# CSCI544, Fall 2016: Assignment 2

## Due Date: October 28st, before 4pm.

## Introduction

The goal of this assignment is to get some experience implementing the simple but effective machine learning model, the perceptron, and applying it to a binary text classification task (i.e., spam detection). This assignment will describe the details of a particular implementation of a standard binary perceptron classifier, and an averaged binary perceptron classifier with the categories spam and ham (i.e., not spam). The teaching assistants have created a solution following these same directions, and your work will be evaluated based on its performance on an unseen test set in comparison to this solution. In addition, we ask you to submit a short report describing your experiments with the perceptron.

1. The performance of your standard perceptron and averaged perceptron on the development data.
2. The performance of your standard perceptron and averaged perceptron on the development data with models built from only 10% of the training data.

You will use Vocareum to submit your assignment, and later to check your grade. You can submit your assignment as many times as you like, although we will grade the last version submitted applying any appropriate late penalties. So only submit after the deadline if you think your new version will result in a better grade after the late penalty. Each time you submit your code, Vocareum will run it, perform some basic checks, and tell you the results.

The email data is the same as assignment 1. It is already on Vocareum, and you should have already downloaded a copy via Blackboard. Remember that the Blackboard data contains a directory "__MACOSX" with metadata that you should ignore. There is training data, development data and test data. When you submit your assignment, Vocareum will do the following:

- Run your per_learn.py script (see below) on the entire training set.
- Run your per_classify.py script (see below) on the development data.
- Evaluate the output of per_classify.py, and report performance, or report any errors.
- Run your avg_per_learn.py script (see below) on the entire training set.
- Run your per_classify.py script on the development data.
- Evaluate the output of per_classify.py, and report performance, or report any errors.

You will need to work locally on your own computer to run some of the experiments described here. However, remember that the majority of your grade will depend on how your code performs on Vocareum not your computer. You will be writing your code in Python3. Vocareum is currently using version 3.4.3 of Python.

# Perceptron Classifiers in Python

You will write the following programs:

**per_learn.py** will learn a perceptron model from labeled data using the standard training algorithm.

**avg_per_learn.py** will learn a perceptron model from labeled data using the averaged perceptron training algorithm.

**per_classify.py** will use a stored perceptron model to classify new data.

per_learn.py and avg_per_learn.py will be invoked in the following way:

>python3 per_learn.py /path/to/input

And

>python3 avg_per_learn.py /path/to/input

Please follow these guidelines:

1. /path/to/input is a data directory containing no spaces. The script should search through the directory recursively looking for subdirectories containing the folders: "ham" and "spam". Note that there can be multiple "ham" and "spam" folders in the data directory. Emails are stored in files with the extension ".txt" under these directories. The file structure of the training and development data on Blackboard is exactly the same as the data on Vocareum although these directories (i.e., "dev" and "train") will be located in different places on Vocareum and on your personal computer.
2. "ham" and "spam" folders contain emails failing into the category of the folder name (i.e., a spam folder will contain only spam emails and a ham folder will contain only ham emails). Each email is stored in a separate text file. The emails have been preprocessed removing HTML tags, and leaving only the body and the subject. The files have been tokenized such that white space always separates tokens. Note, in the subject line, "Subject:" is considered as one token. The text has already been converted to lower case except for the initial "Subject:". You should consider all the tokens and all the characters in those tokens when building a vocabulary list even non alpha-numeric characters.
3. Because of special characters in the corpus, you should use the following command to read files: open(filename, "r", encoding="latin1") and correspondingly use the same encoding when writing model files: open(filename, "w", encoding="latin1").
4. For this assignment, you are required to use tokens as features. During testing, the official solution will ignore unknown tokens not seen in training (i.e., pretend they did not occur). Smoothing does not apply since your perceptrons are not calculating probabilities.

per_learn.py and avg_per_learn.py will learn a perceptron model from the training data, and write the model parameters to a file called **per_model.txt** in the current working directory. The format of the model is up to you, but it should contain sufficient information for per_classify.py to successfully classify new data. You are allowed to use pickle to save your data structures to per_model.txt in binary form. You are also allowed to use the standard python json library.

per_classify.py will be invoked in the following way:

>python3 per_classify.py /path/to/input output_filename

The first argument is again a data directory but you should not make any assumptions about the structure of the directory. Instead, you should search the directory for files with the extension ".txt". per_classify.py should read the parameters of a perceptron model from the file per_model.txt, and classify each ".txt" file in the data directory as "ham" or "spam", and write the result to a text file whose name is given in the second argument in the following format:

LABEL path_1
LABEL path_2
⋮

In the above format, LABEL is either “spam” or “ham” and path is the absolute path to the file, and the filename (e.g., on Windows a path might be: "C:\dev\4\ham\0001.2000-01-17.beck.ham.txt"). The order of lines in the file doesn't matter and case doesn't matter for the label.

