Department of Computing

Imperial College of Science, Technology and Medicine

NLP and Classificaiton

Intro to how one would develop classification algorithms for natural language processing

Author: Anton Zhitomirsky

Contents

1 Classification

1.1 NLP Classification tasks

Definition 1.1 (Classification)**.**

$$\hat{y} = \arg\max_y P(y|x) \tag{1}$$

Predicting which 'class' an observation belongs to

A Model produces a score (logit), a sigmoid makes this between 0 and 1, and 0.5 is our decision boundary. In multi-class classification we then use softmax.

1.1.1 Natural Language Inference

a model is presented with a pair of sentences and must classify the relationship between their meanings. For example in the MultiNLI corpus, pairs of sentences are given one of 3 labels: entails, contradicts and neutral. These labels describe a relationship between the meaning of the first sentence (the premise) and the meaning of the second sentence (the hypothesis) [1]. Here are representative examples of each class from the corpus:

- **Premise**: The kitten is climbing the curtains again
- **Hypothesis**: The kitten is sleeping
- **Entailment**: If the hypothesis is implied by the premise
- **Contradiction**: If the hypothesis contradicts the premise (in this example, the hypothesis is contradicting)
- **Neutral**: otherwise (neither is necessarily true)

1.2 Naive Bayes

(Generative Algorithm)

Definition 1.2 (Bayes Rule)**.**

$$\underbrace{P(y|x)}_{\text{Posterior}} = \frac{\overbrace{P(x|y)}^{\text{Likelihood}}\overbrace{P(y)}^{\text{Prior}}}{\underbrace{P(x)}_{\text{Evidence}}}$$

Since $P(x)$ won't change for different classes

> ☞ why? - because I believe it is the training data. The training data in a model won't change unless it is retrained, in which case the probabilities will change but the classification outcome shouldn't change because for each x we are always dividing by the same $P(x)$.

$$\hat{y} = \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$$

We can further make an assumption that x is a set of features $x_1, \ldots, x_I$ that are independent:

Definition 1.3 (Naiive Bayes Classifier)**.**

$$\hat{y} = \arg\max_y \overbrace{P(x_1, \ldots, x_I|y)}^{P(x_1|y)\cdot\ldots\cdot P(x_I|y)} P(y) = \arg\max_y P(y) \prod_{i=1}^{I} P(x_i|y)$$

1.2.1 Bag of Words Input Representations

Raw input is transformed into a numerical representation - i.e. each input x is represented by a feature vector by using a Bag of Words approach (count of each word in the input) "This was another good movie for holiday watchers. There was a nice little twist at the end"

Collect statistics from our training data (find what $P(y|x)$ is) after performing limited data-preprocessing.

Training corpus	good	movie	bad	class
another good movie for holiday watchers . a little twist from the ordinary scrooge movie . enjoyable .	1	1	0	+
it seems like just about everybody has made a christmas carol movie . others are just bad and the time period seems to be perfect .	0	0	1	+
if you 're looking for the same feel good one but in a new setting , this one 's for you .	1	0	0	+
this is a first for me , i didn 't like this movie . it was really bad .	0	1	1	-
it was good but the christmas carol by dickens was emotionally moving .	1	0	0	- 26

Figure 1: Example of training corpus. Here, we are only concerned with the words 'good', 'movie', and 'bad' for classes '+' and '-'

$$P(y) \to P(+) = \frac{3}{5}, \quad P(-) = \frac{2}{5}$$

$$P(good|+) = \frac{2}{4} \qquad \text{i.e. } \frac{\text{frequency of word for class}}{\text{total count of words for this class}}$$

1.2.2 One smoothed Naive Bayes Classifier

this introduces the problem that one of our probabilities could be zero; therefore, adjust naive bayes classifier with 'Add-one smoothing'

Definition 1.4 (Add-one smoothing)**.**

$$P(x_i|y) = \frac{count(x_i, y) + 1}{\sum_{x \in V}(count(x, y) + 1)} = \frac{count(x_i, y) + 1}{(\sum_{x \in V} count(x, y)) + |V|}$$

$$P(good|+) = \frac{2+1}{4+3} = \frac{3}{7} \qquad P(good|-) = \frac{1+1}{3+3} = \frac{2}{6}$$
$$P(movie|+) = \frac{1+1}{4+3} = \frac{2}{7} \qquad P(movie|-) = \frac{1+1}{3+3} = \frac{2}{6}$$
$$P(bad|+) = \frac{1+1}{4+3} = \frac{2}{7} \qquad P(bad|-) = \frac{1+1}{3+3} = \frac{2}{6}$$

