Computer Science 401 6 January 2014
St. George Campus University of Toronto

Homework Assignment #1
Due: Friday, 7 February 2014 at 7 PM

Twit Classification

TA: Varada Kolhatkar (t9kolhat@cdf.toronto.edu)

1 Introduction

This assignment will give you experience with web corpora (i.e., a collection of tweets from Twitter), Python programming, part-of-speech (PoS) tags, and the WEKA machine learning software package.

Your task is to split the tweets into sentences, tag them with a PoS tagger that we will provide, gather some feature information from each tweet, and use this information in machine learning classifiers that estimate the Twitter account (i.e., the *twit*) that produced a given tweet.

You should check the course bulletin board frequently for announcements and discussion pertaining to this assignment.

2 Tweet Corpus

Founded in 2006, Twitter (`www.twitter.com`) has grown to become one of the most popular social media services, known for its 140-character restriction on each post. In addition to a large general user base, Twitter is used extensively by celebrities, politicians, and news services to entertain, engage, or inform their followers. For the purposes of this assignment, we have collected a small set of 1000 tweets from each of 20 Twitter accounts, including the top 10 most popular, as measured by the number of followers. Each of the 20 files in the */u/cs401/A1/tweets/* directory represent one account and contain the tweets pertaining to that account. To save space in your cdf account, these files should be accessed from that directory.

Each line of these data files is a single tweet, which may contain multiple sentences despite their short length. The tweets have been collected directly from the web, so they may contain html tags (e.g., anchor tags $\langle a \rangle and \langle /a \rangle$). Also, tweets contain features specific to Twitter such as the use of 'hashtags' (e.g., #NLCAssn1) to indicate the relevant topic, or user tags to identify another Twitter account (e.g., @user).

3 Your tasks

1. Pre-processing, tokenizing, and tagging [25 marks]

The tweets, as given, are not in a form amenable to feature extraction for classification – there is too much 'noise'. Therefore, the first step is to write a Python program named *twtt.py* (tweet tokenize and tag), in accordance with the "General specifications" section below, that will take a tweet file in its provided form and convert it to a normalized form where:

1. All html tags and attributes (i.e., everything not visible on a browser) are removed.
2. Html character codes (i.e., `&...;`) are replaced with an ASCII equivalent.
   See `http://www.asciitable.com`.
3. All URLs (i.e., tokens beginning with `http` or `www`) are removed.
4. The first character in Twitter user names (@) and hash tags (#) are removed.
5. Each sentence within a tweet is on its own line.

   - 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. I'll be 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 here (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.

6. Ellipsis (i.e., '...'), and other kinds of multiple punctuation (e.g., '!!!') are not split.
7. Each token, including punctuation and clitics, is separated by spaces.

   - Clitics are contracted forms of words, such as *n't*, that are concatenated with the previous word.
   - Note that 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 apostrophe on plurals must be separated.

8. Each token is tagged 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.

9. Between each tweet is the pipe symbol '|', which occurs on its own line.

   - We will later need to differentiate between sentences in a tweet, hence the explicit demarcation.

Note that:

- If a tweet is empty after pre-processing, it should be preserved as such.
- Periods in abbreviations (e.g., e.g.) are not split off from the rest of their tokens. Therefore, *e.g.* stays as the token *e.g.*.
- 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.

The *twtt.py* program takes two arguments: the input raw tweet file and the output tokenized and tagged tweet file. Use '.twt' as the extension for the output file. For example:

```
python twtt.py /u/cs401/A1/tweets/justinbieber justinbieber.twt
```

It would be advisable to perform these normalization tasks in the order listed (e.g., you should tag tokens only after punctuation and clitics have been isolated). For preprocessing and tokenization, you may only use standard Python libraries. For tagging, we are providing a tagger for you to use in */u/cs401/A1/tagger/*. Copy the contents of that directory to your working directory and invoke *NLPlib.py* in *twtt.py*. For example,

```
import NLPlib

tagger = NLPlib.NLPlib()
sent = ['tag', 'me']
tags = tagger.tag(sent)
```

The `tag` method takes an array of strings (i.e., the tokens in the order that they appear in the sentence) and returns an ordered list of PoS tags (one for each token). As each tweet is normalized, you should write it out to a new file in your working directory. This output file will be used in Section 2. Table 2 shows an example of an original tweet and its normalized version.

After debugging your program, run it on all files in our Twitter corpus. You may write another script that runs *twtt.py* over all data in */u/cs401/A1/tweets/*, but *twtt.py* must process only one file at a time.

