Kaggle Competition report

The Wordsmiths
Thach, Jean-Pierre Ploujnikov, Artem
Department of Computer Science and Operations Research, University of Montreal

1. Introduction

The objective of this competition is to design a machine learning algorithm that can automatically sort short texts into a predetermined set of topics. The source of the data comes from posts in subreddits. Our approach begins with designing and selecting features that would hypothetically improve performance of the algorithms, e.g. Naive Bayes, Logistic Regression, etc. And explore various embeddings from different pretrained models such as Word2Vec, Universal Sentence Encoder, etc. and see how they compare to machine learning algorithms. We concluded that ensembling our multinomial naive bayes, the Universal Sentence Encoder model and a linear SVM model with soft voting brought the highest public test accuracy of 63.7% in the competition.

2. Feature Design

2.1 Preprocessing methods

From the previous phase, we've implemented a Naive Bayes model with simple preprocessing methods such as lowercasing, removing non-alphanumeric words, removing stop words and tokenization. In the tokenization step of this phase, the words in the corpus are converted into lowercase format. The underlying reason is that, in most cases, capitalization of words has no effect on the meaning. The rationale of stopwords removal and keeping words that contain alphabetic characters is to not consider words that may not convey much information as we are looking to keep meaningful tokens for each document representation. Furthermore, removing urls is another pre-processing method applied for the tokenization process. Our intuition is attributed on the same principle that they may not contain useful information. However, they could be for feature extraction.

In addition, since social media posts likely contain many grammatical errors and noisy data, we opted to try and apply stream editing for lexical normalization with *sed*. The purpose of this preprocessing method is to transform real-world data into an understandable format and to obtain a consistent input that allows separation of concern before processing it.

2.2 Feature selection

In order to obtain a baseline classification, normalized, TF-IDF-weighted bag-of-words features were used.

The following features were hypothesized to have an effect on the probability of a given comment being in a given subreddit, and therefore, to improve the performance of the learning algorithm:

Comment length, word and sentence count
Some subreddits might call for long discussions on elaborate topics, whereas others are more meant for sharing small tidbits of information.

Complexity & Reading Difficulty
Similarly, some subreddits contain more "light-hearted" content than others, whereas some call for elaborate discussions using complex vocabulary and sentence structure.

The following measures were calculated by using the *textstat* library: *Flesch Reading Ease*; indicates the reading complexity of the material with higher scores being easier to read. *Flesch-Kincaid Grade*; describes the U.S. grade

level of education generally required to understand the material. *Smog Index*; a metric characterizing the number of years of education needed to understand a text. And finally, the number of polysyllabic words.

Swear words
In everyday life, some settings and discussion subjects are notorious for use of profanity, whereas others tend to be more civil. For instance, profanity is used extensively in stand-up comedy, but not, for instance, in discussions about science or current events.

In the preliminary exploratory data analysis, the following measures of profanity were computed for each subreddit: Percentage of comments with profanity, the average number of swear words per comment and the average number of swear words in a comment that has swear words.

Part-of-speech Tagging
By combining words with their context, an overview of the structure representation of comments can be obtained. With this added complexity, part-of-speech was hypothesized in resulting in better performance.

3. Algorithms

In order to analyze the impact of the features and preprocessing steps, several classification algorithms deemed to be adequate for text classification are chosen for the analysis.

3.1 Naive Bayes with handcrafted features

For this learning algorithm, the motive behind using the multinomial distribution for text classification comes from the representation of the words as discrete features. Also, the Laplace smoothing is taken into account for unseen words by tuning the hyper-parameter *alpha*. A Bayesian estimator for Gaussian processes from the *scikit-optimize* was used for hyperparameter fitting.

3.2 Logistic Regression with handcrafted features

For this learning algorithm, an advantage of selecting this model for this task is that there exists a slight collinearity of features in addition to a regularizer that penalizes high variance. Whereas the Naive Bayes algorithm assumes complete independence of features. Also, Logistic Regression tends to catch up or even outperform Naive Bayes in terms of performance as the size of the data increases.

3.3 Neural Network Models

The principal motive for selecting this model is its high complexity. Additionally, since the features are learned from the model, handcrafted features would be unnecessary. Initially, a very simple model was prototyped with a multi level perceptron trained on the same data as the baseline Naive Bayes model. Subsequently, a more sophisticated sequential model was built consisting of an embedding layer, a spatial dropout layer, a LSTM layer with dropout and recurrent dropout and a dense layer. The spatial dropout layer is defined as a layer that dropout entire feature maps. In the case of standard dropout for the LSTM layer, each input and output of each RNN unit will be considered independently in the dropout. With the same principle, a fraction of the recurrent connections between units will be dropped out. The dense layer will be used in the output layer to connect the neurons and produce probabilities for each class with a softmax activation function.