Therefore for a test example: "Not as **good** as the old **movie**, rather **bad**" we use Equation 1.3 to get $P(+)P(x|+) = \frac{3}{5} \times \frac{3 \times 2 \times 2}{7^3} = 0.021$ and $P(-)P(x|-) = \frac{2}{5} \times \frac{2 \times 2 \times 2}{6^3} = 0.014$. So final outcome of the model is $+$. However, this sentence is clearly negative, yet we classify as positive.

1.2.3 Binary Naive Bayes

Within binary naive bayes, we only consider if a feature is present, rather than considering every time it occurs. Note, this means re-calculating the conditional probabilities from the training data.
E.g. "Not as **good** as the old **movie**, rather **bad movie**" we have $P(+)P(x|+) = \frac{3}{5} \times \frac{3\times2\times2}{7^3} = 0.021$ and $P(-)P(x|-) = \frac{2}{5} \times \frac{2\times2\times2}{6^3} = 0.014$. If we were still operating under the same formula in Equation 1.3 then we would have iterated over the label 'movie' twice, and thus multiplied by $P(good|+)$ twice (in the positive case).

1.2.4 Controlling for negation

We append 'Not_' after any logical negation (e..g n't, not, no, never) until the next punctuation mark.
"I didn't like the movie, but it was better than Top Gun" becomes "I didn't NOT_like NOT_the NOT_movie, but it was better than Top Gun"

1.2.5 Problems

- Conditional independence assumption
- Features considered equally important
- Context of words not taken into account
- New words (not seen at training) cannot be used

1.2.6 Summary

Very quick to train, Some evidence it works well on very small datasets.

1.3 Logistic Regression

(Discriminative Algorithm) because we directly learn $P(Y|X)$ since we don't care about $P(Y)$ or $P(X)$. They learn the input feauters most useful to discriminate between the different classes, without considering the likelihood of the input itself.

$$y(x) = g(z) = \frac{1}{1+e^{-z}}$$
$$z = w \cdot x + b$$
$$P(y=1) = \frac{1}{1+e^{-(w\cdot x+b)}}$$

where w is how important an input feature is to the classification decision and we have a threshold at 0.5 for decision making.

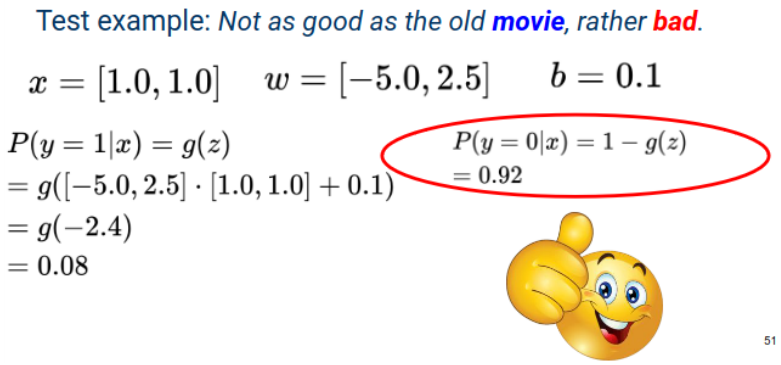


Figure 2: Here the vector $\boldsymbol{x}$ is made up of the collection of words, here, $x_1 = count(movie) \wedge x_2 = count(bad)$

If we don't have the w, then we learn parameters to make the model predictions close to our labels by using a loss function (to measure the distance between the true and predicted labels) and optimization algorithm (to minimize the function usually through gradient descent.)

Definition 1.5 (Logistic Regression)**.** How close is the predicted distribution Q to the true distribution P

$$H(P, Q) = -\sum_i P(y_i) \log Q(y_i)$$

1.3.1 Multiple Classes

In the case where we have words

$$x_1(bad) = 1$$
$$x_2(good) = 1$$
$$x_3(and) = 2$$

With sentiment analysis over 3 classes (+, -, and neutral), weights and bias are learnt per class. Then, they use a softmax function over the result of finding z_i.

$$y = g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

1.3.2 Summary

Logistic Regression considers the importance of feautres, so is better (than Naive Bayes) at dealing with correlated feautres, also better (than Naive Bayes) with larger datasets.

