WAYNE STATE UNIVERSITY

Electrical and Computer Engineering

ECE 7995: ST AI for NLP

Lecture-3: Machine Learning for NLP

Mohammed Alawad, Ph.D.

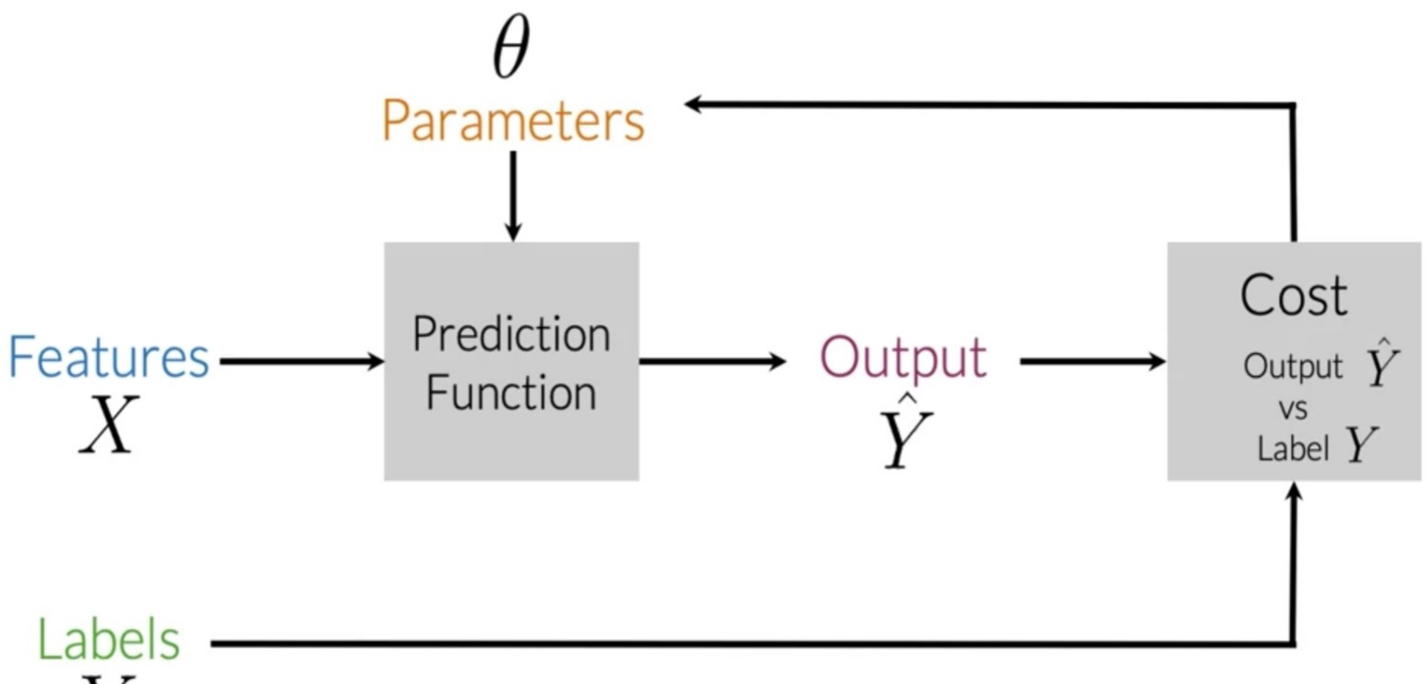
Machine Learning

- **Machine learning** allows computers to learn and infer from data by applying algorithms to observed data and make predictions based on the data.
- **Supervised learning** trains a model with data points that have known outcomes.
  - Classification: outcome is a category
  - Regression: outcome is continuous (numerical)
- **Unsupervised learning** trains a model with data points that have unknown outcomes.
  - Clustering
  - Recommendation

Supervised Learning (classification)

Binary classification: split the data into two categories
Multi-class classification: split the data into more than two categories

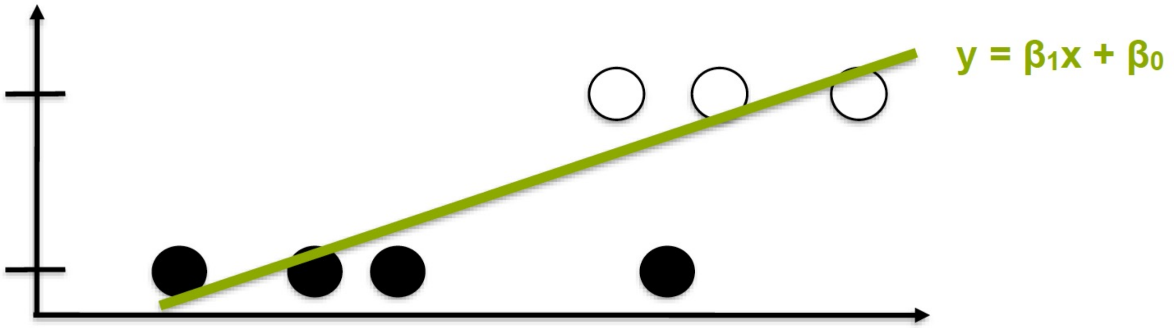
	bias	positive	negative	sentiment
0	1.0	3020.0	61.0	1.0
1	1.0	3573.0	444.0	1.0
2	1.0	3005.0	115.0	1.0
3	1.0	2862.0	4.0	1.0
4	1.0	3119.0	225.0	1.0
5	1.0	2955.0	119.0	1.0
6	1.0	3934.0	538.0	1.0
7	1.0	3162.0	276.0	1.0
8	1.0	628.0	189.0	1.0