3.4 Universal Sentence Encoder

This pre-trained model from the TensorFlow Hub encodes text into high dimensional vectors using a Deep Averaging Network (DAN) architecture, which, unlike many other models used in deep NLP, averages word and bigram embeddings prior to passing them through a deep neural network. It is trained and optimized for sentences, or short paragraphs on different data sources aimed to accommodate a variety of

natural language processing tasks. It takes an input of variable length and outputs a 512 dimensional vector. Our strategy and goal in using this encoder is to build a model on top of it and see how well it would perform on noisy and unclean dataset.

3.5 Ensembling model

A simple ensemble model was built incorporating two classifiers (Naive Bayes with handcrafted features and the Universal Sentence Encoder) a soft voting scheme where the probabilities from each classifier are averaged for each class, and the class with the highest average probability is selected.
More sophisticated methods, such as bagging or boosting, were not attempted with these two models because neither model appeared to exhibit a high variance problem, which bagging attempts to solve, and boosting is typically used with weak classifiers.

4. Methodology

For this set of experiments, the training data is split into 90% training data and 10% validation data for the purpose of hyper-parameter tuning in the validation phase. Additionally, we refer to the Naive Bayes with smoothing proposed of 55.488% as baseline of our experiments.

We've chosen the multinomial distribution for naive bayes given that words can be represented as discrete features. In order to capture all signs of performance increase or decrease, our strategy in tuning the Laplace smoothing *alpha* begins with a uniform space of real values from 0.0 to 5.0. Then we use the *scikit-optimize* library to optimize the search of the best hyperparameter value in 100 calls.

With the same approach for the regularization strength *C* of the multinomial logistic regression, we used an interval from 0.0 to 5.0 to determine the value of *C*. Furthermore, the solver was set to *lbfgs* given that it is appropriate for multiclass problems. The random state was fixed to 42 in order to obtain idempotent results. And finally, the maximum number of iterations was set to 200 to allow the learning algorithm to attempt convergence, and also to prevent the algorithm from overfitting by letting it run forever.

Since neural networks are complex models, several hyperparameter values are tuned in order to control the computational complexity, or essentially the time and resources required to train the model. For this reason, in our LSTM model, the vocabulary size was set to 10000, the maximum number of words as input sequence per comment was fixed to 250 and the dimensions of the embedding to 100. The dropout, the recurrent dropout and the spatial dropout was settle to 0.4. The structure of our sequential model involves an embedding layer, a spatial dropout layer, a LSTM layer with dropout and recurrent dropout, a dense layer and uses the Adam optimizer with categorical cross entropy loss function. The number of epochs in the training phase was set to 10.

Our next model uses the Universal Sentence Encoder to preprocess the input to the following layers: an input layer, two dense layers of 1024 units each with *ReLU* activation function, a dropout layer with the dropout hyperparameter set to 0.1 and a dense layer with a softmax activation function. The model is compiled with the Adam optimizer and categorical cross entropy loss function. Moreover, the number of epochs was fixed to 100 in the training phase.

For the ensembling model, our approach boils down to using multiple models together with their respective hyperparameters set to the optimum values in the validation phase without overfitting.

5. Results

5.1 Impact of preprocessing

Normalization
By applying 28 *sed* handcrafted commands to attempt in normalizing the corpus, we achieved

a reduction of 0.058742% with an initial training dataset vocabulary of 164431 to 154772. As well as a reduction of 0.0422% for the test dataset with an initial vocabulary of 94626 to 90633. Cleaning and normalizing the data by using regular expressions through a series of sed commands yielded a slightly lower performance overall for all models. By standardizing certain features believed to convey the same meaning, there is a possibility that important details are lost. Despite reducing the size of the vocabulary, certain informative features could be discarded in the process.

Tokenization
Previously, our tokenization process included lowercasing, removal of non-alphanumeric words, stop words and urls. This method saw substantial decrease in performance for our learning algorithms compared to lowercasing and stop word removal only. For instance, for the Multinomial Naive Bayes algorithm, we went from 57.31% to 56.71% in the validation phase. Similarly to normalization, this can be explained by the loss of information. Thus, we ignored tokenization of comments for the rest of our experiments except lowercasing and stop word removal as they brought increase in validation accuracy of 0.1428% for the multinomial naive bayes model.

5.2 Impact of handcrafted features

The following features' impact were evaluated in the validation phase for the Multinomial Naive Bayes algorithm.

Comment length, word and sentence count
In *Figure 1*, we show an overview of the distribution of the length of comment per category. Given the variation across the groups, the effect of this feature would, in our intuition increase performance. As we tested this feature on different learning algorithms, there were very slight improvements in performance. We also expected that the word count and sentence count would have little to no effect and wouldn't be a feature that would bring more information due to its similarity to the comment length feature as the *Figure 2* and *Figure 3* outlines. As a result of our experiments, the word count and sentence count features did bring a combined minor increase in performance; 0.057% respectively.

