B551 Assignment 4: Machine Learning

Fall 2020
Due: Tuesday December 15, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you a chance to practice some machine learning techniques.

Coding requirements. For fairness and efficiency, we use a semi-automatic program to grade your submissions. As usual, we require that: 1. You must code this assignment in Python 3; 2. You should test your code on one of the SICE Linux systems; 3. Your code must obey the input and output specifications given below. 4. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and data structures, as long as they are already installed on the SICE Linux servers; and 5. Make sure to use the program file name we specify.

Groups. For this assignment, we ask everyone to work alone.

Coding style and documentation. We will not explicitly grade coding style, but it's important that you write your code in a way that we can easily understand it. Please use descriptive variable and function names, and use comments when needed to help us understand code that is not obvious. For each of these problems, you will face some design decisions along the way. Your primary goal is to write clear code that finds the correct solution in a reasonable amount of time. To encourage innovation, we will conduct a competition among programs to see which can solve the hardest problems in the shortest amount of time.

Report. Please put a report describing your assignment in the Readme.md file in your Github repository. For each problem, please include: (1) a description of how you formulated each problem; (2) a brief description of how your program works; (3) and discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made. These comments are especially important if your code does not work perfectly, since it is a chance to document the energy and thought you put into your solution.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may consult printed and/or online references, but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

Part 1: Tweet classification

Let's examine one particular problem where machine learning may be helpful: estimating where a Twitter "tweet" was sent, based only on the content of the tweet itself. We'll use a bag-of-words model, which means that we won't model the order of words or grammatical structure. In other words, a tweet can be modeled as simply a histogram over the words of the English language (or, more generally, all possible tokens that occur on Twitter). If, for example, there are 100,000 possible tokens on Twitter, then a tweet can be represented as a 100,000-dimensional binary vector, where in each dimension there is a 1 if the word appears in the tweet and a zero otherwise. Of course, vectors will be very sparse (most entries are zero).

1. To get started, clone the github repository using one of the two commands:

   `git clone git@github.iu.edu:cs-b551-fa2020/`*your-repo-name*`-a4`
   `git clone https://github.iu.edu/cs-b551-fa2020/`*your-repo-name*`-a4`

   To get you started, we've provided a dataset in your GitHub repo of tweets, labeled with their actual geographic locations, split into training and testing sets. We've restricted to a set of a dozen cities, so your task is to classify each tweet into one of twelve categories. The file format of the training and testing files is simple: one tweet per line, with the first word of the line indicating the actual location.

2. Learn a Naive Bayes classifier for this problem. For a given tweet D, your classifier will evaluate $P(L = l|w_1, w_2, ..., w_n)$, the posterior probability that a tweet was taken at one particular location (e.g., $l =$ Chicago) given the words in that tweet. Make the Naive Bayes assumption, which says that for any $i \neq j$, w_i is independent from w_j given L. When run like this:

   `python3 ./geolocate.py train bayes` *`training-input-file bayes-model-output-file`*

   your program should use the training tweets in *training-input-file* to learn the parameters of the Naive Bayes classifier, and then output the trained model to *bayes-model-output-file*. Your training program should also output (to the screen, in a human-readable format of your choice) the top 5 words associated with each of the 12 locations that are also not uncommon (i.e. the words for which $P(L = l|w)$ is the highest for each l subject to the constraint that w occurs at least 5 times in the training set).

3. Learn a Decision Tree for this problem. At each node, the decision tree will check for the presence or absence of a given word and branch accordingly. Of course, the structure of the tree (and the words to check) should be learned automatically; when run like this:

   `python3 ./geolocate.py train dtree` *`training-input-file dtree-model-output-file`*

   your program should use the training tweets in the training file to learn a Decision Tree classifier, and write the learned tree to the given output file. (You can set the number of levels of the tree (and other parameters) ahead of time (i.e., hard-coded in your training code), but you'll probably want to experiment to get values that work reasonably well.) Your program should also output (to the screen, in a human-readable format of your choice) the top three levels of the decision tree.

4. Now implement the classifiers for each of the above two models (Bayes and Decision Tree). In testing mode, your program should accept the name of trained model file (either bayes or dtree), apply the classifier to each tweet in the a given testing data file, and output its decisions (estimated geolocations):

   `python3 ./geolocate.py test` *`model-input-file testing-input-file testing-output-file`*

   The format of the *t*esting-output-file should be very similar to the *t*esting-input-file, except that the first word of each line should be your estimated label, the second word should be the actual correct label, and the rest of the line should be the tweet itself. Your program should output (to the screen, in a human-readable format) the accuracy (percentage of tweets correctly classified).

The goal is to get as high an accuracy as possible in testing, including on the separate train and test dataset we'll use to test your code. You can assume that any training and test files we use will have 12 categories, but the categories may differ from the ones in the datasets in your repo. You'll have to make various design decisions in doing this, e.g. whether to use all "words" (i.e. Twitter tokens) or just the most common ones, whether to keep punctuation and capitalization or remove it, how many levels to use in the Decision Tree, etc. Please describe these design decisions and any experimentation you use to arrive at them in your report.

Hints: Don't worry, at least at first, about whether the "words" in your model are actually words. Just treat every unique space-delimited token you encounter as a "word," even if it's misspelled, a number, a punctuation mark, etc. It may be helpful to ignore tokens that do not occur more than a handful of times, however. The model files may be stored in any format that you'd like, as long as they can be successfully read by your testing code. They do not have to be human-readable.

What to turn in. Turn in by putting the finished version on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.