Machine Exercise 04 – NAIVE BAYES

REPORT | Dana Kathleen Redeña

Introduction:

The process of making this machine exercise is long and brutal. Many implementations were done but the submitted files only include 2 versions that vary in their final dictionaries. The 1st version uses certain keywords that represents certain strings that occur frequently as a "pattern" and these keywords are listed below and removes the words that only occur once. The 2nd version simply takes the words that occur in all the files. Both versions exclude words in the "exception.txt" file which contains common words in the English dictionary (e.g articles, conjunctions) and a "word" refers to a string that contains only alphabets that may be subdivided by dashes or underscores.

Implementation:

Programming Language: Python 3

For building the dictionary:

- "dictionary.py" - Reads and collects all the words in the training files (under "data") and put it into "dictionary.txt". This excludes all the words in the "exception.txt" which contains common English words. For this step, word is considered if it is either separated by a white space or a comma. The words are then changed to their lowercase form.
- "sorter.py" - Reads "dictionary.txt" and sorts (by using python's *.sort()*) all the words in it, making sure that each word appears only once (by using python's *set* function). The sorted words will then be listed in "dictionary-sorted.txt".
- (Can be skipped for Version 2) "filter.py" - Reads through "dictionary-sorted.txt" and classifies certain "words"/strings according to a common format (e.g. strings who have "mb/s" under "_mb/s_") and stores it in "filtered.txt". This step uses python's regex module for classifying which strings belong to a format.

String code	**Explanation**
_n--n_	Two number sequences separated by 2 dashes
_n-n_	Two number sequences separated by a dash
date-	Date format separated by dashes
tel	Typical telephone number formats: either XXX-XXX-XXXX or XXX-XX-XX
serial	Typical serial number sequence: XXX-XXXX-XX

_n-n-n_	Three number sequences separated by dashes
isbn	Typical ISBN format
fax	Typical fax number format: 1-XXX-XXX-XXXX
_n-n+_	N number sequences separated by dashes
_mb/s_	String with substring “mb/s”
dimen	Number sequences separated by x’s presumably for dimensions
each	String with substring “each”
mph	String with substring “mph”
bin	Following a sequences of eight 0 or 1’s ending with a b
time	Typical time formats, may or may not have “pm”/”am”
_n:n+_	N number sequences separated by colons
email	Typical email formats (@)
date/	Date format separated by slashes
_n/n+_	N number sequences separated by slashes
_nh_	A sequence of N numbers ending with an h
pne	String with substring “”pne”
_f9_	String with substring “f9”
_@_	String with substring “@(“”
_b8_	String with substring “b8”
_x(_	x followed by a sequences of N letters ending with (
bhj	String with substring “bhj”
_7e_	String with substring “7e”
dots	N letter sequences separated by periods
func	Typical programming language function syntax

- "sorter-2.py" - Reads "filtered.txt" and sorts all the words in it, making sure that each word/string appears only once. The sorted words will then be listed in "dictionary-sorted.txt" and "f-dictionary.txt"

For the preparation of getting the probabilities:

- "counter.py" - counts the number of times that each word/string in the "f-dictionary.txt" occurs in all the files under the "data" subfolder. The occurrences are recorded in the "occ_count.txt". Version 1 still uses the same special word classification
- "category.py" - counts the number of times a certain category/classification appears in the training files and puts it in "cat_count.txt"

For the Naive Bayes (the hard stuff):

- "bayes.py" - builds an array for the probabilities of the words in the final dictionary occurring for category, as well as the complement probabilities. The probabilities will then be lambda smoothed according to lambda equals 0.01, 0.1, 0.2, 0.5, 1.0. Runs through the files under the "classify" subfolder to see which words/strings that belong to the final dictionary appears in each file. Version 1 still uses the same special word classification. Performs Maximum A Posteriori on the files under "classify" folder according to the collected information on which word/string appears on each file, the Maximum A Posteriori probabilities will vary depending on what lambda in lambda smoothing is used. The highest probability for corresponding to the 20 categories will be chosen as the classification. The file numbers with their classifications is stored in "f_classify" text files with their corresponding lambda values (e.g. "f_classify_0.01.txt" for the classifications using 0.01 as the lambda value)
- "checker.py" - computes the number and percentage of correct classifications for the 0.01, 0.1, 0.2, 0.5, and 1.0 lambda values.

Experiments:

Version	Lambda = 0.01	Lambda = 0.1	Lambda = 0.2	Lambda = 0.5	Lambda = 1.0
1	55.91%	54.11%	52.62%	49.48%	44.52%
2	55.78%	55.34%	53.63%	48.49%	39.97%

Analysis:

Version 1 may have yielded a smaller maximum accuracy, but the discrepancy of the accuracy between the five varying lambda accuracies is much closer than Version 2.