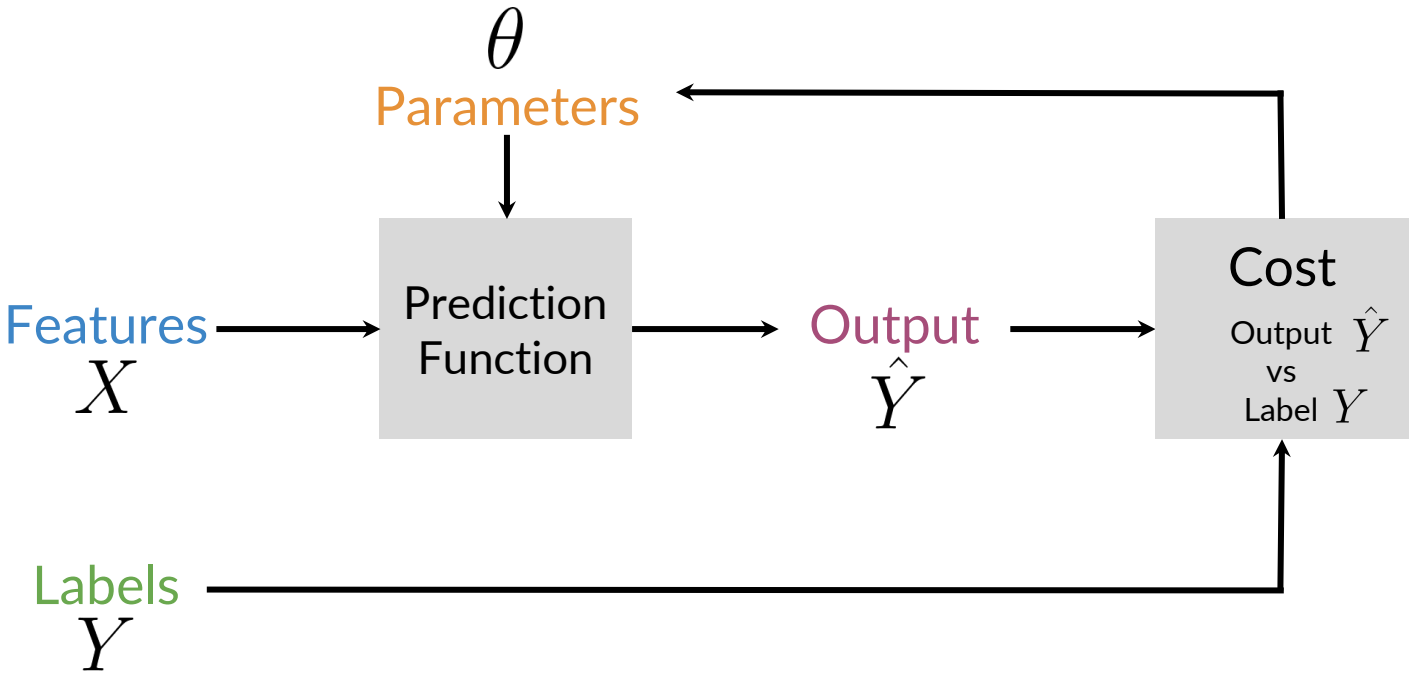
deeplearning.ai

Supervised ML and Sentiment Analysis

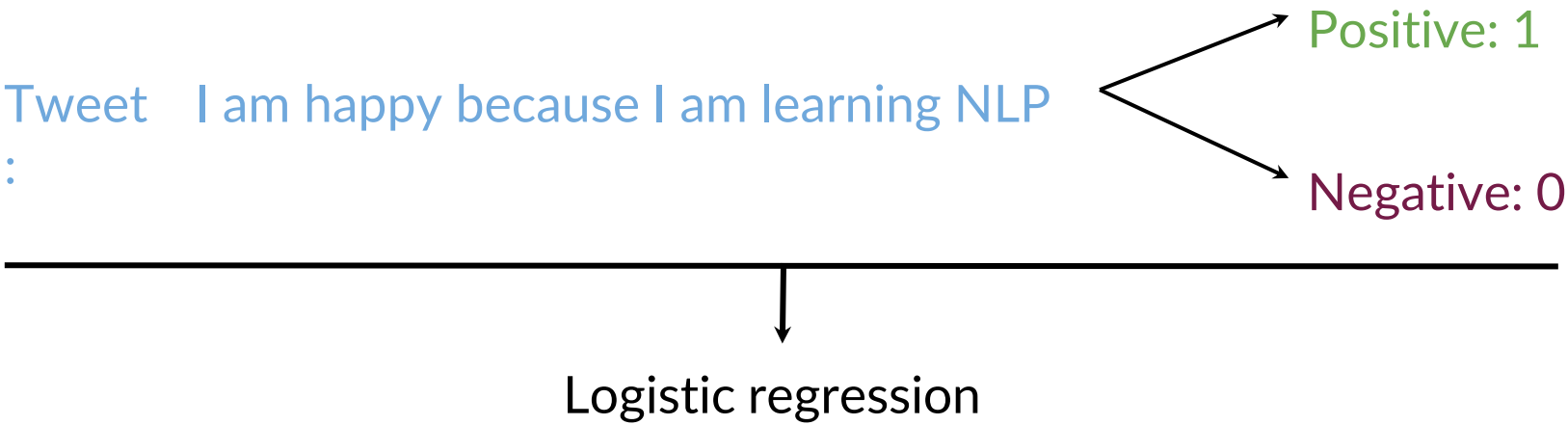
Outline

- Review Supervised ML
- Build your own tweet classifier!

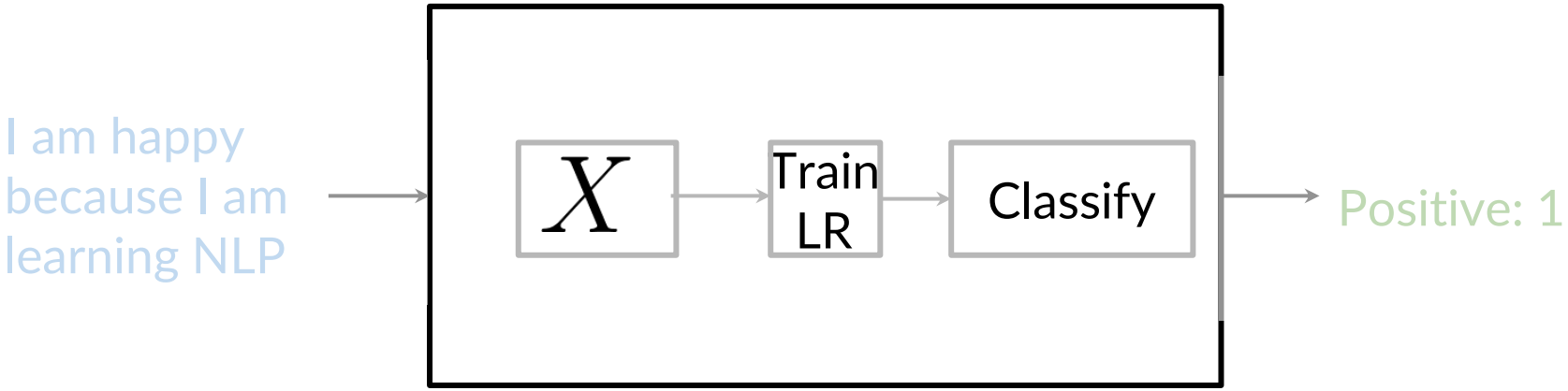
Supervised ML (training)



Sentiment analysis



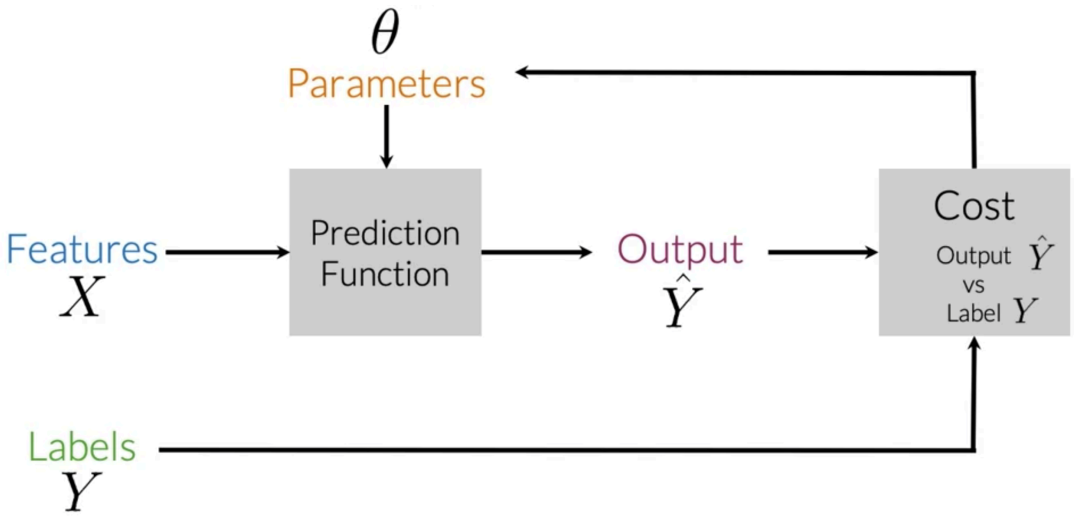
Sentiment analysis



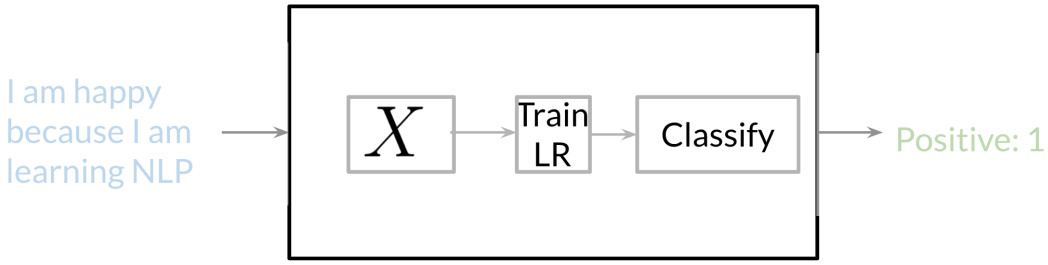
Summary

- Features, Labels → Train → Predict
- Extract features → Train LR → Predict sentiment

In supervised machine learning, you usually have an input X, which goes into your prediction function to get your $\hat{Y}$. You can then compare your prediction with the true value Y. This gives you your cost which you use to update the parameters θ. The following image, summarizes the process.



To perform sentiment analysis on a tweet, you first have to represent the text (i.e. "I am happy because I am learning NLP ") as features, you then train your logistic regression classifier, and then you can use it to classify the text.



Note that in this case, you either classify 1, for a positive sentiment, or 0, for a negative sentiment.

deeplearning.ai

Vocabulary and Feature Extraction

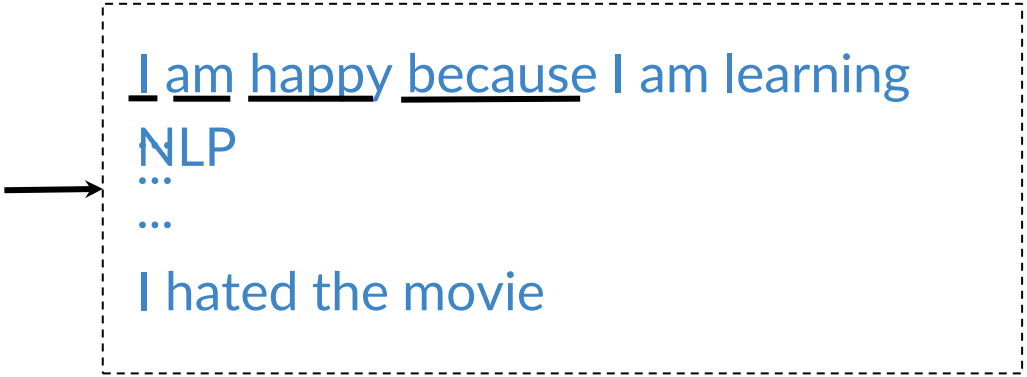
Outline

- Vocabulary
- Feature extraction
- Sparse representations and some of their issues

Vocabulary

Tweets:

[tweet_1, tweet_2, ..., tweet_m]



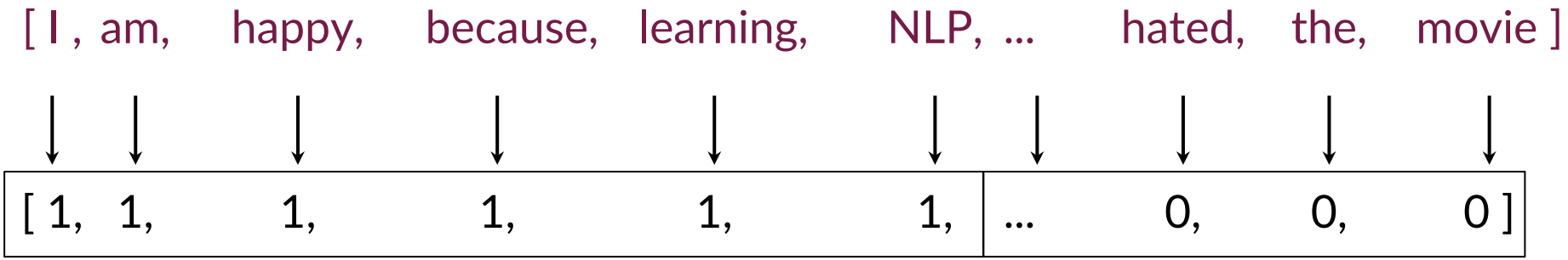
(Unique words!)

$$V =$$

[I, am, happy, because, learning, NLP, ... hated, the, movie]

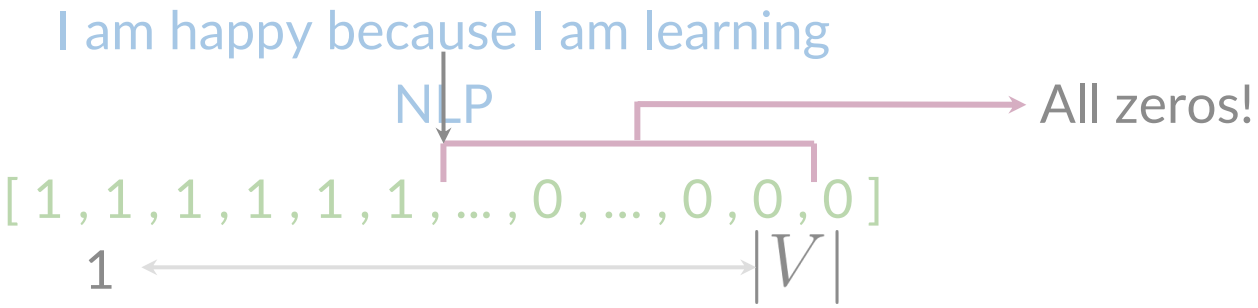
Feature extraction

I am happy because I am learning NLP



A lot of zeros! That's a sparse representation.

Problems with sparse representations



$$[\theta_0, \theta_1, \theta_2, \ldots, \theta_n]$$

$$n = |V|$$

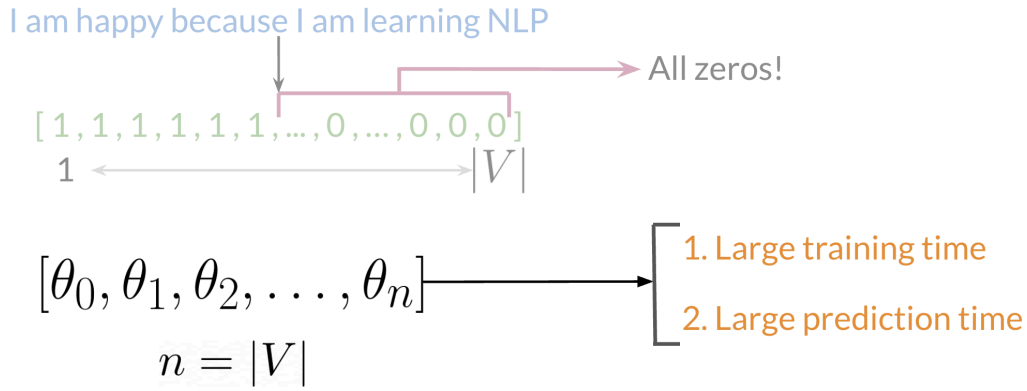
(n+1 parameters!)

1. Large training time
2. Large prediction time

Summary

- Vocabulary: set of unique words
- Vocabulary, Text ⟶ [1 ..... 0 ..... 1 .. 0 .. 1 .. 0]
- Sparse representations are problematic for training and prediction times

Given a tweet, or some text, you can represent it as a vector of dimension V, where V corresponds to your vocabulary size. If you had the tweet "I am happy because I am learning NLP", then you would put a 1 in the corresponding index for any word in the tweet, and a 0 otherwise.



As you can see, as V gets larger, the vector becomes more sparse. Furthermore, we end up having many more features and end up training θ V parameters. This could result in larger training time, and large prediction time.

deeplearning.ai

Negative and Positive Frequencies

Outline

- Populate your vocabulary with a frequency count for each class

Positive and negative counts

Corpus

I am happy because I am learning NLP

I am happy

I am sad, I am not learning NLP

I am sad

A corpus is a large collection of written or spoken texts that is used for language research.

Vocabulary
I
am
happy
because
learning
NLP
sad
not

Positive and negative counts

Positive tweets	Negative tweets
I am happy because I am learning NLP	I am sad, I am not learning NLP
I am happy	I am sad

Positive and negative counts

Positive tweets

I am happy because I am learning NLP
I am happy

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

Positive and negative counts

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

Negative tweets

I am sad, I am not learning NLP

I am sad

Word frequency in classes

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

freqs: dictionary mapping from (word, class) to frequency

Summary

- Divide tweet corpus into two classes: positive and negative
- Count each time each word appears in either class

➔ Feature extraction for training and prediction!

deeplearning.ai

Feature extraction with frequencies

Outline

- Extract features from your frequencies dictionary to create a features vector

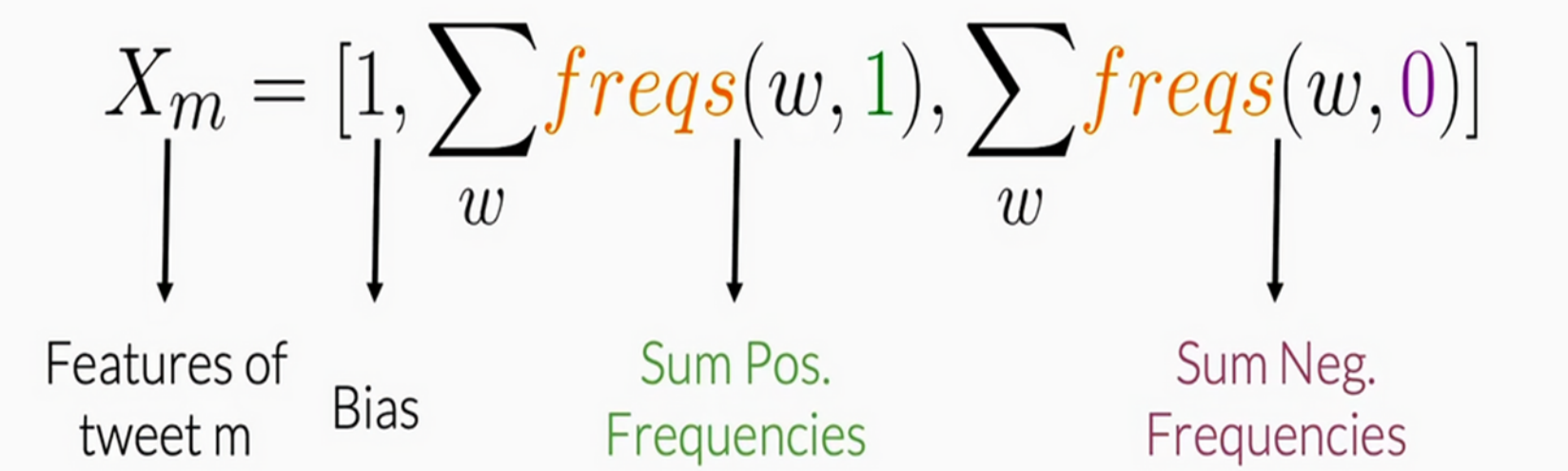
Word frequency in classes

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

freqs: dictionary mapping from (word, class) to frequency

Feature extraction

freqs: dictionary mapping from (word, class) to frequency

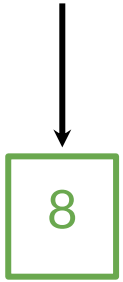


Feature extraction

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

I am sad, I am not learning NLP

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



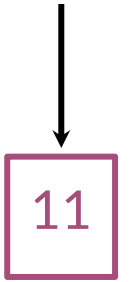
8

Feature extraction

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

I am sad, I am not learning NLP

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



Feature extraction

I am sad, I am not learning NLP

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

$$\downarrow$$

$$X_m = [1, 8, 11]$$

Summary

- Dictionary mapping (word,class) to frequencies

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

➔ Cleaning unimportant information from your tweets

Given a corpus with positive and negative tweets as follows:

Positive tweets	Negative tweets
I am happy because I am learning NLP	I am sad, I am not learning NLP
I am happy	I am sad

You have to encode each tweet as a vector. Previously, this vector was of dimension V. Now, as you will see in the upcoming videos, you will represent it with a vector of dimension 3. To do so, you have to create a dictionary to map the word, and the class it appeared in (positive or negative) to the number of times that word appeared in its corresponding class.

