Problem Set 2: Discrete Distributions and Classification

Solutions

Name:

E-Mail:

There is a mathematical component and a programming component to this homework. Please submit a PDF export of this notebook Canvas. If a question requires you to make any plots, please include those in the writeup.

```
# Initialize notebook.
!pip install -qU plotly torch daft
!rm -fr start; git clone --single-branch -b demos2018 -q https://github.com/harvard-ml
import cs281
from plotly.offline import iplot
import torch
import daft
```

```
tcmalloc: large alloc 1073750016 bytes == 0x5630677fe000 @  0x7f0d85a252a4 0x56
```

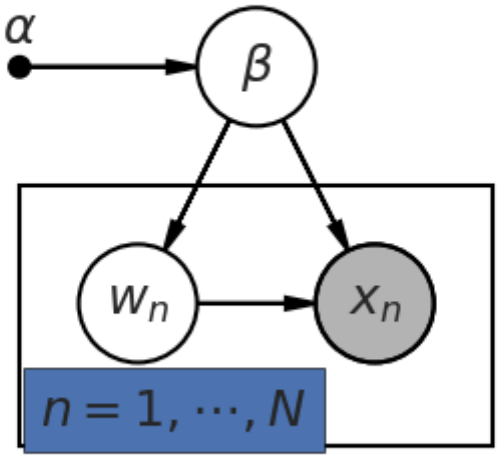
(Run to initialize math commands)

Prelude - Drawing Graphical Models

You can use the following library to draw graphical models in the notebook.

```
# Instantiate the PGM with 3 rows and 3 columns.
pgm = daft.PGM([3, 3])

# Deterministic parameter at column 0.5, row 2
pgm.add_node(daft.Node("alpha", r"$\alpha$", 0.5, 2, fixed=True))

# Latent parameter.
pgm.add_node(daft.Node("beta", r"$\beta$", 1.5, 2))

# Latent variable.
pgm.add_node(daft.Node("w", r"$w_n$", 1, 1))

# Data.
pgm.add_node(daft.Node("x", r"$x_n$", 2, 1, observed=True))

# Add in the edges.
pgm.add_edge("alpha", "beta")
```

```
pgm.add_edge("beta", "w")
pgm.add_edge("w", "x")
pgm.add_edge("beta", "x")

# And a plate.
pgm.add_plate(daft.Plate([0.5, 0.5, 2, 1], label=r"$n = 1, \cdots, N$",
    shift=-0.1))

# Render at a given resolution.
pgm.render()
pgm.figure.dpi = 150
```



Problem 1 - Warmup: The Dirichlet and Categorical Distributions

[12pts]

The Dirichlet distribution over K categories is a generalization of the beta distribution. It has a parameter $\alpha \in \mathbb{R}^K$ with non-negative entries and is supported over the set of K-dimensional positive vectors whose components sum to 1. Its density is given by

$$p(\theta_{1:K}|\alpha_{1:K}) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^K \theta_k^{\alpha_k - 1} \propto \prod_{k=1}^K \theta_k^{\alpha_k - 1}$$

Where the gamma function Γ is an extension of the factorials to non-integers and satisfies the property:

$$\Gamma(a) = (a-1)\Gamma(a-1)$$

(Notice that when $K = 2$, this reduces to the density of a beta distribution.) For the rest of this problem, assume a fixed $K \geq 2$.

1. Show that the Dirichlet is a member of the exponential family. State the key functional values: $u(\theta), g(\eta), h(\theta), \eta$.

$$
\begin{aligned}
p(\theta_{1:K}|\alpha_{1:K}) &= \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \\
&= \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \exp\left(\sum_{k=1}^{K} (\alpha_k - 1)\log\theta_k\right) \\
&= \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \exp\left(\begin{bmatrix} \alpha_1 - 1 \\ \alpha_2 - 1 \\ \vdots \\ \alpha_K - 1 \end{bmatrix}^T \begin{bmatrix} \log\theta_1 \\ \log\theta_2 \\ \vdots \\ \log\theta_K \end{bmatrix}\right)
\end{aligned}
$$

$$u(\theta) = \log(\theta)$$

$$\eta = \alpha - 1$$

$$g(\eta) = \frac{\Gamma\left(\sum_k (\eta_k + 1)\right)}{\prod_k \Gamma(\eta_k + 1)}$$

$$h(\theta) = 1$$

Check 1 (3pts): Show that the Dirichlet is exponential (1pt). Give the correct parameters (2pts).

2. Suppose θ is Dirichlet-distributed with shape parameter α. Derive the value of $\mathbb{E}(\theta_k)$.

$$
\begin{aligned}
\mathbb{E}(\theta_j) &= \int \theta_j \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} d\theta \\
&= \int \frac{\alpha_j}{\sum_k \alpha_k} \frac{\Gamma\left(\sum_k \alpha'_k\right)}{\prod_k \Gamma(\alpha'_k)} \prod_{k=1}^{K} \theta_k^{\alpha'_k - 1} d\theta \\
&= \frac{\alpha_j}{\sum_k \alpha_k} \int \mathrm{Dir}_{\alpha'}(\theta) d\theta \\
&= \frac{\alpha_j}{\sum_k \alpha_k}
\end{aligned}
$$

where $\alpha'_i = \alpha_j + 1$ when $i = j$ and $\alpha'_i = \alpha_i$ otherwise.

Check 2 (3pts): Derive the correct value for $\mathbb{E}(\theta_k)$.

3. Suppose that $\theta \sim \mathrm{Dir}(\alpha)$ and that $X \sim \mathrm{Cat}(\theta)$, where Cat is a Categorical distribution. That is, suppose we first sample a K-dimensional vector θ with entries in $(0, 1)$ from a Dirichlet distribution and then roll a K-sided die such that the probability of rolling the number k is θ_k. Prove that the posterior $p(\theta|X)$ also follows a Dirichlet distribution. What is its shape parameter?

$$
\begin{aligned}
p(\theta|X=j) &= \frac{p(X=j|\theta)p(\theta)}{p(X=j)} \\
&\propto p(X=j|\theta)p(\theta) \\
&\propto \theta_j \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \\
&= \prod_{k=1}^{K} \theta_k^{\alpha'_k - 1}
\end{aligned}
$$

Where $\alpha'_i = \alpha_j + 1$ when $i = j$ and $\alpha'_i = \alpha_i$ otherwise. α' is the posterior shape parameter.

Check 3 (3pts): Show that $p(theta|X)$ is Dirichlet for 1pt. Give the correct shape parameter for 3pts.

