```
import matplotlib.pyplot as plt
import numpy as np
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
```

```
% pip install pomegranate
#Pomegranate is a graphical models library for Python, implemented in Cython for speed
```

```
Collecting pomegranate
  Downloading https://files.pythonhosted.org/packages/60/8a/51bb4268722c26f67738a0da8ab43df49bbb01016a135b6aa1c45bd33670/pomegra
     |████████████████████████████████| 3.3MB 2.8MB/s
Requirement already satisfied: numpy>=1.8.0 in /usr/local/lib/python3.6/dist-packages (from pomegranate) (1.18.5)
Requirement already satisfied: joblib>=0.9.0b4 in /usr/local/lib/python3.6/dist-packages (from pomegranate) (0.15.1)
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-packages (from pomegranate) (2.4)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from pomegranate) (1.4.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages (from pomegranate) (3.13)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.6/dist-packages (from networkx>=2.0->pomegranate) (4.4
Building wheels for collected packages: pomegranate
  Building wheel for pomegranate (setup.py) ... done
  Created wheel for pomegranate: filename=pomegranate-0.13.3-cp36-cp36m-linux_x86_64.whl size=10950002 sha256=d05b25ec82387428f0
  Stored in directory: /root/.cache/pip/wheels/92/a4/39/0f9e71a9134d03801477bffa6e02257020f12e5cbb031a8489
Successfully built pomegranate
Installing collected packages: pomegranate
Successfully installed pomegranate-0.13.3
```

```
import sys
sys.path.append("/content/drive/My Drive/Colab Notebooks/assignment")
```

```
from IPython.core.display import HTML
from itertools import chain
from collections import Counter,defaultdict
```

```
from helpers import show_model, Dataset
```

```
data = Dataset("/content/drive/My Drive/Colab Notebooks/assignment/tags-universal.txt", "/content/drive/My Drive/Colab Notebooks/assi

print("There are {} sentences in the corpus.".format(len(data)))
print("There are {} sentences in the training set.".format(len(data.training_set)))
print("There are {} sentences in the testing set.".format(len(data.testing_set)))

assert len(data) == len(data.training_set) + len(data.testing_set), \
       "The number of sentences in the training set + testing set should sum to the number of sentences in the corpus"
```

```
There are 57340 sentences in the corpus.
There are 45872 sentences in the training set.
There are 11468 sentences in the testing set.
```

```
key = 'b100-38532'
print("Sentence: {}".format(key))
print("words:\n\t{!s}".format(data.sentences[key].words))
print("tags:\n\t{!s}".format(data.sentences[key].tags))
```

```
Sentence: b100-38532
words:
        ('Perhaps', 'it', 'was', 'right', ';', ';')
tags:
        ('ADV', 'PRON', 'VERB', 'ADJ', '.', '.')
```

```
print("There are a total of {} samples of {} unique words in the corpus."
      .format(data.N, len(data.vocab)))
print("There are {} samples of {} unique words in the training set."
      .format(data.training_set.N, len(data.training_set.vocab)))
print("There are {} samples of {} unique words in the testing set."
```

```
      .format(data.testing_set.N, len(data.testing_set.vocab)))
print("There are {} words in the test set that are missing in the training set."
      .format(len(data.testing_set.vocab - data.training_set.vocab)))

assert data.N == data.training_set.N + data.testing_set.N, \
       "The number of training + test samples should sum to the total number of samples"
```

```
There are a total of 1161192 samples of 56057 unique words in the corpus.
There are 928458 samples of 50536 unique words in the training set.
There are 232734 samples of 25112 unique words in the testing set.
There are 5521 words in the test set that are missing in the training set.
```

```
# accessing words with Dataset.X and tags with Dataset.Y
for i in range(5):
    print("Sentence {}:".format(i + 1), data.X[i])
    print()
    print("Labels {}:".format(i + 1), data.Y[i])
    print()
```

```
    Sentence 1: ('Mr.', 'Podger', 'had', 'thanked', 'him', 'gravely', ',', 'and', 'now', 'he', 'made', 'use', 'of', 'the', 'advice',
```

```
# use Dataset.stream() (word, tag) samples for the entire corpus
print("\nStream (word, tag) pairs:\n")
for i, pair in enumerate(data.stream()):
    print("\t", pair)
    if i > 10: break
```

```
    Stream (word, tag) pairs:

             ('Mr.', 'NOUN')
             ('Podger', 'NOUN')
             ('had', 'VERB')
             ('thanked', 'VERB')
             ('him', 'PRON')
             ('gravely', 'ADV')
             (',', '.')
