ITP 20002 Discrete Mathematics, Fall 2020

Homework 4. Detecting Negative Twitter Messages Using Naïve Bayes Text Classifier

Shin Hong

1. Overview

In this homework you are asked to implement a Naïve Bayes text classifier and apply it for detecting negative sentiment text messages. Based on the given design (Section 2), you need to construct a text classifier by (1) building a module that constructs a text classification model (calling this module *trainer*) and (2) a module that determines whether a given text is negative or not according to the given classification model (calling this module *predictor*).

For training and testing your text classifier, you are given the preprocessed dataset of the Twitter US Airline Sentiment corpus[1]. The preprocessed dataset consists of a set of tweet messages manually labeled as *negative* and another set of tweet messages labeled as *positive* or *neutral* (i.e., non-negative) You must apply your text classifier to this given corpus data and evaluate the performance. The corpus dataset and the related code can be found at the `hw4` branch of the class Github repository.

`https://github.com/hongshin/DiscreteMath/tree/hw4`

2. Twitter Airline Sentiment Dataset

The dataset given for this homework is originated from the Twitter US Airline Sentiment dataset of Crowdflower's Data for Everyone library. The original dataset was constructed by scraping twitter messages automatically from February of 2015 and classifying each message manually as either *Negative*, *Neural*, or *Positive*.

The training and testing datasets for this homework were collected from the original corpus as binary training datasets. These datasets are given as the following four files as you can be found at the `data` directory of the `hw4` repository.

- `train.negative.csv`: 9078 text messages that human classifiers found negative
- `train.non-negative.csv`: 5565 text messages that the human found positive or neutral.
- `test.negative.csv`: 100 text messages that human classifiers found negative. This data is separated from `train.negative.csv`.
- `test.non-negative.csv`: 100 text messages that are classified the human classifiers found positive or neutral. This data is separated from `train.non-negative.csv`

3. Naïve Bayes Text Classifier

3.1. Classification

Despite its simple computational structure, Naïve Bayes text classifiers are widely used in real-world because the prediction can be made quickly, and accurately once sufficient data is given for training the models.

Naïve Bayes text classifiers model an occurrence of a word in a text as an independent event, and an occurrence of a text as an event where all containing words occur at the same time. As it assumes all words independent with each other, Naïve Bayes text classifiers estimates the likelihood of a text of being a certain class by measuring the probabilities of its containing words appearing in the texts of the class.

Here is the scheme of Naïve Bayes text classifier that you need to implement in this homework. A text t_i consisting of a set of words $\{w_{i,1}, w_{i,2}, ..., w_{i,n_i}\}$ can be classified as C_1 or C_2 (i.e., binary classification). By Bayes theorem, $P(C_x \mid t_i)$, the likelihood of t_i being a C_x can be computed as follows:

$$P(C_x \mid t_i) = \frac{P(t_i \mid C_x)\,P(C_x)}{P(t_i)}$$

From this formula, we can drop the denominator $P(t_i)$ by assuming that every text is unique and has an equal chance in its appearance. In addition, we can drop $P(C_x)$ as we use the same number of data for two classes C_1 and C_2. Lastly, we can estimate $P(t_i \mid C_x)$ as the product of all probabilities of words given class C_x:

$$P(C_x \mid t_i) \propto \prod_{w_{i,j} \in t_i} P(w_{i,j} \mid C_x)$$

Using this relation, we can determine the class of the given text t_i as C_1 if and only if $\Pi_{w_{i,j} \in t_i} P(w_{i,j} \mid C_1) > \Pi_{w_{i,j} \in t_i} P(w_{i,j} \mid C_2)$, and vice versa.

From given training datasets, we can measure the number of text messages that contains a word w and classified as a class C_x (i.e., $N(w, C_x)$). With this measurement, the following schemes are used to obtain the final estimation of $P(t_i \mid C_x)$:

- Smoothing: Since the training data is limited in its size and scope, a word w in a text t rarely or never appears in a training dataset. To avoid $P(w|C_x)$ being estimated as zero or unrealistically small, it would be better to add a constant α to both $N(w, C_x)$ and $N(C_x)$ for smoothing $P(t|C_x)$. Specifically, $P(w|C_x)$ is estimated as follow:

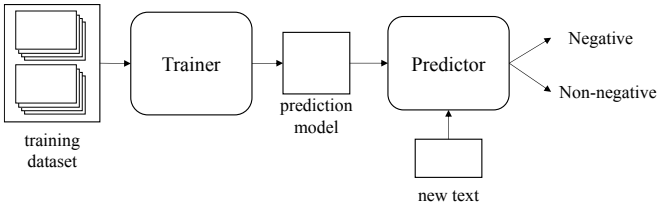
$$P(w|C_x) = \frac{N(w, C_x) + \alpha}{\sum_{w' \in V} (N(w', C_x) + \alpha)}$$

- Log scaling: $\Pi_{w_{i,j} \in t_i} P(w_{i,j} \mid C_x)$ may easily become a very small value if there are many words in a text. Since real number computation errors significantly involve in computing very small values, it would be better to use log probabilities rather than the original one, as follows:

$$P(C_x \mid t_i) \propto \sum_{w_{i,j} \in t_i} \log P(w_{i,j} \mid C_x)$$

Note that $log(ab) = log(a) + log(b)$ and $log(a) < log(b)$ if $a < b$ for real numbers.

