosine Similarity for tag position 4 is 0.32632
Cosine Similarity for tag position 5 is 0.12326

3. How was the data preprocessed?

Approach

We trained 3 classifier models using the first three tag positions of the stackoverflow questions. For each classifier, we found the 500 most common tags for each tag position and filtered the dataset to keep only questions with tags in the 500 most common tags. Due to the limited processing capacity of Databrick community edition and with permission from Dr. Uddin, we sampled the datasets using stratified sampling and trained the classifiers with about 100,000 records.

Following the paper's approach, we cleaned the textual features using two processing methods: a traditional preprocessing and a "lazy" ptraditional preprocessing consists of a series of text cleaning tasks including the conversion to lowercase letters, the removal of html, urls, and stop words, and the application of a word stemmer. Additional cleaning tasks were explored, including the removal of punctuations (not including +, #) and numbers, as well as tokens with less than 3 characters, but resulted in reduced model performance. In lazy preprocessing, instead of word stemming and stop words removal, we utilized the maxDF parameter of CountVectorizer to remove the non-informative tokens.

We performed a series of test runs using small datasets with about 6000 random samples to compare the performance of traditional versus lazy preprocessing. Out of the three classifier models tested, which include Naive Bayes, Logistic Regression, and Random Forest; all three models performed better on datasets preprocessed with traditional preprocessing. Thus, we proceeded with traditional preprocessing.

Results

The size of the original dataset is almost 2000 times bigger than the newly added data. As a result, the original and combined dataset have very similar features and labels, as shown in Tables 1-3. Review of the top ten tokens for the Body and Title text showed that title text have more meaningful tokens as many of the top title tokens were direct reference to tags. In addition, many of the top body tokens were numbers which are non-informative for tag classifications.

Table 1: Top Ten Tags for Tag Position 1, 2 and 3

	Tag 1		Tag 2		Tag 3	
	Original	Combined	Original	Combined	Original	Combined
1	c#	android	arrays	regex	html	html
2	java	python	mysql	eclipse	ios	ios
3	php	ruby-on-rails	css	arrays	css	ajax

4	javascript	java	regex	multithreading	ajax	css
5	android	php	multithreading	css	xcode	xcode
6	c++	jquery	linq	jquery	database	database
7	python	javascript	templates	.htaccess	json	json
8	iphone	mysql	xml	xml	query	query
9	jquery	c++	eclipse	facebook	forms	asp.net
10	ruby-on-rails	iphone	jquery	mysql	cocoa-touch	jquery

Table 2: Top Ten Body Tokens for Tag Position 1, 2 and 3

	Tag 1		Tag 2		Tag 3	
	Original	Combined	Original	Combined	Original	Combined
1	0	0	0	0	0	0
2	1	1	use	use	use	use
3	use	use	1	1	1	1
4	2	2	new	new	class	new
5	new	file	class	get	new	class
6	get	new	get	class	id	get
7	file	get	file	file	get	id
8	class	class	id	id	name	name
9	id	name	2	name	file	file
10	code	code	name	2	valu	code

Table 2: Top Ten Title Tokens for Tag Position 1, 2 and 3

	Tag 1		Tag 2		Tag 3	
	Original	Comrepro cessing [1]. The bined	Original	Combined	Original	Combined
1	use	use	use	use	use	use

2	file	file	file	file	c	c
3	c	c	c	c	file	file
4	get	get	get	get	jqueri	jqueri
5	jqueri	jqueri	jqueri	jqueri	get	get
6	php	php	php	php	php	php
7	error	error	net	error	net	data
8	android	android	data	net	data	net
9	work	valu	error	data	java	object
10	data	window	valu	android	error	work

4. How do the models perform on the original data vs the new + original data?

Approach

As discussed above, we obtained the original dataset from Kaggle competition website, and added new data by using Stack Overflow query. We applied stratified sampling, pre-processed data and applied feature engineering to train data under the same models.

Results

Tag position1:

Model	Logistic Regression		Random Forest	
Data	new+existing	existing	new+existing	existing
Weighted F1	0.468	0.473	0.465	0.463
Precision	0.557	0.563	0.721	0.732
Recall	0.417	0.422	0.404	0.408

Tag position2:

Model	Logistic Regression		Random Forest	
Data	new+existing	existing	new+existing	existing
Weighted F1	0.187	0.157	0.247	0.246
Precision	0.195	0.172	0.599	0.575
Recall	0.186	0.145	0.186	0.183

Tag position3:

Model	Logistic Regression		Random Forest	
Data	new+existing	existing	new+existing	existing
Weighted F1	0.201	0.202	0.158	0.160
Precision	0.288	0.290	0.492	0.485
Recall	0.174	0.175	0.117	0.118

The results for combined (new & existing) data and existing data results are shown on the tables above. The results for the combined dataset and existing dataset were very similar. For tag position 1 and 3, the existing dataset has slightly better results compared with combined dataset. For tag2, the combined dataset performed slightly better than the existing dataset. For tag position 3, Logistic Regression model has slightly higher performance than the combined dataset, but the difference is very small (0.001 for F1, 0.002 for Precision and 0.001 for recall).

5. How does the performance of the models change based on the choice of hyperparameters?

Approach

For each of the three models deployed in our classification task, we first tuned their hyperparameters using ML Lib's TrainValidationSplit class. Instances of this class are

instantiated with one of the models and a set of hyperparameters to test. The results of these trials is a set of hyperparameters which provide the best F1 score for the input data and selected model type.

Hyperparameter tuning was conducted on each tag position for each classifier. While some minor differences were found between different tag positions, several trends were identified that appeared to be true regardless of tag position. The following section discusses the results of the hyperparameter tuning for each model type. Please note that the metrics and F1 scores reported below are for the Combined dataset (new + existing).

Results

Please note that our team found a bug in PySpark's TrainValidationSplit class for Multi-Class classification problems. The TrainValidationSplit class requires a MultiClassClassificationEvaluator which can be set to optimized on one of several metrics, for example "f1", "accuracy", and "weightedF1Measure". However, if this is left as "f1" or "weightedF1Measure", the evaluator will optimize the results for a single class only (specifically, tag class index = 1.0), rather than the aggregate totals. We found that by setting the metric to "accuracy", we found the best hyperparameters to achieve the optimum weighted F1 Score. Therefore, please note that there are some discrepancies noted between the tabular results and plotted results below.

The plots have been included to provide a visual reference of the general trends observed; however, the plots were generated based on the output of hyperparameter tuning with only 10,000 records rather than the full 100,000 set reported in the tables below.

Naive Bayes:

For this model, only the smoothing parameter can be tuned. Smoothing is used to correct the model when it encounters an unknown token in the test set that did not appear in the training corpus. In our case, increasing smoothing to 100 improved the performance of the model. Further smoothing degraded performance.

