Computer Science 401 — 15 January 2024
St. George Campus — University of Toronto

Homework Assignment #1
Due: Friday, 9 February 2024 at 23h59 (11:59pm)

Red Team, Blue Team: Identifying Political Affiliations on Reddit

TAs: Mahya Mirbagheri and Yuntao Wu. Email: csc401-2024-01-a1@cs.toronto.edu.

Overview

In this assignment, you are tasked with creating classifiers that predict a user's political affiliation by analysing their posts. This assignment will give you experience with a social media corpus (viz., a collection of posts from Reddit), Python programming on natural language data, part-of-speech (PoS) tags, sentiment analysis, and machine learning with scikit-learn.

First, you will create a new feature-based classifier from beginning to end. This includes preprocessing the corpus, designing and implementing an extractor to gather feature information from each post, and creating classifiers to predict the user's affiliation using various machine-learning-based algorithms. Afterwards, you will create a neural classifier by fine-tuning an off-the-shelf language model called POLITICS and compare it to the feature-based classifiers.

Please monitor the course bulletin board (Piazza) regularly for announcements and discussion pertaining to this assignment.

Reddit Corpus

We have curated data from Reddit by scraping subreddits, using Pushshift, according to perceived political affiliation. Table 1 shows the subreddits assigned to each of four categories: left-leaning, right-leaning, center/neutral, and "alternative facts." Although at least the first three are often viewed as ordinal segments on a unidimensional spectrum, here we will treat all four categories as nominal classes. Here, we use the terms "post" and "comment" interchangeably to refer to short segments of user-produced text.

Left (598,944)	Center (599,872)	Right (600,002)	Alt (200,272)
twoXChromosomes (7,720,661)	news (2,782,9911)	theNewRight (19,466)	conspiracy (6,767,099)
occupyWallStreet (397,538)	politics (60,354,767)	whiteRights (118,008)	911truth (79,868)
lateStageCapitalism (634,962)	energy (416,926)	Libertarian (3,886,156)	
progressive (246,435)	canada (7,225,005)	AskTrumpSupporters (1,007,590)	
socialism (1,082,305)	worldnews (38,851,904)	The_Donald (21,792,999)	
demsocialist (5269)	law (464,236)	new_right (25,166)	
Liberal (151,350)		Conservative (1,929,977)	
		tea_party (1976)	

Table 1: Subreddits assigned to each category, with the *total* posts in each. Since there are over 181M posts, we sample randomly within each category – the resulting number of *available* posts for each category in this assignment is shown on the top row.

These data are stored on the teach.cs servers under `/u/cs401/A1/data/`. To save space, these files should *only* be accessed from that directory (and not copied). All data are in the JSON format.

Each datum has several fields of interest, including:

ups:	the integer number of upvotes.
downs:	the integer number of downvotes.
score:	$[ups - downs]$
controversiality:	a combination of the popularity of a post and the ratio between ups and downs.
subreddit:	the subreddit from which the post was sampled.
author:	the author ID.
body:	the main textual payload of the post, and our primary interest.
id:	the unique identifier of the comment.

Starter Code

We provide four starter code files: `a1_preprocess.py`, `a1_extract_features.py`, `a1_classify.py` and `a1_hf.py` that correspond to the four tasks of this assignment. Please read the instructions in both the starter code and this handout carefully. The starter code files are located in `/u/cs401/A1/code/`.

1 Pre-processing, Tokenizing and Tagging [18 marks]

The comments, as given, are not in a form amenable to feature extraction for classification – there is too much "noise." Therefore, the first step is to complete a Python program named `a1_preprocess.py`, in accordance with Section 4. This will read subsets of JSON files and, for each comment, perform the following steps, in order, on the "body" field of each selected comment:

