Reddit Comments Classification

Azfar Khoja[20148734] and Bhavya Patwa[20164870]

IFT:6390, Team name: Fafda Jalebi

1 Introduction

Our task is text classification in a supervised setting. We are asked to predict the topic for a given reddit post. The dataset consists of 100,000 posts taken from 20 predetermined topics (subreddits),where 70% of the posts are labeled and we evaluate our model on the remaining 30%. Prediction accuracy is the metric used for the task.

Our final submission, which helped us achieve 0.606 categorization accuracy on the private leaderboard is a soft voting ensemble of four different classifiers. 1) Complement Naive Bayes , 2) ULMFiT (language model) [4] 3) Multi-Layer Perceptron [9] 4) SGD Classifier (Linear) with Modified Huber Loss [11].

2 Feature Design

2.1 Preprocessing:

Initially, we first clean the data with unnecessary symbols like '\b' and convert all examples into lowercase as we want uniform tokens regardless of the alphabet case. Then, we perform tokenization for splitting words as individual tokens. The simplest way to get tokens is to split at "white spaces". There are smart ways of tokenizing which are available through packages like nltk [6], so we use the nltk tokenizer. After observing the vocabulary of tokens generated, we found there were many tokens that were in form of URLs, so we also split them across the _,/ (underscore,forwardslash) symbols and processed the split word list as tokens. We also ignore tokens that match the list of stopwords (nltk.corpus.stopwords).

For the initial milestone, we use counts of all words in every class ($count(w_i, c)$) and word count of a every class ($\sum_i(count(w_i, c))$) as features. For the latter part of the competition, we use tf-idf features to represent the data. Depending on the model used, we use spaCy, nltk[6] or fastai libraries to numerically represent the vector of features.

For speeding up pre-processing and computation, we optimized our look-up calls for preprocessing tasks like lemmatization, getting probability of an occurence of a word given a class by using the lru_cache feature in Python (functools.lru_cache).

2.2 Feature Design

Vectorization We need to transform the information present in tokens to numerical features that feed into a machine learning model. This is vectorization.

Bag Of Words (BOW): In BOW, we represent each reddit post as a long vector where each element represents the occurance count of a particular token

TF-IDF (Term Frequency-Inverse Document Frequency) [1] As in BOW we represent each reddit post as a long vector, but in this case we try to capture the importance of a particular token to a given post.

TF-IDF (for term (t) document (d))

$$tf\ idf(t,d) = \frac{count(t,d)}{|d|}.log\frac{|D|}{|\{d \in D : t \in d\}|}$$

where D represents the set of all documents.

3 Algorithms

3.1 Naive Bayes

Based on the Bayes Theorem, it assumes conditional independence of features (count(wi,cj), in this instance), and all predictors have equal effect on the outcome. The likelihood of a class is the product of the probabilities of the input features for that class. It is regularized with Laplacian Smoothing to incorporate the problematic issue of a given class and value not occuring together in the training data where the probability estimate will be zero.

3.2 Complement Naive Bayes

[7] For Complement Naive Bayes, instead of calculating the likelihood of a word occuring in a class, we calculate the likelihood that it occurs in other classes. We take an inverse of the prior-probabilities and minimize the objective of the likelihood function.

3.3 Linear Classifier using SGD

[7] A linear classifier where the gradient of the loss is estimated for every sample and the model is updated in an online fashion at every sample or mini-batch. We use a generalized version of the Quadratically Smoothed Hinge Loss, the Modified Huber Loss [11].

3.4 Multi Layer Perceptron (MLP)[9]

MLP is an artificial neural network that can have multiple layers of fully connected neurons between the input and the output. The output layer makes a decision or prediction about the input and the model uses backpropagation for training its parameters. According to the universal approximation theorem, MLPs with one hidden layer is capable of approximating any continuous function.

3.5 ULMFiT [4]

ULMFiT uses the AWD-LSTM language model. It consists of three stages: a) The language model is first trained on a general domain corpus to capture general features of the language. b) The language model is then fine tuned on the target task to learn task specific features. c) We then augment the language model by using two additional linear blocks and fine tune the model to perform target task classification.

4 Methodology

We use the same 90%-10% stratified split of the labeled data for the train and validation sets across three algorithms. The only difference was in the case of ULMFiT where we used a random split of 15% for validation. We briefly describe the methodology for each algorithm.