2. Gathering feature information [20 marks]

The second step is to write a Python program named *buildarff.py*, in accordance with the "General specifications" section below, that takes tokenized and tagged tweets from Section 1 and builds an 'arff' datafile that will be used to classify tweets in Section 3. Feature extraction is basically the process of analyzing the preprocessed data in terms of variables that are indicative of the source of the data. For example, if the use of the word *me* is indicative of people rather than news organizations, then the number of first-person pronouns in a tweet will tell you something about whether the tweet came from either, e.g., Justin Bieber or the CBC. The source of the data (i.e., the class) can be an individual Twitter account or a group of accounts (e.g., celebrities can be grouped together into one class, as can news organizations).

The arff file consists of two main sections: the definition of features (attributes) and the instances (data points). Attributes include a name and a type; for this assignment, we will use numeric types except for the class name. Each line in the data section represents a single tweet encoded by a list of numbers separated (i.e., the features) by commas, and the class that produced the tweet, e.g.:

```
0,1,3,0,0...,justinbieber
```

The feature values must appear in the same order in which they are defined in the attribute definition section. An example arff file is shown in Table 5. For details, see the online specification at `http://www.cs.waikato.ac.nz/ml/weka/arff.html`.

The *buildarff.py* program can take a variable number of arguments. If the first argument starts with a hyphen (-), then it indicates the number of tweets from **each** .twt file that will be used to build the arff; all tweets are used if this argument is missing. The next n arguments represent the n classes you are using; classes may contain one or more .twt files joined by pluses (+). In order to specify the name of the class in the arff, a class may be prefaced by an optional class name followed by a colon (:); if there is no classname, the entire class definition is taken as the class name. The last argument is the output arff file. For example:

```
buildarff.py -500 justinbieber.twt britneyspears.twt jbbs.arff
buildarff.py pop:rihanna.twt+katyperry.twt news:bbcnews.twt+cnn.twt popvnews.arff
```

Note: your mark here will depend partially on your implementation. To get full marks, you must implement the feature extraction in a modular way which allows for easy addition of new features with minimal modification.

The features to be gathered

For each tweet, you need to extract 20 features and write these to the arff file. These features are listed in Table 3. Many of these involve counting tokens in a tweet that have certain characteristics that can be discerned from its tag. For example, counting the number of adverbs in a tweet involves counting the number of tokens that have been tagged as RB, RBR, or RBS. Table 4 explicitly defines some of the features in Table 3; these definitions are available on CDF in the directory */u/cs401/Wordlists/*. You may copy and modify these files, but do not change their filenames. Be careful about capitalization; in all cases you should count both capitalized and lower case forms (e.g., both *he* and *He* count towards the number of third person pronouns).

When your feature extractor works, build an arff file, *cnncbc.arff*, that classifies CNN and CBC tweets using the first 100 tweets from each source.

3. Classifying tweets using WEKA [25 marks]

The third step is to use the feature extraction program from Section 2 to classify tweets using the WEKA machine learning package. To use this package, you will invoke WEKA by indicating the data, classifier, and other options on the command line, as described in Appendix 2.

We will now experiment a little. Create a text file, *discussion.txt* to discuss and answer the questions in the following subsections. Each of the following subsections should be given its own section in that file (e.g., by use of headings). In each case, indicate the command line options you used for your experiments.

See Appendix 1 for a description of accuracy, precision, and recall.

3.1 Celebrity potpourri

Build an arff file for distinguishing the following twits: Barack Obama, Stephen Colbert, Ashton Kutcher, Kim Kardashian, Niel deGrasse Tyson, and Shakira (i.e., you have 6 classes). This first experiment involves testing different classification algorithms. Among support vector machines (SVMs), naïve Bayes, and decision trees, which is the best for this task? Evaluate using 10-fold cross-validation. You should mention the accuracies in your discussion. Pipe the output of WEKA for the 10-fold cross-validation of the best classifier to the file *3.1output.txt*. *Use the best classification algorithm for the following questions.*

3.2 Pop stars

Classify the tweets of the following pop stars: Britney Spears, Justin Bieber, Katy Perry, Lady Gaga, Rihanna, and Taylor Swift. Use only the best classifier from section 3.1, with 10-fold cross validation. How does the accuracy compare to that in section 3.1? Now, instead of 10-fold cross-validation, try using the training set as a test set (note: this is not normally done in practice!) and pipe the output of WEKA to the file *3.2output.txt*. Compare accuracies and comment.

3.3 News

Build an arff file to distinguish these news feeds: CBC, CNN, the Toronto Star, Reuters, the New York Times, and the Onion. Are news feeds easier or harder to distinguish from each other, as compared to the pop stars? Calculate the precision and recall for each of the news feeds, based on the 10-fold cross-validation confusion table, and include them in your discussion. Based on these numbers, which of these news feeds would you say is the most distinct from the others? Which is the least distinct?