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

freqs: dictionary mapping from (word, class) to frequency

In the past two videos, we call this dictionary `freqs`. 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. Given this dictionary and the tweet, "I am sad, I am not learning NLP", you can create a vector corresponding to the feature as follows:

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

I am sad, I am not learning NLP

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

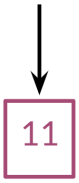
8

To encode the negative feature, you can do the same thing.

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

I am sad, I am not learning NLP

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



Hence you end up getting the following feature vector $[1, 8, 11]$. 1 corresponds to the bias, 8 the positive feature, and 11 the negative feature.

deeplearning.ai

Preprocessing

Outline

- Removing stopwords, punctuation, handles and URLs
- Stemming
- Lowercasing

Preprocessing: stop words and punctuation

@YMourri and @AndrewYNg are tuning a GREAT AI model at https://deeplearning.ai!!!

Stop words
and
is
are
at
has
for
a

Punctuation
,
.
:
!
“
‘

Preprocessing: stop words and punctuation

@YMourri and @AndrewYNg are tuning a GREAT AI model at https://deeplearning.ai!!!

@YMourri @AndrewYNg tuning GREAT AI model https://deeplearning.ai!!!

Stop words
and
is
are
at
has
for
a

Punctuation
,
.
:
!
“
‘

Preprocessing: stop words and punctuation

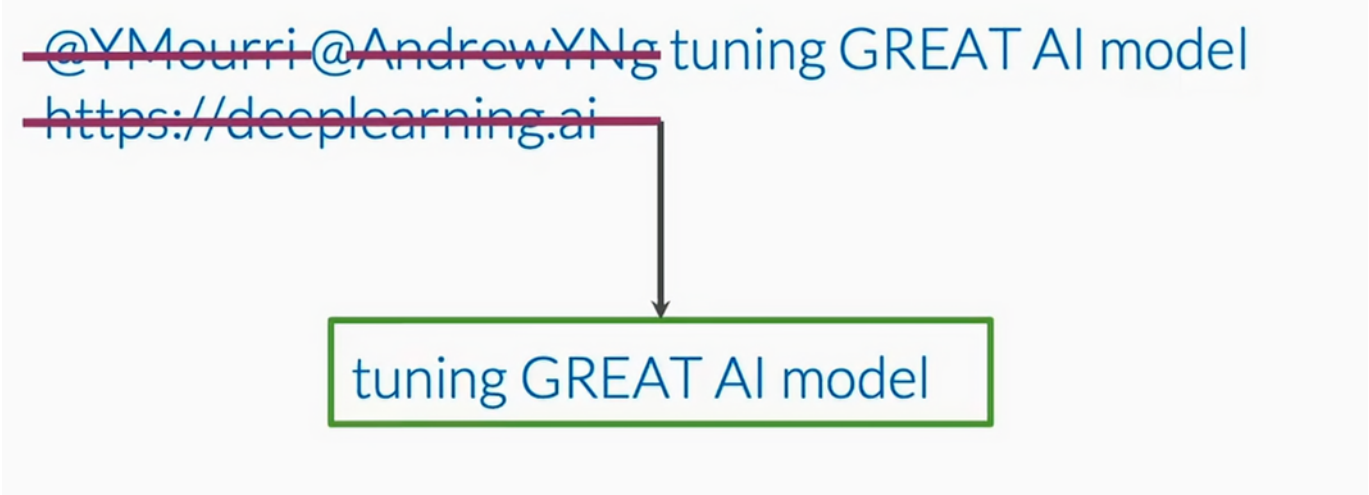
@YMourri @AndrewYNg tuning
GREAT AI model
https://deeplearning.ai!!!

@YMourri @AndrewYNg tuning
GREAT AI model
https://deeplearning.ai

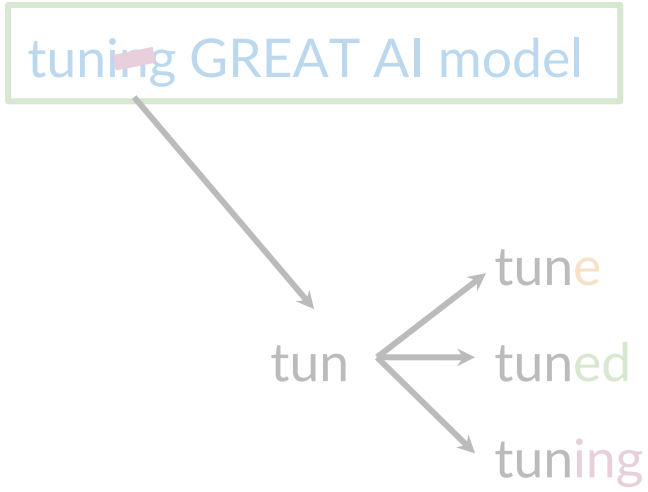
Stop words
and
is
a
at
has
for
of

Punctuation
,
.
:
!
“
‘

Preprocessing: Handles and URLs



Preprocessing: Stemming and lowercasing



Preprocessed tweet:
[tun, great, ai, model]

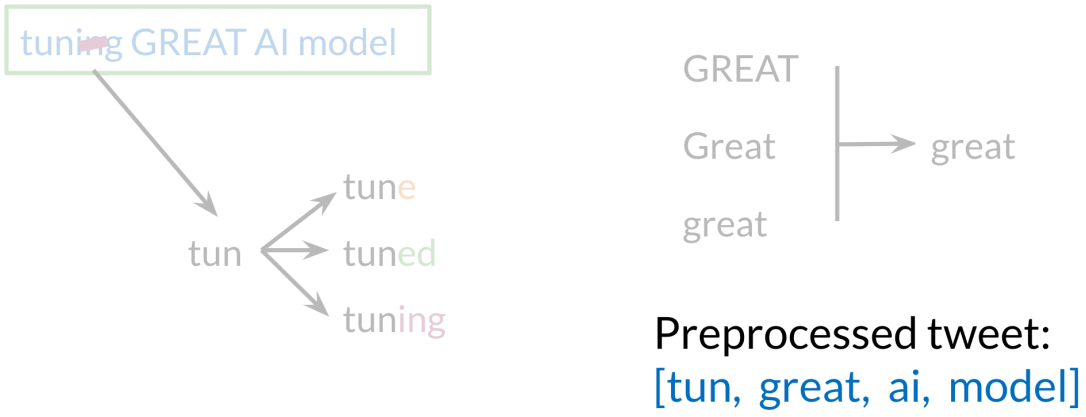
Summary

- Stop words, punctuation, handles and URLs
- Stemming
- Lowercasing
- Less unnecessary info ⟶ Better times

When preprocessing, you have to perform the following:

1. Eliminate handles and URLs
2. Tokenize the string into words.
3. Remove stop words like "and, is, a, on, etc."
4. Stemming- or convert every word to its stem. Like dancer, dancing, danced, becomes 'danc'. You can use porter stemmer to take care of this.
5. Convert all your words to lower case.

For example the following tweet "@YMourri and @AndrewYNg are tuning a GREAT AI model at https://deeplearning.ai!!!" after preprocessing becomes



$[tun, great, ai, model]$. Hence you can see how we eliminated handles, tokenized it into words, removed stop words, performed stemming, and converted everything to lower case.

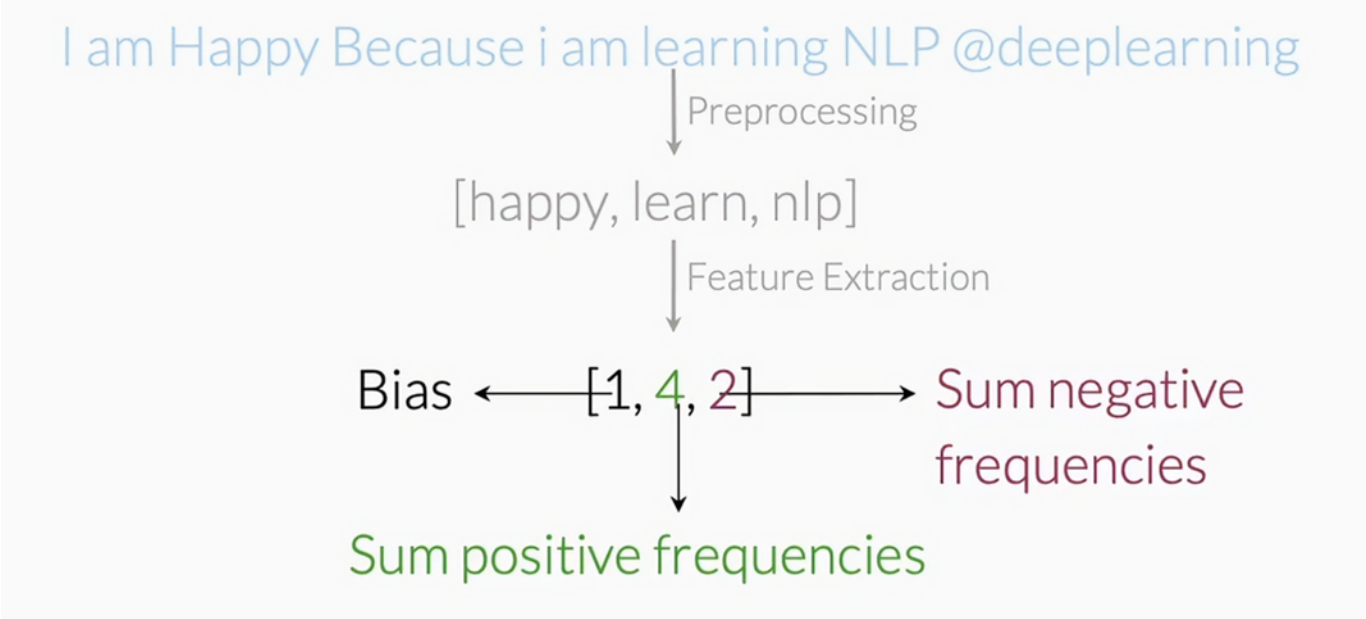
deeplearning.ai

Putting it all together

Outline

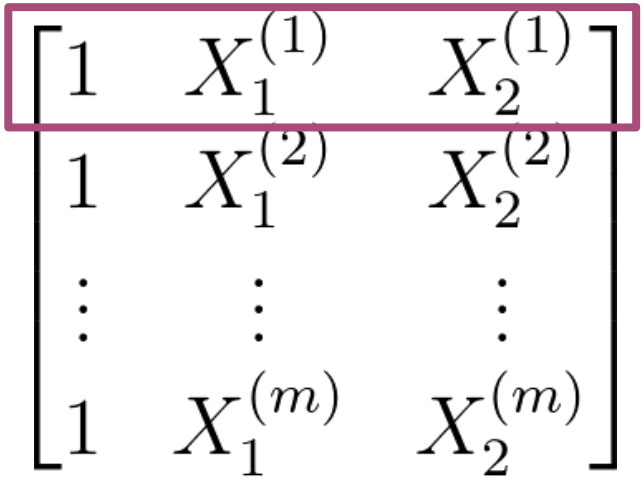
- Generalize the process
- How to code it!

General overview



General overview

for m tweets



[1, 40, 20],
[1, 20, 50],
...
[1, 5, 35]]

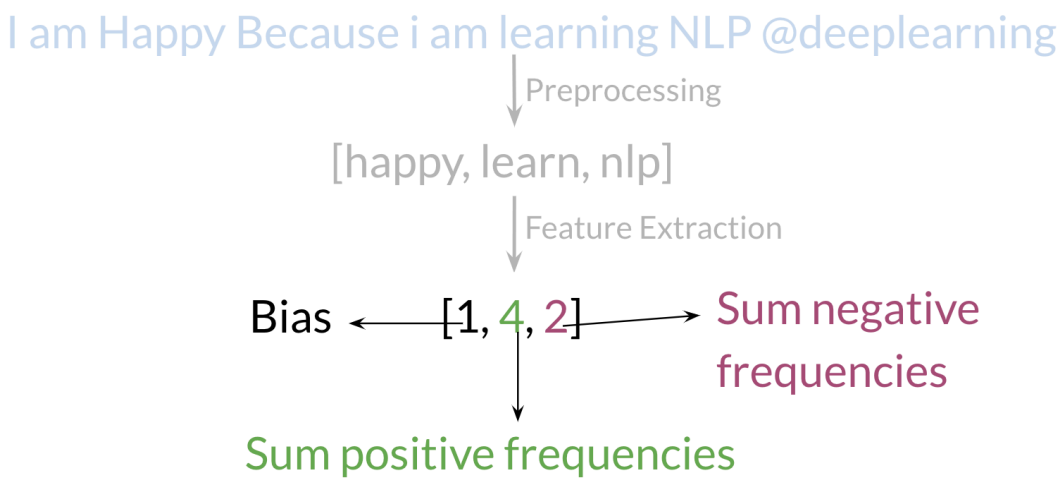
General Implementation

```
freqs = build_freqs(tweets,labels) #Build frequencies dictionary

X = np.zeros((m,3)) #Initialize matrix X

for i in range(m): #For every tweet

        p_tweet = process_tweet(tweets[i]) #Process tweet

    X[i,:] = extract_features(p_tweet,freqs) #Extract Features
```

Summary

- Implement the feature extraction algorithm for your entire set of tweets
- Almost ready to train!

Over all , you start with a given text, you perform preprocessing, then you do feature extraction to convert text into numerical representation as follows:



Your X becomes of dimension $(m, 3)$ as follows.

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix}$$

When implementing it with code, it becomes as follows:

```
freqs = build_freqs(tweets,labels) #Build frequencies dictionary

X = np.zeros((m,3)) #Initialize matrix X

for i in range(m): #For every tweet

    p_tweet = process_tweet(tweets[i]) #Process tweet

    X[i,:] = extract_features(p_tweet,freqs) #Extract Features
```

You can see in the last step you are storing the extracted features as rows in your X matrix and you have m of these examples.

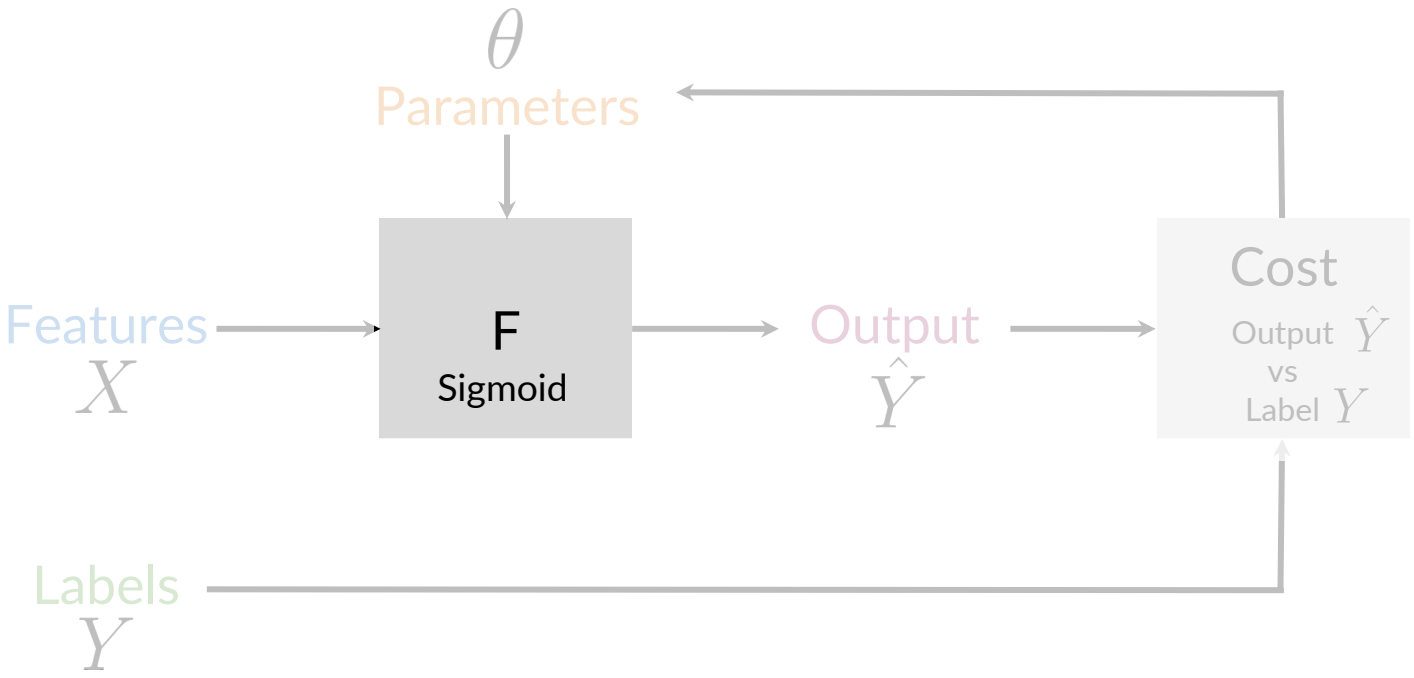
deeplearning.ai

Logistic Regression Overview

Outline

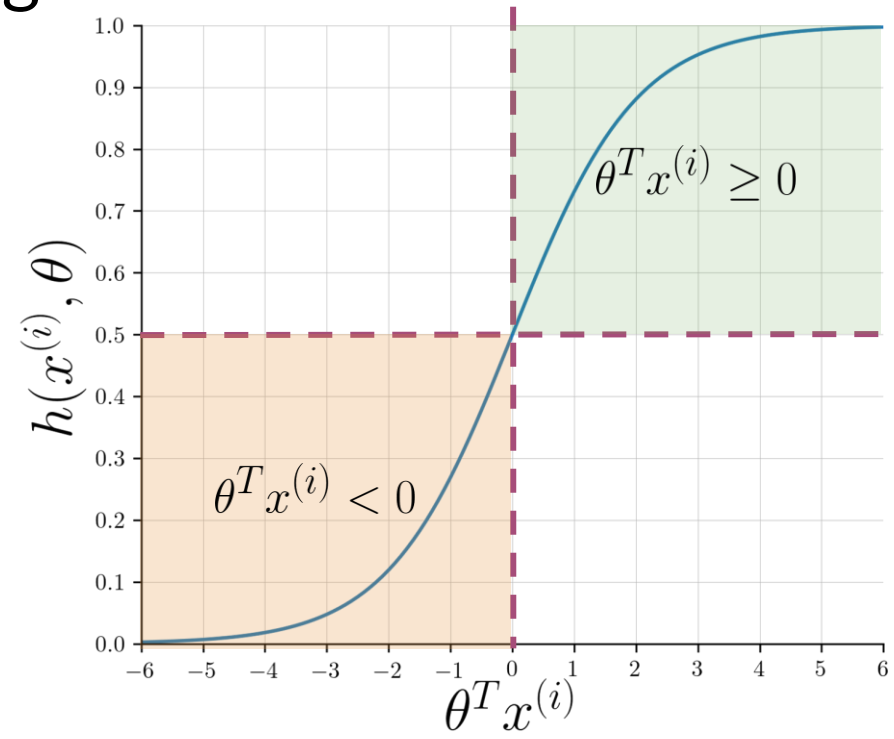
- Supervised learning and logistic regression
- Sigmoid function

