CSCI381/CSCI780 NATURAL LANGUAGE PROCESSING

Homework 2

#1

p(pos) = 0.4
p(S|pos) = 0.09 * 0.07 * 0.29 * 0.04 * 0.08 = 0.0000058464
p(pos) * p(S|pos) = 0.4 * 0.0000058464 = 0.00000233856

p(neg) = 0.6
p(S|neg)=0.16 * 0.06 * 0.06 * 0.15 * 0.11 = 0.000009504
p(neg)*p(S|neg)=0.6 * 0.000009504 = 0.0000057024

After simplifying the two products above, we conclude that:

[p(neg)*p(S|neg) = 0.0000057024] > [p(pos)*p(S|pos) = 0.00000233856]

Thus, the model predicts the class **negative** for the sentence “I always like foreign films”.

#2A

A Naïve-Bayes classifier with bag-of-word features and Add-one smoothing is implemented in the `NB.py` script, with `pre-process.py` which preprocesses the training and test documents (see attached source code).

#2B

The file `movie-review-small-BOW.NB` contains the pre-processed small corpus of movies stored in vector format as follows:

```
{"comedy": {"couple": 1, "fly": 1, "fast": 1, "fun": 2}}
{"comedy": {"fun": 1, "couple": 1, "love": 2}}
{"action": {"fly": 1, "fast": 1, "shoot": 1, "love": 1}}
{"action": {"furious": 1, "shoot": 2, "fun": 1}}
{"action": {"fast": 1, "furious": 1, "shoot": 1}}
```

Feature values were saved as dictionaries instead of sparse vectors in order to save space.

The parameters of the model after training it on the small corpus are included in the file `movie-review-small.NB`, a snippet of which follows:

```
Log prior probability of each class:
{'comedy': -1.3219280948873622, 'action': -0.7369655941662062}

Log likelihood of each word:
P(seduced | comedy) = -16.450180247164752
P(walid | comedy) = -16.450180247164752
...
P(gabba-gabba | action) = -16.450212472837737
P(furious | action) = -14.86524997211658
P(scholes | action) = -16.450212472837737
...
```

#2C

For the test document `{fast, couple, shoot, fly}`

- The log probability for class `comedy` is -63.537686582825216
- The log probability for class `action` is -61.630924889908634

The `action` class has a higher probability and therefore is the more likely class.

#2D

Accuracy

Out of 25,000 test documents, 20,368 were predicted correctly and 4,632 were predicted incorrectly. This gives an accuracy of 81.472%.

Investigation

In the 4,632 *incorrectly* predicted documents, there were 989,015 tokens. The word "`not`" comprised 652 of those tokens (0.066%) and the form "`-n't`" appeared in 547 of the tokens (0.055%).

To contrast, in the 20,368 *correctly* predicted documents there were 4,443,763 tokens, the word "`not`" comprised 765 of those tokens (0.017%), and "`-n't`" was in 965 tokens (0.022%). These tokens occurred substantially less frequently in correctly predicted documents.

The negation expressed by "`not`" or "`-n't`" might completely alter the inferences we draw from the predicate following it in that document. In addition, negation can modify a negative word to produce a positive review, and so give misleading results in a Naïve-Bayes Classification.

When investigating the probabilities for each word, I found the model reported:

```
P(not | pos) = -7.7155410739560235
P(not | neg) = -7.496980855242182
```

which are both very high numbers when compared with most words which received a log probability of less than -20.

Experimentation

Although the standard Naïve Bayes text classification with Bag-of-Words features did work well for the movie review sentiment analysis, I experimented with a small change to improve performance.

I noticed in the reviews that whether a word occurs in the document or not seemed to matter more than the frequency of the word in the document. Therefore, I tried to improve performance by clipping the word counts in each document at 1 (binary multinomial Naïve Bayes). For each document, I did not take into consideration how many times a word appeared in the document, but only whether or not the word appeared in the document (essentially removing duplicate tokens from each document).

