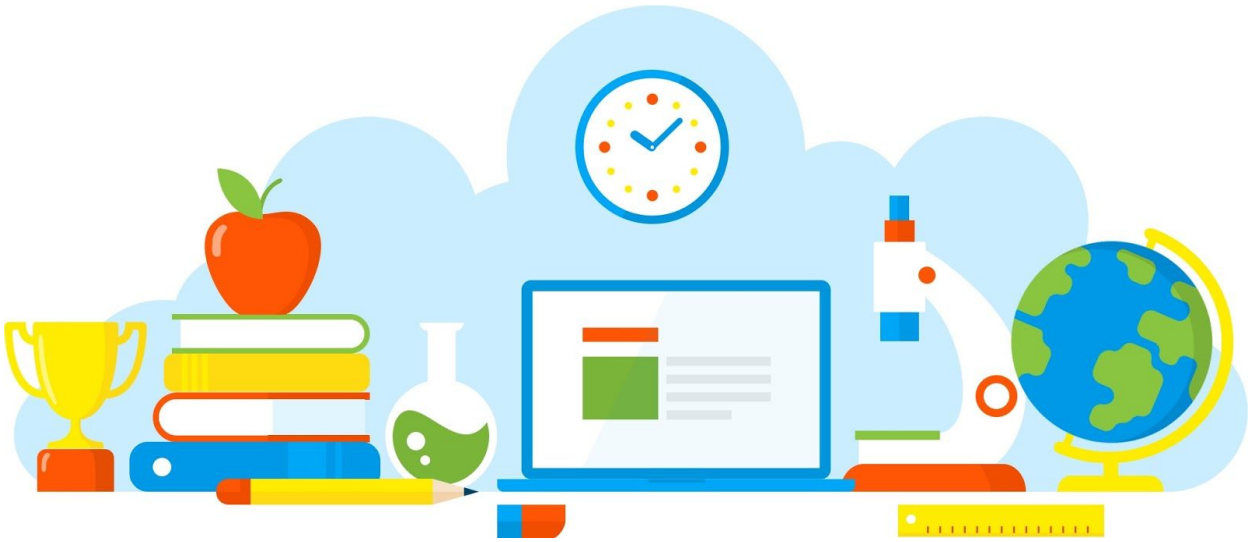


Machine Learning HW2 Report

Teacher: Dr. Zare

Mahdi Sadeghi
830598035
1398/10/23

1. Tokenization, Bag of Words and Counting

Q 1.1.

We first create a dictionary for storing each word and it's count in the input doc as key value pairs. Then we can pass the doc to "tokenize" function of "tokenizer" module and for each token if it's a "WORD" we increment the corresponding key's value in the word-count map.

```
def tokenize_doc(doc):
   mappings = dict()

   for token in tokenizer.tokenize(text=doc):
       kind, txt, val = token

       if kind == tokenizer.TOK.WORD:
           txt = txt.lower()
           if txt in mappings:
               mappings[txt] += 1
           else:
               mappings[txt] = 1

   return mappings
```

Q 1.2.

For each document, update_model is called once so, at first we increment “class_total_doc_counts” for specific label by 1 for each time the update_model is called. We also increment “class_total_word_counts” for incoming label by the size of “bow”. Then we traverse “bow” and add all the words to “vocab” set. As it is a set, it will check and remove duplicates automatically. We also add count of words to

```
def update_model(self, bow, label):
   # increment number of docs for specific label by 1 for each time
this function is called
   self.class_total_doc_counts[label] += 1

   # increment number of words in specific label by the size of input
bag of words
   self.class_total_word_counts[label] += len(bow.keys())

   # add each word to the vocab set and add it's count to
class_word_counts[label]
   for word in bow.keys():
       self.vocab.add(word)
       if word in self.class_word_counts[label]:
           self.class_word_counts[label][word] += bow[word]
       else:
           self.class_word_counts[label][word] = bow[word]
```

The Vocabulary Size of whole Training Set is **83595.**

Q 1.3.

