```
%%bash
!(stat -t /usr/local/lib/*/dist-packages/google/colab > /dev/null 2>&1) && exit
rm -rf 6864-hw1
git clone https://github.com/lingo-mit/6864-hw1.git
```

```
import sys
sys.path.append("/content/6864-hw1")

import csv
import itertools as it
import numpy as np
np.random.seed(0)

import lab_util
```

Hidden Markov Models

In the remaining part of the lab (containing part 3) you'll use the Baum--Welch algorithm to learn *categ* vocabulary. Answers to questions in this lab should go in the same report as the initial release.

As before, we'll start by loading up a dataset:

```
data = []
n_positive = 0
n_disp = 0
with open("/content/6864-hw1/reviews.csv") as reader:
  csvreader = csv.reader(reader)
  next(csvreader)
  for id, review, label in csvreader:
    label = int(label)

    # hacky class balancing
    if label == 1:
      if n_positive == 2000:
        continue
      n_positive += 1
    if len(data) == 4000:
      break

    data.append((review, label))

    if n_disp > 5:
      continue
```



```
    print("rating:", label, "(good)" if label == 1 else "(bad)")
    print()
```

```
print(f"Read {len(data)} total reviews.")
np.random.shuffle(data)
reviews, labels = zip(*data)
train_reviews = reviews[:3000]
train_labels = labels[:3000]
val_reviews = reviews[3000:3500]
val_labels = labels[3000:3500]
test_reviews = reviews[3500:]
test_labels = labels[3500:]
```

Next, implement the forward--backward algorithm for HMMs like we saw in class.

IMPORTANT NOTE: if you directly multiply probabilities as shown on the class slides, you'll get underf the log domain (remember that `log(ab) = log(a) + log(b)`, `log(a+b) = logaddexp(a, b)`).

```
# hmm model
from scipy.special import logsumexp
import time

class HMM(object):
    def __init__(self, num_states, num_words):
        self.num_states = num_states
        self.num_words = num_words

        self.states = range(num_states)
        self.symbols = range(num_words)

        # initialize the matrix A with random transition probabilities p(j|i)
        # A should be a matrix of size `num_states x num_states`
        # with rows that sum to 1
        self.A = np.random.random(size=(num_states, num_states))
        self.A = np.exp(10 * self.A)
        self.A /= self.A.sum(axis=1, keepdims=True)
        self.A = np.log(self.A)

        # initialize the matrix B with random emission probabilities p(o|i)
        # B should be a matrix of size `num_states x num_words`
        # with rows that sum to 1
        self.B = np.random.random(size=(num_states, num_words))
        self.B = np.exp(10 * self.B)
        self.B /= self.B.sum(axis=1, keepdims=True)
        self.B = np.log(self.B)

        # initialize the vector pi with a random starting distribution
        # pi should be a vector of size `num_states`
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
        self.pi = np.log(self.pi)
```

```
    def generate(self, n):
        """randomly sample the HMM to generate a sequence.
        """
        # we'll give you this one

        sequence = []
        # initialize the first state
        state = np.random.choice(self.states, p=np.exp(self.pi))
        for i in range(n):
            # get the emission probs for this state
            b = np.exp(self.B[state, :])
            # emit a word
            word = np.random.choice(self.symbols, p=b)
            sequence.append(word)
            # get the transition probs for this state
            a = np.exp(self.A[state, :])
            # update the state
            state = np.random.choice(self.states, p=a)
        return sequence

    def forward(self, obs):
        # run the forward algorithm
        # this function should return a `len(obs) x num_states` matrix
        # where the (i, j)th entry contains p(obs[:t], hidden_state_t = i)

        alpha = np.zeros((len(obs), self.num_states))
        alpha[0, :] = self.pi + self.B[:, obs[0]]

        for t in range(1, len(obs)):
          for j in range(0, self.num_states):
            alpha[t, j] = logsumexp(alpha[t-1, :] + self.A[:, j]) + self.B[j, obs[t]]

        return alpha

    def backward(self, obs):
        # run the backward algorithm
        # this function should return a `len(obs) x num_states` matrix
        # where the (i, j)th entry contains p(obs[t+1:] | hidden_state_t = i)

        beta = np.zeros((len(obs), self.num_states))

        for t in range(len(obs)-2, -1, -1):
          for i in range(0, self.num_states):
            beta[t, i] = logsumexp(beta[t+1, :] + self.A[i, :] + self.B[:, obs[t+1]])

        return beta
```



```
        # logprob is the total log-probability of the sequence obs
        # (marginalizing over hidden states)
```

```
        # gamma is a matrix of size `len(obs) x num_states`
        # it contains the marginal probability of being in state i at time t

        # xi is a tensor of size `len(obs) x num_states x num_states`
        # it conains the marginal probability of transitioning from i to j at t

        alpha = self.forward(obs)
        beta = self.backward(obs)

        assert not np.any(np.isnan(alpha))
        assert not np.any(np.isnan(beta))

        logprob = logsumexp(alpha[-1, :])

        xi = np.zeros((len(obs)-1, self.num_states, self.num_states))
        for t in range(len(obs)-1):
          for i in range(self.num_states):
            for j in range(self.num_states):
              xi[t, i, j] = alpha[t, i] + self.A[i, j] + self.B[j, obs[t+1]] + beta[t+
        xi -= logprob

        gamma = alpha + beta - logprob
        assert not np.any(np.isnan(gamma))

        return logprob, np.exp(xi), np.exp(gamma)

    def learn_unsupervised(self, corpus, num_iters):
        """Run the Baum Welch EM algorithm
        """

        start = time.time()
        for i_iter in range(num_iters):
            expected_init = np.zeros(self.num_states)
            expected_si = np.zeros(self.num_states)
            expected_sij = np.zeros((self.num_states, self.num_states))
            expected_six = np.zeros(self.num_states)
            expected_siw = np.zeros((self.num_states, self.num_words))

            total_logprob = 0
            total_count = 0
            total_len = 0
            for i, review in enumerate(corpus):
                logprob, xi, gamma = self.forward_backward(review)
                total_logprob += logprob

                total_count += 1
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
                expected_si += gamma.sum(axis=0)
                expected_sij += xi.sum(axis=0)
```

```
                expected_six += gamma[:-1, :].sum(axis=0)
                for t in range(len(review)):
                  expected_siw[:, review[t]] += gamma[t, :]

                if (i+1)%100 == 0:
                    print(f"{i+1} of {len(corpus)}: {time.time()-start}")
                    start = time.time()

            pi_new = expected_init / total_count
            A_new = expected_sij / expected_six[:, np.newaxis]
            B_new = expected_siw / expected_si[:, np.newaxis]

            print("log-likelihood", total_logprob / len(corpus))

            self.A = np.log(A_new)
            self.B = np.log(B_new)
            self.pi = np.log(pi_new)
```