1.4 Neural Networks (NNs)

Definition 1.6 (Linear Layer)**.**

$$z = w \cdot x + b = \sum_{i=0}^{I} w_i x_i + b$$

Definition 1.7 (Non-linear activation function)**.**

$$y = g(z)$$

Definition 1.8 (Fully-connected layers)**.**

$$FFN(x) = (g^2(g^1(xW^1 + b^1))W^2 + b^2)W^3 + b^3$$

The neural networks allow for automatically learning dense feature representations (or alternatively pre-trained dense representations), not one-hot encodings.

1.4.1 Document Representation

How to get a document representation of sentence of fixed dimensionality?

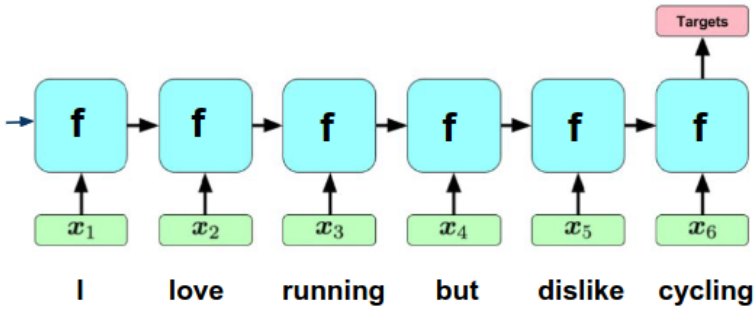
Average of sentence - bad idea:

Model architecutre fixed to sentence length size, model wieghts learnt for specific word positions

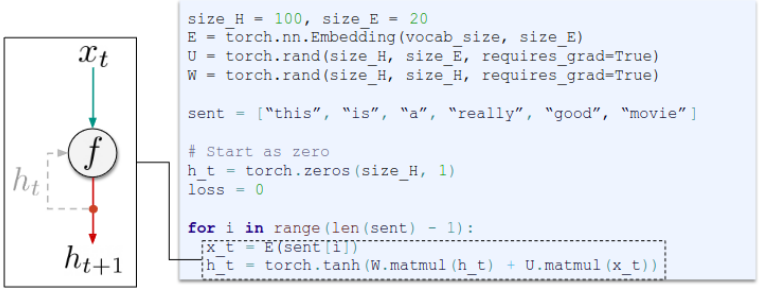
1.4.2 Neural Networks

Automatically learned features; flexibility to fit highly complex relationships in data, but they require more data to learn more complex patterns.

1.5 Recurrent neural networks (RNNs)



Natural language data is made up of sequences, so its natural to represent in a RNN; the value of a unit depends on own previous outputs - the last hidden state is the input to the output layer. The words are input into the model after an embedding layer which vecotrizes the ipnut.



1.5.1 Vanishing gradient problem

The model is less able to learn from earlier inputs (Tanh derivatives are between 0 and 1) (Sigmoid derivatives are between 0 and 0.25) - Gradient for earlier layers involves repeated multiplication of the same matrix W - depending on the dominant eignevalue this can cause gradients to either 'vanish' or 'explode'

☞ RNNs perform better when you need to understand longer range dependencies

1.6 CNNs

CNNs are composed of a series of convolution layers, pooling layers and fully connected layers. Convolutional layers Detect important patterns in the inputs. Pooling layers Reduce dimensionality of features and transform them into a fixed-size. Fully connected layers Train weights of learned representation for a specific task.

We can stack multiple filters on-top of eachother also.

☞ CNNs can perform well if the task involves key phrase recognition.

