NLP Study Notes

Du Jinrui

2022

目录

Chapter 1

Shortest-Path Algorithms and Dynamic Programming

1.1 Graphs

- Adjacency matrix

  For a graph $G = \{V, E\}$, where V is a set of vertices of size m, the adjacency matrix M is of shape $m \times m$ where $M_{i,j}$ is the number of edges connecting the two vertices.

For an undirected matrix, $M^T = M$.

The book says we can use a cost adjacency matrix to represent the weight of edges setting the weight to ∞ when there is no link between two vertices. However I notice that when there are two edges between two vertices, there is only one weight in the cost adjacency matrix for the two edges. It's probably better to use incident matrix together with the weight matrix both to avoid using ∞ and when there are two edges between two vertices? (draft*)

- Incident matrix
  In an incident matrix, the rows names are vertices, the columns names are edges.

- Next-node lists
  For example:
  $\text{next}(v_a) = \{v_b, v_c\}$, $\text{next}(v_b) = \{\}$, $\text{next}(v_c) = \{v_b\}$

1.1.1 Graph coding problems

1.2 Dynamic programming

When designing a DP algorithm, there are two things to consider:

1. Deconstruct a big problem into smaller (recursive) subproblems.

2. Store intermediate results.

1.2.1 DP coding problems

- Nth Fibonacci Number
- Longest Increasing Sub-sequence
- Coin Change
- Edit Distance
- Freedom Trail

1.3 The Viterbi algorithm

Chapter 2

Logistic Regression

2.1 The importance of establishing a baseline

We draw a function that shows decreased marginal accuracy with increasing model complexity. From this graph, we observe an upper limit. This limit helps us making informed decisions like:

1. Is this project feasible? (the requirement is 75% accuracy but the upper limit is 72%.)

2. Is it cost-effective to add model complexity?

Furthermore, if we use a complex model upfront without setting a baseline but the accuracy is bad, then it's hard for us to tell whether there was a mistake when building the model or it's because the problem is too complex.

2.2 Understanding LR

graph of 1d data draft*

Why sigmoid?

2.3 From likelihood to cost function

The likelihood function is defined as $l\left(\theta|D\right) = f\left(D|\theta\right)$. f can be either a PMF or a PDF. $|$ is used instead of ; because we employ the Bayesian view (not frequenist) and see θ as a random variable. l is a function of θ and doesn't integrate to 1 (with respect to θ).

The likelihood function of logistic regression is

$$\prod_{i=1}^{n} \sigma\left(wx_i + b\right)^{y_i} \left(1 - \sigma\left(wx_i + b\right)\right)^{1-y_i}.$$

(see derivation) Maximizing the likelihood is equal to minimizing the negative log-likelihood:

$$\mathit{cost}\left(w, b\right) = -\sum_{i=1}^{n} y_i \ln \sigma\left(wx_i + b\right) + \left(1 - y_i\right) \ln\left(1 - \sigma\left(wx_i + b\right)\right).$$

And we get KL divergence, or binary cross-entropy, which is convex. (Why is it convex?)

2.3.1 On KL-divergence and cross-entropy

2.3.2 Logistic loss

If the outcome space is $y = \{-1, 1\}$ instead of $y = \{0, 1\}$, then

$$p(y_i = 1 | f(x_i)) = \sigma(f(x_i)) = \frac{1}{1 + e^{-f(x_i)}}$$
$$p(y_i = -1 | f(x_i)) = 1 - \sigma(f(x_i)) = \frac{1}{1 + e^{f(x_i)}}.$$

In both cases

$$p(y_i | f(x_i)) = \frac{1}{1 + e^{-y_i f(x_i)}}.$$

The negative log-likelihood is

$$\sum_{i=1}^{n} \log(1 + e^{-y_i f(x_i)}).$$

Which is called the log loss/logistic loss and it's the same thing as the cross-entropy loss.

