Identifying political persuasion on Reddit

Introduction

This assignment will give you experience with a social media corpus (i.e., a collection of posts from Reddit), Python programming, part-of-speech (PoS) tags, sentiment analysis, and machine learning with scikit-learn.

Your task is to split posts into sentences, tag them with a PoS tagger that we will provide, gather some feature information from each post, learn models, and use these to classify political persuasion. Sentiment analysis is an important topic in computational linguistics in which we quantify subjective aspects of language. These aspects can range from biases in social media for marketing, to a spectrum of cognitive behaviours for disease diagnosis.

Reddit Corpus

We have curated data from Reddit by scraping subreddits, using Pushshift, by perceived political affiliation. Table 1 shows the subreddits assigned to each of four categories: left-leaning, right-leaning, center/neutral, and 'alternative facts'. Although the first three (at least) are often viewed as ordinal segments on a unidimensional spectrum, here we treat these categories as nominal classes. Here, we use the terms 'post' and 'comment' interchangeably to mean 'datum', i.e., a short segment 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 message of the post, and our primary interest.

id: the unique identifier of the comment.

Your tasks

1 Pre-processing, tokenizing, and tagging [25 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_preproc.py`, in accordance with Section 5, that 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. Remove all newline characters.
2. Replace HTML character codes (i.e., *&...;*) with their ASCII equivalent (see `http://www.asciitable.com`).
3. Remove all URLs (i.e., tokens beginning with *http* or *www*).
4. Split each punctuation (see `string.punctuation`) into its own token using whitespace except:
   - Apostrophes.
   - Periods in abbreviations (e.g., *e.g.*) are *not* split from their tokens. E.g., *e.g.* stays *e.g.*
   - Multiple punctuation (e.g., *!?!*, *...*) are *not* split internally. E.g., *Hi!!!* becomes *Hi !!!*
   - You can handle single hyphens (-) between words as you please. E.g., you can split *non-committal* into three tokens or leave it as one.
5. Split clitics using whitespace.
   - Clitics are contracted forms of words, such as *n't*, that are concatenated with the previous word.
   - Note: the possessive *'s* has its own tag and is distinct from the clitic *'s*, but nonetheless must be separated by a space; likewise, the possessive on plurals must be separated (e.g., *dogs '*).
6. Each token is tagged with its part-of-speech using spaCy (see below).
   - 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.
7. Remove stopwords. See `/u/cs401/Wordlists/StopWords`.
8. Apply lemmatization using spaCy (see below).
9. Add a newline between each sentence.
   - This will require detecting end-of-sentence punctuation. Some punctuation does not end a sentence; see standard abbreviations here: `/u/cs401/Wordlists/abbrev.english`. It can be difficult to detect when an abbreviation ends a sentence; e.g., in *Go to St. John's St. John is there.*, the first period is used in an abbreviation, the last period ends a sentence, and the second period is used both in an abbreviation and an end-of-sentence. You are not expected to write a 'perfect' pre-processor (none exists!), but merely to use your best judgment in writing heuristics; see section 4.2.4 of the Manning and Schütze text for ideas.
10. Convert text to lowercase.

Functionality: The a1_preproc.py program reads a subset of the (static) input JSON files, retains the fields you 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 preproc.json, you'd run:

```
python a1_preproc.py 999123456 -o preproc.json
```

The output of a1_preproc.py will be used in Task 2.

Your task: Copy the template from /u/cs401/A1/code/a1_preproc.py. There are two functions you need to modify:

1. In preproc1, 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.

For this section, you may only use standard Python libraries, *except* for tagging (step 6) and lemmatization (step 8), as described below. For debugging, you are advised to either use a different input folder with your own JSON data, or pass strings directly to preproc1.

spaCy: spaCy is a Python library for natural language processing tasks, especially in information extraction. Here, we **only** use its ability to obtain part-of-speech tags and lemma. For example:

```
import spacy

nlp = spacy.load('en', disable=['parser', 'ner'])
utt = nlp(u"I know the best words")
for token in utt:
...     print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
...             token.shape_, token.is_alpha, token.is_stop)
```

When performing step (6) above, simply append '/' and the appropriate token.tag_ to the end of each tag, as per the earlier example.

When performing step (8) above, simply 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 doesn't (e.g., *-PRON-* for *I*, just keep the token.

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'.

