Saad Faruque **NLP**

(Question)For the reviews for which your program made incorrect predictions, were there any trends that you observed? That is, can you explain why these incorrect predictions were made?

(Answer)In general, for the reports that my model gave incorrect predictions for, I noticed that the reviews in the given report had both many positive **and** negative words. For example, on test data number three, the user review even though is negative would have many words that would come also in the positive class(such as “wow, leading, etc..”). Though the user meant for the review to be negative(and the review is structured as such), the words being used are commonly used in the positive class. This makes the classification hard as just based off the words the review would have a higher probability of positive(though it is negative).

Pre-Process

```
import sys
import os
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 remove_unseen_words(words, vocab):
    return [word for word in words if word in vocab]


def add_to_feature_vectors(feature_vectors, _class, processed_text):
    processed_text = remove_unseen_words(processed_text, vocabulary)
    features = count_frequencies(processed_text)
    feature_vectors.append({_class: features})


vocabulary = set([line.rstrip() for line in open('Reviews\imdb.vocab')])

punctuation = {'"', '*', '+', ',', '.', '/', '<', '>', '@', '^', '_', '`', '{', '|', '~'}

reviews_folder = sys.argv[1]

classes = os.listdir(reviews_folder)

feature_vectors = []

for _class in classes:
    current_class = os.path.join(reviews_folder, _class)
    if os.path.isdir(current_class):
        for file in os.listdir(current_class):
            if file.endswith(".txt"):
                review_file = open(os.path.join(current_class, file), "r")
                review_text = review_file.read()
                processed_text = ""
                for c in review_text:
                    if c in '!?':
                        processed_text += " " + c.lower()
                    elif c not in punctuation:
                        processed_text += c.lower()
                processed_text = processed_text.split()
                add_to_feature_vectors(feature_vectors, _class, processed_text)


print(feature_vectors)

output_file_name = 'test'
#output_file_name = output_file_name[:output_file_name.index("/")]


output_file = open("movie-review-" + output_file_name + ".NB", 'w')
for vector in feature_vectors:
    output_file.writelines(json.dumps(vector) + "\n")
```

