INTRODUCTION TO
NATURAL LANGUAGE PROCESSING

CLASS NOTES

Mahanth Yalla
M. Tech-AI, IISc

Preface

These notes are based on the lectures delivered by **Prof. Danish Pruthi** in the course **DS 207 - Introduction to Natural Language Processing** at Indian Institute of Science (IISc) Bengaluru - Jan Semester 2025. The notes are intended to be a concise summary of the lectures and are not meant to be a replacement for the lectures.

Disclaimer

These notes are not official and may contain errors. Please refer to the official course material for accurate information.

"I cannot guarantee the correctness of these notes. Please use them at your own risk".

- Mahanth Yalla

Contribution

If you find any errors or have any suggestions, please feel free to open an issue or a pull request on this GitHub repository, I will be happy to incorporate them.

Feedback

If you have any feedback or suggestions on the notes, please feel free to reach out to me via social media or through mail mahanthyalla [at] {iisc [dot] ac [dot] in , gmail [dot] com}.

Acknowledgements: I would like to thank Prof. Danish Pruthi for delivering the lectures and providing the course material. I would like to thank Stanford University for providing the cs224n - Natural Language Processing with Deep Learning course material which was used as a reference for these notes. I would also like to thank the TAs for their help and support.

Contents

Chapter 1

Text Classification

1.1 Introduction

Text classification is a supervised learning task where the goal is to assign a label to a given text. The text can be a document, a sentence, or a paragraph. The labels can be binary or multi-class. Text classification is a fundamental task in natural language processing (NLP) and has many applications such as spam detection, sentiment analysis, topic classification, etc.

1.2 Pre - Distributions

1.2.1 Bernoulli Distribution

The Bernoulli distribution is a discrete probability distribution that models the probability of success of a binary outcome.

$$\text{Bernoulli}(p) = \begin{cases} \text{sucess}, & \text{with probability } p \\ \text{failure}, & \text{with probability } 1-p \end{cases}$$

Lets consider a case when we repeated for n trails of Bernoulli with probability of success being p, and if we observed x wins and $n-x$ losses, then Bernoulli distribution is given by

$$\text{Bernoulli}(p; n, x) = p^x(1-p)^{n-x}$$

Question 1

Now what value of p should we choose to maximize the likelihood of the data?

Solution: We can formualte this into an optimization problem as

$$\arg\max_p \text{Bernoulli}(p; n, x) = p^x(1-p)^{n-x}$$

we can rule out p = 1 and p = 0 from the above equation, p $\in (0, 1)$

$$\begin{aligned} \nabla_p \text{Bernoulli}(p; n, x) &= 0 \\ \frac{\partial}{\partial p}(p^x \left(1-p\right)^{n-x}) &= 0 \\ p^{x-1}\left(x - n\,p\right)\left(1-p\right)^{n-x-1} &= 0 \\ x &= np \qquad (\because p \neq 0, p \neq 1) \\ p &= \frac{x}{n} \end{aligned}$$

Hence the value of p that maximizes the likelihood of the data is $\frac{x}{n}$

Example 1.2.1 (Bernoulli Distribution)

A coin is tossed 10 times and it lands heads 7 times.

Then the probability that maximizes the likelihood of the data is $\frac{7}{10}$

1.2.2 Categorial Distribution

similar to Bernoulli distribution, Categorial distribution is a generalization of Bernoulli distribution.
Say we have N possible outcomes, then the probability of each outcome is given by $p_1, p_2, \cdots, p_N$ such that $\sum_{i=1}^{N} p_i = 1$.

we can estimate the probability of each outcome by counting the number of times each outcome occurs and dividing by the total number of outcomes.

1.2.3 Binomial Distribution

The binomial distribution is a generalization of the Bernoulli distribution.

$$\text{Binomial}(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

1.2.4 Multinomial Distribution

The multinomial distribution is a generalization of the binomial distribution to more than two categories. (vaguely speaking, Binomial + Categorical = Multinomial).

1.3 Text (Topic) Classification

1.3.1 Problem Statement

Given a text document, the goal is to assign a label to the document. The labels can be binary or multi-class.

