

Building Large-scale Text Classifiers on Reddit Comments Data

Aditi Nair and Akash Shah

Machine Learning and Computational Statistics

New York University, Center for Data Science

May 13, 2016

Table of Contents

Abstract.....	2
1. Introduction.....	2
1.1. Problem Statement.....	2
1.2. Business Application.....	2
1.3. Evaluation Methodology.....	3
2. Data Preparation.....	3
2.1. Data Description.....	3
2.2. Sampling Method.....	3
2.3. Data Cleaning and Featurization.....	4
3. Building Single Layer Models.....	5
3.1. Baseline Model.....	5
3.2. Single-layer Methods.....	5
3.2.1. Tree-based Methods.....	5
3.2.2. Linear Classifiers.....	5
3.2.3. Naive-Bayes Classifier.....	6
3.2.4. Ensemble Methods.....	6
3.3. Single-layer Summary and Results.....	6
4. Building Two Layer Models.....	7
4.1. Building Subreddit Clusters.....	7
4.2. Designing an Algorithm for Soft Two-layer Classifiers.....	8
4.2.1. A Soft First-layer Classifier.....	8
4.2.2. Incorporating Probability Distributions from the First-layer Classifier.....	9
4.3. Building Two Layer Models.....	9
4.3.1. Clustering Parameters.....	9
4.3.2. Choosing Classifier Pairings.....	9
4.3.3. Two-layer Results and Analysis.....	10
4.4. Further Experiments with Two-layer Models.....	11
4.4.1. Analysis of LogReg-LogReg versus LogReg.....	11
4.4.2. Non-linear Transformations of First-layer Probability Scores.....	12
4.4.3. Manual Cluster Adjustment.....	13
5. Conclusion.....	15
5.1. Generalization Performance.....	15
5.2. Looking Forward.....	15
Sources and References.....	17

Abstract

Given Reddit comments from May 2015, we build several classifiers that attempt to classify the subreddit of a comment based on its text content. We envision a direct application of these classifiers as a subreddit recommendation and discovery tool. Accordingly, we evaluate the classifiers on Top-5 precision as well as runtime, and explore models using both single-layer and two-layer classifiers. Finally, we recommend a two-step soft classifier for this purpose that achieves 65.1% Top-5 precision on the test set and 8.6 minute runtime.

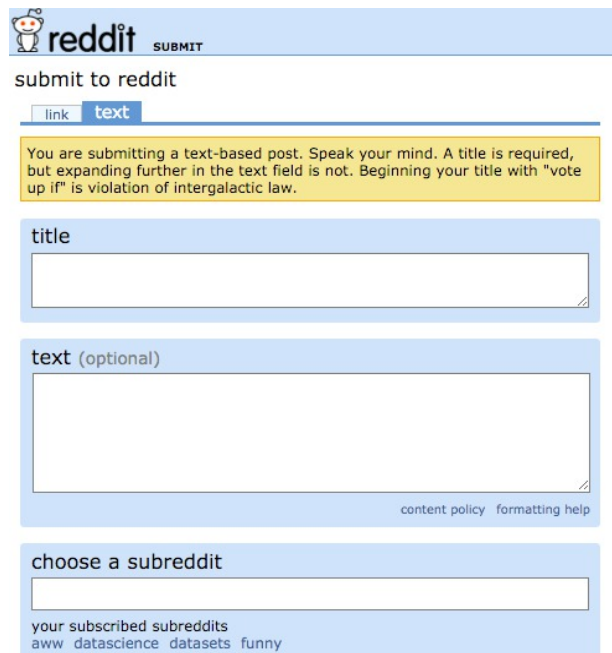
1. Introduction

1.1 Problem Statement

Reddit is a social news website where users can post text or hyperlinks. Reddit is organized into a set of nonoverlapping forums, each with a theme or topic of interest, called “subreddits”. In this project, given the text content of a comment posted on Reddit, we will predict the subreddit to which a comment was posted. We will treat this as a multi-class prediction problem where the classes are individual subreddits.

1.2 Business Application

When submitting a new text post, the user has to manually select the subreddit to which they want to post or select from a subreddit which they are already subscribed to, as shown below. If they are posting to a subreddit they are not subscribed to, the user has to manually search through the existing subreddits to find the one that matches the content of their post, which can be a daunting task since there are over 300,000 subreddits.