4.1 Naive Bayes

For the first milestone, we tried combination of parameter search such as the minimum frequency of vocabulary tokens (integer), lemmatization of tokens (on/off), lemmatization based on POS-tags (Part-Of-Speech) of the sentence (on/off), filtering number tokens in alphabet form (on/off), splitting on _,/ (on/off). We after a parameter search by imposing the minimum frequency on vocabulary for 2,3,4, discovered that imposing minimum frequency of 2 for a vocabulary token gave us the best results on the validation set. Also, default lemmatization without providing POS tags gave higher accuracy than with providing POS tags of the tokens. Splitting on _,/ gave us marginal improvement, so we decided to keep it for little gains. The Laplacian smoothing by adding the size of the vocabulary in the denominator as a regularizer gave us 53.57% accuracy. At last, converting likelihood probabilities to log-probabilities was able to beat the baseline scores on the leaderboard. We chose the train/valid splits of 0.90/0.10 as the accuracy on 10% of train set as our results on the validation were consistently close as the results on the public leaderboard.

4.2 Complement Naive Bayes(CNB)[8]

For the second milestone, we switched to Tf-Idf as our primary features and the Complement Naive Bayes Classifer. With a search of the alpha hyperparameter (best alpha=2.6), it gave us an improvement with a score of 58.5% on the public leaderboard.

4.3 Multi Layer Perceptron (MLP)[9]

In our task we used a two layer MLP with 300 neurons in the hidden layer along with modern practices like cross-entropy loss, ReLu activations, dropout, batchnorm and weight decay. We use the adam optimizer with cyclic momentum to train the network as described [10]. Through manual search, we found p =0.8 and wd =0.03 to be good values of hyperparameters for dropout and weight decay respectively. In regards to learning rate (lr), we use the method described in [2] [10] to find a good range for the learning rate and then try values within the range.

4.4 ULMFiT [4]

We start with using a AWD-LSTM pretrained on 'wikitext-103' dataset of wikipedia articles. . Since we expect there to be significant difference between the wikipedia dataset and our reddit corpus, we finetune and update parameters of the model on our copus and finally add the custom linear head to perform classification. We use the fastai[5] library to perform these steps and resort to using the default parameters. We train the model using discriminative fine-tuning (each layer is tuned with different learning rates) as described in the original paper.

4.5 Stochastic Gradient Descent classifier (SGD)[7]

We train a SGD classifier with modified huber loss using the scikit-learn library. The reason for choosing this over Linear SVM[3] is threefold. 1) We need a probabilistic classifier to complement the three algorithms trained above. 2) This implementation works with data represented as sparse arrays of floating point values. 3) The modified huber loss closely resembles squared hinge loss.

4.6 Hard and Soft Voting Ensemble

We used a hard-majority voting of 3 models (ULMFiT, cNB and MLP). In case of a tie, we used the predictions of Complement Naive Bayes. This made our prediction scores jump by an another 2% at 60.59%. To improve this, we used soft-voting ensemble by using probability scores of the predictions from different models which further made our accuracies jump to 61.01%.

We average out the predictions of four probabilistic classifiers as follows:
For each test input

$$prob = \frac{CNB + MLP + ULMFiT + 0.5.SGD}{4}$$

The class having highest $prob$ is returned as the final prediction.

5 Results

Performance on the public leaderboard: 0.61
Performance on the private leaderboard: 0.606

5.1 Analysis

The number of unique tokens on the training set were about 61090 with 29758 being tokens with a global frequency of just 1. After observing the list of 1-frequency tokens, we found that most of them were unmeaningful jumbo and did not make sense. It also occurred that the low-frequency tokens will heavily distort the final likelihood towards the document-class it belonged in. Thus, a parameter search on the minimum imposed frequency was paramount.

Moreover, we tried different classifiers like SVM [3], Gradient Boosting and Bagging. SVM [3]without gradient descent took an incredible amount of time to converge, so we dropped this idea. For Gradient Boosting, we were not able to cross 49.4% accuracy even after best hyperparameter search (learning rate=0.1, max depth=4, n estimators=200, score=0.494) with cross-validation. We did bagging using MLP with 10 estimators but it didn't improve as expected.

After trying many classifiers and the good ones maxing out accuracy at 55-58% on the validation set, it occurred to us that taking an ensemble of votes (n=3) should be beneficial. To test this, we took the best performing models of ULEM fit, Complement NB and the MLP and applied a hard-majority vote of predicted classes. With soft-majority using average of probabilities, we were able to achieve 0.606 (14^{th} rank) on the private leaderboard.

CNB: smoothing parameter $\alpha = 2.6$. SGD: L2 regularization term (α): 0.0001, $max_iter = 110$ MLP: 1 hidden layer with 300 neurons, epochs: 5, max learning rate (lr):0.003, Dropout Parameter (p): 0.8, weight decay (wd): 0.03.

Table 1. Results on the validation set for different classifiers.

