Natural Language and Processing

Utkarsh Agiwal

June 15, 2022

Contents

1 Introduction

What is Natural Language Processing or usually called NLP? We use it all the time without even knowing it. Each time while texting on WhatsApp, using the auto-correct features, our email filters for spam/non-spam, subtitles/transcript in live videos, lectures etc. everything is an application of NLP.
One of the reasons I am interested in NLP is its growing boom and research in the recent years. Currently our world revolves around automation. These automations wouldn't be possible if it weren't for Artificial Intelligence and one of its major branches, Natural Language Processing (NLP). The world is getting **data rich**. Data is now a commodity for most businesses and a lot of this data is unstructured. This means there is a need for such applications. Here is a report on my learnings of NLP.

2 Basics of NLP

2.1 Regular Expression (RE)

This is a language for specifying text search strings. For example to search for a *word*, we type */word/*. Regex (short for Regular Expression) is case sensitive. So the above expression differentiates between *word* and *Word*. This problem is solved by using []. Now */[Ww]ord/* will not differentiate for the earlier. Similarly in regex there are a lot of syntax rules that solve many such string operation problems in NLP. A gist of them:

RE	Example	Match
[]	/[1234567890]/	any digit
-	/[A-Z]/	an upper case letter
^	/[^A-Z]/	not an upper case letter
*	/a*/	any string of zero or more aa's
+	/a+/	one or more occurrences of a
$	/$/	end of line
()	/pupp(y|ies)/	puppy or puppies
|	/cat|dog/	occurrence of word cat or dog

2.2 Word Tokenization

It is the task of segmenting text into words. For many NLP applications, we need to decide about keeping punctuation and other grammatical tools since they might help the training algorithm, for example in sentiment analysis. Most tokenization schemes have two parts: a token learner, and a token segmenter. The token learner takes a raw training corpus (sometimes roughly pre-separated into words, for example by white-space) and induces a vocabulary, a set of tokens. The token segmenter takes a raw test sentence and segments it into the tokens in the vocabulary. In python this is usually done using *word_tokenize* function imported from **NLTK** library.

2.3 Word Normalization, Lemmatization and Stemming

Word Normalization is the task of putting words/tokens in a standard format so that there are no discrepancies in different datasets used for training. Case folding is one of its kind. Mapping all words to lowercase helps in generalising words.
Lemmatization is the task of determining that two words have the same root, despite their surface differences. For example, 'eat' and 'eating' have the same root eat. Lemmatization algorithms are complex. They involve **morphological parsing** of the word. So, idea of **stemming** was introduced which is a naive way of parsing. It involves chopping of word-final affixes. In python this is usually done using *porter_stemmer* function.

3 Classification Models for Sentiment Analysis

The most basic classification model is the Logisitic Regression. It is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