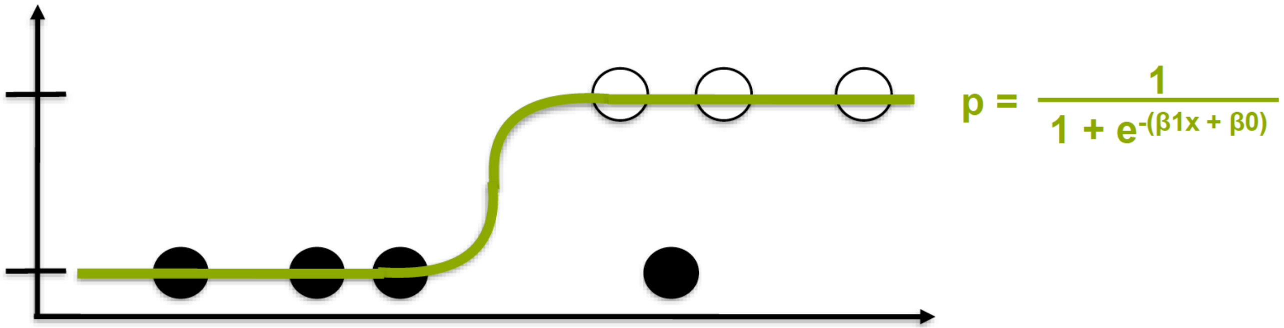


Minimize the cost (the difference between Y and Y^)

Logistic Regression

- One of the most popular machine learning techniques for binary classification
- The most basic regression technique is linear regression
- **Problem:** The y values of the line go from -∞ to +∞
- **Solution:** use the sigmoid function to limit the y values from 0 to 1





Building a Logistic Regression model

1. Prepare the data: Read in labelled data and preprocess the data

2. Split the data: Separate inputs and outputs into a training set and a test set, respectively
    - To avoid overfitting (the model performs well on the training data only)
    - A model is fit on the training data and it is evaluated on the test data
        - Training Set (70-80%)
        - Test Set (20-30%)

Building a Logistic Regression model

3. Numerically encode inputs: Using Count Vectorizer or TF-IDF Vectorizer

4. Fit a model (model training): Fit a model on the training data and apply the fitted model to the test set

Building a Logistic Regression model

5. Evaluate the model: Decide how good the model is by calculating various error metrics

- After fitting a model on the training data and predicting outcomes for the test data, how do you know if the model is a good fit?
- Confusion Matrix
- Error Metrics
  - Accuracy = (TP + TN) / All , where all means TP+FP+FN+TN
  - Precision = TP / (TP + FP)
  - Recall = TP / (TP + FN)
  - F1 Score = 2*(Precision * Recall)/(Precision + Recall)

Example: Sentiment Analysis (classification)

- Tweet: I am happy because I am learning NLP

Positive: 1

Negative: 0

Input Features (X)

LR Model (Classifier)

Positive (Y_pred)

Vocabulary

- **Vocabulary (V):** list of all unique words in a corpus, where words are not repeated
- Example:
- X (tweets): list of tweets, e.g., [“I am happy because I am learning NLP, …, …, “I hated the movie”]

- V = [I, am, happy, because, learning, NLP, ………, hated, the, movie]

Count Vectorizer

Input Text	I am happy because I am learning NLP
Vocabulary	I, am, happy, because, learning, NLP, ………, hated, the, movie
Features	2, 2, 1, 1, 1, 1, ………, 0, 0, 0

Sparse representation because there are a lot of zeros in features

Problems with Sparse Representations

- Number of featured equals to the size of the vocabulary (a few ones and many zeros)
- In this representation a LR will learn n+1 parameters, where n=|V|

I, am, happy, because, learning, NLP, ………, hated, the, movie

2, 2, 1, 1, 1, 1, ………, 0, 0, 0

V

Zeros

- LR Parameters: $[\theta_0, \theta_1, \theta_2, \dots, \theta_n]$ → large training time, and large prediction time

Feature Extraction

- Counts are used as input features to the LR
- For each word in the vocabulary:
  - Positive frequency counts the number of times a word appears in the positive class
  - Negative frequency count the number of times a word appears in the negative class
  - Use these two counts to extract features (input of LR)

- We can also use the following techniques to filter the vocabulary:
  - Max document frequency
  - Min document frequency

Positive and Negative Counts

Data corpus (tweets)	Labels
I am happy because I am learning NLP	positive
I am happy	positive
I am sad, I am not learning NLP	negative
I am sad	negative

Vocabulary	Pos_freq (1)	Neg_freq (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

- Frequencies: dictionary mapping from (word, class) to frequency
- In the table above, you can see how words like happy and sad tend to take clear sides, while other words like "I, am" tend to be more neutral.