             ('and', 'CONJ')
             ('now', 'ADV')
             ('he', 'PRON')
             ('made', 'VERB')
             ('use', 'NOUN')
```

```
from collections import Counter,defaultdict

def pair_counts(tags,words):
  d=defaultdict(lambda: defaultdict(int))
  for tag,word in zip(tags,words):
    d[tag][word]+=1
  return d
  raise NotImplementedError
tags = [tag for i,(word, tag) in enumerate(data.training_set.stream())]
words = [word for i,(word, tag) in enumerate(data.training_set.stream())]

emission_counts =pair_counts(tags,words)

tags
```

```
['ADV',
 'NOUN',
 '.',
 'ADV',
 '.',
 'VERB',
 'ADP',
 'ADJ',
 'NOUN',
 'CONJ',
 'VERB',
 'ADJ',
 'NOUN',
 '.',
 'DET',
 'VERB',
 'ADJ',
 'PRT',
 'ADP',
 'NUM',
 'NOUN',
 '.',
 'PRON',
 'VERB',
 'PRT',
 'VERB',
 'NOUN',
 '.',
 'VERB',
 'NOUN',
 'NUM',
 '.',
 'NUM',
 '.',
 '.',
 '.',
 'ADP',
 'ADV',
 'NUM',
 'NOUN',
 '.',
```

```
'.',
'DET',
'NOUN',
'CONJ',
'DET',
'NOUN',
'VERB',
'ADP',
'NOUN',
'VERB',
'ADP',
'DET',
'NOUN',
'ADP',
'NOUN',
'CONJ',
'NOUN',
'.',
'NOUN',
'.',
'NOUN',
'.',
'NOUN',
'.',
'VERB',
'.',
'VERB',
'ADP',
'NOUN',
'.',
'VERB',
'.',
'VERB',
'.',
'VERB',
'.',
'VERB',
'.',
'ADJ',
'NOUN',
'ADV',
```

```
'.',
'NOUN',
'.',
'PRON',
'VERB',
'VERB',
'ADP',
'DET',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'.',
'ADV',
'DET',
'ADJ',
'NOUN',
'DET',
'NOUN',
'CONJ',
'NOUN',
'ADP',
'NOUN',
'VERB',
'ADJ',
'.',
'NOUN',
'VERB',
'DET',
'ADJ',
'VERB',
'NOUN',
'NOUN',
'ADP',
'NOUN',
'NUM',
'.',
'NUM',
'.',
'PRT',
```

```
'DET',
'NOUN',
'DET',
'NOUN',
'VERB',
'ADV',
'CONJ',
'VERB',
'NOUN',
'NOUN',
'.',
'DET',
'NOUN',
'ADJ',
'NOUN',
'ADP',
'DET',
'NOUN',
'VERB',
'NOUN',
'VERB',
'DET',
'.',
'NOUN',
'NOUN',
'.',
'.',
'PRON',
'VERB',
'ADV',
'ADV',
'ADJ',
'ADP',
'PRON',
'VERB',
'DET',
'NOUN',
'ADP',
'NOUN',
'CONJ',
```

```
 'DET',
'NOUN',
'ADP',
'NOUN',
'.',
'ADP',
'ADP',
'NOUN',
'PRON',
'VERB',
'VERB',
'ADV',
'PRT',
'VERB',
'DET',
'ADJ',
'NOUN',
'.',
'DET',
'VERB',
'NOUN',
'VERB',
'VERB',
'PRT',
'VERB',
'DET',
'ADJ',
'NOUN',
'ADP',
'DET',
'VERB',
'NOUN',
'.',
'DET',
'NOUN',
'ADP',
'DET',
'NOUN',
'VERB',
'ADV',
'ADJ',
```

```
'ADP',
'DET',
'NOUN',
'ADP',
'DET',
'ADJ',
'NOUN',
'.',
'NOUN',
'VERB',
'.',
'VERB',
'DET',
'NOUN',
'.',
'NOUN',
'VERB',
'.',
'NOUN',
'NOUN',
'.',
'PRON',
'VERB',
'PRT',
'DET',
'NOUN',
'CONJ',
'NOUN',
'ADP',
'NOUN',
'ADJ',
'ADJ',
'NOUN',
'NOUN',
'.',
'VERB',
'VERB',
'DET',
'ADJ',
'ADJ',
'NOUN',
```

```
'.',
'ADP',
'DET',