3.1 Logisitic Regression

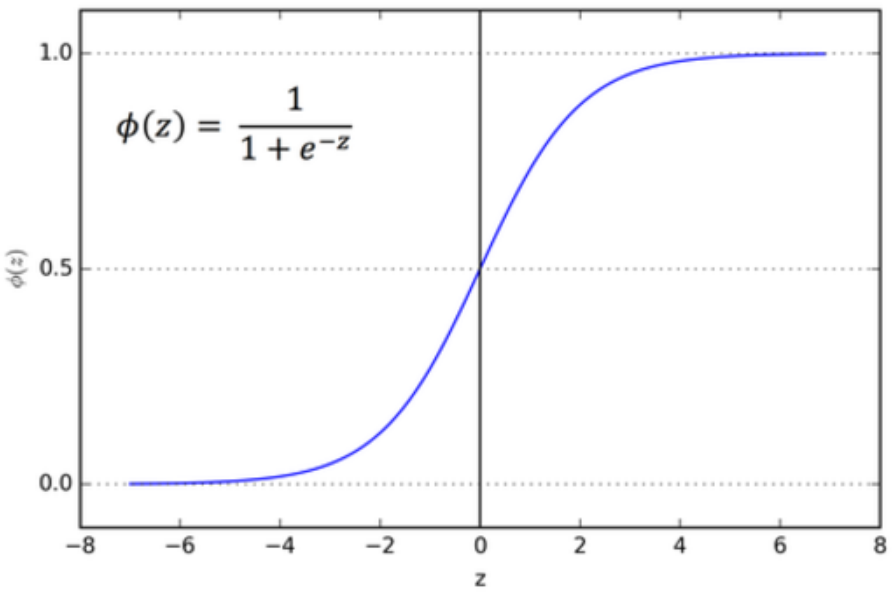


Figure 1: Logistic Regression

$$h_\theta(x) = g(\theta^T x) \tag{1}$$

$$g(z) = \frac{1}{1+e^{-z}} \tag{2}$$

$$\text{Predict y} = 1 \text{ if } h_\theta(x) \geq 0.5 \Rightarrow \theta^T x \geq 0$$
$$\text{Predict y} = 0 \text{ if } h_\theta(x) \leq 0.5 \Rightarrow \theta^T x \leq 0$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^i), y^i) \tag{3}$$

$$Cost(h_\theta(x), y) = \begin{cases} \text{-log}(\text{h}_\theta(x)) \text{ if y} = 1 \\ \text{-log}(\text{1-h}_\theta(x)) \text{ if y} = 0 \end{cases} \tag{4}$$

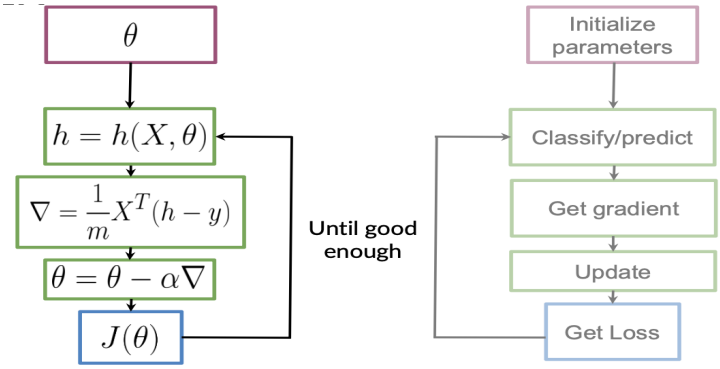


Figure 2: Training LR

Given a tweet/sentence, we need a way to classify if its positively(1) or negatively(0) sentimented. Given a sentence in english(WLG) corpus, the first step is to mathematically represent any input so that it can be fed into the algorithm.

How to mathematically represent tweets?

After preprocessing the tweets as explained in the basics of NLP using tokenization and lemmatization, we create a vocabulary of all the unique words we have counting their frequency in positive/negative tweets as shown in figure.

Vocabulary	PosFreq (1)	NegFreq (0)
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

Figure 3: Frequency Table

For each tweet, we calculate the positive and negative frequency using the vocabulary table just shown above. For example: If the tweet is : "I am sad." It can be represented in vectorial form as:

$$\begin{bmatrix} 1 & 6 & 8 \end{bmatrix}$$

$$\begin{bmatrix} bias\_vector & sum\_of\_positive\_freq & sum\_of\_neg\_freq \end{bmatrix}$$

Hence after converting each tweet into a vector of 1x3, the training set is put into the logistic regressor and accuracy is tested out.

3.2 Naive Bayes

It is based upon the Bayes' rule (conditional probability).

$$P\left(\frac{X}{Y}\right) = P\left(\frac{Y}{X}\right) * \frac{P(X)}{P(Y)} \tag{5}$$

Lets look at usage of this algorithm in sentiment classification.
Using the table in Figure 3, a probability table is created with each value corresponding to the value $P\left(\frac{w_i}{class}\right)$.

Word	Pos	Neg
I	0.24	0.25
am	0.24	0.25
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.08
not	0.08	0.17

Tweet : I am happy today; I am learning. To classify this tweet, ratio of it being positive to negative is calculated. If its > 1, the tweet is positive or else the opposite. For example, for this tweet:

$$\prod_{i=1}^{m} \frac{P(w_i|pos)}{P(w_i|neg)} = \frac{0.14}{0.10} = 1.4 \quad > 1$$

$$\frac{0.20}{0.20} * \frac{0.20}{0.20} * \boxed{\frac{0.14}{0.10}} * \frac{0.20}{0.20} * \frac{0.20}{0.20} * \frac{0.10}{0.10}$$

Figure 4: Naive Bayes classification

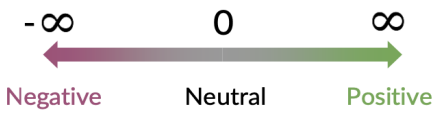
As it is visible in the table of probabilities that some probabilities can turn out to be zero. That might create problems in the equation we used to classify the tweet. That's where **smoothing** jumps in.

