CSCI 544: Applied Natural Language Processing
Course Project: SubReddit Classifier
Report

Arka Sadhu, Lakshmi Chirumamilla

November 27, 2018

1 Introduction

Reddit [1] is a popular social news aggregation, content rating and discussion site. It has many "sub-reddits" under its domain and forms a common place for people to share and discuss about any particular topic. A sub-reddit will receive multiple posts possibly from different users. Each sub-reddit is defined by its own set of rules and regulations, about what is allowed to be posted. These rules are decided by a group of moderators who are elected by the community and the moderators of a sub-reddit can remove any posts which fall out of the subreddit and/or violates one or more of its rules. This removal process is both automated and manual.

In this work, we propose to create a dataset from posts on various sub-reddits for the purpose of Text Classification. The input to the text classifier would be the title of the post and optionally the content of the post and the output would be the sub-reddit title to which the post most likely belongs to. A model trained on such a dataset can help a user in various ways. It can give suggestions about new sub-reddits to explore based on a user's history of engaging with different posts. This can guide the user the most appropriate subreddit for his/her post so that the post is more engaging, appealing and is valued.

Our dataset which we name SubReddit-Classifier has content scraped from over 1000 different popular sub-reddits. The total size of the data set is [Add the total Number here]. We believe the research community can benefit from this data and use it for other downstream tasks such as auto-labeling of emails, auto suggestion of slack and discord channels to name a few.

2 Tasks

The main task for the dataset is text classification, that is classifying a particular post into the correct subreddit based on the title and optionally the post. We propose two tracks for this purpose:

1. Coarse Track : Contains a small set of 31 popular subreddits (see 3.1 for details). The subreddits are fairly diverse with little to no connection between them. This makes things easier for a classifier. We provide both the title and the post information for this task.

2. Fine-grained Track: Contains a larger set of 1000 subreddits (see 3.1 for details). While some subreddits are fairly diverse, there are quite a few sub-reddits which are very similar to each

other and yet they are distinct because the exact kind of posts allowed are different. Only title information is provided.

Other Tasks We also propose some auxiliary tasks which are trivial extensions of existing datasets. However, we provide no results on these and keep it open as a future direction of work.

1. Given the post and the subreddit name, predict a boolean True or False which says whether the post is relevant to the particular subreddit or not.
2. Given the post and the subreddit rules, predict a boolean True or False on whether the post adheres to all the rules of a particular subreddit or not.

3 Data

We describe the data collection procedure in 3.1. Once the data has been collected, it is cleaned via text processing pipelines (details in 3.3) and then assigned the appropriate labels (details in 3.2).

3.1 Data Collection

We use PRAW [2] for scraping data from various subreddits. It allows us to get posts sorted by popularity, new, controversial among many others. This mechanism is very fast if we are retrieving only the post titles, around 1000 titles from a subreddit within 15 seconds.

Getting List of subreddits to consider: We describe getting the coarse and fine-grained lists of the subreddits to consider.

1. We get the list of all sub-reddits created on or before May 2018.
2. We then set a minimum subscriber count (set to 2000000 for coarse and 2000 for fine-grained) and remove any subreddit which has a lower count than this.
3. We then query each of the remaining subreddit for the flags: "allow_images", "allow_gifs", "allow_videos". We remove any subreddit which has any of the flags set as True. This is required since we are only doing Text Classification. We note that by removing this step, we can make the dataset multi-modal in requiring language, computer vision and speech techniques to understand the content better. However, since we are focused only on Text Classification task in this work, we keep this as a future direction for research.
4. Additionally, we remove any subreddits which require the user to be more than 18 years of age, to allow usability of the dataset in more general setting.
5. This in turn gives us two lists: a small list of subreddits (total 31) each with a large number of subscribers which are well-moderated and hence all the posts are highly relevant and a larger list of subreddits (total 1514).
6. Finally, we also store the public descriptions of the subreddits which is an additional meta-data which can be used by a model to provide predictions.
7. Generating the lists takes nearly 1 day. The most time consuming part is getting flags of each of the subreddit and the public descriptions. It requires the crawler to send and receive requests to a large number of subreddits. Fortunately, this is a one-time process.

