Step 1: Compute the positive label condition: P(L+|Φ₀:v) = P(Φ₀:v|L+)P(L+)/P(Φ₀:v₁)

```

def prob\_pos\_given\_features(x\_train, y\_train):

log\_probs = np.array([0] \* x\_train.shape[1])

#YOUR CODE HERE

raise NotImplementedError()

return log\_probs

```

Function goal: calculate the posterior probability at the feature-level

1. Count the number of positive instances for each feature.

2. Calculate the likelihood probability P(phi\_0 | L\_+), P(phi\_1 | L\_+), ..., P(phi\_v | L\_+).

3. Calculate the prior P(L\_+).

4. Calculate the evidence P(phi\_0:v).

5. Calculate the posterior probability P(L\_+ | phi\_0), P(L\_+ | phi\_1), P(L\_+ | phi\_n).

6. All the probabilities are in log scale, and operations are vectorized.

Absolutely! Here's the text from the image:

**Naive Bayes**

Step 2: Compute the negative label condition: P(L\_|Φ₀:v) = P(Φ₀:v|L\_)P(L\_)/P(Φ₀:v₁)

def prob\_neg\_given\_features(x\_train, y\_train):

log\_probs = np.array([0] \* x\_train.shape[1])

#YOUR CODE HERE

raise NotImplementedError()

return log\_probs

Function goal: calculate the posterior probability at the feature-level

1. Count the number of negative instances for each feature.
2. Calculate the likelihood probability P(phi\_0 | L\_-), P(phi\_1 | L\_-), ..., P(phi\_v | L\_-).
3. Calculate the prior P(L\_-).
4. Calculate the evidence P(phi\_0:v).
5. Calculate the posterior probability P(L\_- | phi\_0), P(L\_- | phi\_1), P(L\_- | phi\_n).
6. All the probabilities are in log scale, and operations are vectorized.

**Naive Bayes**

Step 3: Make a label prediction. Subtract (in log scale) the positive from the negative. If the result is greater than zero then it is a prediction of + label.1

def naive\_bayes(x, pos\_probs, neg\_probs):

label = 0

#YOUR CODE HERE

raise NotImplementedError()

return label

Function goal: predict the label (positive/negative) for each input x

1. Calculate the posterior probability for positive label P(L\_+ | phi\_0:v)
2. Calculate the posterior probability for negative label P(L\_- | phi\_0:v)
3. Compare two probabilities and predict the label
4. All the probabilities are in log scale, and operations are vectorized
5. Use the feature-level posterior probabilities calculated in previous functions

**Logistic Regression - Part 1**

#Defining neural network structure

class BowClassifier(nn.Module): #inheriting from nn.Module!

def \_\_init\_\_(self, num\_labels, vocab\_size):

super(BowClassifier, self).\_\_init\_\_()

raise NotImplementedError() #YOUR CODE HERE

def forward(self, bow\_vec):

#YOUR CODE HERE

raise NotImplementedError()

return out

Function goal (**init**): initialize the parameters, used in neural network

1. Initialize the linear layer (be careful about the size)

Function goal (forward): produce the probability of the input example being positive

1. Apply linear and necessary non-linear layers

**Logistic Regression - Part 2**

def get\_batch(i, batch\_size, x\_data, y\_data):

#Make some empty tensors

x = torch.zeros([batch\_size, x\_data.shape[1]])

y = torch.zeros((batch\_size, 1])

#YOUR CODE HERE

raise NotImplementedError()

return x, y

Function goal: prepare input x and y for batch compute

1. Slice the x and y
2. Make sure the output shapes are correct (x: batch x vocab, y: batch)
3. Make sure output x and y are torch.float tensors

**Multinomial Regression**

#Defining neural network structure

class MultinomialBowClassifier(nn.Module): #inheriting from nn.Module!

def \_\_init\_\_(self, max\_word\_len, embedding\_dim, num\_labels):

super(MultinomialBowClassifier, self).\_\_init\_\_()

self.max\_word\_len = max\_word\_len

self.embedding\_dim = embedding\_dim

self.num\_labels = num\_labels

#YOUR CODE HERE

raise NotImplementedError()

def forward(self, x):

out = None

#YOUR CODE HERE

raise NotImplementedError()

return out

Function goal (**init**): initialize the parameters, used in neural network

1. Initialize the linear, non-linear, or dropout layers you will use

2. We allow more than 1 linear layer

Function goal (forward): produce the probabilities of the input example with diff. labels

1. Chain all the used layers together