**X = country**

**Y = tweet**

Country = us Tweet = @Addictd2Success thx u for following

Country = us Tweet = Let's just say, if I were to ever switch teams, Khalesi would be top of the list. #girlcrush

Country = ph Tweet = Taemin jonghyun!!! **Your** birits make me go~ <http://t.co/le8z3dntlA>

Country = id Tweet = depart.senior 👻 rapat perdana (with Nyayu, Anita, and 8 others at Ruang Aescullap FK Unsri Madang) — <https://t.co/swRALlNkrQ>

Country = ph Tweet = Done with internship with this pretty little lady!  (@ Metropolitan Medical Center w/ 3 others) [pic]: <http://t.co/1qH61R1t5r>

Country = gb Tweet = Wow just Boruc's clanger! Haha Sunday League stuff that, Giroud couldn't believe his luck! #clown

Country = my Tweet = I'm at Sushi Zanmai (Petaling Jaya, Selangor) w/ 5 others <http://t.co/bcNobykZ>

Country = us Tweet = Mega Fest!!!! Its going down🙏🙌 @BishopJakes

Country = gb Tweet = @EllexxxPharrell wow love the pic babe xx

Country = us Tweet = You have no clue how much you hurt me

**Question 1 (1.0 mark)**

**Instructions**: Next we need to preprocess the collected tweets to create a bag-of-words representation.

The preprocessing steps required here are:

(1) tokenize each tweet into individual word tokens (using NLTK TweetTokenizer);

(2) lowercase all words;

(3) remove any word that does not contain any English alphabets (e.g. {\_hello\_, \_#okay\_, \_abc123\_} would be kept, but not {\_123\_, \_!!\_}) and

(4) remove stopwords (based on NLTK stopwords). An empty tweet (after preprocessing) and its country label should be **excluded** from the output **(x\_processed and y\_processed).**

**Task**: Complete the preprocess\_data(data, labels) function. The function takes **a list of tweets** and **a corresponding list of country labels** as input, and returns **two lists**. For the first list, each element is a bag-of-words representation (dictionary) of a tweet. For the second list, each element is a corresponding country label. Note that while we do not need to preprocess the country labels (y), we need to have a new output list (y\_processed) because **some tweets maybe removed after the preprocessing (due to having an empty set of bag-of-words**).

**Check**: Use the assertion statements in **"For your testing"** below for the expected output.

**Question 2 (1.0 mark)**

**Instructions**: Our task here to tokenize the hashtags, by implementing the **MaxMatch algorithm** discussed in class.

NLTK has a list of words that you can use for matching, see starter code below (words). Be careful about efficiency with respect to doing word lookups.

One extra challenge you have to deal with is that the provided list of words (words) includes only lemmas: your MaxMatch algorithm should match inflected forms by converting them into lemmas using the NLTK **lemmatizer** before matching (provided by the function lemmatize(word)). Note that the list of words (words) is the only source that you'll use for matching (i.e. you do not need to find other external word lists). If you are unable to make any longer match, your code should default to matching a single letter.

For example, given "#newrecord", the algorithm should produce: ["#", "new", "record"].

**Task**: Complete the tokenize\_hashtags(hashtags) function by implementing the MaxMatch algorithm.

The function takes as input **a set of hashtags**, and returns **a dictionary** where key="hashtag" and value="a list of tokenised words".

**Check**: Use the assertion statements in **"For your testing"** below for the expected output.

### Question 4 (1.0 mark)

**Instructions**: The two versions of MaxMatch will produce different results for some of the hashtags.

For a hashtag that has different results, our task here is to use a **unigram language model** (lecture 3) **to score them to see which is better**. Recall that in a unigram language model we compute P(*#*, *hello*, *world* = **P(*#*)\*P(*hellow*)\*P(*world*).**

You should:

(1) **use the NLTK's Brown corpus (brown\_words) for collecting word frequencies** (note: the words are already tokenised so no further tokenisation is needed);

(2) lowercase all words in the corpus;

(3) use add-one smoothing when computing the unigram probabilities; and

(4) work in the log space to prevent numerical underflow.