Laplace Smoothing

$$P\left(\frac{w_i}{class}\right) = \frac{freq(w_i, class) + 1}{N_{class} + V_{class}} \tag{6}$$

where, N_{class} = frequency of all words in class
V_{class} = number of unique words in class
To get better scaling logarithm of the conditions is taken which produces the results:

$$\sum_{i=1}^{m} log \frac{P(w_i|pos)}{P(w_i|neg)} > 0$$



Naive Bayes Assumptions and Errors

- Independence - Often the words are not independent, they are connected to each other. Naive Bayes doesn't include the effect of previous words.
- Relative frequencies in corpus
- Adversarial Attacks (Sarcasm and Irony) - 'This is a ridiculously movie. The plot was gripping and I cried through till the ending!' Processed tweet: [ridicul, power, movi, plot, grip, cry, end] would classify this tweet as negative and in fact its positively sentimented.

4 Vector Space Models

Vector space models help in designating similarity to sentences which have same meaning. This model represents words as vectors as a representation that captures relative meaning.

Word by Word design - Construction of vector spaces

Idea: Number of times two words occur together within a certain distance k. Consider two sentences:

- I like simple data
- I prefer simple raw data

Taking k=2 in the given example, vector space would look like:

	simple	raw	like	I
data	2	1	1	0

Since 'data' is within 2 words distance of 'simple' 2 times and similarly for other pairs.

Word by Document design - Construction of vector spaces

Idea: Number of times a word occurs within a certain category. For example:

	Entertainment	Economy	Machine Learning
data	500	6620	9320
film	7000	4000	1000

Vector Space visualization

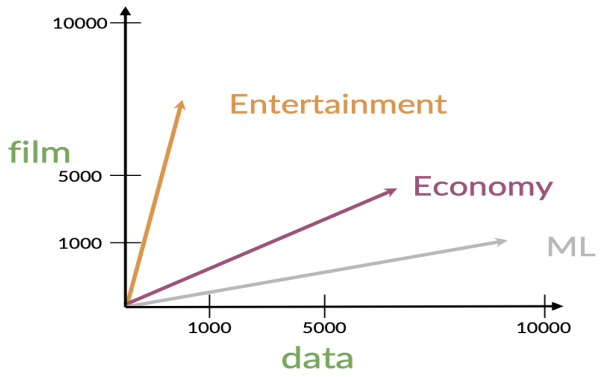


Figure 5: Vector space

4.1 Euclidean distance for n-dimensional vectors

$$d(\vec{v}, \vec{w}) = \sqrt{\sum_{i=1}^{n}(v_i - w_i)^2} \tag{7}$$

It is basically the length of straight line joining two points. Also called as the **norm** of the difference between two vectors.

4.2 Cosine Similarity

There are certain drawbacks with the euclidean distance. It doesn't properly classify corpus related to each other. Cosine of the angle between two vectors is observed to be more accurate as shown in the figure.

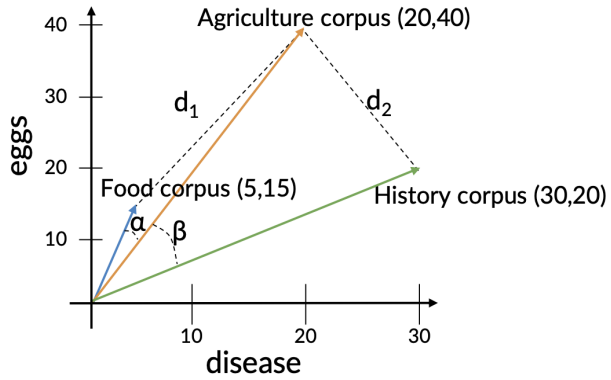


Figure 6: Cosine similarity

Euclidean distance : $d_2 > d_1$
Angles : $\alpha < \beta$

$$\cos\theta = \frac{\vec{v} \cdot \vec{w}}{||\vec{v}||||\vec{w}||} \tag{8}$$

The cosine similarity idea can be used to find words with similar relations. For example: India : Delhi :: Japan : ?
In vector language: $\vec{Delhi} - \vec{India} = \vec{?} - \vec{Japan}$. In a given corpus, the maximum value of the cosine similarity between the unknown word and $\vec{Delhi} - \vec{India} + \vec{Japan}$ would give the related word.