4. Now suppose that $\theta \sim \text{Dir}(\alpha)$ and that $x^{(1)}, x^{(2)}, \ldots \overset{iid}{\sim} \text{Cat}(\theta)$. Show that the posterior predictive after $n-1$ observations is given by,

   $$P(x^{(n)} = k|X^{(1)}, \ldots, x^{(n-1)}) = \frac{\alpha_k^{(n)}}{\sum_k \alpha_k^{(n)}}$$

   where for all k, $\alpha_k^{(n)} = \alpha_k + \sum_{i=1}^{n-1} \mathbf{1}\{x^{(i)} = k\}$.

First let's repeat the process from 3 for multiple i.i.d. variables.

$$
\begin{aligned}
P(\theta|X^{(1)}, \ldots X^{(n-1)}) &= \frac{P(X^{(1)}, \ldots, P(X^{(n-1)}|\theta)P(\theta)}{P(X^{(1)}, \ldots, P^{(n-1)})} \\
&\propto p(\theta) \prod_{i=1}^{n-1} P(X^{(i)}|\theta) \\
&\propto \prod_{j=1}^{K} \theta^{\alpha_j} \prod_{i=1}^{n-1} \prod_{j=1}^{K} \theta_j^{\mathbf{1}\{X^{(i)}=j\}}
\end{aligned}
$$

So $P(\theta|X^{(1)}, \ldots, X^{(n-1)})$ is Dirichlet with shape parameter

$$\alpha_j^{(n-1)} = \alpha_j + \sum_{i=1}^{n-1} \mathbf{1}\{X^{(i)} = j\}$$

Then:

$$
\begin{aligned}
P(x^{(n)} = k|X^{(1)}, \ldots, X^{(n-1)}) &= \int_\theta P(x^{(n)} = k|\theta) P(\theta|X^{(1)}, \ldots, X^{(n-1)}) d\theta \\
&= \int_\theta \theta_k \frac{\Gamma\left(\sum_k \alpha_k^{(n-1)}\right)}{\prod_k \Gamma(\alpha_k^{(n-1)})} \prod_{k=1}^{K} \theta_k^{\alpha_k^{(n-1)} - 1} d\theta
\end{aligned}
$$

which is the same form we had when taking the expectation. Thus

$$P(x^{(n)} = k|X^{(1)}, \ldots, X^{(n)}) = \frac{\alpha_k^{(n)}}{\sum_{i=1}^{n} \alpha_i^{(n)}}$$

5. Consider the random vector $Z_k = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{x^{(i)} = k\}$ for all k. What is the mean of this vector? What is the distribution of the vector? (If you're not sure how to rigorously talk about convergence of random variables, give an informal argument. Hint: what would you say if θ were fixed?) What is the marginal distribution of a single class $p(Z_k)$?

By the central limit theorem, the limit is normally distributed with mean

$$\begin{aligned} Z_k &= \mathbb{E}[\mathbf{1}\{X = k\}] \\ &= P(X = k) \\ &= \int_\theta P(X = k|\theta)P(\theta)d\theta \\ &= \int_\theta \theta_k P(\theta) \\ &= \mathbb{E}(\theta_k) \\ &= \frac{\alpha_k}{\sum_{i=1}^K \alpha_i} \end{aligned}$$

using the results from above.

Check 5 (1pt): Give and justify the correct mean.

Problem 2- Joint Distributions vs. Naive Bayes

[10pts]

This problem focuses on modeling a joint distribution of random variables, $p(y, x_1, \ldots, x_J)$, consisting of discrete variables. These variables represent a class label $y \in \{1, \ldots, C\}$ and features $x_1 \ldots, x_V$ each of each can take on a values $x_j \in \{0, 1\}$.

1. Let $J = 4$. Use the chain rule to select any valid factorization of this joint distribution into conditional univariate distributions. If each distribution is represented as a conditional probability table i.e. all values of $p(A = a|B = c, \ldots)$, what is the size of these tables? Would this value differ if you had chosen a different factorization?

$$P(y, x_1, x_2, x_3, x_4) = P(y|x_1, x_2, x_3, x_4)P(x_1|x_2, x_3, x_4)P(x_2|x_3, x_4)P(x_3|x_4)P(x_4)$$

Counting up the possibilities for the discrete variables we get:

$P(y|x_1, x_2, x_3, x_4)$ has a table with $C \cdot 2^4$ entries.

$P(x_1|x_2, x_3, x_4)$ has a table with 2^4 entries.

$P(x_2|x_3, x_4)$ has a table with 2^3 entries.

$P(x_3|x_4)$ has a table with 2^2 entries.

$P(x_4)$ has a table with 2 entries.

The particular sizes of each table would vary with a different factorization (e.g. we could isolate $P(y)$, which would have size C). However, the size of the table for the joint distribution is independent of the factorization.

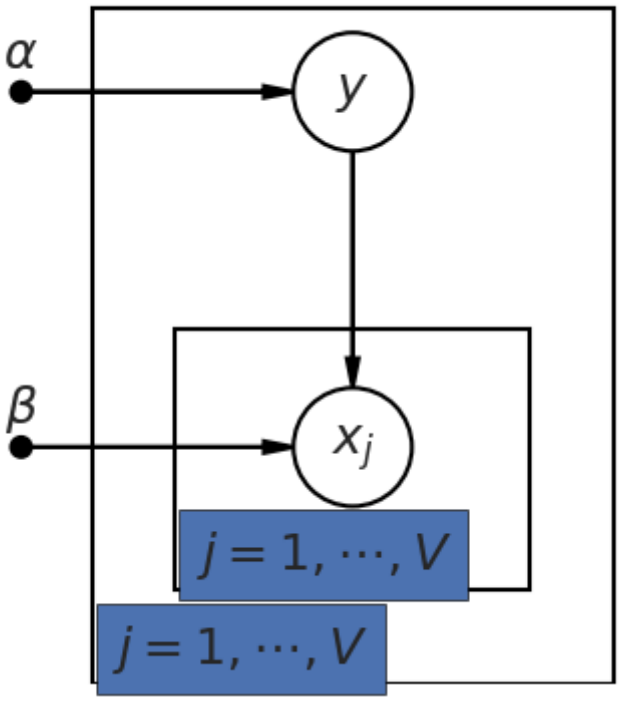
Check 1 (2pts): Give the correct table sizes (.5pt per correct value, up to 2pts max).