Overview of logistic regression



Overview of logistic regression

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



Overview of logistic regression

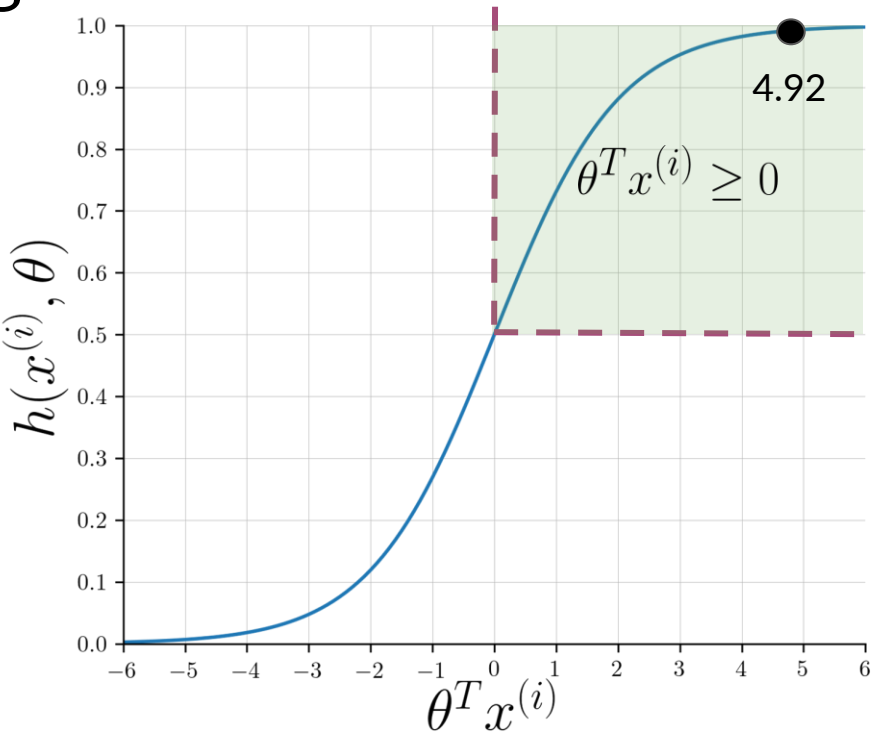
@YMourri and @AndrewYNg are tuning a GREAT AI model

↓

[tun, ai, great, model]

↓

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$

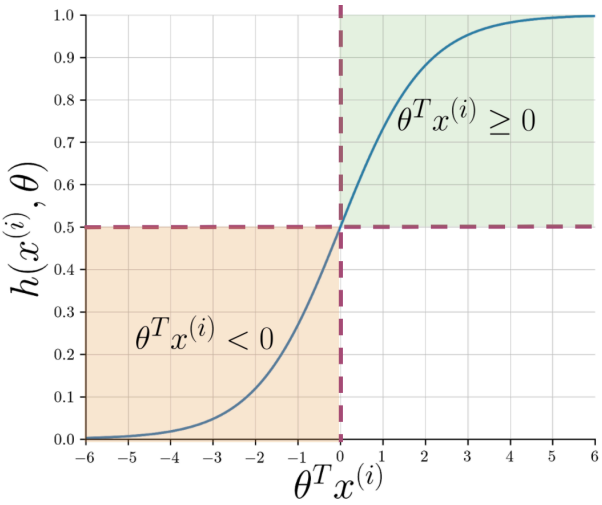


Summary

- Sigmoid function
- $\theta^T x^{(i)} \geq 0 \longrightarrow h(x^{(i)}, \theta) \geq 0.5$, positive
- $\theta^T x^{(i)} < 0 \longrightarrow h(x^{(i)}, \theta) < 0.5$, negative

Logistic regression makes use of the sigmoid function which outputs a probability between 0 and 1. The sigmoid function with some weight parameter θ and some input $x^{(i)}$ is defined as follows.

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



Note that as $\theta^T x^{(i)}$ gets closer and closer to $-\infty$ the denominator of the sigmoid function gets larger and larger and as a result, the sigmoid gets closer to 0. On the other hand, as $\theta^T x^{(i)}$ gets closer and closer to ∞ the denominator of the sigmoid function gets closer to 1 and as a result the sigmoid also gets closer to 1.

Now given a tweet, you can transform it into a vector and run it through your sigmoid function to get a prediction as follows:

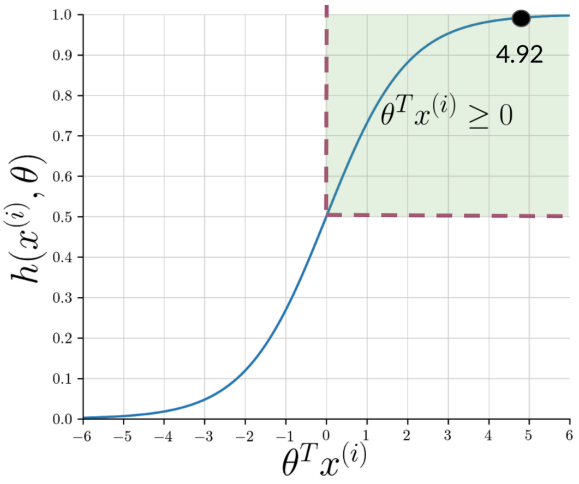
@YMourri and @AndrewYNg are tuning a GREAT AI model

↓

[tun, ai, great, model]

↓

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$



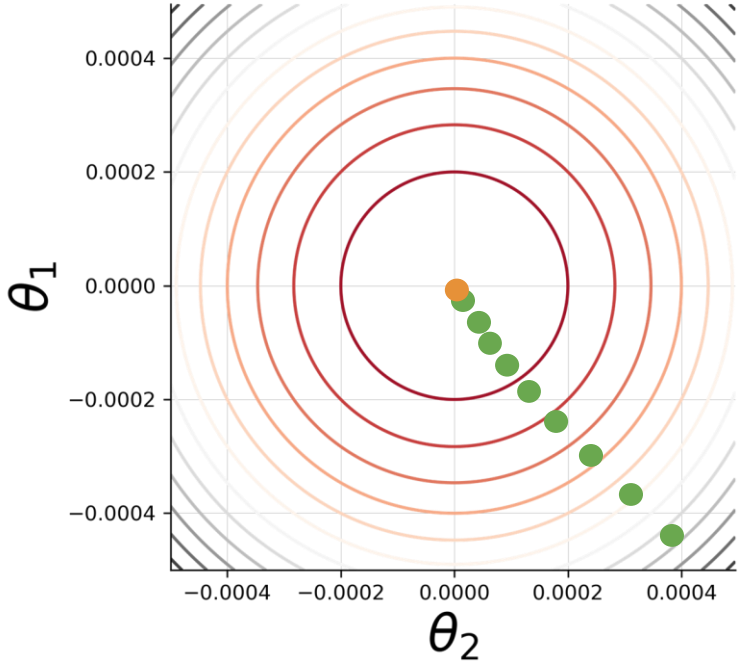
deeplearning.ai

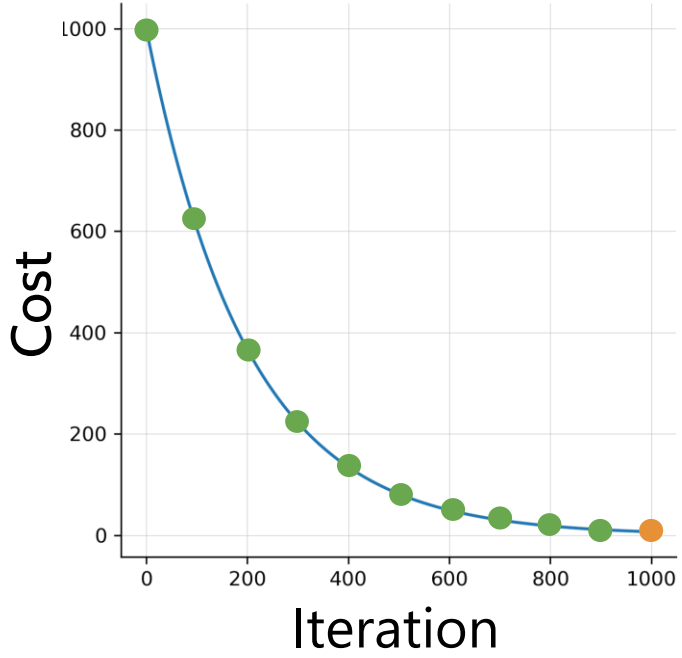
Logistic Regression: Training

Outline

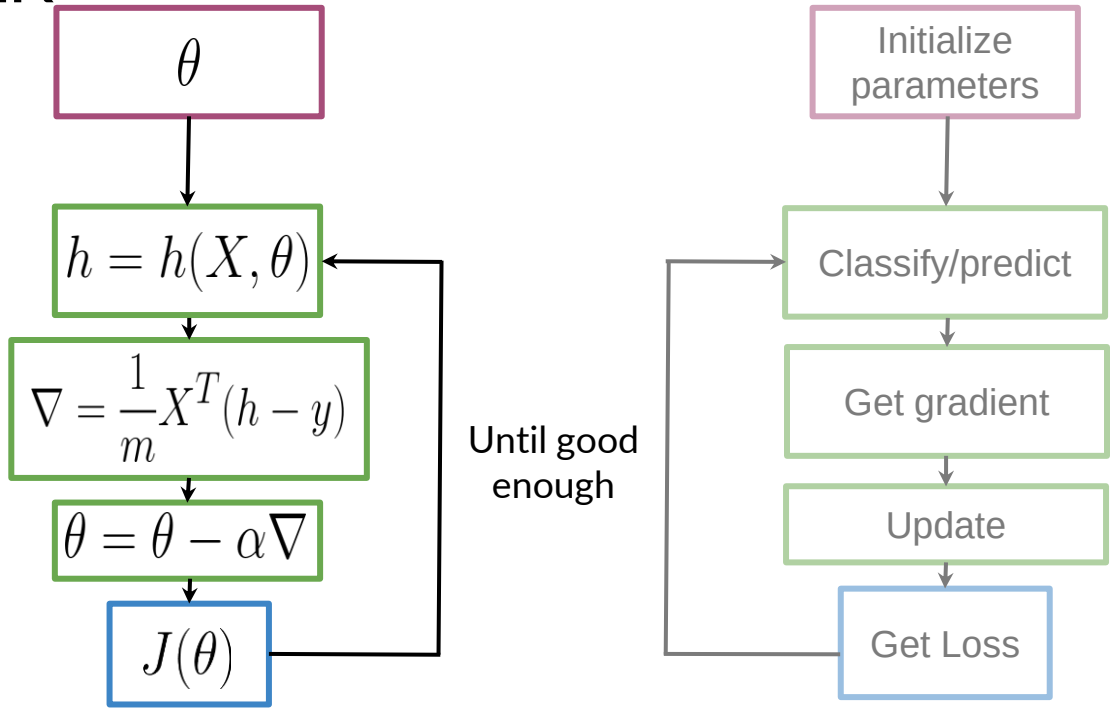
- Review the steps in the training process
- Overview of gradient descent

Training LR


0.0004
0.0002
0.0000
−0.0002
−0.0004
θ_1
−0.0004
−0.0002
0.0000
0.0002
0.0004
θ_2


1000
800
600
400
200
0
Cost
0
200
400
600
800
1000
Iteration

Training LR

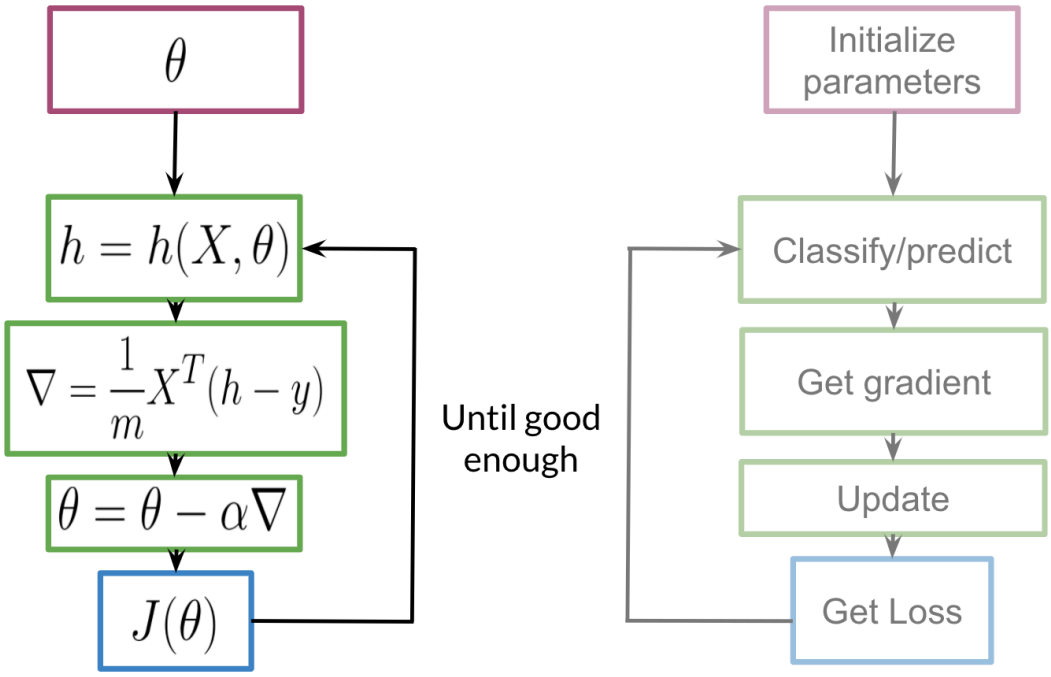

θ
$h = h(X, \theta)$
$\nabla = \frac{1}{m} X^T (h - y)$
$\theta = \theta - \alpha \nabla$
$J(\theta)$
Until good enough
Initialize parameters
Classify/predict
Get gradient
Update
Get Loss

Summary

- Visualize how gradient descent works
- Use gradient descent to train your logistic regression classifier

➔ Compute the accuracy of your model

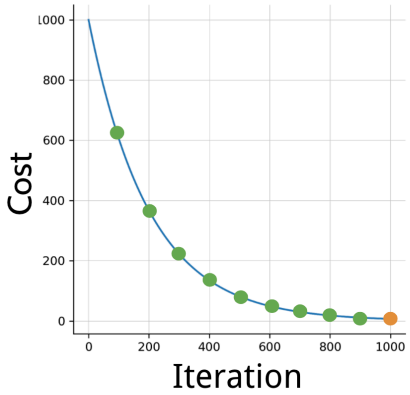
To train your logistic regression function, you will do the following:



You initialize your parameter θ, that you can use in your sigmoid, you then compute the gradient that you will use to update θ, and then calculate the cost. You keep doing so until good enough.

Note: If you do not know what a gradient is, don't worry about it. I will show you what it is at then end of this week in an optional reading. In a nutshell, the gradient allows you to learn what θ is so that you can predict your tweet sentiment accurately.

Usually you keep training until the cost converges. If you were to plot the number of iterations versus the cost, you should see something like this:



deeplearning.ai

Logistic Regression: Testing

Outline

- Using your validation set to compute model accuracy
- What the accuracy metric means

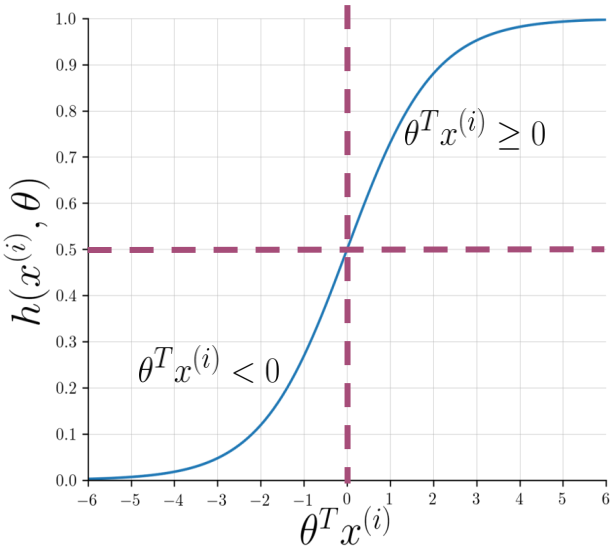
Val = Validation set

Testing logistic regression

- X_{val} Y_{val} θ

$$h(X_{val}, \theta)$$

$$pred = h(X_{val}, \theta) \geq 0.5$$



Testing logistic regression

- X_{val} Y_{val} θ

$$h(X_{val}, \theta)$$

$$pred = h(X_{val}, \theta) \geq 0.5$$

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

Testing logistic regression

- X_{val} Y_{val} θ

$$h(X_{val}, \theta)$$

$$pred = h(X_{val}, \theta) \geq 0.5$$

$$\sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m == Y_{val_m} \end{bmatrix}$$

Testing logistic regression

$$Y_{val} = \begin{bmatrix} 0 \\ 1 \\ \underline{1} \\ 0 \\ 1 \end{bmatrix} \quad pred = \begin{bmatrix} 0 \\ 1 \\ \underline{0} \\ 0 \\ 1 \end{bmatrix} \quad (Y_{val} == pred) = \begin{bmatrix} 1 \\ 1 \\ \underline{0} \\ 1 \\ 1 \end{bmatrix}$$

$$\text{accuracy} = \frac{4}{5} = 0.8$$

Summary

- X_{val} Y_{val} ⟶ Performance on unseen data
- Accuracy ⟶ $\sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$

To improve model: step size, number of iterations, regularization, new features, etc.

To test your model, you would run a subset of your data, known as the validation set, on your model to get predictions. The predictions are the outputs of the sigmoid function. If the output is $\geq = 0.5$, you would assign it to a positive class. Otherwise, you would assign it to a negative class.

- X_{val} Y_{val} θ

$h(X_{val}, \theta)$

$pred = h(X_{val}, \theta) \geq 0.5$

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

In the video, I briefly mentioned X validation. In reality, given your X data you would usually split it into three components. $X_{train}, X_{val}, X_{test}$. The distribution usually varies depending on the size of your data set. However, an $80, 10, 10$ split usually works fine.