Complexity & Reading Difficulty
In *Figure 4*, the Flesch Reading Ease metric illustrates some variance for between the categories, this may indicate that this feature would be useful to include. Also, introducing the Flesch-Kincaid Grade Level, the Smog Index and the polysyllabic words would be redundant because they appear to be producing similar results. Thus, only one should be enough. In our experiments, we measured the increase or decrease of accuracy of these metrics. We obtained a 0.02857% decrease for the Smog Index as well as a 0.02857% decrease for the Flesch-Kincaid Grade Level metric. On the other hand, we gain 0.02857% for the Flesch Reading Ease and also 0.02857% for the polysyllabic words metric.

Swear words
In *Figure 5*, we can see that some categories appear to have an unusually high or an unusually low number of curse words. The impact of this feature on learning algorithms was hypothesized to bring positive result in regards to performance. The experiments revealed a slight boost of 0.028% in accuracy.

Part-of-speech Tagging
Combining the Part-of-speech tags with the tokens as a feature for the supervised learning algorithms resulted in a decrease of 0.1428% in accuracy. This can be attributed to the fact that by introducing context to tokens, it creates sparsity in the feature representation.

5.3 Naive Bayes with handcrafted features

The multinomial naive bayes model with the following handcrafted features: comment length, word count, sentence count, flesch reading ease, word difficulty and swear word count, resulted in a validation accuracy of 57.8% with an *alpha*

of 0.27836 and a test accuracy of 58.633%. Furthermore, in the validation phase, experiments showed that when the hyperparameter *alpha* is too low or too high, e.g. 0.1 and 0.6, its impact on performance was substantial. Notably, we saw a decrease neighbouring 1% in accuracy.

5.4 Logistic Regression with handcrafted features

Similarly as the naive bayes model, we've applied the same handcrafted features for the logistic regression model. The hyperparameter *C* brought comparable pattern in which having *C* too low or too high, e.g. 0.01 and 10, decreased the validation accuracy from 2% to 10%. We've found through optimization, *C* equal to 1.3054 and giving us a validation accuracy of 56.1%

5.5 Neural Network Models

The baseline MLP model could not exceed a validation accuracy of 52%, which is inferior to Naive Bayes. Surprisingly, using word-level transfer learning with Word2Vec embeddings decreased its performance to 42% compared to the baseline.

An LSTM RNN trained using standard Keras vocabulary embeddings achieved extremely poor results because it appeared to be overfitting the dataset, see *Figure 6*. An attempt was made to enhance it with GloVe embeddings while imposing an extreme amount of regularization on the model with as many as 70% of all neurons being randomly turned off, following a similar successful example used on the News Aggregator dataset (see references) . While this addressed the overfitting issue, performance was maximized at 54%, which is inferior to Naive Bayes. Attempts to replace Keras embeddings with Word2Vec did not result in a performance improvement.

5.6 Universal Sentence Encoder

Features extracted using the pre-trained Universal Sentence Encoder resulted in a dramatic performance improvement over any methods using raw TF-IDF word vectors, particularly neural network methods. A fully-connected neural network trained with TF-IDF features achieved a maximum validation accuracy of 52%, whereas a similar neural network trained with USE embeddings achieved around 61.5%. Moreover, by experimenting with the number of units in each dense layer, we obtained slightly higher validation accuracy on average by using more units; e.g. from 128 units to 1024 units.

5.7 Ensembling model

Using a soft voting model comprised of Naive Bayes with engineered features and the USE embeddings model achieved a maximum test accuracy of 63.3%. An interesting result obtained from this observation is that a rather unsophisticated ensembling technique can yield a classifier that yields better performance than either classifier can achieve on its own.

For a modest improvement, an additional enhancement was made to the voting scheme:

$$\alpha_i = \frac{1}{2} \ln \frac{1 - \epsilon_i}{\epsilon_i}$$

α_i = the weight of the ith classifier
ϵ_i = the error rate obtained by the ith classifier on the training set.

The final vote was obtained using the following formula:

$$c_{pred} = arg \max_c \frac{1}{n_c} \sum_{i=1}^{n_c} \alpha_i P_i(y = c)$$

n_c = the number of classifiers
α_i = the weight of the ith classifier
$P_i(y = c)$ = the probability of class c according to the ith classifier.

While theoretically, this formula is more suited towards weak binary classifiers and could benefit from refinement for the general case with

multiclass classification, it produced reasonable weightings in this competition because the error rates of the classifiers was in the same range as what would be expected from weak binary classifiers (50% - 65%).

Also, a third classifier was added to the ensemble: a linear Support Vector Machine. The performance of the final ensembling model on the public test set was measured to be 63.7%

Attempts to use the AdaBoost algorithm with the Naive Bayes optimizer resulted in decreased performance compared to the baseline and were not pursued further.