2. Now consider a naive Bayes factorization of this model, given by,

$$p(y, x_1, \ldots, x_V) \approx p(y) \prod_j p(x_j|y).$$

Draw a directed graphical model for this factorization. What is the size of the conditional probability tables required to fully express any factored distribution of this form?

```
def draw_dgm1():
    # Instantiate the PGM with 3 rows and 3 columns.
    pgm = daft.PGM([3, 3])

    # Fill in.
    pgm.add_node(daft.Node("y", r"$y$", 1.7, 2.5))
    pgm.add_node(daft.Node("x_j", r"$x_j$", 1.7, 1))

    pgm.add_node(daft.Node("alpha", r"$\alpha$", 0.3, 2.5, fixed=True))
    pgm.add_node(daft.Node("beta", r"$\beta$", 0.3, 1, fixed=True))

    pgm.add_edge("y", "x_j")
    pgm.add_edge("alpha", "y")
    pgm.add_edge("beta", "x_j")

    pgm.add_plate(daft.Plate([0.95, 0.5, 1.5, 1], label=r"$j = 1, \cdots, V$",
    shift=-0.1))
    pgm.add_plate(daft.Plate([0.6, 0.1, 2.2, 2.75], label=r"$j = 1, \cdots, V$",
    shift=-0.1))

    # Render at a given resolution.
    pgm.render()
    pgm.figure.dpi = 150
draw_dgm1()
```



Representing conditional distributions of models of this form requires $C \cdot 2^V$ table entries.

Check 2 (1.5pts): Draw the correct DGM (it's okay if you don't have a plate around y and the x_js)

3. In class, we parameterized naive Bayes such that the class distribution is Categorical with a Dirichlet prior, and the class-conditional distributions are Bernoulli with a Beta prior. Construct a function for sampling from this model using PyTorch distributions.

```
def make_parameters(C, V):
    cp = torch.rand(C) # Dirichlet params
    cc = torch.rand(2) # Beta params
    return dict(class_prior=cp, class_cond=cc, V=V)

def sample(parameters):
    "Create torch distributions and sample."
    C = parameters['class_prior'].size(0)
    V = parameters['V']

    cp = torch.distributions.dirichlet.Dirichlet(parameters['class_prior'])
    c = torch.distributions.categorical.Categorical(cp.sample())
    classes = c.sample(torch.Tensor(V).size()).view(-1, 1)

    ccp = torch.distributions.beta.Beta(parameters['class_cond'][0], parameters['class
    cc = torch.distributions.bernoulli.Bernoulli(ccp.sample(torch.Tensor(V, C).size())
    s = cc.sample()

    return torch.gather(s, 1, classes.expand(s.size()))[:, 0]


parameters = make_parameters(4, 12)
sample(parameters)
```

```
tensor([1., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 1.])
```

Check 3 (2pts): Give a function that samples variables in the correct order, and from the correct distributions.

4. Assuming the data obeys the naive Bayes assumptions, answer the following questions as true/false using your directed graphical model. Justify your answer.
    - For a given example, features x_1 and x_2 are independent.
    - The class labels y are always conditionally independent of the class-conditional parameters.
    - Upon observing the class distribution parameters, the class labels are conditionally independent.
    - Upon observing the class distribution parameters, the features are conditionally independent.
    - Upon observing the class distribution hyper-parameters, the class labels are conditionally independent.

- False: CI given class label
- False: independent so long as x_i is not observed
- True: the class labels are CI given that the parents are observed
- False: there are still 'active paths' between each feature through y
- False: there are still 'active paths' between each class labels through the class distribution parameters

Check 4 (3pts): .5pts per T/F question (up to 2.5pts) and .5pt for giving explanations for each

5. We will utilize naive Bayes for a problem where each example has a *bag* or multiset of features. A bag is a set that may contain multiple instances of the same value. One approach is to ignore this

property and use x_j as an indicator function for each item type. An alternative is to model x_j with sample space $\{0, \ldots, D\}$, where D is the maximum times an item appears and to use a Dirichlet-Categorical for the class-conditional.

- Compare and contrast these approaches in terms of their modeling power and required parameters.
- Propose a third option for modeling this distribution.

Option 2 requires more parameters but has higher modeling power than option 1.

Another option would be to use a multinomial distribution rather than a catergorical distribution in the setup of option 2, thereby allowing any number of occurences for any given item.

Check 5 (1.5pts): .5pt each for providing a benefit, a drawback, and an alternative. One benefit of the indicator function is that it overlooks outliers. One drawback is that it does not consider how often a word appears in the reviews.

Problem 3 - Hurdle Models for Count Data

[10pts]

In this problem we consider predictive models of count data. For instance given information about the student x, can we predict how often they went to the gym that week y? A natural choice is to use a Poisson Regression i.e. y conditioned on x is modeled as a Poisson distribution.

1. Show that a Poisson distribution $p(x; \mu) = \frac{\mu^x}{x!}\exp -\mu$ is a member of the exponential family. Confirm that its mean is μ using properties of the exponential family.

$$\begin{aligned} p(x; \mu) &= \frac{\mu^x}{x!}\exp(-\mu) \\ &= \frac{1}{x!}\exp\{x \log \mu - \mu\} \end{aligned}$$

By the properties of the exponential family:

$$\begin{aligned} \mathbb{E}[x] &= -\frac{\frac{d}{d\mu}(-\mu)}{\frac{d}{d\mu}\log \mu} \\ &= \mu \end{aligned}$$

Check 1 (1pt): Correctly show that the Poisson distribution is exponential.

2. Write out the model for linear Poisson regression, where the parameters of the Poisson are computed using a linear function of the data x. How do you ensure that the output can act as a Poisson parameter?

$$P(y|x) = \frac{1}{y!}\exp\{y \log(Ax + b) - Ax - b\}$$

In this case we need to maintain $Ax + b \geq 0$. We can do this by modeling $\log \mu$ instead:

$$P(y|x) = \frac{1}{y!}\exp\{y\,(Ax + b) - \exp\{Ax + b\}\}$$

Check 2 (2pts): Correctly model μ in terms of x (1pt). Correctly substitute into log space (1pt).

However, in practice, it is common for count data of this form to follow a bi-modal distribution over count data. For instance, our data may come from a survey asking students how often they went to the gym in the past week. Some would do so frequently, some would do it occasionally but not in the past week (a random zero), and a substantial percentage would never do so.

We may observe more zero examples than expected from our model. In the case of a Poisson, the mode of the distribution is the integer part of the mean. Our model may therefore be inadequate when means can be relatively large but the mode of the output is 0. Such data is common when many data entries have 0 outputs and many also have much larger outputs, so the mode of output is 0 but the overall mean is not near 0. This problem is known as *zero-inflation*.

This problem considers handling zero-inflation with a two-part model called a *hurdle model*. One part is a binary model such as a logistic model for whether the output is zero or positive. Conditional on a positive output, the "hurdle is crossed" and the second part uses a truncated model that modifies an ordinary distribution by conditioning on a positive output. This model can handle both zero inflation and zero deflation.