5 Minimum Edit Distance - Auto-correct

Autocorrect is one of the applications of NLP which is widely used in everyday life. For example, autocorrect for graffe can be giraffe, grail or graf. We observe that each of these corrections involve a certain number of edits. **Edit Distance** gives us a way to quantify both of these intuitions about string similarity. The **minimum edit distance** between two strings is defined as the minimum number of editing operations needed to transform one string to another.

5.1 Edit operations

- Insert (add a letter) ⇒ 'to' : 'top', 'two'
- Delete (remove a letter) ⇒ 'hat' : 'ha', 'at'
- Switch (swap two adjacent letters) ⇒ 'eta': 'eat'
- Replace (change one letter to another) ⇒ 'jaw':'jar'

5.2 Finding the minimum distance

Given two strings, the source string X of length n, and target string Y of length m, $D[i,j]$ is defined as the edit distance between $X[1...i]$ and $Y[1...j]$ i.e the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus $D[n,m]$.Dynamic programming is used to compute $D[n,m]$ bottom up, combining solutions to subproblems.
In the base case, with a source substring of length i but an empty target string, going from i characters to 0 requires i deletes. With a target substring of length j but an empty source going from 0 characters to j characters requires j inserts. Having computed $D[i,j]$ for small i,j we then compute larger $D[i,j]$ based on previously computed smaller values. The value of $D[i,j]$ is computed by taking the minimum of the three possible paths through the matrix which arrive there:

$$D[i,j] = min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 2; \text{ if } source[i] \neq target[j] \\ 0; \text{ if } source[i] = target[j] \end{cases} \end{cases} \tag{9}$$

The assumptions involve insertion and deletion having cost of 1 and substitutions with a cost of 2. An example of the matrix $D[i,j]$ (# represents empty string):

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	l	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4

6 Part of Speech Tagging

It involves tagging of words with parts of speech (noun, adverb, adjective etc.). The applications involve:

- Named entities
- Co-reference resolution
- Speech recognition

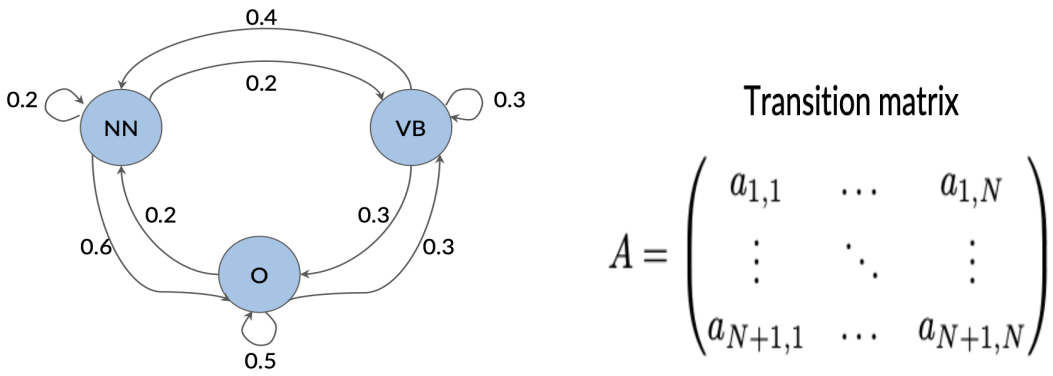


Figure 7: Transition matrix

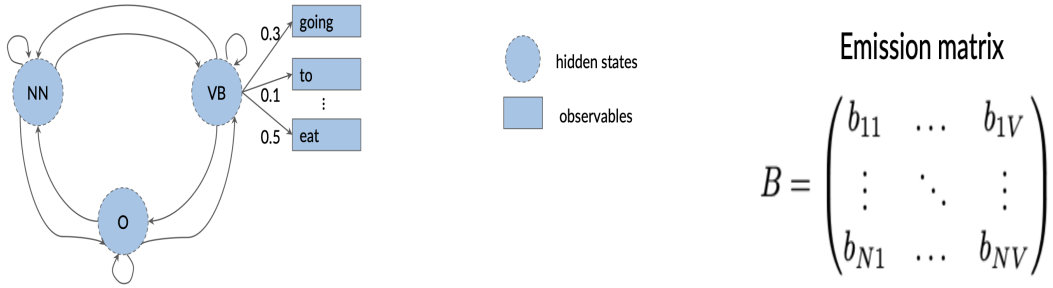


Figure 8: Emission matrix

6.1 Markov Chains

A Markov chain or Markov process is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. POS tags are represented as states with transition probabilities as shown in the figure 7.