2.4 Implement LR with mini-batch GD

The cost function can't be solved analytically, hence we use gradient descent. The derivative of the sigmoid function is:

$$\sigma(x)(1 - \sigma(x)).$$

Knowing this facilitates the calculation of the gradient:

$$\frac{\partial l(w,b)}{\partial w} = \sum_{i=1}^{n}(\sigma(wx_i + b) - y_i)x_i$$
$$\frac{\partial l(w,b)}{\partial b} = \sum_{i=1}^{n}\sigma(wx_i + b) - y_i.$$

Now we update the parameters:

$$w^{t+1} = w^t - \eta_t \sum_{i=1}^{n}(\sigma(wx_i + b) - y_i)x_i$$
$$b^{t+1} = b^t - \eta_t \sum_{i=1}^{n}\sigma(wx_i + b) - y_i.$$

Now we've got the updates using GD. The updates using mini-batch GD and stochastic GD become apparent. The former is:

$$w^{t+1} = w^t - \eta_t \sum_{x_i, y_i \in batch}(\sigma(wx_i + b) - y_i)x_i$$
$$b^{t+1} = b^t - \eta_t \sum_{x_i, y_i \in batch}\sigma(wx_i + b) - y_i.$$

Between GD and stochastic GD, mini-batch GD finds the balance between robustness and efficiency. Moreover, it works well with GPU, and it helps escaping the saddle point.

code draft*

Chapter 3

Generalization

3.1 When w goes to infinity

When the problem is linearly separable, as w goes to infinity:

$$\lim_{w\to\infty} p\left(y_i = 1|x_i; w, b\right) = \lim_{w\to\infty} \frac{1}{1+e^{-(wx_i+b)}} = 1 \ for \ wx_i + b > 0,$$

$$\lim_{w\to\infty} p(y_i = 0|x_i; w, b) = \lim_{w\to\infty} \frac{e^{-(wx_i+b)}}{1+e^{-(wx_i+b)}} = 0 \ for \ wx_i + b < 0.$$

At this time, MLE is the largest:

$$MLE = \arg\max_{w,b} \prod_{i=1}^{n} p\left(y_i = 1|x_i; w, b\right)^{y_i} p\left(y_i = 0|x_i; w, b\right)^{1-y_i}.$$

It is consistent with our goal of maximizing the likelihood function to aim for a large w. For a linearly separable problem, w doesn't converge, and regularization gives bounded solution.

For a non-linearly separable problem, w can converge (mathematically, why?). But when there are too many features, the non-separable becomes the separable, again, w goes to infinity, and uncertainty regions shrink to 0. At this point, limiting the magnitude of w leads to better generalization and gives back uncertainty regions. How are all these happening? 1 2 Graphically, higher degree terms variables with smaller w doesn't disappear, but go 'out of range', e.g. $y = 6x_1 + 3x_2^2$ vs $y = 6x_1 + 0.1x_2^2$. draft*

We don't discuss feature selection here, why don't we just use feature selection? Is there an algorithm for separability testing?

3.2 L1 and L2 regularization

3d geometric moving representation of l1 and l2 and why l1 makes some parameters 0. draft*

There are some disadvantages of l1 regularization:

1. It's not differentiable everywhere, so gradient descent doesn't work, in this case we can use subgradient descent (I don't need to know the details).

2. When a group of collinear features exist, it randomly selects one feature, but we want the best feature. The lecturer says using elastic net can counter this problem but I don't know how. It's another topic. draft*

3.3 K-fold CV

When dataset is small, we can increase k. One extreme case is leave-one-out CV.

3.4 MLE, MAP and L1, L2

MLE:

$$p(D|\theta).$$

MAP:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \propto p(D|\theta)p(\theta).$$

MAP estimator:

$$\theta_{MAP} = \arg\max_\theta \ prior \ \cdot \ likelihood.$$

Assume prior is $p(\theta) \sim N(0, \sigma^2)$,

$$\begin{aligned} p(\theta) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\theta^2}{2\sigma^2}\right) \\ &\propto \exp\left(-\frac{\theta^2}{2\sigma^2}\right), \\ \arg\max_\theta \ \log(p(\theta)) &= \arg\max_\theta \ \log\left(\exp\left(-\frac{\theta^2}{2\sigma^2}\right)\right) \\ &= \arg\max_\theta \ -\frac{\theta^2}{2\sigma^2}, \\ \theta_{MAP} &= \arg\min_\theta - \log \ likelihood + \frac{1}{2\sigma^2}\theta^2. \end{aligned}$$

This looks very familiar. MAP estimator with Gaussian prior equals adding a l2 regularization term to the cost function (and how does the λ coefficient relates to the variance? draft*).

Similarly when $p(\theta) \sim Laplace(0, b)$, the resulting cost function is added by l1 term.

