```
import numpy as np
import matplotlib.pyplot as plt
import string
from sklearn.model_selection import train_test_split
input_files = [
  'edgar_allan_poe.txt',
  'robert_frost.txt',
]

# collect data into lists
input_texts = []
labels = []

for label, f in enumerate(input_files):
  print(f"{f} corresponds to label {label}")

  for line in open(f):
    line = line.rstrip().lower()
    if line:
      # remove punctuation
      line = line.translate(str.maketrans('', '', string.punctuation))

      input_texts.append(line)
      labels.append(label)
edgar_allan_poe.txt corresponds to label 0
robert_frost.txt corresponds to label 1


train_text, test_text, Ytrain, Ytest = train_test_split(input_texts, labels)
len(Ytrain), len(Ytest)
(1615, 539)
train_text[:5]
['made shift to shelter them without the help',
 'to watch his woods fill up with snow',
 'the meaning of it all is out of you',
 'like harmodious the gallant and good',
 'the board we had laid down to walk dryshod on']
Ytrain[:5]
[1, 1, 1, 0, 1]
idx = 1
word2idx = {'<unk>': 0}



# populate word2idx
for text in train_text:
    tokens = text.split()
    for token in tokens:
      if token not in word2idx:
        word2idx[token] = idx
        idx += 1
word2idx
len(word2idx)
2545
```

```
# convert data into integer format
train_text_int = []
test_text_int = []

for text in train_text:
  tokens = text.split()
  line_as_int = [word2idx[token] for token in tokens]
  train_text_int.append(line_as_int)

for text in test_text:
  tokens = text.split()
  line_as_int = [word2idx.get(token, 0) for token in tokens]
  test_text_int.append(line_as_int)
train_text_int[100:105]
[[142, 71, 389, 390, 391, 94, 7, 392],
 [54, 157, 393, 394, 395],
 [54, 71, 244, 71, 244, 32, 245],
 [71, 396, 5, 212, 397, 398, 53, 223, 7, 116, 242],
 [7, 324, 7, 399, 400, 13, 7, 401]]
```

```
# initialize A and pi matrices - for both classes
V = len(word2idx)

A0 = np.ones((V, V))
pi0 = np.ones(V)

A1 = np.ones((V, V))
pi1 = np.ones(V)
```

```
# compute counts for A and pi
def compute_counts(text_as_int, A, pi):
  for tokens in text_as_int:
    last_idx = None
    for idx in tokens:
      if last_idx is None:
        # it's the first word in a sentence
        pi[idx] += 1
      else:
        # the last word exists, so count a transition
        A[last_idx, idx] += 1

      # update last idx
      last_idx = idx


compute_counts([t for t, y in zip(train_text_int, Ytrain) if y == 0], A0, pi0)
compute_counts([t for t, y in zip(train_text_int, Ytrain) if y == 1], A1, pi1)
```

```
# normalize A and pi so they are valid probability matrices
# convince yourself that this is equivalent to the formulas shown before
A0 /= A0.sum(axis=1, keepdims=True)
pi0 /= pi0.sum()

A1 /= A1.sum(axis=1, keepdims=True)
pi1 /= pi1.sum()

# log A and pi since we don't need the actual probs
logA0 = np.log(A0)
logpi0 = np.log(pi0)

logA1 = np.log(A1)
logpi1 = np.log(pi1)
```

```
# compute priors
count0 = sum(y == 0 for y in Ytrain)
count1 = sum(y == 1 for y in Ytrain)
total = len(Ytrain)
p0 = count0 / total
p1 = count1 / total
logp0 = np.log(p0)
logp1 = np.log(p1)
p0, p1
(0.33126934984520123, 0.6687306501547987)
```

```
# build a classifier
class Classifier:
  def __init__(self, logAs, logpis, logpriors):
    self.logAs = logAs
    self.logpis = logpis
    self.logpriors = logpriors
    self.K = len(logpriors) # number of classes

  def _compute_log_likelihood(self, input_, class_):
    logA = self.logAs[class_]
    logpi = self.logpis[class_]

    last_idx = None
    logprob = 0
    for idx in input_:
      if last_idx is None:
        # it's the first token
        logprob += logpi[idx]
      else:
        logprob += logA[last_idx, idx]

      # update last_idx
      last_idx = idx

    return logprob
  def predict(self, inputs):
    predictions = np.zeros(len(inputs))
    for i, input_ in enumerate(inputs):
      posteriors = [self._compute_log_likelihood(input_, c) + self.logpriors[c]
\
             for c in range(self.K)]
      pred = np.argmax(posteriors)
      predictions[i] = pred
    return predictions

# each array must be in order since classes are assumed to index these lists
clf = Classifier([logA0, logA1], [logpi0, logpi1], [logp0, logp1])
Ptrain = clf.predict(train_text_int)
print(f"Train acc: {np.mean(Ptrain == Ytrain)}")
Train acc: 0.9969040247678018
Ptest = clf.predict(test_text_int)
print(f"Test acc: {np.mean(Ptest == Ytest)}")
Test acc: 0.8256029684601113

from sklearn.metrics import confusion_matrix, f1_score

# read about F-score: https://en.wikipedia.org/wiki/F-score
cm = confusion_matrix(Ytrain, Ptrain)
cm
array([[ 530,    5],
       [   0, 1080]])
cm_test = confusion_matrix(Ytest, Ptest)
cm_test
array([[ 97,  86],
       [  8, 348]])
f1_score(Ytrain, Ptrain)
0.997690531177829
f1_score(Ytest, Ptest)
0.8810126582278481
```