Feature Extraction

- Instead of learning V features, we are learning 3 features

- $X_m = [1, \sum_w freq(w,1), \sum_w freq(w,0)\ ]$

- X_m : Features of tweet (m)
- $\sum_w freq(w,1)$: sum of positive frequencies
- $\sum_w freq(w,0)$: sum of negative frequencies

Feature Extraction

- I am happy because I am learning NLP

- $X_m = [1, \sum_w freq(w,1), \sum_w freq(w,0)\ ]$

- $\sum_w freq(w,1)$ = 3+3+2+1+1+1 = 11
- $\sum_w freq(w,0)$ = 3+3+0+0+1+1 = 8

- X_m : [1,11,8]

Vocabulary	Pos_freq (1)	Neg_freq (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

Feature Extraction

- I am sad, I am not learning NLP

- $X_m = [1, \sum_w freq(w,1), \sum_w freq(w,0)\ ]$

- $\sum_w freq(w,1)$ = 3+3+1+1+0+0 = 8
- $\sum_w freq(w,0)$ = 3+3+1+1+2+1 = 11

- X_m : [1,8,11]

Vocabulary	Pos_freq (1)	Neg_freq (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

General Overview

I am happy because I am learning NLP

↓ Preprocessing

[happy, learn, nlp]

↓ Feature Extraction

[1, 4, 2]

1: bias

4: sum of positive frequencies

2: sum of negative frequencies

General Overview

Corpus (tweets)	Cleaning	Feature Extraction
I am happy because I am learning NLP	[happy, learning, nlp]	[1, 40, 20]
I am sad, I am not learning NLP	[sad, not, learning, nlp]	[1, 20, 50]
…	…	…
I am sad	[sad]	[1,5,35]

General Overview

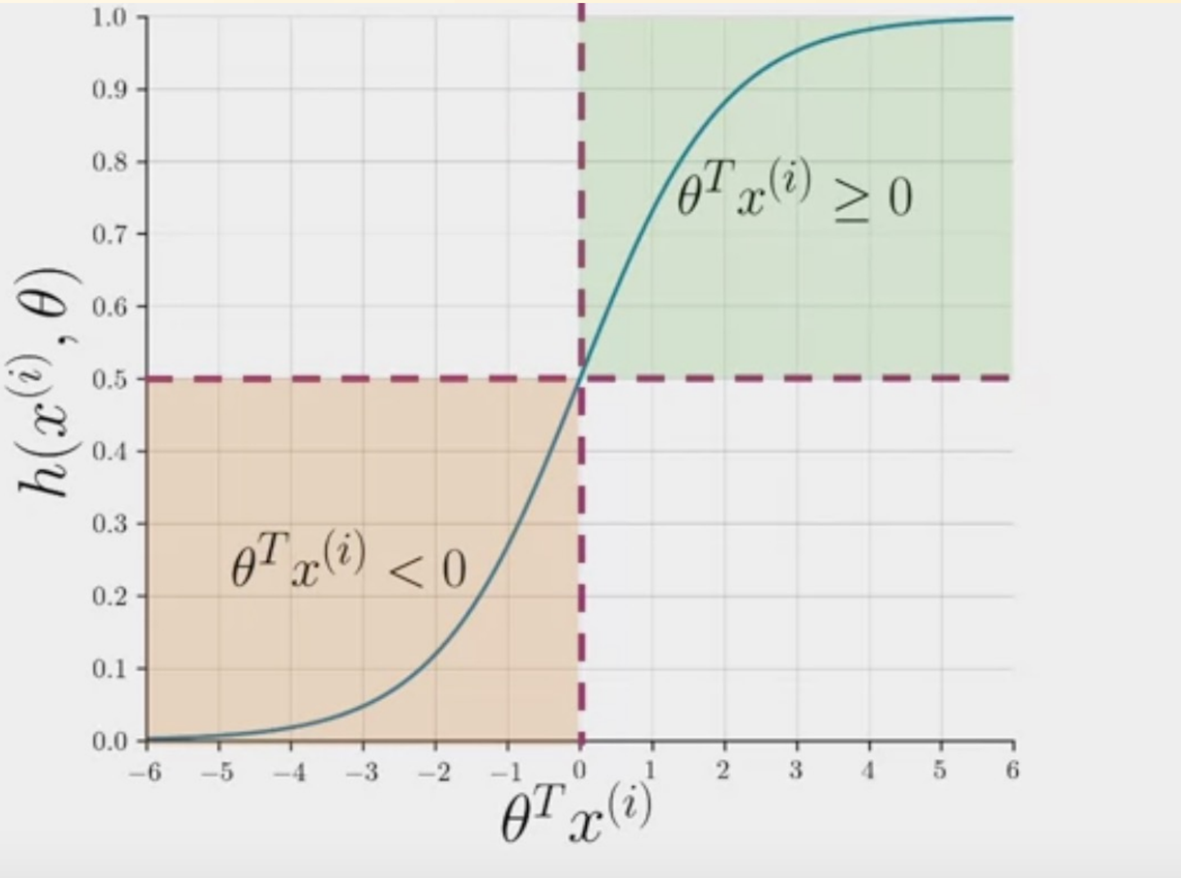
- *m* rows: number of input documents (tweets)
- 3 cols: three features for each tweet

$$\begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix} \longleftrightarrow \begin{array}{c} [[1, 40, 20] \\ [1, 20, 50], \\ \dots \\ [1, 5, 35]] \end{array}$$

