Name: Jakob Germann
Completion Date: 2/23/2022

Homework 2 Report

Part 1: Viterbi Algorithm

Viterbi algorithm code was taken from the textbook and used in this implementation.

```
def Viterbi(model, observations):

    # Length of observations
    T = len(observations)

    # Number of states
    N = len(model.states)

    viterbiMatrix = np.zeros(shape=(N, T))

    translation = Translate(observations)
    backpointer = np.zeros(shape=(N, T))

    # Initialization step
    for s in range(N):
        viterbiMatrix[s][0] = model.startProbabilities[s] * model.emissions[translation[0]][s]
        backpointer[s][0] = 0

    # For the remaining number of observations
    for t in range(1, T):
        # For each of the hidden states, N
        for s in range(N):
            transition_probs = list()
            for sprime in range(N):
                transition_probs.append(viterbiMatrix[sprime][t-1] * model.states[sprime][s] *
model.emissions[translation[t]][s])

            viterbiMatrix[s][t] = max(transition_probs)


            backpointer[s][t] = np.argmax(viterbiMatrix[:, t-1])

    # Get the overall probability of the best path through the HMM
    bestPathProbability = 0
    bestPathPointer = list()
    TViterbi = viterbiMatrix.transpose()
```

Name: Jakob Germann
Completion Date: 2/23/2022

```
    bestPathProbability = max(TViterbi[T-1][:])

    bestPathPointer = np.argmax(TViterbi[T-1][:])

    bestPath = FindBestPath(viterbiMatrix, T)

    print("\n")
    print("Viterbi Matrix: ")
    print(viterbiMatrix)
    print("\n")
    print("Backtrace Matrix: ")
    print(backpointer)
    print("\n")
    print("Best Path(from t = 0 to t = T): ", end="")
    bestPath.reverse()
    print(bestPath)
```

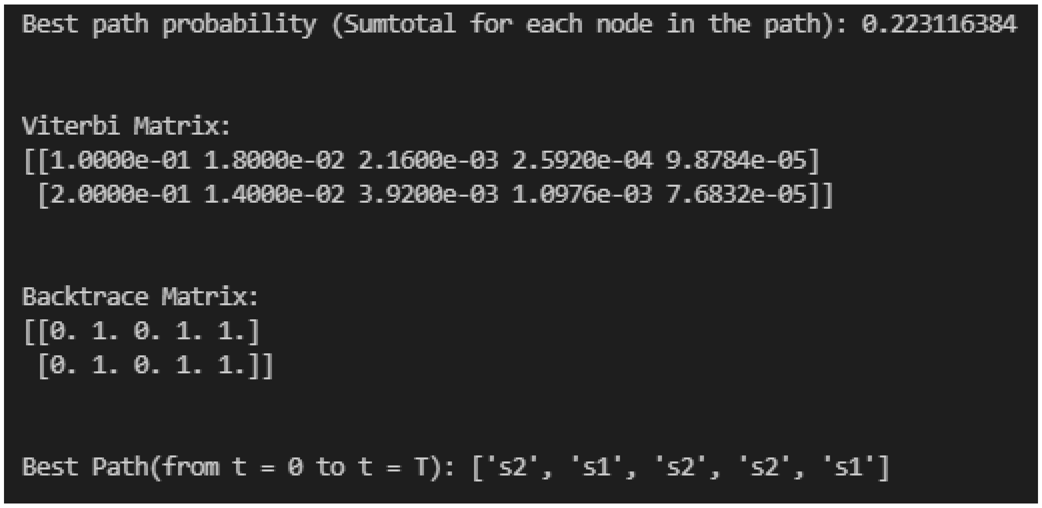
Below are the auxiliary functions mentioned throughout the algorithm:

```
def FindBestPath(viterbiMatrix, T):
    bestPath = list()
    TViterbi = viterbiMatrix.transpose()
    bestPathProbability = 0

    i = T-1

    # For the number of observations within backpointer, starting at the position of
bestPathPointer
    while i >= 0:

        # Get the position with the maximum probability at each observation
        best_state = np.argmax(TViterbi[i])
        if best_state == 0:
            bestPath.append('s1')
            bestPathProbability += TViterbi[i][best_state]
        else:
            bestPath.append('s2')
            bestPathProbability += TViterbi[i][best_state]
        i -= 1
    print("Best path probability (Sumtotal for each node in the path): " +
str(bestPathProbability))
    return bestPath
```