The image shows a screenshot of the Reddit submission interface. At the top, there is a blue header with the Reddit logo and the word "reddit" in lowercase, followed by "SUBMIT" in all caps. Below this, the text "submit to reddit" is displayed. There are two tabs: "link" and "text", with "text" being the active tab. A yellow warning box contains the text: "You are submitting a text-based post. Speak your mind. A title is required, but expanding further in the text field is not. Beginning your title with 'vote up if' is violation of intergalactic law." Below the warning box is a "title" label and an empty text input field. Underneath is a "text (optional)" label and a larger empty text area. At the bottom of the text area, there are links for "content policy" and "formatting help". Below the text area is a "choose a subreddit" label and a dropdown menu. At the very bottom, there is a section for "your subscribed subreddits" with a list of subreddits: "aww", "datascience", "datasets", and "funny".

Our vision for the business application of our model is a subreddit recommendation tool that suggests five subreddits for a new post based on the text the user has written. This tool will have the twofold advantage of simplifying the posting process for users as well as introducing users to new subreddits that they may be interested in.

1.3 Evaluation Methodology

Our evaluation methodology was shaped directly by our business application. Since we would like to suggest five subreddits to users based on their text input, we would prefer a model that contained the correct classification in this list of suggestions. Accordingly, we evaluated the success of different models using a top-5 precision metric in which a classification is considered correct if the list of suggestions contains the correct label according to our labelled dataset.

Since our intended application involves providing near-instantaneous suggestions, runtime was also an important factor in our evaluation of different models. Given two models with equal or similar top-5 precision, we generally preferred the faster model: while a model with slightly poorer top-5 precision over another does not present a significant detriment to our application, a model that requires hours to run will essentially render the application useless.

2. Data Preparation

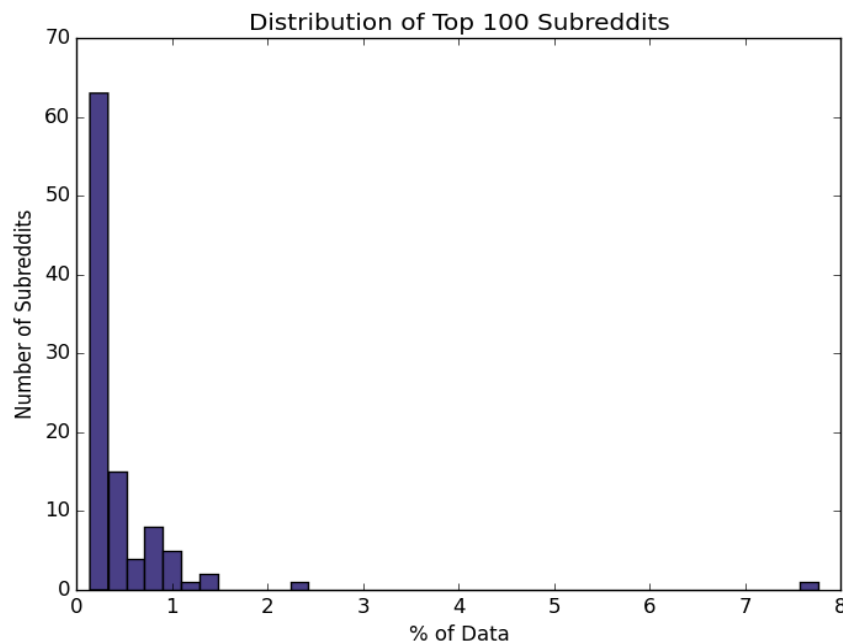
2.1 Data Description

Recently, Reddit released all of their comments data from May of 2015, an enormous dataset comprising of 1.7 billion comments (over 1TB of data). Our data source was Kaggle, which publicly hosted approximately 50 million comments from the Reddit comments data. The raw dataset contained many features, including the UTC created date, the score of the post (net number of upvotes and downvotes), the author, the body of text in the comment, and the subreddit name. However, we decided to use only the comment text as the input and the subreddit name as the label. This is consistent with our business application described above, as we wanted to be able to suggest subreddits to a user for a new post based only on the text provided. Among these 50 million comments, there were approximately 50,000 distinct subreddits. Given the staggering size of this dataset and the large number of classes, we decided it was necessary to sample an appropriate dataset from this source data.

2.2 Sampling Method

When choosing the size of our sampled dataset and the number of classes in it, we wanted to strike a balance between confronting some of the challenges encountered in a production setting, while still being able to run algorithms on our computers locally. As a result, we settled on sampling 500,000 comments from 100 subreddit classes.