6. Discussion

6.1 Pros and cons of the approach and methodology

The methodology allows for quick experimentation with and empirical evaluation of machine learning suited for text classification via rapid prototyping against a medium-sized labeled dataset that requires little preprocessing. The scope of the task also allows for some comparison between traditional methods and modern deep learning approaches in a realistic scenario. However, the evaluation of algorithms is limited to performance on that particular dataset, and few conclusions can be drawn on how the methods would compare on text data drawn from a different medium (e.g. scientific articles) or even another discussion forum. Also, no attempt was made to ground the findings in theory beyond intuitive explanations of the empirical findings.

6.2 Exhaustive Hyperparameter tuning

For some of the models previously mentioned, a distinct set of user-defined hyperparameters are tested to find the optimal values. There exists several hyperparameter optimization approaches that could provide a more refined search of the best hyperparameter values. For example, grid search is, for instance, an exhaustive hyperparameter sweeping technique.

6.3 Refined Lexical Normalization Process and In-Depth Analysis

Given the more general scope of this competition, all the various normalization steps were analyzed concurrently. It would be insightful to break down each component and measure the individual impact. For instance, analyzing the effects of lexical normalization operations on a certain set of related words compared to another set could provide a finer and a more detailed understanding of their impact. Essentially, a refined micro-analysis of each component in the normalization process could potentially lead to unveiling new findings.

7. Statement of Contribution

Jean-Pierre Thach: Naive Bayes and Linear Methods, NLP data preprocessing and analysis (lexical normalization, vectorization, tokenization, stemming), fine-tuning deep learning methods, report structure and content, detailed evaluation of results.

Artem Ploujnikov: Exploratory data analysis, experimentation with deep learning methods and transfer learning, ensembling methods, feature engineering (readability statistics, length, use of profanity), contributions to the report.

We hereby state that all the work presented in this report is that of the authors.

8. References

Andrew Y. Ng, and Michael I. Jordan. On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. *23-Sept, 2002*.

Yarin Gal, and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *arXiv:1512.05287*, 2016.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil. Universal Sentence Encoder. *arXiv:1803.11175v2*, *12 Apr, 2018*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 1532–1543.