First we sort words in the "class_word_counts" by their values ("How Do I Sort a Dictionary by Value?" n.d.) then we select top n and return them.

```
def top_n(self, label, n):
   # first sort class_word_counts of input label
   sorted_values = sorted(self.class_word_counts[label].items(),
key=lambda kv: kv[1], reverse=True)

   most_frequent_words = []
   for i in range(n):
       most_frequent_words.append(sorted_values[i])

   return most_frequent_words
```

Outputs for top 10 words of each class are as follows:

Positive Top 10	Negative Top 10
• the: 173284 • and: 89733 • a: 83513 • of: 76849 • to: 66736 • is: 57246 • in: 50210 • br: 49228 • it: 39354 • i: 36106	• the: 163367 • a: 79208 • and: 74379 • of: 69000 • to: 68973 • br: 52631 • is: 50086 • in: 43746 • this: 40899 • i: 40612

Q 1.4.

No, because all of them are similar between both document types. These words are called stop words as they are almost always found as most used words in any english document.

2. Word Probabilities And Pseudocounts

Q 2.1.

We can get the probability of a word occurring in a specific class (p(w|y)) by dividing count of that word occurring in that class to the total number of word counts in that class.

```
def p_word_given_label(self, word, label):
   # divide count of input word in that class to the total number of
word counts in that class
   total_class_word_count =
sum(self.class_word_counts[label].values())
   return self.class_word_counts[label][word] /
total_class_word_count
```

Q 2.2.

	Positive	Negative
fantastic	0.0002189896873938954 7	4.940748756749783e-05
boring	0.0001111690767809989 2	0.0005064267475668527

This is no surprise as it tells us that the probability of having “fantastic” in a positive comment is about twice as having “boring” in a positive comment. It also states that the probability of having “fantastic” in negative comment is almost zero while there is more chance to see “boring” in a negative comment.

Q 2.3.

When a word is not in one class, then the probability of p(w | y) for the respective word and class will become zero. In naive bayes we multiply probabilities of each word in the test document with each other to get the probability of that document being of a specific class. When we have one word in the test document with the p(w | y) = 0, the probability of that document being of class y will become 0 no matter what probabilities other words of the document have.

Q 2.4.

We can do this by adding alpha to numerator and adding (alpha * number of different words in that class) to the denominator. This is called Additive or Laplace Smoothing (Contributors to Wikimedia projects 2008).

```
def p_word_given_label_and_psuedocount(self, word, label, alpha):
   total_class_word_count =
sum(self.class_word_counts[label].values())

   # add alpha to numerator
   numerator = self.class_word_counts[label][word] + alpha
   # add alpha * number of words in that class to denominator
   denominator = total_class_word_count + (alpha *
len(self.class_word_counts[label].keys()))

   return numerator / denominator
```

3. Prior and Likelihood

Q 3.1.

$$P(w_{d_1},\ ...,\ w_{d_n} \mid y) = \prod_{i=1}^{n} P(w_{d_i} \mid y) \xrightarrow{\log} \log\left(P(w_{d_1},\ ...,\ w_{d_n} \mid y)\right) =$$

$$\log\left(\prod_{i=1}^{n} P(w_{d_i} \mid y)\right) = \sum_{i=1}^{n} \log\left(P(w_{d_i} \mid y)\right)$$

Q 3.2.

```
def log_likelihood(self, bow, label, alpha):
    log_lh = 0
    for word in bow.keys():
        p = self.p_word_given_label_and_psuedocount(word, label,
alpha)
        log_lh += math.log(p)

    return log_lh
```

Q 3.3.

```
def log_prior(self, label):
    # divide number of documents with this label to total documents
count
    p = self.class_total_doc_counts[label] /
sum(self.class_total_doc_counts.values())
    # return logarithm of probability
    return math.log(p)
```

4. Normalization and The Decision Rule

Q 4.1.

The normalizer is the probability of seeing w_d despite its class, so if “k” is the number of different classes (2 in this homework) we have:

$$P(w_d) = \sum_{i=1}^{k} P(w_d, y_i)$$

Q 4.2.

$$P(y \mid w_d) = \frac{P(y)P(w_d \mid y)}{P(w_d)} \xrightarrow{\log} \log\left(P(y|w_d)\right) = \log\left(\frac{P(y)P(w_d \mid y)}{P(w_d)}\right) = \log\left(P(y)\right) + \log\left(P(w_d \mid y)\right) - \log\left(P(w_d)\right)$$

Q 4.3.

As the P(w_d) is going to be the same for all P($w_d \mid y$) of different labels, and as we are interested in comparisons of these probabilities and not the exact value of them then the P(w_d) won’t play any role in comparisons and in the final classification result.

