Classifying Reddit Posts With Natural Language Processing and Machine Learning

Exploring text transformation and the classification modeling process.

Britt Allen
Jan 8, 2019 · 6 min read

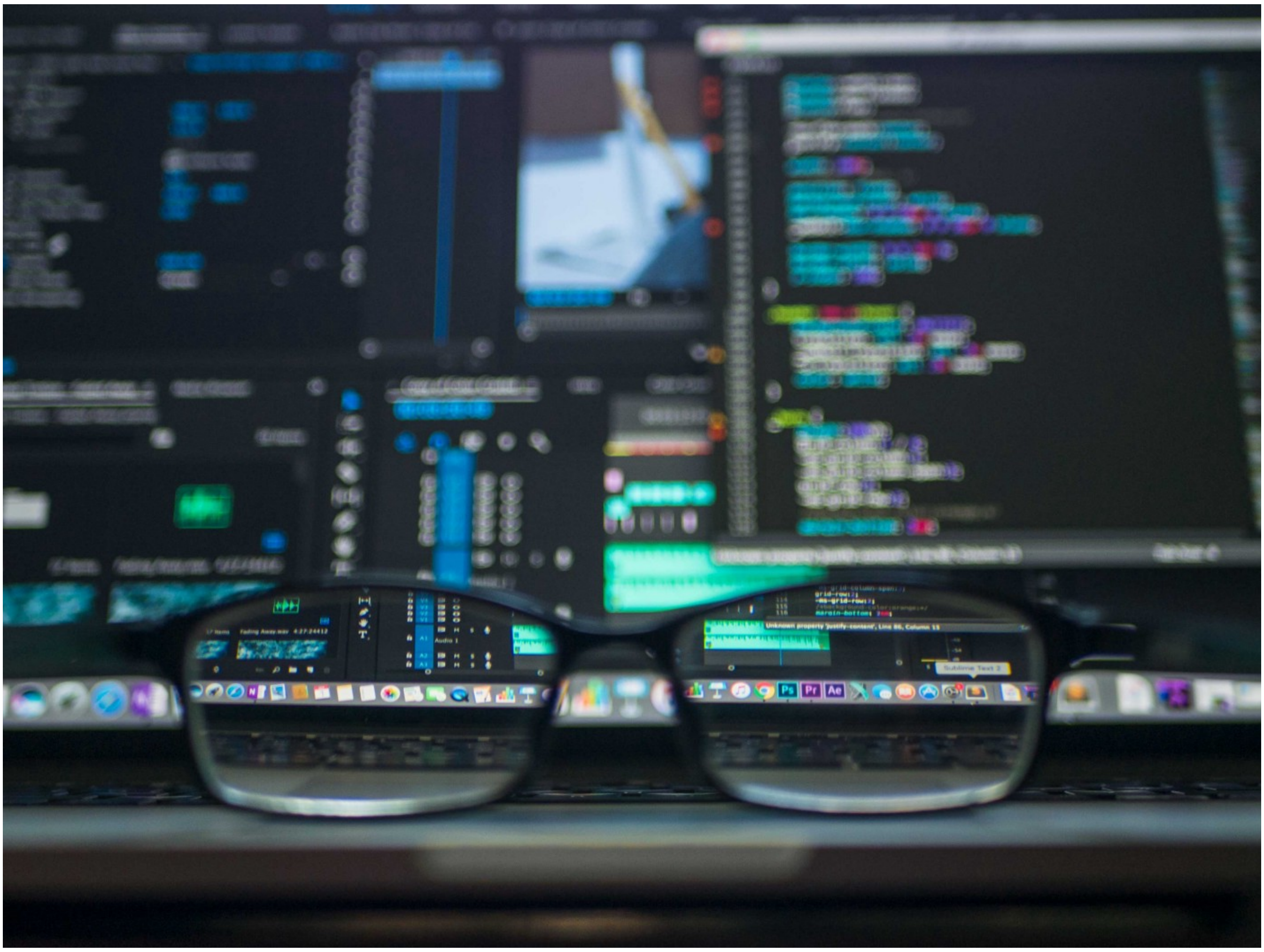


Photo by Kevin Ku on Unsplash

In my last post I walked you through my data science process for using machine learning to predict home prices (below).

Using Machine Learning to Predict Home Prices

In this post I will walk you through my data science process for using machine learning to predict home prices. Before…

towardsdatascience.com

In this post I will walk you through the same process, but for using Natural Language Processing (NLP) and classification modeling to classify Reddit posts from r/BabyBumps and r/menstruation. Before I begin, let's recall the data science process (outlined below) followed by a fun ice breaker!