Logistic Regression

- $h(x^{(i)}, \theta) = \dfrac{1}{1+ e^{-\theta^T x^{(i)}}}$

- LR uses sigmoid function which has an output between 0 and 1
- $X^{(i)}$: the features of i^{th} tweet
- θ : parameters
- For classification, we need to set a threshold (usually 0.5)



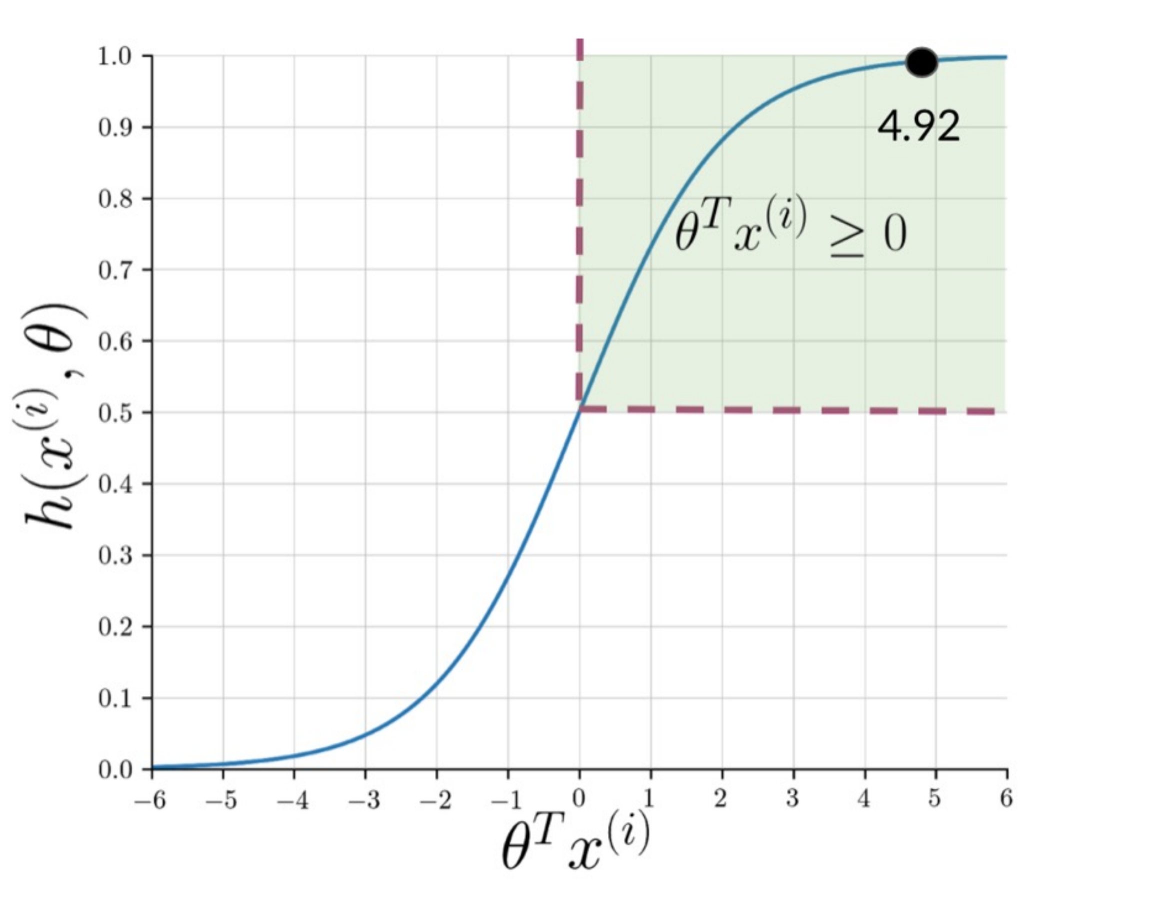
Logistic Regression

- [tune, ai, great, model]