1. Replace all non-space whitespace characters, including newlines, tabs, and carriage returns, with spaces.
2. Remove "[deleted]" or "[removed]" statements.
3. Replace HTML character codes (i.e., *&...;*) with their ASCII equivalents (see `http://www.asciitable.com`).
4. Remove all URLs (i.e., tokens beginning with *http* or *www*).
5. Remove duplicate spaces between tokens.
    - Each token is now be separated by a single space.
6. Apply the following steps using spaCy (see below):
    - Tagging: Tag each token with its part-of-speech. A tagged token consists of a word, the '/' symbol, and the tag (e.g., *dog/NN*). See below for information on how to use the tagging module. The tagger can make mistakes.
    - Lemmatization:
        - Replace the token itself with the `token.lemma_`; e.g., *words/NNS* becomes *word/NNS*. If the lemma begins with a dash ('-') when the token does not (e.g., *-PRON-* for *I*), just keep the token.
        - Retain the case of the original token when you perform this replacement. We make two distinctions here: if the original token is entirely in uppercase, then so is the lemma; otherwise, keep the lemma in lowercase.
    - Sentence segmentation: Add a newline between each sentence. For this assignment, we will use spaCy's `sentencizer` component to segment sentences in a post. Remember also to mark the end of the post with a newline (watch out for duplicates!).

Additional Specifications

spaCy The package spaCy is very handy for many NLP tasks. Here, we **only** use its ability to obtain PoS tags and lemmata, along with sentence segmentation. For example:

```
import spacy

nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
nlp.add_pipe("sentencizer")

utt = nlp("I know words. I have the best words")

for sent in utt.sents:
    print(sent.text)
    for token in sent:
        print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
            token.shape_, token.is_alpha, token.is_stop)
```

Functionality The `a1_preprocess.py` program reads a subset of the (static) input JSON files, retains the fields we care about, including **id**, which you'll soon use as a key to obtain pre-computed features, and **body**, which is text that you preprocess and replace before saving the result to an output file. To each comment, also add a **cat** field, with the name of the file from which the comment was retrieved (e.g., Left, Alt, ...).

The program takes three arguments: your student ID (mandatory), the output file (mandatory), and the maximum number of lines to sample from each category file (optional; default=10,000). For example, if you are student 999123456 and want to create `preprocessed.json`, you'd run:

```
python3 a1_preprocess.py 999123456 -o preprocessed.json
```

The output of `a1_preprocess.py` will be used in Task 2.

Subsampling By default, you should only sample 10,000 lines from each of the Left, Center, Right, and Alt files, for a total of 40,000 lines. From each file, start sampling lines at index $[ID \ \% \ len(X)]$, where ID is your student ID, % is the modulo arithmetic operator, and $len(X)$ is the number of comments in the given input file (i.e., `len(data)`, once the JSON parse is done). Use circular list indexing if your start index is too close to the end.

Your Tasks Copy the template from `/u/cs401/A1/code/a1_preprocess.py`. There are two functions you need to modify:

1. In `preprocess`, fill out each **if** statement with the associated preprocessing step above.
2. In `main`, replace the lines marked with `TODO` with the code they describe. By default, `args.a1_dir` points to `/u/cs401/A1/`, so load the data from `args.a1_dir`'s data subdirectory.

For this section, you may only use standard Python libraries, *except* for Step 5. For debugging, you are advised to either use a different input folder with your own JSON data, or pass strings directly to `preprocess`.

2 Feature Extraction [22 marks]

The second step is to complete a Python program named `a1_extract_features.py`, in accordance with Section 4, that takes the preprocessed comments from Task 1, extracts features that are relevant to political bias detection, and builds an `.npz` datafile that will be used to train models and classify comments in Task 3.

For each comment, you need to extract 173 features and write these, along with the category, to a single NumPy array. These features are listed below. Several of these features involve counting tokens based on their tags. For example, counting the number of *adverbs* in a comment involves counting the number of tokens that have been tagged as RB, RBR, or RBS. Table 4 explicitly defines some of these features (many of which we have provided as constants in the template); other definitions are available on teach.cs in `/u/cs401/Wordlists/`. You may copy and modify these files, but do not change their filenames.