- Define the problem
- Gather & Clean the data
- Explore the data
- Model the data
- Evaluate the model
- Answer the problem

. . .

Ice Breaker!

Can you guess which subreddit the posts (pictured below) came from? Your options are **r/BabyBumps** and **r/menstruation**. Share your guess in the comments!

Define the Problem

As you may have guessed I was tasked with using machine learning to do what you just tried to do above! In other words, **creating a classification model that can distinguish which of two subreddits a post belongs to**.

The assumption for this problem is that a disgruntled, Reddit back-end developer went into every post and replaced the subreddit field with "(͡° ͜ʖ ͡°)". As a result, none of the subreddit links will populate with posts until the subreddit fields of each post are re-assigned.

As you may have gathered, posts in r/BabyBumps and r/menstruation will definitely have a lot of crossover. For example, think about women talking about food cravings, cramps, or mood swings in either channel. I like a challenge so I purposely picked two closely-related subreddits as I wanted to see how well I could leverage natural language processing and machine learning to accurately re-classify the posts to their respective subreddit.

**The answer to the ice breaker can be found in the last sentence of the following section, but keep reading to see if you're smarter than the algorithm I built!*

Gather & Clean the Data

My data acquisition process involved using the `requests` library to loop through requests to pull data using Reddit's API which is pretty straightforward. To get posts from /r/menstruation, all I had to do was add .json to the end of the url. Reddit only provides 25 posts per request and I wanted 1000 so I iterated through the process 40 times. I also used the `time.sleep()` function at the end of my loop to allow for a one second break in between requests.

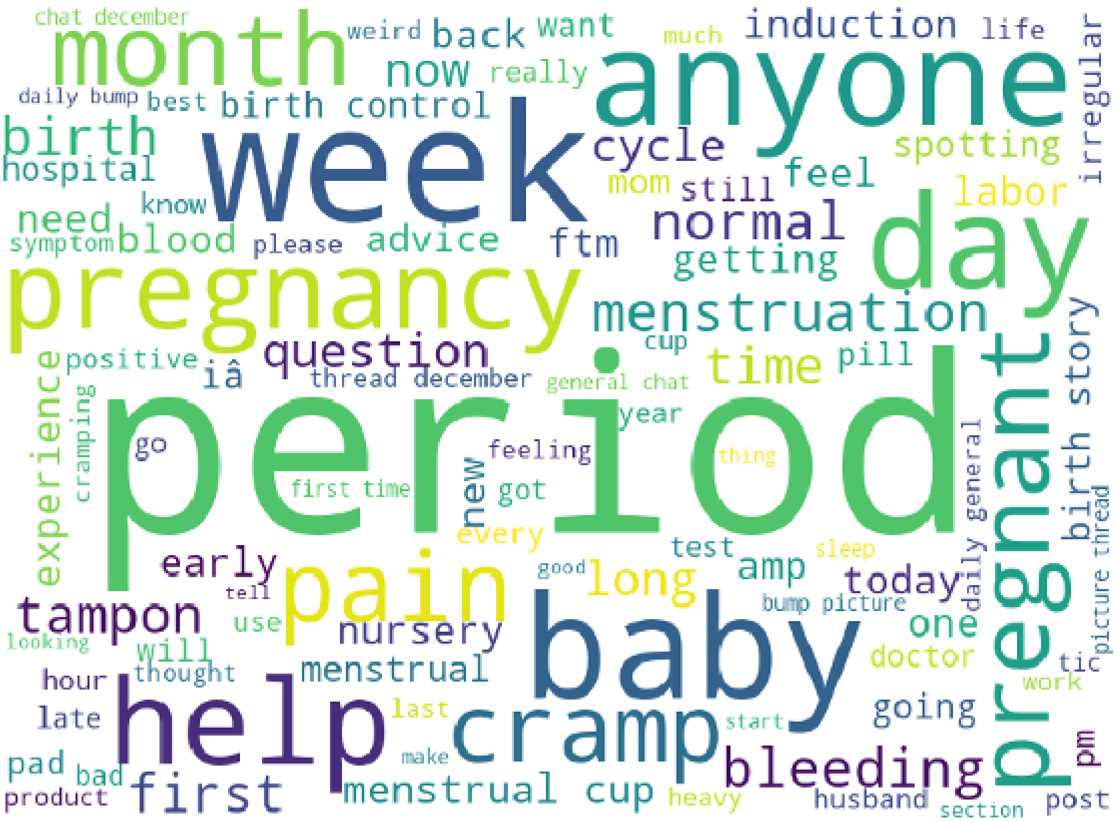
My for loop outputted a list of nested json dictionaries of which I indexed to pull out my desired features, *Post Text* and *Title*, while simultaneously adding them to two Pandas DataFrames one for r/BabyBumps-related posts and the other for r/menstruation-related posts.

