CSC 380 Final Project, Spring 2023

Instructor: Kyoungseok Jang

To goal is to participate in the Kaggle competition
https://www.kaggle.com/competitions/nbme-score-clinical-patient-notes/discussion
in order to

- get familiar with Kaggle competition
- experience workflow of data science/ML
- experience/practice natural language processing
- have fun (hopefully).

Key dates

- Final project report due: 5/5 11:59pm (gradescope)

Evaluation

- Each problem is worth 10 points
- Students with an exceptionally good final submission score will earn extra 2 points (it will be added to the participation score in D2L though, just for convenience).

Submission

- Entry "Final Project": Make a pdf submission, as usual, locating the answers. Be sure to show the code in the pdf and the answers.
    - Make sure you clearly separate the code part from the answer part. Report the answers per each subproblem with explicitly written subproblem identifiers (e.g., problem 7. (a) and then provide your answer). Points will be deducted otherwise.
- Entry "Final Project Code": Zip all the codes you used and submit them.

Problem 1. Familiarize yourself with the NBME task

Visit https://www.kaggle.com/competitions/nbme-score-clinical-patient-notes/discussion and click on the 'Overview' tab to read the description, evaluation, timeline, etc.

Click on 'Data' and try to understand the data and how the prediction on the test data must be made. If you do not understand it, please ask in Piazza. If you understand it, please try to help your peers by answering questions in Piazza.

'Leaderboard' is where you can see how well people are doing. NOTE: the competition is over already, so all you see is how people have done it in the past.

'Rules' has relevant rules.

For this problem, there is nothing to report. Just explicitly acknowledge that you have read it. I will trust you that you have read it carefully.

Problem 2. Follow tutorial

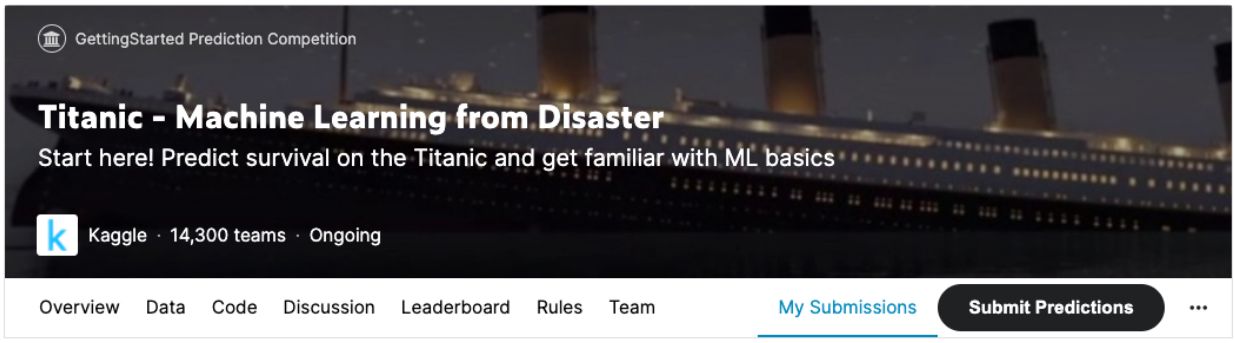
To help people get familiar with Kaggle, they have made a tutorial for an open-ended competition on the Titanic dataset:

https://www.kaggle.com/code/alexisbcook/titanic-tutorial/notebook

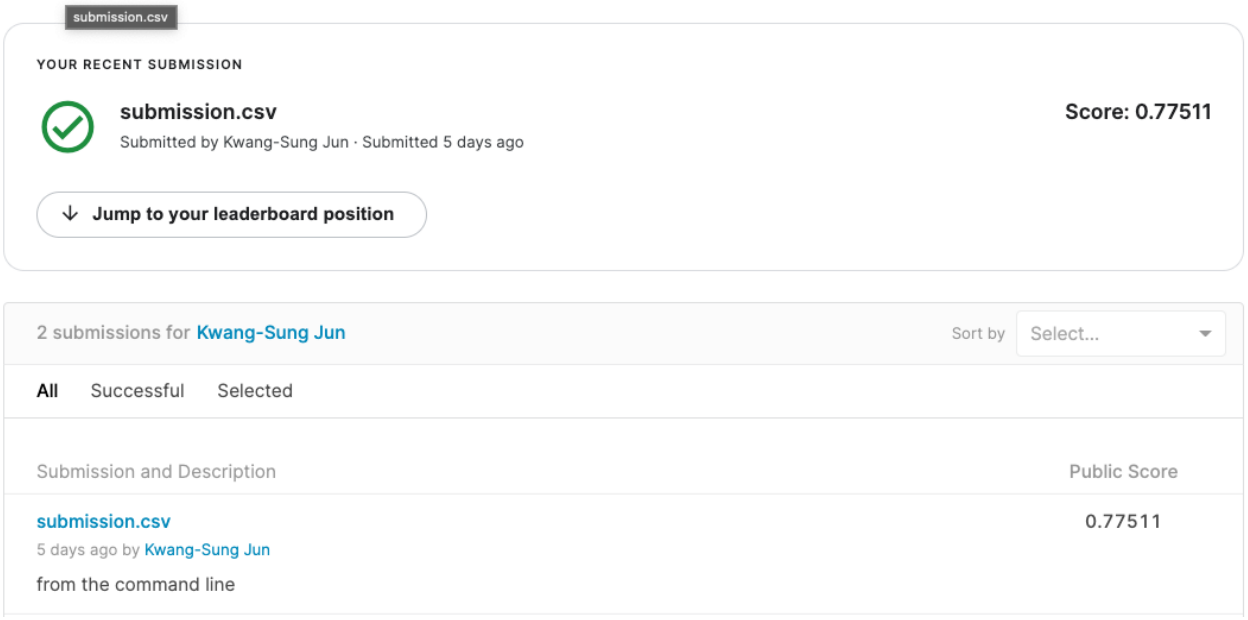
Please go carefully follow the tutorial and try it by yourself.