'ADJ',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'NOUN',
'NOUN',
'.',
'ADP',
'PRON',
'VERB',
'ADP',
'DET',
'NOUN',
'NOUN',
'.',
'NOUN',
'NOUN',
'.',
'DET',
'NOUN',
'NOUN',
'NOUN',
'.',
'.',
'NOUN',
'.',
'ADJ',
'.',
'NOUN',
'.',
'ADP',
'NOUN',
```

```
'NOUN',
'VERB',
'ADJ',
'ADV',
'PRT',
'VERB',
'PRON',
'.',
'VERB',
'ADV',
'ADV',
'ADJ',
'ADP',
'NOUN',
'.',
'CONJ',
'PRON',
'VERB',
'NOUN',
'.',
'PRON',
'VERB',
'VERB',
'DET',
'ADJ',
'NOUN',
'ADP',
'ADJ',
'NOUN',
'.',
'CONJ',
'PRON',
'VERB',
'VERB',
'NOUN',
'DET',
'ADV',
'VERB',
'NOUN',
'CONJ',
'NOUN',
'ADP',
```

```
'ADP',
'PRON',
'VERB',
'DET',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'PRON',
'VERB',
'ADP',
'DET',
'NOUN',
'.',
'ADV',
'ADP',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'CONJ',
'ADV',
'ADP',
'ADJ',
'NOUN',
'.',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'.',
'DET',
'NOUN',
'VERB',
'ADV',
'VERB',
'CONJ',
'VERB',
```

```
'VERB',
'VERB',
'PRT',
'VERB',
'ADJ',
'NOUN',
'ADP',
'VERB',
'NOUN',
'.',
'DET',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'VERB',
'DET',
'VERB',
'NOUN',
'ADV',
'ADJ',
'NOUN',
'.',
'ADV',
'.',
'DET',
'NOUN',
'VERB',
'ADP',
'NOUN',
'VERB',
'ADP',
'.',
'ADJ',
'NOUN',
'.',
'.',
'.',
'NOUN',
'.',
'.',
```

```
'.',
'ADJ',
'NOUN',
'.',
'CONJ',
'ADJ',
'NOUN',
'PRON',
'VERB',
'VERB',
'VERB',
'ADP',
'DET',
'NOUN',
'.',
'DET',
'NOUN',
'NOUN',
'VERB',
'NUM',
'NOUN',
'.',
'ADV',
'VERB',
'NOUN',
'ADP',
'ADP',
'NOUN',
'.',
'NOUN',
'ADP',
'NOUN',
'VERB',
'VERB',
'ADP',
'NUM',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'ADJ'
```

```
 'ADJ',
'NOUN',
'NOUN',
'.',
'CONJ',
'PRON',
'VERB',
'DET',
'NOUN',
'.',
'ADV',
'ADJ',
'ADP',
'PRT',
'DET',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'PRON',
'DET',
'ADJ',
'NOUN',
'NOUN',
'ADP',
'DET',
'ADJ',
'ADJ',
'NOUN',
'VERB',
'VERB',
'ADV',
'.',
'PRT',
'NUM',
'NUM',
'NOUN',
'VERB',
'VERB',
'.',
```

```
'DET',
'ADJ',
'NOUN',
'ADP',
'ADJ',
'NOUN',
'.',
'PRON',
'VERB',
'VERB',
'PRON',
'VERB',
'VERB',
'DET',
'NOUN',
'CONJ',
'NOUN',
'ADP',
'NOUN',
'DET',
'NOUN',
'.',
'NOUN',
'.',
'NOUN',
'NUM',
'DET',
'NOUN',
'VERB',
'ADP',
'DET',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'.',
'ADP',
'ADP',
'PRT',
'VERB',
'NUM',
```

```
'NOUN',
'NOUN',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'ADP',
'NOUN',
'.',
'PRON',
'VERB',
'.',
'NOUN',
'VERB',
'ADP',
'PRON',
'.',
'CONJ',
'ADV',
'NOUN',
'VERB',
'ADP',
'PRON',
'PRON',
'ADV',
'VERB',
'PRT',
'VERB',
'DET',
'NUM'.