As shown in the histogram below, the distribution of comments in the 100 most popular subreddits is very skewed, with most of the subreddits containing less than 1% of the data each, while a couple of the subreddits are many multiples larger. In fact, the most commented subreddit has over 50 times as many comments as the 100th most commented subreddit. We wanted to have enough instances from each subreddit to be able to predict each class accurately, but sampling each subreddit exactly equally might have skewed our results. Thus, we decided to sample half of the 500,000 comments proportionally to the class priors, while sampling the other half equally among each class. As we were sampling, we only sampled comments that had more than three words in the body of text in order to filter out comments that would have little to no signal.



For the purposes of testing various algorithms and tuning them, we did a 60-20-20 split of the data into a train set, validation set, and test set.

2.3 Data Cleaning and Featurization

Our data cleaning process first involved dropping non-ASCII characters from comments so that they could be processed by scikit-learn’s text processing libraries. Then, we removed all punctuation and special characters from the comments. Afterwards, the comments only consisted of alphanumeric text.

Initially, we hypothesized that there would be some common online language amongst all of the subreddits that wouldn’t be captured by a default stop word list, such as “lol”, “haha”, “btw”, etc. We counted the most common words in the corpus with the intention of creating a custom stop word list that would include Reddit-specific common words and Internet lingo.

However, our exploration showed that many of the most popular words were already contained in regular stop word lists. Furthermore, going far enough down the list of common words to include the types of words mentioned above would also have excluded words that were domain- and context-specific. As a result, we decided to use a default English stop words list instead of constructing our own.

Finally, we featurized our text data by using a TF-IDF representation of the data in scikit-learn. We initially tested implementations using and not using IDF weighting, as well as Laplace smoothing, and eventually settled on using IDF weighted, Laplace smoothed representations as the most robust featurization of our text data.

3. Building Single Layer Models

3.1 Baseline Model

Our baseline model was very simple - it guessed the 5 most popular subreddits every time. These 5 subreddits - AskReddit, leagueoflegends, nba, funny, and pics - made up 17% of our dataset. Thus, the top-5 precision of our baseline model was 17%.

3.2 Single-layer Methods

At the single-layer stage, our goal was to try a variety of classifiers and identify what types of algorithms performed well on the featurized data. Using both top-5 precision and runtime as evaluation metrics, we wanted to be able to generalize learnings from the results that would prove helpful in understanding the challenges of working with this data and give insight into what algorithms we should try at the two-layer stage.

3.2.1 Tree-based Methods

As a simple base classifier, we first tried a shallow decision tree of depth 2. The tree performed barely above random, with a top-5 precision of 17.5% on the validation set, and took 54 minutes to run. We then tried building a random forest classifier with default parameter values to see if it would perform better. However, the random forest did not finish running in 24 hours and so we terminated it. It seemed that decision trees had poor computational performance because of the high-dimensionality and sparsity of our data. As a last attempt at a tree-based classifier, we decided to use an XGBoost classifier. XGBoost is an external library consisting of a distributed gradient boosting tree algorithm that has been optimized in many ways, and is considered to be one of the most efficient out-of-the-box tree-based ensembles. While the XGBoost algorithm performed better than the previous tree-based methods, it still only achieved a top-5 precision of 52.7% with a runtime of 280 minutes.

3.2.2 Linear Classifiers

Linear classifiers were also explored, as they are commonly used in text classification. For the logistic regression model, we looked at two versions. The first was the multinomial logistic regression model, which builds a multinomial distribution over all of the classes in its modeling process. The other was the one-versus-rest logistic regression model, which essentially builds a model for each class, attempting to predict whether an instance is in that class or in the other remaining classes. We tried to run the multinomial logistic regression model on our dataset, but it was too computationally intensive and crashed. We empirically verified that the model ran on smaller datasets with significantly less than 100 classes, so we concluded that computing a multinomial distribution over 100 classes was too heavy of a load for our computers. The one-versus-rest logistic regression model had promising results however, as it achieved a top-5 precision of 64.3% on the validation set in a runtime of 11 minutes. We also tried an SVM classifier. However, the SVM classifier does not inherently provide probability estimates, so the probability parameter must be enabled. This causes the model to use Platt scaling and a computationally expensive cross-validation process. This was evident in our attempt, as we were forced to terminate the model after it did not finish running in 24 hours.

3.2.3 Naïve-Bayes Classifier

A multinomial Naïve-Bayes model was also tested due to its speed and generally decent performance on text applications. It achieved a top-5 precision of 51.6% with a runtime of 11 minutes.

3.2.4 Ensemble Methods

Since even non-ensemble classifiers took 10 - 50 minutes to run on the dataset, it was obvious that other ensemble methods such as AdaBoost would also have runtimes that were too high. Thus, they were not explored at the single-layer stage, but shelved for later exploration.