1.3.2 Example Data

Binary (2 - Class) Classification Dataset :

Text	Label
I love this movie	Positive
I hate this movie	Negative
I like this movie	Positive
I dislike this movie	Negative

M - Class Classification Dataset :

Text	Label
Kohli scores another century	Sports
Scam in the banking sector	Finance
India wins the match	Sports
Amitab Bachan praises Allu Arjun for his performance	Entertainment
Stock market crashes	Finance
He announces a metro project	Politics
Pushpa 2 is an Industry hit	Entertainment
Pawan Kalyan to contest in the upcoming elections	Politics
SS Rajamouli comes from Dubai to vote	Politics

1.3.3 Modeling data distribution

Joint probability of the text (X) and the label (y) is given by

$$f_{X,Y}(x, y) = f_{Y|X}(y|x)f_X(x) = f_{X|Y}(x|y)f_Y(y)$$

Example 1.3.1 (M class classification)
Consider the M class classification dataset from above.

Then the joint probability of the text and the label is given by

$f_{X,Y}(x = \text{"Amitab Bachan praises Allu Arjun for his performance"}, y = \text{"Entertainment"}) = 0.7654$ (say)

similarly,

$f_{X,Y}(x = \text{"Amitab Bachan praises Allu Arjun for his performance"}, y = \text{"Politics"}) = 0.1544$ (say)
$f_{X,Y}(x = \text{"Amitab Bachan praises Allu Arjun for his performance"}, y = \text{"Cricket"}) = 0.0023$ (say)
$f_{X,Y}(x = \text{"Amitab Bachan praises Allu Arjun for his performance"}, y = \text{"Politics"}) = 0.0779$ (say)

1.4 Naive Bayes Classifier

Assumption: The features are IIDs (Independent and Identically Distributed).
Hence the joint probability of the text (words x_i) and the label is given by

$$f_{X,Y}(x, y) = \prod_{i=1}^{N} f_{X_i,Y}(x_i, y)$$

where, N is the number of examples in the dataset.

Theorem 1.4.1 Bayes Theorem
Bayes' theorem is stated mathematically as the following equation:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_y P(x|y)P(y)}$$

similarly, for continuous random variables, the theorem is stated as:

$$f_{X|Y=y}(x) = \frac{f_{Y|X=x}(y)f_X(x)}{f_Y(y)}.$$

1.4.1 Generative Naive Bayes Classifier

Using Naive Bayes Classifier, we can model the **joint probability** (Generative Model) of the text and the label.
The joint probability of the text and the label is given by

$$P(X, y) = P(X|y)P(y)$$

since we assumed iids, we can write the probability of the text given the label as

$$P(X_i|y_k) = \prod_{j=1}^{m_i} P(X_{ij}|y_k)$$

where,
X_{ij} is the j^{th} word of the i^{th} example and,

m_i is the number of words in X_i.

Thus,

$$P(X, y) = P(y_k) \prod_{j=1}^{m_i} P(X_{ij}|y_k)$$

We can estimate the parameters of the model using the training data and use the model to predict the label of the text. As,

- $P(y_k)$ is the prior probability of the label y_k

$$P(y_k) = \frac{\text{number of examples with label} y_k}{\text{total number of examples}}$$

- $P(X_{ij}|y_k)$ is the likelihood of the word X_{ij} given the label y_k

$$P(X_{ij}|y_k) = \frac{\text{number of times word } w_i \text{ occurs in examples with label } y_k}{\text{total number of words in examples with label } y_k}$$

Inference:

Now, using bayes theorem, we can write the probability of the label(y_k) given the text (X_i) as

$$P(y_k|X_i) = \frac{P(X_i|y_k)P(y_k)}{P(X_i)} = \frac{P(y_k)\prod_{j=1}^{m_i} P(X_{ij}|y_k)}{P(X_i)}$$

Note:-

- $P(y|X) = \frac{P(X|y)P(y)}{P(X)}$ is inferred as posteriror $= \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$
- The denominator $P(X)$ (i.e, *evidence*) is constant for all the classes, hence we can ignore it while calculating the probability of the label given the text.
- Thus, The probability of the label given the text is proportional to the product of the probabilities of the features given the label.

Thus, the probability of the label given the text is given by

$$P(y_k|X_i) \propto P(y_k) \prod_{j=1}^{m_i} P(X_{ij}|y_k)$$

where,

$P(y_k)$ is the prior probability of the label y_k and,
$P(X_{ij}|y_k)$ is the likelihood of the word X_{ij} given the label y_k.