1. Number of tokens in uppercase ($\geq$ 3 letters long).
2. Number of first-person pronouns.
3. Number of second-person pronouns.
4. Number of third-person pronouns.
5. Number of coordinating conjunctions.
6. Number of past-tense verbs.
7. Number of future-tense verbs.
8. Number of commas.
9. Number of multi-character punctuation tokens.
10. Number of common nouns.
11. Number of proper nouns.
12. Number of adverbs.
13. Number of *wh-* words.
14. Number of slang acronyms.
15. Average length of sentences, in tokens.
16. Average length of tokens, excluding punctuation-only tokens, in characters.
17. Number of sentences..
18. Average of AoA (100-700) from Bristol, Gilhooly, and Logie norms.
19. Average of IMG from Bristol, Gilhooly, and Logie norms.
20. Average of FAM from Bristol, Gilhooly, and Logie norms.
21. Standard deviation of AoA (100-700) from Bristol, Gilhooly, and Logie norms.
22. Standard deviation of IMG from Bristol, Gilhooly, and Logie norms.
23. Standard deviation of FAM from Bristol, Gilhooly, and Logie norms.
24. Average of V.Mean.Sum from Warringer norms.
25. Average of A.Mean.Sum from Warringer norms.
26. Average of D.Mean.Sum from Warringer norms.
27. Standard deviation of V.Mean.Sum from Warringer norms.
28. Standard deviation of A.Mean.Sum from Warringer norms.
29. Standard deviation of D.Mean.Sum from Warringer norms.

30-173. LIWC/Receptiv*iti* features.

Note 1: All of the provided wordlists contain tokens in lowercase. After extracting feature 1 above, you may convert the tokens in the comment to lowercase as well. Take care to modify only the text and not the PoS tags , i.e, *Dog/NN* should become *dog/NN* and not *dog/nn*.

Note 2: While calculating features 18-29, only consider those tokens that are present in the text as well as the wordlists. You may find that some tokens are present in the wordlists with an N/A value; ignore these.

Task Specification

Functionality The `a1_extract_features.py` program reads a preprocessed JSON file and extracts features for each comment therein, producing and saving a $D \times 174$ NumPy array, where the i^{th} row is the slice of features for the i^{th} comment, followed by an integer for the class (0: Left, 1: Center, 2: Right, 3: Alt), as per the **cat** JSON field.

The program takes two arguments: the input filename (i.e., the output of `a1_preproc`), and the output filename. For example, given input `preprocessed.json` and the desired output `feats.npz`, you'd run:

```
python3 a1_extract_features.py -i preprocessed.json -o feats.npz
```

The output of `a1_extract_features.py` will be used in Task 3.

Norms Lexical norms are aggregate subjective scores given to words by a large group of individuals. Each type of norm assigns a numerical value to each word. Here, we use two sets of norms:

1. **Bristol+GilhoolyLogie**:
   These are found in `/u/cs401/Wordlists/BristolNorms+GilhoolyLogie.csv`, specifically the fourth, fifth, and sixth columns. These measure the Age-of-acquisition (AoA), imageability (IMG), and familiarity (FAM) of each word, which we can use to measure lexical complexity. More information can be found, for example, in this article.
2. **Warringer**:
   These are found in `/u/cs401/Wordlists/Ratings_Warriner_et_al.csv`, specifically the third, sixth, and ninth columns. These norms measure the valence (V), arousal (A), and dominance (D) of each word, according to the VAD model of human affect and emotion. You can read more information on this particular data set.

When you compute features 18-29, only consider those words that exist in the respective norms file. **Assume the default value for all of the above features is zero.** Treat the mean and standard deviation of zero words to be zero.

LIWC/Receptiv*iti* The Linguistic Inquiry & Word Count (LIWC) tool has been a standard across NLP research, including authorship and sentiment analysis. This tool provides 85 measures mostly related to word choice. The company Receptiv*iti* provides an extended set of these features, with 59 measures of personality derived from text. The company has graciously donated this course access to its API.

To simplify things, we have already extracted these 144 features for you. Simply copy the pre-computed features from the appropriate *uncompressed* `.npy` files stored in `/u/cs401/A1/feats/`. Specifically:

1. Comment IDs are stored in `_IDs.txt` files (e.g., `Alt_IDs.txt`). When processing a comment, find the index (row) i of the ID in the appropriate ID text file for the category, and copy the 144 elements of that row from the associated `_feats.dat.npy` file.
2. The file `feats.txt` provides the names of these features in the order they are marshalled in the `.dat.npy` files. For this assignment, these names will suffice as to the meaning of these features, but you are welcome to obtain your own API license from Receptiv*iti* in order to get access to their documentation.

