Reddit Comment Classification by Team Nicoine

Marc-Antoine Provost, Nicolas Trudel-Mallet

November 15, 2019

1 Introduction

Reddit is a social media website where users can anonymously join and participate in various communities (”subreddits”) centered around a specific topic (a country, a video game, etc.). Users can create *posts* in these communities and can leave *comments* on these posts. In this project, we attempt to create a model that can predict which subreddit a comment originates from. This is a supervised, multiclass, text classification task with 20 labels corresponding to 20 different subreddits.

We begin by assuming that words in a comment are independent from each other and build a Naive Bayes classifier with smoothing. This model beat all three baselines with a score of 0.55944. We then try two other models: a multilayer perceptron (0.59444) and logistic regression (0.55057).

2 Feature Design

Each comment is given to us as a string. We performed the following preprocessing steps on each comment.

Tokenization. We parse the string into a list of words to facilitate manipulation. More specifically, we used a Regex Tokenizer which looks for strictly alphanumeric sequences of characters; each of these is considered a ”word” in our model. This has the added effect of removing symbols and punctuation, which generally do not carry significant information about the source subreddit.

Word normalization. Where applicable, we transform each word into its etymological root using lemmatization. This relies on the assumption that variants of a root word provide the same (or close to the same) information as the root word itself. This reduces the number of unique words, ultimately reducing the dimensionality of the data.

Lowercase. We turn each word into lowercase characters only. This relies on the assumption that capitalization does not carry a significant amount of information for this task. Similarly to lemmatization, this step reduces the dimensionality of the data by decreasing the number of unique words.

Stop-word removal. It is generally safe to assume that small words such as articles, conjunctions, and basic verbs like "be" carry little to no information. As such, we remove these words from the list.

Feature selection and representation. All unique words were used as inputs to the models, with no limit on the number of features. For the Naive Bayes model, we use the Bag-Of-Words representation: we store each unique word along with its frequency. For the other two models, we use a TF-IDF matrix.

3 Algorithms

3.1 Naive Bayes

A Bayes classifier uses Bayes' rule along with the estimated density of each class. For a given test point x, it predicts the class c with the highest posterior probability $P(c \mid x)$ using a slightly altered Bayes' rule:

$$P(c \mid x) \propto P(x \mid c)P(c)$$

The classes in this task are equiprobable (they all have 3500 comments), and so it suffices to calculate the likelihood. Since we assume that the components are independent, we can calculate the likelihood as follows:

$$P(\mathbf{x} \mid c) = \prod_{k=1}^{d} P(x_k \mid c) = \prod_{k=1}^{d} \frac{n_k}{n}$$

To avoid having zero probabilities for unseen words, we add a smoothing parameter α:

$$P(x_k \mid c) = \frac{n_k + \alpha}{n + \alpha|V|}$$

where $|V|$ is the total number of unique words.

3.2 Multilayer Perceptron

A multilayer perceptron is a simple feedforward neural network and an extension of Rosenblatt's perceptron. As the name implies, a multilayer perceptron adds one or more *hidden layers* between the input and output layers. Each *node* in the hidden layer(s) learns a linear classifier wrapped by a non-linear *activation function.* The one used in this task is the rectified linear unit, or ReLU:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We used a single hidden layer for our model, as this architecture can approximate any continuous function given enough nodes[1]. We did not see any significant increase in performance when increasing the number of hidden layers.

3.3 Logistic regression

Multinomial logistic regression is a classification algorithm that assigns observations to a discrete set of classes using multiple binary classifiers, following either a one-vs.rest or a one-vs.-one approach. It transforms its output using an activation function, the softmax function, which returns a discrete probability function over the k classes. The softmax function can be defined as

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\Sigma_{j=1}^{k} e^{x_j}}$$

The loss function optimized by gradient descent in multinomial logistic regression is the *cross-entropy* loss that measures how different probability distributions are to each other.

4 Methodology

The training and test sets were created using an 80/20 split of the data. Models were fit on the training set, and hyperparameters were selected based on the test set. Generalization performance was measured using the public Kaggle score. The random seed was set to 6390, assuring a shuffled yet consistent split.

4.1 Naive Bayes

For the Naive Bayes classifier, the only hyperparameter is the smoothing parameter α used when calculating the conditional probabilities (Section 3.1). While we had initially intended to perform a grid search over α, we found that performance very quickly decreases as we move away from the value of 1. As such, for our Naive Bayes classifier, $\alpha = 1$.

Instead of using a term-frequency, inverse-document-frequency (TF-IDF) matrix, we created three lists for each of the 20 classes:

- *words*: all the unique words present in the class;
- *counts*: the number of occurrences of word i in the class;
- *probs*: log of the probability of word i calculated using the formula above.

This format was more intuitive for us to work with when building our classifier from scratch. While it did make training and testing the model relatively slow, we were still able to obtain good results in a reasonable amount of time.

[1]https://en.wikipedia.org/wiki/Universal_approximation_theorem

4.2 Multilayer Perceptron

Our first hyperparameter selection was a manual search over the number of neurons h in the hidden layer. We initially found that $h = 40$ provided consistently good results. We later changed to $h = 200$ when altering the learning rate.

With further testing, we discovered that accuracy decreases after only a few epochs. Since the training error remains very high while the test error decreases, this is likely due to overfitting (Section 5, Fig. 1). One way to control this is by early stopping, where we manually set the maximum number of iterations. We find the optimal number of iterations using a simple grid search.

We shifted our attention to the learning rate, which is a hyperparameter that controls the speed at which the model learns. Generally speaking, when the learning rate is too large, gradient descent can inadvertently increase rather than decrease the training error. When the learning rate is too small, training is not only slower, but may become permanently stuck with a high training error.[2] Therefore, we should not use a learning rate that is too small or too large, but one that will produce the best generalization on out-of-sample data. We did a grid search of the learning rates on a logarithmic scale, from 10^{-1} to 10^{-5}.