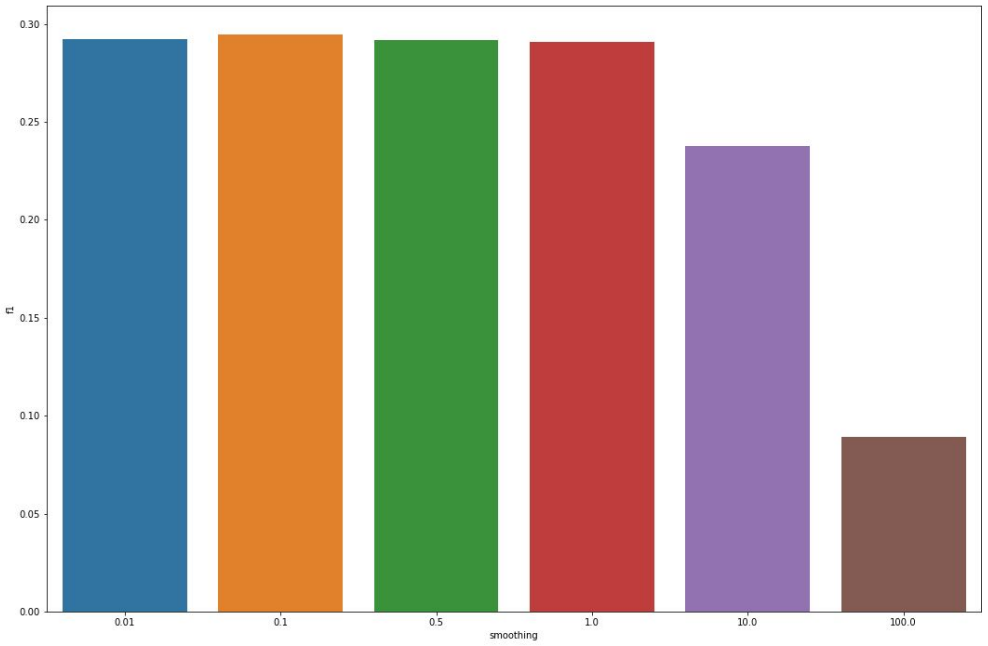
A multinomial Naive Bayes model was selected from the onset as it is the preferred model for document classification. Multinomial Naive Bayes considers each observation as a document and each feature represents a term whose value is the frequency of the term. Please note that the Naive Bayes model was not tested on Tags 2 or 3 as it was the worst performing model on tag 1.

Table X: Naive Bayes Hyperparameter Tuning

Model & Tag Position	Smoothing	Weighted F1 Score
NB - LP - Tag 1	100	0.391
NB - TP - Tag 1	100	0.431
NB - TP - Tag 2	N/A	N/A

NB - TP - Tag 3	N/A	N/A

Figure X: Naive Bayes Hyperparameter Tuning



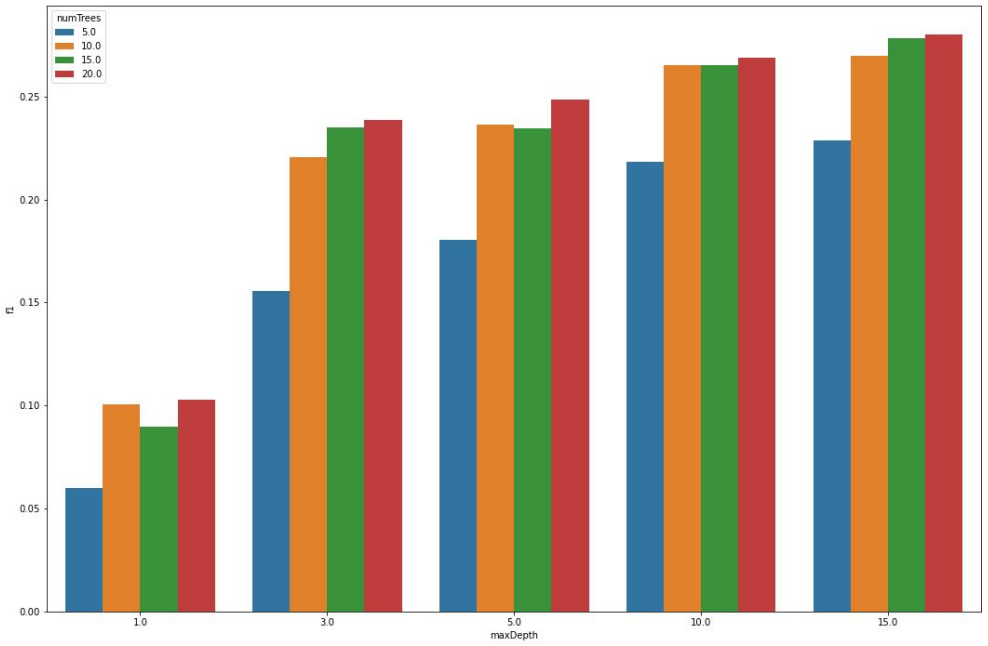
Random Forest:

For the Random Forest Classifier, we tuned the MaxDepth and NumTrees parameters. By increasing the number of trees and their depths, we found that the performance of the model greatly improved. Our parameter search found the best values at the edge of our parameter range; however, we had insufficient memory available to test higher values for these hyper parameters. Based on the trends observed, it appeared that adding additional trees and depth would further increase the performance of the model. The hyperparameters noted below were the best performing model that our team was able to fit without running out of memory.

Table X: Random Forest Hyperparameter Tuning

Model & Tag Position	MaxDepth	NumTrees	Weighted F1 Score
RF - LP - Tag 1	15	15	0.408
RF - TP - Tag 1	20	15	0.466
RF - TP - Tag 2	15	15	0.247
RF - TP - Tag 3	15	20	0.160

Figure X: Random Forest Hyperparameter Tuning



Logistic Regression:

For the Logistic Regression models, we tuned the regularization, elastic net, and fit intercept hyperparameters. The regularization and elastic net parameters are both intended to regularize the data to help our models generalize better and avoid overfitting.

For tags 1 and 3, we found that increasing regularization beyond a small amount decreased model performance. A small amount of regularization (0.01) provided the best results in most cases. The fit intercept specifies a constant bias to apply to the decision function, in our case leaving this parameter as false provided the best performance, likely due the lack of any systemic bias in the underlying corpus.

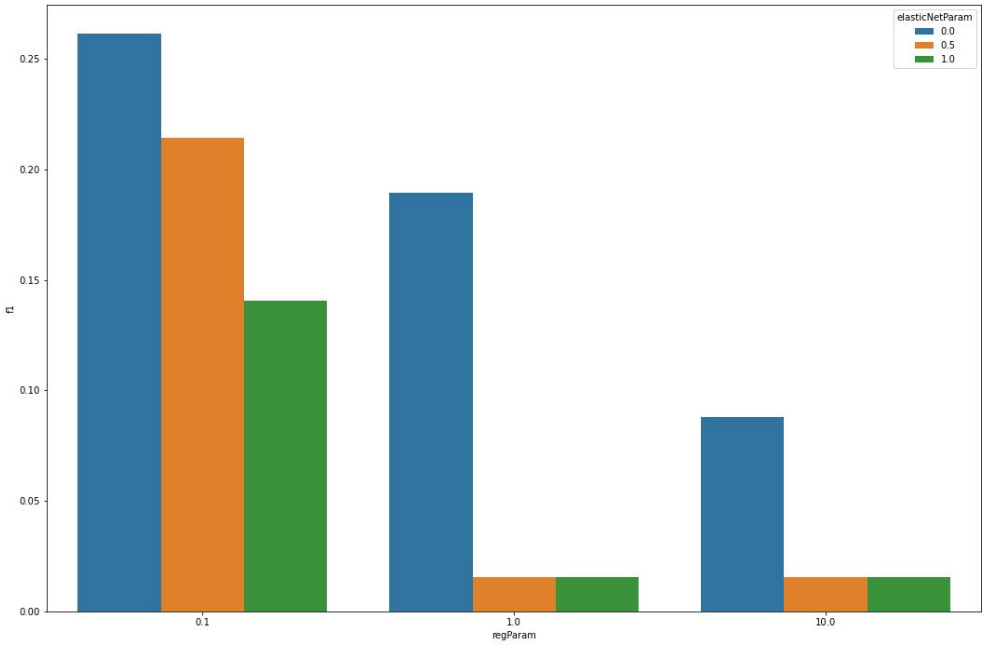
We noted that tag 2 had slightly different hyperparameters after tuning than tag 1 and 3. See the below for a summary of these differences.