Example 1.4.1 (Consider the binary classification dataset from above)

Then the probability of the label given the text is given by

$$P(\text{Positive}|\text{"I love this movie"}) \propto P(\text{Positive}) \times \prod_{\text{words} \in \text{text}} P(\text{word}_i|\text{Positive})$$

$$P(\text{Negative}|\text{"I love this movie"}) \propto P(\text{Negative}) \times \prod_{\text{words} \in \text{text}} P(\text{word}_i|\text{Negative})$$

for postive class,

$$P(\text{Positive}|\text{"I love this movie"}) \propto P(\text{Positive}) \times P(\text{"I"}|\text{Positive}) \times P(\text{"love"}|\text{Positive}) \times P(\text{"this"}|\text{Positive}) \times P(\text{"movie"}|\text{Positive})$$

Generating Samples using Generative Naive Bayes

Note:-

Implementation done in notebook

Algorithm 1: Generative Naive Bayes

```
word ← ""
foreach k in 1..N do
    y_k ← Categorial(μ)
    word += Multinomial(θ_{y_k})
end
words = word.concat()
return words
```

Estimation of the parameters

parameters :

- μ : prior probability of the label

$$\mu = \frac{\text{number of examples with label } y_k}{\text{total number of examples}} = P(y_k)$$

- θ_{y_k} : likelihood of the word given the label

$$\theta_{y_k} = \frac{\text{number of times word } w_i \text{ occurs in examples with label } y_k}{\text{total number of words in examples with label } y_k} = P(X_{ij}, y_k)$$

$$\begin{aligned} \text{parameters} &= |\mu| + |\theta_{y_k}| \\ &= \mathbb{k} + \mathbb{k} \times \mathbb{v} \end{aligned}$$

where,

$\mathbb{k}$ is the number of categories [a.k.a classes] and,
$\mathbb{v}$ is the number of unique words in the dataset [a.k.a vocab size].

parameters = $O(\mathbb{k}\mathbb{v})$

1.4.2 Discriminative Naive Bayes Classifier

In the discriminative model, we directly model the **conditional probability** of the label given the text, avoiding the need to model the joint probability of the text and the label.

$$P(y_k|X_i) \propto P(y_k) \prod_{j=1}^{m_i} P(X_{ij}|y_k)$$

treating $P(y_k|X_i)$ as primary object, we optimise the parameters of the model to maximize the likelihood of the data.

Loss Function : Negative Log Likelihood

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \log P(y_k|X_i) = -\sum_{i=1}^{N} \log P(y_k) + \sum_{i=1}^{N} \sum_{j=1}^{m_i} \log P(X_{ij}|y_k)$$

Optimization : Gradient Descent

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta)$$

Inference in Discriminative Naive Bayes

Given a text, calculate the probability of the label given the text using the generative Naive Bayes model. Assign the label with the highest probability to the text.

$$\hat{y} = \arg\max_{y_k} P(y_k|X)$$

where, $\hat{y}$ is the predicted label, X is the text, y_k is the label and, $P(y_k|X) = P(y_k)\Pi_{j=1}^{m} P(X_{ij}|y_k)$ Some of the modifications that can be done to the Naive Bayes model are:

- **UNKNOWN words :** If the probability of the label given the text is less than a threshold, then assign the label as "Unknown".
- **Smoothing :** Add a small value ($\alpha > 0$) to the likelihood to avoid zero probabilities and Normalize.
- **Threshold :** A threshold can be set based on the validation set and Unknown can be used for unseen words as well.

1.4.3 Implementation of Naive Bayes

Algorithm 2: Naive Bayes Classifier

Input: X,y,α
Output: $\log \mu, \log \theta$

$\mu \leftarrow$ zeros($\mathbb{k}$)
$\theta \leftarrow$ zeros($\mathbb{k}, \mathbb{v}$)
foreach *example in X* **do**
  foreach *word in example* **do**
    $\theta[y_k][word] \leftarrow \theta[y_k][word] + 1$
  end
  $\mu[y_k] \leftarrow \mu[y_k] + 1$
end
foreach y_k *in* y **do**
  $\mu[y_k] \leftarrow \mu[y_k] / \sum_{y_k} \mu[y_k]$
end
foreach y_k *in* y **do**
  foreach *word in* $\theta[y_k]$ **do**
    $\theta[y_k][word] \leftarrow (\theta[y_k][word] + \alpha) / (\sum_{word} \theta[y_k][word] + \alpha \times \mathbb{v})$
  end
end
return $\log \mu, \log \theta$