3.3 Single-layer Summary and Results

The following table summarizes the top-5 precisions on the validation set and the runtimes for the single-layer models that were tested:

Model	Train Top-5 Precision	Validation Top-5 Precision	Runtime
Baseline	17%	17%	0 min
LogReg OVR	77.3%	64.6%	11 min
LogReg Multi	-	-	∞
DT (depth=2)	17.5%	17.5%	54 min
RF	-	-	∞
XGBoost	57.5%	52.7%	280 min
Multi NB	60.5%	51.6%	11 min
SVM	-	-	∞

Ultimately, we saw the best performance in both runtime and top-5 precision with the one-versus-rest formulation of logistic regression. After fine-tuning the regularization parameter C of the model, we were able to improve the performance to 64.6% precision by using a value of $C=3.16$.

From our modeling process, it became clear that certain aspects of our dataset such as the high-dimensionality and sparsity of the features, as well as the large number of classes, caused many of the models to have prohibitively high runtimes. In a production setting where suggestions need to be provided instantaneously, such algorithms would not be feasible options. Furthermore, we suspected that even the models that performed moderately well might have had performance issues due to the large number of classes, and would perform better in a two-step process where at either step there are fewer classes to choose from. These learnings as a whole motivated our exploration of the two-stage classifier as an alternative that might give better results.

4. Building Two Layer Models

Motivated by the strong results shown by Shen et al. in their paper “Large Scale Item Categorization for e-Commerce”, we built a soft two-layer classifier that partitions the output space into clusters of subreddit classes before predicting the subreddit label of input. We hoped that partitioning the output space into clusters of subreddit classes would allow us to effectively use more computationally demanding algorithms at the second layer. These algorithms included XGBoost and AdaBoost which, as discussed above, did not meet our standard for relatively fast runtimes.

Below, we briefly outline our algorithm for the soft two-layer classifier:

1. Build clusters of subreddits using TF-IDF representations of their respective comment corpuses.
2. Label the training data according to the subreddit clusters assigned to each comment.
3. Use a soft first-layer classifier to compute a probability distribution on the most likely subreddit clusters for input data.
4. Use soft second-layer classifiers - unique to each subreddit cluster - to compute a probability distribution on the most likely subreddit labels within a given cluster.
5. For each input point, emit the five subreddit labels with the highest probability scores in the following set:

$$\{p(x_i \in cluster_j) \times p(x_i \in subreddit_k | x_i \in cluster_j) \mid \forall cluster_j \text{ and } \forall subreddit_k \in cluster_j\}$$

4.1 Building Subreddit Clusters

Our first challenge in building the two-step classifier was to create a single, real-valued vector representation of individual subreddits and their comments. To do this, we treated each subreddit as a document by concatenating all comments in that subreddit and created a corpus of subreddit “documents”. Next we extracted TF-IDF scores for all tokens in the subreddit documents. Here we also used a standard English stop words list, applied smoothing to the inverse document frequencies, and enforced a minimum document frequency of 1.0. Finally, for each document, we sorted the tokens based on their TF-IDF scores and extracted the 250 tokens with highest scores.

Intuitively, we represented each subreddit by the comments contained within it and then used the top 250 most “influential” tokens to represent each subreddit. With this vector representation of subreddits, we were able to apply scikit-learn clustering methods to generate clusters of subreddits. We explored several clustering algorithms: k-means, mean shift, and spectral clustering. Since our choice of clustering algorithm had limited effect on our final Top-5 precision values and since it ran fairly quickly on our dataset, we chose to use k-means clustering as our subreddit clustering algorithm.

4.2 Designing an Algorithm for Soft Two-layer Classifiers

The results of Shen et al provided strong motivation to partition our large-scale text classification problem into a series of smaller-scale classification problems. However, they used this methodology to build a hard classifier for item categorization whereas our application necessitated a soft classifier that would output an array of subreddit likelihoods. For this reason, we had to independently make certain crucial algorithm design decisions which we will outline below.

4.2.1 A Soft First-layer Classifier

Initially it wasn’t readily apparent to us that we required a soft first-layer classifier - we felt that a hard-classifier that predicted a single cluster would suffice. However, it eventually occurred to us that the first-layer classifier may predict a cluster containing fewer than five subreddits. In this case, we would not be able to suggest five subreddits during classification. This presented an unfair advantage to our single layer classifier, which would always be able to suggest five subreddits since it could choose subreddits from the entire output space of 100 subreddits. Accordingly, we chose to build first-layer classifiers which emitted a probability distribution over the space of subreddit clusters. This enabled us to consult other clusters for subreddit recommendations if the first cluster was too small. (Since we wanted the clustering to actually approximate content similarity between subreddits, we ruled out the possibility of imposing a minimum size limit on subreddit clusters.)