Classifier	Categorization Accuracy
CNB	0.582
MLP	0.576
SGD	0.57
ULMFiT	0.5826

5.2 MLP[9] Loss Curves

Below is the loss curve obtained for MLP

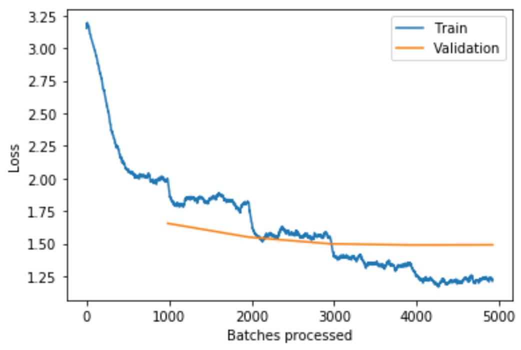

Fig. 1. Train and Validation loss curves

6 Discussion

6.1 Pros:

Complement Naive Bayes [8] was designed to correct the severe assumptions made by the standard Multinomial Naive Bayes classifier such as feature independence and particularly suited for imbalanced data sets. It was able to improve upon Naive Bayes as the number of tokens across each class varied vastly.

Soft-Ensembling using four different families of models helped us achieve a score of 0.606. Thats a 2% increase in performance compared to our best performing model.

6.2 Cons:

Since we didn't have the same train/validation set for all the models even after same split ratios, we ended up using the public leaderboard score to validate our soft-ensemble. This also prevented us from using a weighted average of probabilities.

The '0.5' weight assigned to output of SGD is an arbitary number with no justification. We didn't set a random seed for NN parameter initialization. This led to different results every time.

6.3 Idea's for improvement

Use the same train and validation sets across all models you wish to ensemble. Normally, splitting the train and validation in the first place would be a good idea to avoid this. Setting a random seed from the start would have also helped mitigate this.

If we observe the confusion matrix (Appendix), the worst performing classes are AskReddit, funny and worldnews. Sometimes comments were generic enough that even a human cannot classify it. Models focused on correctly classifying these 3 classes can significanly improve the accuracies.

7 Statement of contributions

7.1 Bhavya Patwa

Bhavya coded the initial Naive Bayes algorithm with laplacian smoothing for the first milestone. He also trained classifiers namely bagging on MLP, Gradient Boosting etc. and visualized the results using confusion matrices for further insights. His idea of voting based classifier was key behind crossing the 60% accuracy mark.

7.2 Azfar Khoja

Implemented ULMFit, MLP, SGD models and probabilistic soft-voting that helped reach 14th on the leaderboard

Both have contributed equally in writing the report.
We hereby state that all the work presented in this report is that of the authors.

References

1. Text mining, https://ift6758.github.io/lectures/NLP_part1.pdf
2. Gugger, S.: The 1cycle policy, https://sgugger.github.io/the-1cycle-policy.htmlthe-1cycle-policy
3. Hearst, M.A.: Support vector machines. IEEE Intelligent Systems **13**(4), 18–28 (Jul 1998). https://doi.org/10.1109/5254.708428, https://doi.org/10.1109/5254.708428
4. Howard, J., Ruder, S.: Universal language model fine-tuning for text classification (2018)
5. Howard, J., et al.: fastai. https://github.com/fastai/fastai (2018)
6. Loper, E., Bird, S.: Nltk: The natural language toolkit. In: Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1. pp. 63–70. ETMTNLP '02, Association for Computational Linguistics, Stroudsburg, PA, USA (2002). https://doi.org/10.3115/1118108.1118117, https://doi.org/10.3115/1118108.1118117
7. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
8. Rennie, J., Shih, L., Teevan, J., Karger, D.: Tackling the poor assumptions of naive bayes text classifiers. Proceedings of the Twentieth International Conference on Machine Learning **41** (07 2003)
9. Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review pp. 65–386 (1958)
10. Smith, L.N.: A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay (2018)
11. Wikipedia contributors: Hinge loss — Wikipedia, the free encyclopedia (2019), https://en.wikipedia.org/w/index.php?title=Hinge_lossoldid=924574237, [Online; accessed 16-November-2019]