Crawling the subreddits

1. For each subreddit we get 1000 posts sorted by "hot", "controversial" and "top" provided by the reddit api.

2. "hot" posts are those which have garnered attention in a small amount of time. "top" posts are all time voted to be the most popular in the particular subreddit. "controversial" posts are those which have garnered very similar number of upvotes and downvotes which means that even though the community is interested in the topic, they are divided on their view points.

3. This is the reason we do not use a simple metric like setting a certain threshold on the normalized (by time since posted) number of upvotes the post has received since it doesn't always reflect the relevance of the post to the community.

3.2 Data Labelling

We note that for the dataset that we are creating the data labeling part is trivial. The label to each post is simply the subreddit that the post has been retrieved from. This is because the moderators of each subreddit have already removed posts which do not belong to the particular subreddit. Also, posts which don't gain lot of attraction wouldn't have been retrieved in the first place as they wouldn't belong to any of the "hot", "top" or "controversial" topics.

In our particular case, the label of data are free of noise, however, the text data are noisy. So we don't need any kind of manual labeling for the data (which has been already done by the moderators of the individual subreddit) however we need to clean the data for it to be useful. This is because simply extracting the title can have many cues which lead to identifying the subreddit it belongs to and defeating the purpose of our dataset. However, manual cleaning is quite expensive and hence we resort to auto-cleaning which is described next in 3.3

3.3 Data Cleaning

We describe the primary places where data cleaning is required and how solve them:

1. **Text in Parenthesis**: Quite often there are posts which have some text in the parenthesis to denote some kind of tag. For example, on subreddits like "r/MachineLearning", one needs to put "[D]" for Discussion and "[R]" for Research as tags before every title of a post. These are often subreddit specific and gives strong indication to the classifier, however, we want our classifier to work independent of this. Hence we remove such words inside parenthesis using regular expressions.

2. **Text before Title** Some subreddits require a post to have the name of the subreddit or an acronym before the main word. For example, the subreddit "r/tifu" requires "TIFU" before every title. To remove such cases, we get the most common words in the titles, and if there is any particular word which occurs at least once in every title, we remove the word. This also removes posts which have some week numbers included in the title.

3. **Inactive subreddits** We found that even subreddits with high number of subscribers are in fact in active. This means that even if the subreddit was more suited to a user's interest, there would be little meaning in posting there as it is inactive. Moreover, the training data for such subreddits becomes very less. Additionally, it also brings the quality of moderation

into question and the dataset quality may be sacrificed. Hence, we filter subreddits with less than a certain number of "hot" posts. We noted that this problem didn't arise with the coarse list, but with the fine-grained list only.

4. **Removing Discussion Posts** We also remove posts which are titled discussions. These mostly belong to subreddits about a tv show and the titles are like "Discussion Episode 54". These are not interesting examples and don't have enough information to help us categorize.

5. **Removing small titles** We also remove posts with very small titles (of length < 5). We found these often lacked any cues that allowed us to classify these examples uniquely.

6. **Class Imbalance** We note that for most classifiers the input data should have similar proportions of the different classes to avoid any kind of implicit biases. For this we create a subset of titles such that each subreddit has exactly 300 posts. We sample these at random. This also helps us create a subset which can be processed with our computational resources. This is used in 3.4.2

3.4 Training and Test Set Sizes and Distribution

Here we provide the different training, validation and test splits we created.