4.2.2 Incorporating Probability Distributions from the First-layer Classifier

A second crucial decision concerned the actual use and incorporation of the probability distributions returned by the soft first-layer classifiers. Initially we considered doing the following: use the most likely cluster classification and apply the corresponding second-layer classifier for this cluster to predict as many subreddit labels as possible. As long as there are not five subreddit recommendations, repeat this process on the next most likely cluster.

This method contained an obvious advantage in that you will only have to use at most five distinct second-layer classifiers (in the case that each of the five most likely clusters only contain one subreddit). However, it did not seem entirely foolproof: if the first cluster had probability 0.25 and the second cluster had probability 0.24, did it consistently make sense to potentially recommend the least likely subreddit in the first cluster over the most likely subreddit in the second cluster? Moreover, given that we had designed a small-scale example of the problem with only 100 classes and 10 clusters - compared to the several hundred thousand classes and several thousand clusters that Reddit might use in production - it seemed likely that some of the subreddits that were clustered together in our implementation might actually be in very different clusters in production settings. Accordingly, we felt hesitant to put too much weight into the most likely cluster classification output by our first layer classifier.

Ultimately, then, to optimize top-5 precision, we wanted to make recommendations with maximal *global* probability, and this meant evaluating the probability distribution over the subreddit classes in every cluster. As before, this means that mathematically, we recommended the five subreddits corresponding to the highest values in the following set:

$$\{p(x_i \in cluster_j) \times p(x_i \in subreddit_k | x_i \in cluster_j) \mid \forall cluster_j \text{ and } \forall subreddit_k \in cluster_j\}$$

4.3 Building Two Layer Models

4.3.1 Clustering Parameters

The two key parameters in our clustering process were the number of clusters and the number of top TF-IDF words to associate a subreddit with. Keeping the first and second-layer classifiers fixed with placeholder classifiers, we varied over both parameters, trying number of clusters in the range of 5 - 20 and top TF-IDF words in the range of 100 - 500. Our results were generally inconclusive. There were very small variations in performance for various cluster settings without an obvious performance-maximizing pair. As a result, we decided to use 10 clusters and the top 250 TF-IDF words as our clustering parameters, which performed adequately well during our experiments.

4.3.2 Choosing Classifier Pairings

At the first layer in the two-stage process, we wanted a coarse classifier that would have strong performance in predicting the cluster to which a comment belonged. It is noteworthy that

this classifier must still deal with the same feature set as the single-layer classifier, but a smaller number of classes. As a result, all of the tree-based methods were not good options since they didn't handle the high-dimensionality of the data well. Likewise, the SVM classifier still had to calculate the computationally expensive class probability estimates for all of the instances, so we concluded that it would also not fare well at this stage. While multinomial logistic regression failed to run in the 100-class setting, we thought that it might fare better when only predicting as many classes as there are clusters. Thus, multinomial logistic regression along with one-versus-rest logistic regression and multinomial Naïve Bayes, the two strongest performing single-layer models, were explored as first-layer classifiers.

At the second-layer, we wanted a fine classifier that would be able to distinguish subreddits that were deemed to be similar by our clustering method. At this level, we expanded the range of model types considered. In addition to some of the models tested as single-layer algorithms, we also explored an AdaBoost classifier composed of base logistic regression classifiers and an AdaBoost classifier composed of base multinomial Naïve-Bayes classifiers as more complex second-layer classifiers.

4.3.3 Two-layer Results and Analysis

We tested a variety of combinations from the first-layer and second-layer classifiers described above. We summarize key results and learnings below, and provide full results at the end of the section.