Name: Jakob Germann
Completion Date: 2/23/2022

```python
def Translate(observations):

    translation = list()

    for i in observations:
        if i == 'A':
            translation.append(0)
        elif i == 'C':
            translation.append(1)
        elif i == 'G':
            translation.append(2)
        elif i == 'T':
            translation.append(3)
        else:
            print("Unrecognized character. Exiting program...")
            exit()

    return translation
```

The output printed to the terminal is listed below:



```
Best path probability (Sumtotal for each node in the path): 0.223116384


Viterbi Matrix:
[[1.0000e-01 1.8000e-02 2.1600e-03 2.5920e-04 9.8784e-05]
 [2.0000e-01 1.4000e-02 3.9200e-03 1.0976e-03 7.6832e-05]]


Backtrace Matrix:
[[0. 1. 0. 1. 1.]
 [0. 1. 0. 1. 1.]]


Best Path(from t = 0 to t = T): ['s2', 's1', 's2', 's2', 's1']
```

NOTES ON THE OUTPUT:
- The best path is printed starting from position 0 in the matrix.
- The probability is the sum of all greatest probabilities from each time step.
    - Otherwise, the best probability at the last time step is 0.000009.

Name: Jakob Germann
Completion Date: 2/23/2022

Part 2: Multinomial Naive Bayes w/t SpaCy & Sklearn

This part included two implementations. Model 1 uses pure sklearn libraries which were permitted for use by Dr. Liu on 2/22/2022. Model 2 incorporates a SpaCy pipeline for text preprocessing, then passes the processed data to Sklearn's multinomial naive bayes model.

The implementation was done in a colaboratory environment and is recommended that the included source code be run the same way.

Model 1: Pure Sklearn:

The first model used a simple pipeline consisting of three components:

1. CountVectorizer: This created a vectorized bag of words from all documents in the training section.
2. TfidfTransformer: This takes the vectorized bag of words and normalizes it. This is to reduce the negative impact of frequently occurring features within the bag. Features that appear less frequently can thus have a similar impact and the model will not be affected by biases.
3. MultinomialNB: The sklearn multinomial naive bayes model is then fed the normalized bag of words, fitted to the training set, and tested.

```
"""HW2_Part2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1uiDIF6UQZpaIQzQHr3dCBIvqrGDfV3Md
"""

# NOTE: As of 2/22/2022, Dr. Liu allowed the use of sklearn preprocessing
methods along with
# spaCy to develop the classifier.  She also allowed the use of sklearn's
premade 20newsgroups dataset.

import numpy as np
from sklearn import datasets
from sklearn.naive_bayes import MultinomialNB
from sklearn import pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

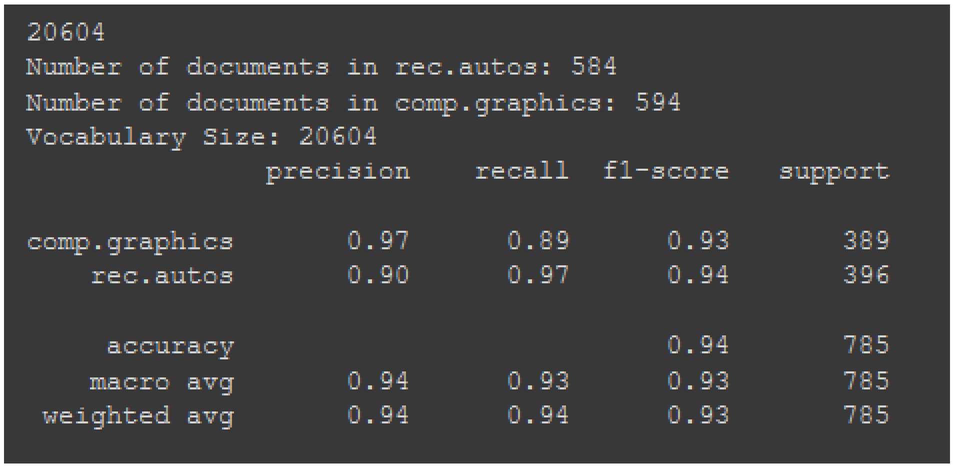
Name: Jakob Germann
Completion Date: 2/23/2022