https://en.wikipedia.org/wiki/Flesch–Kincaid_readability_tests

https://en.wikipedia.org/wiki/SMOG

https://pypi.org/project/textstat/

https://tfhub.dev/google/universal-sentence-encoder/2

https://www.kaggle.com/highflyingbird/swear-words

https://www.kaggle.com/ngyptr/multi-class-classification-with-lstm

9. Appendix

Figure 1 - Comment length

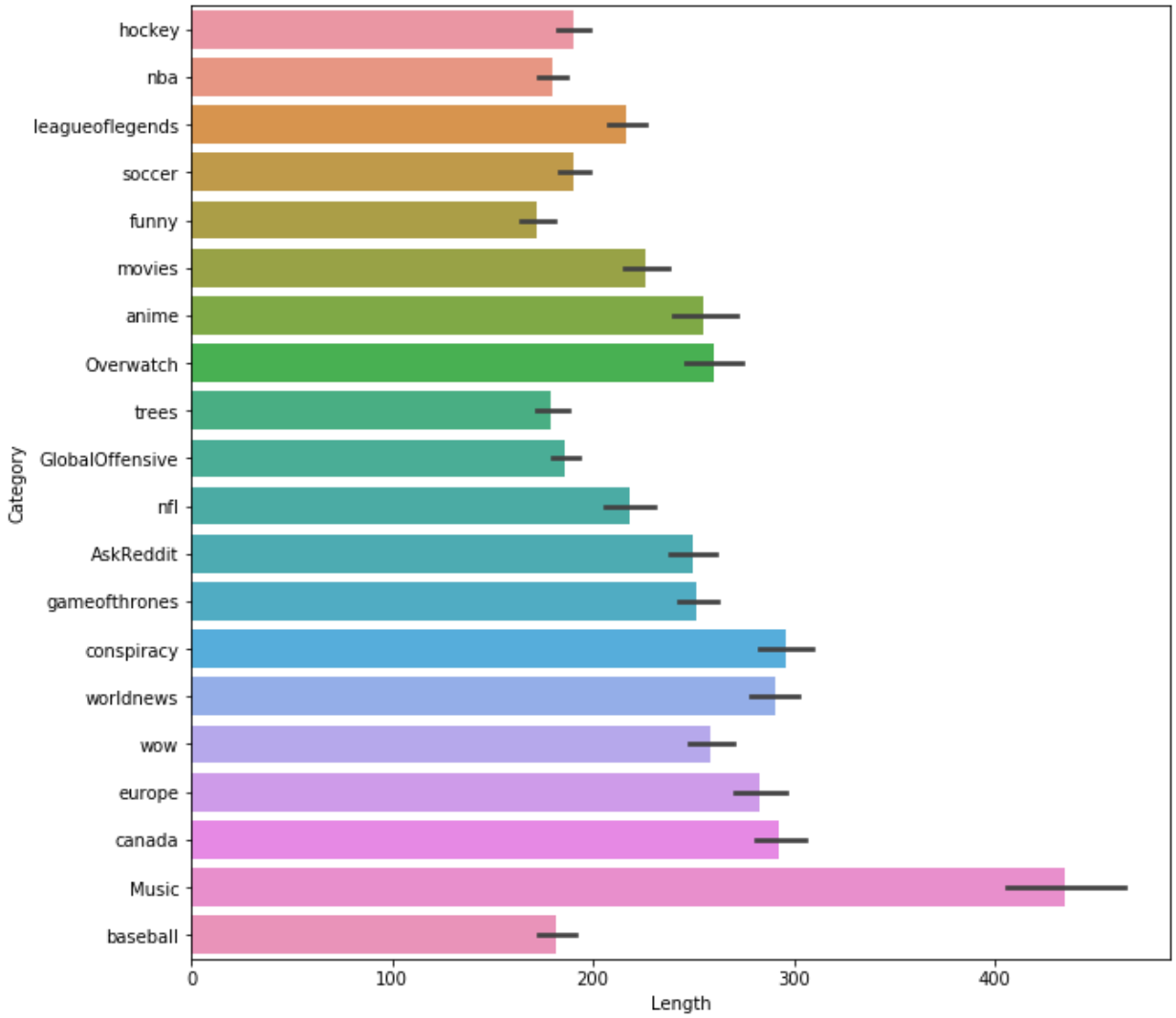

Figure 2 - Word count

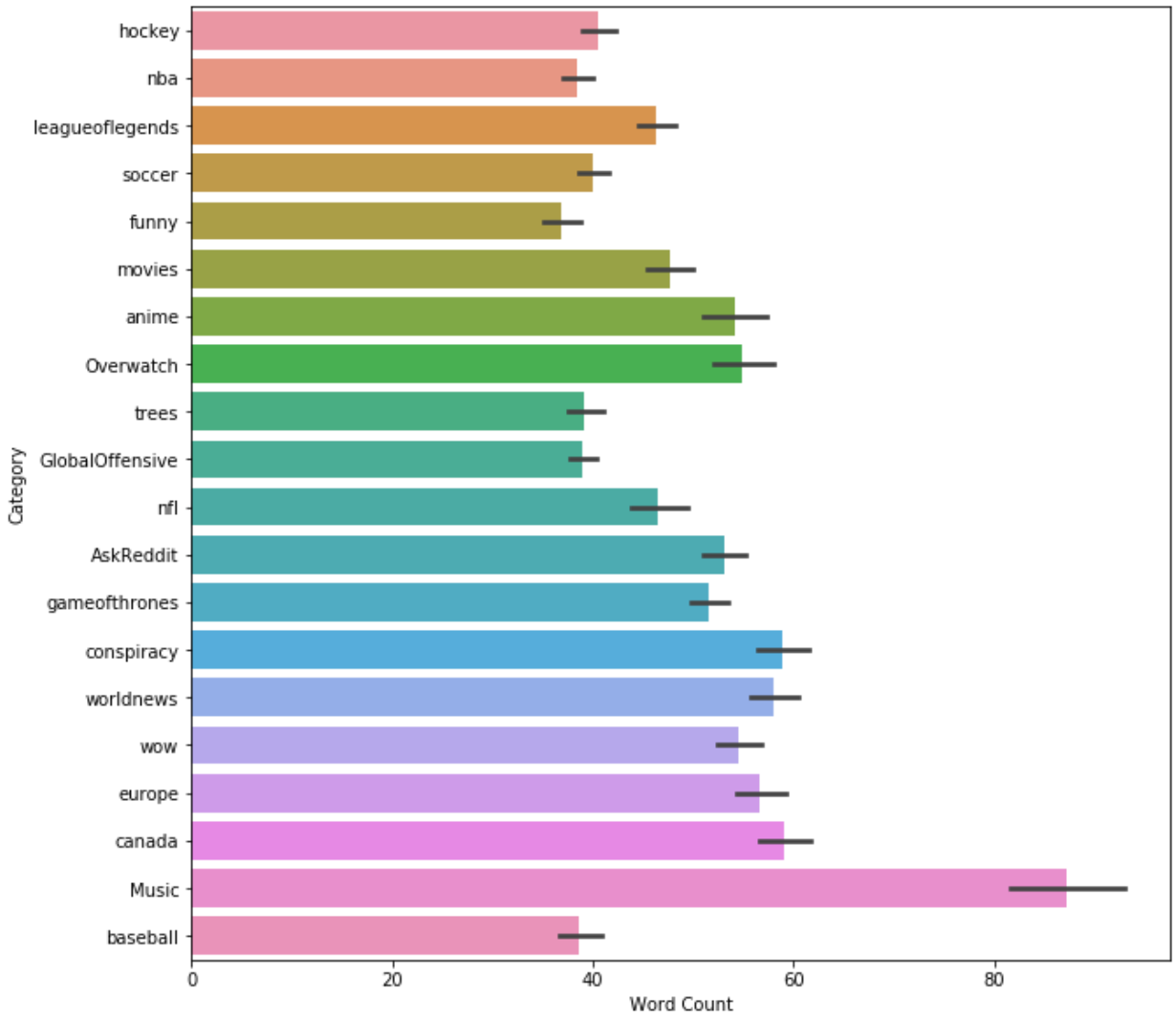

Figure 3 - Sentence count

