Used Google Colab, need this extra step to mount the drive. Other Jupyter notebooks can skip this step

In [115…
```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [116…
```
import pickle
import sys
import nltk
from nltk import word_tokenize
from nltk.util import ngrams
```

In [117…
```
nltk.download('punkt')
```

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

Out[117]:
```
True
```

Path of where the pickle data (from part 1) is stored on Google Drive

In [118…
```
pickle_path = '/content/drive/MyDrive/NLP_Data'
```

Read in the pickled dictionaries

In [119…
```
with open(pickle_path + 'english_unigram.pickle', 'rb') as handle:
  english_unigram = pickle.load(handle)

with open(pickle_path + 'english_bigram.pickle', 'rb') as handle:
  english_bigram = pickle.load(handle)

with open(pickle_path + 'french_unigram.pickle', 'rb') as handle:
  french_unigram = pickle.load(handle)

with open(pickle_path + 'french_bigram.pickle', 'rb') as handle:
  french_bigram = pickle.load(handle)

with open(pickle_path + 'italian_unigram.pickle', 'rb') as handle:
  italian_unigram = pickle.load(handle)

with open(pickle_path + 'italian_bigram.pickle', 'rb') as handle:
  italian_bigram = pickle.load(handle)
```

Calculate the probability of the simplified Laplace smoothing.

In [153…
```
# Adapted code from https://github.com/kjmazidi/NLP/blob/master/Part_2-Words/Chapter_08_ngrams/8_ngrams_1.ipynb
# Only care about the p_laplace part of this link

import math
def compute_prob(text, unigram_dict, bigram_dict, V):
  # V is the vocabulary size in the training data (unique tokens)

  unigrams_text = word_tokenize(text)
  bigrams_text = list(ngrams(unigrams_text, 2))

  p_laplace = 1
  # UNCOMMENT for p_log method
  p_log = 0

  # (b + 1) / (u + v)
  # where b is the bigram count, u is the unigram count of the first word in the bigram, and v is the total
  # vocabulary size (add the lengths of the 3 unigram dictionaries)
  for bigram in bigrams_text:
    b = bigram_dict[bigram] if bigram in bigram_dict else 0
    u = unigram_dict[bigram[0]] if bigram[0] in unigram_dict else 0
    p_laplace = p_laplace * ((b + 1) / (u + V))
    # UNCOMMENT for p_log method
    # p_log = p_log + math.log((b + 1) / (u + V))

  # UNCOMMENT for p_log method
  #return p_log
  return p_laplace
```

In [147…
```
# Given the Probability of each language, return the highest
def find_largest_prob(english_prob, french_prob, italian_prob):
    predicted_language = ""
    if english_prob == max(english_prob, french_prob, italian_prob):
      predicted_language = "English"
    elif french_prob == max(english_prob, french_prob, italian_prob):
      predicted_language = "French"
    elif italian_prob == max(english_prob, french_prob, italian_prob):
      predicted_language = "Italian"

    return predicted_language
```

In [148…
```
# Testing the previous two functions
# raw_text = "A mon avis , cette deuxième hypothèse signifierait le rejet de nos responsabilités en tant que Parlement , outre l ' introduction d ' une thèse originale , d ' une méthode inconnue qui consiste à communiquer aux grou
raw_text = "Madam President , on a point of order ."
V = len(english_unigram) + len(french_unigram) + len(italian_unigram)
print(V)

english_prob = compute_prob(raw_text, english_unigram, english_bigram, V)
french_prob  = compute_prob(raw_text, french_unigram, french_bigram, V)
italian_prob = compute_prob(raw_text, italian_unigram, italian_bigram, V)
print("English Probability:", english_prob)
print("French Probability:", french_prob)
print("Italian Probability:", italian_prob)

predicted_language = find_largest_prob(english_prob, french_prob, italian_prob)
print("Predicted Language:", predicted_language)
```

```
24842
English Probability: 1.3621600025735524e-24
French Probability: 1.6773305088455522e-33
Italian Probability: 5.713973624389234e-36
Predicted Language: English
```

Process each line from 'LangId.test.txt'

In [149…
```
# Where the data is stored on Google Drive
folder_path = '/content/drive/MyDrive/NLP_Data/data/'
```

In [150…
```
file_name = 'LangId.test.txt'
file_path = folder_path + file_name
lines = []
# Attempt to open the file
try:
  with open(file_path, 'r') as file:
    for line in file:
      # Could process the Probabilities here instead of appending. But for readability reasons processed it later

      # Append each line to lines
      lines.append(line.strip())  # Use strip() to remove leading/trailing whitespaces

# Catch if file could not be found
except FileNotFoundError:
  print("The file, ", file_name, ", cannot be found")
  sys.exit(1)