NOTE: This is a change from the first assignment in that the name of the output file is given as the second command line argument rather than a hardcoded file name. This gives you and us the flexibility to save different output files rather than have them overwritten each time per_classify.py is run.

Another change is that we are no longer estimating probabilities but rather adjusting weights meaning that we have to iterate over the training data more than once. It is a difficult problem to decide how many times to iterate over the training data. Too few iterations mean the weights have not been sufficiently adjusted and you will get poor performance. Too many iterations mean the weights are too specific to the training data (i.e., "overfitting") and you will get poor performance on the test data. Another issue with trying to get perfect performance on the training data is that sometimes it is impossible (i.e., the data is not linearly separable) and you would end up adjusting the weights forever.

Generally, researchers experiment with different numbers of training iterations, and may stop training early if they see a performance plateau. Because you will be evaluated based on performance on unseen data, we are telling you the number of training iterations to use:

- For per_learn.py: 20.
- For avg_per_learn.py: 30.

Another issue when adjusting weights is that the order of training examples matters even for the averaged perceptron. If we have a large batch of examples with the same label (e.g., "ham"), then there will be some initial weight adjustments but then the perceptron will stop learning since all the examples are "ham". If we then move to a large batch of examples with the opposite label (e.g., "spam"), there will again be some initial weight adjustments but then the perceptron will stop learning since all the examples are "spam". To avoid this, you should shuffle the order of the training examples for each training iteration.

As in assignment 1, you will need to be able to calculate precision, recall and F1 score assuming that you run per_classify.py on a labeled corpus (i.e., a data directory with "ham" and "spam" subdirectories). You can write additional Python script(s) or modify per_classify.py to include this functionality. You must include this functionality in the code you submit. Otherwise, we will apply a penalty to your grade. You cannot rely upon the submission script on Vocareum to calculate precision, recall and F1 score for you.

However, if you modify per_classify.py, it must still work as shown above so that we can test (i.e., we should be able to run it by simply including the data directory and the output filename as the sole command line arguments, and it should still work even if there are no "ham" and "spam" subdirectories). You can print the numbers to the screen, but do not print them to the output file.

You will also need a way to train a model with 10% of the training data. You need to be careful how you divide the data (e.g., it would be a disaster if your training data only had one class). We recommend calculating a target number of files (i.e., 10% of the total) and selecting half of those files to be ham examples, and half to be spam examples. You can do this manually (e.g., copying files to create a smaller training directory) or by writing Python code (i.e., separate scripts or modifying per_learn.py and avg_per_learn.py). If you modify per_learn.py and avg_per_learn.py, they must still work as shown above so that we can test your code (i.e., we should be able to run it by simply including the data directory as the sole command line argument, and by default, it should use all the data for training the model).

## What to turn in and Grading

You need to submit the following on Vocareum:

- **PerceptronReport.txt**.
    - Download and fill in the template from the class website. Include the results from your experiments as well as a description of how you calculated performance, and created a 10% training set.
    - Make sure that none of your code writes to a file called PerceptronReport.txt. We don't want to accidently damage the file when we test your code.
    - You can round off the values to 2 decimal points, but rounding is not mandatory.
- **All your code**: per_learn.py, avg_per_learn.py, per_classify.py and any additional code you created for the assignment. This is required and the academic honesty policy (see rules below) applies to all your code. Do not submit data even if you manually created your 10% training corpus.

10% of your grade (10 points) is based on the report. Make sure to answer every question. The other 90% is based on the performance of your code compared to the official solution written by the teaching assistants following the directions above.