```

```
'NOUN',
'PRON',
'ADV',
'VERB',
'ADP',
'PRON',
'.',
'ADP',
'DET',
'ADJ',
'NOUN',
'VERB',
'ADJ',
'ADV',
'.',
'DET',
'NOUN',
'ADV',
'VERB',
'VERB',
'PRON',
'ADP',
'DET',
'NOUN',
'.',
'.',
'ADJ',
'ADP',
'VERB',
'DET',
'ADJ',
'NOUN',
'ADP',
'DET',
'NOUN',
'DET',
'PRON',
'VERB',
'ADV',
'VERB',
'.'
```

```
 . ,
'DET',
'NOUN',
'VERB',
'PRT',
'VERB',
'ADJ',
'NOUN',
'ADV',
'DET',
'ADJ',
'NOUN',
'ADP',
'DET',
'NOUN',
'VERB',
'ADJ',
'.',
'ADP',
'ADJ',
'NOUN',
'NOUN',
'.',
'ADP',
'NOUN',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'ADP',
'NOUN',
'.',
'ADP',
'ADJ',
'NOUN',
'ADP',
'ADJ',
'NOUN',
'.',
'ADP',
```

```
'DET',
'VERB',
'NOUN',
'.',
'ADP',
'DET',
'NOUN',
'NOUN',
'.',
'ADP',
'ADJ',
'NOUN',
'.',
'ADV',
'ADP',
'DET',
'ADJ',
'NOUN',
'.',
'NOUN',
'ADJ',
'ADP',
'DET',
'NUM',
'NOUN',
'VERB',
'PRON',
'ADP',
'DET',
'NOUN',
'.',
'DET',
'NOUN',
'ADP',
'NOUN',
'ADP',
'NOUN',
'ADV',
'ADP',
'NOUN',
'ADP',
```

```
'ADJ',
'NOUN',
'ADP',
'DET',
'VERB',
'NOUN',
'ADP',
'NUM',
'VERB',
'VERB',
'PRT',
'VERB',
'ADV',
'NUM',
'NUM',
'NOUN',
'NOUN',
'.',
'ADJ',
'NOUN',
'ADP',
'PRON',
'.',
'ADP',
'ADJ',
'NOUN',
'CONJ',
'ADJ',
'NOUN',
'.',
'DET',
'NOUN',
'VERB',
'ADP',
'PRON',
'VERB',
'ADJ',
'PRT',
'VERB',
'NUM',
'ADP'
```

'ADP',
'NUM',
'NOUN',
'ADP',
'NOUN',
'ADP',
'NOUN',
'ADP',
'NOUN',
'NOUN',
'VERB',
'.',
'ADP',
'VERB',
'ADP',
'DET',
'NOUN',
'NOUN',
'.',
'PRT',
'VERB',
'VERB',
'DET',
'NOUN',
'PRT',
'VERB',
'DET',
'NOUN',
'VERB',
'ADJ',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'ADV',
'ADP',
'DET',
'NOUN',
'ADP',
'DET',

```
'ADJ',
'NOUN',
'ADP',
'NOUN',
'.',
'PRON',
'VERB',
'ADP',
'NOUN',
'CONJ',
'VERB',
'PRT',
'DET',
'NOUN',
'.',
'CONJ',
'NOUN',
'NOUN',
'VERB',
'PRT',
'VERB',
'VERB',
'PRON',
'VERB',
'VERB',
'VERB',
'DET',
'NOUN',
'PRT',
'VERB',
'NOUN',
'NOUN',
'NOUN',
'.',
'DET',
'.',
'ADV',
'.',
'VERB',
'ADV',
'VERB',
```

```
'NOUN',
'ADP',
'DET',
'ADP',
'DET',
'VERB',
'NUM',
'NOUN',
'.',
'DET',
'ADJ',
'NOUN',
'VERB',
'VERB',
'PRT',
'VERB',
'.',
'ADP',
'PRON',
'VERB',
'VERB',
'PRT',
'ADP',
'DET',
'NOUN',
'ADP',
'PRON',
'VERB',
'.',
'VERB',
'ADP',
'NOUN',
'ADP',
'DET',
'NOUN',
'VERB',
'NOUN',
'.',
'PRON',
'VERB',
'ADV',
```

```
'ADV',
'DET',
'ADJ',
'NOUN',
'VERB',
'VERB',
'ADV',
'PRON',
'VERB',
'ADP',
'DET',
'NOUN',
'.',
'PRON',
'VERB',
'DET',
'NOUN',
'VERB',
'CONJ',
'VERB',
'ADV',
'.',
'VERB',
'NOUN',
'.',
'CONJ',
'ADP',