Chapter 4

Naive Bayes

Chapter 5

Theory of Convex Optimization

5.1 Mathematical optimization

A mathematical optimization problem has the form:

$$\begin{gathered}\text{minimize } f_0(x)\\ \text{subject to } f_i(x) \leq b_i,\ i = 1, 2, \ldots, m,\end{gathered}$$

in which $f_0, f_1, \ldots, f_i$ are functions that map R^d to R.

In a linear programming problem, $f_0, f_1, \ldots, f_i$ are linear, which means

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y).$$

For a convex optimization problem, we have

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y).$$

, with $\alpha \geq 0, \beta \geq 0$ and $\alpha + \beta = 1$. Comparing linear programming problem and convex optimization problem, because convexity is less restricted, all linear problems are convex optimization problems. (I have problem understanding this, this link helps. draft*)

5.2 Least-squares, linear programming and convex optimization

Generally, mathematical optimization problems are hard to solve, but we present 3 exceptions in this section.

5.2.1 Least-squares

5.2.2 Linear programming

5.2.3 Convex optimization

5.3 Non-linear optimization

Non-linear problems are optimization problems that are not linear, but not known to be convex.

Finding a global minima is time-consuming, and finding a local minima is more of an art than a science.

Formulating a non-linear problem is relatively straightforward, but the difficulty lies in solving it. While for a convex optimization problem, solving it is straightforward, but the challenge is in problem formulation.

5.4 Convex sets

5.5 Convex functions

Chapter 6

SVM

6.1 Decision theory and the statistical learning framework

input space: X
action space: A
output space: y
loss:

$$l : A \times y \to R$$

decision function:

$$f : X \to A$$

risk:

$$R(f) = E[l(f(x), y)]$$

Bayesian decision function:

$$f^* = \arg\min_f R(f)$$

empirical risk:

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

empirical risk minimizer:

$$\hat{f} = \arg\min_f \hat{R}_n(f)$$

hypothesis space: F

constrained empirical risk minimizer:

$$\hat{f}_n = \arg\min_{f \in F} \hat{R}$$

constrained risk minimizer:

$$f_F^* = \arg\min_{f \in F} R.$$

6.2 Define SVM

input space: R^d

action space: R

output space: $\{-1, 1\}$
hypothesis space:

$$F = \{f(x) = w^T x | w \in R^d\}$$

loss/hinge loss:

$$l\left(f(x), y\right) = \max(1 - y_i f(x_i), 0) = \left(1 - y_i f(x_i)\right)_+$$

l2 regularization:

$$min_{w \in R^d} \sum_{i=1}^{n} \left(1 - y_i f_w(x_i)\right)_+ + \lambda \|w\|_2^2,$$

How does minimizing w relates to minimizing the risk function? draft*
Can we derive the loss function from a probability distribution like we did for logistic regression? draft*
$f_w(x_i)$ is called the score, and $y_i f_w(x_i)$ is called the margin.

graph of logistic loss, hinge loss, perceptron loss, zero-one loss, square loss with respect to margin draft*

6.3 Duality

6.4 Kernel

Chapter 7

Tree Based Models

Chapter 8

Text Processing

The text analytic workflow:

raw data → word segmentation → cleaning (stopwords, lowercase, special character) → normalization (stemming, lemmazation) → feature extraction (tf-idf, word2vec) → modeling (classification, similarity measures)

8.1 Segmentation

8.1.1 Tools for Chinese words segmentation

- Jieba: https://github.com/fxsjy/jieba
- SnowNLP: https://github.com/isnowfy/snownlp

- LTP: http://www.ltp-cloud.com/
- HanNLP: https://github.com/hankcs/HanLP/

8.1.2 Max matching algorithm

sentence: 经常有意见分歧
dictionary: [“我们”, “经常”, “有”, “有意见”, “意见”, “分歧”]
window size: 5

- forward-max matching:
  经常有意见
  经常有意
  经常有
  经常（check）
- backward-max matching:
  有意见分歧
  意见分歧
  见分歧
  分歧（check）

Both approaches gives 经常 | 有意见 | 分歧, which is not a meaningful split.

8.1.3 Segmentation considering semantic links

To get a semantically meaningful split, we use language model. For example, the unigram model. To use the model,

calculate the probability of the whole sentence P(经常有意见分歧) for every possible way of segmentation:
by iid:
P(经常 | 有意见 | 分歧) = P(经常) * P(有意见) * P(分歧)
P(经常 | 有 | 意见 | 分歧) = P(经常) * P(有) * P(意见) * P(分歧),
calculating the product of small numbers is not possible because of limited floating point precision, so we apply log transformation.