Your Tasks Copy the template from `/u/cs401/A1/code/a1_extract_features.py`. There are two functions you need to modify:

1. In `extract1`, extract each the first 29 of the aforementioned features from the input string. Features 30-173 should be extracted in `extract2`.
2. In `main`, call `extract1` and `extract2` on each datum and add the results (+ the class) to the `feats` array.

When your feature extractor works to your satisfaction, build `feats.npz`, from all input data.

3 Experiments and Classification [30 marks]

The third step is to use the features extracted in Task 2 to classify comments using the scikit-learn machine learning package. Here, you will modify various hyper-parameters and interpret the results analytically. As everyone has different slices of the data, there are no expectations on overall *accuracy*, but you are expected to discuss your findings with scientific rigour. Copy the template from `/u/cs401/A1/code/a1_classify.py` and complete the `main` body and the functions for the following experiments according to the specifications therein.

The program takes two arguments: the input feature file (the output of `a1_extract_features`), and an output directory.

```
python3 a1_classify.py -i feats.npz -o .
```

You should create the output directory if it doesn't already exist. In `main`, you are expected to load the data from the input file, partition the input into a train and test set, and call the experimental functions in order. Use the `train_test_split` method to split the data into a random 80% for training and 20% for testing. For part 3.3, use the entire loaded data set.

3.1 Classifiers

Train the following 5 classifiers (see hyperlinks for API) with `fit(X_train, y_train)`:

1. `SGDClassifier`: support vector machine with a linear kernel.
2. `GaussianNB`: a Gaussian naive Bayes classifier.
3. `RandomForestClassifier`: with a maximum depth of 5, and 10 estimators.
4. `MLPClassifier`: A feed-forward neural network, with $\alpha = 0.05$.
5. `AdaBoostClassifier`: with the default hyper-parameters.

Here, `X_train` is the first 173 columns of your training data, and `y_train` is the last column. Obtain predicted labels with these classifiers using `predict(X_test)`, where `X_test` is the first 173 columns of your testing data. Obtain the 4×4 confusion matrix C using `confusion_matrix`. $c_{i,j}$, the element at row i, column j in C, is the number of instances belonging to class i that were classified as class j. Compute the following manually, using the associated function templates:

Accuracy : The total number of correctly classified instances over all classifications: $A = \frac{\sum_i c_{i,i}}{\sum_{i,j} c_{i,j}}$.

Recall : For each class κ, the fraction of cases that are truly class κ that were classified as κ, $R(\kappa) = \frac{c_{\kappa,\kappa}}{\sum_j c_{\kappa,j}}$.

Precision : For each class κ, the fraction of cases classified as κ that truly are κ, $P(\kappa) = \frac{c_{\kappa,\kappa}}{\sum_i c_{i,\kappa}}$.

Write the results to the text file `a1_3.1.txt` in the output directory. You must write to this file using the format strings provided in the template. **If you do not follow the format, you may receive a mark of zero**. For each classifier, you will print the accuracy, recall, precision, and confusion matrix. You may include a written analysis if you are so inclined, but only after the results.

3.2 Amount of Training Data

Many researchers attribute the success of modern machine learning to the sheer volume of data that is now available. Modify the amount of data that is used to train your preferred classifier from above in five increments: 1K, 5K, 10K, 15K, and 20K. These can be sampled arbitrarily from the training set in Section 3.1. *Using only the classification algorithm with the highest accuracy from Section 3.1*, report the accuracies of the classifier in the file `a1_3.2.txt` using the format string provided in the template. On one or more lines following the reported accuracies, comment on the changes to accuracy as the number of training samples increases, in two sentences or more, on a possible explanation. Is there an expected trend? Do you see such a trend? Hypothesize as to why or why not.