Algorithm 3: Naive Bayes Predict

Input: X,$\log \mu, \log \theta$
Output: y

foreach *example in X* **do**
  foreach *label in y* **do**
    $P \leftarrow \log \mu[y_k]$
    foreach *word in example* **do**
      $P \leftarrow P + \log \theta[y_k][word]$
    end
    $P_{y_k} \leftarrow P$
  end
  $y \leftarrow \arg\max_{y_k} P_{y_k}$
end
return y

Note:-

Implementation of Naive Bayes is done in the notebook.

- Train Accuracy : 0.9362
- Test Accuracy : 0.6210

which implies the Naive Bayes model is over fitting the training data or the data is insufficient to generalize.

The Algorithm 2 (Naive Bayes Fit) and Algorithm 3 (Naive Bayes Predict) are implemented in the python file **naiveBayes.py**.

The Naive Bayes model is trained on the twitter training data and achieved a score of **0.6632** on test data, which is when same compared to sklearn's Implementation *(details in notebook)* .

1.5 Generative vs Discriminative Model

To summarize, the difference between the generative and discriminative model is given below:

Generative Model	**Discriminative Model**
Model the joint probability $P(X, Y) = P(X\|Y)P(Y)$	Model the conditional probability $P(Y\|X) = P(X\|Y)P(Y)$
Assumes the features are IIDs	Does not assume the features are IIDs
Learns $P(X\|y)$ and $P(y)$ separately	learns $P(X, y)$ directly by optimization
Use : Can generative new samples	*Use :* Can discriminate samples
More complex (Eg. Naive Bayes)	Less complex (Eg. Logistic Regression)

Table 1.1: Generative vs Discriminative Model

Chapter 2

Word Representations

2.1 Introduction

Words are the basic building blocks of any language, and are the smallest unit of meaning. Words are represented in the form of **vectors** in NLP. Word representations are used in various NLP tasks like text classification, machine translation, sentiment analysis, etc.

2.2 Word2vec - Skip-gram model

The Word2vec model is a popular model used for learning word embeddings, published in NIPS 2013 paper by Mikolov et al.[1] [2]

Word2vec is a shallow neural network model that is trained to reconstruct linguistic contexts of words. Word2vec is trained on a large corpus of text data and learns to predict the context of a word given its neighboring words. Word2vec is trained using two algorithms:

Key idea:

- grouping similar words,
- Use a large corpus of text
- For center word and context ("outside") words be
- Use the similarity of word vectors and to compute the probability of the word being used in the context (and vice versa)
- Keep adjusting (aka learning) the word vectors to optimize the probability

2.2.1 Skip-gram model

The Skip-gram model is trained to predict the context words given a center word.

Likelihood :

$$\begin{aligned} \mathrm{L}(\theta) &= \prod_{t=1}^{T} P(\text{context}|\text{center};\theta) \\ &= \prod_{t=1}^{T} \prod_{-m\leqslant j\leqslant m, j\neq 0} P(w_{t+j}|w_t;\theta) \end{aligned}$$

we can learn the parameters θ by minimizing the loss, ie maximizing the likelihood of the context words given the center words.

Loss Function: We learn to minimize the cross entropy loss objective with the true distribution

$$\min_{\theta} \mathbb{E}\left[\text{Loss}; \theta)\right]$$

We use the negative log likelihood as loss function (empirical), i.e.,

$$\begin{aligned}
\text{J}(\theta) &= -\frac{1}{T} log L(\theta) \\
&= -\frac{1}{T} log \left(\prod_{t=1}^{T} \prod_{-m \leqslant j \leqslant m, j \neq 0} P(w_{t+j} | w_t; \theta) \right) \\
&= -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leqslant j \leqslant m, j \neq 0} \log p(w_{t+j} | w_t; \theta)
\end{aligned}$$

we can learn the parameters θ by minimizing the loss, ie maximizing the likelihood of the context words given the center words.