To compute accuracy, you solve the following equation:

$$\text{Accuracy} \longrightarrow \sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$$

In other words, you go over all your training examples, m of them, and then for every prediction, if it was wright you add a one. You then divide by m.

deeplearning.ai

Logistic Regression: Cost Function

Outline

- Overview of the logistic cost function, AKA the binary cross-entropy function

Cost function for logistic regression

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

1/m * summation -> average

-: for having a positive cost

Cost function for logistic regression

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

Cost function for logistic regression

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

$y^{(i)}$	$h(x^{(i)},\theta)$	
0	any	0
1	0.99	~0
1	~0	-inf

Cost function for logistic regression

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

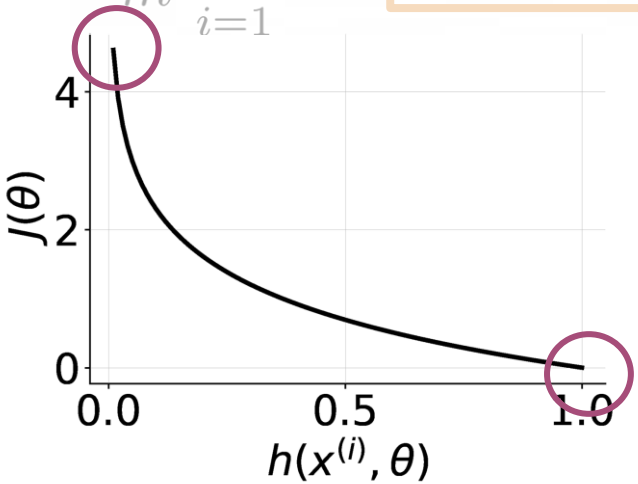
$y^{(i)}$	$h(x^{(i)},\theta)$	
1	any	0
0	0.01	~0
0	~1	-inf

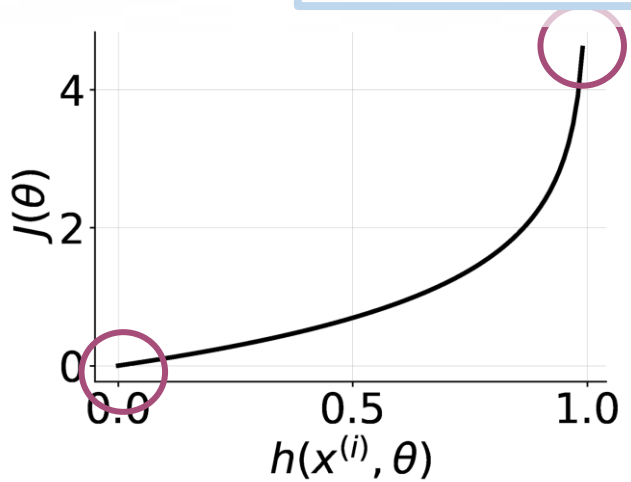
Cost function for logistic regression

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

Cost function for logistic regression

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





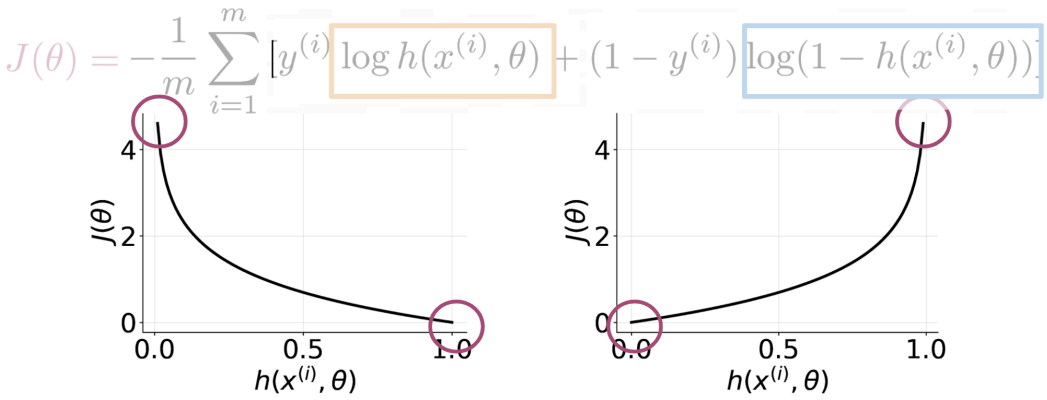
Summary

- Strong disagreement = high cost
- Strong agreement = low cost
- Aim for the lowest cost!

This is an advanced optional reading where we delve into the details.. If you do not get the math, do not worry about it - you will be just fine by moving onto the next component. In this part, I will tell you about the intuition behind why the cost function is designed the way it is. I will then show you how to derive the logistic regression cost function to get the gradients.

The logistic regression cost function is defined as

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



As you can see in the picture above, if $y = 1$ and you predict something close to 0, you get a cost close to ∞. The same applies for then $y = 0$ and you predict something close to 1. On the other hand if you get a prediction equal to the label, you get a cost of 0. In either, case you are trying to minimize $J(\theta)$.

Math Derivation

To show you why the cost function is designed that way, let us take a step back and write up a function that compresses the two cases into one case.

$$P(y|x^{(i)},\theta) = h(x^{(i)},\theta)^{y^{(i)}}(1-h(x^{(i)},\theta)^{(1-y^{(i)})}$$

From the above, you can see that when $y = 1$, you get $h(x^{(i)},\theta)$, and when $y = 0$, you get $(1-h(x^{(i)},\theta))$, which makes sense, since the two probabilities equal to 1. In either case, you want to maximize the function $h(x^{(i)},\theta)$ by making it as close to 1 as possible. When $y = 0$, you want $(1-h(x^{(i)},\theta))$ to be 0, and therefore $h(x^{(i)},\theta)$ close to 1. When $y = 1$, you want $h(x^{(i)},\theta) = 1$.

Now we want to find a way to model the entire data set and not just one example. To do so, we will define the likelihood as follows:

$$L(\theta) = \prod_{i=1}^{m} h(\theta, x^{(i)})^{y^{(i)}}(1-h(\theta,x^{(i)}))^{(1-y^{(i)})}$$

The $\prod$ symbol tells you that you are multiplying the terms together and not adding them. Note that if we mess up the classification of one example, we end up messing up the overall likelihood score, which is exactly what we intended. We want to fit a model to the entire dataset where all data points are related. One issue is that as m

gets larger, what happens to $L(\theta)$? It goes close to zero, because both numbers $h(x^{(i)}, \theta)$ and $(1 - h(x^{(i)}, \theta))$ are bounded between 0 and 1. Since we are trying to maximize $h(\theta, x^{(i)})$ in $L(\theta)$, we can introduce the log and just maximize the log of the function. (We are maximizing the same function just in a different space). Introducing the log, allows us to write the log of a product as the sum of each log. Here are two identities that will come in handy:

$$\log a * b * c = \log a + \log b + \log c$$

$$\log a^b = b \log a$$

Given the two identities above, we can rewrite the equation as follows:

$$\max_{h(x^{(i)},\theta)} \log L(\theta) = \log \prod_{i=1}^{m} h(x^{(i)}, \theta)^{y^{(i)}} (1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} \log h(x^{(i)}, \theta)^{y^{(i)}} (1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} \log h(x^{(i)}, \theta)^{y^{(i)}} + \log(1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}, \theta) + (1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))$$

Hence, we now divide by m, because we want to see the average cost.

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

Remember that we were maximizing $h(\theta, x^{(i)})$ in the equation above. It turns out that maximizing an equation is the same as minimizing its negative. Think of x^2, feel free to plot it to see that for you yourself. Hence we add a negative sign and we end up minimizing the cost function as follows.

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

A vectorized implementation is:

$$\begin{aligned} h &= g(X\theta) \\ J(\theta) &= \frac{1}{m} \cdot \left(-y^T \log(h) - (1 - y)^T \log(1 - h)\right) \end{aligned}$$

This is an optional reading where I explain gradient descent in more detail. Remember, previously I gave you the gradient update step, but did not explicitly explain what is going on behind the scenes.

The general form of gradient descent is defined as:

$$
\begin{aligned}
&Repeat\ \{ \\
&\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&\}
\end{aligned}
$$

For all j. We can work out the derivative part using calculus to get:

$$
\begin{aligned}
&Repeat\ \{ \\
&\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h(x^{(i)}, \theta) - y^{(i)}) x_j^{(i)} \\
&\}
\end{aligned}
$$

A vectorized implementation is:

$\theta := \theta - \frac{\alpha}{m} X^T (H(X, \theta) - Y)$

Partial derivative of J(θ)

First calculate derivative of sigmoid function (it will be useful while finding partial derivative of J(θ)):

$$
\begin{aligned}
h(x)' &= \left(\frac{1}{1+e^{-x}} \right)' = \frac{-(1+e^{-x})'}{(1+e^{-x})^2} = \frac{-1' - (e^{-x})'}{(1+e^{-x})^2} = \frac{0 - (-x)'(e^{-x})}{(1+e^{-x})^2} = \frac{-(-1)(e^{-x})}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \left(\frac{1}{1+e^{-x}} \right) \left(\frac{e^{-x}}{1+e^{-x}} \right) = h(x) \left(\frac{+1 - 1 + e^{-x}}{1+e^{-x}} \right) = h(x) \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}} \right) = h(x)(1 - h(x))
\end{aligned}
$$

Note that we computed the partial derivative of the sigmoid function. If we were to derive $h(x^{(i)}, \theta)$ with respect to θ_j, you would get $h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta))x_j^{(i)}$. Note that we used the chain rule there, because we multiply by the derivative of $\theta^T x^{(i)}$ with respect to θ_j. Now we are ready to find out resulting partial derivative:

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} log(h(x^{(i)}, \theta)) + (1 - y^{(i)}) log(1 - h(x^{(i)}, \theta)) \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \frac{\partial}{\partial \theta_j} log(h(x^{(i)}, \theta)) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} log(1 - h(x^{(i)}, \theta)) \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} \frac{\partial}{\partial \theta_j} h(x^{(i)}, \theta)}{h(x^{(i)}, \theta)} + \frac{(1 - y^{(i)}) \frac{\partial}{\partial \theta_j} (1 - h(x^{(i)}, \theta))}{1 - h(x^{(i)}, \theta)} \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} \frac{\partial}{\partial \theta_j} h(x^{(i)}, \theta)}{h(x^{(i)}, \theta)} + \frac{(1 - y^{(i)}) \frac{\partial}{\partial \theta_j} (1 - h(x^{(i)}, \theta))}{1 - h(x^{(i)}, \theta)} \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{h(x^{(i)}, \theta)} + \frac{-(1 - y^{(i)}) h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{1 - h(x^{(i)}, \theta)} \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)} h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{h(x^{(i)}, \theta)} - \frac{(1 - y^{(i)}) h(x^{(i)}, \theta)(1 - h(x^{(i)}, \theta)) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)}}{1 - h(x^{(i)}, \theta))} \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} (1 - h(x^{(i)}, \theta)) x_j^{(i)} - (1 - y^{(i)}) h(x^{(i)}, \theta) x_j^{(i)} \right] \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} (1 - h(x^{(i)}, \theta)) - (1 - y^{(i)}) h(x^{(i)}, \theta) \right] x_j^{(i)} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} - y^{(i)} h(x^{(i)}, \theta) - h(x^{(i)}, \theta) + y^{(i)} h(x^{(i)}, \theta) \right] x_j^{(i)} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} - h(x^{(i)}, \theta) \right] x_j^{(i)} \\
&= \frac{1}{m} \sum_{i=1}^{m} \left[h(x^{(i)}, \theta) - y^{(i)} \right] x_j^{(i)}
\end{aligned}
$$

The vectorized version:

$\nabla J(\theta) = \frac{1}{m} \cdot X^T \cdot (H(X, \theta) - Y)$

Congratulations, you now know the full derivation of logistic regression :) !

deeplearning.ai

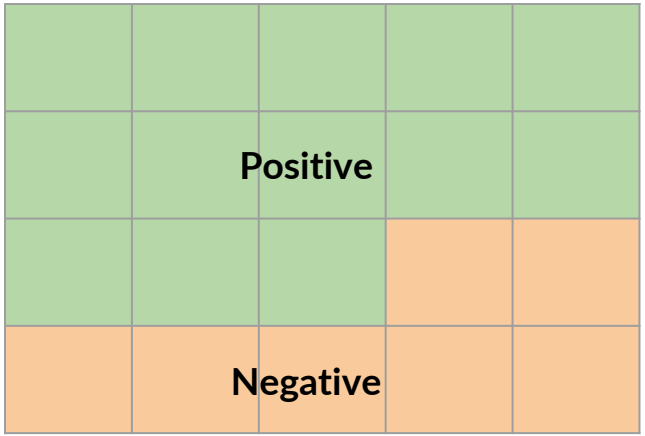
Probability and Bayes' Rule

Outline

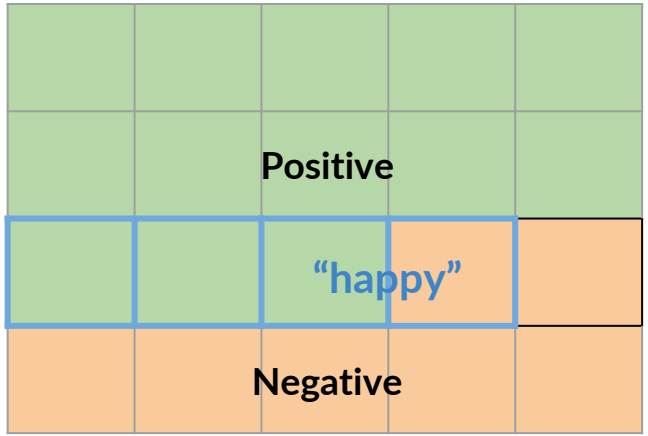
- Probabilities
- Bayes' rule (Applied in different fields, including NLP)
- Build your own Naive-Bayes tweet classifier!

Introduction

Corpus of tweets

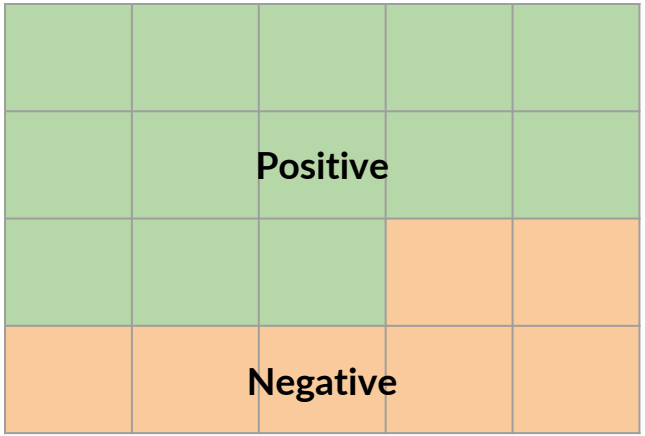


Tweets containing the word “happy”



Probabilities

Corpus of tweets

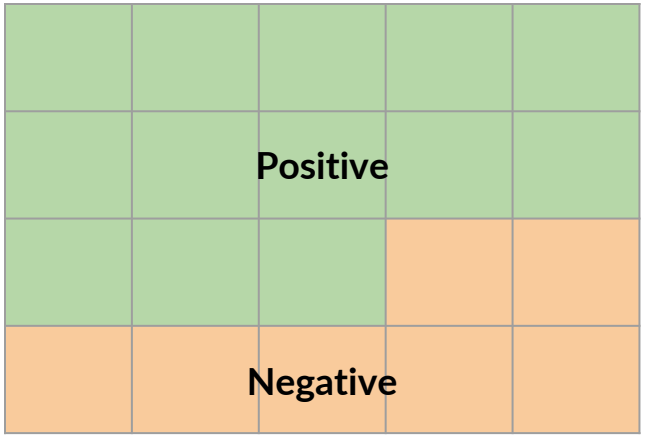


A → Positive tweet

$P(A) = P(\text{Positive}) = N_{pos} / N$

Probabilities

Corpus of tweets



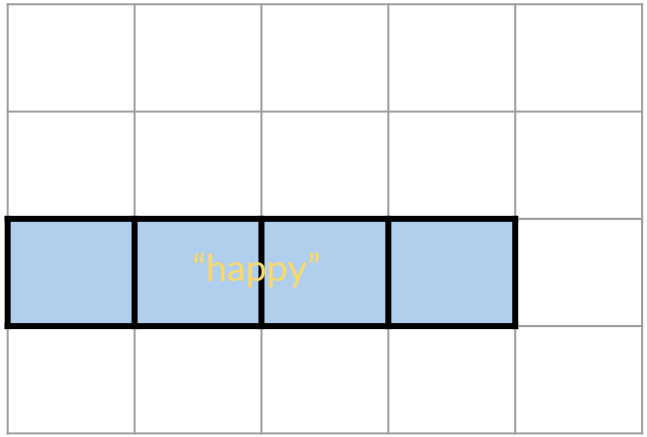
A → Positive tweet

$P(A) = N_{pos} / N = 13 / 20 = 0.65$

P(Negative) = 1 - P(Positive) = 0.35

Probabilities

Tweets containing the word “happy”

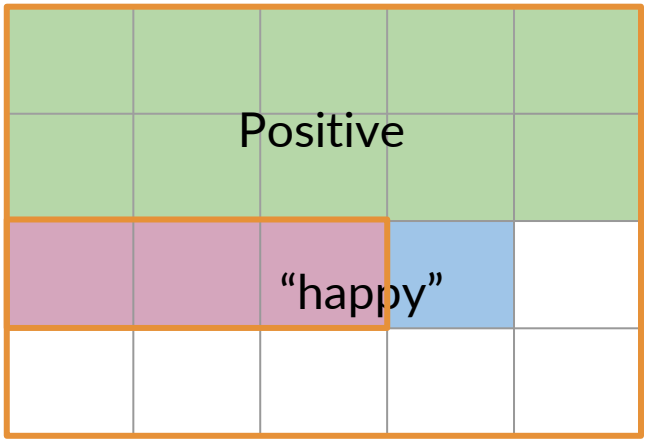


B → tweet contains “happy”.

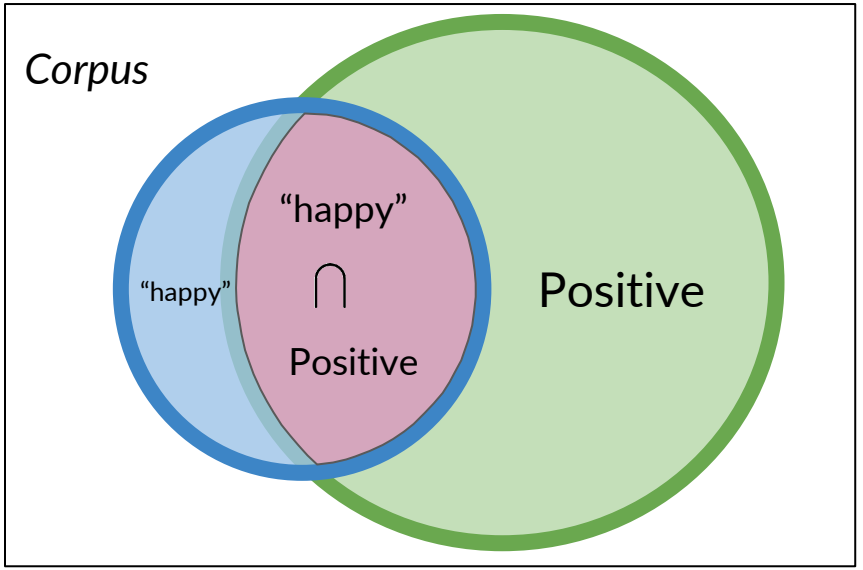
$P(B) = P(happy) = N_{happy} / N$

$P(B) = 4 / 20 = 0.2$

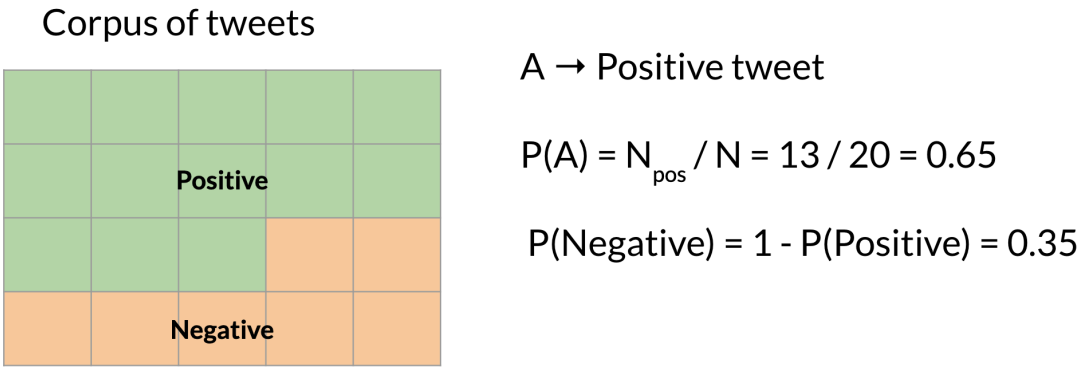
Probability of the intersection



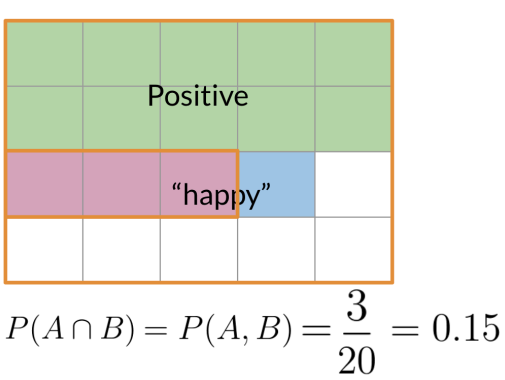
$$P(A \cap B) = P(A, B) = \frac{3}{20} = 0.15$$



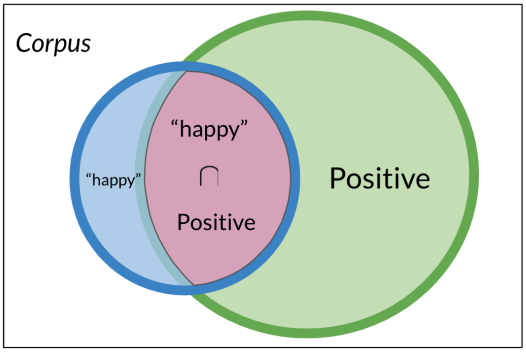
You learned about probabilities and Bayes' rule.