'DET',
'NOUN',
'DET',
'NOUN',
'.',
'ADJ',
'ADP',
'DET',
'NOUN',
'.',
'VERB',
'NOUN',
'CONJ',
'VERB',
```

'VERB',
'ADP',
'NOUN',
'ADP',
'DET',
'NOUN',
'.',
'.',
'PRT',
'.',
'NOUN',
'.',
'PRON',
'ADV',
'VERB',
'VERB',
'ADP',
'NOUN',
'.',
'CONJ',
'PRON',
'VERB',
'PRT',
'VERB',
'PRON',
'NOUN',
'VERB',
'DET',
'NUM',
'NUM',
'.',
'.',
'PRON',
'VERB',
'PRON',
'ADP',
'DET',
'NOUN',
'ADV',
'.',
'.',

```
      'CONJ',
      'DET',
      'NOUN',
      'VERB',
      'ADV',
      'ADP',
      'NOUN',
      'DET',
      'VERB',
      'ADV',
      'VERB',
      'PRT',
      'VERB',
      'ADJ',
      'NOUN',
      'ADP',
      'DET',
      'ADJ',
      'CONJ',
      'ADJ',
      'NOUN',
      'NOUN',
      '.',
      '.',
      'PRT',
      'VERB',
```

```
# Create a lookup table mfc_table where mfc_table[word] contains the tag label most frequently assigned to that word
from collections import namedtuple

FakeState = namedtuple("FakeState", "name")

class MFCTagger:
    # NOTE: You should not need to modify this class or any of its methods
    missing = FakeState(name="<MISSING>")

    def __init__(self, table):
        self.table = defaultdict(lambda: MFCTagger.missing)
        self.table.update({word: FakeState(name=tag) for word, tag in table.items()})

    def viterbi(self, seq):
```

```
        """This method simplifies predictions by matching the Pomegranate viterbi() interface"""
        return 0., list(enumerate(["<start>"] + [self.table[w] for w in seq] + ["<end>"]))


# TODO: calculate the frequency of each tag being assigned to each word (hint: similar, but not
# the same as the emission probabilities) and use it to fill the mfc_table

tags = [tag for i, (word, tag) in enumerate(data.training_set.stream())]
words = [word for i, (word, tag) in enumerate(data.training_set.stream())]
#Since this is the word_counts we will pass first words and then counts
word_counts = pair_counts(words,tags)

mfc_table = dict((word, max(tags.keys(), key=lambda key: tags[key])) for word, tags in word_counts.items())

# DO NOT MODIFY BELOW THIS LINE
mfc_model = MFCTagger(mfc_table) # Create a Most Frequent Class tagger instance
```

```
def replace_unknown(sequence):
    """Return a copy of the input sequence where each unknown word is replaced
    by the literal string value 'nan'. Pomegranate will ignore these values
    during computation.