Suppose that the first part of the process is governed by probabilities $p(y > 0 \; ; \; x) = \pi$ and $p(y = 0 \; ; \; x) = 1 - \pi$; and the second part depends on $\{y \in \mathbb{Z} \mid y > 0\}$ and follows a probability mass function $f(y \; ; x)$ that is truncated-at-zero. The complete distribution is therefore:

$$p(y = 0 \mid x) = 1 - \pi$$
$$p(y = j \mid x) = \pi \frac{f(j \; ; \; x)}{1 - f(0 \; ; \; x)}, \; j = 1, 2, \ldots$$

One choice of parameterization is to use a logistic regression model for π:

$$\pi = \sigma(x^\top \mathbf{w}_1)$$

and use a Poisson for f with mean parameters μ :

$$\mu = \exp(x^\top \mathbf{w}_2)$$

3. Suppose we observe N data samples $\{(x_n, y_n)\}_{n=1}^N$. Write down the log-likelihood for the hurdle model assuming an unspecified mass function f. Give an maximum likelihood estimation approach for the free deterministic variables.

$$L \propto \prod_{n=1}^N (1 - \pi_n)^{\mathbf{1}\{y_n = 0\}} \prod_{n=1}^N \left(\pi_n \frac{f(y_j; x_j)}{1 - f(0; x_j)} \right)^{\mathbf{1}\{y_n = 1\}}$$
$$= \prod_{n=1}^N (1 - \sigma(x_n^T \mathbf{w}_1))^{\mathbf{1}\{y_n = 0\}} \prod_{n=1}^N \left(\sigma(x_n^T \mathbf{w}_1) \frac{f(y_j; x_j)}{1 - f(0; x_j)} \right)^{\mathbf{1}\{y_n = 1\}}$$

and the log-likelihood is

$$\log L = \sum_{n=1}^N \mathbf{1}\{y_n = 0\} \log\big(1 - \sigma(x_n^T \mathbf{w}_1)\big)$$
$$+ \sum_{n=1}^N \mathbf{1}\{y_n = 1\} \left(\log \sigma(x_n^T \mathbf{w}_1) + \log f(y_j; x_j) - \log(1 - f(0; x_j)) \right)$$
$$= \sum_{n=1}^N \mathbf{1}\{y_n = 1\} \left(x_n^T \mathbf{w}_1 + \log f(y_j; x_j) - \log\big(1 - f(0; x_j)\big) \right) - \sum_{n=1}^N \log\big(1 + \exp(x_n^T \mathbf{w}_1)\big)$$

Maximum likelihood is possible via SGD or another optimization technique.

Check 3 (2pts): Give the correct form of the log likelihood (1.5pts). GIve an approach to the MLE (.5pt).

4. Assume now that we select a Poisson distribution for f. Show that this truncated-at-zero Poisson distribution is a member of the exponential family. Give its the sufficient statistics and natural parameters.

$$f(j;\mu) = \frac{\mu^y}{y!}\exp(-\mu)$$
$$f(0;\mu) = \exp(-\mu)$$

$$\begin{aligned} p(y = k|y > 0) &= \frac{f(j;\mu)}{1 - f(0;\mu)} \\ &= \frac{\mu^y \exp(-\mu)}{y!(1 - \exp(-\mu))} \\ &= \frac{\mu^y}{y!(\exp \mu - 1)} \\ &= \frac{1}{y!}\exp\{y \log \mu - \log\{\exp \mu - 1\}\} \end{aligned}$$

The natural parameters are $\log \mu$. The sufficient statistics are y.

Check 4 (2pts): Correctly show that the truncated Poisson is exponential (1pt). Give the correct sufficient statistics and natural parameters (1pt).

5. What is the mean and variance of a truncated Poisson model with mean parameter μ? If we observe n i.i.d. samples from a truncated Poisson distribution, what is the maximum likelihood estimate of μ? (Note: Give an equation which could be solved numerically to obtain the MLE.)

$$\mathbb{E}[y] = \frac{\mu \exp \mu}{\exp \mu - 1}$$
$$\text{var}[y] = \frac{\mu + \mu^2}{1 - \exp -\mu} - \frac{\mu^2}{(1 - \exp -\mu)^2}$$

The MLE $\hat{\lambda}$ can be found by setting the derivative of the log-likelihood to zero. Alternatively, use the property of exponential families which ensures that the empirical mean of the sufficient statistics equals its expectation:

$$\frac{\hat{\lambda}}{1 - e^{-\hat{\lambda}}} = \bar{y}$$

Check 5 (2pts): Give the correct mean and variance (1pt). Describe a process for finding the MLE (1pt).

6. Write out the log-likelihood of this model in pytorch. Specify which are the parameters.

```
def hurdle_log_likelihood(x, y):
  pi = torch.sigmoid(torch.mm(torch.mm(x.t(), w1)))
  lbda = torch.exp(torch.mm(x.t(), fw2))
```

```
  if(y==0):
    return (1-pi).log()
  else:
    return pi.log()+(lbda.pow(y)/math.factorial(y.item())*torch.exp(-lbda)/(1-torch.ex
```

Check 6 (2pts): Give a correct expression of the log-likelihood in pytorch.

Problem 4: Naive Bayes Implementation

[10pts]

Despite its simplicity naive Bayes is widely used in practice. In this problem you will now implement a naive Bayes classifier for sentiment classification, a common problem in NLP. For this task you will use the IMDB Movie Reviews dataset which consists of positive and negative movie reviews .

Here are two example reviews:

Negative

- `there is no story! the plot is hopeless! a filmed based on a car with a stuck accelerator, no brakes, and a stuck automatic transmission gear lever cannot be good! ... i feel sorry for the actors ... poor script ... heavily over-dramatized ... this film was nothing but annoying, stay away from it!`

Positive

- `i had forgotten both how imaginative the images were, and how witty the movie ... anyone interested in politics or history will love the movie's offhand references - anyone interested in romance will be moved - this one is superb.`

As noted above, it is common to think of the input data as a bag/multiset. In text applications, sentences are often represented as a *bag-of-words*, containing how many times each word appears in the sentence. For example, consider two sentences:

- We like programming. We like food.
- We like CS281.