After getting my posts in their respective DataFrames I checked for duplicate and null values, both of which occurred. For duplicate values I got rid of them by utilizing the `drop_duplicates()` function. Null values only occurred in my *Post Text* column, this happens when a Reddit user decides to use only the title field. I decided not to drop null values as I did not want to lose valuable information in the accompanying rows of my *Title* feat so I filled the nulls with unique and arbitrary text instead.

After cleaning and concatenating my data, my final DataFrame contained 1818 documents (rows) and 3 features (columns). The third feature was my target, which had a balance of classes of 54% for class 1 (r/BabyBumps) and 46% for class 0 (r/menstruation) — *ice breaker answer!*

Explore the data

I created a word cloud because they're fun and we're working with text data!



This word cloud contained 100 words from both subreddits. I generated it to get a visual understanding of the frequencies (bigger/bolder words have higher frequencies) of words and how their commonality across subreddits might throw my model off; or how it may work in my model's favor if it's a word/phrase like "menstrual cup" which has a medium frequency and likely only appears or mostly appears in r/menstruation posts.

Model the data

I began my modeling process by creating my *X* and my *y* and splitting my data into training and test sets. I then moved on to my feature engineering process by instantiating two CountVectorizers for my *Post Text* and *Title* features. CountVectorizer converts a collection of text documents (rows of text data) to a matrix of token counts. The hyperparameters (arguments) I passed through them were:

- stop_words='english' (*Post Text* & *Title*)
- strip_accents='ascii' (*Post Text* & *Title*)
- ngram_range=(1, 6), min_df=.03 (*Post Text*)
- ngram_range=(1, 3), min_df=.01 (*Title*)

Stop words removes words that commonly appear in the English language. Strip accents removes accents and performs other character normalization. Min_df ignores terms that have a document frequency strictly lower than the given threshold.

An n-gram is just a string of n words in a row. For example, if you have a text document containing the words "I love my cat." — setting the n-gram range to (1, 2) would produce: "I love | love my | my cat". Having n-gram ranges can be helpful in providing the models with more context around the text I'm feeding it.

I assumed that setting the *Title* feature with an n-gram range of (1, 4) would clean up noise and be more helpful to my model by adding more context than if just left alone. I still set a gentle min_df to help clean up any additional noise. I made similar assumptions for my *Post Text* feature, although I gave it a higher n-gram range since post texts tends to be lengthier.

This resulted in 393 features which I fed into two variations of the models listed below. I built four functions to run each pair of models and Gridsearched over several hyperparameters to find the best ones to fit my final model with.

- Logistic Regression

The difference in variations were the *penalty* and *solver* parameters. The 'newton-cg', 'lbfgs' and 'sag' solvers only handle L2 penalty (ridge regularization), whereas 'liblinear' and 'saga' handle L1 (lasso regularization).

- Decision Trees & Random Forests

The difference in variations for these two models was the *criterion* parameter. One was set to 'gini' (Gini impurity) while the other was set to 'entropy' (information gain).

- Multinomial Naive Bayes

The difference in variations was the *fit_prior* argument which decides whether to learn class prior probabilities or not. If false, a uniform prior will be used. One was set to True, while the other was set to False.

Evaluate the model

My second Multinomial Naive Bayes model performed the best. With the best parameters being — alpha=0 and fit_prior=False. The accuracy score was 92.4% on training data and 92.2% on unseen data. This means our model is slightly and probably inconsequentially overfit. This also means that 92.2% of our posts will be accurately classified by our model.

Answer the problem

Considering the small amount of data gathered and minimal amount of features used, the Multinomial Naive Bayes model was the most outstanding. It handled unseen data well and balanced the tradeoff between bias and variance the best among the eight models so I would use it to re-classify reddit posts.

However if given more time and data to answer the problem I would recommend two things: 1) spending more time with current features (e.g. engineering a word length feature) and 2) exploring new features (e.g. upvotes or post comments).

. . .

Check out **my code** (which I split into 3 Jupyter notebooks — collection, cleaning/exploration, and modeling — for readability and organization purposes) and **my presentation**. As always, please comment feedback and questions. Thanks for reading!

Machine Learning Naturallanguageprocessing Algorithms Reddit Data Science