Q 4.4.

We can obtain P(y) from “log_prior” and P($w_d \mid y$) from “log_likelihood”, the unnormalized posterior would be the product of these values.

```
def unnormalized_log_posterior(self, bow, label, alpha):
    return self.log_prior(label) * self.log_likelihood(bow, label,
alpha)
```

Q 4.5.

In this function we call “unnormalized_log_posterior” for both positive and negative labels, and then we return positive or negative based on which one of them had a higher log_posterior.

```
def classify(self, bow, alpha):
   positive_unnormalized_lp = self.unnormalized_log_posterior(bow,
POS_LABEL, alpha)
   negative_unnormalized_lp = self.unnormalized_log_posterior(bow,
NEG_LABEL, alpha)

   if positive_unnormalized_lp >= negative_unnormalized_lp:
       return POS_LABEL
   else:
       return NEG_LABEL
```

5. Evaluation

Q 5.1.

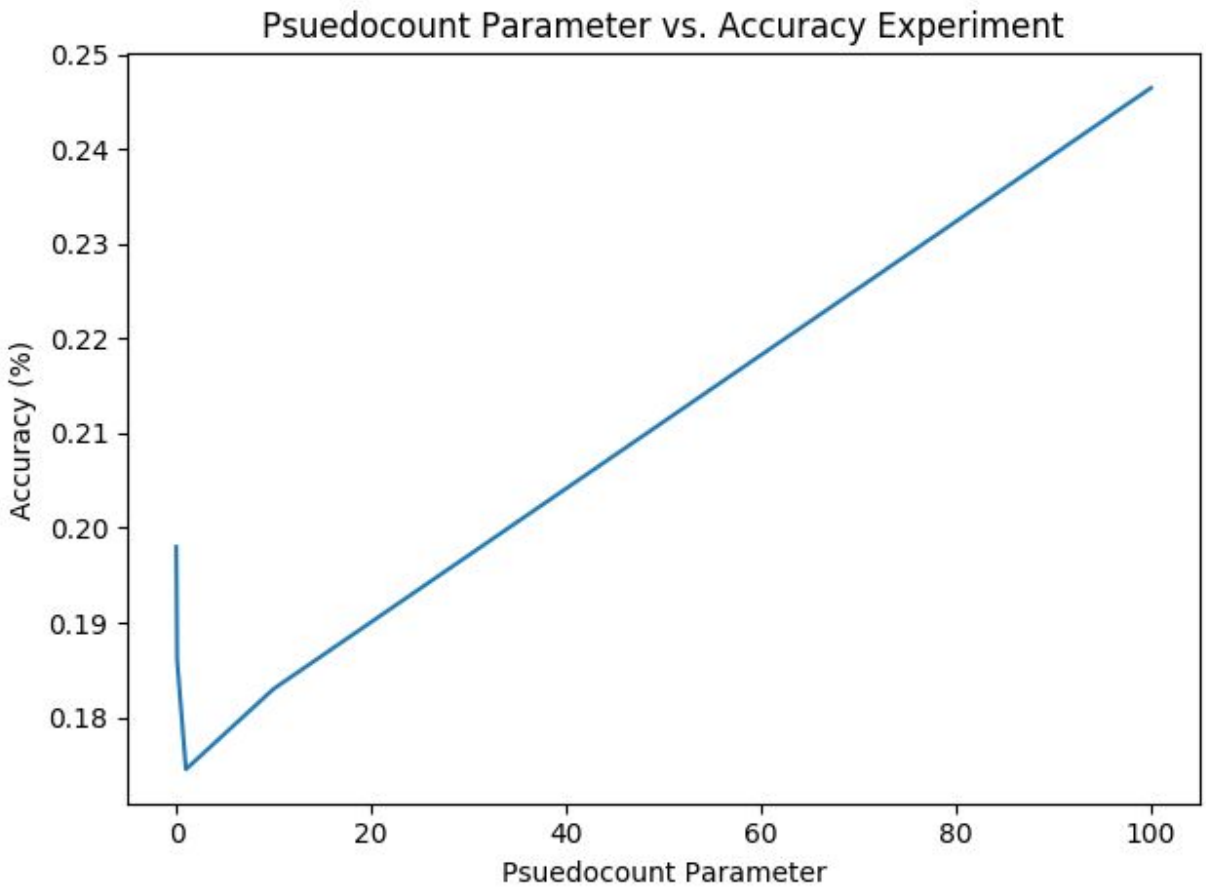
In this function we will tokenize each document, and pass it to “classify” function. Then we compare the return value with the expected value to evaluate correctness of prediction.

```
def evaluate_classifier_accuracy(self, alpha):
    # compute pos and neg paths for test data
    pos_path = os.path.join(TEST_DIR, POS_LABEL)
    neg_path = os.path.join(TEST_DIR, NEG_LABEL)

    correct_predictions = 0
    total_predictions = 0

    for (p, label) in [(pos_path, POS_LABEL), (neg_path, NEG_LABEL)]:
        filenames = os.listdir(p)

        for f in filenames:
            with open(os.path.join(p, f), 'r') as doc:
                content = doc.read()

                # tokenize
                bow = tokenize_doc(content)

                # classify
                prediction = self.classify(bow, alpha)
                total_predictions += 1

                # add to correct predictions if it was right
                if prediction == label:
                    correct_predictions += 1

    return correct_predictions / total_predictions
```