3.4 Pop stars versus news

Build an arff with two classes, the pop stars from 3.2 in one class and the news feeds from 3.3 in the other. How does the 10-fold cross-validation accuracy compare to the other sections? Is such a comparison valid? Why or why not? Build another arff that uses only half the data available (use the first 500 from each file) and pipe the output of WEKA to the file *3.4output.txt*. Is there a difference in performance? What if you use some other proportion of the data? Based on this, do you think getting more twitter data would greatly improve performance?

3.5 Feature analysis

For the first arff in each of the previous sections, run WEKA's information gain attribute selector. Pipe the output of WEKA for the first arff from section 3.3 to the file *3.5output.txt*. Is there one feature that is especially useful in all tasks? In the discussion, note a few commonalities and differences that you see among the different tasks, in terms of the most and least informative features. Provide a possible explanation as to why this might be.

4. Bonus [15 marks]

We will give up to 15 bonus marks for innovative work going substantially beyond the minimal requirements. These marks can make up for marks lost in other sections of the assignment, but your overall mark for this assignment cannot exceed 100%. The obtainable bonus marks will depend on the complexity of the undertaking, and are at the discretion of the marker. Imporantly, your bonus work should not affect our ability to mark the main body of the assignment in any way.

You may decide to pursue any number of tasks of your own design related to this assignment, although you should consult with the instructor or the TA before embarking on such exploration. Certainly, the rest of the assignment takes higher priority. Some ideas:

- Identify words that the PoS tagger tags incorrectly and add code that fixes those mistakes. Does this code introduce new errors elsewhere? E.g., if you always tag *dog* as a noun to correct a mistake, you will encounter errors when *dog* should be a verb. How can you mitigate such errors?
- For the task in section 4.4, hypothesize and test what would happen if you train models with data from 5 pop stars and 5 newspapers and test with data from the remaining (held-out) pair. Does it matter which data is held out?
- Explore alternative features to those extracted in section 2. What other kinds of variables would be useful in distinguishing between celebrities, or between people and organizations? Test your features empirically as you did in section 3.5 and discuss your findings.
- Explore alternative classification methods to those used in section 3.1. Explore different parameters that control those methods. Which parameters give the best empirical performance, and why? It may help to explore the WEKA GUI for this task, as described in Appendix 2.

4 General specifications

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

If a program uses a file, such as the word lists, whose name is specified within the program, the file must be read either from the directory in which the program is being executed, or from a subdirectory of */u/cs401* whose path is completely specified in the program. 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.

5 Submission requirements

This assignment is submitted electronically. You should submit:

1. All your code for *twtt.py* and *buildarff.py* (including helper scripts, if any).
2. The tagged and tokenized file *CBCNews.twt* from section 1. **Do not submit other twt files.**
3. The *cnncbc.arff* file produced in section 2.
4. *3.1output.txt*: The 10-fold cross-validation output from WEKA for the best classifier in section 3.1.
5. *3.2output.txt*: The test-on-training-data output from section 3.2.
6. *3.4output.txt*: The (10-fold cross-validation) output using half the training data from 3.4.
7. *3.5output.txt*: The information gain feature selection output, using the arff from 3.3.
8. Any lists of words that you modified from the original version.
9. Your discussion, *discussion.txt*.

In another file called *ID* (available on the course web page), provide the following information:

1. your first and last name.
2. your student number.
3. your CDF 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 CDF with the *submit* command:

`submit -N a1 csc401h` *filename filename ...*

Do **not** `tar` or `compress` your files, and do not place your files in subdirectories. Do not format your discussion as a PDF or Word document — plain text only.

6 Working outside the lab

If you want to do some or all of this assignment on your laptop or home computer, for example, you will have to do the extra work of downloading and installing the requisite software and data. You take on the risk that your computer might not be adequate for the task. You are strongly advised to upload regular backups of your work to CDF, so that if your home machine fails or proves to be inadequate, you can immediately continue working on the assignment at CDF. When you have completed the assignment, you should try your programs out on CDF to make sure that they run correctly there. **A submission that does not work on CDF will get zero marks.**

The course web page, *http://www.cs.toronto.edu/∼csc401h*, will offer links to versions of Python for various systems, the WEKA software suite, a gzipped file of the tweet corpus, and the lists of word-category definitions. WEKA can be run on any system with a recent Java interpreter.

Appendix 1: Accuracy, precision, and recall

WEKA does not compute precision and recall for you, but it does give you a 'confusion table', C, summarizing how the tweets in the test data were classified. These figures allow you to compute accuracy, precision and recall yourself. Here is an example table:

```
 (a)  (b)  (c)  (d)      <- classified as
---- ---- ---- ----
 132    6    9    4      (a): class I
   2   75    1    6      (b): class II
   8    9  119   12      (c): class III
   8    3    4   97      (d): class IV
```

An element at row i, column j (i.e., $c_{i,j}$) is the number of instances belonging to class i that were classified as class j. For example, here 75 tweets belonging to class II were correctly classified, but 1 tweet belonging to class II was mistakenly classified as class III.