Another problem is that, for a large text, there can be lots of ways to segment it, in this case we use the Viterbi algorithm to reduce the complexity.

8.2 Filtering of words

Stop words.

Low frequency words. Dropping low frequency words helps reducing computation time. Usually 80%(a majority) of words are low frequency.

8.3 Normalization of words

Lemmazation.

Stemming.

8.4 Spell correction

After segmentation, if we find some words are not in the dictionary, we use edit distance to correct these words. (How to segment a sentence if there is no corresponding words to a part of the sentence in the dictionary? What to do if multiple candidates have the same edit distance? draft*)

To find the candidates for correction, we can ether compare a wrong word with all the other words in the dictionary, or, more efficiently, we generate 'words' that are small distance away from the wrong word, then filter them according to the dictionary. Finally, we rank the rest candidates by the probability $\hat{c} = \arg\max_{c \in candidates} p(c|s)$, by the Bayes theorem, $\hat{c} \propto \arg\max_{c \in candidates} p(s|c) * p(c)$. The first term, the probability of typing the wrong word given the intended word is c, can be obtained from historical data. The second term, the probability of c appeared in the text, can be obtained from counting the number of appearance of the word in the corpus.

Here are a list of open source Chinese spell correction tools.

Chapter 9

Text Representation

9.1 TF-IDF

To represent a text, there are several options. Boolean vector being one of them doesn't consider the number of occurrence of every word. Count vector does consider the occurrence, but it's not that the higher the frequency, the more important the word is. To classify several documents, if the same word appears in every document, then the word is not very useful for classifying that document. Conversely, those words appears only in certain types of documents are more useful for their classification.

Tf-idf can solve this problem. It is a great baseline, and often better than neural networks. It is defined as:

$$tf\text{-}idf(t, d, D) = tf(t, d) \times idf(t, D).$$

Term frequency is the same thing as the count vector, defined as the number of occurrence of a term t in a document d. Document frequency is the size of a subset of all the documents D where all the documents containing a term t. We divide document frequency by the total number of all the documents D to get inverse document frequency. See the effect of the log function in idf. See Why add 1 to idf. Idf is the same for every document when t is fixed.

9.2 Text similarity

- Euclidean distance

  $$d(x, y) = \sqrt{(x_1 - y_1)^+(x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}.$$

  To visualize Euclidean distance, think about Pythagorean theorem.

  Euclidean distance doesn't consider vector direction, but vector has both magnitude and direction.

- Cosine distance
  Contrary to Euclidean distance, cosine distance only considers direction. It measures the angle between the vectors.

  $$d(x, y) = \frac{x \cdot y}{\|x\| \|y\|}.$$

To understand this formula, think about the geometric interpretation of dot product: $x \cdot y = \|x\|\|y\|\cos\theta$.

9.3 Word2Vec

We talked about how to represent text, we will now talk about word representation. We could represent words by one-hot encoding:

“我们”：	[1, 0, 0]
“和”：	[0, 1, 0]
“他们”：	[0, 0, 1]

But how to calculate the similarities between two words? The Euclidean distance of every pair of different words is $\sqrt{2}$, the cosine distance is 0.

We introduce word vector. While one-hot matrix is sparse, word vector is dense, but every word vector has lesser elements than word vector, because the size of a one-hot vector is the number of all the words of a text, can be millions, while the size of every word vector is a hyperparamter that we can tune. Normally, the more words we have, the longer the vectors we could set, but long vectors can leads to overfitting.

To evaluate word vectors, we can use some dimension reduction method and visualize them.

Once we have the word vectors, we can use them to represent sentence. One simple way is to calculate the average of the vectors of the words appeared in the sentence (bag of words).

The purpose of word vector is to quantify meaning. We have developed several ways of represent meaning, for example, denotational meaning, WordNet, and the aforementioned one-hot vector, but the most successful one for the computer to grasp is the distributional semantics, which says: a word's meaning is given by the words that frequently appear close-by.