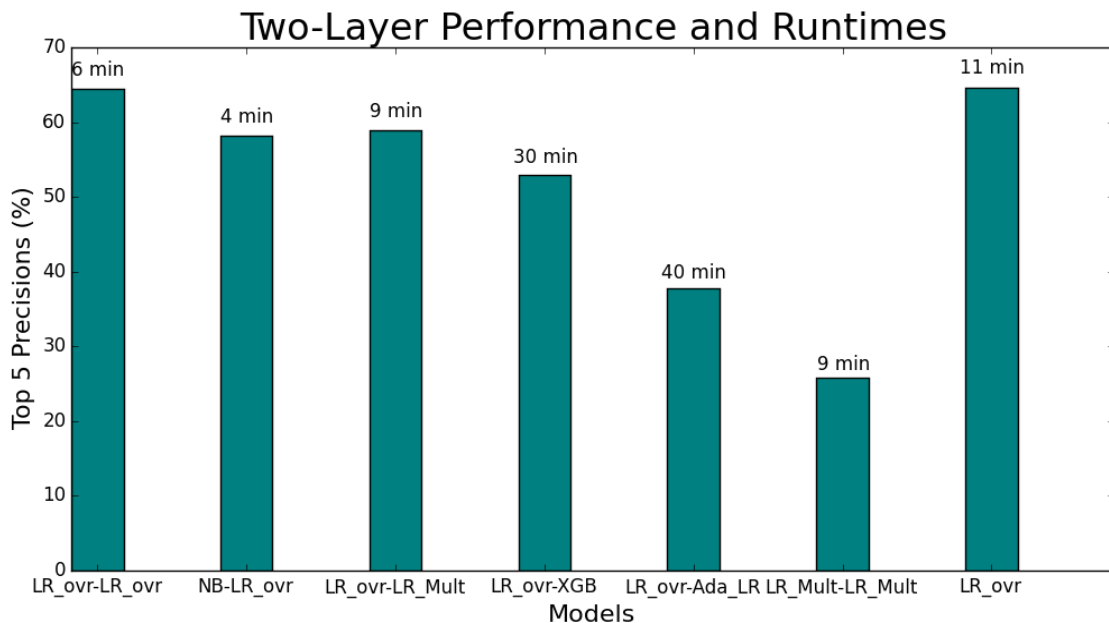
Generally speaking, models that performed well in the single-layer analysis also performed well in the two-stage process, while those that didn't perform well before had similarly poor performance. When keeping the second-layer classifier fixed, the one-versus-rest formulation of logistic regression had the strongest performance as a first-layer classifier of the three considered, followed by multinomial Naïve-Bayes, and finally multinomial logistic regression. While multinomial logistic regression had computational issues as a single-layer algorithm, now with a smaller number of classes it ran as quickly as one-versus-rest logistic regression.

For the second-layer classifier, the strongest performer was once again the one-versus-rest logistic regression model. While all of the algorithms ran faster than in the single-layer application, the tree-based models saw a significant improvement over their previous runtimes, though they still showed low Top-5 precision.

In addition to these models, we also ran an AdaBoost classifier, a sequential learner that can have strong performance. We tested using both one-versus-rest logistic regression and multinomial Naïve-Bayes as the base classifiers in the AdaBoost model. However, the performance of both models was not competitive with using a one-versus-rest logistic regression model for both classifiers. Upon further analysis, we concluded that the AdaBoost model had suboptimal performance because of the high label noise (comments with similar text in different subreddits); because of this the model placed too much weight on examples that it could never correctly classify.

Ultimately, the model with the best top-5 precision was a two-layer classifier with one-versus-rest logistic regression used for both layers. While its precision was 64.4%, which is almost exactly that of the one-versus-rest logistic regression model at the single-layer stage, it had a runtime of 6 minutes, which is almost twice as fast as before. Another notable model was a two-layer classifier composed of a multinomial Naïve-Bayes classifier at the first-layer and a one-versus-rest logistic regression classifier at the second layer. While its top-5 precision of 58.2% was not as high as the best model, its total runtime was under 4 minutes.

Below are the performances of two-layer models on the validation set, with runtimes listed above the bars. The naming convention used is ‘1st classifier’-’2nd classifier’, so LR_ovr-Ada_LR refers to a one-vs-rest logistic regression used for the first classifier and an AdaBoost classifier composed of one-vs-rest logistic regression models for the second. The best performing single-layer model is shown at the end for comparison.



4.4 Further Experiments with Two-layer Models

4.4.1 Analysis of LogReg-LogReg versus LogReg

As mentioned earlier, both two-layer logistic regression and single-layer logistic regression performed exceptionally well on our dataset. Moreover, using the same model with the same parameters as the single-layer algorithm and at both layers of the two-layer algorithm enabled us to better analyse how “splitting” the classification step in the two-layer algorithm affected our ability to recommend subreddits.

To get a better understanding of how splitting the classification affected our precision, we evaluated ‘Top-X’ precision for values less than five, as shown below:

Precision	Two-Layer Train	Two-layer Validation	Single-layer Train	Single-layer Validation
Top 1	36.2	25.2	63.6	40.7
Top 2	60.3	42.4	77.8	50.9
Top 3	75.7	53.8	83.6	56.3
Top 4	84.8	61.5	86.9	60.0
Top 5	87.2	64.4	87.7	64.6

As you can see, the two-layer classifier is actually weaker than the single-layer classifier for more stringent precision metrics: allowing the two-layer model to recommend five suggestions actually allows it to match the generally superior precision of the single layer model. This was a surprising result and one which we believe suggests the inferior quality of clustering that is occurring in the two-step classifier; this finding also motivates our next analysis below.