```
from sklearn import metrics
import re

"""# Model 1: Pure Sklearn Setup

Sklearn and re library documentation was used in the creation of this program.
"""

# Extract only the train and test datasets for our categories and remove
unecessary components
categories = ['rec.autos', 'comp.graphics']
train = datasets.fetch_20newsgroups(subset='train', categories=categories,
remove=('headers', 'footers', 'quotes'), shuffle=True)
test = datasets.fetch_20newsgroups(subset='test', categories=categories,
remove=('headers', 'footers', 'quotes'), shuffle=True, random_state=42)

# Get the number of documents in each category for train set
cat1 = 0
cat2 = 0
for i in range(len(train.target)):
  if train.target[i] == 0:
    cat1 += 1
  else:
    cat2 += 1

# Remove all numbers and special characters from document texts
bad_patterns = "[^a-zA-Z. ]"

for doc in range(len(train.data)):
  new_doc = re.sub(bad_patterns, '', train.data[doc])
  train.data[doc] = new_doc

# Tokenize all documents in our training set and get the vocabulary.
vectorizer = CountVectorizer()
vectorizer.fit_transform(train.data)
vocabulary = vectorizer.vocabulary_
print(len(vocabulary))

# Use TFidfVectorizer() as initial pipeline to handle current setup of the
dataset.
model = pipeline.make_pipeline(CountVectorizer(), TfidfTransformer(),
MultinomialNB())

model.fit(train.data, train.target)
```

Name: Jakob Germann
Completion Date: 2/23/2022

```
predicted = model.predict(test.data)

print("Number of documents in rec.autos: " + str(cat1))
print("Number of documents in comp.graphics: " + str(cat2))
print("Vocabulary Size: " + str(len(vocabulary)))
print(metrics.classification_report(test.target, predicted,
target_names=test.target_names))
```

The results of model 1 are shown below:



```
20604
Number of documents in rec.autos: 584
Number of documents in comp.graphics: 594
Vocabulary Size: 20604
               precision    recall  f1-score   support

comp.graphics       0.97      0.89      0.93       389
    rec.autos       0.90      0.97      0.94       396