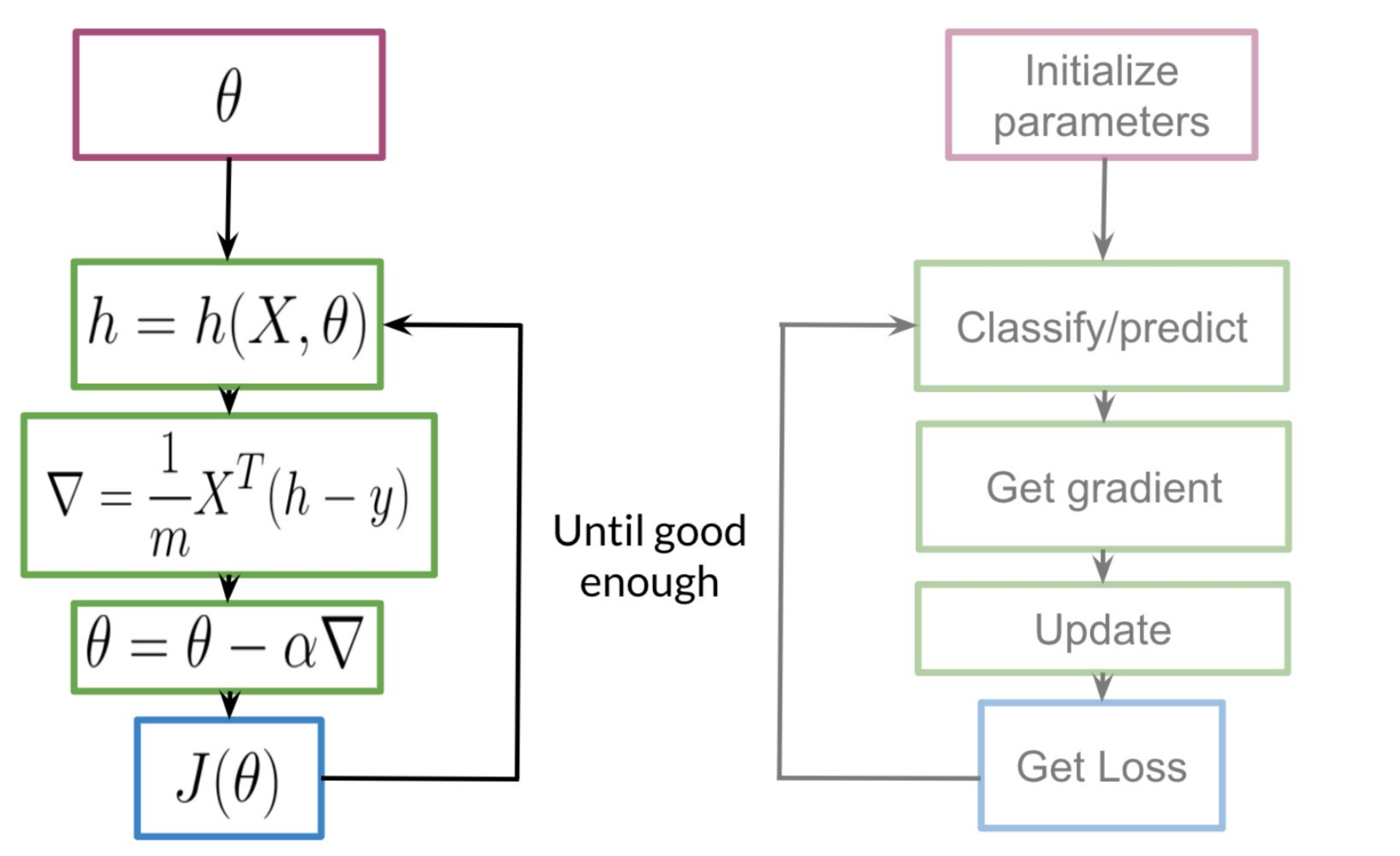
- $x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix}$

1: bias
3476: sum of pos freq.
245: sum of neg freq.

- $\theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$



Logistic Regression Training (Gradient Descent)



Cost Function for LR

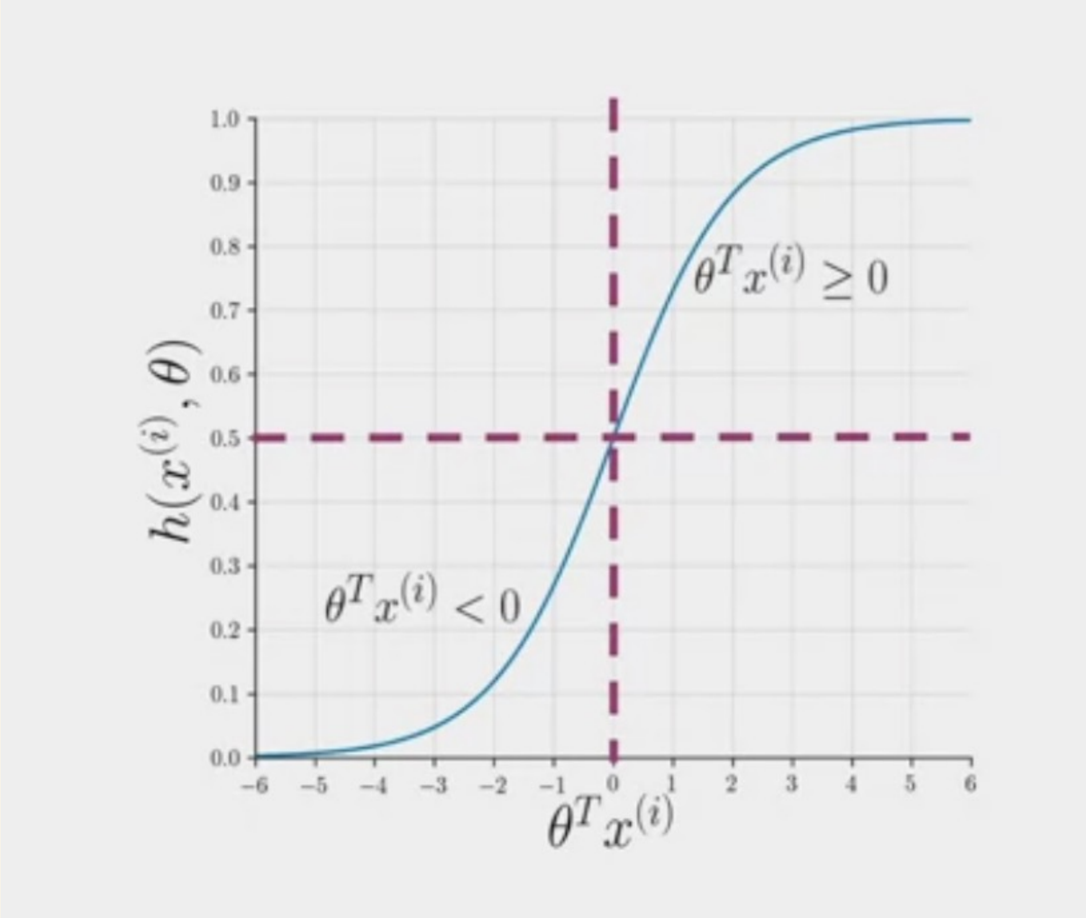
- $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h\left(x^{(i)},\theta\right) + \left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)},\theta\right)\right)\right]$