This actually did improve my accuracy: the model predicted 20,739 documents correctly, and predicted 4,261 documents incorrectly, giving an accuracy of 82.956% (up from 81.472%).

The parameters and predictions are stored in `movie-review-EXPERIMENT-BOW.NB` and `EXPERIMENT-output.txt`.

pre-process.py source code

```
import os
import sys
import json


def count_frequencies(text):
    freq = {}
    for word in text:
        if word in freq:
            freq[word] += 1
        else:
            freq[word] = 1
    return freq


def ignore_unseen_words(words, vocab):
    return [word for word in words if word in vocab]


def remove_punctuation(text):
    punctuation_to_remove = {'"', '*', '+', '.', '/', '<', '>', '@', '^', '_', '`', '{', '|',
'~', ','}
    new_text = ""
    for char in text:
        if char is '!' or char is "?":
            new_text += " " + char
        elif char not in punctuation_to_remove:
            new_text += char.lower()
    return new_text.split()


def preprocess():
    feature_vectors = []
    for label in os.listdir(directory):                          # for each label
        folder = os.path.join(directory, label)
        if os.path.isdir(folder):
            for filename in os.listdir(folder):                  # for each document
                if filename.endswith(".txt"):
                    f = open(os.path.join(folder, filename), "r")
                    words = remove_punctuation(f.read())
                    words = ignore_unseen_words(words, vocab)
                    feature_vectors.append({label: count_frequencies(words)})
                    f.close()

    output_name = "movie-review-" + directory.replace("/", "").replace("all-reviews", "BOW") +
".NB"
    output = open(output_name, "w")
    for line in feature_vectors:
        output.write(json.dumps(line) + '\n')
    # output.write(parsed)
    output.close()


directory = sys.argv[1]
vocab = set([line.rstrip() for line in open('all-reviews/imdb.vocab')])
preprocess()
```