    """
    return [w if w in data.training_set.vocab else 'nan' for w in sequence]

def simplify_decoding(X, model):
    """X should be a 1-D sequence of observations for the model to predict"""
    _, state_path = model.viterbi(replace_unknown(X))
    return [state[1].name for state in state_path[1:-1]]  # do not show the start/end state predictions
```

```
for key in data.testing_set.keys[:5]:
    print("Sentence Key: {}\n".format(key))
    print("Predicted labels:\n-----------------")
    print(simplify_decoding(data.sentences[key].words, mfc_model))
    print()
    print("Actual labels:\n--------------")
    print(data.sentences[key].tags)
    print("\n")
```

```
Sentence Key: b100-28144

Predicted labels:
-----------------
['CONJ', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'CONJ', 'NOUN', 'NUM', '.', '.', 'NOUN', '.', '.']

Actual labels:
--------------
('CONJ', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'CONJ', 'NOUN', 'NUM', '.', '.', 'NOUN', '.', '.')


Sentence Key: b100-23146

Predicted labels:
-----------------
['PRON', 'VERB', 'DET', 'NOUN', 'ADP', 'ADJ', 'ADJ', 'NOUN', 'VERB', 'VERB', '.', 'ADP', 'VERB', 'DET', 'NOUN', 'ADP', 'NOUN', '
```

```
def accuracy(X, Y, model):

    correct = total_predictions = 0
    for observations, actual_tags in zip(X, Y):

        # The model.viterbi call in simplify_decoding will return None if the HMM
        # raises an error (for example, if a test sentence contains a word that
        # is out of vocabulary for the training set). Any exception counts the
        # full sentence as an error (which makes this a conservative estimate).
        try:
            most_likely_tags = simplify_decoding(observations, model)
            correct += sum(p == t for p, t in zip(most_likely_tags, actual_tags))
        except:
            pass
        total_predictions += len(observations)
    return correct / total_predictions
```

```
Predicted labels:
```

```
mfc_training_acc = accuracy(data.training_set.X, data.training_set.Y, mfc_model)
print("training accuracy mfc_model: {:.2f}%".format(100 * mfc_training_acc))

mfc_testing_acc = accuracy(data.testing_set.X, data.testing_set.Y, mfc_model)
```

```
mfc_testing_acc = accuracy(data.testing_set.X, data.testing_set.Y, mfc_model)
print("testing accuracy mfc_model: {:.2f}%".format(100 * mfc_testing_acc))
```

```
training accuracy mfc_model: 95.72%
testing accuracy mfc_model: 93.01%
Predicted labels:
```

IMPLEMENTATION of Unigram

```
def unigram_counts(sequences):
    return Counter(sequences)
    raise NotImplementedError
tag_unigrams = unigram_counts(tags)
```

IMPLEMENTATION of Bigram

```
def bigram_counts(sequences):
    return Counter(zip(sequences, sequences[1:]))
    raise NotImplementedError

tag_bigrams = bigram_counts(tags)
```

IMPLEMENTATION of Sequence Starting Counts

```
def starting_counts(sequences):
    return Counter([x[0] for x in sequences])
    raise NotImplementedError
tag_starts = starting_counts(data.training_set.Y)
```

IMPLEMENTATION of Sequence Ending Counts

```
def ending_counts(sequences):
    return Counter([x[-1] for x in sequences])
```

```
    raise NotImplementedError

tag_ends = ending_counts(data.training_set.Y)
```

IMPLEMENTATION of HMM Tagger

```
from pomegranate import State, HiddenMarkovModel, DiscreteDistribution
```

```
basic_model = HiddenMarkovModel(name="base-hmm-tagger")
```

```
tag_counts = pair_counts(tags, words)
tag_state = {}
for tag in data.training_set.tagset:
    for word in data.training_set.vocab:
        try:
            tag_counts[tag][word] /= tag_unigrams[tag]
        except:
            tag_counts[tag][word] = 0
    emission = DiscreteDistribution(dict(tag_counts[tag]))
    tag_state[tag] = State(emission, name=tag)
basic_model.add_states(list(tag_state.values()))
```