- m: number of training examples in the training set
- 1/m: average
- Negative: to make sure that the overall cost is always positive
- First term: when the label is ‘0’, it becomes ‘0’. If the label and prediction do not match, it becomes –inf
- Second term: when the label is ‘1’, it becomes ‘0’. If the label and prediction do not match, it becomes -inf

Logistic Regression Testing

- Input: X_{val}
- Output: Y_{val}
- Parameters: θ
- pred = $h(X_{val}, \theta)$

- $Y_{pred} = \begin{cases} 1\ if\ pred \geq 0.5 \\ 0\ if\ pred < 0.5 \end{cases}$



Logistic Regression Testing

- $h = \begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \dashrightarrow \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} \dashrightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ Y_{pred_m} \end{bmatrix}$

LR outputs

Compare with threshold

Y prediction

- Each row is the output of LR for each tweet
- Each output is compared to the threshold to see if the prediction is 1(pos or 0 (neg))

- Accuracy = $\sum_{i=1}^{m} \frac{(Y_{pred}^{(i)} == Y_{val}^{(i)})}{m} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ Y_{pred}^{(i)} == Y_{val}^{(i)} \end{bmatrix}$

Y actual

Compare Y actual with Y prediction

Logistic Regression Testing

- $Y_{val} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \dashrightarrow Y_{pred} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \dashrightarrow Y_{val} == Y_{pred} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Y actual

Y prediction

Compare Y actual
with Y prediction
'1' if $Y_{val} == Y_{pred}$
'0' otherwise

- Accuracy = $\frac{4}{5} = 0.8$

4: number of matching between true and predict
5: total number of observations

TF-IDF

- Term Frequency-Inverse Document Frequency
- tf(t,d) = number of times a term (t) appears in a document (d)/total number of terms in the document
  - A term importance is relevant to its frequency in a document
- df(t,D) = number of documents that have (t)/number of documents
  - terms that appear in almost all documents are usually not important

tfidf = tf/df = tf *idf

TF-IDF

- TF-IDF gives a measure of how important a term for a specific document
- The log of the IDF is used to uniform the word distribution

log(IDF)

- Thus:

$$tfidf = \frac{\text{Term Count in Document}}{\text{Total Terms in Document}} * \log\left(\frac{\text{Total Documents}}{\text{Documents Containing the Term}}\right)$$

Calculating TF

Data corpus (tweets)	Labels
I am happy because I am learning NLP	positive
I am happy	positive
I am sad, I am not learning NLP	negative
I am sad	negative

Vocabulary	Tweet-1	Tweet-2	Tweet-3	Tweet-4
I	0.25	0.333	0.25	0.333
am	0.25	0.333	0.25	0.333
happy	0.125	0.333	0	0
because	0.125	0	0	0
learning	0.125	0	0.125	0
NLP	0.125	0	0.125	0
sad	0	0	0.125	0.333
not	0	0	0.125	0

$$tf = \frac{\text{Term Count in Document}}{\text{Total Terms in Document}}$$

tf(I,tweet-1) = 2/8= 0.25

tf(not,tweet-3) = 1/8= 0.125

Calculating IDF

Data corpus (tweets)	Labels
I am happy because I am learning NLP	positive
I am happy	positive
I am sad, I am not learning NLP	negative
I am sad	negative