"You shall know a word by the company it keeps" (J.R.Firth 1957:11)

9.3.1 SkipGram

SkipGram or Skip-NGram, opposite to CBOW(Continuous Bag of Words) which uses surrounding words to predict a center word, uses a center word to predict surrounding words. The number of surrounding words is a hyperparameter we can tune, usually between 5 and 20.

The objective function is

$$\prod_{t=1}^{T} \prod_{-m \leq c \leq m, c \neq 0} P(w_{t+c} | w_t; \theta),$$

where w stands for word, T is the total number of words, m is the window size, and θ, is the only parameter (not the only hyperparameter) we need to learn. It consists of 2 sets of vectors, a vector for each word as a context word, and a vector for each word as a center (target) word.

As an example, the objective function of the sentence “We love natural language processing” with the window size equal to 1 is:
P(love | we)P(we | love)P(natural | love)P(love | natural)P(language | natural)P(natural | language)P(processing | language)P(language | processing) parameterized by context vector and center word vector of each word.
By using 2 kinds of vector for each word, we significantly simplify the calculation. Otherwise we have to apply Bayes theorem to calculate, for example, P(love | we) and P(we | love).

We don't consider the distance between a context word and the center word. There are other algorithms that consider this, and they are useful for syntax parsing, but for semantics, position doesn't matter a lot.

Taking the negative log of the objective function, and apply the average of the function (see the motivation of averaging the llh), we get:

$$-\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq c\leq m, c\neq 0}\log P(w_{t+c}|w_t;\theta).$$

Now, what is the function of $P(w_{t+c}|w_t;\theta)$? For each center (target) word, the conditional probability must sum to 1, and for any number, to make it positive, we apply the exponential transform. The result is the softmax function:

$$\frac{e^{u_{c+t}\cdot v_t}}{\sum_{c'=-m,c'\neq 0}^{m} e^{u_{c'+t}\cdot v_t}},$$

where u_c and $u_{c'}$ are the context vector of a word, v_t is the vector of the target word. (Why use dot product but not cosine similarity? Why apply softmax to the output layer of the neural net? How to map our function to layers of the neural net? draft*) Demystifying Neural Network in Skip-Gram Language Modeling

Now we calculate the gradient:

$$\begin{aligned}
\frac{\partial}{\partial v_w} \log \frac{e^{u_o \cdot v_w}}{\sum_x e^{u_x \cdot v_w}} &= \frac{\partial}{\partial v_w} \log e^{u_o \cdot v_w} - \log \sum_x e^{u_x \cdot v_w} \\
&= \frac{\partial}{\partial v_w} u_o \cdot v_w - \log \sum_x e^{u_x \cdot v_w} \\
&= u_o - \frac{\partial}{\partial v_w} \log \sum_x e^{u_x \cdot v_w} \\
&= u_o - \frac{1}{\sum_x e^{u_x \cdot v_w}} \frac{\partial}{\partial v_w} \sum_y e^{u_y \cdot v_w} \\
&= u_o - \frac{1}{\sum_x e^{u_x \cdot v_w}} \sum_y \frac{\partial}{\partial v_w} e^{u_y \cdot v_w} \\
&= u_o - \frac{1}{\sum_x e^{u_x \cdot v_w}} \sum_y e^{u_y \cdot v_w} u_y \\
&= u_o - \sum_y \frac{e^{u_y \cdot v_w}}{\sum_x e^{u_x \cdot v_w}} u_y \\
&= u_o - \sum_y softmax(u_y \cdot v_w) u_y \\
&= u_o - \sum_y P(y|w) u_y,
\end{aligned}$$

which is the difference between the context word and the expected context word.

9.3.2 Negative sampling & hierarchical softmax

9.4 Other word2vec algorithms

9.4.1 Matrix factorization

9.4.2 GloVe

9.4.3 Gaussian embedding

Chapter 10

Language Model

10.1 Intro

The probability chain rule:

$$\begin{aligned} p(w_1, w_2, w_3, \ldots, w_n) =& p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \\ & \ldots \\ & p(w_n|w_1, w_2, \ldots, w_{n-1}). \end{aligned}$$

The Markov assumption:

- 1st-order Markov assumption:

$$p(t_n|t_{n-1}, t_{n-2}, \ldots, t_1) = p(t_n|t_{n-1})$$

- 2nd-order Markov assumption:

$$p(t_n|t_{n-1}, t_{n-2}, \dots, t_1) = p(t_n|t_{n-1}, t_{n-2})$$

- 3rd-order Markov assumption:

$$p(t_n|t_{n-1}, t_{n-2}, \dots, t_1) = p(t_n|t_{n-1}, t_{n-2}, t_{n-3})$$

Now let's see an example, by the chain rule:

P(I love nlp) = P(I) P(love | I) P(nlp | I, love)

By iid, we get the **unigram** model, but the unigram model doesn't consider the order of the words:

P(I love nlp) = P(I)P(love)P(nlp) = P(love I nlp)

By the 1st-order Markov assumption, we get the **bigram** model:

P(I love nlp) = P(I)P(love | I)P(nlp | love)

By the 2nd-order Markove assumption, we get the trigram model:

P(I love nlp) = P(I)P(love | I)P(nlp | I, love)

With the Nth-order Markove assumption, we train the model to get $P(w_n|w_{n-1}, w_{n-2}, \ldots, w_1)$ for every word w. When N gets large, it becomes in-efficient, and not every word is surrounded by the same group of words so the resulting matrix is sparse. So we avoid using large N. But how to train an n-gram model, or in another word, how to get $P(w_n|w_{n-1}, w_{n-2}, \ldots, w_1)$? Simply count the number of appearance of each term then divide by the corresponding total. Because it's very simple, I won't elaborate.

10.1.1 Perplexity

We could do extrinsic evaluation on trained models, by applying the models directly to a task, e.g., spell correction, then compare the efficacy. However, this is time-consuming. Another way is to apply train-val-test set split (intrinsic evaluation), so we can evaluate the models and then apply the model to different tasks (not only spell correction). However, this can be a bad approximation to the actual performance of the real task. Yet it is still helpful in pilot experiments.

The metric we will use is perplexity:

$$2^{-\frac{1}{n}\log P(w_1, w_2, \ldots, w_n)} = P(w_1, w_2, \ldots, w_n)^{-\frac{1}{n}} = \sqrt[n]{\frac{1}{P(w_1, w_2, \ldots, w_n)}}.$$

It is 2 (or any number, depending on the base of log) to the negative log likelihood. It measures the degree of complexity of a text, the lower the score, the higher the probability, and the text sounds more naturally to human. (See this video for intuition of perplexity)

Perplexity of bigram models looks like this:

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m}\prod_{j=1}^{|s_i|}\frac{1}{P(w_j^{(i)}|w_{j-1}^{(i)})}}, w_j^{(i)} \rightarrow \text{j-th word in i-th sentence.}$$

If we concatenate the sentences in W:

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m}\frac{1}{P(w_i|w_{i-1})}}, w_i \rightarrow \text{i-th word in test set.}$$

Log perplexity is easier to compute:

$$\log PP(W) = -\frac{1}{m}\sum_{i=1}^{m}\log P(w_i|w_{i-1}).$$

10.1.2 Smoothing

When a word appears in the test set but not in the training set, the probability of this word is 0, hence the probability of the whole sentence is 0, thus the model loses its power. To counter this, we use smoothing.

- Add-one Smoothing / Laplace Smoothing
  Also used in naive Bayes:

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_i)}$$

$$P_{Add1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_i) + V},$$

where c means count. V is the total number of words in the text, so that the sum of the probability of none-existing words is 1.

- Add-K Smoothing

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_i) + kV},$$

where k is a hyperparameter. We choose the k that minimize the perplexity.

- Interpolation

$$\begin{aligned} P_{inter}(w_n|w_{n-1}, w_{n-2}) =& \lambda_1 P(w_n|w_{n-1}, w_{n-2}) + \\ & \lambda_2 P(w_n|w_{n-1}) + \\ & \lambda_3 P(w_n) \\ \sum_{n=1}^{3} \lambda_i =& 1. \end{aligned}$$

(prove that mixture of random variables is also a random variable draft*)

The justification of adding bigram and unigram when using trigram is as follows:

$$\begin{aligned} &P(\text{kitchen}) = 0.2 && P(\text{kitchen}|\text{in the}) = 0 \\ &P(\text{in the kitchen}) = 0 && P(\text{domus}|\text{in the domus}) = 0 \\ &P(\text{domus}) = 0 \end{aligned}$$

If $P(\text{kitchen}) > P(\text{domus})$, then it shouldn't be that $P(\text{kitchen}|\text{in the}) = P(\text{domus}|\text{in the domus})$