4.3 Logistic regression

For the multinomial logistic regression, we did a grid search over the C parameter, which is the regularization term over the L2 penalty. As implemented in scikit-learn, the C parameter is the inverse of the regularization strength, meaning that a smaller C specifies stronger regularization, which will heavily penalize large weight coefficients. We decided to do so in order to prevent overfitting and improve the generalization of the model on unseen data.

5 Results

The following table compares the best scores obtained using the three models on the public Kaggle dataset. h is the number of nodes in the hidden layer, lr is the learning rate, and i is the number of iterations:

	Naive Bayes	**Multilayer Perceptron**	**Logistic regression**
Score	0.55944	0.59444	0.55057
Hyperparameters	$\alpha = 1$	$h = 200$, $i = 3$, lr $= 0.0005$	$C = 1.5$

[2] https://www.deeplearningbook.org/

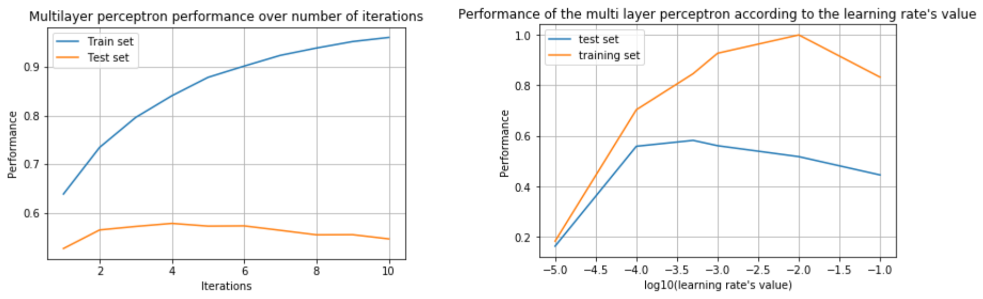

Figure 1: Grid search over number of iterations (left, $h = 40$) and learning rate (right, $h = 200$) for the multilayer perceptron. Performance is measured by the model's accuracy.

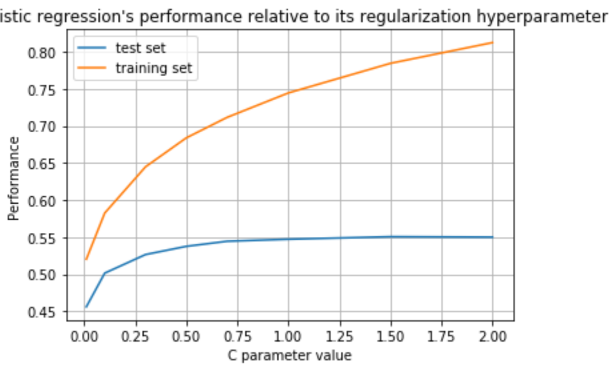

Figure 2: Grid search over C for the logistic regression model. Performance is measured by the model's accuracy.

6 Discussion

6.1 Feature Design

As mentioned in section 2, the advantage of using a Regex tokenizer over simply delineating words by spaces is the added effect of stripping punctuation and symbols, which generally do not carry much information on the source subreddit. However, we also ignore URLs, emoticons, and other such character sequences that may be useful. While the Regex tokenizer ultimately performed better in our tests, it may be worth considering URLs and/or emoticons as special cases and include them.

While we used lemmatization in our models, we could have used stemming instead. The main difference is that lemmatization performs a dictionary lookup to find the etymological root of a word, whereas stemming does not. This makes lemmatization slower, but more accurate (e.g. stemming cannot turn "better" or "best" into "good"). We opted for lemmatization as we preferred the increased accuracy over the shorter pre-processing.

One way to potentially improve performance is to use *topic modeling*, an unsupervised task which clusters similar words based on their topic[3]. It would be interesting to analyze model performance after implementing such an algorithm and adding its output to our features.

6.2 Algorithms

Text classification tasks generally have very high dimensionality due to the large amount of unique words. Consequently, Naive Bayes classifiers have an advantage over regular Bayes classifiers: since we assume that the components of an example are independent, we can estimate univariate densities rather than multivariate densities. This largely eliminates the curse of dimensionality. In exchange, Naive Bayes classifiers have a much lower capacity.

On the other hand, multilayer perceptrons have a very high capacity. With this more complex model, we were able to capture the intricacies of the data and obtain our best score of 0.59444. However, it was also the slowest model to train. Moreover, due to its large number of learned parameters, it is very likely to overfit the data: strict regularization methods such as early stopping are critical.

One way to improve our score could be to use pre-trained models, such as BERT or XLNet. These deep learning models are specifically built for text classification and can deliver state-of-the-art performance on certain tasks.

7 Statement of Contributions

We both worked on building the Naive Bayes model, with Nicolas doing more of the programming (e.g. creating the bags of words) and Marc-Antoine working on the model itself (e.g. implementing smoothing correctly). Marc-Antoine worked on the logistic regression model. The multilayer perceptron was implemented and optimized by Nicolas, except for the final learning rate tuning. Data analysis and visualization was mainly done by Marc-Antoine. Nicolas wrote most of the report except for sections 3.3 and 4.3, as well as the last part of section 4.2.

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

[3]https://en.wikipedia.org/wiki/Topic_model

8 References

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

www.deeplearningbook.org

https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

https://scikit-learn.org/stable/modules/feature_extraction.html#customizing-the-vectorizer-classes

https://stackoverflow.com/questions/15547409/how-to-get-rid-of-punctuation-using-nltk-tokenizer