**Task**: Build a unigram language model with add-one smoothing using the word counts from the Brown corpus.

Iterate through the hashtags, and for each hashtag where MaxMatch and reversed MaxMatch produce different results,

print the following: (1) the hashtag; (2) the results produced by MaxMatch and reversed MaxMatch; and

(3) the log probability of each result as given by the unigram language model. Note: you **do not** need to print the hashtags where MaxMatch and reversed MaxMatch produce the same results.

An example output:

1. #abcd

MaxMatch = [#, a, bc, d]; LogProb = -2.3

Reversed MaxMatch = [#, a, b, cd]; LogProb = -3.5

2. #efgh

MaxMatch = [#, ef, g, h]; LogProb = -4.2

Reversed MaxMatch = [#, e, fgh]; LogProb = -3.1

Have a look at the output, and see if the sequences with better language model scores (i.e. less negative) are generally more coherent.

### Question 5 (1.0 mark)

**Instructions**: Here we are interested to do text classification, to predict the country of origin of a given tweet.

The task here is to create training, development and test partitions from the preprocessed data (x\_processed) and

training, development and test 70%/15%/15% x\_train, x\_dev, x\_test feature vectors y\_train, y\_dev and y\_test

convert the bag-of-words representation into feature vectors.

**Task**: Create training, development and test partitions with a 70%/15%/15% ratio. Remember to preserve the ratio of the classes for all your partitions. That is, say we have only 2 classes and 70% of instances are labelled class A and 30% of instances are labelled class B, then the instances in training, development and test partitions should also preserve this 7:3 ratio. You may use sklearn's builtin functions for doing data partitioning.

Next, turn the bag-of-words dictionary of each tweet into a feature vector. You may also use sklearn's builtin functions for doing this.

You should produce 6 objects: x\_train, x\_dev, x\_test which contain the input feature vectors, and y\_train, y\_dev and y\_test which contain the labels.

### Question 6 (1.0 mark)

**Instructions**: Now, let's build some classifiers. Here, we'll be comparing Naive Bayes and Logistic Regression.

For each, you need to first find a good value for their main regularisation hyper-parameters, which you should identify using the scikit-learn docs or other resources.

Use the development set you created for this tuning process; do **not** use cross-validation in the training set, or involve the test set in any way. You don't need to show all your work, but you do need to print out the **accuracy** with enough different settings to strongly suggest you have found an optimal or near-optimal choice. We should not need to look at your code to interpret the output.

**Task**: Implement two text classifiers: Naive Bayes and Logistic Regression. Tune the hyper-parameters of these classifiers and print the task performance (accuracy) for different hyper-parameter settings.

### Question 7 (1.0 mark)

**Instructions**: Using the best settings you have found, compare the two classifiers based on performance in the test set.

**Print out both accuracy and macro-averaged F-score for each classifier. Be sure to label your output.** You may use sklearn's inbuilt functions.

**Task**: Compute test performance in terms of accuracy and macro-averaged F-score for both Naive Bayes and Logistic Regression, using their optimal hyper-parameter settings based on their development performance.

### Question 8 (1.0 mark)

**Instructions**: Print the most important features and their weights for each class for the two classifiers.

**Task**: For each of the classifiers (Logistic Regression and Naive Bayes) you've built in the previous question, **print out the top-20 features (words) with the highest weight for each class (countries).**

An example output:

Classifier = Logistic Regression

Country = au

aaa (0.999) bbb (0.888) ccc (0.777) ...

Country = ca

aaa (0.999) bbb (0.888) ccc (0.777) ...

Classifier = Naive Bayes

Country = au

aaa (-1.0) bbb (-2.0) ccc (-3.0) ...

Country = ca

aaa (-1.0) bbb (-2.0) ccc (-3.0) ...

Have a look at the output, and see if you notice any trend/pattern in the words for each country.