```
#!/usr/bin/env python

import sys
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Implementation of ln with 0 in domain
def ln(x):
  if x == 0:
    return -math.inf
  else:
    return math.log(x)

# Maximum-likelihood estimate
def ml(k, m, a, N, K):
  return m[k]/N

# Maximum a posteriori estimation
def mp(k, m, a, N, K):
  return (m[k] + a[k] - 1)/(N + a[0] - K)

# Predictive distribution estimation
def pd(k, m, a, N, K):
  return (m[k] + a[k])/(N + a[0])

# Gamma function
def r(n):
  if n == 0:
    return math.inf
  else:
    return math.factorial(n - 1)


# Training data class
class TrainingData:

  # Initialize vocabulary
  def __init__(self, vocabulary, training, **kwargs):
    a = kwargs.get('a', 2)
    hi_freq = kwargs.get('hi_freq', None)

    # Construct dictionary with words as key and # of
    # occurrences in training data as value
    words = list(set(vocabulary))
    self.vocab = dict(zip(words, [0]*len(words)))
    for word in training:
      self.vocab[word] += 1
```

```
    if hi_freq:
      for word in training:
        if self.vocab[word] < 50:
          self.vocab[word] = 0

    # Construct dictionary with words as key and
    # fixed prior (2) as value
    self.a = dict(zip([0] + words, [a*len(words)] + [a]*len(words)))

  def perplexity(self, model, data):
    m = self.vocab
    a = self.a
    N = len(data)
    K = len(self.vocab)
    sum_lnpr = sum(map(lambda k:ln(model(k, m, a, N, K)), data))
    return math.exp((-1 / N) * sum_lnpr)

  def ln_evidence(self, data):
    m = self.vocab
    a = self.a
    N = len(data)
    K = len(self.vocab)
    ln_prod_r_am = sum(map(lambda k:ln(r(a[k] + m[k])), data))
    ln_prod_r_a  = sum(map(lambda k:ln(r(a[k])), data))
    return (ln(r(a[0])) + ln_prod_r_am) - (ln(r(a[0] + N)) + ln_prod_r_a)


# Read in data files and parse into lists
train = open("data/training_data.txt", 'r').read().split()
test = open("data/test_data.txt", 'r').read().split()
vocab = train + test

# Initialize t#raining data for all partitions
trainingData_n    = TrainingData(vocab, train)
trainingData_n4   = TrainingData(vocab, train[:int(len(train)/4)])
trainingData_n16  = TrainingData(vocab, train[:int(len(train)/16)])
trainingData_n64  = TrainingData(vocab, train[:int(len(train)/64)])
trainingData_n128 = TrainingData(vocab, train[:int(len(train)/128)])
```