1.7 Accuracy and F1

Definition 1.9 (accuracy)**.**

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Definition 1.10 (f1-measure)**.**

$$f1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + 0.5(FP + FN)}$$

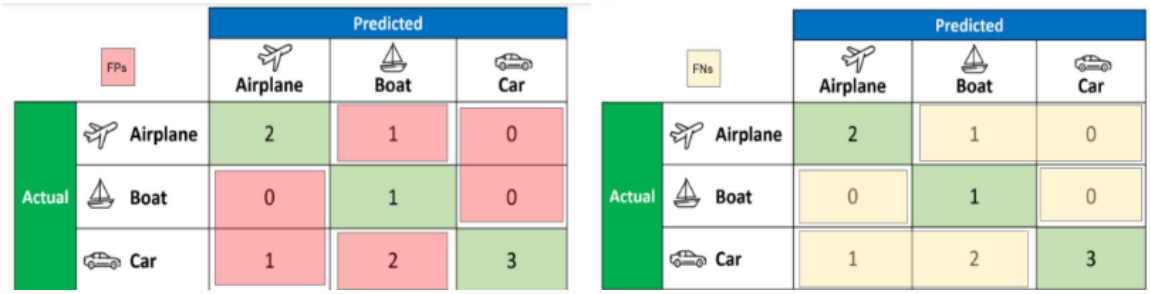
1.7.1 Macro averaging

averaging of each class F1 scores: increases the emphasis on less frequent classes

1.7.2 Micro averaging

TPs, TNs, FNs. FPs are summed across each class.

1.7.3 Microaveraged F1



$$\frac{\sum_i^C TP_i}{\sum_i^C TP_i + 0.5(\sum_i^C FP_i + \sum_i^C FN_i)} = \frac{\sum_i^C TP_i}{|Dataset|} = Accuracy$$

2 Language Models

2.1 Definition

Definition 2.1. Language modeling involves assigning probabilities to sequences of words. This could involve, predicting the next word in a sqeunce of words, or predicting a masked word in a sentence.

"Models that assign probabilities to sequences of words are called language models"[1]

2.1.1 Uni-directional

Here we use information from the left of the sentence to generate predictions about words that will appear to the right.

2.1.2 Bi-directional

Here we use information from both sides of the mask to 'fill in the gap'

2.2 N-Gram Modelling

The task of computing $P(w|h)$, the probability of a word w given some history h.

$$= P(w_n|w_1^{n-1})$$

One way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see the history and count the number of times that w directly followed the history.

$$P(\text{the}|\text{its water is so transparent that}) = \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}$$

However, this doesn't work for long histories. This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentence. Even simple legal extensions of the example sentence may have counts of zero on the web.

The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

$$\begin{aligned} P(\text{the}|\text{its water is so transparent that}) = Full\ context &= P(w_n|w_1^{n-1}) \\ Unigram\ approximation &\approx P(w_n) = P(\text{the}) \\ Bigram\ approximation &\approx P(w_n|w_{n-1}) = P(\text{the}|\text{that}) \\ Triagram\ approximation &\approx P(w_n|w_{n-2,n-1}) = P(\text{the}|\text{transparent that}) \end{aligned}$$

Definition 2.2 (N-gram modelling)**.** The N-gram models approximat ehistory by just the few last words with relation

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

We can then estimate these probabilities as 'MLE as relative frequencies'

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

The larger, the better the counts - larger n possible, however, trigams are often enough.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

(a) shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of eight words would be even more sparse [1]

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

(b) Shows total occurrances of each word [1]

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

(c) Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray. [1]

Figure 3: Bigram example

E.g. to find the probability of "want to" $\equiv P(to|want)$ use Definition 2.2

$$\begin{aligned} P_{N=2}(w_n|w_1^{n-1}) &\approx P(w_n|w_{n-2+1}^{n-1}) \\ &= \frac{C(w_{n-1}^{n-1}w_n)}{C(w_{n-1}^{n-1})} \\ &= \frac{C(\text{to}|\text{want})}{C(\text{want})} \\ &= \frac{C(608)}{C(927)} = 0.65587918 \end{aligned}$$

2.3 Evaluating Language Models

We can evaluate a sequence by considering the product of conditional probabilities.