NB.py source code

```
import sys
import json
import math


def get_inputs():
    training = sys.argv[1]
    test = sys.argv[2]
    model_output_file = sys.argv[3]
    predictions_output_file = sys.argv[4]
    vocab = set([line.rstrip() for line in open('all-reviews/imdb.vocab')])
    documents = []
    classes = {}
    test_docs = {}

    # training file
    training_file = open(training, "r")

    for line in training_file.readlines():
        vector = json.loads(line)
        documents.append(vector)
        label = list(vector.keys())[0]
        if label in classes:
            classes[label].append(vector[label])
        else:
            classes[label] = [vector[label]]
    # The following code replaces the previous loop if using binary NB
    # for line in training_file.readlines():
    #     vector = json.loads(line)
    #     documents.append(vector)
    #     label = list(vector.keys())[0]
    #     clipped = {word: 1 if count > 0 else 0 for word, count in vector[label].items()}
    #     if label in classes:
    #         classes[label].append(clipped)
    #     else:
    #         classes[label] = [clipped]
    training_file.close()

    # test file
    test_file = open(test, "r")

    for line in test_file.readlines():
        vector = json.loads(line)
        label = list(vector.keys())[0]
        if label in test_docs:
            test_docs[label].append(bow_to_list(vector[label]))
        else:
            test_docs[label] = [bow_to_list(vector[label])]
    # # The following code replaces the previous loop if using binary NB
    # for line in test_file.readlines():
    #     vector = json.loads(line)
    #     label = list(vector.keys())[0]
    #     if label in test_docs:
    #         test_docs[label].append(vector[label].keys())
    #     else:
    #         test_docs[label] = [vector[label].keys()]
    test_file.close()

    return documents, classes, vocab, test_docs, model_output_file, predictions_output_file


def train_nb(documents, classes, vocab):
    total_num_of_documents = len(documents)
    log_prior = {}
    bow_for_each_class = {}
    log_likelihood = {}
    num_of_words_in_each_class = {}
    for label, docs_in_the_class in classes.items():

        # calculate P(c) terms
        num_of_documents_in_this_class = len(docs_in_the_class)
        log_prior[label] = math.log2(num_of_documents_in_this_class / total_num_of_documents)
        bow_for_each_class[label] = {}
        num_of_words_in_each_class[label] = 0
        for doc in docs_in_the_class:
            for word, value in doc.items():
                num_of_words_in_each_class[label] += value
                if word in bow_for_each_class[label]:
                    bow_for_each_class[label][word] += value
                else:
                    bow_for_each_class[label][word] = value

        # calculate P(w|c) terms
        for word in vocab:
            count = 0
            if word in bow_for_each_class[label]:
                count = bow_for_each_class[label][word]
            log_likelihood[(word, label)] = math.log2(
                (count + 1) /
                (num_of_words_in_each_class[label] + len(vocab)))
    return log_prior, log_likelihood, bow_for_each_class


def arg_max(d):
    v = list(d.values())
    k = list(d.keys())
    return k[v.index(max(v))]


def test_nb(test_doc, classes, vocab, log_prior, log_likelihood):
    sum_of_log_probs = {}
    for label, docs_in_the_class in classes.items():
        sum_of_log_probs[label] = log_prior[label]
        for word in test_doc:
            if word in vocab:
                sum_of_log_probs[label] += log_likelihood[(word, label)]
        # print("probability of class", label, "is", sum_of_log_probs[label])
    return arg_max(sum_of_log_probs)


def answer_questions():
    documents, classes, vocab, test_docs, model_output, predictions_output = get_inputs()
    log_prior, log_likelihood, bow_in_each_class = train_nb(documents, classes, vocab)
    results = {True: 0, False: 0}
    predictions = "Document # \t Predicted Label \t Actual Label\n"
    num = 1
    for label, documents in test_docs.items():
        for document in documents:
            test_result = test_nb(document, classes, vocab, log_prior, log_likelihood)
            results[test_result == label] += 1
            predictions += "\t" + str(num) + "\t\t | \t\t" + test_result + "\t\t | \t\t" + 
label + "\n"
            num += 1
    model_output_file = open(model_output, "w")
    model = "Log prior probability of each class:\n" + str(log_prior) + \
            '\n\nLog likelihood of each word: \n' + pretty_prob(log_likelihood)
    model_output_file.write(model)
    model_output_file.close()
    predictions_output_file = open(predictions_output, "w")
    accuracy = results[True] / (results[False] + results[True]) * 100
    predictions += "Total: " + str(results) + ". Accuracy: " + str(accuracy) + '%'
    predictions_output_file.write(predictions)
    predictions_output_file.close()


def pretty_prob(dic):
    pretty = ""
    for key, val in dic.items():
        w = str(key[0])
        c = str(key[1])
        pretty += 'P(' + w + ' | ' + c + ') = ' + str(val) + '\n'
    return pretty


def bow_to_list(bow):
    output = []
    for word, freq in bow.items():
        for i in range(freq):
            output.append(word)
    return output


answer_questions()
```

movie-review-BOW.NB (parameters of the model) snippet

```
Log prior probability of each class:
{'pos': -1.0, 'neg': -1.0}

Log likelihood of each word:
P(mache | pos) = -21.480827172679707
P(succeeded | pos) = -15.988974076350031
P(underscripted | pos) = -21.480827172679707
P(fdny | pos) = -19.158899077792345
P(dankness | pos) = -21.480827172679707
P(consciously | pos) = -19.158899077792345
P(paint-by-numbers | pos) = -20.480827172679707
...
P(punch-drunk | neg) = -21.44954011629902
P(caswell | neg) = -21.44954011629902
P(hyper-critical | neg) = -21.44954011629902
```

```
P(assigns | neg) = -21.44954011629902
P(sonar | neg) = -21.44954011629902
P(mailed | neg) = -20.44954011629902
P(first-aid | neg) = -20.44954011629902
```

movie-review-BOWtrain.NB snippet (training file)

