Homework 2 Notebook 2

```
import nltk
from nltk import word_tokenize
from nltk.util import ngrams
nltk.download('punkt')

import pickle
import functools
import pprint

def calculate_accuracy(my_predicted_languages, sol_predicted_languages):
    num_predicted_langs = len(sol_predicted_languages)
    correctly_predicted_lines = [
        int(my_predicted_languages[itr] == sol_predicted_languages[itr])
        for itr in range(num_predicted_langs)
    ]

    accuracy = 100 * float(sum(correctly_predicted_lines)) / float(num_predicted_langs)
    incorrectly_predicted_line_nums = list(map(
        lambda x: x[0],
        list(filter(
            lambda x: x[1] == 0,
            enumerate(correctly_predicted_lines)
        ))
    ))

    return ( accuracy, incorrectly_predicted_line_nums )

def write_file(filename, desired_file_contents):
    with open(filename, "w+") as input_file:
        input_file.write(desired_file_contents)

def read_file(filename):
    with open(filename, "r+") as input_file:
        contents = input_file.read()
    return contents

def read_pickle_file(filename):
    with open(filename, "rb") as pickle_file:
        ngram_dict = pickle.load(pickle_file)
    return ngram_dict

def get_sentence_probabilities(sentences, ngram_data):
    sentence_probabilities = list()
    vocabulary_size = 0

    train_word_count = dict()
    print("\n")
    for language in ['english', 'french', 'italian']:
        train_word_count[language] = functools.reduce(
            lambda a,b: a + b,
            list(
                map(
                    lambda x: x[1],
                    ngram_data[language]['unigram'].items()
                )
            )
        )
        print(language, "word count :", train_word_count[language])
        vocabulary_size += len(ngram_data[language]['unigram'].keys())

    print("Vocabulary size:", vocabulary_size)

    for sentence in sentences:
        curr_sentence_probabilities = dict()
        for language in ['english', 'french', 'italian']:
            unigrams = word_tokenize(sentence)
            total_prob = 1.0
            for idx in range(len(unigrams)):
                if idx == 0:
                    continue

                curr_bigram_count = ngram_data[language]['bigram'].get(
                    (unigrams[idx - 1], unigrams[idx]),
                    0
                )

                total_prob *= (
                    ( float(curr_bigram_count) + 1.0 ) / (
                        float(ngram_data[language]['unigram'].get(unigrams[idx - 1], 0 )) +
                        float(vocabulary_size)
                    )
                )

            curr_sentence_probabilities[language] = total_prob
        sentence_probabilities.append(curr_sentence_probabilities)

    return sentence_probabilities

def get_ngram_data():
    en_unigram_count = read_pickle_file("english_train_unigram.pkl")
    en_bigram_count = read_pickle_file("english_train_bigram.pkl")
    fr_unigram_count = read_pickle_file("french_train_unigram.pkl")
    fr_bigram_count = read_pickle_file("french_train_bigram.pkl")
    it_unigram_count = read_pickle_file("italian_train_unigram.pkl")
    it_bigram_count = read_pickle_file("italian_train_bigram.pkl")

    ngram_data = {
        'english': {
            'unigram' : en_unigram_count,
            'bigram' : en_bigram_count,
        },
        'french': {
            'unigram' : fr_unigram_count,
            'bigram' : fr_bigram_count,
        },
        'italian': {
            'unigram' : it_unigram_count,
            'bigram': it_bigram_count,
        },
    }

    return ngram_data

def main():
    ngram_data = get_ngram_data()

    sol_txt_contents = read_file("LangId.sol.txt")
    out_txt_contents = read_file("wordLangId.out.txt")
    test_txt_contents = read_file("LangId.test.txt")

    sol_predicted_languages = list(map(lambda x: x.split()[1], sol_txt_contents.split("\n")[:300]))

    test_sentences = test_txt_contents.split("\n")[:300]
    sentence_probabilities = get_sentence_probabilities(test_sentences, ngram_data)
    predicted_sentence_languages = [
        max(prob_dict.items(), key=lambda item: item[1])[0].capitalize()
        for prob_dict in sentence_probabilities
    ]
    my_predicted_languages = "\n".join([
        str(itr + 1) + " " + str(predicted_lang)
        for itr, predicted_lang in enumerate(predicted_sentence_languages)
    ])
    output_filename = "LangId.myOut.txt"
    write_file(output_filename, my_predicted_languages)

    accuracy, incorrectly_classified = calculate_accuracy(predicted_sentence_languages, sol_predicted_languages)

    print("\n")
    print("Accuracy:", "{acc:.1f}%".format(acc=accuracy))
    print("Incorrectly Classified Line Numbers: ")
    pprint.pprint(incorrectly_classified)

main()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!


english word count : 83309
french word count : 95535
italian word count : 83803
Vocabulary size: 24844


Accuracy: 97.3%
Incorrectly Classified Line Numbers: 
[23, 43, 91, 186, 190, 246, 276, 278]
```

Start coding or generate with AI.