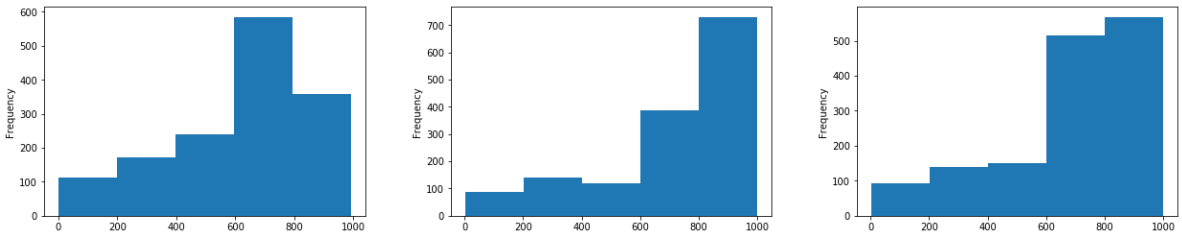
3.4.1 Total Data

We first show the distribution of all the data collected. After the data cleaning as described in 3.3, we are left with 1466 different subreddits. The total collected data in each of the type: "hot", "top" and "controversial" is given in Table 1.

Type	Hot	Top	Controversial
Full Data	908501	1050328	1002432

Table 1: Number of posts scraped in each type

We also produce the distribution in the form of histograms in Figure 1



(a) Hot (b) Top (c) Controversial

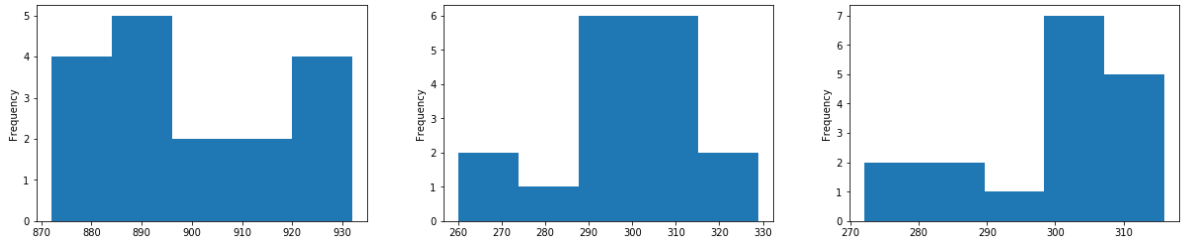
Figure 1: Histogram plot by different sorting methods. x-axis is the number of posts of a particular subreddit in the particular category ("hot", "top", "controversial") and y-axis is the frequency of that number. 5 bins are chosen in each case. Note that there are some subreddits which have low posts in the hot/top/controversial category, these are filtered out before creating the coarse and fine grained sets.

3.4.2 Coarse Dataset

We use the small list as described in 3.1 and then require each of subreddit to have at least 500 posts in each of the three categories. Initially, the small list has a total of 31 subreddits, after pruning there are 17 subreddits. We sample 500 from each category leading to 1500 posts for each subreddit. We do a train, validation and test split in the ratio $60:20:20$. We provide the numbers in 2 and histogram distributions in 2

Type	#Labels	Train	Validation	Test	Total
Coarse Data	17	15300	5100	5100	25500

Table 2: Number of posts in train, test and validation split of the coarse data



(a) Train (b) Validation (c) Test

Figure 2: Train, validation and test distributions of the coarse dataset.

We note the following about the coarse split:

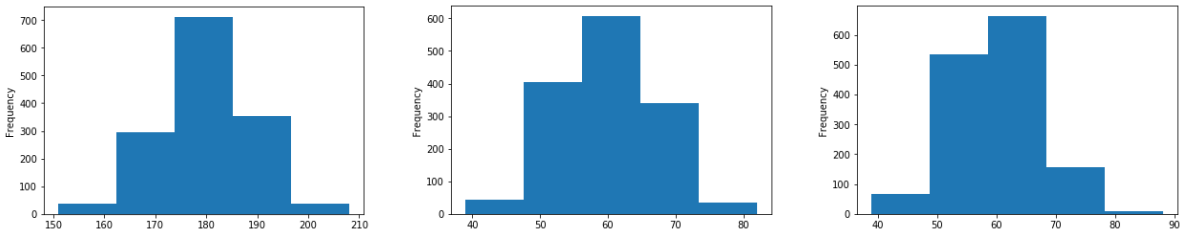
1. There are only 17 labels, which are more or less well separated from each other, and hence classification in this setting is easy.
2. We also note that the total number of examples is small which is only $25k$ examples. Note that the full data has $\approx 3000k$ examples. However, because of computational resources constraint, we are not using the full data. However, the coarse data still gives us plenty to make observations (noted in 5).