To calculate a probability of a certain event happening, you take the count of that specific event and you divide by the sum of all events. Furthermore, the sum of all probabilities has to equal 1.



$$P(A \cap B) = P(A, B) = \frac{3}{20} = 0.15$$

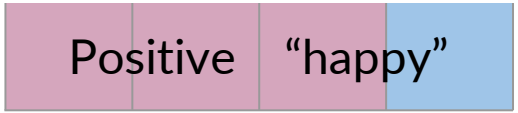


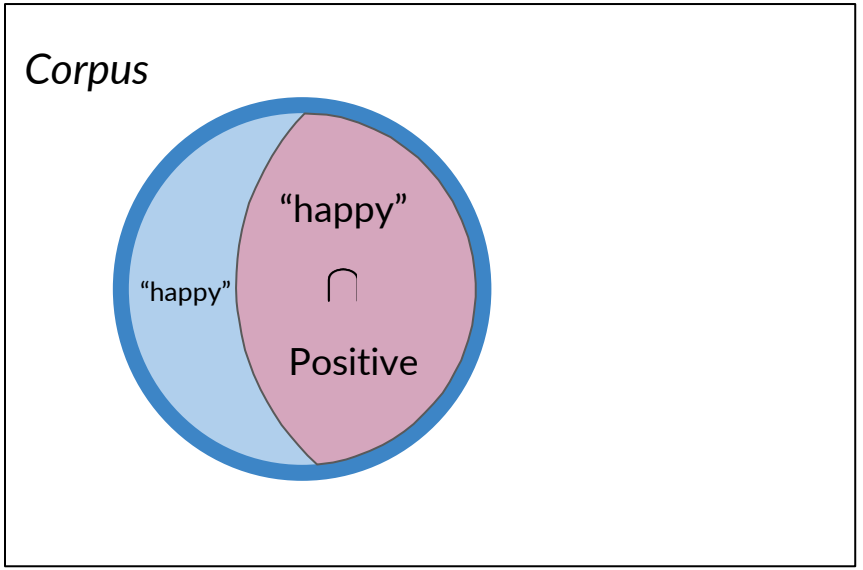
To compute the probability of 2 events happening, like "happy" and "positive" in the picture above, you would be looking at the intersection, or overlap of events. In this case red and blue boxes overlap in 3 boxes. So the answer is $\frac{3}{20}$.

deeplearning.ai

Bayes' Rule

Conditional Probabilities

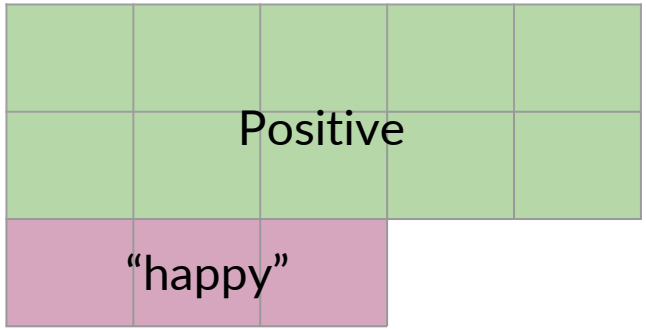


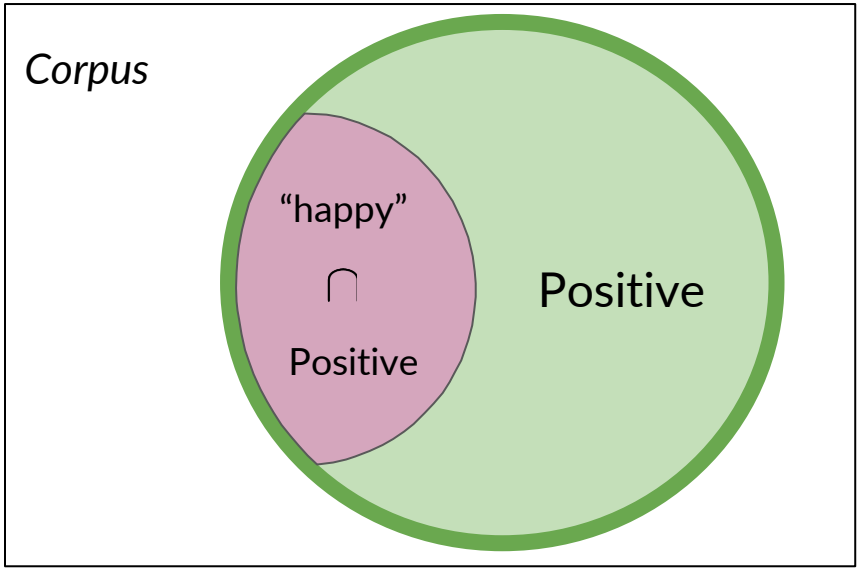


P(A | B) = P(Positive | “happy”)

P(A | B) = 3 / 4 = 0.75

Conditional Probabilities

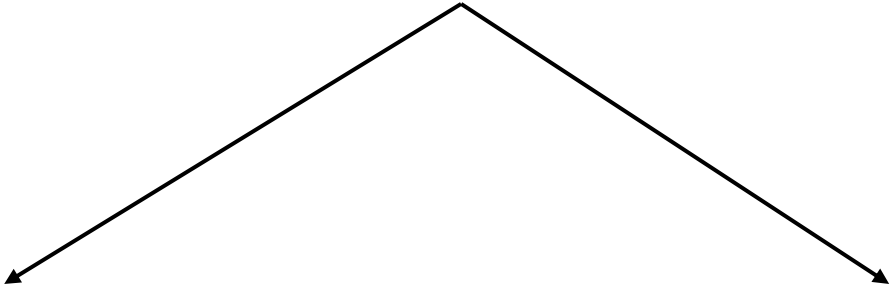




P(B | A) = P(“happy”| Positive)

P(B | A) = 3 / 13 = 0.231

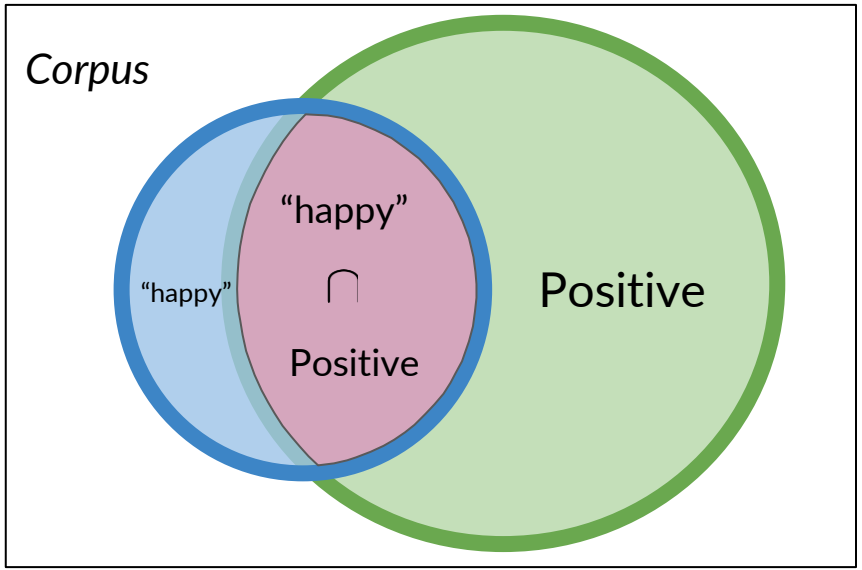
Conditional probabilities



Probability of B, given A happened

Looking at the elements of set A, the chance that one also belongs to set B

Conditional probabilities



$$P(\text{Positive} | \text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

Bayes' rule

$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

$$P(\text{“happy”}|\text{Positive}) = \frac{P(\text{“happy”} \cap \text{Positive})}{P(\text{Positive})}$$

Quiz

Objective: Derive Bayes' rule from the equations given on the last slide.

Question:

From the equations presented below, express the probability of a tweet being positive given that it contains the word happy in terms of the probability of a tweet containing the word happy given that it is positive

$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})} \qquad P(\text{“happy”}|\text{Positive}) = \frac{P(\text{“happy”} \cap \text{Positive})}{P(\text{Positive})}$$

Type: Multiple Choice, single answer

Options and solution:

$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$ That's right. You just derived Bayes' rule.

$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{“happy”})}{P(\text{Positive})}$ The ratio is upside-down in this equation.

$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”} \cap \text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$ Your result should not include any intersection probabilities.

$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”} \cap \text{Positive}) \times \frac{P(\text{“happy”})}{P(\text{Positive})}$ Your result should not include any intersection probabilities.

Bayes' rule

$$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$$

$$P(X|Y) = P(Y|X) \times \frac{P(X)}{P(Y)}$$

Quiz: Bayes' Rule Applied

Objective: Compute conditional probability using Bayes Rule

Question:

Here, again, is Bayes' rule:

$$P(X|Y) = P(Y|X) \times \frac{P(X)}{P(Y)}$$

Suppose that in your dataset, 25% of the positive tweets contain the word 'happy'. You also know that a total of 13% of the tweets in your dataset contain the word 'happy', and that 40% of the total number of tweets are positive. You observe the tweet: "happy to learn NLP'. What is the probability that this tweet is positive?

Type: Multiple Choice, single answer

Options and solution:

A: P(Positive | "happy") = 0.77 That's right. You just applied Bayes' rule.

B: P(Positive | "happy") = 0.08 Oops, looks like you might have the ratio of P(X) and P(Y) upside-down.

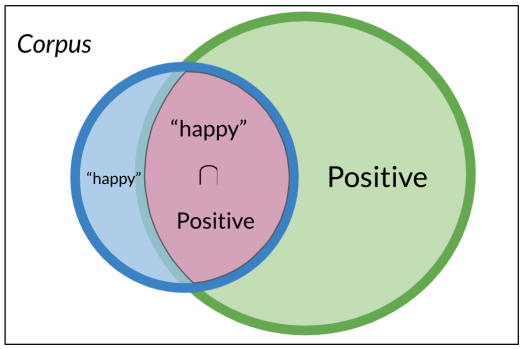
C: P(Positive | "happy") = 0.10 Remember to calculate the ratio in the formula for Bayes' rule.

D: P(Positive | "happy") = 1.92 Did you use the probability of a tweet being positive? Remember that a fractional probability must be between 0 and 1.

Summary

- Conditional probabilities $\longrightarrow$ Bayes' Rule
- $P(X|Y) = P(Y|X) \times \frac{P(X)}{P(Y)}$

Conditional probabilities help us reduce the sample search space. For example given a specific event already happened, i.e. we know the word is happy:



$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

Then you would only search in the blue circle above. The numerator will be the red part and the denominator will be the blue part. This leads us to conclude the following:

$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

$$P(\text{“happy”}|\text{Positive}) = \frac{P(\text{“happy”} \cap \text{Positive})}{P(\text{Positive})}$$

Substituting the numerator in the right hand side of the first equation, you get the following:

$$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$$

Note that we multiplied by P(positive) to make sure we don't change anything. That concludes Bayes Rule which is defined as

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}.$$

deeplearning.ai

Naïve Bayes Introduction

(NB is an example of supervised ML)

Naïve Bayes for Sentiment Analysis

Naive: Because this method make the assumption that the features you're using for classification are all independent (which in reality is rarely the case)

Positive tweets

I am happy because I am learning NLP

I am happy, not sad.

Negative tweets

I am sad, I am not learning NLP

I am sad, not happy

word	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~~ 13

$P(w_i \mid \text{class})$

word	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
Nclass	13	12

$$p(I|Pos) = \frac{3}{13}$$

word	Pos	Neg
I	0.24	0.24

$P(w_i \mid \text{class})$

word	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
Nclass	**13**	**12**

word	Pos	Neg	
I	0.24	0.25	0.24
am	0.24	0.25	0.24
happy	0.15	0.08	0.07
because	0.08	0.00	0
learning	0.08	0.08	0.07
NLP	0.08	0.08	0.07
sad	0.08	0.17	0.15
not	0.08	0.17	0.15
Sum	**1**	**1**	

$P(w_i \mid class)$

nearly identical conditional prob * words that are equally probable, don't add anything to the sentement

power words. carry alot of weight in determining your tweet sentement

it's problematic! it's conditional towards neg class = 0 -> would make problem in calculations

word	Pos	Neg	
I	0.24	0.25	0.24
am	0.24	0.25	0.24
happy	0.15	0.08	0.07
because	0.08	0	0
learning	0.08	0.08	0.07
NLP	0.08	0.08	0.07
sad	0.08	0.17	0.15
not	0.08	0.17	0.15

Naïve Bayes

today is not in vocab, so we don't include any term in our score

Tweet: I am happy today; I am learning.

Naive bayes inference condition rule for binary classification

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

score greater than 1 indicates that the class is positive, otherwise it is negative.

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

word	Pos	Neg
I	0.20	0.20
am	0.20	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	0.10
not	0.10	0.15

Summary

- Naive Bayes inference condition rule for binary classification

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

- Table of probabilities

To build a classifier, we will first start by creating conditional probabilities given the following table:

Positive tweets

I am happy because I am learning NLP

I am happy, not sad.

Negative tweets

I am sad, I am not learning NLP

I am sad, not happy

word	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

This allows us compute the following table of probabilities:

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	0.17
not	0.08	0.17

Once you have the probabilities, you can compute the likelihood score as follows

Tweet: I am happy today; I am learning.

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

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

word	Pos	Neg
I	0.20	0.20
am	0.20	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

A score greater than 1 indicates that the class is positive, otherwise it is negative.

deeplearning.ai

Laplacian Smoothing

Laplacian Smoothing

A technique you can use to avoid your probabilities being zero

$$P(w_i|class) = \frac{freq(w_i, class)}{N_{class}}$$

class ∈ {Positive, Negative}

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

N_{class} = frequency of all words in class

V = number of unique words in Vocab

Introducing $P(w_i \mid class)$ with smoothing

word	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
Nclass	**13**	**12**

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

V = 8

word	Pos	Neg
I	0.19	

Introducing $P(w_i \mid \text{class})$ with smoothing

word	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
Nclass	**13**	**12**

Laplacian Smoothing

$V = 8$

word	Pos	Neg	
I	0.19	0.20	0.19
am	0.19	0.20	0.19
happy	0.14	0.10	0.10
because	0.10	0.05	0.04
learning	0.10	0.10	
NLP	0.10	0.10	
sad	0.10	0.15	0.14
not	0.10	0.15	0.14
Sum	**1**	**1**	

Summary

- Laplacian smoothing to avoid $P(w_i|class) = 0$

- Naïve Bayes formula

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

We usually compute the probability of a word given a class as follows:

$$P\left(\mathrm{w_i}|\ \text{class}\ \right) = \frac{\text{freq}\ (\mathrm{w_i}, \text{class}\ )}{\mathrm{N_{class}}} \quad \text{class}\ \in \{\ \text{Positive, Negative}\ \}$$

However, if a word does not appear in the training, then it automatically gets a probability of 0, to fix this we add smoothing as follows

$$P\left(\mathrm{w_i}\ |\ \text{class}\right) = \frac{\text{freq}(\mathrm{w_i}, \text{class}\ )+1}{(\mathrm{N_{class}} + \mathrm{V})}$$

Note that we added a 1 in the numerator, and since there are V words to normalize, we add V in the denominator.

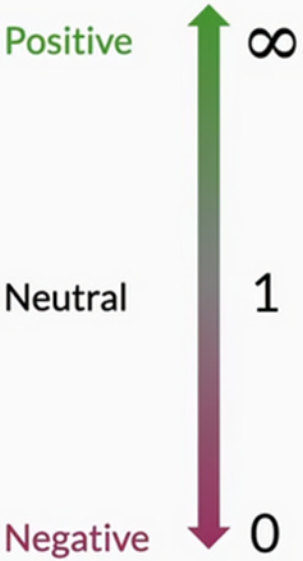
N_{class}: frequency of all words in class

V: number of unique words in vocabulary

deeplearning.ai

Log Likelihood, Part 1

Ratio of probabilities



word	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

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

$$\approx \frac{freq(w_i, 1) + 1}{freq(w_i, 0) + 1}$$

Naïve Bayes' inference

class $\in$ *{pos, neg}*

w *-> Set of m words in a tweet*

prior ratio

the likelihood

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

- A simple, fast, and powerful baseline
- A probabilistic model used for classification

Log Likelihood

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

- Products bring risk of underflow
- $log(a * b) = log(a) + log(b)$
- $log(\frac{P(pos)}{P(neg)} \prod_{i=1}^{n} \frac{P(w_i|pos)}{P(w_i|neg)}) \Rightarrow log\frac{P(pos)}{P(neg)} + \sum_{i=1}^{n} log\frac{P(w_i|pos)}{P(w_i|neg)}$

all unique words!

log prior + log likelihood

Summing the Lambdas

doc: I am happy because I am learning.

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

$$\lambda(\text{happy}) = log\frac{0.09}{0.01} \approx 2.2$$

log score = sum all the lambdas

word	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

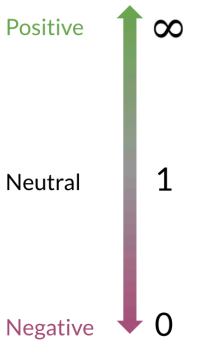
Summary

- Word sentiment

$$ratio(w) = \frac{P(w|pos)}{P(w|neg)}$$

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

To compute the log likelihood, we need to get the ratios and use them to compute a score that will allow us to decide whether a tweet is positive or negative. The higher the ratio, the more positive the word is:



word	Pos	Neg	ratio
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	

$$\text{ratio}(w_i) = \frac{P(w_i \mid \text{Pos})}{P(w_i \mid \text{Neg})} \approx \frac{freq(w_i, 1) + 1}{freq(w_i, 0) + 1}$$

To do inference, you can compute the following:

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

As m gets larger, we can get numerical flow issues, so we introduce the log, which gives you the following equation:

$$\log\left(\frac{P(pos)}{P(neg)} \prod_{i=1}^{n} \frac{P(w_i|pos)}{P(w_i|neg)}\right) \Rightarrow \log \frac{P(pos)}{P(neg)} + \sum_{i=1}^{n} \log \frac{P(w_i|pos)}{P(w_i|neg)}$$

The first component is called the log prior and the second component is the log likelihood. We further introduce λ as follows:

doc: I am happy because I am learning.