4.4.2 Non-linear Transformations of First-layer Probability Scores

The findings above led us to believe that either changing the quality of our two-layer clustering or changing the weight of the first-layer probabilities in our final likelihood estimations might result in more precise models. As shown in detail below, we tested models in which we squared, cubed, etc. the first layer probability scores before incorporating them into global probability scores:

$$\{p(x_i \in cluster_j)^p \times p(x_i \in subreddit_k | x_i \in cluster_j) | p \in \mathbb{Z}, \forall cluster_j \text{ and } \forall subreddit_k \in cluster_j\}$$

The intuition was that, according to our original configuration, it is entirely possible that the worst subreddit classification in the best cluster had the same or similar probability as the best subreddit classification in the worst cluster. To us, this seemed unreasonable: we wanted the worst classification in the best cluster to have priority over the best classification in the worst cluster. Accordingly, we explored mathematical transformations like squaring which would dampen scores near 1.0 far less than they would dampen scores near 0. (Since we were taking the ‘‘argmax’’ over all possible scores, we were not concerned that these scores were smaller than the initial values.)

Unfortunately, as shown below, this was not a particularly fruitful approach. (Each column reflects the train or validation results when the probabilities of the first layer are raised to the provided power.)

Precision	p^2 Train	p^2 Validation	p^4 Train	p^4 Validation	p^8 Train	p^8 Validation
Top 1	34.9	24.3	34.3	23.7	33.0	23.5
Top 2	58.2	41.6	55.4	38.9	53.6	38.6
Top 3	72.4	52.3	68.3	48.7	66.4	48.7
Top 4	81.3	59.4	75.9	55.2	73.6	55.0
Top 5	83.4	62.1	77.6	57.4	75.0	57.1

4.4.3 Manual Cluster Adjustments

The above results were a sign to us that “better” clustering might improve the performance of the model. Different iterations of K-means clustering return somewhat arbitrary results based on the randomly selected seeds for the clusters, so we felt that it might be fruitful to manually tweak the results of the clustering to create more homogenous clusters. Of course, this was an extremely subjective process, but we were interested in seeing how this adjustment affected our results.

This was an initial clustering returned by K-Means:

motorcycles, cars, newsokur
gifs, mildlyinteresting, tifu
Smite, Guildwars2, EliteDangerous, bloodborne, GlobalOffensive, smashbros, witchers, xoboxone, PS4, heroesofthestorm, wow, leagueoflegends, magicTCG, ffxiv, 2007scape, explainlikeimfive, Eve, DotA2, DestinyTheGame, gaming, Games, hearthstone
Showerthoughts, AskReddit, SubredditDrama, AskWomen, Christianity, TwoXChromosomes, AdviceAnimals, worldnews, relationships, OkCupid, news, pics, todayilearned, TrollXChromosomes, fatpeoplehate, KotakuInAction, AskMen, videos, india, conspiracy, funny, nottheonion, atheism, TumblrInAction
electronic_cigarette, rupaulsdragrace, whowouldwin, WTF, gonewild, trees, fivenightsatfreddys, Fitness, CasualConversation, peresonalfinance, anime, SquaredCircle, hip hopheads, Random_Acts_Of_Amazon, television, science, movies, thebutton, IAmA, amiibo, aww
Bitcoin, unitedkingdom, canada, europe, ukpolitics, politics
pcmasterrace, buildapc
hockey, soccer, nfl, FIFA, nab, baseball, survivor, csgobetting, MMA, formula1, CFB
gameofthrones, asoiaf
Fireteams, pokemontrades, Music, technology, Android, GlobalOffensiveTrade