REPORT:

Click on 'my submissions'



Then, take a screenshot of your submission. It must contain your name somewhere.



Problem 3. Make a dummy submission to NBME

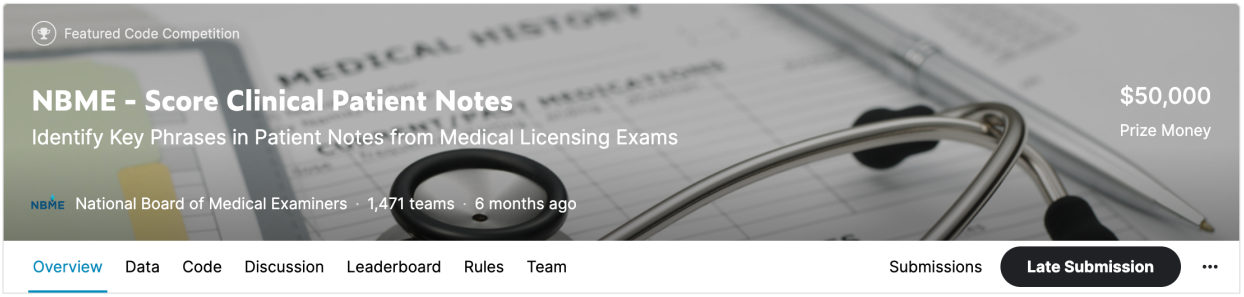
Let's now come back to NBME. We will make a dummy submission to ensure we understand how the submission should be made because the submission here is a bit different from Titanic.

The main difference is that NBME does not reveal the patient notes for the test set (for Titanic, you knew which ones were the test point). This is because if people knew the patient notes for the test set, they could just cheat and manually mark the answers (based on their own knowledge of the task).

So, the way it works is that you work with the given data from the competition to build a model and write a script that takes in the test cases from test.csv and writes the answers into 'submission.csv'. Now, when you submit your code, they will run your code with their own version of test.csv and then measure the score for you.

As you have done for Titanic, create your notebook in Kaggle for NBME, read test.csv, place dumb predictions like "0 10; 12 14", and write it out to the final "submission.csv".

NOTE: Unlike Titanic, we do not have the button "Submit Predictions". Instead, we have a "Late Submission" button, as shown below. This is because the competition is already over. Your score will not be on the leaderboard, but you can still make a submission and check the score.



REPORT:

- Your code (notebook)
- The screenshot that shows your score

Problem 4. Understanding NBME data

For this problem and on, I highly recommend that you do it in your local environment, where you are likely to have more control over the environment, package installations, and development tools.

In this problem, we will get some statistics and visualization so we understand NBME data better.

First, let us understand the distribution of 'annotations.' The file train.csv kindly has the column 'annotation'. We like to know how many words we have for each annotation. Note that each value in the column 'annotation' may have more than one phrase like ['intermittent episodes', 'episode'].

(a) Let's sweep through train.csv and gather all annotation phrases.

REPORT:

- How many phrases do we have in total?

- A plot for the histogram of 'the number of characters of a phrase' (basically, the distribution of the string lengths).
- Show the top-5 phrases with the largest number of characters.

(b) We also wonder what the distribution of the number of words in the phrases is.

REPORT:

- A plot for the histogram of 'the number of words in a phrase'. (you can use split() function in python)
- Show the top-5 phrases with the largest number of words.