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)} \tag{10}$$

These probabilities are used to create the transition matrix. where $C(t_{i-1}, t_i)$ represents the count of occurences of the tag pairs. Smoothing is applied to these probabilities in similar fashion as in case of Naive Bayes.

Emission Probabilities

This is used to create the emission matrix. The transition matrix denotes the probability for going from one hidden state to another (Tags).

6.2 The Viterbi Algorithm

As an instance of dynamic programming, Viterbi resembles the dynamic programming minimum edit distance algorithm. The Viterbi algorithm first sets up a probability matrix or lattice, with one column for each observation o_t and one row for each state in the state graph. Each column thus has a cell for each state q_i in the single combined automaton.
Each cell of the lattice, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_1, ..., q_{t1}$, given the HMM (Hidden Markov Model) λ . The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability:

$$v_t(j) = \max_{\{q_1 \cdots q_{t-1}\}} P(q_1 ... q_{t-1}, o_1 ... o_t, q_t = j|\lambda) \tag{11}$$

We represent the most probable path by taking the maximum over all possible previous state sequences $\max_{\{q_1 \cdots q_{t-1}\}}$. Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t-1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the value $v_t(j)$ is computed as:

$$v_t(j) = \max_{\{i=1\}}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \tag{12}$$

where $v_{t-1}(i)$ denote the previous Viterbi path probability from the previous step, a_{ij} is the transition probability from previous state q_i to current state q_j and $b_j(o_t)$ is the state observation likelihood of the observation symbol o_t given the current state j.

7 N-grams and Probabilities

An N-gram is a sequence of N words. Lets see some starter grams. Corpus: 'I am happy because I am learning'. Size of corpus m = 7.

Unigram Probability: $P(w) = \frac{C(w)}{m}$
Bigram Probability: $P(y|x) = \frac{C(xy)}{C(x)}$
Trigram Probability: P(happy | I am) $= \frac{C(Iamhappy)}{C(Iam)} = 1/2 \Rightarrow P(w_3|w_1^2) = \frac{C(w_1^2 w_3)}{C(w_1^2)}$
Probability of N-gram: $P(w_N|w_1^N - 1) = \frac{C(w_1^{N-1} w_N)}{C(w_1^{N-1})}$
where C is the count.
The assumption that the probability of a word depends only on the previous word is

called a **Markov assumption**. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the n-gram (which looks $n1$ words into the past) as in the N-gram probability above.

7.1 Starting and Ending sentences

Each sentence need to be augmented with a special symbol $< s >$ at the beginning of the sentence, to give us the bigram context of the first word. We also need a special end-symbol $< /s >$. An example for bigram is shown in the figure.

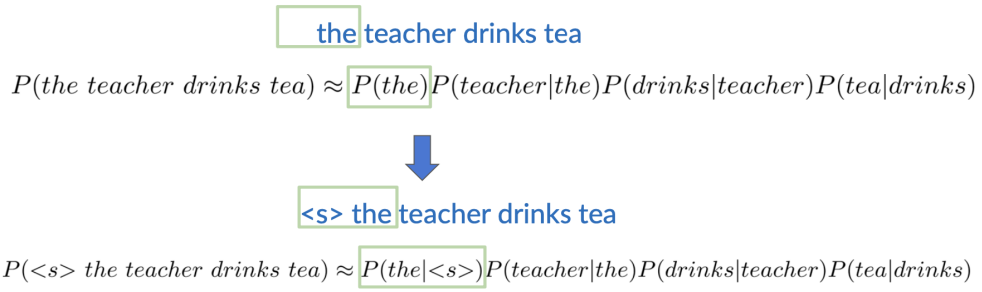


Figure 9: Bigram Updation

N-gram model: add $N - 1$ start tokens $< s >$

7.2 Model Outline

Count Matrix

The rows and columns each contain unique corpus words along with start tokens.