Below, we tweaked it according to our (subjective) expectations of similar subreddits:

motorcycles, cars, newsokur
gifs, mildlyinteresting, tifu, explainlikeimfive, Showerthoughts, pics, funny, aww
Smite, Guildwars2, EliteDangerous, bloodborne, GlobalOffensive, smashbros, witcher, xboxone, PS4, heroesofthestorm, wow, leagueoflegends, magicTCG, ffxiv, 2007scape, Even, DotA2, DestinyTheGame, gaming, Games, hearthstone, Fireteams, pokemontrades
AskReddit, SubredditDrama, AskWomen, Christianity, TwoXChromosomes, AdviceAnimals, relationships, OkCupid, todayilearned, TrollXChromosomes, fatpeoplehate, KotakuInAction, AskMen, videos, conspiracy, nottheonion, atheism, TumblrInAction
Electronic_cigarette, rupaulsdragrace, whowouldwin, WTF, gonewild, trees, fivenightsatfreddys, Fitness, CasualConversation, personalfinance, anime, SquaredCircle, hiphopheads, Random_Acts_Of_Amazon, television, science, movies, thebutton, IAmA, amiibo
Bitcoin, unitedkingdom, canada, europe, ukpolitics, politics, worldnews, india, news
Pcmasterrace, buildapc
hockey, soccer, nfl, FIFA, nba, basebal, survivor, csgobet, MMA, formula1, cfb
Gameofthrones, asoiaf
Music, technology, Android, GlobalOffensiveTrade

Finally, the results of the two clusterings are shown below:

Clustering	Train Top-5 Precision	Validation Top-5 Precision
K-Means	87.1	64.2
Manual	87.1	64.1

Despite what seemed to us like significant changes in the clusters, the results were identical to the tenth decimal digit. A possible take-away here is that our clustering methodology may be weak beyond minute differences in what subreddits are assigned to what cluster. Rather, because we are using only 100 subreddit classes, there is inevitably a cluster that is in the ‘other’ category and is fairly heterogenous. In this case, that cluster is the fifth cluster in both the K-Means and manual clusterings. More generally, even subreddits in seemingly more

homogenous clusters might not be all that similar. Hopefully, this problem would be mitigated in a production setting where it is more feasible to use a larger number of classes.

5. Conclusion

5.1 Generalization Performance

We combined the train and validation sets and trained both the best single-layer and the best two-layer models over the full combined set, and evaluated their generalization performance on the test set. The table below summarizes the results:

Model	Train Top-5 Precision	Test Top-5 Precision	Runtime
LogReg OVR	86.4%	65.5%	17 min
2-Layer LogReg OVR	85.6%	65.1%	8.6 min

Because of its considerably shorter runtime, we recommend the use of two-layer LogReg OVR (that is, one-versus-rest logistic regression at both layers) over the use of single-layer LogReg OVR. Moreover, it is worth noting that in a production setting, the two-layer methods can be easily parallelized so that the work of training, fitting and predicting with distinct second-layer models on individual subreddit clusters can be done simultaneously by different machines. This would likely provide a significant speedup over the reported runtime of 8.6 minutes, and would provide a final model with strong Top-5 precision and exceptional speed.

5.2 Looking forward

We believe that there are still many ideas to be explored in the implementation of the two-layer algorithm. We briefly present these below.

1. *Different algorithms for different clusters*

Currently we use the same algorithm with the same parameters for the second-layer model in each cluster. It may be fruitful to explore implementations of two-layer classifications in which each cluster is assigned its own algorithm that is specifically chosen and tuned for prediction on that subset of the output space. While this method risks overfitting, it may also enable us to develop more powerful algorithms that are better capable of understanding the different kinds of signals that appear in the very different language styles of different subreddits.

2. *More Data and Better Data*

Though we were limited by the computational abilities of our machines, we believe that in a production setting a company like Reddit would be able to use a much larger amount of data and accordingly a much larger number of subreddit classes in their modelling process. Not only would this improve performance by reducing variance, it would mean that the clusters in the two-step algorithm could be more homogenous. This could mitigate the weaknesses that we found with respect to the first layer clustering and probability scoring and ultimately generate more intuitive clusters.

Moreover, it is worth noting that though we envisioned a tool which predicts subreddit classes based on the text of an original post, we were forced to build our model on data that only contains texts from comments. Since comments are often much shorter than original posts, and since original posts often use careful language and titles to clearly explain the topic at hand, we believe that a model built with this data may perform better.

3. *Analyse Similarity of Recommendations*

Thus far we have only evaluated the success of our models based on Top-5 precision and runtime. However, since our intended application of the models is as a subreddit recommendation and discovery tool, we do not necessarily care only about Top-5 precision: we also want to know how similar the provided recommendations are to the correct recommendation. By emitting more similar and relevant subreddits at the prediction stage, we would be able to build an effective discovery tool that helps users discover subreddits they may be interested in.

Sources and References

Shen, D., Ruvini, J., & Sarwar, B. “Large-scale Item Categorization for e-Commerce”. *Proceedings of the 21st ACM international conference on Information and knowledge management*.

“May 2015 Reddit Comments | Kaggle.”

<https://www.kaggle.com/reddit/reddit-comments-may-2015>