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

$$\lambda(\text{happy}) = log \frac{0.09}{0.01} \approx 2.2$$

word	Pos	Neg	λ
I	0.05	0.05	0
am	0.04	0.04	0
happy	0.09	0.01	
because	0.01	0.01	
learning	0.03	0.01	
NLP	0.02	0.02	
sad	0.01	0.09	
not	0.02	0.03	

Having the λ dictionary will help a lot when doing inference.

deeplearning.ai

Log Likelihood, Part 2

Log Likelihood

doc: I am happy because I am learning.

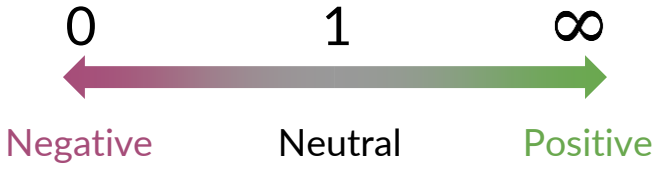
$$\sum_{i=1}^{m} log \frac{P(w_i|pos)}{P(w_i|neg)} = \sum_{i=1}^{m} \lambda(w_i)$$

log likelihood = 0 + 0 + 2.2 + 0 + 0 + 0 + 1.1 = **3.3**

word	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

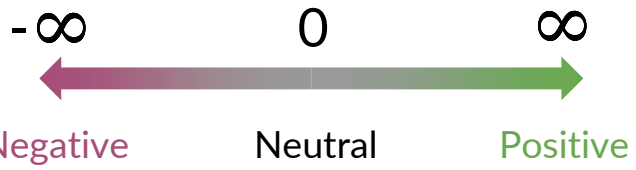
Log Likelihood

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



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

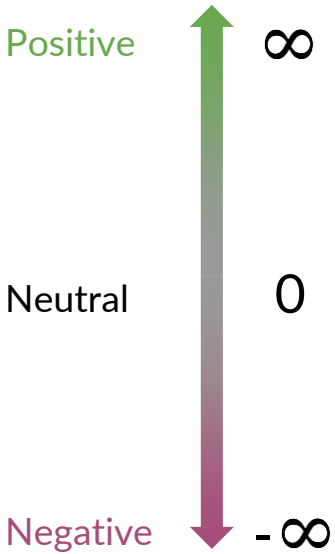
3.3 > 0



Summary

Tweet sentiment:

$$log \prod_{i=1}^{m} ratio(w_i) = \sum_{i=1}^{m} \lambda(w_i) \quad > 0$$



Once you computed the λ dictionary, it becomes straightforward to do inference:

doc: I am happy because I am learning.

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

log likelihood = 0 + 0 + 2.2 + 0 + 0 + 0 + 1.1 = **3.3**

word	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

As you can see above, since $3.3 > 0$, we will classify the document to be positive. If we got a negative number we would have classified it to the negative class.

deeplearning.ai

Training Naïve Bayes

Outline

- Five steps for training a Naïve Bayes model

Training Naïve Bayes

- Lowercase
- Remove punctuation, urls, names
- Remove stop words
- Stemming
- Tokenize sentences

Step 0: Collect and annotate corpus

Positive tweets

I am happy because I am learning NLP

I am happy, not sad. @NLP

Negative tweets

I am sad, I am not learning NLP

I am sad, not happy!!

Step 1: Preprocess

Positive tweets

[happi, because, learn, NLP]

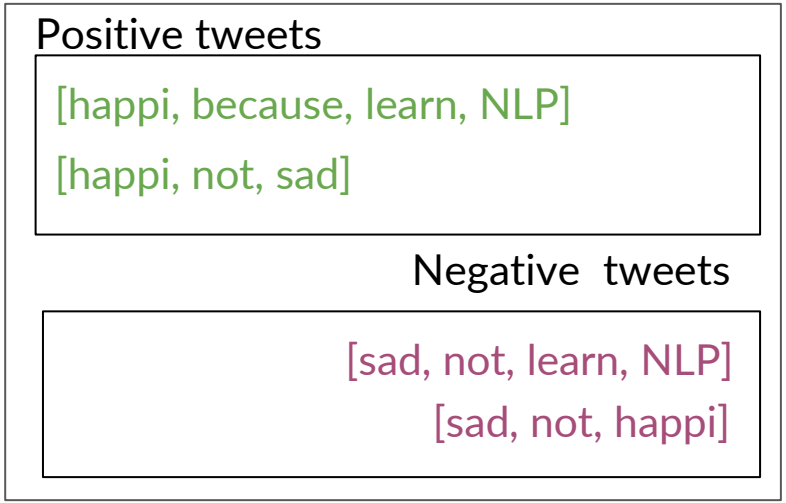
[happi, not, sad]

Negative tweets

[sad, not, learn, NLP]

[sad, not, happi]

Training Naïve Bayes



Step 2: Word count

freq(w, class)

word	Pos	Neg
happi	2	1
because	1	0
learn	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	7	7

Training Naïve Bayes

freq(w, class)

word	Pos	Neg
happi	2	1
because	1	0
learn	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	7	7

Step 3: $P(\mathrm{w}|\mathrm{class})$

$V = 6$

$$\frac{\mathrm{freq}(\mathrm{w}, \mathrm{class}) + 1}{N_{\mathrm{class}} + V}$$

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

Step 4: Get lambda

word	Pos	Neg	λ
happy	0.23	0.15	0.43
because	0.15	0.07	0.6
learning	0.08	0.08	0
NLP	0.08	0.08	0
sad	0.08	0.17	-0.75
not	0.08	0.17	-0.75

Training Naïve Bayes

Step 5:
Get the
log prior

D_{pos} = Number of positive tweets
D_{neg} = Number of negative tweets

$$\text{logprior} = log\frac{D_{pos}}{D_{neg}}$$

If dataset is balanced, $D_{pos} = D_{neg}$ and logprior = 0.

Summary

1. Get or annotate a dataset with positive and negative tweets
2. Preprocess the tweets: process_tweet(tweet) → $[w_1, w_2, w_3, ...]$
3. Compute freq(w, class)
4. Get P(w | pos), P(w | neg)
5. Get λ(w)
6. Compute logprior = log(P(pos) / P(neg))

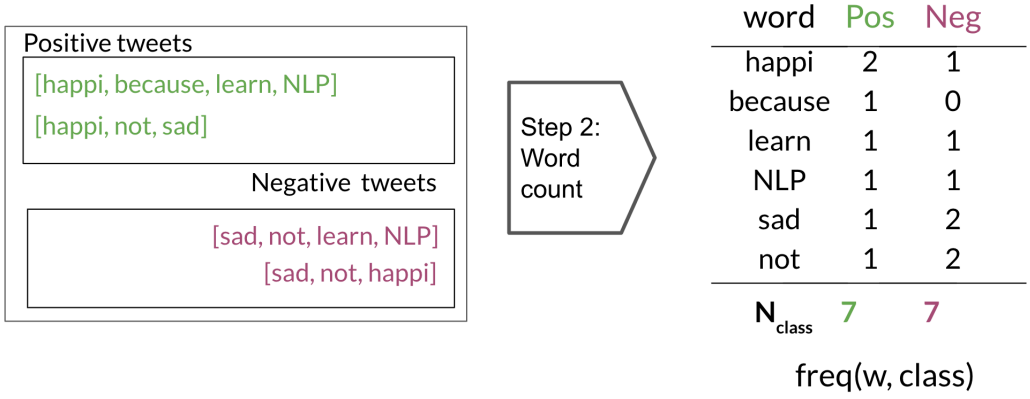
To train your naïve Bayes classifier, you have to perform the following steps:

1) Get or annotate a dataset with positive and negative tweets

2) Preprocess the tweets: process_tweet(tweet) ➞ [w1, w2, w3, ...]:

- Lowercase
- Remove punctuation, urls, names
- Remove stop words
- Stemming
- Tokenize sentences

3) Compute freq(w, class):



word	Pos	Neg
happi	2	1
because	1	0
learn	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	7	7

freq(w, class)

4) Get $P(w|pos), P(w|neg)$

You can use the table above to compute the probabilities.

5) Get $\lambda(w)$

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

6) Compute $logprior = \log(P(pos)/P(neg))$

$\text{logprior} = \log \frac{D_{pos}}{D_{neg}}$, where D_{pos} and D_{neg} correspond to the number of positive and negative documents respectively.

deeplearning.ai

Testing Naïve Bayes

Outline

- Predict using a Näive Bayes Model
- Using your validation set to compute model accuracy

Predict using Naïve Bayes

- log-likelihood dictionary $\lambda(w) = log\frac{P(w|pos)}{P(w|neg)}$

$$logprior = log\frac{D_{pos}}{D_{neg}} = 0$$

- Tweet: [I, pass, the, NLP, interview] 

$$score = -0.01 + 0.5 - 0.01 + 0 + logprior = 0.48$$

$$pred = score > 0$$

word	λ
I	-0.01
the	-0.01
happi	0.63
because	0.01
pass	0.5
NLP	0
sad	-0.75
not	-0.75

Testing Naïve Bayes

- X_{val} Y_{val} λ $logprior$

$$score = predict(X_{val}, \lambda, logprior)$$

$$pred = score > 0$$

$$\begin{bmatrix} 0.5 \\ -1 \\ 1.3 \\ \vdots \\ score_m \end{bmatrix} > 0 = \begin{bmatrix} 0.5 > 0 \\ -1 > 0 \\ 1.3 > 0 \\ \vdots \\ score_m > 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

Testing Naïve Bayes

- X_{val} Y_{val} λ $logprior$

$$score = predict(X_{val}, \lambda, logprior)$$

$$pred = score > 0$$

$$\frac{1}{m}\sum_{i=1}^{m}(pred_i == Y_{val_i})$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m == Y_{val_m} \end{bmatrix}$$

Summary

- X_{val} Y_{val} $\longrightarrow$ Performance on unseen data
- Predict using λ and $logprior$ for each new tweet
- Accuracy $\longrightarrow$ $\frac{1}{m}\sum_{i=1}^{m}(pred_i == Y_{val_i})$
- What about words that do not appear in λ(w)?

- log-likelihood dictionary $\lambda(w) = log\frac{P(w|pos)}{P(w|neg)}$
- $logprior = log\frac{D_{pos}}{D_{neg}} = 0$
- Tweet: [I, pass, the, NLP, interview] 

$score = -0.01 + 0.5 - 0.01 + 0 + logprior = 0.48$

$pred = score > 0$

word	λ
I	-0.01
the	-0.01
happi	0.63
because	0.01
pass	0.5
NLP	0
sad	-0.75
not	-0.75

The example above shows how you can make a prediction given your λ dictionary. In this example the *logprior* is 0 because we have the same amount of positive and negative documents (i.e. $\log 1 = 0$).

deeplearning.ai

Applications of Naïve Bayes

Applications of Naïve Bayes

$$P(pos|tweet) \approx P(pos)P(tweet|pos)$$
$$P(neg|tweet) \approx P(neg)P(tweet|neg)$$

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

Applications of Naïve Bayes

Author identification:

$$\frac{P(\text{  }|\text{book})}{P(\text{  }|\text{book})}$$

Spam filtering:

$$\frac{P(\text{spam}|\text{email})}{P(\text{nonspam}|\text{email})}$$

Applications of Naïve Bayes

Information retrieval:

$$P(\text{document}_\text{k}|\text{query}) \propto \prod_{i=0}^{|query|} P(\text{query}_\text{i}|\text{document}_\text{k})$$

Retrieve document if $P(\text{document}_\text{k}|\text{query}) > \text{threshold}$

"Icon made by Vector Market from www.flaticon.com"

Applications of Naïve Bayes

Word disambiguation:

$$\frac{P(\text{river}|\text{text})}{P(\text{money}|\text{text})}$$

Bank:

"Pictures with CC"

Naïve Bayes Applications

- Sentiment analysis
- Author identification
- Information retrieval
- Word disambiguation
- Simple, fast and robust!

deeplearning.ai

Naïve Bayes Assumptions

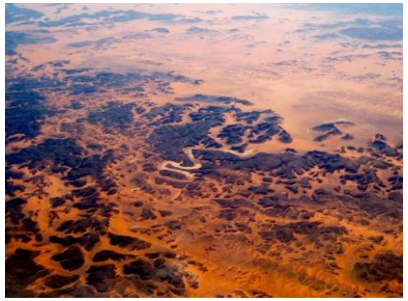
Outline

- Independence
- Relative frequency in corpus

Naïve Bayes Assumptions

- Independence

"It is sunny and hot in the Sahara desert."



Naïve Bayes Assumptions

"It's always cold and snowy in ___ ."

spring?? summer? fall?? winter??

all 4 of these have equal prob for Naive Bayes

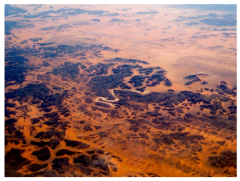
Naïve Bayes Assumptions

- Relative frequencies in corpus

Summary

- Independence: Not true in NLP
- Relative frequency of classes affect the model

Naïve Bayes makes the independence assumption and is affected by the word frequencies in the corpus. For example, if you had the following



"It is sunny and hot in the Sahara desert."

"It's always cold and snowy in ___."

In the first image, you can see the word sunny and hot tend to depend on each other and are correlated to a certain extent with the word "desert". Naive Bayes assumes independence throughout. Furthermore, if you were to fill in the sentence on the right, this naive model will assign equal weight to the words "spring, summer, fall, winter".

Relative frequencies in corpus

On Twitter, there are usually more positive tweets than negative ones. However, some "clean" datasets you may find are artificially balanced to have to the same amount of positive and negative tweets. Just keep in mind, that in the real world, the data could be much noisier.

deeplearning.ai

Error Analysis

Outline

- Removing punctuation and stop words
- Word order
- Adversarial attacks

Processing as a Source of Errors: Punctuation

Tweet: My beloved grandmother~~X~~(

processed_tweet: [belov, grandmoth]

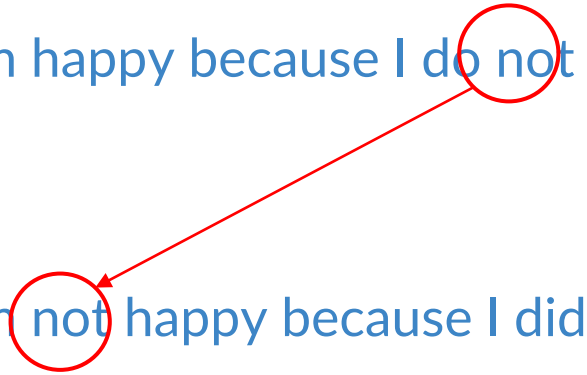
Processing as a Source of Errors: Removing Words

Tweet: This is not good, because your attitude is not even close to being nice.

processed_tweet: [good, attitude, close, nice]

Processing as a Source of Errors: Word Order

Tweet: I am happy because I do not go.



Tweet: I am not happy because I did go.

Adversarial attacks

Sarcasm, Irony and Euphemisms

Tweet: This is a ridiculously powerful movie. The plot was gripping and I cried right through until the ending!

processed_tweet: [ridicul, power, movi, plot, grip, cry, end]

Summary

- Removing punctuation
- Removing words
- Word order
- Adversarial attacks

There are several mistakes that could cause you to misclassify an example or a tweet. For example,

- Removing punctuation
- Removing words

Tweet: This is not good, because your attitude is not even close to being nice.

processed_tweet: [good, attitude, close, nice]

Tweet: My beloved grandmother :(

processed_tweet: [belov, grandmoth]

- Word order

Tweet: I am happy because I did not go. 😆

Tweet: I am not happy because I did go. 😭

- Adversarial attacks

These include sarcasm, irony, euphemisms.

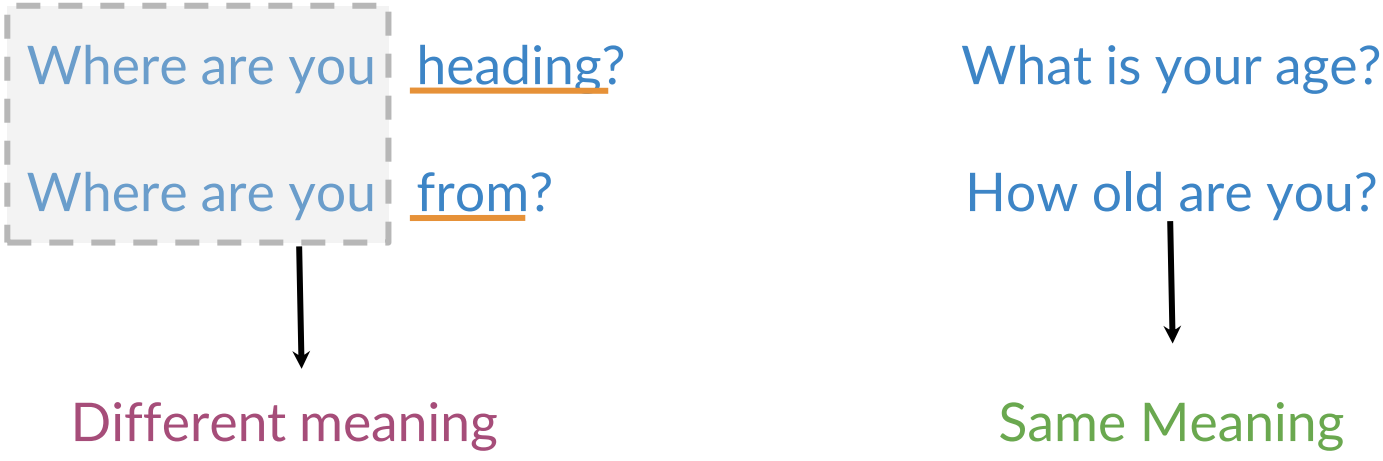
deeplearning.ai

Vector Space Models

Outline

- Vector space models
- Advantages
- Applications

Why learn vector space models?

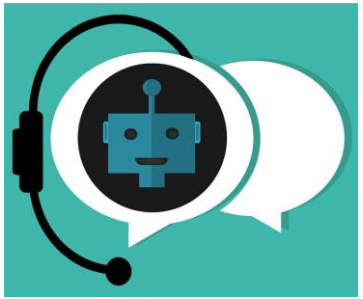


Vector space models applications

- You eat *cereal* from a *bowl*
- You *buy* something and someone else *sells* it

Information Extraction

Machine Translation

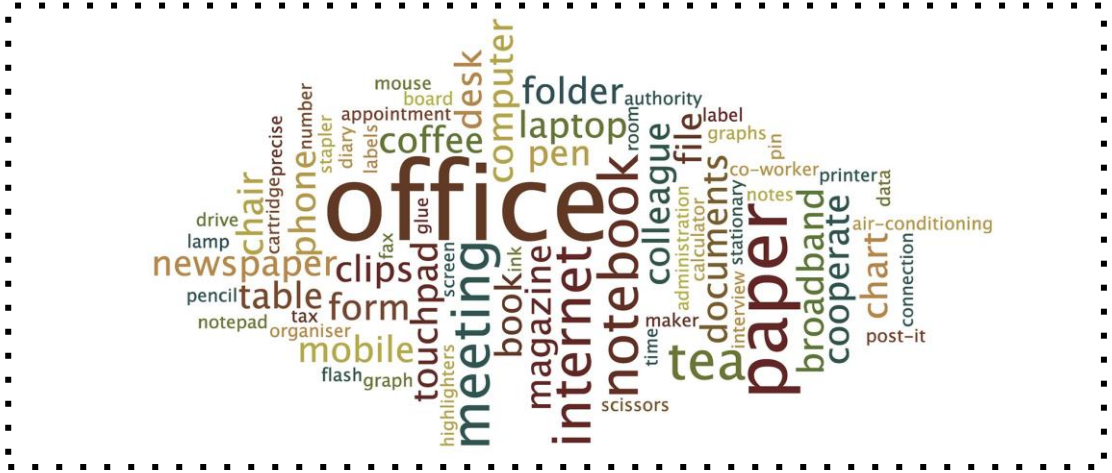


Chatbots

Fundamental concept

"You shall know a word by the company it keeps"

Firth, 1957



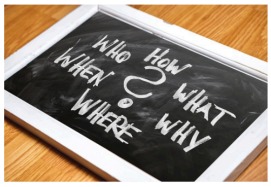
(Firth, J. R. 1957:11)

Summary

- Represent words and documents as vectors
- Representation that captures relative meaning

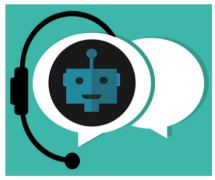
Vector spaces are fundamental in many applications in NLP. If you were to represent a word, document, tweet, or any form of text, you will probably be encoding it as a vector. These vectors are important in tasks like information extraction, machine translation, and chatbots. Vector spaces could also be used to help you identify relationships between words as follows:

- You eat *cereal* from a *bowl*
- You *buy* something and someone else *sells* it



Information Extraction

Machine Translation


Chatbots

The famous quote by Firth says, **"You shall know a word by the company it keeps".** When learning these vectors, you usually make use of the neighboring words to extract meaning and information about the center word. If you were to cluster these vectors together, as you will see later in this specialization, you will see that adjectives, nouns, verbs, etc. tend to be near one another. Another cool fact, is that synonyms and antonyms are also very close to one another. This is because you can easily interchange them in a sentence and they tend to have similar neighboring words!

deeplearning.ai

Word by Word and Word by Doc.