A vocabulary is constructed based on these two sentences:

```
["We", "like", "programming", "food", "CS281"]
```

Then the two sentences are represented as the index of each wordof each word in the vocabulary (starting from position 1).

```
!wget http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar xf aclImdb_v1.tar.gz
```

```
import glob, re
from collections import Counter
def collect(fs, max_len=500):
    y = []
    X = []
```

```
        for i, f in enumerate(fs):
            for f in glob.glob(f + "*.txt"):
                for l in open(f):
                    sent = [w.lower()
                                for w in re.split("[ ,.></]", l)]
                    if len(sent) < max_len:
                        X.append(sent)
                        y.append(i)
        return X, y

    train, train_label = collect(["aclImdb/train/neg/", "aclImdb/train/pos/"])
    test, test_label = collect(["aclImdb/test/neg/", "aclImdb/test/pos/"])
```

```
    def build_vocab(data, vocab_size=1000):
        counts = Counter([w for l in data for w in l ])
        vocab = {}
        rev_vocab = {}
        for i, (w, _) in enumerate(counts.most_common(vocab_size), 2):
            vocab[w] = i
            rev_vocab[i] = w
        rev_vocab[0] = "<PAD>"
        rev_vocab[1] = "<UNK>"
        return vocab, rev_vocab

    vocab, rev_vocab = build_vocab(train)
    print(len(vocab))
    print(" ".join([rev_vocab.get(vocab.get(w, 1)) for w in train[5]]))
```

```
    1000
        so don't even think about <UNK> this from the <UNK>  because this is one hell o
```

```
    # (Takes a couple minutes to run.)
    def vectorize(data, vocab):
        x_len = max([len(l) for l in data])
        X = torch.zeros(len(data), x_len).long()
        for i, sent in enumerate(data):
            for j, w in enumerate(sent):
                X[i, j] = vocab.get(w, 1)
        return X
    X, y = vectorize(train, vocab), torch.tensor(train_label)
    X_test, y_test = vectorize(test, vocab), torch.tensor(test_label)
```

```
    # Data Format
    print(" ".join([rev_vocab.get(x.item()) for x in X[0]]))
    print(X[0])
    print(y[0])
```

```
ok  i knew this would be a back <UNK> <UNK> <UNK> <UNK> <UNK> <UNK> going into
tensor([551,   2,  11, 667,  12,  57,  27,   4, 149,   1,   1,   1,   1,   1,
          1, 175,  93,  13,   2,  35,  11, 186,   2,   1,   2,  11,  96, 371,
          4,  48, 417,   2,  35, 988,  62,  47,   1, 582, 262,  87, 171,  76,
          2,  52,  11, 179,  96,  26,  81, 245,  31, 119, 713,   1,   1,   1,
          2,   1,   1, 135,   2,  51, 115, 968,   2,   5,  60, 109,   2,   9,
          2,   2,   2,   9,   2,   2,  18, 514,   2,  28, 141,  10, 886,  11,
        342,   1,  95,   1,   1,   1, 183, 104, 123,   1,  22,   1,   8, 125,
          3, 122, 279, 108, 657, 184, 342,  70,   1,   1,   2,   2,  73,  16,
          2,   1,  25,   1, 362,   2,  17,  36, 694, 113,   2,   5,   3,   1,
         15,   1,   6,   1,   5,   1,   9,   2,   2,   2,   9,   2,   2,   1,
          1,   1, 130,   2, 140,   1,   2,  23, 174,   7, 204,   1,   1,   1,
          2,  11,  67,   1,  18,   4, 144, 171,   1,   1, 130,   2,   3,   1,
        530,   1,   3,   1,   1,   1,   1,  25,  23,   1, 140,  24,  37,   1,
          1,   1, 551,   2, 302, 127,  13,   1, 171,   1,   1,  99, 171,   1,
          7, 204, 134,   1,   2,   1,   2,   1,   2,   5,   1,   1,   9,   2,
          2,   2,   9,   2,   2, 159,   2, 242,  59,  11,   1,   2, 199, 111,
         68,  24,  10,   3,   1,   1,   6, 368, 355,   1, 312,  93,   1,   2,
         99,  23, 137,   1,   2,   5,  13, 358,   1,  27,  10,   3,   1,   1,
         31,   1,   2,  82,  78,  66,   1, 140, 358,  24,  28,   6, 148,  79,
         39, 754,   1,   2,  19,  11,  63,  94,  13,  55,   1,   5, 358,  82,
        174,   7,  94,  13,  42, 140, 291,  18,   4,  16,  10,   3,   1,   1,
          2,  17,  12,  28,  11,   1,  15,  31,  28,   6,   3, 122,   1,  11,
        892, 392, 243,  31,   3,  87, 101,  42,  11, 407,   5,   1,   2,   9,
          2,   2,   2,   9,   2,   2, 217,  16, 117,   2,  60,  28, 133, 675,
         13,   2,   5,  13, 133,  63,  27,   1,  18,   4,   1,   1,   1,   2,
         42,  11,  96, 187,  13,   4,   1,   2,  11,  57,   2,  19,  11,  67,
         63, 187,  13,   4,   1,   6, 155,   2,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

For all of these problems implement the full Bayesian model where predictions come from the posterior predictive distribution.

- Implement a Naive Bayes classifier using a Bernoulli class-conditional with a Beta prior where each feature is an indicator that a word appears at least once in the bag.
- Implement a Naive Bayes classifier using a Categorical class-conditional with a Dirichlet prior. Here the features represent that count of each word in the bag.
- For both models, experiment with various settings for the priors. For the Dirichlet prior on the class, begin with $\boldsymbol{\alpha} = \mathbf{1}$. Keeping uniformity, vary the magnitude to $.5$ and smaller. Optionally, choose class-conditional priors based on an outside text source. Validate your choices on the validation set, and report accuracy on the test set.
- (Optional) With the bag-of-words representation, would the model be able to capture phrases like "don't like"? An alternative to the bag-of-words model is known as the bag-of-bigrams model, where a bigram is two consecutive words in a sentence. See *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification*\ http://www.aclweb.org/anthology/P/P12/P12-2.pdf#page=118\

```
# reformat data

def one_hot(X,rev_vocab):
  one_hot = torch.zeros((X.shape[0],len(rev_vocab)))
  mapping = torch.eye(len(rev_vocab))
```

```
    for i in range(X.shape[0]):
      one_hot[i,:] = (mapping[X[i]].sum(0)>0)
    return one_hot

def cat_hot(X,rev_vocab):
    cat_hot = torch.zeros((X.shape[0],len(rev_vocab)))
    mapping = torch.eye(len(rev_vocab))
    for i in range(X.shape[0]):
      temp = mapping[X[i]].sum(0)
      temp[temp>=4] = 4
      cat_hot[i,:] = temp
    return cat_hot
```

```
X_onehot = one_hot(X, rev_vocab)
X_test_onehot = one_hot(X_test, rev_vocab)
X_categorical = cat_hot(X, rev_vocab)
X_test_categorical = cat_hot(X_test, rev_vocab)
```