Objective : find word representations that are useful for predicting the surrounding words in a sentence or a document by minimizing the loss function $\text{J}(\theta)$, hence maximizing the likelihood of the context words given the center words.

$$\begin{aligned}
&\min_{\theta} \quad \text{J}(\theta) \\
&\max_{\theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leqslant j \leqslant m, j \neq 0} \log p(w_{t+j} | w_t; \theta)
\end{aligned}$$

where, w_t is the center word, w_{t+j} is the context word, m is the context window size, T is the total number of words in the corpus, θ are the parameters of the model and

The probability $p(w_{t+j} | w_t; \theta)$ is computed using the softmax function (in basic Skip gram model) as

$$p(w_{t+j} | w_t; \theta) = \frac{e^{v_{w_{t+j}}^T v_{w_t}}}{\sum_{w=1}^{\mathbb{v}} e^{v_w^T v_{w_t}}}$$

where v_{w_t} is the vector representation of the center word w_t, $v_{w_{t+j}}$ is the vector representation of the context word w_{t+j}, and $\mathbb{v}$ is the size of the vocabulary.

[*Notation note : I am using* ***claim*** *notation to show some interpretation and facts, but are not mathematical claims.*]

Claim 2.2.1 softmax function interpretation

we need similarity between the center word and context word vectors to be high, and as cosine similarity is a measure of similarity between two non-zero vectors, we can use the dot product of the two vectors as a measure of similarity.

$$\begin{aligned}
\text{cosine similarity}(a, b) &= \frac{a \cdot b}{\|a\| \|b\|} \\
\text{similarity}(a, b) &\propto a \cdot b \\
\text{similarity}(v_{w_{t+j}}, v_{w_t}) &= v_{w_{t+j}}^T v_{w_t}
\end{aligned}$$

and we normalize the similarity scores, thus

$$p(w_{t+j} | w_t; \theta) = \frac{e^{v_{w_{t+j}}^T v_{w_t}}}{\sum_{w=1}^{\mathbb{v}} e^{v_w^T v_{w_t}}}$$

where, v_{w_t} is the vector representation of the center word w_t, $v_{w_{t+j}}$ is the vector representation of the context word w_{t+j}, and $\mathbb{v}$ is the size of the vocabulary.

It can be interpreted as the probability of the context (outer) word (o) given the center word (c) is proportional to the similarity between the two word vectors.

$$p(o|c) = \frac{e^{u_o^T v_c}}{\sum_{w \in \mathbb{v}} e^{u_w^T v_c}}$$

where parameters are

$$\theta = \begin{bmatrix} V \\ U \end{bmatrix} \in \mathbb{R}^{(2\mathbb{v}) \times d}$$

where, V is the matrix of center word vectors and U is the matrix of context word vectors, $V, U \in \mathbb{R}^{\mathbb{v} \times d}$.

Derivative of the loss function

The partial derivative of the loss function with respect to the center word vector v_c is given by

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial v_c} &= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(\frac{\partial}{\partial v_c} \log p(w_o | w_c; \theta) \right) \\
&= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(\frac{\partial}{\partial v_c} \log \frac{e^{u_o^T v_c}}{\sum_{w \in \mathbb{v}} e^{v_w^T v_c}} \right) \\
&= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(\frac{\partial}{\partial v_c} \left(u_o^T v_c - \log \sum_{w \in \mathbb{v}} e^{v_w^T v_c} \right) \right) \\
&= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(u_o - \frac{\sum_{w \in \mathbb{v}} e^{v_w^T v_c} u_w}{\sum_{w \in \mathbb{v}} e^{v_w^T v_c}} \right)
\end{aligned}$$

rewriting,

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial v_c} &= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(u_o - \frac{\sum_{w \in \mathbb{v}} e^{v_w^T v_c} u_w}{\sum_{w \in \mathbb{v}} e^{v_w^T v_c}} \right) \\
&= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(u_o - \sum_{w \in \mathbb{v}} \left(\frac{e^{v_w^T v_c}}{\sum_{w \in \mathbb{v}} e^{v_w^T v_c}} \right) u_w \right) \\
&= -\frac{1}{T} \sum_{c \in \mathbb{v}} \sum_{o \in \text{outer}} \left(u_o - \sum_{w \in \mathbb{v}} p(o|c) u_w \right)
\end{aligned}$$

interpretation : The derivative of the loss function with respect to the center word vector v_c is the difference between *observed context word vectors* (u_o) and the *expectation of context word vectors* $\left(\sum_{w \in \mathbb{v}} p(o|c) \cdot u_w \right)$.

