```python
import string
import os
import json


# --------------------------------------------------- load file ------------------------------------------------------ #

def load_vocab(vocab_path):
    vocabs = set()
    with open(vocab_path) as vocab_file:
        for word in vocab_file:
            vocabs.add(word.strip())
    return vocabs


def save_file(file_content, file_path):
    with open(file_path, 'w') as file:
        file.write(file_content)


# ------------------------------------------------ preprocess files -------------------------------------------------- #

def is_punc(char):
    return char in string.punctuation


def lowercase_sentence(comment: str) -> str:
    return comment.lower()


def separate_punctuation(comment: str) -> str:
    res = []
    for word in comment.split():
        start = 0 # start index of current separation
        for i, c in enumerate(word):
            if is_punc(c):
                if start < i:
                    res.append(word[start: i])
                res.append(c)
                start = i + 1
        if start < len(word):
            res.append(word[start:])
    return ' '.join(res)


def contains_strong_pos_word(words:list) -> bool:
    positive_words = {'excellent', 'amazing', 'great', 'fantastic', 'outstanding', 'terrific', 'phenomenal', 'superb',
                      'brilliant', 'impressive'}
    for word in words:
        if word in positive_words:
            return True
    return False


def contains_strong_neg_word(words: list) -> bool:
    negative_words = {'disappointing', 'terrible', 'awful', 'horrible', 'dreadful', 'abysmal', 'appalling', 'atrocious',
                      'repulsive', 'disgusting'}
    for word in words:
        if word in negative_words:
            return True
    return False


def preprocess_comment(comment: str) -> str:
    return separate_punctuation(lowercase_sentence(comment))


def preprocess_file(file_path: str) -> str:
    with open(file_path) as file:
        comment = ' '.join([line.strip() for line in file])
        comment = lowercase_sentence(comment)
        comment = separate_punctuation(comment)
        return comment


def build_bag_of_word_vector(comment: str, vocabs: set):
    vector = {}
    for word in comment.split():
        if word in vocabs:
            if word in vector:
                vector[word] += 1
            else:
                vector[word] = 1
    return vector


def preprocess_folder(folder_path: str, vocabs):
    vector_list = []
    for filename in os.listdir(folder_path):
        comment = preprocess_file(f'{folder_path}/{filename}')
        vector_list.append(build_bag_of_word_vector(comment, vocabs))
    return vector_list


def preprocess(folder_path1, folder_path2, vocab_path, path1_class, path2_class, output_path):
    # label######{json format of vector}
    # ###### is the separator of column to access easier
    vocabs = load_vocab(vocab_path)
    folder_path1_vectors = preprocess_folder(folder_path1, vocabs)
    folder_path2_vectors = preprocess_folder(folder_path2, vocabs)

    res = []
    for vector in folder_path1_vectors:
        res.append(f'{path1_class}#####{json.dumps(vector)}#####{int(contains_strong_pos_word(list(vector.keys())))}#####{int(contains_strong_neg_word(list(vector.keys())))}')
    for vector in folder_path2_vectors:
        res.append(f'{path2_class}#####{json.dumps(vector)}#####{int(contains_strong_pos_word(list(vector.keys())))}#####{int(contains_strong_neg_word(list(vector.keys())))}')
    save_file('\n'.join(res), output_path)
```

```python
import os
import json
import decimal

from pre_process import preprocess_comment, preprocess, preprocess_file


# --------------------------------------------------- load file ------------------------------------------------------ #

def load_vocab(vocab_path) -> set:
    vocabs = set()
    with open(vocab_path) as vocab_file:
        for word in vocab_file:
            vocabs.add(word.strip())
    return vocabs


def save_model(file_content, file_path):
    with open(file_path, 'w') as file:
        json.dump(file_content, file)


def save_file(file_content, file_path):
    with open(file_path, 'w') as file:
        file.write(file_content)


def load_json(file_path: str) -> dict:
    with open(file_path) as file:
        return json.load(file)


def load_one_vector(vector_file_path: str) -> dict:
    return load_json(vector_file_path)


def load_naive_bayes_classifier(classifier_path: str) -> dict:
    return load_json(classifier_path)


# ------------------------------------------ train naive bayes classifier -------------------------------------------- #

def initialize_counter(vocabs) -> dict:
    '''
    load vocab and build  a dictionary that contains all vocab as key, and value set to 0
    :return: a dictionary that contains all vocab as key, and value set to 0
    '''

    return {vocab: 0 for vocab in vocabs}


def preprocessed_file_decoder(training_file_path: str):
    with open(training_file_path) as file:
        line = file.readline()
        while line:
            line = line.rstrip('\n')
            line = line.split('#####')
            yield line[0], json.loads(line[1]), bool(line[2]), bool(line[3])
            line = file.readline()


def aggregate_vector_into_counter(counter: dict, vector: dict, class_type, contains_strong_pos_word, contains_strong_neg_word):
    total_token = 0
    for word, freq in vector.items():
        counter[word] += freq
        total_token += freq

    if class_type == 'pos' and contains_strong_pos_word or class_type == 'neg' and contains_strong_neg_word:
        for word, freq in vector.items():
            counter[word] += 100
            total_token += 100
    else:
        for word, freq in vector.items():
            if counter[word] > 100:
                counter[word] -= 100
                total_token -= 100
            else:
                total_token -= counter[word]
                counter[word] = 0
    return total_token


def train_naive_bayes_class_recognizer(counter: dict, total_token: int) -> dict:
    recognizer = {}
    total_vocab = len(counter)
    add_one_smoothing_total_token = total_vocab + total_token
    for word, freq in counter.items():
        recognizer[word] = (freq + 1) / add_one_smoothing_total_token
    return recognizer


def naive_bayes(training_file_path, result_model_path, vocab_path="", class_1="pos", class_2="neg"):
    vocabs = load_vocab(vocab_path)
    counter = {
        class_1: {
            'counter': initialize_counter(vocabs),
            'total_token': 0,
            'class_recognizer': None,
            'prior_prob': 0
        },
        class_2: {
            'counter': initialize_counter(vocabs),
            'total_token': 0
        },
    }

    for class_type, vector, contains_strong_pos_word, contains_strong_neg_word in preprocessed_file_decoder(training_file_path):
        counter[class_type]['total_token'] += aggregate_vector_into_counter(counter[class_type]['counter'],

        vector,
        class_type,
        contains_strong_pos_word,
        contains_strong_neg_word
        )

    total_token = counter[class_1]['total_token'] + counter[class_2]['total_token']
    naive_bayes_classifier = {class_1: train_naive_bayes_class_recognizer(counter[class_1]['counter'],
                                                                          counter[class_1]['total_token']
                                                                          ),
                              class_2: train_naive_bayes_class_recognizer(counter[class_2]['counter'],
                                                                          counter[class_2]['total_token']
                                                                          ),
                              f'{class_1}_prior': counter[class_1]['total_token'] / total_token,
                              f'{class_2}_prior': counter[class_2]['total_token'] / total_token,
                              "class_1": class_1,
                              "class_2": class_2
                              }

    save_model(naive_bayes_classifier, result_model_path)
    return naive_bayes_classifier


# ----------------------------------------------- evaluate test data ----------------------------------------------- #

def compute_prob(comment: str | list, class_recognizer: dict, prior_prob: float) -> decimal.Decimal:
    if type(comment) is str:
        comment = comment.split()

    # the min of a float is around 1e-310, it's very likely to have the prob of the sentence less than this
    prob = decimal.Decimal(prior_prob)