```
import torch.distributions as ds

class NBBernoulli:
    def __init__(self, X, y,b0=1.,b1=1.):
      self.w = len(rev_vocab)
      self.N = X.size(0)

      self.X = X.float()
      self.y = y.float().view((1, self.N))
      self.Nc = (self.y==1).sum().float()

      self.pi = self.Nc/float(self.N)
      self.theta_pos = torch.zeros(self.w)
      self.theta_neg = torch.zeros(self.w)

      for i in range(self.w):
        self.theta_pos[i] = (torch.mm(self.y, self.X[:,i].view(-1,1))+b1)/(self.Nc+b0+b1
        self.theta_neg[i] = (torch.mm((1-self.y), self.X[:,i].view(-1, 1))+b1)/(self.N-s

    def predict(self, X):
      log_pos = self.theta_pos.log().view(-1, 1)
      log_neg = self.theta_neg.log().view(-1, 1)

      log_pos_not = (1-self.theta_pos).log().view(-1, 1)
      log_neg_not = (1-self.theta_neg).log().view(-1, 1)
      return ((torch.mm(X, log_pos) + torch.mm(1-X, log_pos_not) - torch.mm(X, log_neg)
               + self.pi.log() + (1-self.pi).log())>0).float()

    def accuracy(self, X, y):
      return (self.predict(X).view(-1)==y.float()).sum().item()/float(y.size(0))
```

```
class NBCategorical:
    # one-hot
    def one_hot(self,X,i,j):
      return (X[:,i]==j).float()
    def one_hot2(self, X, j):
      return (X==j).float()

    def __init__(self, X, y, b=torch.Tensor([1., 1., 1., 1., 1.]), C=5):
      self.w = len(rev_vocab)
      self.N = X.shape[0]
```

```
    self.y = y.float().view((1,self.N))

    self.Nc = (self.y==1).sum().float()
    self.X = X.float()

    self.C = C

    self.pi = self.Nc/float(self.N)
    self.theta_pos = torch.zeros((self.C,self.w))
    self.theta_neg = torch.zeros((self.C,self.w))

    for i in range(self.w):
      for j in range(self.C):
        self.theta_pos[j,i] = (torch.mm(self.y, self.one_hot(X,i,j).view(-1, 1))+b[j])
        self.theta_neg[j,i] = (torch.mm((1-self.y), self.one_hot(X,i,j).view(-1, 1))+b

  def predict(self, X):
    probs = torch.zeros(X.shape[0])
    to_add = self.pi.log() - (1-self.pi).log()
    for i in range(self.C):
      probs += torch.mm(self.one_hot2(X,i), self.theta_pos[i,:].log().view(-1, 1)).vie
      probs -= torch.mm(self.one_hot2(X,i), self.theta_neg[i,:].log().view(-1, 1)).vie
    return ((to_add + probs)>0).float()

  def accuracy(self, X, y):
    return (self.predict(X).view(-1)==y.float()).sum().item()/float(y.size(0))
```

```
nbb = NBBernoulli(X_onehot,y,b0=1.,b1=1.)
print(nbb.accuracy(X_test_onehot,y_test))
nbb = NBBernoulli(X_onehot,y,b0=200.,b1=200.)
print(nbb.accuracy(X_test_onehot,y_test))
nbb = NBBernoulli(X_onehot,y,b0=0.5,b1=0.5)
print(nbb.accuracy(X_test_onehot,y_test))
```