3.3 Feature Analysis

Certain features may be more or less useful for classification, and too many can lead to overfitting or other problems. Here, you will select the best features for classification using `SelectKBest` according to the f_classif metric as in:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

selector = SelectKBest(f_classif, you_figure_it_out)
X_new = selector.fit_transform(X_train, y_train)
pp = selector.pvalues_
```

In the example above, `pp` stores the p-value associated with doing a χ^2 statistical test on each feature. A smaller value means the associated feature better separates the classes. Do this:

1. For the 32k training set and each $k \in \{5, 50\}$, find the best k features according to this approach. Write the associated p-values to `a1_3.3.txt` using the format strings provided.
2. Train the best classifier from section 3.1 on the 1K training set as well as the 32K training set, using only the best $k = 5$ features. Write the accuracies of both classifiers on the full test to `a1_3.3.txt` using the format strings provided.
3. Extract the indices of the top $k = 5$ features using the 1K training set and take the intersection with the $k = 5$ features using the 32K training set. Write using the format strings provided.
4. Format the top $k = 5$ feature indices extracted from the 32K training set to the file using the format string provided.
5. Following the above, answer the following questions:
   (a) Provide names for the features found in the above intersection of the top $k = 5$ features. Provide a possible explanation as to why these features may be especially useful.
   (b) Are p-values generally higher or lower given more or less data? Why or why not?
   (c) Name the top 5 features chosen for the 32K training case. Hypothesize as to why those particular features might differentiate the classes.

3.4 Cross-validation

Many papers in machine learning stick with a single subset of data for training and another for testing (occasionally with a third for validation). This may not be the most honest approach. Is the best classifier from Section 3.1 *really* the best? For each of the classifiers in Section 3.1, run 5-fold cross-validation given all the initially available data. Specifically, use `KFold`. **Set the shuffle argument to true**.

For each fold, obtain the accuracy on the test partition after training on the rest for each classifier. Report the mean accuracy of each classifier for each of the 5 folds in the order specified in 3.1 to `a1_3.4.txt` using the format strings provided. Next, determine whether the accuracy of your best classifier, across the 5 folds, is *significantly* better than any of the others. *i.e.*, given vectors `a` and `b`, one for each classifier, containing the accuracy values for each of the respective 5 folds, obtain the p-value from the output `S`, below:

```
from scipy import stats
S = stats.ttest_ind(a, b)
print(S.pvalue)
```

You should have 4 p-values. Report them using the provided format string in the same order as the accuracies, excluding the self-comparison. For example, if the best classifier from 3.1 was the `RandomForestClassifier`, then the p-values should be reported in the order: 1 vs. 3, 2 vs. 3, 4 vs. 3, 5 vs. 3.

4 LM-based Neural Classifier [20 marks]

Finally, you need to create a neural classifier on top of a language model (LM) and compare its performance with the feature-based classifier you have just created. In particular, you will fine-tune a foundational model called POLITICS. You can quickly pull together a functioning, task-specific model using platforms such as Hugging Face. In `a1_hf.py`, you need to complete the entire training pipeline:

1. **Data Loading and Preprocessing** [2 marks]: The `load_data` function loads and processes data from the Reddit corpus. Then it returns a Python dictionary containing text and corresponding labels. This part of the assignment is provided to you. Same as in Preprocessing 1, you need to use your student ID for subsampling.

2. **Tokenization** [2 marks]: Implement the `tokenize_text` function to tokenize the input text using the Hugging Face tokenizer. This step is crucial for preparing the data for model training.

3. **Model Training** [2 marks]: Finish the `train_model` function to set up the Trainer object from Hugging Face. Use the provided model, tokenizer, and training arguments to train a sequence classification model.

4. **Evaluation Metrics** [2 marks]: Complete the `compute_metrics` function to calculate evaluation metrics on the trained model's predictions. You only need to calculate the accuracy, but you should follow the specified return format.

5. **The Main Training Loop** [7 mark]: In the function `main`, load the data, tokenize it, train the model and save the evaluation results into `evaluation_result.txt`. Submit this `.txt` file.

Comparison to Feature-based Classifiers [5 marks]

Finally, in `analysis.txt` you need to provide a brief comparison between the feature-based classifiers and this LM-based classifier. In particular, you need to cover the following points:

- How are the relative performances? How should we best use the different evaluation metrics in our comparison?

- What are the potential strengths and weaknesses of the feature-based classifiers? What are the potential strengths and weaknesses of the LM-based classifier?

- In $5 \sim 10$ sentences, describe one experiment to verify a pair of one strength and one weakness. You don't need to actually code or run the experiment. What are the possible outcomes? What conclusion would you draw from each of these outcomes?

Additional Specifications

Hugging Face Transformers You will mostly interact with Hugging Face through the `transformers` package. Here is its documentation.

Debugging and Using GPUs on teach.cs In `a1_hf.py`, we provide a `--toy` option to make your model only train on a tiny sliver of the dataset (40 examples). You should use this option for debugging during your development phase. You can train this toy model using the CPU, which will take ~ 1 minute to complete. Again, if you are student 999123456, you can run the following command.