3.4.3 Fine-grained Dataset

We use the bigger list as described in 3.1 and require each subreddit to have at least 100 posts in each of the three categories. The initial list of 1515 subreddits is reduced to 1416 subreddits. We sample 100 from each category leading to 300 posts for each subreddit. We do a train, validation and test split in the ratio $60:20:20$. We provide the numbers in Table 3 and histogram distributions in Figure 3.

Type	#Labels	Train	Validation	Test	Total
Fine-grained Data	1416	257580	85860	85860	429300

Table 3: Number of posts in train, test and validation split of the coarse data



(a) Train (b) Validation (c) Test

Figure 3: Train, validation and test distributions of the fine-grained dataset.

We note the following about the fine-grained split:

1. There are 1416 labels and there are many closely related subreddits, however, each having some distinctive feature. This makes classification tough.
2. We also note that the total number of examples $429k$is larger than coarse which is only $25k$ examples. However, it is still an order of magnitude lower than the full data. The primary reason for this is to make the data balanced along the different classes to avoid any kind of bias.

4 Method

We describe a baseline approach (details in 4.1) and another Classification Approach based on LSTMs (details in 4.2).

4.1 Baseline Approach -TFIDF

4.1.1 Theory

We use Term Frequency - Inverse Document Frequency (TFIDF) [3] as the baseline approach. Tfidf assigns a weight assigned to a term to evaluate how much important it is to a particular document when compared to the whole corpus. The weight increases proportionally to the frequency of the term in the document offset by the frequency of the term in the whole corpus. Mathematically, the term frequency of a term t for a document d is given as:

$$TF(t,d) = \frac{|\{x \in d | x = t\}|}{|\{x \in d\}|} \tag{1}$$

Letting D to be set of all documents, we get the inverse document frequency to be:

$$IDF(t) = log\left(\frac{|D|}{|\{d \in D | t \in d\}|}\right) \tag{2}$$

The final weight TF-IDF is the product of the two:

$$tfidf(t,d) = TF(t,d) * IDF(t) \tag{3}$$

Thus, a term t with high $tfidf$ for document d means that it occurs with a much higher probability in the document d than in other documents and therefore is highly discriminative. Low $tfidf$ but high tf (low idf) implies that the word is a common occurrence across multiple documents and hence doesn't provide any discriminative features and can be ignored. These are likely to be words like "a", "the", "in" etc. A low $tfidf$ and a low tf implies that the word is a rare occurrence in the document itself and therefore we cannot infer anything from that word.

4.1.2 Implementation

For implementation purposes, we use the scikit-learn library [4] and spacy [5] library. We use the stopwords provided by the spacy library and use the tf-idf implementation in scikit-learn followed by logistic regression. For a corpus of size 18k (i.e. 18k labeled titles) it takes around 10 seconds for convergence, for 200k it takes 2 minutes.

4.2 Classification Approach - ULMFiT

4.2.1 Theory

We use Universal Language Model Fine-tuning for Text Classification (ULMFiT) [6] which appeared in ACL 2018 for our dataset. The paper claims state of the art on various text classification tasks, in particular, the IMDB [7] dataset which does sentiment analysis.

The core idea of ULMFiT is to first train a language model on a large corpus which like WikiText103 [8]. It uses AWD-LSTM [9] as the underlying language model. AWD-LSTM is a regular LSTM [10] without any shortcuts or attentions but has various tuned dropout hyper parameters.