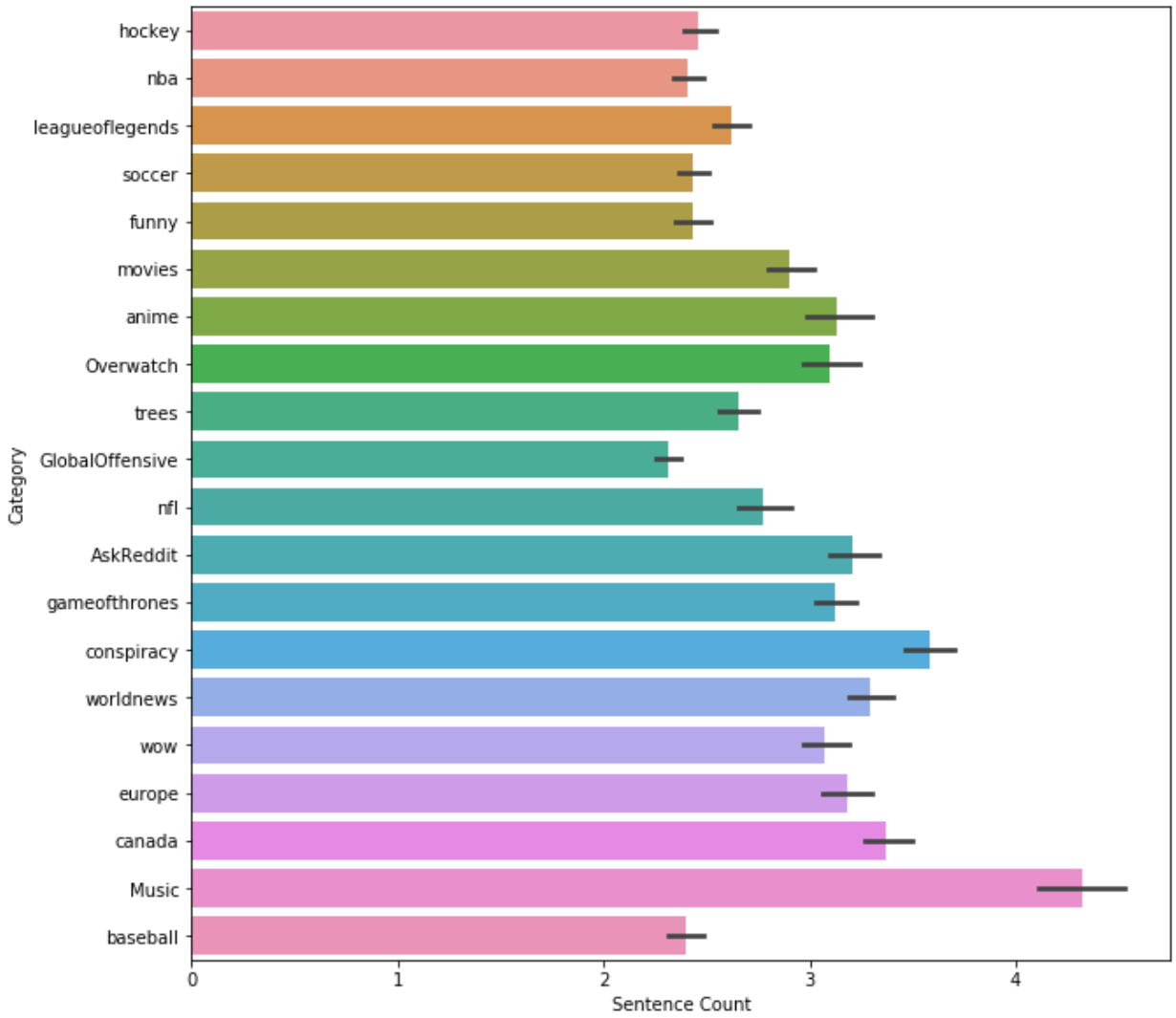

Figure 4 - Complexity and reading difficulty metrics

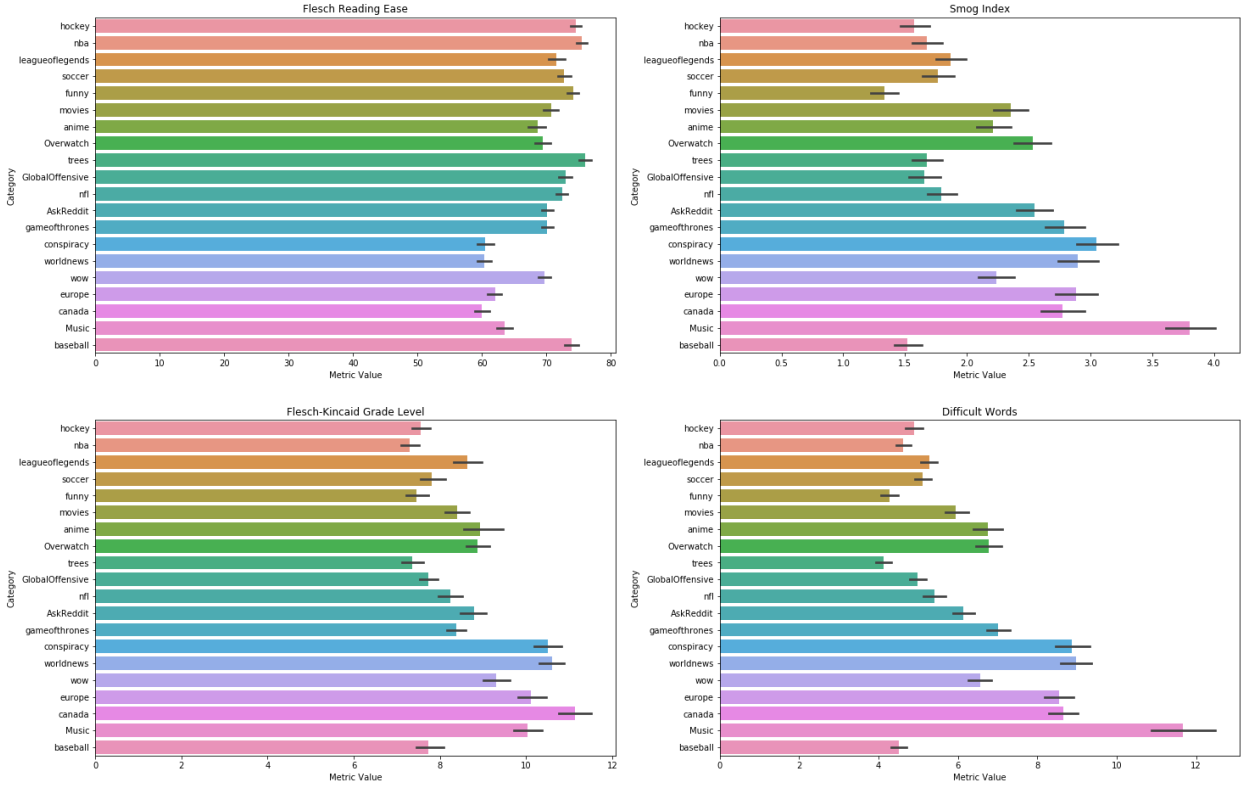

Figure 5 - Swear word count

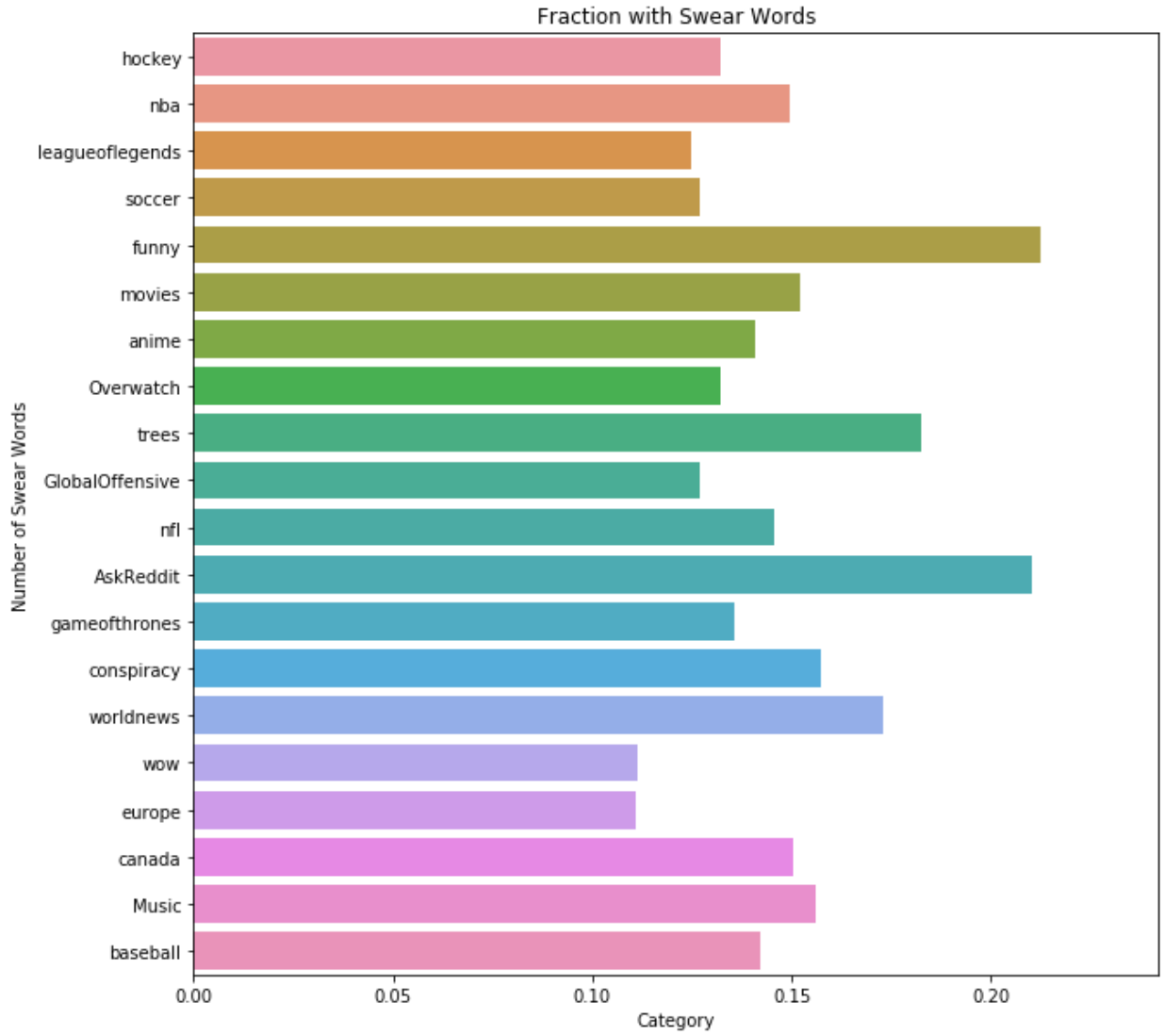

Figure 6 - LSTM Model