NB.py

```
import os
import sys
import json
import math


def turn_to_list(count):
    doc = []
    for key, value in count.items():
        for i in range(value):
            doc.append(key)
    return doc

def create_documents(train_documents, test_documents, classes):
    training_file = r'C:\Users\Saad\Desktop\NLP\movie-review-testdata.NB'
    processed_feature_vectors = open(training_file, 'r')
    for line in processed_feature_vectors.readlines():
        vector = json.loads(line)
        train_documents.append(vector)
        key = list(vector.keys())[0]
        if key in classes:
            classes[key].append(vector[key])
        else:
            classes[key] = [vector[key]]

    processed_feature_vectors.close()

    test_file = r'C:\Users\Saad\Desktop\NLP\movie-review-test.NB'
    processed_feature_vectors = open(test_file, 'r')
    for line in processed_feature_vectors.readlines():
        vector = json.loads(line)
        key = list(vector.keys())[0]
        if key in test_documents:
            test_documents[key].append(turn_to_list(vector[key]))
        else:
            test_documents[key] = [turn_to_list(vector[key])]
    processed_feature_vectors.close()



def train_naive_bayes(number_of_documents, classes, vocabulary):
    prior_probabilities = {}
    num_of_words_in_each_class = {}
    bow = {}
    each_word_probability = {}
    for label, document in classes.items():
        num_of_documents_in_class = len(document)
        prior_probabilities[label] = math.log2(num_of_documents_in_class/number_of_documents)
        bow[label] = {}
        num_of_words_in_each_class[label] = 0
        for doc in document:
            for word, value in doc.items():
                if word in {"the", "in", "a", "it", "are", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he", "is", "it", "of", "on", "that", "to",
"were", "was", "will", "with"}:
                    continue
                num_of_words_in_each_class[label] += value
                if word in bow[label]:
                    bow[label][word] += value
                else:
                     bow[label][word] = value
        for word in vocabulary:
            count = 0
            if word in bow[label]:
                count = bow[label][word]
            each_word_probability[(word, label)] = math.log2((count+1) / (num_of_words_in_each_class[label] + len(vocabulary)))
    return prior_probabilities, each_word_probability, bow


def test_naive_bayes(test_documents, classes, vocabulary, prior_probabilities, each_word_probabilities):
    sum_prior_probabilities = {}
    for label, documents_in_each_class in classes.items():
        sum_prior_probabilities[label] = prior_probabilities[label]
        for word in test_documents:
            if word in {"the", "in", "a", "it", "are", "an", "and", "as", "at", "be", "by", "for", "from", "has", "he",
                        "is", "it", "of", "on", "that", "to", "were", "was", "will", "with"}:
                continue
            if word in vocabulary:
                sum_prior_probabilities[label] += each_word_probabilities[(word, label)]
    return arg_max(sum_prior_probabilities)


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


def pretty_print(each_word_probabilities):
    pretty = ""
    for key, val in each_word_probabilities.items():
        w = str(key[0])
        c = str(key[1])
        pretty += 'Probability of "' + w + '" being classified as ' + c + ' is ' + str(val) + '\n'
    return pretty



parameters_file = 'parameters'
predictions_file = 'predictions'

vocabulary = set([line.rstrip() for line in open('Reviews\imdb.vocab')])

train_documents = []
test_documents = {}
classes = {}

create_documents(train_documents, test_documents, classes)
print("Finished Creating Documents\n")
number_of_documents = len(train_documents)
prior_probabilities, each_word_probabilities, bow = train_naive_bayes(number_of_documents, classes, vocabulary)
print("Finished Computing Prior Probabilities\n")

results = {True: 0, False: 0}
```

```
predictions = "Predictions for Test Reviews: \n\n"
num = 1
for label, documents in test_documents.items():
    for document in documents:
        test_result = test_naive_bayes(document, classes, vocabulary, prior_probabilities, each_word_probabilities)
        results[test_result == label] +=1
        predictions += "\t" + str(num) + "\t\t | \t\t Predicted Class: " + test_result + "\t\t | \t\t Actual Class: " + label + "\n"
        num += 1
        print("Finished First Document of " + label + "\n")
    print("finished " + label + "\n")

parameters_file = open(parameters_file, 'w')
parameters_file.write(pretty_print(each_word_probabilities))
parameters_file.close()
predictions_file = open(predictions_file, 'w')
accuracy = (results[True]/(results[False] + results[True])) *100
predictions += "\n\n Total Reviews: " + str(results[False] + results[True])
predictions += "\n Amount Predicted Correctly: " + str(results[True])
predictions += "\n Amount Predicted Incorrectly: " + str(results[False])
predictions += "\n Accuracy: " + str(accuracy) + "%"
predictions_file.write(predictions)
predictions_file.write("\n\n Prior Probabilities: \n\t Negative: " + str(prior_probabilities["neg"]) + "\n\t Positive: " +str(prior_probabilities["pos"]))
predictions_file.close()
print("Done")
```

Instructions:

Using the pre-process.py, on line **26** , first specify the relative path to your vocabulary list. Also on line **55** be sure to specify if you are going to be pre-processing the training data or the test(this only matters for the output file name).
Now, when running pre-process.py in your terminal(or whatever you prefer), run it with your first argument being the file you want to pre-process. After this is done(Taking around 5 minutes), you will now have a .NB file.
Next, using the NB.py, on line **15**, be sure to specify the relative path to your training .NB file. Also on line **28** specify the path to your test .NB file. This process should take a few hours depending on your cpu and if your using from a compiler or your terminal itself(terminal is faster). At the end you should get a BOW file and a predictions file. The predictions file will have what the NB has predicted your report to be(based on the negative and positive possible classes).