...
{"pos": {"lars": 1, "von": 2, "triers": 1, "europa": 2, "is": 5, "an": 2, "extremely": 1, "good": 2, "film": 4, "that": 1, "?": 1, "trier": 1, "has": 1, "a": 3, "very": 1, "stylized": 1, "way": 1, "to": 2, "tell": 1, "story": 1, "at": 1, "least": 1, "he": 1, "did": 2, "have": 1, "with": 2, "me": 1, "the": 9, "whole": 1, "was": 1, "like": 1, "experience": 1, "even": 2, "if": 1, "i": 2, "see": 1, "it": 2, "on": 2, "small": 1, "television": 1, "screen": 1, "all": 2, "tricks": 1, "in": 1, "my": 2, "opinion": 1, "this": 2, "most": 4, "complete": 2, "real": 1, "and": 2, "moving": 1, "piece": 1, "of": 3, "cinema": 1, "then": 1, "films": 1, "top": 1, "list": 1, "also": 1, "think": 1, "perhaps": 1, "scariest": 1, "gothic": 1, "around": 1, "right": 1, "there": 1, "are": 1, "other": 1, "ones": 1, "too": 1, "but": 1, "one": 2, "favorite": 1, "final": 1, "scene": 1, "harrowing": 1, "scenes": 1, "ever": 1}}
{"pos": {"spoiler": 1, "!": 2, "br": 4, "i": 1, "saw": 1, "this": 2, "film": 1, "a": 5, "few": 1, "years": 1, "back": 1, "its": 2, "lovely": 2, "story": 2, "about": 2, "young": 1, "fella": 1, "who": 3, "wants": 1, "to": 2, "drink": 1, "his": 2, "mothers": 1, "milk": 1, "at": 1, "the": 1, "breast": 1, "but": 1, "she": 1, "thinks": 1, "he": 2, "is": 1, "old": 1, "for": 1, "it": 1, "ends": 2, "up": 2, "lusting": 1, "after": 1, "another": 1, "ladies": 1, "breasts": 1, "and": 3, "in": 2, "competition": 1, "with": 1, "brother": 1, "fancies": 1, "her": 1, "throw": 1, "jealous": 1, "husband": 1, "of": 2, "woman": 1, "cannot": 1, "get": 1, "aroused": 1, "you": 1, "have": 1, "cheeky": 1, "yet": 1, "warm": 1, "love": 1, "friendship": 1, "pairs": 1, "jugs": 1, "dont": 1, "be": 1, "put": 1, "off": 1, "by": 1, "sub-tit-les": 1, "hehe": 1}}
{"neg": {"this": 3, "was": 1, "the": 5, "worst": 1, "film": 2, "i": 3, "have": 1, "seen": 1, "for": 2, "a": 3, "long": 1, "time": 3, "br": 7, "not": 3, "only": 1, "that": 3, "it": 1, "has": 1, "nearly": 1, "nothing": 1, "to": 2, "do": 2, "with": 1, "other": 1, "american": 2, "pie": 1, "movies": 1, "story": 1, "is": 2, "obvious": 1, "flat": 1, "and": 1, "absolutely": 1, "funny": 1, "girls": 1, "are": 1, "nice": 1, "though": 1, "but": 2, "spending": 1, "your": 2, "watching": 2, "cheap": 1, "soft": 1, "porno": 1, "would": 2, "possibly": 1, "be": 2, "greater": 1, "than": 2, "seems": 1, "very": 1, "bad": 1, "made": 2, "sex": 1, "ad": 1, "an": 2, "audience": 1, "older": 1, "never": 1, "visited": 1, "college": 1, "seriously": 1, "doubt": 1, "anyone": 1, "who": 1, "did": 1, "could": 1, "really": 1, "laugh": 1, "about": 1, "any": 1, "of": 1, "save": 1, "something": 1, "else": 1}}
{"neg": {"in": 3, "what": 1, "could": 1, "have": 1, "been": 1, "an": 1, "otherwise": 1, "run": 1, "of": 2, "the": 9, "mill": 1, "mediocre": 1, "film": 4, "about": 3, "infidelity": 1, "sixties": 1, "subtle": 1, "free-love": 1, "creators": 1, "this": 2, "pile": 1, "on": 2, "ridiculous": 2, "scenario": 2, "after": 2, "and": 4, "top": 2, "it": 2, "all": 1, "off": 1, "with": 1, "a": 2, "trite": 1, "little": 1, "cherry": 1, "happily": 1, "ever": 2, "ending": 1, "at": 1, "no": 1, "time": 1, "did": 2, "i": 2, "feel": 2, "sympathy": 1, "for": 2, "diane": 2, "lane": 2, "or": 1, "anna": 1, "paquin": 1, "their": 1, "troublesome": 1, "middle-class": 1, "care": 1, "free": 1, "life": 1, "nor": 1, "emasculated": 1, "liev": 1, "shrieber": 1, "story": 1, "line": 1, "plods": 1, "along": 1, "slowly": 1, "to": 2, "its": 1, "predictable": 1, "pathetic": 1, "conclusion": 1, "only": 1, "thing": 1, "interesting": 1, "watchable": 1, "is": 1, "stunning": 1, "topless": 1, "hint": 1, "occurs": 1, "minutes": 1, "into": 1, "fast": 1, "forward": 1, "that": 1, "part": 1, "skip": 1, "rest": 1}}
...