The accuracy for alpha = 1 was 0.1745 (using 2000 test reviews)

Q 5.2.

The plot of accuracies for alphas 0.01, 0.1, 1, 10, 100 with 2000 test reviews is:



Q 5.3.

“let me first just say that in the past, i have been a huge carlin fan. i think george is one of the smartest people and best comedians on the planet. what made george so great in the past was his ability to look at things in his own twisted way, and give us his unique perspective on those things. it wasn't always meant to be funny, but you always respected his opinions, because they were presented in such a clever way. but you are all diseased is just a long rant. he doesn't give us any unique perspective on anything, he just gives us a long list of stuff that he's p.o.'d at. there is no insight, no cleverness, just an old man complaining for one hour straight about things that we have all complained about. and on top of that, it wasn't even funny. you are all diseased appeals to dumb people who can't handle anything more advanced than something simple and direct. i don't mind anger fueled comedy, but george could have done so much better. i really hope that george carlin's next show will live up to the quality that george has shown in the past.”

Unnormalized Log Posterior for Positive Label: **642.0300538632924**

Unnormalized Log Posterior for Negative Label: **636.2733009301932**

As we can see the reviewer has complimented before complaining, so we have a decent amount of positive words in this review while it’s negative in meaning. I think we can turn some words to switch meaning of words before or after them. For example we can make “but” word to inverse effect of the words in the sentence before it, or “is not” to inverse word effects after it.

6. Exploratory Analysis

Q 6.1.

Likelihood can take any value from 0 (if the word is not seen in positive comments at all) to ∞ (if the word is not seen in negative comments). The 0 and ∞ cases do not occur if we use pseudo counts, in that case the minimum will be a very low number (close to 0) and the maximum will be a large number (close to ∞).

Q 6.2.

We can get likelihood from “p_word_given_label_and_psuedocount” function:

```
def likelihood_ratio(self, word, alpha):
   numerator = self.p_word_given_label_and_psuedocount(word,
POS_LABEL, alpha)
   denominator = self.p_word_given_label_and_psuedocount(word,
NEG_LABEL, alpha)

   return numerator / denominator
```

Q 6.3.

word	Likelihood Ratio
fantastic	**4.407150939036379**
boring	**0.2199623572553934**
the	**1.034852862682174**
to	**0.9439867411412745**

We can see that likelihood ratio of the more common words like “the” and “to” are closer to 1 while the least common words tend to get further from 1 either by getting higher or lower.

Q 6.4.

If a word has LR=1 or close to 1, it means that the probability of seeing this word in positive comments is almost the same as seeing it in negative ones, so this word cannot help us to determine the label of an unlabeled comment. If LR for a word is higher than 1 (e.g. 100) it means that there is 100 times more chance to see this word in a positive review rather than a negative one and if LR for a word is lower than 1 (e.g. 0.01) it means there is 100 times more chance to see this word in negative comments rather than positive ones.

Bibliography

Contributors to Wikimedia projects. 2008. “Additive Smoothing - Wikipedia.” Wikimedia Foundation, Inc. April 25, 2008. https://en.wikipedia.org/wiki/Additive_smoothing.

“How Do I Sort a Dictionary by Value?” n.d. Stack Overflow. Accessed October 26, 2019. https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value.