Outline

- Co-occurrence ⟶ Vector representation
- Relationships between words/documents

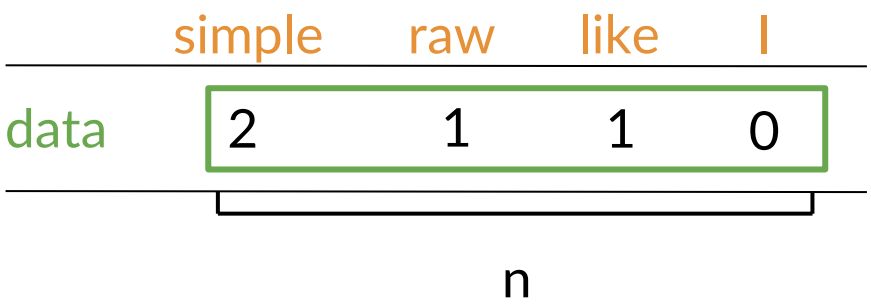
Word by Word Design

Number of times they *occur together within a certain distance k*

I like simple data

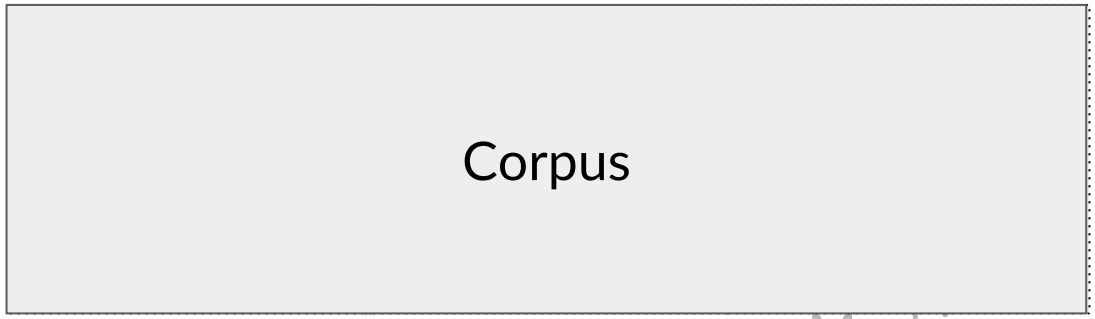
I prefer simple raw data

k=2



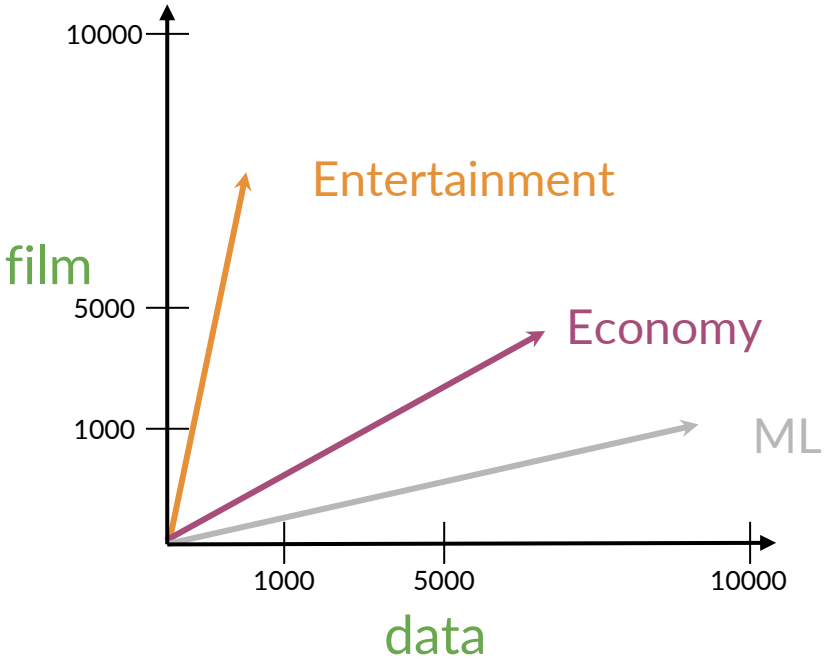
Word by Document Design

Number of times a word *occurs within a certain category*



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

Vector Space



	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Measures of “similarity:”
Angle
Distance

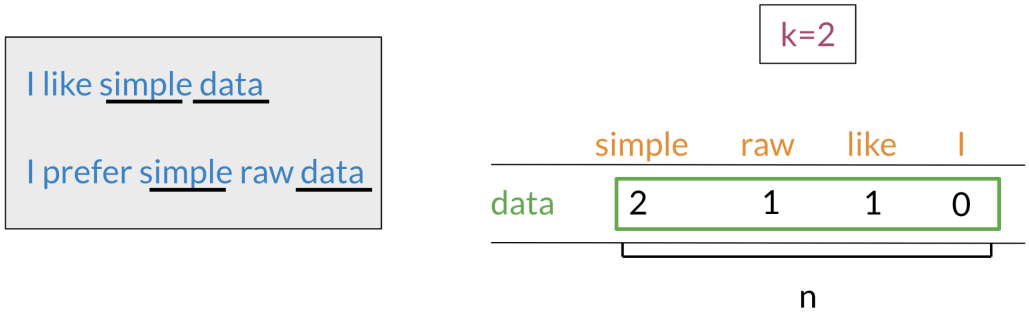
Summary

- W/W and W/D, *counts* of occurrence

- Vector Spaces ⟶ Similarity between words/documents

Word by Word Design

We will start by exploring the word by word design. Assume that you are trying to come up with a vector that will represent a certain word. One possible design would be to create a matrix where each row and column corresponds to a word in your vocabulary. Then you can iterate over a document and see the number of times each word shows up next each other word. You can keep track of the number in the matrix. In the video I spoke about a parameter K. You can think of K as the bandwidth that decides whether two words are next to each other or not.



In the example above, you can see how we are keeping track of the number of times words occur together within a certain distance k. At the end, you can represent the word data, as a vector $v = [2, 1, 1, 0]$.

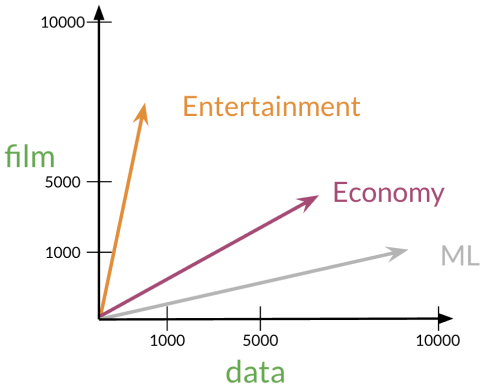
Word by Document Design

You can now apply the same concept and map words to documents. The rows could correspond to words and the columns to documents. The numbers in the matrix correspond to the number of times each word showed up in the document.

Corpus

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

You can represent the entertainment category, as a vector $v = [500, 7000]$. You can then also compare categories as follows by doing a simple plot.



	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Measures of "similarity:"
Angle
Distance

Later this week, you will see how you can use the angle between two vectors to measure similarity.

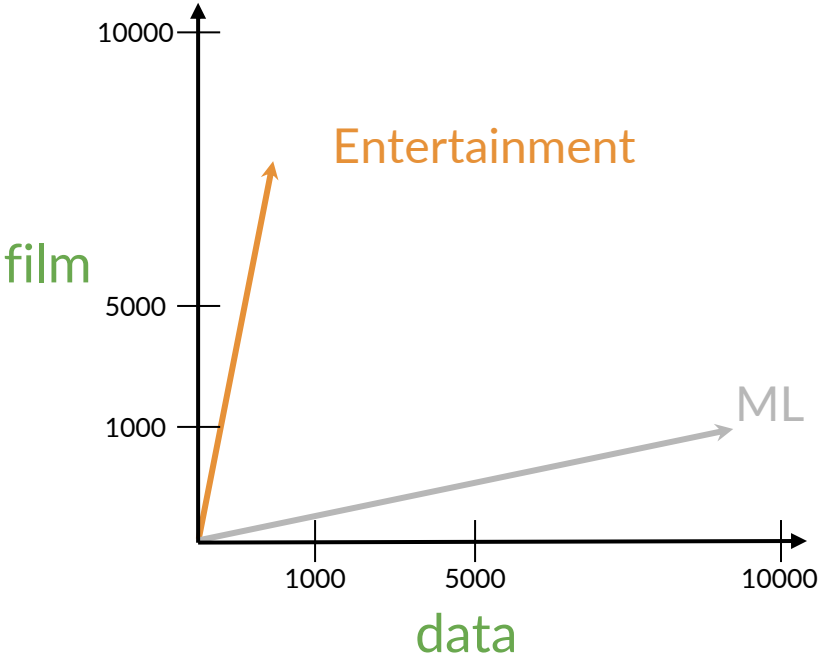
deeplearning.ai

Euclidean Distance

Outline

- Euclidean distance
- N-dimension vector representations comparison

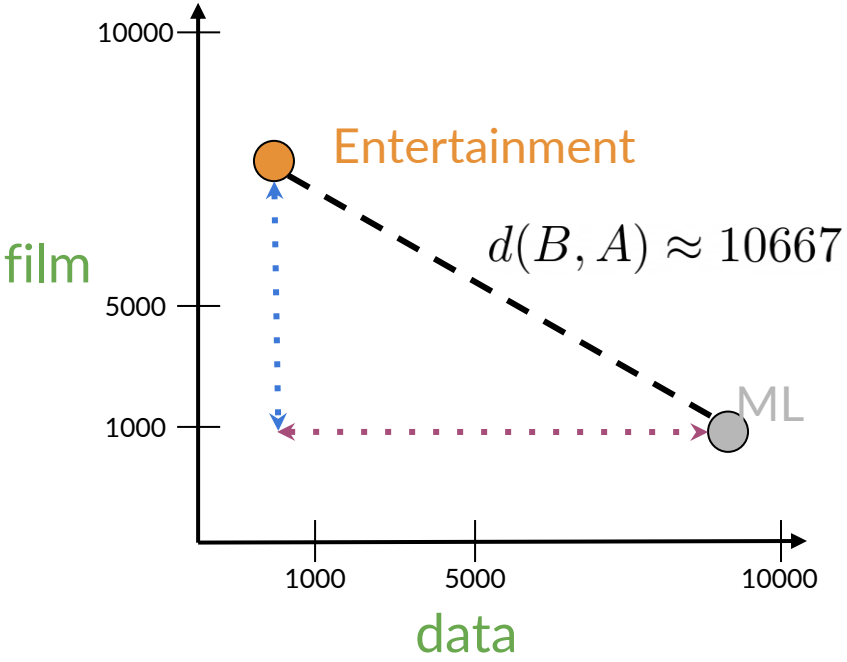
Euclidean distance



Corpus A: (500,7000)

Corpus B: (9320,1000)

Euclidean distance



Corpus A: (500,7000)

Corpus B: (9320,1000)

$$d(B, A) = \sqrt{(B_1 - A_1)^2 + (B_2 - A_2)^2}$$

$$c^2 = a^2 + b^2$$

$$d(B, A) = \sqrt{(-8820)^2 + (6000)^2}$$

Euclidean distance for n-dimensional vectors

	data	$\vec{w}$ boba	$\vec{v}$ ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$= \sqrt{(1-0)^2 + (6-4)^2 + (8-6)^2}$$

$$= \sqrt{1+4+4} = \sqrt{9} = 3$$

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

Norm of $(\vec{v} - \vec{w})$

Euclidean distance in Python

```
# Create numpy vectors v and w
v = np.array([1, 6, 8])
w = np.array([0, 4, 6])

# Calculate the Euclidean distance d
d = np.linalg.norm(v-w)
# Print the result
print("The Euclidean distance between v and w is: ", d)
```

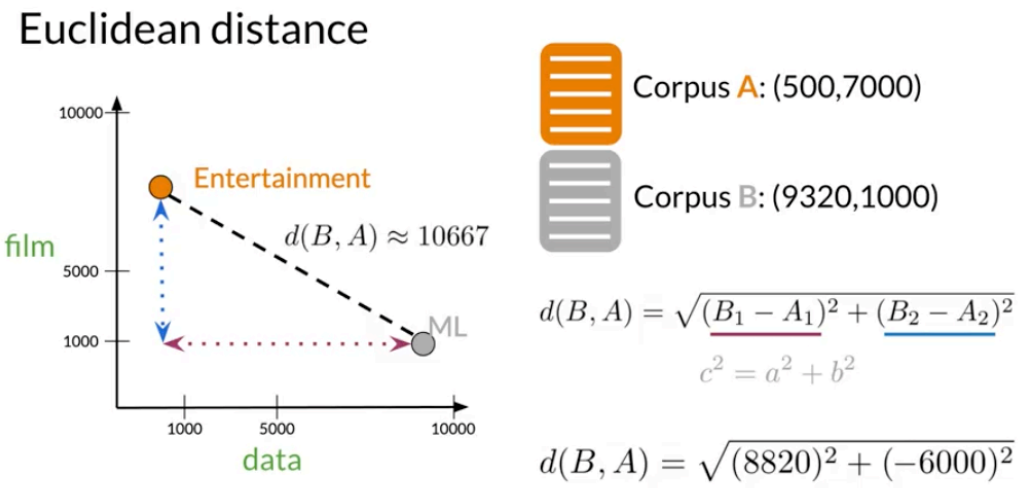
```
The Euclidean distance between v and w is: 3
```

Summary

- Straight line between points
- Norm of the difference between vectors

Let us assume that you want to compute the distance between two points: A, B. To do so, you can use the euclidean distance defined as

$$d(B, A) = \sqrt{((B_1 - A_1)^2 + (B_2 - A_2)^2)}$$



You can generalize finding the distance between the two points (A, B) to the distance between an n dimensional vector as follows:

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

Here is an example where I calculate the distance between 2 vectors ($n = 3$).

	data	$\vec{w}$ boba	$\vec{v}$ ice-cream
AI	6	0	1
drinks	0	4	6
food	0	6	8

$$= \sqrt{(1-0)^2 + (6-4)^2 + (8-6)^2}$$
$$= \sqrt{1+4+4} = \sqrt{9} = 3$$

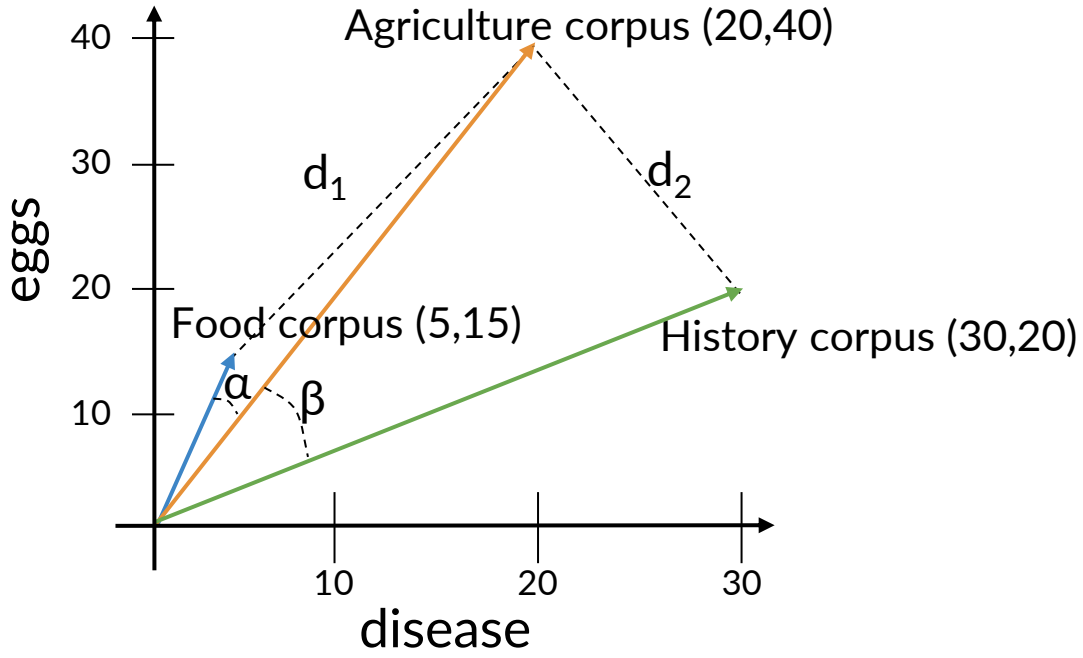
deeplearning.ai

Cosine Similarity: Intuition

Outline

- Problems with Euclidean Distance
- Cosine similarity

Euclidean distance vs Cosine similarity



Euclidean distance: $d_2 < d_1$

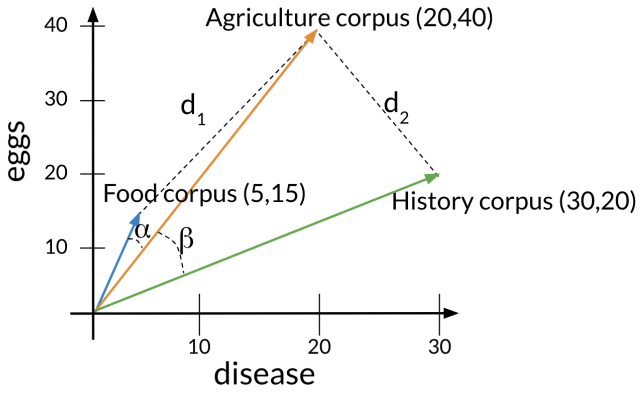
Angles comparison: $\beta > \alpha$

The cosine of the angle between the vectors

Summary

- Cosine similarity when corpora are different sizes

One of the issues with euclidean distance is that it is not always accurate and sometimes we are not looking for that type of similarity metric. For example, when comparing large documents to smaller ones with euclidean distance one could get an inaccurate result. Look at the diagram below:



Euclidean distance: $d_2 < d_1$

Angles comparison: $\beta > \alpha$

The cosine of the angle between the vectors

Normally the **food** corpus and the **agriculture** corpus are more similar because they have the same proportion of words. However the food corpus is much smaller than the agriculture corpus. To further clarify, although the history corpus and the agriculture corpus are different, they have a smaller euclidean distance. Hence $d_2 < d_1$.

