Data Mining Lab 2 Report

113062641 王佑禎

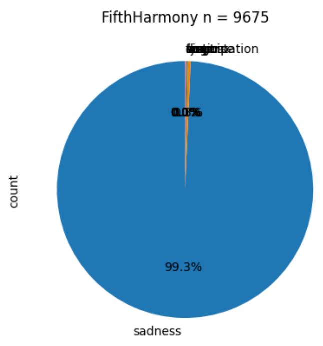
0. Observation + Preprocessing:

After making the data in proper pandas dataframe format, I was curious about the “hashtags” part of the data, why hashtags and why should it help since the hashtags are supposedly more random and sparser, so I go check for the @ part of the data (weird conclusion but that’s what I decided.)

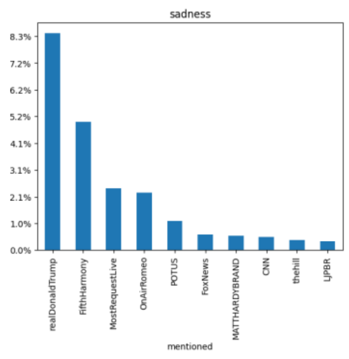
A. Mentioned people:

The reason I go for this first is because that I noticed that the mine date of these data is from 2015~2017, which is kind or close to USA’s president election season, so there must be a lot of emotion associated with each candidates / organization right?

After mining all tweet’s mentioned people and plot them out, I found out that indeed there’re some mentions that strongly associates with some emotion, and there are some emotion that strongly associate with some mentioned individual.



For the second most mentioned individual, 99.3% of the tweets mentioning it is labeled “sad”



For the sadness category, 8.3% of the tweets mentioned “Donald J. Trump”

By the above observation, I decided to get the 25 most mentioned account (roughly chosen by the percentage of tweets that proportion to the top-n mentioned people) and make them a new binary label. I also use “tweepy” package to transform those account’s id to their account name, since I was going to use pretrained transformers to build the classifier, and having the one-word account id is more likely to confuse the model.

B. Emojis:

Then, I figured that emoji would certainly have the same impact on the tweet's emotion, and since it also is a problematic token for the Bert model I was going to use, I decided to analyze them next.

I first use the emoji package to mine out the emoji used in each tweet, and do similar analysis like what I did to mentioned account. The result is similar, but the emojis seem more random and sparser than the mentioned accounts. And since emoji is a much more complicated concept for Bert to realize, and their name usually has slight difference from what they're used for (😂 is called face with tears of joy, but we use it as a reaction to something funny, not joyful.) Also, there are emojis with specified skin-colors, which will add more unnecessary complexity to the tweet, so I decided to "Demotize" those emoji with high enough appearance and seem to have a more dominating impact on the emotion of the associated tweet and simply remove the rest emojis.

C. Hashtags:

We're finally here. After dealing with the first two things I'm interested in, I looked at the hashtags label. Not to my surprise, this data is indeed more random, overall has less tweets including them, and has less impact on the emotion of the tweet. Since the hash tagged things are quite easy to transform into words Bert can read, I simply choose the top 200 hashtags to build a binary vector for each tweet and add a space between the # and the tagged word.

1. Doing the datamining fashion:

After all the above, I decided to do the thing we're taught to do in this course: Do some datamining. I created term document for each emotion, use PAMI to build database and mine the patterns using FPGrowth. Try to use dimensional reduction to observe the data cluster but has no valuable findings. Lastly, I tried a naïve Bayes model but got a terrible validation score. The above process took a "HUGE" amount of time, since the original data is very large, and I don't think we should sample those data (it may change the word frequency and other stuff.) The result turned me down a lot for this method, and I then moved on to the next thing without looking back.

2. Trying to utilize the Mask token of Bert model:

   As all CS students would most likely know, the Bert model is trained in predicting the result of what the original word should be when it was replaced with a MASK token. And that’s where this idea comes from: Why don’t we use an instruction to make it predict the word for us just like using an LLM? In this attempt, I simply add an instruction: " This makes me feel so [MASK]" to the end of every single tweet and observe the output word. Probably to only my surprise, this does not work at all. Not only does the Bert model for MASK prediction not provide embedding for word, but also the output really has no pattern. The same combination of the top 5 possible words could appear for both sadness and joyful, and there are some outputs that are not even adjectives. But I didn’t give up, I still trained a simple classifier using the logits Bert provides to predict emotion. The result is comparably bad with the NB model. They utilize CLS token for a reason after all.

3. Using the Bert model in the intended way:

   I sampled 10% of the data to tune a Bert-base-uncased pretrained model and a 4-linear-layer classification head. Since it has some knowledge about words and context, I was less worried about it having bad result due to sampling. At the time I was aiming for the top three of the leader board, which needs the f1 score of around 0.53 at that moment, but the model seems to be struggling at around 0.5, so I know at least using only the processed text data and 10% of the data isn’t enough for my goal.

   The improvement I tried to implement is the sample 10% of the data part. Since the text data seems to be complex enough to overfit, adding more data should fix the issue. Instead of using the entire dataset to train, I still only use 10% of the data each epoch, but I create 10 data loaders so that for every epoch the fraction of data used for training will change. This is good for two reasons: First, I don’t need to wait for the model to train for 10 years to get the result. Second, by monitoring the result every 1/10 epoch (if we see training with the whole dataset as one epoch), it should be easier for me to find out when it is overfitting and when the model is learning general knowledge. My best validation score is 0.53, which hits right on my goal, but there seems to be some problem with my sampling, making the model see the validation data for a small period, resulting in slight overfitting, so the public score on Kaggle ended up being at around 0.49.

4. Utilizing the binary data from prior processing:

   With the help of ChatGPT, I kind of fully understand how to utilize the binary data I built in the neural network model. Also, this is around the time when a kind person publishes their code onto Kaggle, giving me the insight that tf-idf with basic random forest classifier could yield a decent accuracy at nearly 0.5, so I also add that in. With the help of the additional labels, my validation score jumps to nearly 0.6, but since I still didn't fix the problem (rather I didn't realize it), the Kaggle score only improved by 0.2, resulting in 0.51 public set score.

5. The final method: Ensemble:

   After all the above methods, I finally trained each of the models (in addition to a XGB and RF model) with one same training set and use a NN layer to act as a "attention mask" to learn which model to choose given the TF-IDF and the binary values as information. I was expecting this to do the best, but this come out not perform better than the one using the BERT model with TF-IDF data.

6. Things I learned:

   Aside from the limit of data mining, the biggest thing I learned in this lab is memory management and time complexity of various datatypes in python. I'd like to share but only if any of the TAs or teachers are interested. Otherwise this report will be even more terribly long.