(c) How about the fraction of text that is annotated (at the character level) for each pair of (case_num, feature_num)? For case_num=c and feature_num=f, the fraction of text means

$$\frac{\sum\_\{\text{i: patient note in case\_num} = \text{c}\}\ (\#\ \text{of characters that are ever annotated as feature num f in note i})}{\sum\_\{\text{i: patient note in case\_num} = \text{c}\}\ (\#\ \text{of characters in patient note i})}$$

REPORT:

- A 2d table with case_num as row and the last two digits of feature_num as column (fill in np.nan or None value for irrelevant cells since the number of features are different for each case_num) where each cell is the average fraction of annotated part (i.e., the # of annotated characters over the # of total characters in pn_history).
- Compute the same statistics but aggregate it over feature_num (take micro average, not macro average). That is, you will report total 10 fractional numbers.

Problem 5. Random Guess

Let us now try to obtain a baseline score that will roughly (but not exactly) correspond to the majority vote classifier in the usual classification. This will also give you an opportunity to write the code that computes the score.

For each pair of (case_num, feature_num) we have computed the fraction of annotated part in the previous problem. The plan is to attempt to make predictions on the train set (pn_history for pn_notes in train.csv, to be precise) by randomly selecting a block of character locations whose length match with the fraction computed above.

Note that the score metric used in the competition is micro F1. To learn the differences between micro F1 vs macro F1, please read https://vitalflux.com/micro-average-macro-average-scoring-metrics-multi-class-classification-python/ . There is more description in the slide 'final project' in our course website.

REPORT:

- Write a function that, given (case_num, feature_num, pn_history), generates the prediction described above.
- Try to make those predictions for every row in train.csv and evaluate the micro F1.
- Report both the code and the output micro F1 score.

The obtained micro F1 score should now be a baseline score you want to always beat. Whenever your fancy ML-based prediction does worse than this score, you are doing something wrong.

Some tips:

- I recommend that you make your prediction as 'numpy.array(dtype=bool)' with 'length=len(pn_history)' where the entry is True for predicted locations and False otherwise. You can turn the ground truth (train.csv's location column) into this kind of True/False array and simply apply & (and) operation, | (or) operation, or ~ (negation) operation.
- The attached code that I wrote has similar routines. Feel free to adapt it.

Problem 6. First ML

The NBME task does not seem to fit the standard supervised learning setup well. So, I have developed a way to convert it into a standard supervised learning problem.

Let us focus on case_num = 0. There are total 13 features. Note that this 'feature' is not the feature we consider in machine learning. To distinguish it, I will call it the **evaluation feature** hereafter. So, the evaluation feature is the actual category/class we want to classify.

The idea is that given a pn_history, we split it into a list of words and make each word a datapoint where the feature vector is a character frequency vector (e.g., a vector of length 26 with the first component being the frequency of 'a' in the word, etc.) in the word and the label

is X if the word is part of the annotated phrase for feature_num=X and -1 if the word is not part of any annotated phrase.

The hope is that the combination of certain characters should be informative for deciding the feature_num it belongs to.

So, the idea is that we construct data points as described above (which should be much larger than the total number of 'pn_history' that is annotated since each of them consists of many words). It is now a 14-class classification task at the word level! So, we can train a classifier, and then for prediction on a given pn_history, we can use our classifier to classify each word. Finally, for each feature_num, we can collect the word locations that were classified as feature_num, and this is our final prediction!

Now, if this makes sense, why not just one word? We can consider all the two consecutive words. If the phrase is like "A B C D", then we can also extract 3 data points: "A B", "B C", and "C D". In the provided code, I ended up using up to 5 consecutive words (the choice of 5 is somewhat arbitrary).

The provided code performs basic data processing and saves the preprocessed data as a pickle file. It then performs a 5-fold CV with a decision tree and computes the micro F1 score.

Note that there is more description in the 'final project' slides in D2L. I will also discuss it in class.

Please run main_fp_prep.py, which generates 'pdata_v01_RENAME.pkl'. Then, you can rename it to whatever you like (say pdata_v01.pkl – it's now a default). Then, modify main_fp_ml.py to have opt.pdata_name as the final name you just changed to. Then, run main_fp_ml.py to see the result.

Finally, make a submission in Kaggle by creating a notebook and copy-pasting mylib.py, main_fp_prep.py, and main_fp_final.py. Be sure to change the flags opt.kaggle_submission to True.

Please take some time to read the code and understand the big picture.

REPORT:

- Screenshot of your successful run (that contains the final f1 score)
- Screenshot of the Kaggle submission score with your name on it.

NOTE

- The evaluation of cross-validation is a bit different from what we learned in class. Although we are using 5-fold cross-validation, we just get a single overall f1 score rather than 5 different f1 scores. I will explain this in class (see the slides also).