Table X: Logistic Regression Hyperparameter Tuning

Model & Tag Position	regParam	elasticNetParam	fitIntercept	Weighted F1 Score
LR - LP - Tag 1	0.01	0.0	False	0.456
LR - TP - Tag 1	0.01	0.0	False	0.468

LR - TP - Tag 2	0.00	0.0	True	0.187
LR - TP - Tag 3	0.01	0.0	False	0.201

Figure X: Logistic Regression Hyperparameter Tuning



6. How are the misclassifications of the best performing model distributed?

Approach

We found that the Logistic Regression classifier with traditional pre-processing on the combined (existing + new) dataset was the best performing model on tag positions 1 and 3; however, the best performing model on Tag 2 was the Random Forest classifier. We took the top 10 and bottom 10 tags ranked by F1 score and reviewed the data for any significant trends that may explain the models performance.

Results

Table X: Best Performing Models

Tag Position	Best Performing Model	Model's Weighted F1 Score

1	Logistic Regression	0.468
2	Random Forest	0.247
3	Logistic Regression	0.201

Generally, we found that the common tags were the easiest to classify. We observed that a tag's frequency within the original dataset was correlated to the model's F1 score for that tag. See the below figures:

Figure X: Tag 1 F1 Score vs Tag Frequency

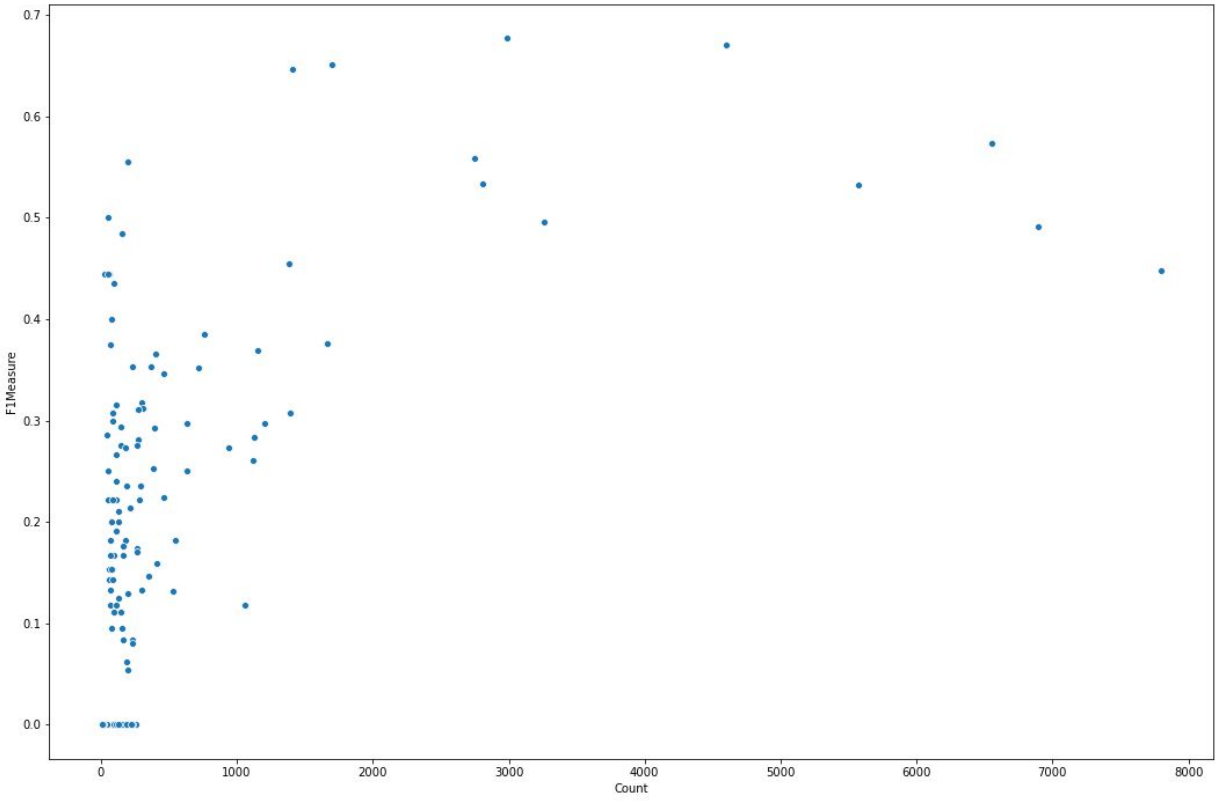


Figure X: Tag 2 F1 Score vs Tag Frequency

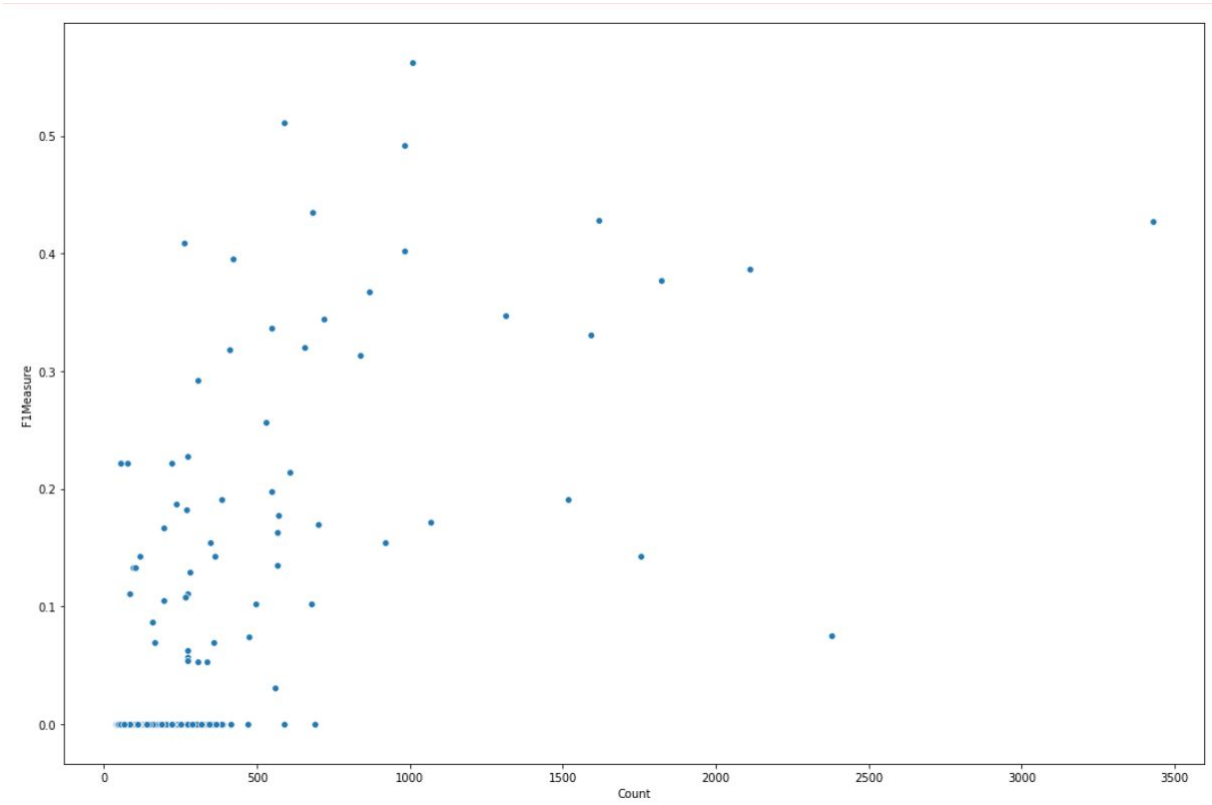
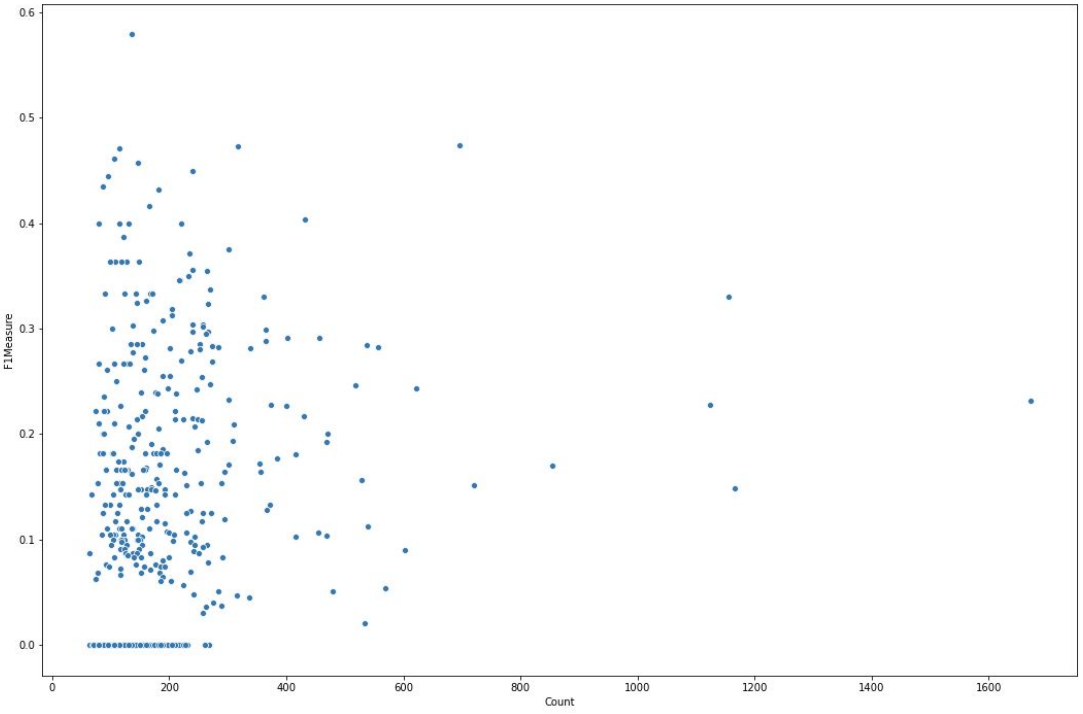


Figure X: Tag 3 F1 Score vs Tag Frequency



In the above figures, we can see that tags with less than a few hundred occurrences have very low F1 scores. Generally, as the tag frequency increases, the model's ability to accurately predict the tag increases. The training / test data was sampled from the full dataset in a stratified manner to preserve the class imbalances; however, it appears that we have insufficient records of uncommon tags in our training dataset to provide the classifiers enough data to predict on.

For each of the three tag positions (1,2,3), we determined the top ten and bottom ten performing tags by F1 score. These values were also compared with the tag frequencies. See the figures below.

Tag 1:

Figure X: Tag 1 - Best Tags vs. F1 Score

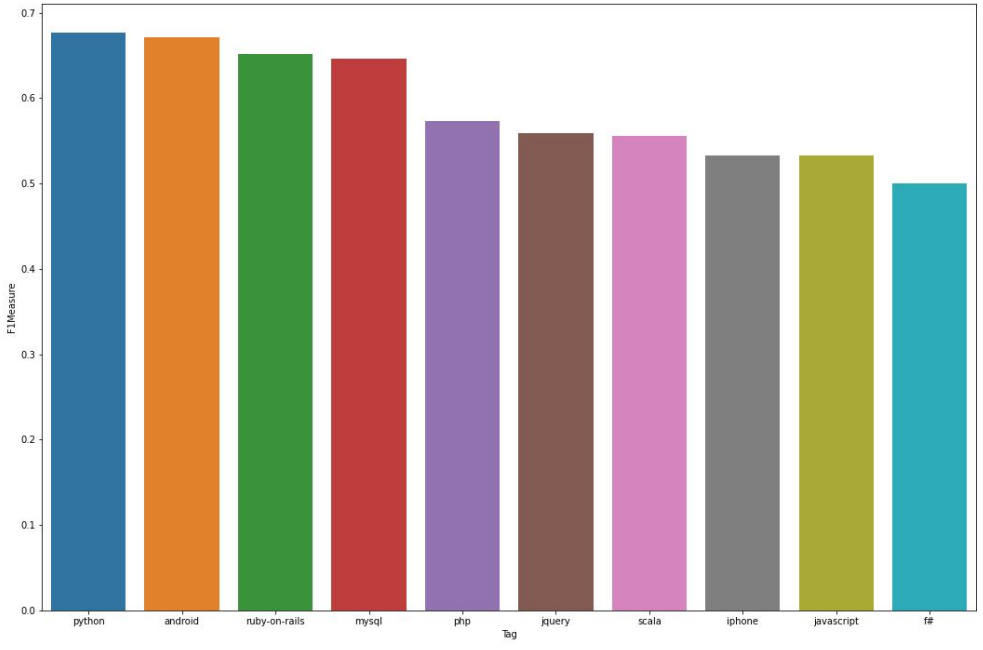


Figure X: Tag 1 - Best Tags vs. Count

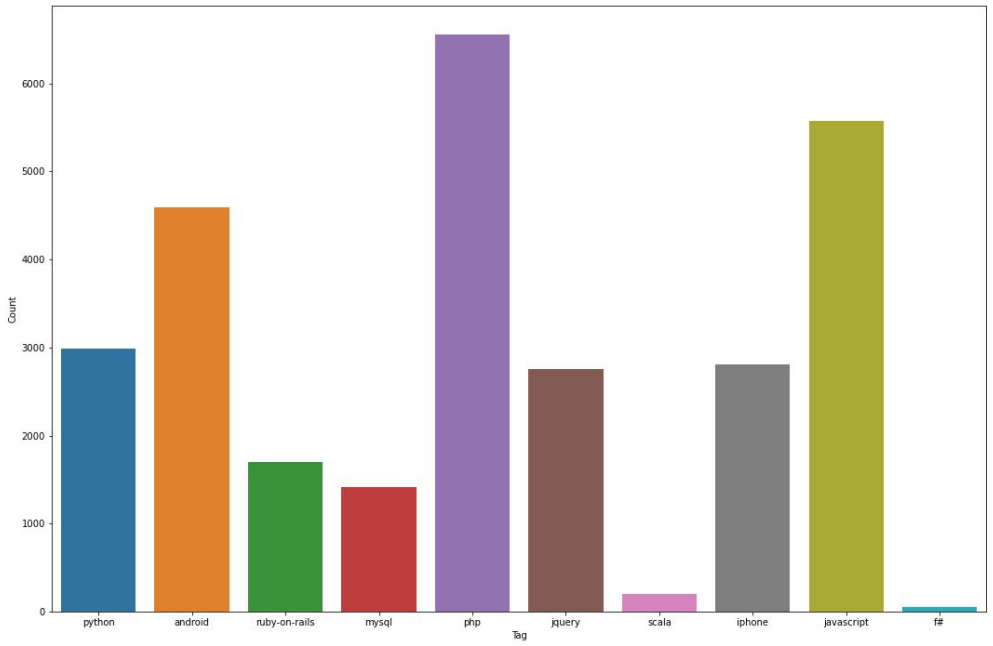
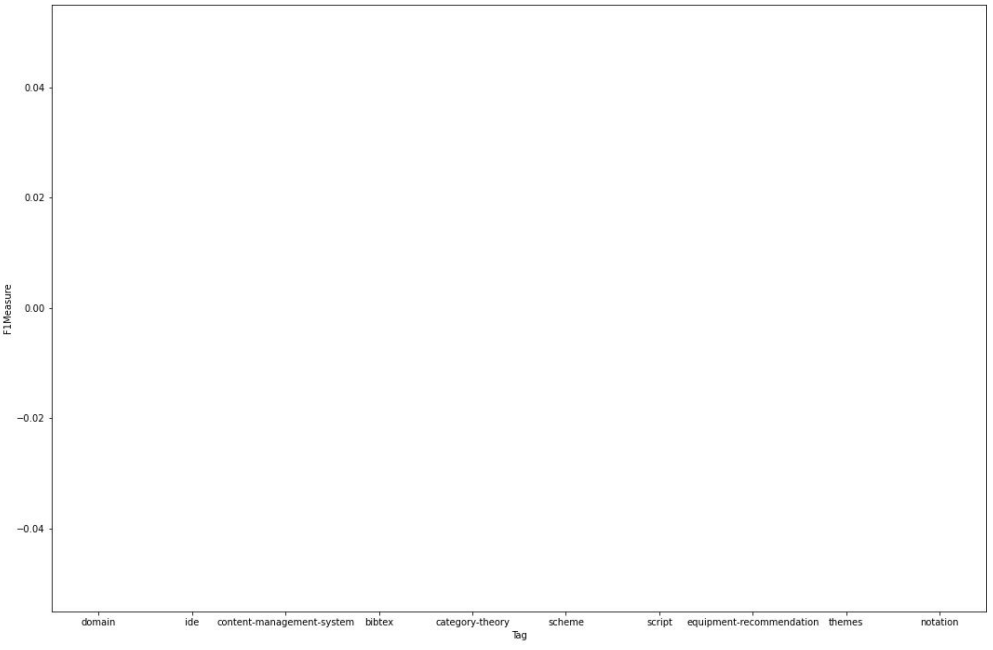
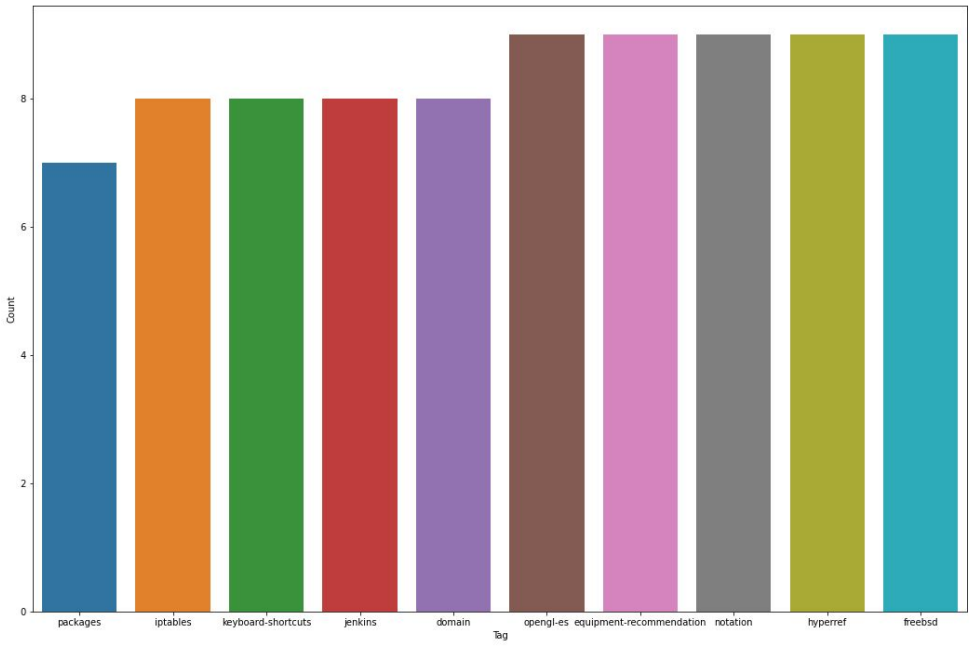


Figure X: Tag 1 - Worst Tags vs. F1 Score



*Note: : Note each tag received 0.0 as an F1 score

Figure X: Tag 1 - Worst Tags vs. Count



Tag 2:

Figure X: Tag 2 - Best Tags vs. F1 Score

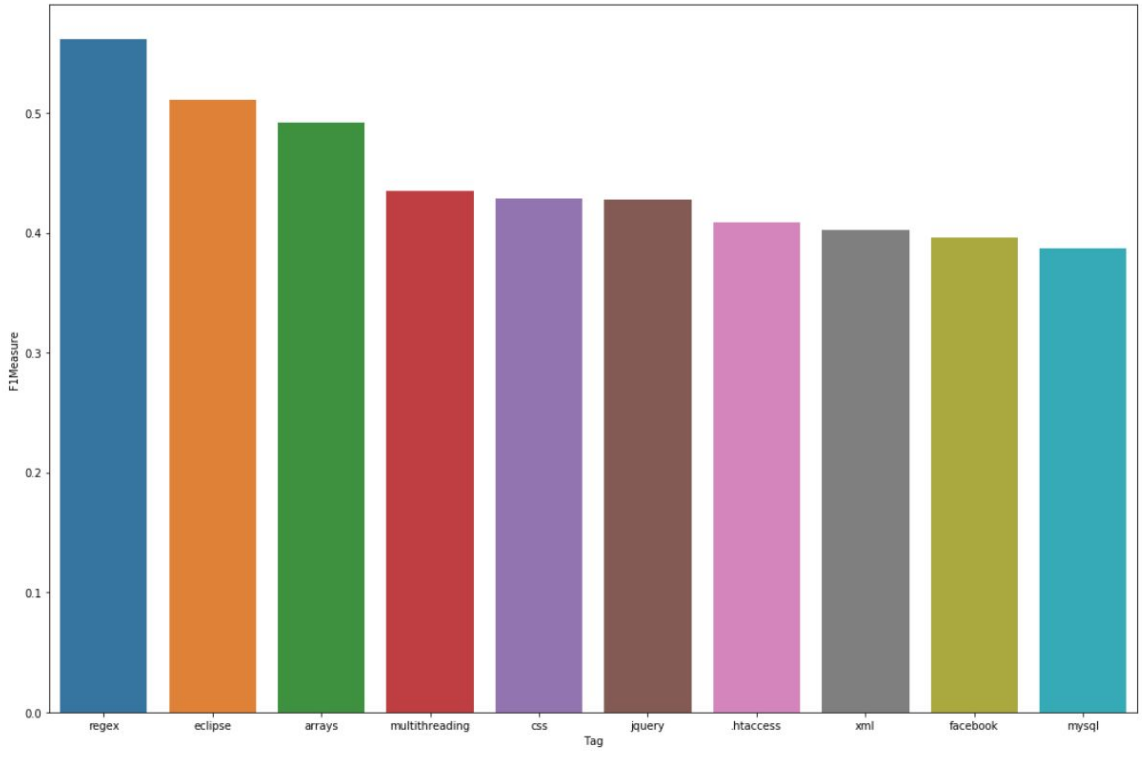


Figure X: Tag 2 - Best Tags vs. Count

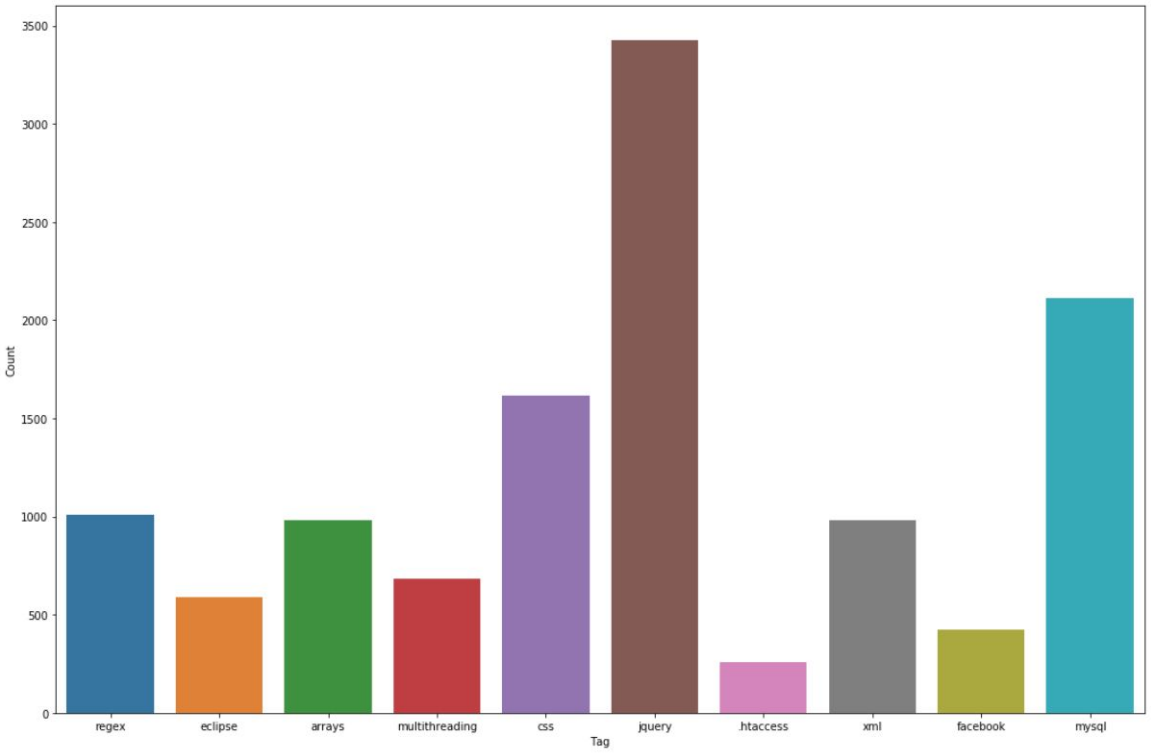
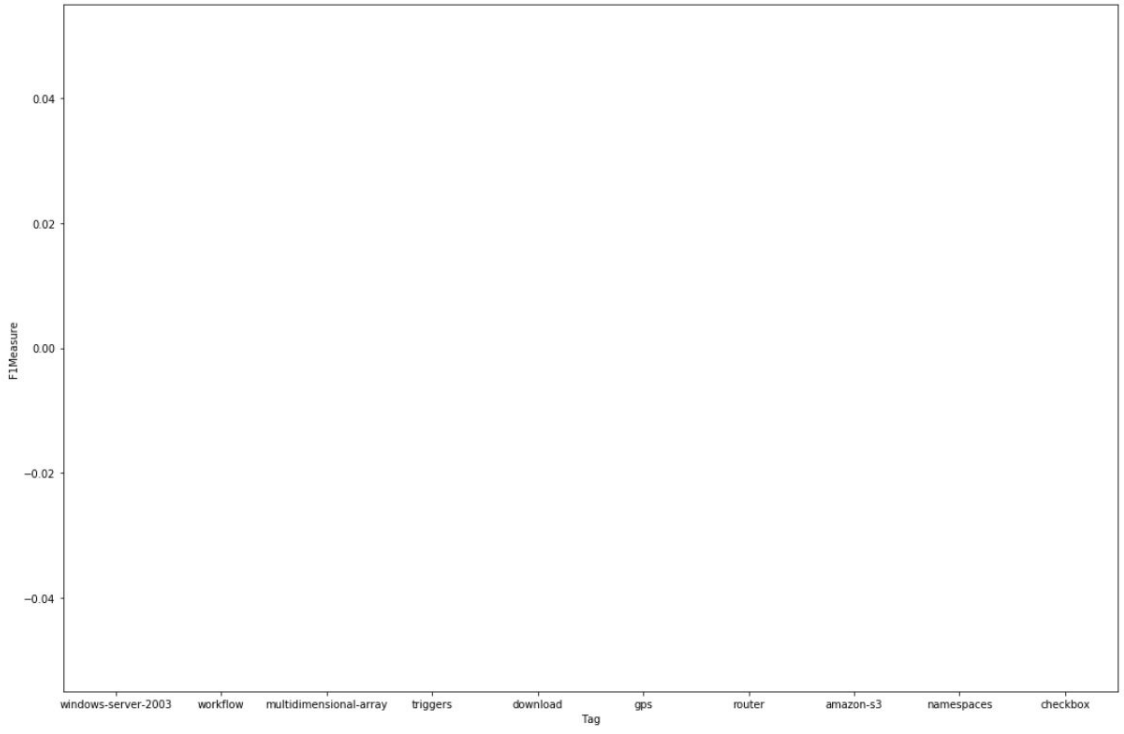


Figure X: Tag 2 - Worst Tags vs. F1 Score



*Note each tag received 0.0 as an F1 score

Figure X: Tag 2 - Worst Tags vs. Count

Tag 3:

Figure X: Tag 3 - Best Tags vs. F1 Score

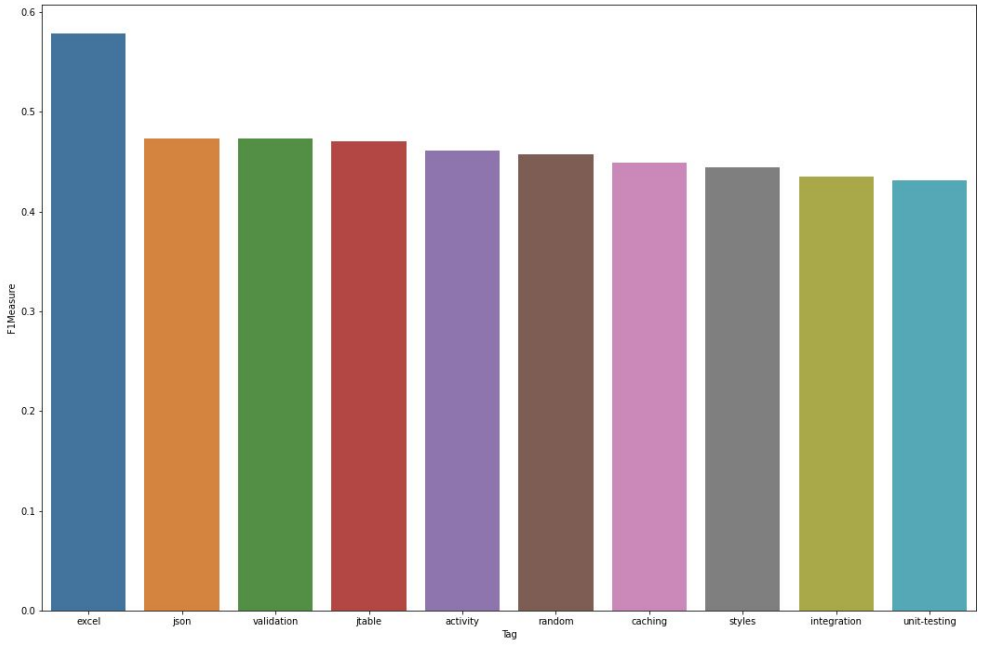


Figure X: Tag 3 - Best Tags vs. Count

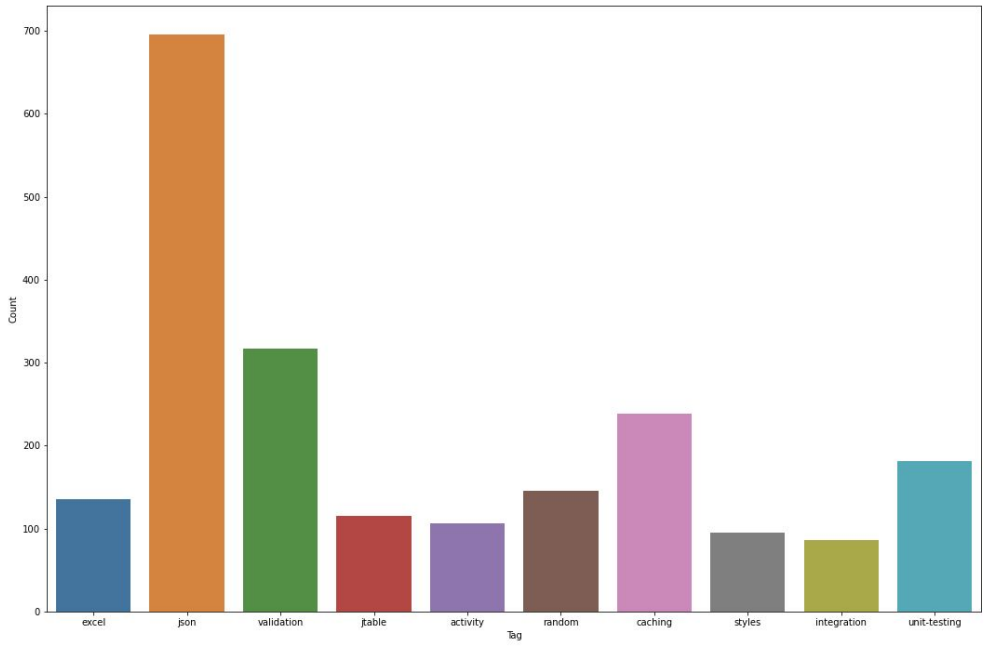


Figure X: Tag 3 - Worst Tags vs. F1 Score

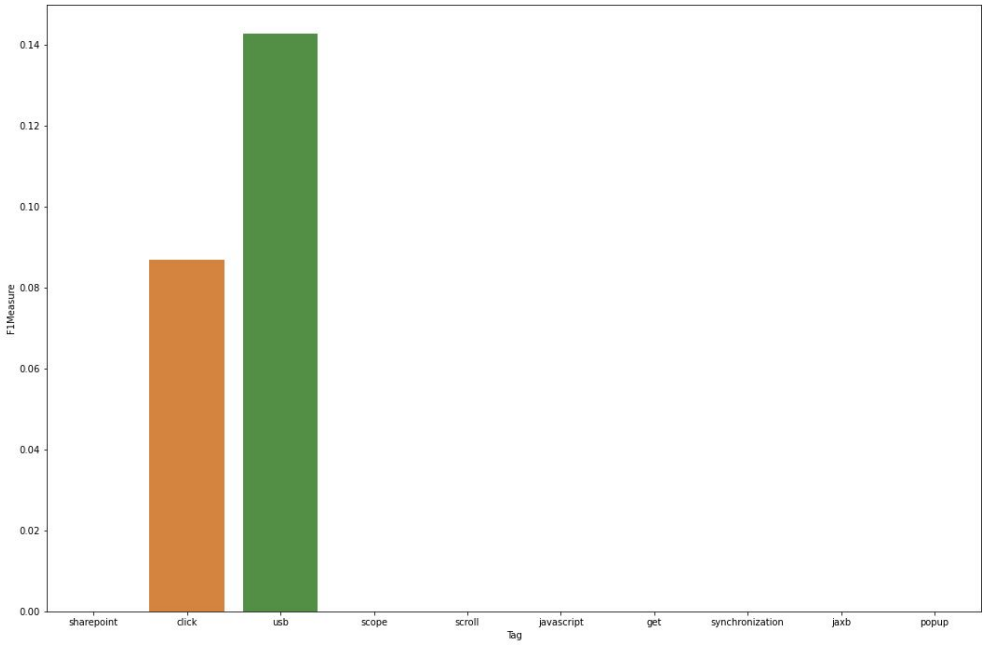
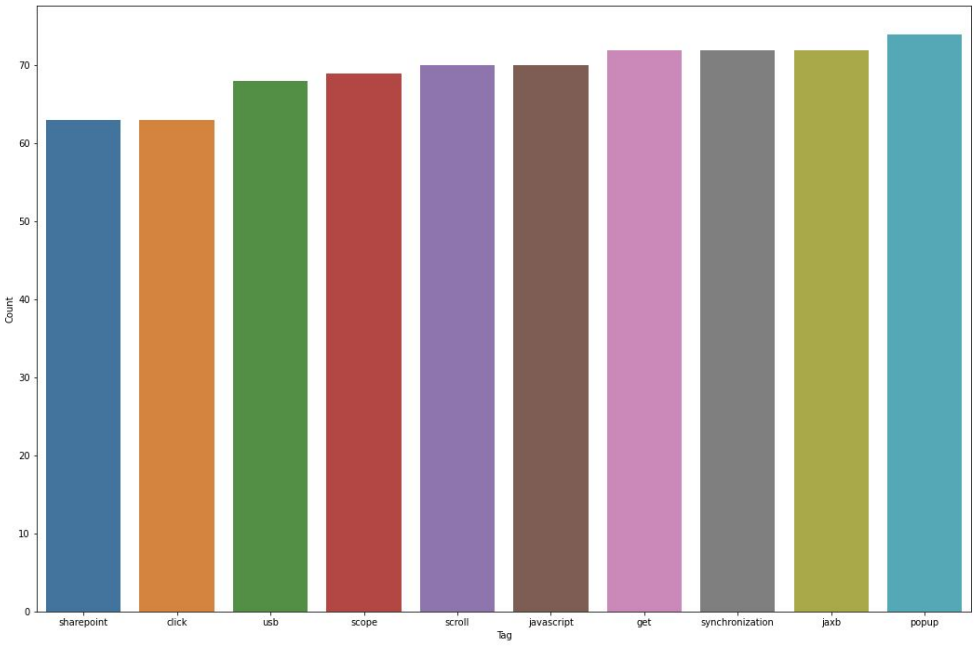


Figure X: Tag 3 - Worst Tags vs. Count



Misclassifications were challenging to identify since rare tags only may only occur once or even zero times in a given test set, even with 10,000 to 20,000 records in the test set. As a result, the “bottom ten” tags are somewhat arbitrary. However, we can see from the above that the worst performing tags are generally less common in the training data than the top ten tags. For tag 1 for example, any tags with more than 2000 records in the training dataset had a F1 score greater than 0.35. The model did not perform as well on tags with fewer records in the training dataset.