Probability Matrix

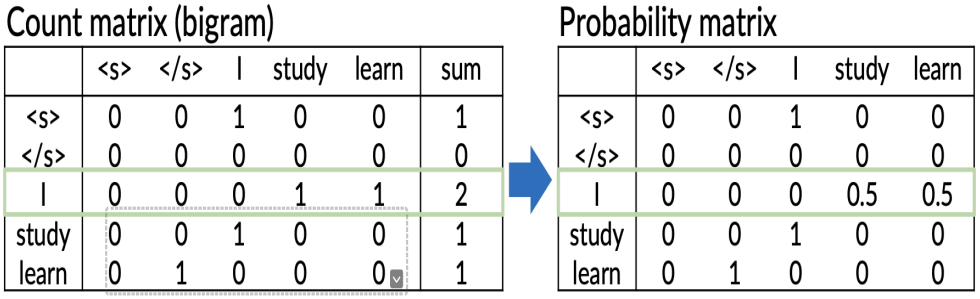
Obtained by dividing each cell by its row sum. For example: Let corpus be 'I study I learn'. An example of the matrix is shown in Figure 10.

Language Model

Let's say we want to find the probability of the sentence: $< s >$ I learn $< /s >$

$$P(sentence) = P(I| < s >)P(learn|I)P(< /s > |learn) = 1 * 0.5 * 1 = 0.5$$

For N-grams as N becomes larger the probability values tend to shrink. Therefore, log of probabilities is taken as it is easier for machines to carry out calculations with larger numbers.



	<s>	</s>	I	study	learn	sum
<s>	0	0	1	0	0	1
</s>	0	0	0	0	0	0
I	0	0	0	1	1	2
study	0	0	1	0	0	1
learn	0	1	0	0	0	1

	<s>	</s>	I	study	learn
<s>	0	0	1	0	0
</s>	0	0	0	0	0
I	0	0	0	0.5	0.5
study	0	0	1	0	0
learn	0	1	0	0	0

Figure 10: Bigram Count and Probability Matrix

7.3 Evaluating Language Models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called **extrinsic evaluation**.
Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model. An **intrinsic evaluation** metric is one that measures the quality of a model independent of any application.
For an intrinsic evaluation of a language model we need a **test set**. As with many of the statistical models in our field, the probabilities of an n-gram model come from the corpus it is trained on, the training set or training corpus. We can then measure the quality of an n-gram model by its performance on some unseen data called the test set or test corpus. Given two probabilistic models, the better model is the one that has a tighter fit to the test data or that better predicts the details of the test data, and hence will assign a higher probability to the test data.
But sometimes, we let the training set get induce into the test set. This introduces a bias that makes probabilities look high and huge inaccuracies in perplexity.

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 PP for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1 w_2 ... w_N, :$

$$\sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i|w_{i-1})}} \tag{13}$$

Perplexity can also be mathematically taken with logarithm. Smaller is the perplexity, better is the model.

7.4 Out of Vocabulary Words

In some cases, we deal with words, we haven't seen before which we call **unknown words** or **out of vocabulary**. An open vocabulary system is one in which we model these potential unknown words in the test set by adding a pseudo-word called $< UNK >$.

Common ways to train the probabilities of the unknown word model:

- Choose a vocabulary (word list) that is fixed in advance.
- Convert in the training set any word that is not in this set (any OOV word) to the unknown word token $< UNK >$ in a text normalization step.
- Estimate the probabilities for $< UNK >$ from its counts just like any other regular word in the training set.

8 PoA

PoA: So far, I have been able to complete all of the topics under Classification and Vector spaces and three of the four topics under Probabilistic Models. The report majorly consists of applications of NLP like auto-correct, sentiment analysis supported by theory from various domains such as vector mathematics, dynamic programming, machine learning and probability. Every topic has been provided explanation through proper visualization through figures. I have also started working over the hobby project mentioned in the PoA and due to this reason, I am behind by the expected schedule by less than week. Since, I have done some of the work of project now, it seems that everything is right on track.

Modified Plan of Action:
Week 5: CBOW Models and different kinds of word embeddings
Week 6 + Week 7 : Attention Models
Week 8 : Hobby Project and completion of report

References

1. Daniel Jurafsky and James H. Martin. Speech and Language Processing
2. DeepLearning.AI, NLP Specialisation Course
3. DeepLearning.AI, Sequence Models
4. https://stackoverflow.com/
5. Wikipedia.org