Once the language model is trained on a large corpus, it is fine-tuned on the target corpus. After this, the knowledge can be transferred to text classification tasks. The language model is augmented with two additional linear blocks and Relu activation. The linear blocks take input the last hidden states of the LSTM, and concatenates it with mean and max pool of all the hidden states. Mathematically, if the hidden states of the LSTM are represented with $H = [h_1, ...h_T]$, the last linear block takes in input

$$h_c = [h_T, meanpool(H), maxpool(H)] \tag{4}$$

Here [] is the concatenation operator. The paper proposes detailed information about fine-tuning the hyper-parameters and we refer the reader to Section 3 of [6] for more details.

4.2.2 Implementation

We use the code provided by the author and adapt it to our dataset. It requires at least 1 GPU to train and is computationally expensive. For a corpus of size 18k labeled titles, to attain the same performance of TF-IDF it takes around 30 minutes (180 times slower) which is expected. However, on the whole it gives better performance when trained till convergence.

5 Results and Discussions

In this section, we report results of both TF-IDF and ULMFiT on our datasets with coarse and fine-grained settings.

5.1 Evaluation Criteria

We use top1 as the evaluation criteria. We also add the training time required with different amounts of data. We additionally provide the confusion matrix obtained for the coarse setting since the number of classes are less (17).

5.2 Experimental Setup

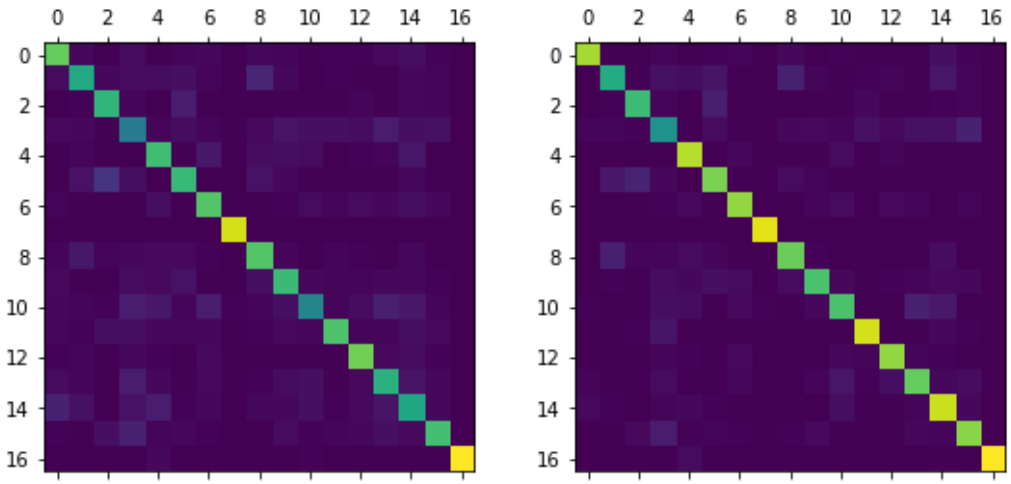
For TF-IDF we use logistic regression and use the Limited Memory BFGS optimizer and train till convergence. This is trained on a 16 core CPU machine.

For ULMFiT we first fine-tune the language model on the text of our dataset. We train this till training loss is lower than validation loss to ensure convergence of the fine-tuned Language model. Then we train just the classification layer, keeping the language model fixed, with learning rate of $1e^{-2}$. We noted that this always lead to underfitting, i.e., the training loss was always higher than the validation loss. We then unfreeze the layers of the language model and allow them to be trained as well, however we use much lesser learning rates (in the range of $1e^{-5}$ to $1e^{-3}$ similar to the author's work). Again, training till convergences (till the model starts to overfits) gives us the final results. This is trained on 1 P100 GPU provided by Amazon Web Services (AWS). In all experiments related to ULMFiT we keep the batch size fixed to be 42.

5.3 Results and Discussions on Coarse Dataset

Methods	Accuracy-Top1	Train-Time
TF-IDF	70	3.4 seconds
ULMFiT	80.2	30 mins

Table 4: Accuracy and Train-Time for coarse dataset



(a) Confusion Matrix for TF-IDF (b) Confusion Matrix for ULM-FiT

Figure 4: The confusion matrix are plotted for TF-IDF and ULMFiT for the coarse dataset.