     accuracy                           0.94       785
    macro avg       0.94      0.93      0.93       785
 weighted avg       0.94      0.94      0.93       785
```

Model 2: Sklearn + SpaCy

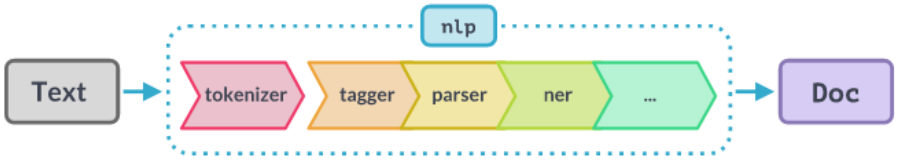
For this model, I pre-processed all documents from both categories through a SpaCy pipeline. **Please note that the vocabulary size and number of documents within both categories does not change between models. Thus, this information was outputted only in model 1's results.**

The pipeline consisted of the following components:

1. Sentencizer: Apply sentence segmentation
2. Tokenizer: Apply word tokenization (Works in the background prior to Tagger)
3. Tagger: Assigned part-of-speech tags
4. Parser: Assigning dependency labels
5. Entity Recognizer: Detect and label entities

Name: Jakob Germann
Completion Date: 2/23/2022

The below illustration details a general overview of how the pipeline works, excluding the sentence segmentation component:



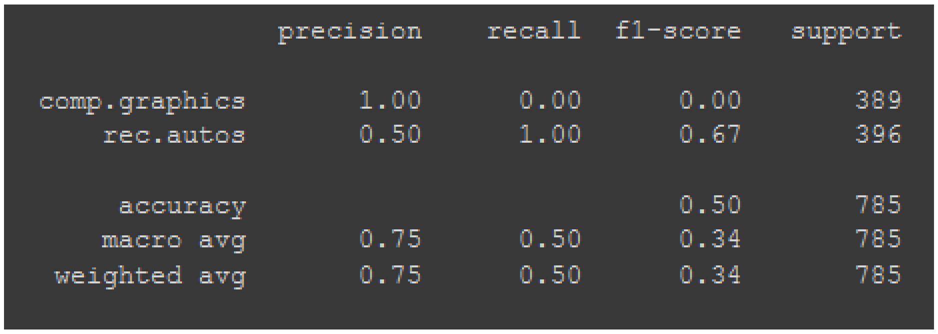
The results shown at the bottom displayed very poor results overall, when in comparison to the pure sklearn model. No errors were thrown during this implementation, but the current pipeline may be insufficient or be in the incorrect order for SpaCy to properly process the documents.

```
"""# Model 2: Combo of Sklearn and SpaCy Features

Predefined sklearn pipeline is used below and then fed
to sklearn's CountVectorizer method.

Sklearn, re, and spaCy documentation was used in the creation of this program.

"""

import spacy as sp
from spacy.lang.en.stop_words import STOP_WORDS

# Re-import unedited datasets for new model
train2 =  datasets.fetch_20newsgroups(subset='train', remove=('headers',
'footers', 'quotes'), categories=categories,  shuffle=True)
test2 = datasets.fetch_20newsgroups(subset='test', categories=categories,
remove=('headers', 'footers', 'quotes'), shuffle=True, random_state=42)

# Remove unecessary characters like numbers and special non-punctuation
characters
bad_patterns = "[^a-zA-Z.]"

for doc in range(len(train2.data)):
  new_doc = re.sub(bad_patterns, '', train2.data[doc])
  train2.data[doc] = new_doc

# Process each document individually using the below steps
# NOTE: Colab's spaCy library version is 2.2.4.  Only version 3.0 has
# lemmatization as a separate pipeline component.  Thus, lemmatization is
# implemented, but acts behind the scenes of the parser component.

# Import premade English processing pipeline
nlp = sp.load("en_core_web_sm")
```

Name: Jakob Germann
Completion Date: 2/23/2022

```
# Add sentence segmentation
sentencizer = nlp.create_pipe("sentencizer")
nlp.add_pipe(sentencizer)

# spaCy pipeline for pre-processing
def spacy_pipeline(document):
  doc = nlp(document)
  return doc

# Create bag of words with spacy tokenizer
vectorizer = CountVectorizer(tokenizer=spacy_pipeline)

# Train the model
# model2 = pipeline.Pipeline([("bow", bag_of_words), ("classifier",
classifier)])
model2 = pipeline.make_pipeline(vectorizer, TfidfTransformer(),
MultinomialNB())
model2.fit(train2.data, train2.target)

predicted2 = model2.predict(test2.data)

print(metrics.classification_report(test2.target, predicted2,
target_names=test2.target_names, zero_division=1))
```

The results of model 2 are shown below:



```
               precision    recall  f1-score   support

comp.graphics       1.00      0.00      0.00       389
    rec.autos       0.50      1.00      0.67       396

     accuracy                           0.50       785
    macro avg       0.75      0.50      0.34       785
 weighted avg       0.75      0.50      0.34       785
```