Your performance will be measured based on models output by your per_learn.py and avg_per_learn.py scripts and used by your per_classify.py script. We will calculate F1 score for ham and F1 for spam, and compute a weighted average F1 score (WA):

(Number_of_Spam_examples * Spam_F1_score)/Total_number_of_examples +
(Number_of_Ham_examples * Ham_F1_score)/ Total_number_of_examples

If your performance is the same or better than the official solution written by the TA, then you receive full credit. Otherwise, your score is proportional to your relative performance:

(1 – (WA_TA_per_learn – WA_your_per_learn ) /WA_TA_per_learn ) * 45 points +

(1– (WA_TA_avg_per_learn – WA_your_avg_per_learn ) / WA_TA_avg_per_learn ) * 45 points

## Late Policy

- On time (no penalty): before October 28th, 4pm.
- One day late (10% penalty): before October 29th, 4pm.
- Two days late (20% penalty): before October 30th, 4pm.
- Three days late (30% penalty): before October 31st, 4pm.
- Zero credit after October 31st, 4pm.

## Other Rules

- DO NOT look for any kind of help on the web outside of the Python documentation.
- DO NOT use any external libraries other than the default Python libraries.
- Questions should go to Piazza first not email. This lets any of the TAs or instructors answer, and lets all students see the answer.
- When posting to Piazza, please check first that your question has not been answered in another thread.
- DO NOT wait until the last minute to work on the assignment. You may get stuck and last minute Piazza posts may not be answered in time.
- This is an individual assignment. DO NOT work in teams or collaborate with others. You must be the sole author of 100% of the code and writing that you turn in.
- DO NOT use code you find online or anywhere else.
- DO NOT turn in material you did not create.
- All cases of cheating or academic dishonesty will be dealt with according to University policy.

## FAQ

1. **The code runs on my computer but not on Vocareum.**
    a. Vocareum has been tested for the same course in the previous terms and has been shown to work well. If your code doesn't run then there are some issues with your code,

and you are responsible for correcting them. Test your code by submitting it on Vocareum incrementally and make sure it works.

2. **How will the TAs/Professors grade my HW?**
    a. We will be writing scripts which will run on Vocareum. Since we automate the process of grading, make sure that you write your code to match the output format "exactly". If you do not do this there will be a penalty. You have been warned!

3. **I found a piece of code on the web. Is it ok to use it?**
    a. No! We run plagiarism checks on the code you write. The plagiarism detector is a smart algorithm which can tell if you've copied the code from elsewhere/others. If you're caught copying the code, strict measures will be enforced. No exceptions!

    b. Please contact TAs/Professors during the office hours with your questions and we will make sure you understand the concepts and avoid unpleasant circumstances.

4. **Vocareum terminates my code before it finishes, but it finishes executing on my computer.**
    a. This usually means that your implementation is inefficient. Keep an eye out for places where you iterate. Run time issues can be solved by using efficient data structures (Hashmap, Hashset etc.).

5. **I assumed my code will run on Vocareum, but I wasn't proactive to test my code earlier. I couldn't meet the deadline because of this.**
    a. The late policy above still applies, and is automatically enforced by the system. Submit your code incrementally to make sure it functions on Vocareum. You have been warned!

6. **How do I log into Vocareum ?**
    a. You will receive an email with instructions. Sometimes, if you have been a part of a course where Vocareum was used previously, you will see the course listed once you log in. Please send the TAs an email if you have problems.

7. **When are the office hours ?**
    a. Please refer to the course website. http://nld.ict.usc.edu/cs544-fall2016/

8. **Like assignment 1, should I try to improve performance by handling unknown words differently, or handling certain tokens differently?**
    a. No, this is not part of assignment 2. Add every distinct sequence of characters (i.e., token) seen in training to your vocabulary.

9. **What about smoothing?**
    a. There is no smoothing because perceptrons do not calculate probabilities. You will create a vocabulary based on the training data, and initialize perceptron weights for

each token in the vocabulary. During testing, if there is an unknown token simply skip it. Otherwise, look up the appropriate weights.

10. **What should I do if a document contains multiple instances of a token?**
    a. The features that should be used are counts of occurrences of each token in the document. For example, suppose that the document used for training is “pharmacy for pharmacy” (labeled as “spam”), the weights for “pharmacy” and “for” are -2 and -1 respectively, and bias = 0. Then the activation is going to be “$\mathbf{w} \bullet \mathbf{x} + b = (-2)*2 - 1 + 0 = -5$” and the weights are going to be updated as follows: weight-for-word-pharmacy = -2 + 2*1 = 0 and weight-for-word-for = -1 + 1 = 0. Likewise for testing.

## Reference

H. Daumé III. (2012). The Perceptron. In A Course in Machine Learning. http://ciml.info/dl/v0_8/ciml-v0_8-ch03.pdf