$$P(w_1, \ldots, w_n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{2}$$

E.g. To find $P(< s >$ I want chinese food $< /s >)$ Given the above table and using Definition 2.2 (given also that $P(i| < s >) = 0.25 \wedge P(< /s > |food) = 0.68$)

$$
\begin{aligned}
P_{N=2}(w_n|w_1^{n-1}) &\approx P(w_n|w_{n-2+1}^{n-1}) \\
&= \frac{C(w_{n-1}^{n-1}w_n)}{C(w_{n-1}^{n-1})} \\
&\Rightarrow P(i|<s>) \cdot P(want|i) \cdot P(chinese|want) \cdot P(food|chinese) \cdot P(</s>|food) \\
&\approx 0.25 \cdot \frac{C(\text{want|i})}{C(\text{i})} \cdot \frac{C(\text{chinese|want})}{C(\text{want})} \cdot \frac{C(\text{food|chinese})}{C(\text{chinese})} \cdot 0.68 \\
&\approx 0.25 \cdot 0.33 \cdot 0.0065 \cdot 0.52 \cdot 0.68 \\
&= 0.000189618
\end{aligned}
$$

However, in Equation 2 we are multiplying many small numbers together that are less than 0. This is a very small number, so we can switch to log space & replace multiplication by addition

$$\log(P(w_1, \ldots, w_n)) = \log(\prod_{k=1}^{n} P(w_k|w_1^{k-1})) = \sum_{k=1}^{n} \log(P(w_k|w_1^{k-1}))$$

We can then take the exponent of this.

2.3.1 Problems with this approach

We have an issue here with longer outputs: The longer the output is, the lower its likelihood (many multiplications give a small number)

2.4 Perplexity

We normalize by the length of the text 'the inverse probability of a text, normalized by the # of the words'

Definition 2.3 (Perplexity)**.** In practice we don't use raw probability as our metric for evaluating language models, but a variant called perplexity. The perplexity (sometimes called PPL for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words.

Here n is the number of words:

$$
\begin{aligned}
PPL(w) &= P(w_1, w_2, \ldots, w_n)^{\frac{-1}{N}} \\
&= \sqrt[n]{\frac{1}{P(w_1, w_2, \ldots, w_n)}}
\end{aligned}
$$

We can then use the chain rule to expand the probability of W

$$PPL(w) = \sqrt[n]{\frac{1}{\prod_{k=1}^{n} P(w_k|w_1, \ldots, w_{i-1})}} = \sqrt[n]{\frac{1}{\prod_{k=1}^{n} P(w_k|w_1^{k-1})}}$$

Note that because of the inverse, the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model. It is a measure of the surprise in an LM when seeing new text.

- if we are finding the perplexity of a single word, the best possible score is 1.
- if our model uniformly picks words across a vocabulary of size $|V|$, the perplexity of a single word is: $|V|$. This is becuase (as mentioned in Section 2.4.1) if each sample has a chance $1/|V|$ of getting chosen.

2.4.1 Summary

Perplxeity allows us to choose the best LM for a test data: LM1 vs LM2: best LM is the one with the lowest perplexity

- However, perplexity is specific to the test-set

  "We can consider perplexity as the weighted average branching factor of a language. The branching factor of a language is the number of possible next words that can follow any word. Consider the task of recognizing the digits n English (zero, one, two,..., nine), given that (both in some training set and in some test set) each of the 10 digits occurs with equal probability $P = 1/10$ The perplexity of this mini-language is in fact 10. To see that, imagine a test string of digits of length N, and assume that in the training set all the digits occurred with equal probability.

  But suppose that the number zero is really frequent and occurs far more often than other numbers. Let's say that 0 occur 91 times in the training set, and each of the other digits occurred 1 time each. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. We should expect the perplexity of this test set to be lower since most of the time the next number will be zero, which is very predictable, i.e. has a high probability. Thus, although the branching factor is still 10, the perplexity or weighted branching factor is smaller" [1]

- How you tokenize the data matters

2.5 Cross Entropy Loss

The cross-entropy is useful when we don't know the actual probability distribution p that generated some data. It allows us to use some m, which is a model of p (i.e., an approximation to p). We don't know the true distribution, e.g. the likelihood of each possible next word, we only know how many times things happen in the training data.