    for word in comment:
        if word not in class_recognizer:
            continue
        prob = prob * decimal.Decimal(class_recognizer[word])
    return prob


class NaiveBayesClassifier:
    def __init__(self, path_to_model="", model=None):
        self.model = model
        self.class_1 = None
        self.class_2 = None
        if not model and path_to_model:
            self.load_model(path_to_model)
```

```python

    def load_model(self, path_to_model: str):
        with open(path_to_model) as model_file:
            self.model = json.load(model_file)
            self.class_1 = self.model["class_1"]
            self.class_2 = self.model["class_2"]

    def classify(self, comment: str):
        comment = preprocess_comment(comment)
        word_list = comment.split()
        class_1_prob = compute_prob(word_list, self.model[self.class_1], self.model[f"{self.class_1}_prior"])
        class_2_prob = compute_prob(word_list, self.model[self.class_2], self.model[f"{self.class_2}_prior"])

        # print(self.class_1, "probability is", class_1_prob)
        # print(self.class_2, "probability is", class_2_prob)

        return self.class_1 if class_1_prob > class_2_prob else self.class_2


# ------------------------------------------ main (training and evaluate) ------------------------------------------- #

# Train a classifier use a small corpus
def problem_2b():
    preprocess(folder_path1="./data/movie_review_small/action",
               folder_path2="./data/movie_review_small/comedy",
               vocab_path="./data/movie_review_small/movie_review_small.vocab",
               path1_class="action",
               path2_class="comedy",
               output_path="./preprocessed/movie_review_small.txt"
               )

    naive_bayes(training_file_path="./preprocessed/movie_review_small.txt",
                result_model_path="models/movie_review_small.NB",
                class_1="action",
                class_2="comedy",
                vocab_path="./data/movie_review_small/movie_review_small.vocab",
                )


def problem_2c():
    comment = "fast, couple, shoot, fly"
    naive_bayes_classifier = NaiveBayesClassifier(path_to_model='models/movie_review_small.NB')
    class_estimation = naive_bayes_classifier.classify(comment)

    print(f"Class of sentence {comment} is: {class_estimation}")


def problem_2d():
    # preprocess training data and train model
    preprocess(folder_path1="./data/train/pos",
               folder_path2="./data/train/neg",
               vocab_path="./data/imdb.vocab",
               path1_class="pos",
               path2_class="neg",
               output_path="./preprocessed/movie_review_BOW.txt"
               )

    naive_bayes(training_file_path="./preprocessed/movie_review_BOW.txt",
                result_model_path="./models/movie_review_BOW.NB",
                class_1="pos",
                class_2="neg",
                vocab_path="./data/imdb.vocab",
                )

    naive_bayes_classifier = NaiveBayesClassifier(path_to_model='./models/movie_review_BOW.NB')
    pos_test_folder = './data/test/pos'
    neg_test_folder = './data/test/neg'
    pos_test_files = os.listdir(pos_test_folder)
    neg_test_files = os.listdir(neg_test_folder)

    result = []  # [[estimation, comment],...]
    incorrect = []
    total_est = len(neg_test_files) + len(pos_test_files)

    for file in pos_test_files:
        comment = preprocess_file(file_path=f'{pos_test_folder}/{file}')
        class_est = naive_bayes_classifier.classify(comment)
        result.append(f'{class_est}    {comment}')
        if class_est != 'pos':
            incorrect.append(f'{class_est}    {comment}')

    for file in neg_test_files:
        comment = preprocess_file(file_path=f'{neg_test_folder}/{file}')
        class_est = naive_bayes_classifier.classify(comment)
        result.append(f'{class_est}    {comment}')
        if class_est != 'neg':
            incorrect.append(f'{class_est}    {comment}')

    accuracy = (total_est - len(incorrect)) / total_est
    result.append(f'Accuracy: {accuracy}    Total Estimations: {total_est}    Incorrect Estimations: {len(incorrect)}')
    save_file('\n'.join(result), './report.txt')
    save_file('\n'.join(incorrect), './incorrect.txt')

problem_2d()
```