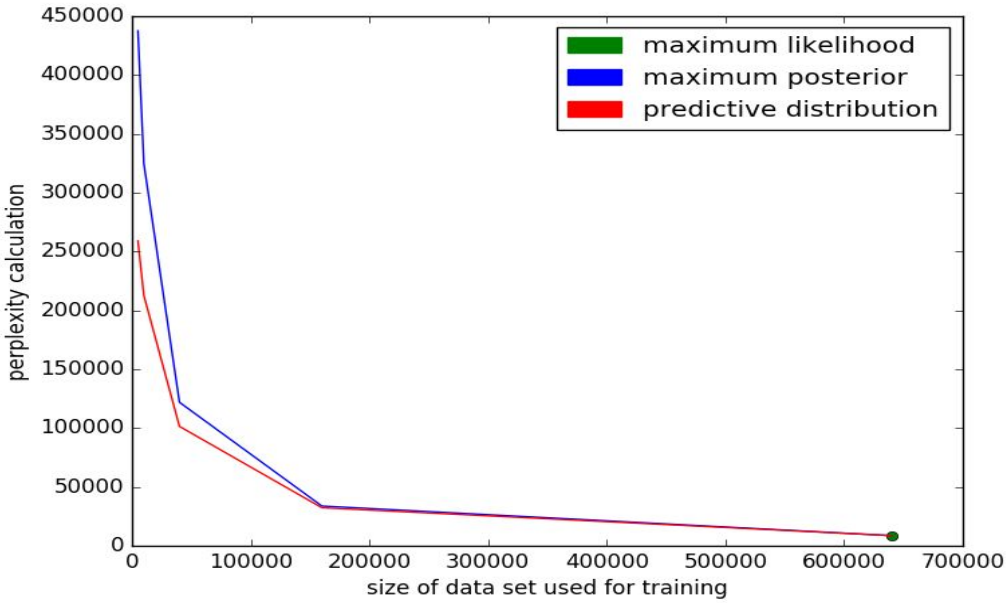
```
# TASK 1:
# Run tests for Maximum-likelihood estimate
ml_results = [
    trainingData_n.perplexity(ml, test),
    trainingData_n4.perplexity(ml, test),
    trainingData_n16.perplexity(ml, test),
    trainingData_n64.perplexity(ml, test),
    trainingData_n128.perplexity(ml, test)]
plt.plot([640000, 160000, 40000, 10000, 5000], ml_results, 'go')

# Run tests for Maximum a posteriori estimation
mp_results = [
    trainingData_n.perplexity(mp, test),
    trainingData_n4.perplexity(mp, test),
    trainingData_n16.perplexity(mp, test),
    trainingData_n64.perplexity(mp, test),
    trainingData_n128.perplexity(mp, test)]
plt.plot([640000, 160000, 40000, 10000, 5000], mp_results, 'b')

# Run tests for Predictive distribution estimation
pd_results = [
    trainingData_n.perplexity(pd, test),
    trainingData_n4.perplexity(pd, test),
    trainingData_n16.perplexity(pd, test),
    trainingData_n64.perplexity(pd, test),
    trainingData_n128.perplexity(pd, test)]
plt.plot([640000, 160000, 40000, 10000, 5000], pd_results, 'r')

# Generate graph to plot results
green = mpatches.Patch(color='green', label='maximum likelihood')
blue = mpatches.Patch(color='blue', label='maximum posterior')
red = mpatches.Patch(color='red', label='predictive distribution')
plt.legend(handles=[green, blue, red])

plt.ylabel('perplexity calculation')
plt.xlabel('size of data set used for training')
plt.show()
```



1. What happens to the test set perplexities of the different methods with respect to each other as the training set size increases? Please explain why this occurs.

As the data set increases the perplexity decrease. Since perplexity measures how well the training data generalizes to the unseen testing data, it is intuitive that as the size of the training data increases the observations in the testing data become more likely with respect to the training data.

2. What is the obvious shortcoming of the maximum likelihood estimate for a unigram model? How do the other two approaches address this issue?

Maximum likelihood estimate has no provision for new observations not in the training data set. So if any new observation is made in the testing data the our model identifies it as impossible. This is why the perplexity of Maximum likelihood perplexity is infinite for all partial training sets. The other models deal with this problem by assigning a likelihood for any unseen observations in our testing data. Unseens observations increase the perplexity but are not assumed to be impossible.

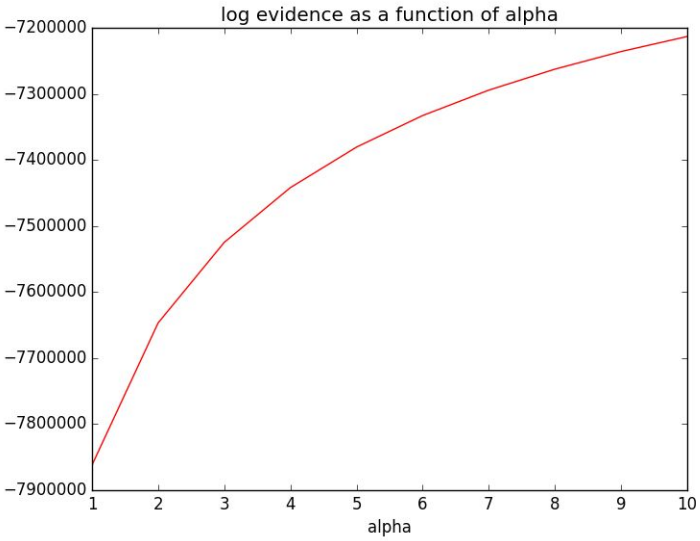
3. For the full training set, how sensitive do you think the test set perplexity will be to small changes in α0? why?