Vocabulary	IDF
I	0
am	0
happy	0.301
because	0.602
learning	0.602
NLP	0.301
sad	0.301
not	0.602

$$idf = \log\left(\frac{\text{Total Documents}}{\text{Documents Containing the Term}}\right)$$

idf(I) = log(4/4) = 0

idf(not) = log(4/1) = 0.602

Calculating TF-IDF

Vocabulary	Tweet-1	Tweet-2	Tweet-3	Tweet-4
I	0.25	0.333	0.25	0.333
am	0.25	0.333	0.25	0.333
happy	0.125	0.333	0	0
because	0.125	0	0	0
learning	0.125	0	0.125	0
NLP	0.125	0	0.125	0
sad	0	0	0.125	0.333
not	0	0	0.125	0

tf

IDF
0
0
0.301
0.602
0.602
0.301
0.301
0.602

idf

Tweet-1	Tweet-2	Tweet-3	Tweet-4
0	0	0	0
0	0	0	0
0.037	0.1	0	0
0.075	0	0	0
0.075	0	0.075	0
0.037	0	0.037	0
0	0	0.037	0.1
0	0	0.075	0

tfidf = tf *idf

sklearn feature extraction

```python
from sklearn.feature_extraction.text import CountVectorizer

corpus = ['I am happy because I am learning NLP', 'I am happy', 'I am sad, I am not learning NLP', 'I am sad']
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
vectorizer.get_feature_names()
```

```
['am', 'because', 'happy', 'learning', 'nlp', 'not', 'sad']
```

```python
print(X.toarray())
```

```
[[2 1 1 1 1 0 0]
 [1 0 1 0 0 0 0]
 [2 0 0 1 1 1 1]
 [1 0 0 0 0 0 1]]
```

sklearn feature extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = ['I am happy because I am learning NLP', 'I am happy', 'I am sad, I am not learning NLP', 'I am sad']
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
vectorizer.get_feature_names()
```