Note:-

The above softmax function is computationally expensive to compute, as it involves computing the exponential of the dot product of the center word and context word vectors for all words in the vocabulary.

Complexity : $O(\mathbb{v})$

[*Notation note : I am using* ***claim*** *notation to show the results form papers form now on, but are not mathematical claims.*]

Claim 2.2.2 Skip-gram model Results [2]

evaluating the Hierarchical Softmax (HS), Noise Contrastive Estimation(NCE), Negative Sampling (NS), and sub-sampling of the training words resulted :

- Negative Sampling outperforms the Hierarchical Softmax and has even slightly better performance than the Noise Contrastive Estimation.
- Sub-sampling of the training words improves the training speed several times and makes the word representations significantly more accurate.

2.2.2 Negative Sampling

Definition 2.2.1: NEG Sampling

Negative Sampling is a technique used to approximate the softmax function in the Skip-gram model. Instead of computing the probability of all words in the vocabulary, Negative Sampling samples a small number of negative examples (words that are not in the context) and computes the probability of the context word given the center word and the negative examples.

Objective : To distinguish the context words from the negative examples.

$$\text{NEG} = \log p(w_{t+j}|w_t;\theta) + k\mathbb{E}_{w\sim P_n(w)}\left[\log p(w|w_t;\theta)\right]$$

where, k is the number of negative samples, $P_n(w)$ is the noise distribution, and $\mathbb{E}_{w\sim P_n(w)}$ is the expectation over the noise distribution.
which replaces every $logp(w_{t+j}|w_t;\theta)$ term in the loss function with the NEG term for Negative Sampling.

$$\max_\theta \sum_{t=1}^{T} \sum_{-m\leqslant j\leqslant m, j\neq 0} \log p(w_{t+j}|w_t;\theta) + k\mathbb{E}_{w\sim P_n(w)}\left[\log p(w|w_t;\theta)\right]$$

Claim 2.2.3 Negative Sampling [2]

- Best k is observed to be 5-20 for small datasets and 2-5 for large datasets.
- The noise distribution $P_n(w)$ is chosen to be the unigram distribution raised to the power of $\frac{3}{4}$ (0.75).

$$P_n(w) \sim U(w)^{\frac{3}{4}}$$

2.2.3 Skip gram with Negative Sampling

In Skip gram, we now use sigmoid function to compute the probability of the context word given the center word and the negative examples. The probability of the context word given the center word is given by

$$\arg\max_\theta \sum_{t=1}^{T} \sum_{-m\leqslant j\leqslant m, j\neq 0} \log p(w_{t+j}|w_t;\theta) + k\mathbb{E}_{w\sim P_n(w)}\left[\log p(w|w_t;\theta)\right]$$

now can be written as

$$\arg\max_{\theta} \prod_{c,o \in D} P(D = 1|c, o; \theta) \prod_{c,n \in D} P(D = 0|c, n; \theta)$$
$$\arg\max_{\theta} \prod_{c,o \in D} \sigma(v_c^T v_o) \prod_{c,n \in D} (1 - \sigma(v_c^T v_n))$$
$$\arg\max_{\theta} \sum_{c,o \in D} \log(\sigma(v_c^T v_o)) + \sum_{c,n \in D} \log(\sigma(-v_c^T v_n))$$

2.2.4 After training skip gram models

After training the Skip-gram model, we can use the word vectors to compute the similarity between words. The similarity between two words can be computed using

- Euclidean distance

$$\begin{aligned} \text{similarity} &= \frac{1}{1 + \text{Euclidean distance}} \\ &= \frac{1}{1 + \|v_1 - v_2\|} \end{aligned}$$

- Cosine similarity

$$\text{similarity} = \cos(\theta) = \frac{v_1 \cdot v_2}{||v_1|| \cdot ||v_2||}$$

where $\|\cdot\|$ is Euclidean distance , for a vector $v \in \mathbb{R}^n$, Euclidean distance is given by

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

2.2.5 Word2vec implementation

There are several libraries available to train Word2vec models. Some of the popular libraries are:

- Gensim
- TensorFlow
- PyTorch
- FastText
- spaCy
- NLTK
- GloVe

the experiments are done based on gensim model : word2vec-google-news-300 model. which is trained on Google News data, and contains 300-dimensional word vectors for a vocabulary of 3 million words and phrases.