[1] https://www.kaggle.com/crowdflower/twitter airline



<Figure 1. Naïve Bayes Text Classifier Framework>

3.2. Framework

Overview. This section gives the required functionalities of each component of the Naïve Bayes text classifier framework. Figure 1 shows the workflow of our Naïve Bayes text classifier which determines a given text as either Negative or Non-negative. The trainer module receives two sets of texts, one for Negative and the other for Non-negative, as training data and generates a prediction model. A prediction model defines conditional probabilities of each word given each class. Given prediction model, the predictor module receives a newly given text as input and produces a classification result as output.

Trainer module. From given training datasets, the trainer module must select a set of words (or vocabulary) which can be used as meaningful features of text messages, and then derive the conditional probabilities of each word. The trainer module must take the following ~~four~~ five steps:

(1) Tokenization

Read each text message and split the text into words which work as components of the given text message. Words can be separated by whitespaces, punctuation marks, or other indicators. Tokenization can employee heuristics or crafted rules for defining meaningful units of words.

(2) Normalization

Transform each word into a normalized form to recognize the variants of a word as a single case. First, all letters of a word must be converted to lower case. And then each word must be passed to a *stemmer*, a tool that derives a representative form of a given word. For instance, a stemmer maps 'connection', 'connections', 'connective', 'connected', 'connecting' to 'connect'. Your Naïve Bayes text classifier must use `libstemmer` which can be found at `libstemmer_c` in the `hw4` branch.

(3) Vocabulary reduction

Exclude the words that infrequently appear in the training dataset, because their conditional probabilities would be measured unreliably. Such words are prone to overfitting problems. Many proper nouns/names would be eliminated at this step.

(4) Prediction model construction

Measure the conditional probabilities of each word and then store the results as a prediction model to a file.

Predictor module. For a given new text, the predictor module determines whether it is likely Negative or not. This determination must be done by comparing estimated probabilities of the two classes given the text (Section 3.1). Note that, to compute the estimated probability The predictor module must apply the same steps (from tokenization to prediction model construction) to a new text as the trainer module does to the training.

4. Tasks

4.1. Implementation

Following the given outlines, you build a trainer module and a predictor module. The followings are detailed requirements on the text classifier implementation parts:

- You must use C programming language.
- The trainer module and the predictor module must be written and built as two separate programs.
- You must put your source code files under the `src` directory of the `hw4` branch. You must clearly declare how to build the program and how to run the program in your report or `README` file.
- You must implement the classification algorithm described in Section 3.1.
- Although the outline of the algorithm is given, still many parts remain open. You are asked to find out an effective way to fill in these missing parts for accurate text classification.
- Your program must use `libstemmer` for word normalization. You can find `libstemmer` at the `libstemmer_c` directory of the `hw4` repository. You can also find an example program using `libstemmer` at `example/stemmer.c`.

4.2. Experiments

Using the given test datasets (i.e., `test.negative.csv`, `test.non-negative.csv`), you are asked to evaluate the performance of your own Naïve Bayes text classifiers. You must report precision (i.e., the ratio of true positives to the sum of true positives and false positives) and recall (i.e., the ratio of true positives to the sum of true positives and false negatives). You may discuss how the performance differs depending on how you configure the classifier.

4.3. Report writing

Write a homework report up to 3 pages in the given template. This report must be written in English.

The report must include all results of your homework, including the design and the implementation of your version of Naïve Bayes text classifier, and the experiment results with the given datasets. The report must describe how you implement a configure the classification algorithm in detail. For example, you should clearly describe the structure of the prediction model

As a part of your report, you must include discussion. For example, you may discuss the limitation of the given Naïve Bayes text classifier design. You may suggest possible ways to improve the performance of Naïve Bayes text classifiers.

5. Submission

Turn in an archive (e.g., as a `tar` or `zip` file) of all your results using Hisnet by **11:00 PM, 5 December (~~Sat~~ Mon)**. Your submission must include a report as a PDF, and all source code files needed for reproducing what you reported in the report. No late submission will be accepted.