Here, **accuracy**, A is the overall measure of the quality of the classification. Specifically, it is the total number of correctly classified instances over all classifications, or

$$A = \frac{\sum_i c_{i,i}}{\sum_{i,j} c_{i,j}}.$$

Precision and recall have to be measured separately for each of the classes. For class κ, **precision** $Prec$ is the fraction of cases classified as κ that truly are κ. In other words, precision is, for each column, the ratio of the diagonal element to the sum of the column,

$$Prec(j) = \frac{c_{j,j}}{\sum_i c_{i,j}}.$$

For each row, the **recall** Rec is the fraction of cases that are truly κ that were classified as κ,

$$Rec(i) = \frac{c_{i,i}}{\sum_j c_{i,j}}.$$

If need be, the sets of precision measures and recall measures can then each be averaged over the classes to get an overall estimate of the precision and recall of the method.

Appendix 2: Using WEKA

WEKA is available on CDF in */u/cs401/WEKA*.

1. Classifier command line

The general syntax for running WEKA from the command line is:

`java -cp` **weka.jar path* *weka classifier* *classifier options**

The path to the jar file on CDF is: `/u/cs401/WEKA/weka.jar`. The three classifiers you will use are:

weka.classifiers.functions.SMO (for SVMs)
weka.classifiers.bayes.NaiveBayes (for Naïve Bayes)
weka.classifiers.trees.J48 (for decision trees)

The key options are:

-t *arff file* (Choose the training file)
-*x* *X* (Use cross-validation, X is the number of folds)
-T *arff file* (Choose the testing file, for section 3.2)
-no-cv (Do not perform cross-validation)
-o (Do not output the classifier. Use this option for your outputs!)

A full list of options are available at `http://www.cs.waikato.ac.nz/~remco/weka_bn/node13.html`, but this is all you need to complete this assignment.

2. Information gain attribute selector

The options for running WEKA's information gain attribute selector (Section 3.4) are different than the classifiers and a bit more complex; ideally, attribute selection is done through the WEKA GUI (and you may do so, see below). Since there is no variation in the parameters, however, we can simply provide you with a sh script that will do the job. To run it on one of your arff files and send the output to a file, use this command:

`sh /u/cs401/WEKA/infogain.sh` **arff file** `>` **output file**

3. WEKA GUI

WEKA includes an excellent GUI for quickly testing various options associated with classification. Since you need to submit correct command line arguments in your discussion, we do not recommend that you use the GUI for producing your final output; however, you may use the GUI for early testing and additional exploration as part of the bonus. If you are on a CDF lab computer or logged in using NX, the GUI is available using the following command:

```
java -jar /u/cs401/WEKA/weka.jar
```

Select the explorer, and open your arff file using the open file button. Then go to the Classify tab to carry out a classification experiment, or Select Attributes to do feature selection. The options which can be selected otherwise correspond to those available at the command line. See
`http://www.cs.waikato.ac.nz/ml/weka/index_documentation.html` for more information.

Table 1a: The Penn part-of-speech tagset—words

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 1b: The Penn part-of-speech tagset—punctuation

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 2: Conversion from raw tweets to tagged tweets

Raw tweet:

```
Meet me today at the FEC in DC at 4. Wear a carnation so I know
it’s you. <a href="Http://bit.ly/PACattack" target="_blank"
class="tweet-url web" rel="nofollow">Http://bit.ly/PACattack</a>.
```

Output from *twtt.py*:

```
...
|
Meet/VB me/PRP today/NN at/IN the/DT FEC/NN in/IN DC/NN at/IN 4/NN ./.
Wear/VB a/DT carnation/NN so/RB I/PRP know/VB it/PRP ’s/POS you/PRP ./.
|
...
```

Table 3: Features to be computed for each text

- Counts:
  - — First person pronouns
  - — Second person pronouns
  - — Third person pronouns
  - — Coordinating conjunctions
  - — Past-tense verbs
  - — Future-tense verbs
  - — Commas
  - — Colons and semi-colons
  - — Dashes
  - — Parentheses
  - — Ellipses
  - — Common nouns
  - — Proper nouns
  - — Adverbs
  - — *wh*-words
  - — Modern slang acroynms
  - — Words all in upper case (at least 2 letters long)
- Average length of sentences (in tokens)
- Average length of tokens, excluding punctuation tokens (in characters)
- Number of sentences

Table 4: Definitions of feature categories

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, ttys, afn, bbs, cya, ez, f2f, gtr, ic, jk, k, ly, ya, nm, np, plz, ru, so, tc, tmi, ym, ur, u, sol

Table 5: Example `weather.arff` for WEKA

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```