For the coarse dataset we observe the following:

1. TF-IDF is blazingly fast for a small corpus. It is nearly 500 times faster than ULMFiT even when run on a CPU, while ULMFiT is run on a GPU. This is because TF-IDF needs to simply go over the terms and document and requires very less complex calculations.

2. The optimization used by TF-IDF is for a quasi-convex problems and the particular method uses second order derivatives which are known to give faster solutions when the Hessian is simple to calculate.

3. Since the terms by themselves are discriminative as the particular subreddits are quite distinct, we see that the baseline TF-IDF itself reaches accuracy of 70%. Note that a random classifier would instead have only 6% accuracy.

4. However, we note that ULMFiT outperforms TF-IDF by a whooping 10%, which is nearly 15% relative improvement.

5. This performance improvement is interesting to note because usual deep learning models are extremely data hungry. However, in this case, we are mostly fine-tuning on our target task and not learning from scratch. This allows us to train on considerably less data. In fact, the author claims to be able to train on as less as 1000 samples of IMDB. We note that each sample of IMDB contains way more data (appx 10 times) than our titles and therefore number of samples is not a fair comparison.

6. We also plot the confusion matrix for the two approaches as shown in 4. We find that the ULM-FiT in general gets confuses less than TF-IDF which is also reflected in the final accuracy.

5.4 Results and Discussions on Fine-Grained Dataset

Methods	Accuracy-Top1	Train-Time
TF-IDF	34.37	39.9 minutes
ULMFiT	39.1	13 hours

Table 5: Accuracy and Timing results for fine-grained setting

For the fine-grained dataset we make the following observations:

1. We first note the drastic drop in performance in both TF-IDF and ULMFiT. This is expected because now the classification is across 1466 classes. Note that a random classifier would have 0.06% accuracy.

2. We find that TF-IDF is surprisingly robust even when larger number of classes are involved as its accuracy is not very low.

3. The training time for TF-IDF is substantially larger than for coarse setting. The training data increased by 15 times than the coarse setting, however the training time increase is nearly 700 times. However, we note that the training time for ULMFiT though larger it is nearly 15 times that of the coarse data.

4. The above point denotes that the time for convergence in the case of TF-IDF doesn't scale linearly, but that for ULMFiT scales linearly. One of the important factors could be the large number of classes. This shows that for TF-IDF, the bottleneck is the softmax layer, while for ULMFiT, the LSTM layers take up significant time and therefore softmax is no longer the bottleneck.

5. We note that the ULMFiT doesn't provide too large an increase than TF-IDF. However, it is important to note that there is no explicit memory about which word goes to which subreddit. Rather it learns just from the structure of the titles. We believe that for such a large scale classification, ULMFiT is still able to provide good enough output. Moreover, it is possible that better selection of hyper-parameters may lead to slightly better performance as well.

6. Another thing to note is that the fine-grained has more data and therefore a 5% increase is infact more significant than that of the coarse grained dataset.

6 Conclusion

We provide a dataset based on reddit titles for the purpose of classifying them into proper subreddits. We consider two subsets a coarse one and a fine grained one and show the differences between the datasets with respect to their classes and the number of training examples. Finally, we compare two methods in both the settings and draw the conclusion that fine-grained classification is significantly harder than coarse classification.

7 Team Information

Author1: Arka Sadhu (asadhu@usc.edu)
Author2: Lakshmi Chirumamilla (chirumam@usc.edu)

References

[1] Wikipedia contributors. Reddit — Wikipedia, the free encyclopedia, 2018. [Online; accessed 26-November-2018].

[2] praw developers. Praw:python reddit api wrapper. *https://github.com/praw-dev/praw*, 2018.

[3] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017.

[6] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 328–339, 2018.

[7] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[8] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[9] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.