Train a model:

```
tokenizer = lab_util.Tokenizer()
tokenizer.fit(train_reviews)
train_reviews_tk = tokenizer.tokenize(train_reviews)
print(tokenizer.vocab_size)

hmm = HMM(num_states=25, num_words=tokenizer.vocab_size)
hmm.learn_unsupervised(train_reviews_tk[:50], 3)
```

Let's look at some of the words associated with each hidden state:

```
for i in range(hmm.num_states):
    most_probable = np.argsort(-hmm.B[i, :])[:20]
    print(f"state {i}")
    for o in most_probable:
        print(tokenizer.token_to_word[o], hmm.B[i, o])
    print()
```

We can also look at some samples from the model!

```
for i in range(10):
    print(tokenizer.de_tokenize([hmm.generate(10)]))
```

Finally, let's repeat the classification experiment from Parts 1 and 2, using the *vector of expected hidd*

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

(Warning: results may not be the same as in earlier versions of this experiment.)

```
def train_model(xs_featurized, xs):
```

```
def train_model(xs_featurized, ys):
  import sklearn.linear_model
  model = sklearn.linear_model.LogisticRegression()
  model.fit(xs_featurized, ys)
  return model

def eval_model(model, xs_featurized, ys):
  pred_ys = model.predict(xs_featurized)
  print("test accuracy", np.mean(pred_ys == ys))

def training_experiment(name, featurizer, n_train):
    print(f"{name} features, {n_train} examples")
    train_xs = np.array([
        hmm_featurizer(tokenizer.tokenize([review])[0])
        for review in train_reviews[:n_train]
    ])
    train_ys = train_labels[:n_train]
    test_xs = np.array([
        hmm_featurizer(tokenizer.tokenize([review])[0])
        for review in test_reviews
    ])
    test_ys = test_labels
    model = train_model(train_xs, train_ys)
    eval_model(model, test_xs, test_ys)
    print()

def hmm_featurizer(review):
    _, _, gamma = hmm.forward_backward(review)
    feature = gamma.sum(axis=0) / gamma.shape[0]
    return feature

training_experiment("hmm", hmm_featurizer, n_train=10)
training_experiment("hmm", hmm_featurizer, n_train=100)
training_experiment("hmm", hmm_featurizer, n_train=1000)
training_experiment("hmm", hmm_featurizer, n_train=3000)
```