For tag 2 and 3, we noted that none of the tags had more than 2000 records in the training dataframe. As we can expect based on our tag 1 and 2 results,

Discussions

The following paragraphs identify some potential real-world applications for our models and discuss our results in more detail.

Real-world Scenarios

Real-word use case scenarios that our model can be used for:

Scenario #1, by Yunying Zhang:
Our model can be used to predict tags when an user is trying to post a question on StackOverflow. After the user enters the title and body of a question, rather than manually entering the tags, the tags can be auto-populated by using our model for prediction.

Scenario #2, by Mike Lasby:
Document classification is an important task in record keeping. Many institutions and businesses are required to maintain accurate records of documents that may be stored using an in-house classification system. Our multi-class multi-tag classification models could be used to reduce the likelihood of misclassifying documents and also automate the classification process when attempting to add many new records to the file store.

Scenario #3, by Tong Xu:
This work approaches the text document classification problem, which has many useful real world applications including automatic spam detection. The spam filter is trained to catch many of the suspicious words and phrases associated with spam emails, therefore protecting the user from malicious content.

Pre-Processing Findings

We explored traditional and lazy preprocessing methods. In both method we applied that standard text cleaning tasks which included changing the text to lowercase letters, and removing urls and html tags.The difference between the two methods was that we applied word stemmer and removed stop words in the traditional preprocessing; in lazy preprocessing, the redundant or non-informative tokens were removed by setting the maxDF parameter of CountVectorizer to 0.85. Models trained with traditional preprocessing outperformed lazy preprocessing for all three models that we explored, which included Logistic Regression, Random Forest, and Naive Bayes. The models' performance suggested that simply using the maxDF parameter in CountVectorizer is insufficient for removing all the stop words, which are redundant or non-informative features for the model. We explored additional text cleaning tasks, such as removing tokens with less than three characters, which resulted in slight reduction in model performance. This can be understood since C# is one of the most frequent tags and we could be removing direct reference to the tag. In addition, we explored the removal of punctuation (not including # and +) and number, which led to significant reduction in model performance. Upon closer look at the dataset, it can be seen that punctuations and numbers are important features for the model as many tags contain punctuations and/or numbers, for instance, ruby-on-rails-3, asp.net, and html5.

Model Selection Discussion:

Gonzalez et al used Naive Bayes and Support Vector Classifier in their work [1]. We attempted both classifiers; however, only Naive Bayes directly supported multiclass classification problems. For the Support Vector classifier, it was used in combination with a One-vs-Rest classifier, and we were unable to fit the model even when the model was run overnight (more than 12 hours). Hence, we proceeded with the Naive Bayes classifier, which is faster than the linear models and is good for very large datasets and high-dimensional data [8]. Additionally, we chose a linear model (Logistic Regression) and an ensemble classifier (Random Forest) for our work. Both models directly support multiclass classification. Linear models are typically used as a base model, and are good for very large datasets and high dimensional data [8]. Ensemble methods improve performance by combining multiple models, although it is not good for high-dimensional sparse data [8].

Multi-Class Challenges

Due to the long process time and hardware challenges discussed below, we could only train on 100,000 data records for each tag position. Since we were attempting to identify 500 classes, it was a challenge for us to have a limited number of data to train. The training set is imbalanced as some tags that occur in the test set may not exist in training data, leading to inaccurate prediction for the tags. Tags that are uncommon usually have very low accuracy. Below is an example of tag position 3, we see a low accuracy obtained from the bottom 10 tags that have the least accuracy:

```
display(bottom_10_accuracy)
```

	tag	correct_sum	count	accuracy
0	rss	0	13	0.0
1	qt	0	12	0.0
2	doctrine	0	14	0.0
3	jar	0	7	0.0
4	network-programming	0	9	0.0
5	map	0	15	0.0
6	charts	0	15	0.0
7	android-widget	0	9	0.0
8	keyboard	0	18	0.0
9	phpmyadmin	0	13	0.0

Hardware Challenges

The biggest challenge we faced was getting results out of this large dataset without access to a cluster or high performance computer. Databricks community edition cluster is actually only a single Driver node with 15 GB of memory. The driver node has 2 CPUs only. In comparison, our laptops have between 4-8 cores and 16 GB of memory. Further, the Databrick community edition cluster will automatically shut off after 2 hours, which is insufficient to run some of the more complex computations such as TrainValidationSplit hyperparameter tuning or running one-vs-rest algorithm for support vector classifiers as outlined above.

We managed to quickly use up 16 GB of memory when trying to run the ML Lib models on datasets larger than 100,000 records in Databricks. See below:

Fig X: Databricks Heap Issues

These out of memory issues would often occur after during a training or grid search task.

We also had challenges training some of the more complex models. For example, our hyperparameter tuning suggested that we should continue to test Random Forest models with depth greater than 20 and number of trees greater than 15; however, each time we tried to go beyond these hyperparameters, we would crash the interactive python kernel. See

below for an example of a laptop trying to fit random forest with numTree=20 and maxDepth = 20.

Fix X: Laptop Heap Issues

Access to a high performance computing solution is likely required to achieve reasonable scores in this classification task. We noted that the winner of the original Kaggle competition, username ‘Owen’, used a machine with 256 GB of ram to achieve his winning score of F1 = 0.814 [7].

Conclusions

We successfully used PySpark to train a multi-class, multi-tag classification system with three different classification models. We added over 2,000 new records to the existing dataset before tuning the models’ hyperparameters. The best performing models achieved weighted F1 scores of X, X, and X on tag positions 1, 2, and 3, respectively.

We found that a significant amount of useful data is encoded in textual components that would normally be discarded as part of a natural language processing pipeline. We found that the models performed best when only a small amount of textual processing was completed. Surprisingly, removing numbers and punctuation decreased the performance of our models. Future work should focus on what components of textual data is important to keep for classification of technical documents. Based on our findings, it appears likely that tokens traditionally classified as “noise” may in fact be significant to models attempting to classify documents from a technical source.

We were unable to directly follow the procedure outlined by the original authors due to a lack of available computing resources, however, we achieved comparable results for tag positions 1, 2, and 3.

Future researchers in this area should first ensure that a significant amount of computing power is available. Due to our hardware constraints, we were unable to achieve high performance on uncommon tags.

References

[1] J. Gonzalez, J. Romero, M. Guerrero, and F. Calderon. Multi-class Multi-tag Classifier System for StackOverflow Questions. In *2015 IEEE International Autumn Meeting on Power, Electronics, and Computing (ROPEC)*, pages 1-6, 2015.

[2]Stack Exchange Inc., "Who we are," [Online]. Available: https://stackoverflow.com/company. [Accessed 13 December 2020].

[3] “What are tags, and how should I use them?,” *Stackoverflow.com*. [Online]. Available: https://stackoverflow.com/help/tagging. [Accessed: 13-Dec-2020].

[4] Sasaki, Y. (2007). "The truth of the F-measure" (PDF).

[5] S. Asiri, “Machine learning classifiers - towards data science,” *Towards Data Science*, 11-Jun-2018. [Online]. Available: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623. [Accessed: 13-Dec-2020]

[6] V. Vural, “Multi-Class Classifiers and Their Underlying Shared Structure”.

[7] “Owen”. Unknown, “Share your Approach?”. [Online]. Accessed on Dec. 13, 2020. Available at: https://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction/discussion/6650.

[8] A. Muller and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O’Reilly, 2016.