8 Appendix

8.1 Confusion Matrix

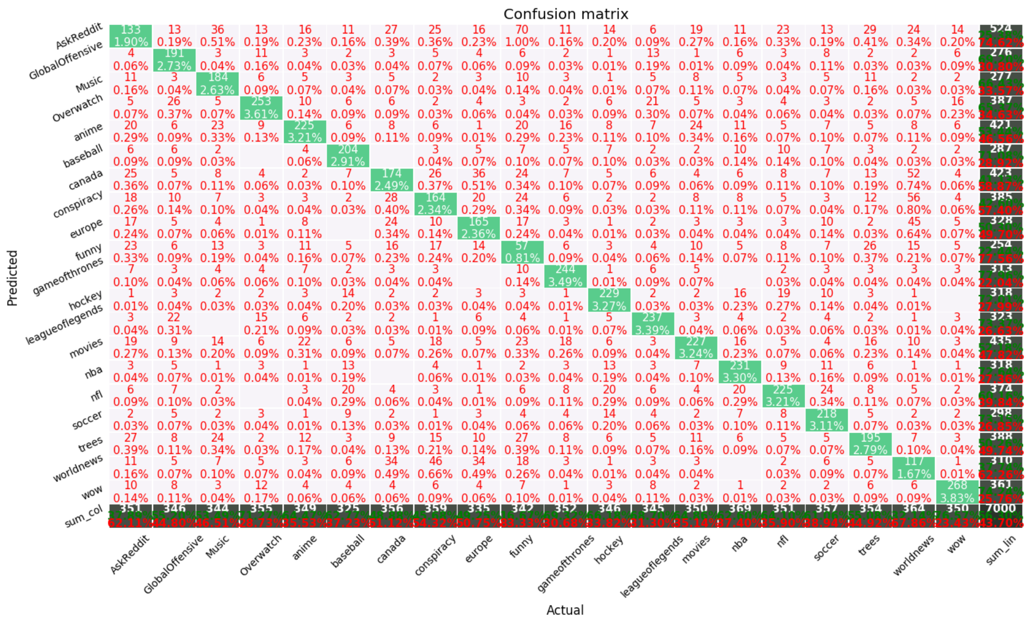

Fig. 2. Confusion Matrix on Naive Bayes with Laplacian smoothing

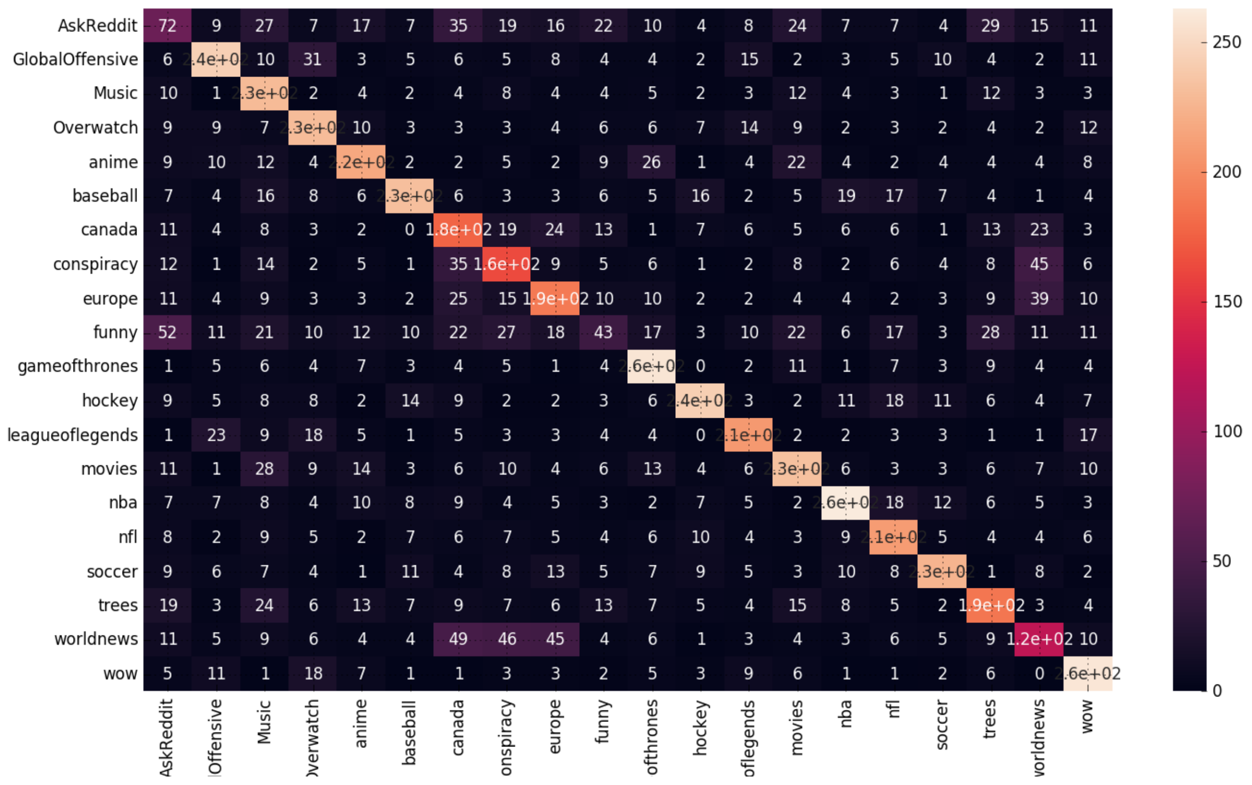

Fig. 3. Confusion Matrix on Complement Naive Bayes