Part 3: Lab writeup

1. What do the learned hidden states seem to encode when you run unsupervised HMM training with only 2 state

2 states:

Qualitative: State 0 seems to have a lot more grammatical terms (stopwords, punctuation, etc.) Meanw
adjectives. When generating sentences, the model does a good job of alternating between the "structu
sentences that were generated are still incoherent.

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

1. `<unk>` -2.4091815008882858

2. `.` -2.8297721496769115
3. `the` -3.1286337775175883
4. `,` -3.3649310921696953
5. `and` -3.554635548350083
6. `a` -3.617232645846918
7. `i` -3.674648654419042
8. `to` -3.7902573852238537
9. `it` -3.9070653492927088
10. `of` -4.086144159867106
11. `is` -4.177037938169066
12. `in` -4.42116388665908
13. `this` -4.635879636688612
14. `that` -4.6398808276511065
15. `but` -4.660972832186275
16. `not` -4.686384551312519
17. `!` -4.760692524573507
18. `are` -4.801460142032286
19. `my` -4.80266681955116
20. `was` -4.818420708739354

`state 1`

1. `.` -2.526359045757392
2. `br` -2.607606125209836
3. `i` -3.0006084941836595
4. `for` -3.010284292817512
5. `,` -3.3361257990920756
6. `this` -3.630407162677757
7. `<unk>` -3.8171019065221663
8. `taste` -3.841497131603046
9. `them` -3.868469819917019
10. `good` -3.9551603687179195
11. `product` -4.008901004204919
12. `like` -4.465775424111041
13. `of` -4.543269243566065
14. `so` -4.643898055949279
15. `these` -4.663801857982545
16. `don't` -4.779432574361799
17. `love` -4.915987823916717
18. `now` -4.964894748960646
19. `mix` -4.990125481251299

5 states:

Qualitative: State 0: `coffee`, `love`, `product`. State 1: `no`, `food`, `taste`. State 2: general stopwords a conjunctions and grammatical terms. State 4: `chips`, `potato`, `drink`, `sugar`, `price`, `better`, `best`

Generated sentence: `expect me to leave a you excellent about the my`

10 states:

Qualitative: stopwords and punctuation occur frequently in all of the states. It was difficult to determir semantic meaning than others. Examples of terms that occurred in the top 20 terms of one hidden sta `amazon`, `product`, `chips`.

Generated sentence: `advertised fructose replacement cups easy has as br was amazon`

2. As before, what's the relationship between # of labeled examples and usefulness of HMM-based sentence representations? Are these results generally better or worse than in Parts might HMM state distributions be sensible sentence representations?

There is generally an upward trend when adding more labeled examples. Although the hidden state dis and 5 hidden states, using 10 hidden states achieves similar and slightly better performance.

Overall, the learned HMM representations are not as good as representations learned in parts 1 and 2 First, we used a much smaller representation size (10 in HMMs vs 100 in word2vec). Second, the moc encoding elements of sentence structure in the hidden states (stopwords, nouns/adj, etc). This is less prediction, where the sentiment of individual words is strongly correlated with the sentiment of the rev

Ideally the HMM model would learn hidden states with word sentiment. Indeed, the improved perform states seems to indicate more of this focus on words. But we did not get any nice sentiment clusters time available to us.

2 states:

Accuracy (10 examples): **0.448**

Accuracy (100 examples): **0.476**

Accuracy (1000 examples): **0.516**

Accuracy (3000 examples): **0.548**

5 states:

Accuracy (10 examples): **0.476**

Accuracy (100 examples): **0.476**

Accuracy (1000 examples): **0.556**

10 states:

Accuracy (10 examples): **0.548**

Accuracy (100 examples): **0.476**

Accuracy (1000 examples): **0.508**

Accuracy (3000 examples): **0.570**