movie-review-BOWtest.NB snippet (test file)

...
{"pos": {"i": 10, "like": 2, "the": 5, "wind": 1, "and": 5, "lion": 1, "very": 1, "much": 2, "it": 14, "was": 6, "a": 4, "good": 4, "movie": 2, "thought": 1, "that": 2, "since": 1, "i'm":

```
1, "young": 1, "made": 1, "so": 2, "long": 1, "ago": 1, "wouldn't": 1, "all": 1, "but": 1,
"after": 1, "saw": 1, "amazed": 1, "of": 3, "how": 2, "my": 5, "family": 1, "liked": 5,
"friends": 1, "everyone": 1, "showed": 2, "to": 2, "because": 1, "arabs": 1, "people": 1,
"in": 1, "treated": 1, "during": 1, "early": 1, "by": 1, "germans": 1, "french": 1, "even": 2,
"americans": 1, "if": 1, "high": 2, "school": 1, "history": 1, "teacher": 1, "would": 1,
"definitely": 1, "show": 1, "from": 1, "point": 1, "view": 1, "give": 1, "this": 1, "out": 1,
"grandparents": 1, "they": 1, "bought": 1, "for": 1, "themselves": 1, "little": 1, "year": 1,
"old": 1, "cousins": 1, "sit": 1, "down": 1, "watched": 1, "br": 1}}
{"pos": {"this": 3, "movie": 4, "has": 1, "an": 1, "outstanding": 1, "acting": 1, "by": 4,
"and": 3, "a": 3, "stunning": 1, "the": 13, "argentine": 1, "hector": 1, "is": 4, "in": 4,
"my": 1, "opinion": 1, "best": 1, "brazilian": 2, "ever": 1, "made": 1, "was": 1, "filmed": 1,
"with": 1, "child": 1, "from": 1, "children": 3, "weren't": 1, "actors": 1, "were": 1,
"casted": 1, "city": 1, "of": 1, "rio": 1, "de": 1, "janeiro": 1, "story": 1, "about": 1,
"criminal": 1, "that": 2, "are": 2, "arrested": 1, "correctional": 1, "prison": 1, "looks": 1,
"much": 1, "worse": 1, "than": 1, "alcatraz": 1, "constantly": 1, "raped": 1, "beaten": 1,
"policeman": 1, "unfortunately": 1, "not": 1, "purely": 1, "fiction": 1, "brazil": 1, "it": 2,
"does": 1, "happen": 1, "till": 1, "today": 1, "must": 1, "see": 1, "for": 1, "those": 1,
"who": 1, "like": 1, "violent": 1, "movies": 1, "but": 1, "take": 1, "your": 1, "mother": 1,
"off": 1, "room": 1, "because": 1, "hard": 1}}
{"neg": {"i": 7, "saw": 1, "this": 9, "with": 2, "my": 5, "kids": 4, "they": 7, "love": 2,
"it": 2, "but": 4, "don't": 1, "she": 4, "did": 3, "not": 1, "get": 2, "run": 1, "overfed": 1,
"by": 3, "a": 4, "reindeer": 1, "in": 6, "the": 9, "song": 3, "what": 2, "heck": 2, "crappy":
3, "movie": 3, "got": 1, "hit": 2, "sleigh": 2, "like": 4, "why": 5, "when": 2, "heard": 1,
"thought": 1, "was": 3, "good": 1, "we": 1, "watched": 1, "were": 1, "daddy": 2, "granny": 1,
"thats": 1, "how": 1, "say": 1, "grandma": 3, "any": 2, "way": 1, "said": 1, "told": 1,