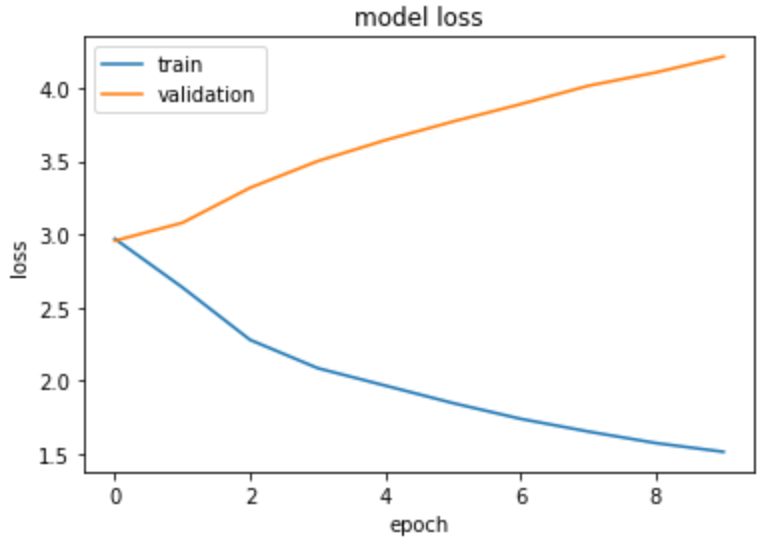

Figure 7 - Universal Sentence Encoder model (2 dense layers with 1024 units each)

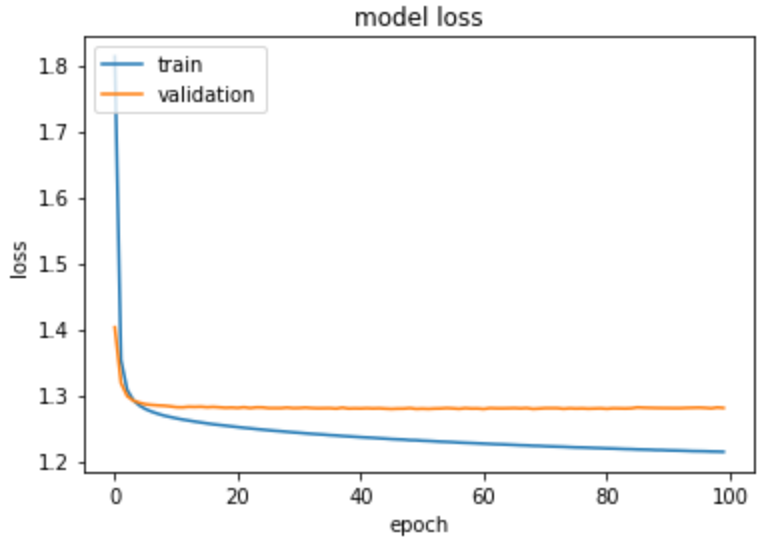

Figure 8 - Ensembling model (Universal Sentence Encoder & Multinomial Naive Bayes)