```
['am', 'because', 'happy', 'learning', 'nlp', 'not', 'sad']
```

```
print(X.toarray())
```

```
[[0.52486474 0.50289672 0.39648955 0.39648955 0.39648955 0.
  0.        ]
 [0.55193942 0.         0.83388421 0.         0.         0.
  0.        ]
 [0.52486474 0.         0.         0.39648955 0.39648955 0.50289672
  0.39648955]
 [0.55193942 0.         0.         0.         0.         0.
  0.83388421]]
```

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

Bayes' Rule

- Bayes rule is based on the mathematical formulation of conditional probabilities.
- We can calculate the probability of x given y if you already know the probability of y given x and the ratio of the probabilities of x and y.
- $P(X|Y) = P(Y|X) \times \frac{P(X)}{P(Y)}$

Bayes' Rule

- Total number of tweets = **20**
- Total number of positive tweets = **13** → P(positive) = 13/20 = 0.65
- Total number of negative tweets = **7** → P(negative) = 1- P(positive) = 0.35
- Total number of positive tweets that have the word "happy" = 3

→ P(positive ∩ happy) = 3/20 = 0.15

Conditional probability

- Total number of tweets = **20**
- Total number of positive tweets = **13**
- Total number of positive tweets that have the word "happy" = **3**
- Total number of negative tweets that have the word "happy" = **1**
- Then, the probability that a tweet is positive, given that it contains the word happy
- P(A|B) = P(positive| "happy") = 3/4 = 0.75
- Then, the probability that a tweet has the word "happy", given that it is positive
- P(B|A) = P("happy" | positive) = 3/13 = 0.231

Conditional probability

- $P(\text{positive} \mid \text{“happy”}) = \dfrac{P(\text{positive} \cap \text{“happy”})}{P(\text{“happy”})}$

- $P(\text{“happy”} \mid \text{positive}) = \dfrac{P(\text{“happy”} \cap \text{positive})}{P(\text{positive})}$

- Bayes’ Rule:
- $P(\text{positive} \mid \text{“happy”}) = P(\text{“happy”} \mid \text{positive}) \dfrac{P(\text{positive})}{P(\text{“happy”})}$

Naive Bayes

- NB is a simple and fast technique
- Naive Bayes tends to perform well on text classifications
- This method is called naive because:
  - It assumes that features are all independent
  - Also, Relative to frequencies in corpus

Naive Bayes for Sentiment Analysis

Data corpus (tweets)	Labels
I am happy because I am learning NLP	positive
I am happy, not sad.	positive
I am sad, I am not learning NLP	negative
I am sad, not happy	negative

Vocabulary	Pos_freq (1)	Neg_freq (0)
I	3	3
am	3	3
happy	2	1
because	1	0
learning	1	1
NLP	1	1
sad	1	2
not	1	2

N_{class} 13 12

P(Wi | class)

Vocabulary	Pos_freq(1)	Neg_freq (0)
I	3	3
am	3	3
happy	2	1
because	1	0
learning	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	13	12

Vocabulary	Pos	Neg
I	0.23	0.25
am	0.23	0.25
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.17
not	0.08	0.17
Sum	1	1

- divide the frequency of each word in a class by the sum of words in the class

P(Wi | class)

- Words that are equally probable don't add anything to the sentiment. (e.g., I, am, learning, and NLP)
- Words that have different probabilities will carry a lot of weight in determining the tweet sentiment. (e.g., happy, sad, and not)
- The word “because” only appears in the positive corpus. It's conditional probability for the negative class is 0.
- This will become a problem for calculations. To avoid this, we will smooth the probability function.

Vocabulary	Pos	Neg
I	0.23	0.25
am	0.23	0.25
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.17
not	0.08	0.17
Sum	1	1

Naive Bayes

Tweet: I am happy today; I am learning.

$$\prod_{i=1}^{m} \frac{P(w_i \mid pos)}{P(w_i \mid neg)} = 1.34 > 1$$

$$\frac{0.23}{0.25} \times \frac{0.23}{0.25} \times \frac{0.15}{0.08} \times \frac{0.23}{0.25} \times \frac{0.23}{0.25} \times \frac{0.08}{0.08}$$

Vocabulary	pos	neg
I	0.23	0.25
am	0.23	0.25
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.17
not	0.08	0.17
Sum	1	1

Laplacian Smoothing

$$P(w_i \mid class) = \frac{freq(w_i, class)}{N_{class}}$$, where class ∈ {positive, negative}

$$P(w_i \mid class) = \frac{freq(w_i, class) + 1}{N_{class} + V_{class}}$$

N_{class}: frequency of all words in a class

V_{class} : number of unique words in a class

P(Wi | class) with smoothing

Vocabulary	Pos	Neg
I	3	3
am	3	3
happy	2	1
because	1	0
learning	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	13	12

Vocabulary	Pos	Neg
I	0.19	0.20
am	0.19	0.20
happy	0.14	0.10
because	0.10	0.05
learning	0.10	0.10
NLP	0.10	0.10
sad	0.10	0.15
not	0.10	0.15

- P(I | Pos) = $\frac{3+1}{13+8}$ $, where\ V = 8$

Ratio of probabilities

Vocabulary	Pos	Neg	ratio
I	0.20	0.20	1
am	0.20	0.20	1
happy	0.14	0.10	1.4
because	0.10	0.10	1
learning	0.10	0.10	1
NLP	0.10	0.10	1
sad	0.10	0.15	0.6
not	0.10	0.15	0.6

- Positive words have ratios>1
- Neutral words have ratio=1
- Negative words have ratio<1

- ratio(w_i) = $\frac{P(w_i \mid Pos)}{P(w_i \mid Neg)}$

Naive Bayes' inference

$$\frac{P(pos)}{P(neg)} \prod_{i=1}^{m} \frac{P(w_i \mid pos)}{P(w_i \mid neg)} > 1$$

$\frac{P(pos)}{P(neg)}$ this term equals to 1 when the dataset is balanced

Log Likelihood

- Sentiments probability calculation requires multiplication of many numbers with values between 0 and 1.
- Carrying out such multiplications on a computer runs the risk of numerical underflow when the number returned is so small if can't be stored on your device.
- Solution:
- $\log(\frac{P(pos)}{P(neg)} \prod_{i=1}^{m} \frac{P(w_i \mid pos)}{P(w_i \mid neg)}) \rightarrow \log\left(\frac{P(pos)}{P(neg)}\right) + \sum_{i=1}^{m} \log \frac{P(w_i \mid pos)}{P(w_i \mid neg)}$

log prior + log likelihood

Calculating Lambda

I am happy because I am learning

$$\lambda(\mathrm{w}) = \log(\frac{P(w|pos)}{P(w|neg)}$$

$$\lambda(\mathrm{I}) = log\frac{0.05}{0.05} = \log(1) = 0 \text{ (Neutral)}$$

Vocabulary	Pos	Neg	λ
I	0.05	0.05	0
am	0.04	0.04	0
happy	0.09	0.01	2.2
because	0.01	0.01	0
learning	0.03	0.01	1.1
NLP	0.02	0.02	0
sad	0.01	0.09	-2.2
not	0.02	0.03	-0.4

Calculating Lambda

I am happy because I am learning

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

Vocabulary	Pos	Neg	λ
I	0.05	0.05	0
am	0.04	0.04	0
happy	0.09	0.01	2.2
because	0.01	0.01	0
learning	0.03	0.01	1.1
NLP	0.02	0.02	0
sad	0.01	0.09	-2.2
not	0.02	0.03	-0.4

$$\text{Log likelihood} = 0+0+2.2+0+0+0+1.1 = 3.3 > 0 \text{ (positive)}$$

Naive Bayes Training

- To train your naïve Bayes classifier, you have to perform the following steps:

1) Get or annotate a dataset with positive and negative tweets
2) Preprocess the tweets:
    - Lowercase
    - Remove punctuation, urls, names
    - Remove stop words
    - Stemming
    - Tokenize sentences
3) Compute freq(w, class):
4) Get probabilities $P(w|pos), P(w|neg)$
5) Get $\lambda(w)$
6) Compute logprior

Naive Bayes Testing

- Loglikelihood ($\lambda(w)$)
- Logprior = 0

- Tweet: [I, pass, the, nlp, interview]
- Score = -0.01+0.5-0.01+0+logprior= 0.48

- Pred = score > 0 → positive

Vocabulary	λ
I	-0.01
the	-0.01
happy	0.63
because	0.01
pass	0.5
NLP	0
sad	-0.75
not	-0.75

References:

- Natural Language Processing, Intel
- Natural Language Processing, DeepLearning.AI
- scikit-learn: https://scikit-learn.org