# Catch all other errors
except Exception as e:
  print(e)
  sys.exit(1)
```

Calculating the probability for each line of 'LangId.test.txt' and write the language with the highest probability to a text file ("predicted_language.txt")

In [151…
```
write_file_path = folder_path + "predicted_language.txt"

# Erase all contents of "predicted_language.txt"
lang_file = open(write_file_path, 'w')
lang_file.write("")
lang_file.close()

# Processing each line in "predicted_language.txt"
for line in lines:
  # Calculating each probability
  english_prob = compute_prob(line, english_unigram, english_bigram, V)
  french_prob  = compute_prob(line, french_unigram, french_bigram, V)
  italian_prob = compute_prob(line, italian_unigram, italian_bigram, V)

  # Predicting the language of the line
  predicted_language = find_largest_prob(english_prob, french_prob, italian_prob)
  # print("Predicted Language:", predicted_language)

  # Write the predicted language to "predicted_language.txt"
  lang_file = open(write_file_path, 'a')
  lang_file.write(predicted_language + '\n')
  lang_file.close()
```

Compute and output your accuracy as the percentage of correctly classified instances in the test set. The file LangId.sol holds the correct classifications.

In [152…
```
test_path = folder_path + "LangId.sol.txt"
predicted_right = 0
predicted_wrong = 0

with open(write_file_path, 'r') as predicted_language, open(test_path, 'r') as test_file:
  for predict_line, test_line in zip(predicted_language, test_file):
    # Gets rid of new lines with strip()
    predicted = predict_line.strip()
    actual = test_line.strip()

    # From the "LangId.sol.txt", split the line and get the second string from split
    line_number = actual.split()[0]
    actual = actual.split()[1]

    # print(predicted)
    # print(actual)
    # print("--------------------")

    if (predicted == actual):
      predicted_right += 1
    else:
      print("Wrong predicted language at line", line_number)
      print("Predicted language:", predicted)
      print("Actual language:", actual)
      print()
      predicted_wrong += 1

print("Number of languages predicted right:", predicted_right)
print("Number of languages predicted wrong:", predicted_wrong)
print("Accuracy: %.2f%%" % ((predicted_right / (predicted_right +predicted_wrong))*100) )
```

```
Wrong predicted language at line 24
Predicted language: English
Actual language: French

Wrong predicted language at line 44
Predicted language: French
Actual language: Italian

Wrong predicted language at line 92
Predicted language: French
Actual language: English

Wrong predicted language at line 187
Predicted language: English
Actual language: Italian

Wrong predicted language at line 191
Predicted language: English
Actual language: French

Wrong predicted language at line 247
Predicted language: English
Actual language: Italian

Wrong predicted language at line 277
Predicted language: English
Actual language: Italian

Wrong predicted language at line 279
Predicted language: English
Actual language: French

Number of languages predicted right: 292
Number of languages predicted wrong: 8
Accuracy: 97.33%
```

When using log probability method, the accuracy of the predicted languages was higher. After manual testing and debugging the simplified Laplace smoothing method, I found out that the probabilty, for some cases, reaches zero in Python when the value reaches very close to zero, otherwise known as underflow. Logging the probability resolves this issue. For this assignment, I kept simplified Laplace smoothing method according to the instructions, but below is the result of the Log of the Laplace smoothing method.

In [145…
```
# NOTE: results of the log of the Laplace smoothing
test_path = folder_path + "LangId.sol.txt"
predicted_right = 0
predicted_wrong = 0

with open(write_file_path, 'r') as predicted_language, open(test_path, 'r') as test_file:
  for predict_line, test_line in zip(predicted_language, test_file):
    # Gets rid of new lines with strip()
    predicted = predict_line.strip()
    actual = test_line.strip()

    # From the "LangId.sol.txt", split the line and get the second string from split
    line_number = actual.split()[0]
    actual = actual.split()[1]

    # print(predicted)
    # print(actual)
    # print("--------------------")

    if (predicted == actual):
      predicted_right += 1
    else:
      print("Wrong predicted language at line", line_number)
      print("Predicted language:", predicted)
      print("Actual language:", actual)
      print()
      predicted_wrong += 1

print("Number of languages predicted right:", predicted_right)
print("Number of languages predicted wrong:", predicted_wrong)
print("Accuracy: %.2f%%" % ((predicted_right / (predicted_right +predicted_wrong))*100) )
```

```
Wrong predicted language at line 44
Predicted language: French
Actual language: Italian

Wrong predicted language at line 92
Predicted language: French
Actual language: English

Number of languages predicted right: 298
Number of languages predicted wrong: 2
Accuracy: 99.33%
```