```
nbc = NBCategorical(X_categorical, y)
print(nbc.accuracy(X_test_categorical,y_test))
nbc = NBCategorical(X_categorical, y, C = 10, b=torch.tensor([100.]*10))
print(nbc.accuracy(X_test_categorical,y_test))
nbc = NBCategorical(X_categorical, y, C = 3, b=torch.tensor([.1, 100, 1000]))
print(nbc.accuracy(X_test_categorical,y_test))
```

Check 1 (4pts): Implement with Bernoulli class-conditional with Beta prior

Let C by the number of classes, indexed by k. Let V be the size of the vocabulary, indexed by j. Let N be the number of reviews. Let N_{jk} be the number of words in class k that had word j present. Let N_k be the number of reviews of class k. Let $\text{Beta}(\alpha, \beta) = \text{Beta}(\beta_1, \beta_0)$. Let the Bernoulli class conditional distribution have a Beta prior parameterized by (β_1, β_0). Let the categorical class distribution have a Dirichlet prior parameterized by $(\alpha_1, \ldots, \alpha_C)$.

You used an estimate for the class conditional distribution of the form

$$\theta_{kj} = \frac{N_{jk} + \beta_1}{N_k + \beta_0 + \beta_1}$$

by using the posterior mean, or

$$\theta_{kj} = \frac{N_{jk} + \beta_1 - 1}{N_k + \beta_0 + \beta_1 - 2}$$

by using the posterior mode.

You used an estimate for the class distribution of the form

$$\pi_k = \frac{N_k + \alpha_k}{N + \sum_{k'} \alpha_{k'}}$$

by using the posterior mean, or

$$\pi_k = \frac{N_k + \alpha_k - 1}{N + \sum_{k'} \alpha_{k'} - C}$$

by using the posterior mode.

Check 2 (4pts): Implement with Categorical class-conditional with Dirichlet prior

Here, let θ_{kj} be a catrogrical distribution over counts for class k and word j. Let Q be the the maximum number of times any word appears in any review of any class. Let counts be indexed by m. Let N_{kjm} be the number of reviews of class k where word j appeared m times. Let the categorical class conditional have a Dirichlet prior parameterized by $(\gamma_1, \ldots, \gamma_Q)$. You used an estimate for the class conditional distribution of the form

$$\theta_{kjm} = \frac{N_{kjm} + \gamma_m}{N_k + \sum_{m'} \gamma_{m'}}$$

by using the posterior mean, or

$$\theta_{kjm} = \frac{N_{kjm} + \gamma_m - 1}{N_k + \sum_{m'} \gamma_{m'} - Q}$$

by using the posterior mode.

Check 3 (2pts): Explored various settings for the priors.

You probably found that using priors of magnitude 1 and below worked better than anything higher. It may have thrown off your data counts to use priors above 1. The larger magnitude you use with a uniform prior, the more "washed away" your count distributions get. Using a non-uniform prior on the class distribution would be useful if you expected to find a non-unform distribution over classes in the real world (test set) and your training data had an even proportion of classes (which it did).

Accuracies on test should be above 81%, for both Bernoulli and Categorical Naive Bayes.

Problem 5: Logistic Regression with Autograd

[15pts]

In the previous problem, you implemented a Naive Bayes classifier for sentiment classification on the IMDB Movie Reviews dataset. In this problem, you will apply logistic regression to the same task.

1. ℓ_1-regularized logistic regression. Consider a model parameterized by w:

$$p(w) = \frac{1}{2b}\exp(-\frac{\|w\|_1}{b})$$
$$p(y = 1|w; x) = \sigma(w^\top \phi(x))$$
$$p(y = 0|w; x) = 1 - \sigma(w^\top \phi(x))$$

   where $\sigma(\cdot)$ is the sigmoid function. Note that we are imposing a Laplacian prior on w, see Murphy, 2.4.4.

   - Given a dataset $\{(x^{(n)}, y^{(n)})\}_{n=1}^N$, derive the necessary gradient updates for MAP of w.

Note: You only need to consider the case where $\forall j, w_j \neq 0$. (If it is bothering you that there is an issue with $\exists j, w_j = 0$, note that you can use subgradients instead.)

- Show that for some constant λ, MAP inference of w is equivalent to minimizing

$$-\frac{1}{N}\sum_{n=1}^{N}\log p(y^{(n)}|w; x^{(n)}) + \lambda\|w\|_1$$

(a) To find the necessary gradient update:

$$\begin{aligned} p(\mathbf{w}|y,\mathbf{x}) &\propto p(y|\mathbf{x},\mathbf{w})p(\mathbf{w}) \\ \log p(\mathbf{w}|y,\mathbf{x}) &= \log p(\mathbf{w}) + \log p(y|\mathbf{x},\mathbf{w}) + C \\ &= -\frac{\|\mathbf{w}\|_1}{b} + \sum_{i=1}^{N} y^{(i)}\log\sigma(\mathbf{w}^\top\mathbf{x}^{(i)}) + (1-y^{(i)})\log(1-\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})) + C \end{aligned}$$

$$\frac{\partial \log p(\mathbf{w}\mid y,\mathbf{x})}{\partial \mathbf{w}} = -\frac{1}{b}\partial\|\mathbf{w}\|_1 + \sum_{i=1}^{N} y^{(i)}\frac{\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})(1-\sigma(\mathbf{w}^\top\mathbf{x}^{(i)}))}{\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})}\mathbf{x}^{(i)} - (1-y^{(i)})\frac{\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})(1-\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})}{1-\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})}$$

Where $\partial\|\mathbf{w}\|_1$ is the subgradient of $\|\mathbf{w}\|_1$\footnote{\url{https://see.stanford.edu/materials/lsocoee364b/01-subgradients_notes.pdf}, page 5}, which is generally a set. In this problem set we only consider the case where $\forall i\ (w_i \neq 0)$, so $\partial\|\mathbf{w}\|_1$ is a single vector $\text{sign}(\mathbf{w})$, and the gradient is:

$$\begin{aligned} &= -\frac{1}{b}\text{sign}(\mathbf{w}) + \sum_{i=1}^{N} y^{(i)}(1-\sigma(\mathbf{w}^\top\mathbf{x}^{(i)}))\mathbf{x}^{(i)} - (1-y^{(i)})\sigma(\mathbf{w}^\top\mathbf{x}^{(i)})\mathbf{x}^{(i)} \\ &= -\frac{1}{b}\text{sign}(\mathbf{w}) + \sum_{i=1}^{N}(y^{(i)} - \sigma(\mathbf{w}^\top\mathbf{x}^{(i)}))\mathbf{x}^{(i)} \end{aligned}$$

Check 1 (1pt): Derived the correct gradient update for MAP

(b)

$$\begin{aligned} \mathbf{w}_{\text{MAP}} &= \arg\max \log p(\mathbf{w}\mid y,\mathbf{x}) \\ &= \arg\max \log p(y\mid\mathbf{w},\mathbf{x}) + \log p(\mathbf{w}) \\ &= \arg\max \sum_{i=1}^{N}\log p(y^{(i)}\mid\mathbf{x}^{(i)},\mathbf{w}) + \log p(\mathbf{w}) \\ &= \arg\min -\frac{1}{N}\sum_{i=1}^{N}\log p(y^{(i)}\mid\mathbf{x}^{(i)},\mathbf{w}) - \log\frac{1}{2b} + \frac{\|\mathbf{w}\|_1}{b} \\ &= \arg\min -\frac{1}{N}\sum_{i=1}^{N}\log p(y^{(i)}\mid\mathbf{x}^{(i)},\mathbf{w}) + \lambda\|\mathbf{w}\|_1, \quad \lambda = \frac{1}{b} \end{aligned}$$

Check 2 (1pt): Show that for some λ, MAP of w is equivalent to minimizing the above loss

2. Implementation

1. Using the bernoulli bag-of-words feature representation from the previous question, i.e. $\phi(x)$ is a binary vector the size of the vocabulary indicating whether each word was used, train a logistic regression model using PyTorch autograd. Report test accuracy. Select regularization strength λ based on validation accuracy.
2. Which 5 words correspond to the largest weight indices, per class, in the learnt weight vectors? Which 5 words correspond to the least weight indices?
3. Study how sparsity (i.e percentage of zero elements in a vector) of the parameter vector changes with different values of λ. Again, tune λ on the validation set and report the test accuracies on the test set. Suggested values to try are $\{0, 0.001, 0.01, 0.1, 1\}$. You can treat parameters with $< 1e-4$ absolute values as zeros.

```
import torch.nn as nn

class LogRegr:
  def __init__(self, X, y, lbda = 0.1):
    self.X = (X>0).float()
    self.y = y.float()

    # sizes
    self.J = self.X.size(1)
    self.N = self.X.size(0)

    # parameters
    self.lbda = lbda
    self.w = torch.randn((self.J,1), requires_grad=True)

  def nllp(self):
    eps = 0.000001
    self.opt.zero_grad()
    y = self.y.view((self.N,1))

    reg = self.w.norm(1)*self.lbda
    likelihood = -1/self.N*(torch.mm(y.t(), torch.log(eps+torch.sigmoid(torch.mm(self.
                          + torch.mm((1-y).t(), torch.log(eps+1-torch.sigmoid(torch.
    nllp = reg + likelihood
    nllp.backward()
    return nllp

  def train(self,steps=50):
    self.opt = torch.optim.LBFGS([self.w],lr=0.01)
    for i in range(steps):
      self.opt.step(self.nllp)

  def accuracy(self, X_test, y):
    correct_num = ((torch.sigmoid(torch.mm(X_test, self.w))>0.5).long()==y.view((len(y
    return correct_num.item()/float(y.size(0))
```