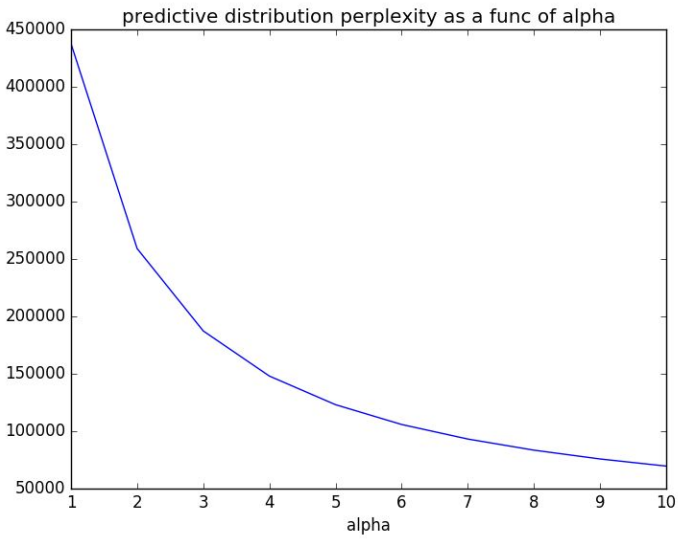
Changing α0 has no effect on maximum likelihood since the model assumes the occurrence of unseen observations is impossible. The perplexity will not be very sensitive to small changes in α0 in the other models because all observations are in the testing data are present in the training data and α0 has the most impact on the likelihood of new observations in our testing data.

```
# TASK 2:
a_vals  = []
ev_vals = []
p_vals  = []
for a in list(range(1,11)):
  trainingData_n128 = TrainingData(vocab, train[:int(len(train)/128)], a=a)
  a_vals.append(a)
  ev_vals.append(trainingData_n128.ln_evidence(test))
  p_vals.append(trainingData_n128.perplexity(pd, test))

# Generate graph to plot results
plt.plot(a_vals, ev_vals, 'r')
plt.ylabel('vals')
plt.xlabel('alpha')
plt.show()

plt.plot(a_vals, p_vals, 'b')
plt.ylabel('vals')
plt.xlabel('alpha')
plt.show()
```





1. Is maximizing the evidence function a good method for model selection on this dataset?

Yes! Maximizing log evidence minimizes perplexity and therefore helps us with model selection.

```
# TASK 3:
pg121  = open("data/pg121.txt.clean", 'r').read().split()
pg1400 = open("data/pg1400.txt.clean", 'r').read().split()
pg141  = open("data/pg141.txt.clean", 'r').read().split()
vocab = pg121 + pg1400 + pg141

trainingData_n = TrainingData(vocab, pg121, 2)
print(trainingData_n.perplexity(pd, pg1400))
Output: 10742.821054704795

trainingData_n = TrainingData(vocab, pg121, 2)
print(trainingData_n.perplexity(pd, pg141))
Output: 7214.313132768223

trainingData_n = TrainingData(vocab, pg121, hi_freq='True')
print(trainingData_n.perplexity(pd, pg1400))
Output: 26465.650651604607

trainingData_n = TrainingData(vocab, pg121, hi_freq='True')
print(trainingData_n.perplexity(pd, pg141))
Output: 20348.336600077062
```

1. One of the test files is by the same author as the training file but the other is not. Was the model successful in this classification task?

I don't know which file was written by the same author so it's hard to tell whether the model successfully classified the texts. What I can say is the model was able to differentiate between the texts. The model predicts pg141 was written by the same author as pg121.

2. Now repeat this evaluation but remove any word that appears less than 50 times in the training file (i.e., their count would be zero and the size of the file is similarly reduced). Was the model successful in this classification task with the reduced training file?

The model made the same prediction, prediction pg141 was written by the same author as pg121. It makes sense that the perplexity increased since there are more observations that the model has not seen in both texts. I'm not sure how to interpret the results relative to one another but I would have thought by removing words that occur infrequently would cause the difference in the perplexities to become smaller. This would make sense since the resulting documents only contain words that form the semantic basis of the language like 'a', 'the', or 'and' which would not differentiate the two authors.