2.2.6 Results

Some of the Analogies are observed by perfoming vector operations on the word vectors, for instance

$$\text{vec('king')} - \text{vec('man')} + \text{vec('queen')} \approx \text{vec('woman')}$$

for some of the analogies the most similar words (based on cosine similarity score) to the vector examples are listed below :

- **king : man :: queen :** woman
- **paris : france :: tokyo :** japan
- **usa : dollar :: india :** rupee
- **usa : english :: india :** hindi

2.3 Word2vec - Bag of Words model

The Bag of Words (BoW) model is a simple model used to represent text data. The BoW model represents a document as a bag of words, ignoring the order of words in the document. The BoW model is used to represent text data in the form of a vector, where each element of the vector represents the frequency of a word in the document.

BoW model

$$\text{BoW} = \begin{bmatrix} f(w_1) \\ f(w_2) \\ \vdots \\ f(w_n) \end{bmatrix}$$

where, $f(w_i)$ is the frequency of the word w_i in the document.

We can use the BoW model to represent a document as a vector, where each element of the vector represents the frequency of a word in the document. The BoW model can be used in text classification, sentiment analysis, etc. which is not a good model for capturing the semantics of words. It is a simple model that ignores the order of words in the document.

2.4 Word2vec - GloVe model

The GloVe (Global Vectors for Word Representation) model is a model used to learn word embeddings. The GloVe model is trained on a large corpus of text data and learns to predict the context of a word given its neighboring words. The GloVe model is trained using the co-occurrence matrix of words in the corpus.

GloVe focused more on ratio of probabilities of co-occurrence of words, rather than the probabilities themselves. They used **log-bilinear** regression model to learn the word vectors,

$$w_i \cdot \tilde{w}_j = \log P(i|j)$$

where, w_i is the vector representation of word i, $\tilde{w}_j$ is the vector representation of word j, b_i and $\tilde{b}_j$ are the biases for words i and j, and P_{ij} is the probability of word j appearing in the context of word i.

Objective function:

$$\sum_{i=1}^{\mathbb{v}} \sum_{j=1}^{\mathbb{v}} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

where, X_{ij} is the number of times word j appears in the context of word i, $f(X_{ij})$ is the weighting function, and $\mathbb{v}$ is the size of the vocabulary.

2.5 Classification

Word representations are used in text classification tasks to represent text data as vectors. Even linear models like Logistic Regression can be used to classify text data using word representations, which yields good results.

Example 2.5.1 (Logistic Regression Text classification)

Using gensim-300-dimension word representations, we can get vector representation of a sentance by averaging the word vectors of the words in the sentence as

$$\text{vec(sentence)} = \frac{1}{n} \sum_{i=1}^{n} \text{vec(word}_i\text{)}$$

where, n is the number of words in the sentence.

The vector representation of the sentence can be used as input to a Logistic Regression model to classify the text data.

On implementing Logistic Regression on Ecommerce dataset, the model achieved an accuracy of 92.92%. (see notebook for implementation details.)

Thus, Word representations can be used to represent text data as vectors, which can be used as input to machine learning models like Logistic Regression for text classification, which captures the semantics of words and yields good results.

Chapter 3

n-gram Language Models

3.1 Introduction

Language models are used to model the probability of a sequence of words. i.e. assigns **probabilities to a sequence of words**.

3.2 Markov Assumption

The probability of a word depends only on the previous $n-1$ words. This is called the n-gram model. The probability of a word w_i given the previous words $w_1, w_2, \ldots, w_{i-1}$ is given by: $P(w_i|w_1, w_2, \ldots, w_{i-1}) = P(w_i|w_{i-(n-1)}, w_{i-(n-2)}, \ldots, w_{i-1})$ The probability of a word w_i given the previous $n-1$ words is given by: $P(w_i|w_{i-(n-1)}, w_{i-(n-2)}, \ldots, w_{i-1}) = \frac{P(w_{i-(n-1)}, w_{i-(n-2)}, \ldots, w_{i-1}, w_i)}{P(w_{i-(n-1)}, w_{i-(n-2)}, \ldots, w_{i-1})}$

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, 2013.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, in Advances in Neural Information Processing Systems, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds., vol. 26, Curran Associates, Inc., 2013.