Problem 7. Try various ML algorithms

In this problem, you will try out a bunch of ML algorithms and report the 5-CV result that is computed by main_fp_ml.py

Try the decision tree with pruning. Note that pruning can be done by ccp_alpha parameter. I found the performance of the decision tree to be sensitive to ccp_alpha. For some ccp_alpha values, I found that the code often crashes – in this case, just ignore it. Try out ccp_alpha values of 10^{-1, -2, -3, -4, -5} and report the best.

Try SVM with RBF kernel. Tune the value of gamma in a reasonable range and report the best. For this, it may take a long time. It is okay to try only 3 values of gammas.

REPORT:

- The list of f1 scores for each method and hyperparameters, and the score of the best one.

Problem 8. Character bigram

Character bigrams are a pair of characters. For example, "word" has three bigrams "wo", "or", "rd". However, it's a better idea to give special treatment to the first character. For example, "word" would actually have 5 bigrams in total by adding " w" and "d ". " w" implies the occurrence of character w at the first location. "d " implies the occurrence of character d as the last character a word.

So, the plan is to have frequency values for "xy" where x or y are characters. This may learn to distinguish phrases better.

I have written all the relevant code except for calc_bigrams(text) which can be found in mylib.py. Given a string `text`, you are expected to output a list of bigrams.

- For example, if text="word", then it returns [" w", "wo", "or", "rd", "d "]. If text="a b c", the output should be [" a", "a ", " b", "b ", " c", "c "].
- You should turn contiguous space characters into a single space character since a feature like " " (double space) probably does not mean anything special. Here, space character includes tabs and newlines as well. So, if text="a\t\r b \r\n", then the output should be [" a", "a ", " b", "b "]. I would use import re; re.sub(). Note that "\s" in regular expression should be handy. Please see the Python documents for the module `re` for more.
- Other punctuations like ",", ".", ";", ":", "(", ")" are probably not so useful. Let's replace these with spaces. I would use re.sub().

After writing calc_bigrams(text), report the evidence that your code is meeting the requirements above. You need to design test cases for each item above and report the test results.

Finally, to use your bigram version for training and validation, you need to do the following:

- From main_fp_prep.py, change opt.use_bigram to True. After running this script, I would change the output filename to descriptive like 'pdata_bigram_v01.pkl'.
- From main_fp_ml.py, change the input filename so you are using the bigram one.

Note that we would not use every single bigram since some bigrams appear only a few times, which would not affect the performance much but just increase the number of features. So, we

will use bigrams that appear only above ` dataopt.bigramcutoff' (default=30). Try to change `dataopt.bigramcutoff` in [7,15,30,60,120] and report how many unique bigrams survive (`bigram_vocab` is the variable that you want to look at).

REPORT:

a) Your calc_char_bigrams function and how it works with your test inputs.
b) Use a decision tree with the default parameter (without any arguments) and see how the f1 score is compared with the problem 6. Use dataopt.bigramcutoff=30.
c) For each `dataopt.bigramcutoff` in [7,15,30,60,120], how many unique bigrams will you have at the end?
d) Redo b) above (decision tree) with `dataopt.bigramcutoff` in [7,15,30,60,120] and report the f1 scores.

Problem 9. Investigation for improvements

This is an open problem. It's okay to use the unigram features only (Problem 7) for simplicity – the bigram one takes a longer time.

- Take the predictions of the best classifier from Problem 7.
- Identify case_num with the smallest f1 score. Call this the *target case.*
- Within the target case, pick top 10 patient notes with the largest number of false positives and top 10 patient notes with the largest number of false negatives.
- Manually inspect the data and report what you think is going on (focus on the most dominating pattern). This includes trying to understand the data overall and manually inspecting patient notes.
- Propose 3 possible fixes for the above.

REPORT:

- The target case number
- Summarize of the mistakes of the classifier (provide enough details like precisely which part of the note is false positive and false negative). Again, you can focus on the dominating pattern.
- Description of your investigation on why these mistakes are made.
- Description of your proposed fix.

Problem 10. Improve it yourself (open ended)

This problem is an open-ended question where you can try to implement the proposed fix in the previous problem. You can also try to do the same for the case with the second smallest f1 score, etc.

Also, there are many possible improvements you could make. See the slides 'final project' in D2L to see some examples.

The evaluation for this problem is not necessarily on whether you improve it or not; it is more on delivering your hypothesis (on what fix should help) and the reasoning (why it should help).

If you do these well, you can still get the full credit even if your fix did not lead to improvements in the f1 score.

REPORT:

- Summary of your attempts, hypotheses for it, and the reasoning.
- Be sure to report the resulting f1 score in 5-CV.

Problem 11. Your final submission score

REPORT:

Submit one of the best versions of your code to Kaggle and report the score here as a screenshot showing your name and score!