Below, $q(x_i)$ is the model predicted probability of the word x_i given the previous words $x_1, \ldots, x_{i-1}$:

$$H(T, q) = -\sum_{i=1}^{N} \frac{1}{N} \ln q_e(x_i)$$

For a single observation (from Definition 1.5):

$$H(P, Q) = -\sum_{i} P(y_i) \log Q(y_i)$$

2.5.1 Maximum Likelihood Estimation

- Consider $\mathbb{X} = x_1, \ldots, x_n$ examples drawn independently form a true but unknown data generating distirbution $p_{data}(x)$.
- Let $p_{model}(x; \theta)$ be a parametric family of probability distributions over the same space indexed by θ. In other words, $p_{model}(x; \theta)$ maps any configuration x to a real number estimating the true probability $p_{data}(x)$.

- The maximum likelihood estimator for θ is $\arg\max_\theta p_{model}(\mathbb{X};\theta) = \arg\max_\theta \prod_{x\in\mathbb{X}} p_{model}(x;\theta)$
- since this is prone to numerical underflow, we go into the logarithm space: $\arg\max_\theta \sum_{x\in\mathbb{X}} \log p_{model}(x;\theta)$
- Because the arg max does not change when we rescale the cost function, we can divide by m to obtain a version of the criterion that is expressed as an expectation with respect to the empirical distribution $\arg\max_\theta \mathbb{E}_{x\sim\hat{p}_{data}} \log p_{model}(x;\theta)$

2.5.2 Conditional Cross-Entropy Loss

The maximum likelihood estimator can readily be generalized to estimate a conditional probability $P(x|y;\theta)$ in order to predict y given x, we denote Y as our observed targets and X as our observed inputs. Here θ represents the parameters of the model that we're triyng to optimize.

We choose parmeters based on the loss for the whole corpus

In the given sequence of equations, the objective is transformed from maximizing the likelihood estimation to minimizing the negative log-likelihood. This is equivalent to minimizing the cross-entropy between the model's predictions and the true data distribution. The following steps are taken from this link.

$$
\begin{aligned}
\hat{\theta}_{ML} &= \arg\max_\theta p_{model}(Y|X;\theta) && (3)\\
&= \arg\max_\theta \frac{1}{N}\prod_{i=1}^{N} p_{model}(y^{(i)}|x^{(i)};\theta) && (4)\\
&= \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N} \log p_{model}(y^{(i)}|x^{(i)};\theta) && (5)\\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log p_{model}(y^{(i)}|x^{(i)};\theta) && (6)\\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log p_{model}(y^{(i)}|x^{(i)};\theta) - \log p(x^{(i)}) && (7)\\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log p_{model}(y^{(i)}|x^{(i)};\theta) - \log p_{model}(x^{(i)}|\theta) && (8)\\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log \left(p_{model}(y^{(i)}|x^{(i)};\theta)p_{model}(x^{(i)}|\theta)\right) && (9)\\
&= \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} -\log p_{model}(y^{(i)},x^{(i)}|\theta) && (10)\\
&\approx \arg\min_\theta \left(\mathbb{E}_{p_{data}(x,y)}\left[-\log p_{model}(y,x|\theta)\right]\right) && (11)\\
&= \arg\min_\theta \left(\mathbb{E}_{p_{data}(x,y)}\left[-\log p_{model(\theta)}(y,x)\right]\right) && (12)
\end{aligned}
$$

In step 7 we turn it into a minimisation problem. In step 8 we subtracted logarithm of true and unknown probability of drawing the sample x_i, which does not change the argmax, as it is independent of θ. In step 9 we redefine $p_{model}(x^{(i)}|\theta) \equiv p_{model}(x^{(i)}|\theta)$ since our model is actually not modeling probability of the sample x_i

Definition 2.4 (Cross-Entropy Loss)**.**

$$H(T, q) = -\sum_{i=1}^{N} \frac{1}{N} \ln q(x_i)$$

Where $q(x_i)$ is the model predicted probability of the word x_i given the previous words $x_1, \ldots, x_{i-1}$:

2.6 Converting Between Cross Entropy and Loss

if you calculate Cross Entropy to the base e (if it is in base two then it is 2^H)

$$Perplexity(M) = e^H$$

2.7 Extrinsic vs Intrinsic Evaluation

If the goal of the languag model is to support with another task, the best choice of language model is the one that improves downstream task performance the most (extrinsic evaluation). Perplexity is less useful in this case (intrinsic evaluation).

References

[1] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 2023. URL: https://web.stanford.edu/~jurafsky/slp3/.