```
for i in [0,0.0005,0.001,0.01,0.1,1.]:
  lr = LogRegr(X_categorical,y,lbda=i)
  lr.train()
  print("Lambda: %f; Accuracy: %f" % (i,lr.accuracy(X_test_cat,y_test)))
  print("Sparsity: %f" % ((lr.w<0.0001).sum().item()/float(lr.w.size(0))))
```

Check 3 (5pts): Implemented LR model and get at least 85% test accuracy

Check 4 (1pt): Provide reasonable positive words and negative words

For example, positive: great, excellent, loved, facorite, wonderful and negative: worst, waste, awful, boring.

Check 5 (2pts): Try various values of λ. Sparsity increases with λ

Problem 6

[Neural Networks, 5pts]

In the previous problem, we have implemented a Logistic Regression classifier using PyTorch. Logistic Regression can be seen as a 1-layer neural network. Implementing a multi-layer neural network only requires incremental change to our logistic regression implementation.

1. Implement a multi-layer neural network for IMDB classification and report accuracy on the test set. You are free to design the network structure (number of hidden units, activation function) and choose the optimization methods (SGD or ADAM, regularization or not, etc.).
2. Implement a model with a pretrained first layer. In the example below we show you how to download weights from the web for a given layer. This approach, known as *pretrained embeddings*, is a very easy and common way to transfer features between models. Here we are using Glove embeddings
3. (Optional) Implement a model using Convolutional Neural Networks. One example is Yoon Kim (2014) *Convolution Neural Networks for Sentence Classification*.

```
# Takes a couple minutes to download.
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
!head glove.6B.50d.txt
```

```
the 0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 -0.17862 -0.000660
, 0.013441 0.23682 -0.16899 0.40951 0.63812 0.47709 -0.42852 -0.55641 -0.364 -0
. 0.15164 0.30177 -0.16763 0.17684 0.31719 0.33973 -0.43478 -0.31086 -0.44999 -
of 0.70853 0.57088 -0.4716 0.18048 0.54449 0.72603 0.18157 -0.52393 0.10381 -0.
to 0.68047 -0.039263 0.30186 -0.17792 0.42962 0.032246 -0.41376 0.13228 -0.2984
and 0.26818 0.14346 -0.27877 0.016257 0.11384 0.69923 -0.51332 -0.47368 -0.3307
in 0.33042 0.24995 -0.60874 0.10923 0.036372 0.151 -0.55083 -0.074239 -0.092307
a 0.21705 0.46515 -0.46757 0.10082 1.0135 0.74845 -0.53104 -0.26256 0.16812 0.1
" 0.25769 0.45629 -0.76974 -0.37679 0.59272 -0.063527 0.20545 -0.57385 -0.29009
's 0.23727 0.40478 -0.20547 0.58805 0.65533 0.32867 -0.81964 -0.23236 0.27428 0
```

```
import torch.utils.data

class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.linear1 = nn.Linear(1002,1)
    self.sigmoid2 = nn.Sigmoid()

  def forward(self, x):
    x = self.linear1(x)
    x = self.sigmoid2(x)
    return torch.cat([x,1-x],dim=1)
```

```
class MovieData(torch.utils.data.Dataset):
  def __init__(self,x,y=None):
    self.x = x
    self.y = y

  def __len__(self):
    return len(self.x)

  def __getitem__(self,idx):
    return self.x[idx], self.y[idx]
```

```
BATCHSIZE = 10
net = Net()

train = MovieData(X_cat, y=y)
trainloader = torch.utils.data.DataLoader(train, batch_size=BATCHSIZE, shuffle=True, r
optimizer = torch.optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
criterion = nn.CrossEntropyLoss()
```

```
for epoch in range(8): # loop over the dataset multiple times
  running_loss = 0.0
  for i, data in enumerate(trainloader,0):
    # get the inputs
    inputs, labels = data
    # zero the parameter gradients
    optimizer.zero_grad()
    # forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs.squeeze(), labels)
    loss.backward()
    optimizer.step()
    # print statistics
    running_loss += loss.item()
    if i % 1000 == 999: # print every 2000 mini•batches
      print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss / 1000))
      running_loss = 0.0
print('Finished Training')

print("Accuracy: ", ((net(X_test_cat)[:,1]>0.5).type(torch.FloatTensor)==y_test.float(
```

```
pretrained = torch.zeros(1002, 50).float()
for l in open("glove.6B.50d.txt"):
  t = l.strip().split()
  if t[0] in vocab:
    pretrained[vocab[t[0]], :] = torch.tensor(list(map(float, t[1:])))
```

```
# Assign these values to a layer
pretrained_layer = nn.Linear(1002, 50)
pretrained_layer.weight.data = pretrained.transpose(0, 1)
```

```
class PretrainedNet(nn.Module):
  def __init__(self, p_layer):
    super(PretrainedNet, self).__init__()

    # import glove data
    self.glove = p_layer

    self.linear1 = nn.Linear(50,1)
    self.sigmoid2 = nn.Sigmoid()

  def forward(self, x):
    x = self.glove(x)
    x = self.linear1(x)
    x = self.sigmoid2(x)
    return torch.cat([x,1-x],dim=1)
```

```
BATCHSIZE=10
glovenet = PretrainedNet(pretrained_layer)
train = MovieData(X_onehot, y=y)
trainloader = torch.utils.data.DataLoader(train, batch_size=BATCHSIZE, shuffle=True, r
optimizer = torch.optim.SGD(glovenet.parameters(), lr=0.01, momentum=0.9)
criterion = nn.CrossEntropyLoss()
for epoch in range(8): # loop over the dataset multiple times
  running_loss = 0.0
  for i, data in enumerate(trainloader,0):
    # get the inputs
    inputs, labels = data
    # zero the parameter gradients
    optimizer.zero_grad()
    # forward + backward + optimize
    outputs = glovenet(inputs)
    loss = criterion(outputs.squeeze(), labels)
    loss.backward()
    optimizer.step()
    # print statistics
    running_loss += loss.item()
    if i % 1000 == 999: # print every 2000 mini•batches
      print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss / 1000))
      running_loss = 0.0
print('Finished Training')

print("Accuracy: ", ((glovenet(X_test_onehot)[:,1]>0.5).float()==y_test.float()).sum()
```

```
[1,  1000] loss: 0.826
[1,  2000] loss: 0.809
[2,  1000] loss: 0.816
[2,  2000] loss: 0.820
[3,  1000] loss: 0.823
[3,  2000] loss: 0.819
[4,  1000] loss: 0.820
[4,  2000] loss: 0.813
[5,  1000] loss: 0.819
[5,  2000] loss: 0.821
[6,  1000] loss: 0.809
[6,  2000] loss: 0.827
[7,  1000] loss: 0.821
[7,  2000] loss: 0.814
[8,  1000] loss: 0.816
[8,  2000] loss: 0.816
Finished Training
```

Check 1 (4pts): Implemented multi-layer neural network with at least 80% test accuracy

Check 2 (1pt): Used pre-trained glove embeddings