2 Feature extraction [20 marks]

The second step is to complete a Python program named `a1_extractFeatures.py`, in accordance with Section 5, that takes the preprocessed comments from Task 1, extracts features that are relevant to 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; other definitions are available on CDF in `/u/cs401/Wordlists/`. You may copy and modify these files, but do not change their filenames.

1. Number of first-person pronouns
2. Number of second-person pronouns
3. Number of third-person pronouns
4. Number of coordinating conjunctions
5. Number of past-tense verbs
6. Number of future-tense verbs
7. Number of commas
8. Number of multi-character punctuation tokens
9. Number of common nouns
10. Number of proper nouns
11. Number of adverbs
12. Number of *wh-* words
13. Number of slang acronyms
14. Number of words in uppercase ($\geq$ 3 letters long)
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

Functionality: The `a1_extractFeatures.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 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 `preproc.json` and the desired output `feats.npz`, you'd run:

```
python a1_extractFeatures.py -i preproc.json -o out.json
```

The output of `a1_extractFeatures.py` will be used in Task 3.

Your task: Copy the template from `/u/cs401/A1/code/a1_extractFeatures.py`. There are two functions you need to modify:

1. In `extract1`, extract each of the 173 aforementioned features from the input string.
2. In `main`, call `extract1` 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.

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:

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, here.

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. More information on this particular data set can be found here.

When you compute features 18-29, only consider those words that exist in the respective norms file.

LIWC/Receptiv*iti*: The Linguistic Inquiry & Word Count (LIWC) tool has been a standard in a variety of NLP research, especially around authorship and sentiment analysis. This tool provides 85 measures mostly related to word choice; more information can be found here. The company Receptiv*iti* provides a superset of these features, which also includes 59 measures of personality derived from text. The company has graciously donated access to its API for the purposes of this course.

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, starting at element $i \cdot 144$, from the associated `_feats.dat.npy` file.
2. The file `feats.txt` provides the names of these features, in the order provided. For this assignment, these names will suffice as to their meaning, but you are welcome to obtain your own API license from Receptiv*iti* in order to get access to their documentation.

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:

3.1 Classifiers

Use the `train_test_split` method to split the data into a random 80% for training and 20% for testing. Train the following 5 classifiers (see hyperlinks for API) with `fit(X_train, y_train)`:

1. `SVC`: support vector machine with a linear kernel.
2. `SVC`: support vector machine with a radial basis function ($\gamma = 2$) kernel.
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`. Given that the element at row i, column j in C (i.e., $c_{i,j}$) 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.csv`, in a comma-separated value format. Each of the first five lines has the following, in order:

1. the number of the classifier (i.e., 1-5)
2. the overall accuracy, recall for the 4 classes, and precision for the 4 classes
3. the confusion matrix, read row-by-row

That is, each of the first five lines should have $1 + (1 + 4 + 4) + 4 \times 4 = 26$ numbers separated by commas. If so desired, add any analytical commentary to the sixth line of this file.

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 resulting *accuracies* in a comma-separated form in the first line of `a1_3.2.csv`. On the second line of that file, comment on the changes to accuracy as the number of training samples increases, including at least two sentences 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 each of the 1K training set from Section 3.2, and the original 32K training set from Section 3.1, and for each number of features $k = \{5, 10, 20, 30, 40, 50\}$, find the best k features according to this approach. On each of the first 6 lines of a1_3.3.csv, write the number of features, and the associated p-values for the 32K training set case, separated by commas.
2. Train the best classifier from section 3.1 for each of the 1K training set and the 32K training set, using only the best $k = 5$ features. On the 7^{th} line of a1_3.3.csv, write the accuracy for the 1K training case and the 32K training case, separated by a comma
3. On lines 8 to 10 of a1_3.3.csv, answer the following questions:
   (a) What features, if any, are chosen at both the low and high(er) amounts of input data? Also provide a possible explanation as to why this might be.
   (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 accuracy on the test partition after training on the rest for each classifier, in the original order, and report these on their own line (one per fold) in a1_3.4.csv, separated by commas. Next, compare the accuracies of your best classifier, across the 5 folds, with each of the other 4 classifiers to check if it is *significantly* better than any 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_rel(a, b)
```

You should have 4 p-values. Report each in a1_3.4.csv, on the sixth line, separated by commas, in the order the classifiers appear in Section 3.1. On the seventh line, comment on any significance you observe, or any lack thereof, and hypothesize as to why, in one to three sentences.

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.