```
python3 a1_hf.py 999123456 --toy
```

But you should also make sure your code works using GPU acceleration. Specifically, on teach.cs, you can add `srun -p csc401 --gres gpu` to the beginning of your command to initiate your program using one of the GPUs. The training process (not including the data loading process) should complete on one instance.

```
srun -p csc401 --gres gpu python3 a1_hf.py 999123456 --toy
```

Finally, **only** when you are sure that your code works, you can start the real training process. This actual training process must be initiated with a GPU available, and you must start your command with `srun -p csc401 --gres gpu`.

```
srun -p csc401 --gres gpu python3 a1_hf.py 999123456
```

Note: Availability of GPU machines at particular times is not guaranteed, especially when it gets close to the deadline. Start early!

Your Tasks

1. Copy the template from `/u/cs401/A1/code/a1_hf.py` and implement the missing parts. Submit the finished file.
2. Run your code to actually train the model. Submit the result file, `evaluation_result.txt`.
3. Write-up the comparison between the two types of classifiers in `analysis.txt`, and submit this file.

General Specifications

As part of grading your assignment, the grader may run your programs and/or python files on test data and configurations that you have not previously seen. This may be done in part by automatic scripts. It is therefore important that each of your programs precisely meets all the specifications, including its own name and the names of the files and functions that it uses. **A program that cannot be evaluated because it varies from specifications will receive zero marks on the relevant sections.**

The flag `--a1-dir` can be used to specify an alternate location than `/u/cs401/A1` to load data from, though the submitted files should be generated on the teach.cs machines. Do **not** hardwire the absolute address of your home directory within the program; the grader does not have access to this directory.

All your programs must contain adequate internal documentation to be clear to the graders.

We use Python version 3.10.13 on teach.cs.

Submission Requirements

This assignment is submitted electronically. You should submit:

1. All your code for `a1_preprocess.py`, `a1_extract_features.py`, `a1_classify.py` and `a1_hf.py`.
2. `a1_3.1.txt`: Report on classifiers.
3. `a1_3.2.txt`: Report on the amount of training data.
4. `a1_3.3.txt`: Report on feature analysis.
5. `a1_3.4.txt`: Report on 5-fold cross-validation.
6. `evaluation_result.txt`: The evaluation result generated from the huggingface training process.
7. `analysis.txt`: The comparison between the two types of classifiers: LM-based and feature-based.

In another file called `ID.txt` (use the template on the course web page), provide the following information:

1. Your first and last name.
2. Your student number.
3. Your teach.cs login ID.
4. Your preferred contact email address.
5. Whether you are an undergraduate or graduate.
6. This statement:

   > By submitting this file, I declare that my electronic submission is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct, as well as the collaboration policies of this course.

You do not need to hand in any files other than those specified above. The electronic submission must be made from teach.cs with the `submit` command:

```
$ submit -c <course> -a A1 <filename-1> ...  <filename-n>
```

where `<course>` is `csc401h` or `csc2511h` depending on which course you're registered in, and `<filename-1>` to `<filename-n>` are the aforementioned 11 files (including `ID`) you are submitting.

Working Outside the Lab

If you want to do some or all of this assignment on your laptop or home computer, you will have to do the extra work of downloading and installing the requisite software and data. If you take this route, you take on all of the associated risks. You are strongly advised to upload regular backups of your work to

teach.cs, so that if your home machine fails or proves to be inadequate, you can immediately continue working on the assignment at teach.cs. When you have completed the assignment, you should try your programs out on teach.cs to make sure that they run correctly there. **Any component that does not work on teach.cs will get zero marks.**

Frequently Asked Questions

In this section, we provide a list of clarifications to possible concerns. In addition to checking this, students needing clarification with respect to the assignment should check Piazza, where we will have another FAQ section based on questions from your fellow students.

- Always take spaCy's output to be correct when completing the tokenization and lemmatization. In the case of lemmata that contain multiple tokens, concatenate the multiple tokens with an underscore and append the tag to that.
- A multi-character punctuation token is a token consisting of only punctuation characters that is **more than** one character long.
- For word-list features, only consider the words that are present in the word-list file.
- When we have 1 data point, the sample std. dev. is 0.
- For Feature 7, "Number of future-tense verbs," capturing all cases of a future tense verb in text is not straightforward, and we do not expect a foolproof solution. Identifying the patterns listed in Table 4 is sufficient.
- To speed up convergence of the `MLPClassifier`, remember to set the *alpha* parameter to 0.05.
- Accuracies in the classification portion may differ from student to student based on a number of benign factors.
- Of {"A11", "AA1", "AAA1"}, only "AAA1" contains at least 3 uppercase letters.
- For duplicated words in BristolNorms+GilhoolyLogie.csv, take the last of each duplication.

Appendix: Tables

Tag	Name	Example
CC	Coordinating conjunction	*and*
CD	Cardinal number	*three*
DT	Determiner	*the*
EX	Existential *there*	*there [is]*
FW	Foreign word	*d'oeuvre*
IN	Preposition or subordinating conjunction	*in, of, like*
JJ	Adjective	*green, good*
JJR	Adjective, comparative	*greener, better*
JJS	Adjective, superlative	*greenest, best*
LS	List item marker	*(1)*
MD	Modal	*could, will*
NN	Noun, singular or mass	*table*
NNS	Noun, plural	*tables*
NNP	Proper noun, singular	*John*
NNPS	Proper noun, plural	*Vikings*
PDT	Predeterminer	*both [the boys]*
POS	Possessive ending	*'s, '*
PRP	Personal pronoun	*I, he, it*
PRP$	Possessive pronoun	*my, his, its*
RB	Adverb	*however, usually, naturally, here, good*
RBR	Adverb, comparative	*better*
RBS	Adverb, superlative	*best*
RP	Particle	*[give] up*
SYM	Symbol (mathematical or scientific)	+
TO	*to*	*to [go] to [him]*
UH	Interjection	*uh-huh*
VB	Verb, base form	*take*
VBD	Verb, past tense	*took*
VBG	Verb, gerund or present participle	*taking*
VBN	Verb, past participle	*taken*
VBP	Verb, non-3rd-person singular present	*take*
VBZ	Verb, 3rd-person singular present	*takes*
WDT	*wh*-determiner	*which*
WP	*wh*-pronoun	*who, what*
WP$	Possessive *wh*-pronoun	*whose*
WRB	*wh*-adverb	*where, when*

Table 2: The Penn part-of-speech tagset—words.

Tag	Name	Example
#	Pound sign	*£*
$	Dollar sign	$
.	Sentence-final punctuation	*!, ?, .*
,	Comma	
:	Colon, semi-colon, ellipsis	
(	Left bracket character	
)	Right bracket character	
"	Straight double quote	
‘	Left open single quote	
“	Left open double quote	
’	Right close single quote	
”	Right close double quote	

Table 3: The Penn part-of-speech tagset—punctuation.

First person:
I, me, my, mine, we, us, our, ours
Second person:
you, your, yours, u, ur, urs
Third person:
he, him, his, she, her, hers, it, its, they, them, their, theirs
Future Tense:
*’ll, will, gonna, going+to+*VB
Common Nouns:
NN, NNS
Proper Nouns:
NNP, NNPS
Adverbs:
RB, RBR, RBS
wh-words :
WDT, WP, WP$, WRB
Modern slang acronyms:
smh, fwb, lmfao, lmao, lms, tbh, rofl, wtf, bff, wyd, lylc, brb, atm, imao, sml, btw, bw, imho, fyi, ppl, sob, ttyl, imo, ltr, thx, kk, omg, omfg, ttys, afn, bbs, cya, ez, f2f, gtr, ic, jk, k, ly, ya, nm, np, plz, ru, so, tc, tmi, ym, ur, u, sol, fml Consider also `https://www.netlingo.com/acronyms.php`, if you want, for no-bonus completion.

Table 4: Miscellaneous feature category specifications.