"them": 1, "that": 2, "agreed": 1, "sad": 1, "would": 2, "one": 4, "name": 1, "there": 1,
"dog": 1, "just": 1, "dumb": 1, "every": 3, "dressed": 1, "black": 1, "looked": 3, "so": 2,
"mean": 1, "daphne": 1, "dang": 1, "emo": 1, "goth": 1, "girl": 1, "found": 1, "on": 1,
"ground": 1, "think": 2, "died": 1, "weird": 1, "gone": 1, "should": 1, "take": 1, "show": 1,
"off": 1, "ok": 1, "give": 1, "out": 1, "of": 1}}
{"neg": {"like": 1, "i'm": 1, "sure": 2, "other": 1, "people": 1, "have": 1, "said": 1,
"this": 1, "guy": 1, "isn't": 1, "a": 3, "very": 1, "worthwhile": 1, "subject": 1, "our": 1,
"society": 1, "has": 1, "morbid": 1, "fascination": 1, "with": 2, "death": 1, "and": 3,
"funny": 1, "hearing": 1, "him": 2, "talk": 1, "about": 1, "how": 2, "much": 2, "he": 3,
"smokes": 1, "coffee": 1, "drinks": 1, "but": 2, "into": 1, "giving": 1, "himself": 1, "an":
1, "unworthy": 1, "mystique": 1, "anyway": 1, "the": 3, "bottom": 1, "line": 1, "is": 2,
"that": 1, "moron": 1, "racist": 1, "using": 1, "feeble": 1, "methods": 1, "to": 2, "try": 1,
"disprove": 1, "mountain": 1, "of": 3, "evidence": 1, "holocaust": 1, "as": 1, "such": 1,
"should": 1, "be": 1, "forgotten": 1, "by": 1, "time": 1, "morris": 1, "in": 1, "love": 1,
"any": 1, "kind": 1, "which": 1, "normally": 1, "i": 1, "wouldn't": 1, "fault": 1, "for": 1}}
...
```

output.txt (predictions) snippet

```
        Document #       Predicted Label       Actual Label
        1            |          pos          |          pos
        2            |          neg          |          pos
        3            |          pos          |          pos
        4            |          pos          |          pos
        5            |          neg          |          pos
        6            |          pos          |          pos
        7            |          pos          |          pos
        8            |          neg          |          pos
        9            |          pos          |          pos
...
```

```
     24994       |          pos          |          neg
     24995       |          neg          |          neg
     24996       |          neg          |          neg
     24997       |          neg          |          neg
     24998       |          neg          |          neg
     24999       |          neg          |          neg
     25000       |          neg          |          neg
Total: {True: 20368, False: 4632}. Accuracy: 81.472%
```