```
tag_transition = defaultdict(lambda:0)
for tag in tag_unigrams.keys():
    for tag2 in tag_unigrams.keys():
        basic_model.add_transition(tag_state[tag], tag_state[tag2],
                                   tag_bigrams[(tag,tag2)] / tag_unigrams[tag])
    basic_model.add_transition(basic_model.start, tag_state[tag],
                               tag_starts[tag] / len(data.training_set))
    basic_model.add_transition(tag_state[tag], basic_model.end,
                               tag_ends[tag] / tag_unigrams[tag])
```

```
basic_model.bake()
```

```
observations = ['what', 'is', 'it']
forward_matrix = np.exp(basic_model.forward(observations))
probability_percentage = np.exp(basic_model.log_probability(observations))
print("        " + "".join(s.name.center(len(s.name)+6) for s in basic_model.states))
for i in range(len(observations) + 1):
    print(" <start> " if i==0 else observations[i - 1].center(9), end="")
    print("".join("{:.4f}%".format(100 * forward_matrix[i, j]).center(len(s.name) + 6)
                  for j, s in enumerate(basic_model.states)))

print("\nThe likelihood over all possible paths " + \
      "of this model producing the sequence {} is {:.10f}%\n\n"
      .format(observations, 100 * probability_percentage))
```

```
            .      ADJ      ADP      ADV      CONJ      DET      NOUN      NUM      PRON      PRT      VERB      X      base-hmm-
 <start> 0.0000% 0.0000%  0.0000%  0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%  0.0000%         10
   what  0.0000% 0.0000%  0.0000%  0.0000%  0.0000%   0.2197%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%  0.0000%          0
     is  0.0000% 0.0000%  0.0000%  0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0008%  0.0000%          0
     it  0.0000% 0.0000%  0.0000%  0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%   0.0000%  0.0000%  0.0000%          0

The likelihood over all possible paths of this model producing the sequence ['what', 'is', 'it'] is 0.0000000006%
```

```
for key in data.testing_set.keys[:3]:
    print("Sentence Key: {}\n".format(key))
    print("Predicted labels:\n-----------------")
    print(simplify_decoding(data.sentences[key].words, basic_model))
    print()
    print("Actual labels:\n--------------")
    print(data.sentences[key].tags)
    print("\n")
```

```
Sentence Key: b100-28144

Predicted labels:
-----------------
['CONJ', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'CONJ', 'NOUN', 'NUM', '.', '.', 'NOUN', '.', '.']

Actual labels:
--------------
('CONJ', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'NOUN', 'NUM', '.', 'CONJ', 'NOUN', 'NUM', '.', '.', 'NOUN', '.', '.')


Sentence Key: b100-23146

Predicted labels:
-----------------
['PRON', 'VERB', 'DET', 'NOUN', 'ADP', 'ADJ', 'ADJ', 'NOUN', 'VERB', 'VERB', '.', 'ADP', 'VERB', 'DET', 'NOUN', 'ADP', 'NOUN', 'A

Actual labels:
--------------
('PRON', 'VERB', 'DET', 'NOUN', 'ADP', 'ADJ', 'ADJ', 'NOUN', 'VERB', 'VERB', '.', 'ADP', 'VERB', 'DET', 'NOUN', 'ADP', 'NOUN', 'A


Sentence Key: b100-35462

Predicted labels:
-----------------
['DET', 'ADJ', 'NOUN', 'VERB', 'VERB', 'VERB', 'ADP', 'DET', 'ADJ', 'ADJ', 'NOUN', 'ADP', 'DET', 'ADJ', 'NOUN', '.', 'ADP', 'ADJ

Actual labels:
--------------
('DET', 'ADJ', 'NOUN', 'VERB', 'VERB', 'VERB', 'ADP', 'DET', 'ADJ', 'ADJ', 'NOUN', 'ADP', 'DET', 'ADJ', 'NOUN', '.', 'ADP', 'ADJ
```

```
hmm_training_acc = accuracy(data.training_set.X, data.training_set.Y, basic_model)
print("training accuracy basic hmm model: {:.2f}%".format(100 * hmm_training_acc))

hmm_testing_acc = accuracy(data.testing_set.X, data.testing_set.Y, basic_model)
print("testing accuracy basic hmm model: {:.2f}%".format(100 * hmm_testing_acc))
```

```
training accuracy basic hmm model: 97.52%
testing accuracy basic hmm model: 95.94%
```