To solve this problem, we look at the cosine between the vectors. This allows us to compare \Beta and α.

deeplearning.ai

Cosine Similarity

Outline

- How to get the cosine of the angle between two vectors
- Relation of this metric to similarity

Previous definitions

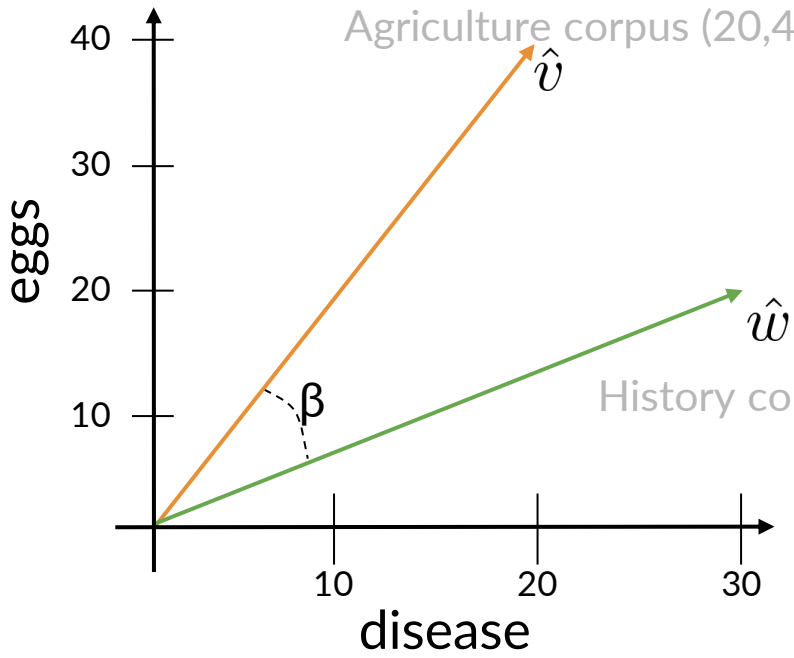
Vector norm

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Dot product

$$\vec{v}.\vec{w} = \sum_{i=1}^{n} v_i.w_i$$

Cosine Similarity



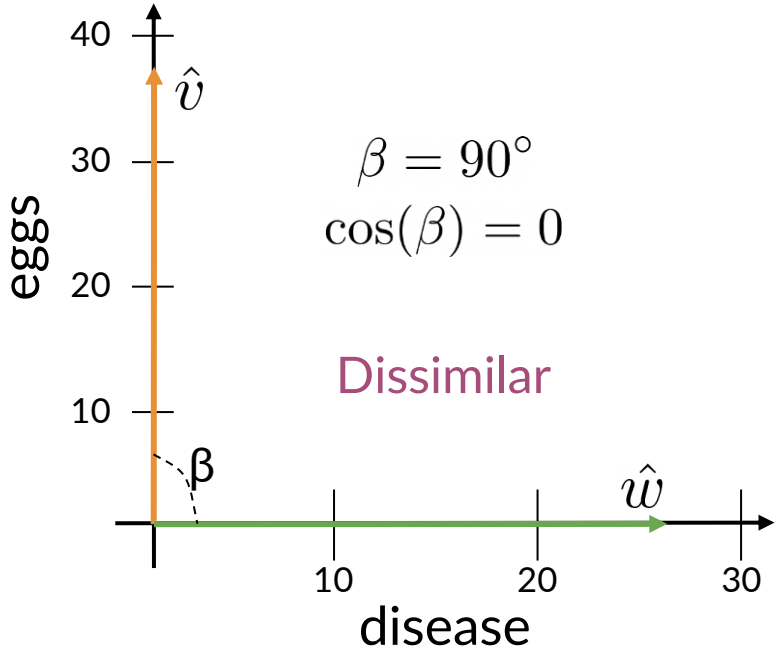
$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

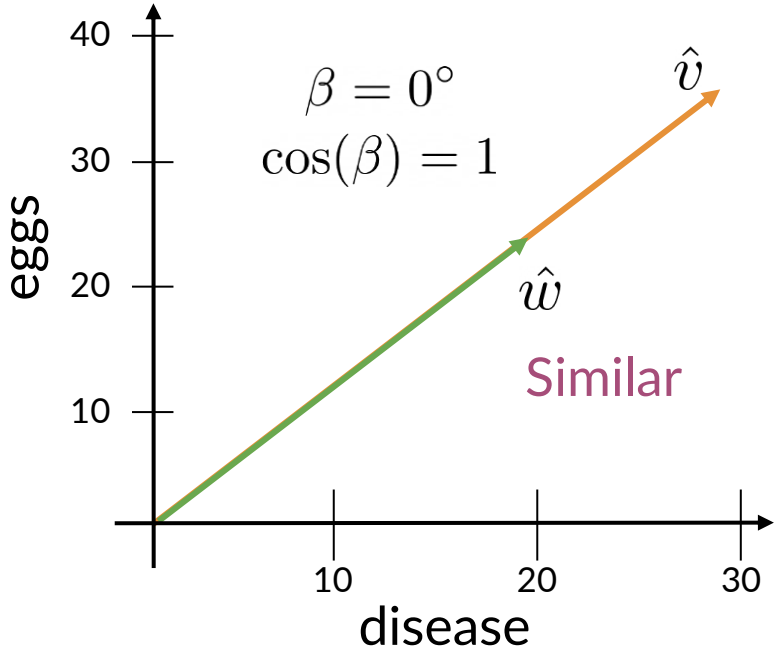
$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

$$= \frac{(20 \times 30) + (40 \times 20)}{\sqrt{20^2 + 40^2} \times \sqrt{30^2 + 20^2}}$$

$$= 0.87$$

Cosine Similarity





Summary

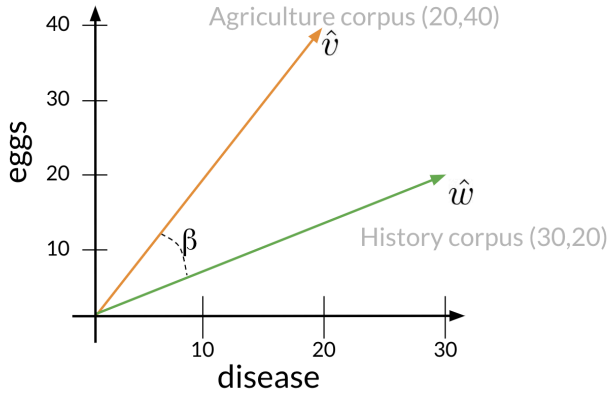
- Cosine $\propto$ Similarity (proportional)
- Cosine Similarity gives values between 0 and 1

Before getting into the cosine similarity function remember that the **norm** of a vector is defined as:

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^{n} |v_i|^2}$$

The **dot product** is then defined as:

$$\vec{v} \cdot \vec{w} = \sum_{i=1}^{n} v_i \cdot w_i$$



$$\hat{v} \cdot \hat{w} = \|\hat{v}\| \|\hat{w}\| \cos(\beta)$$

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

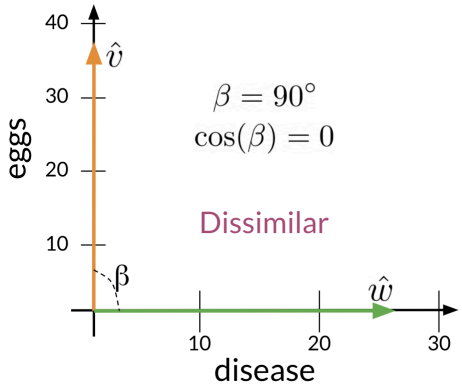
$$= \frac{(20 \times 30) + (40 \times 20)}{\sqrt{20^2 + 40^2} \times \sqrt{30^2 + 20^2}}$$

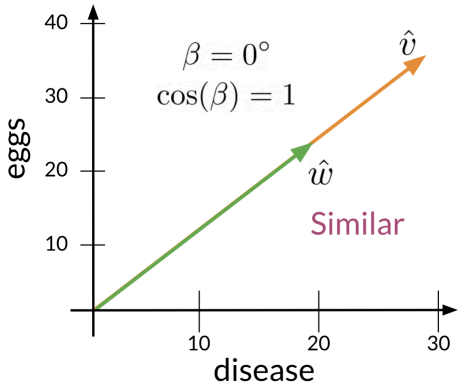
$$= 0.87$$

The following cosine similarity equation makes sense:

$$\cos(\beta) = \frac{\hat{v} \cdot \hat{w}}{\|\hat{v}\| \|\hat{w}\|}$$

If $\hat{v}$ and $\hat{w}$ are the same then you get the numerator to be equal to the denominator. Hence $\beta = 0$. On the other hand, the dot product of two orthogonal (perpendicular) vectors is 0. That takes place when $\beta = 90$.





deeplearning.ai

Manipulating Words in Vector Spaces

Outline

- How to use vector representations

Manipulating word vectors

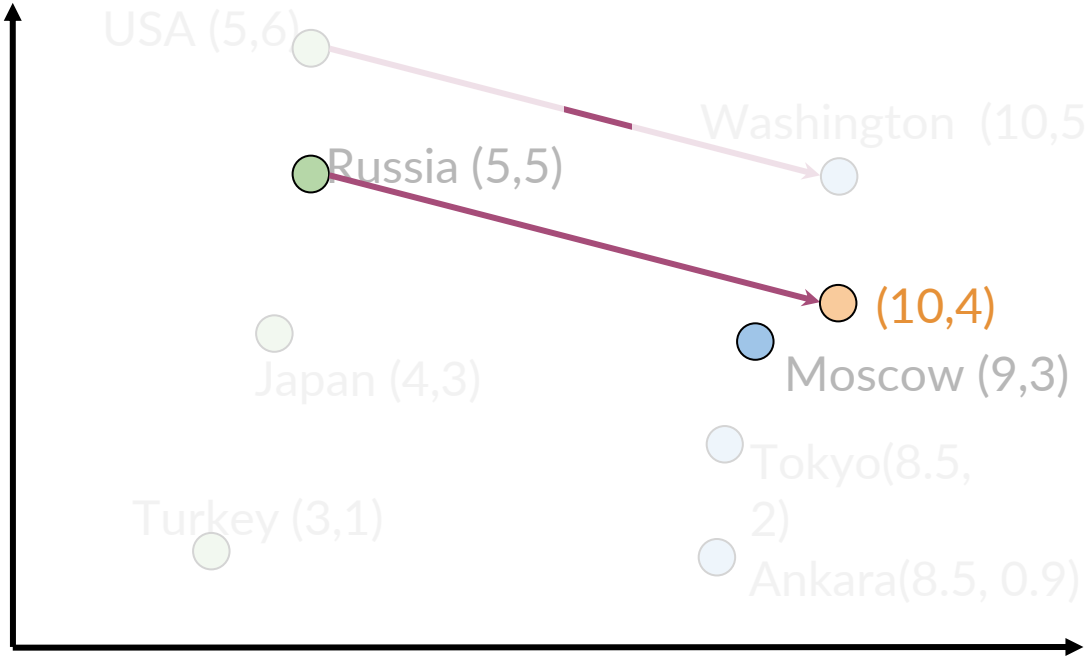
USA

Washington DC

Russia

?

Manipulating word vectors



Washington - USA = $\begin{bmatrix} 5 & -1 \end{bmatrix}$

Russia + $\begin{bmatrix} 5 & -1 \end{bmatrix}$ = $\begin{bmatrix} 10 & 4 \end{bmatrix}$

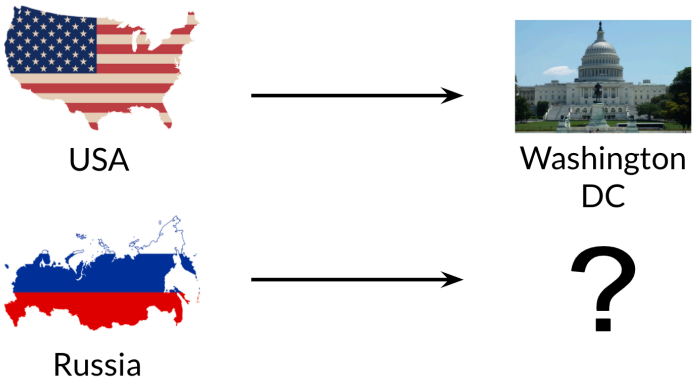
Moscow

[Mikolov et al, 2013, Distributed Representations of Words and Phrases and their Compositionality]

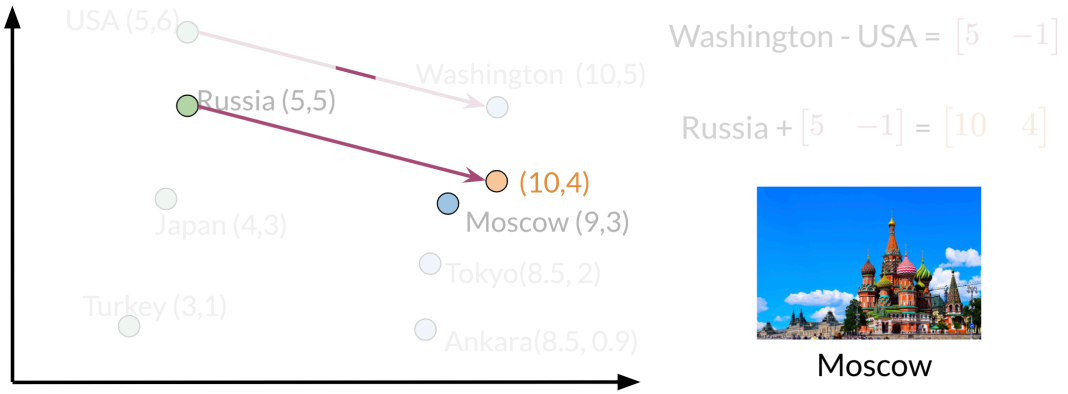
Summary

- Use known relationships to make predictions

You can use word vectors to actually extract patterns and identify certain structures in your text. For example:



You can use the word vector for Russia, USA, and DC to actually compute a **vector** that would be very similar to that of Moscow. You can then use cosine similarity of the **vector** with all the other word vectors you have and you can see that the vector of Moscow is the closest. Isn't that cool?



Note that the distance (and direction) between a country and its capital is relatively the same. Hence manipulating word vectors allows you identify patterns in the text.

deeplearning.ai

Visualization and PCA

Outline

- Some motivation for visualization
- Principal Component Analysis

Visualization of word vectors

$d > 2$

oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

How can you visualize if your representation captures these relationships?

oil & gas

town & city

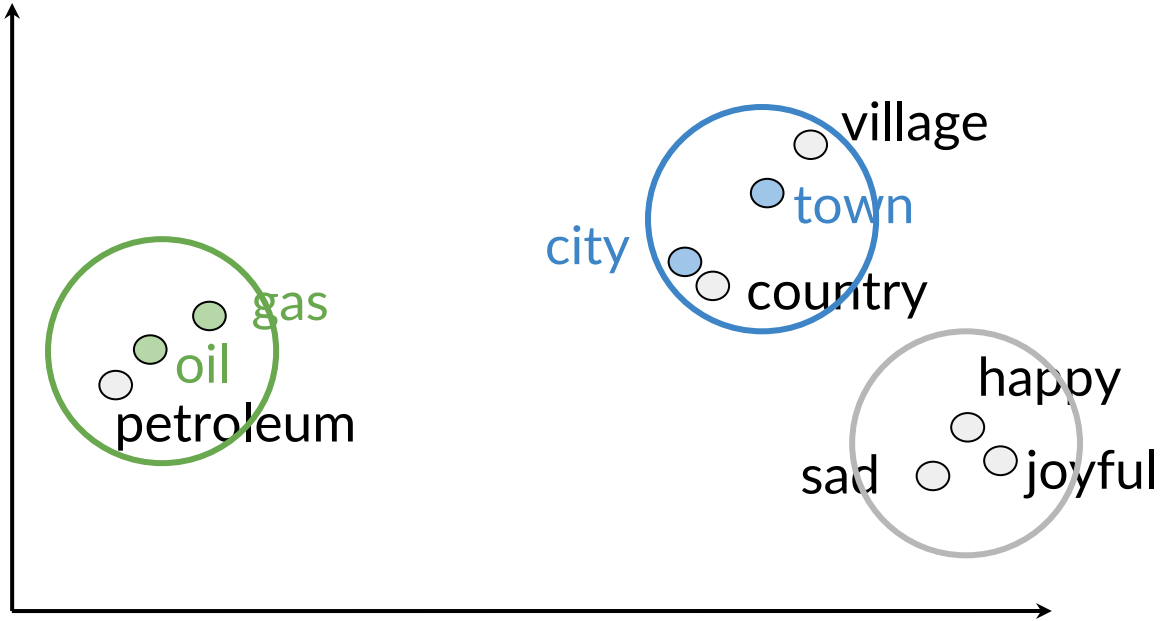
Visualization of word vectors

	$d > 2$		
oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

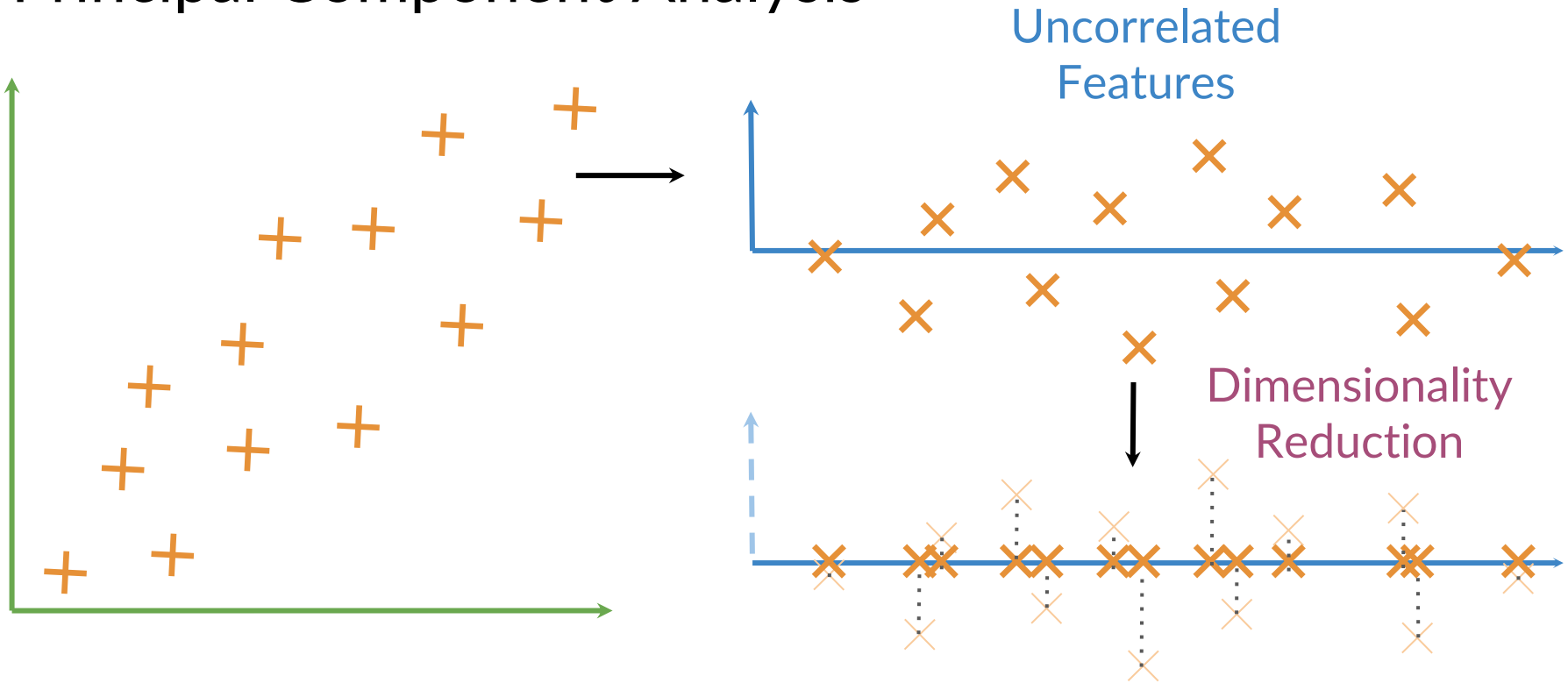
PCA →

	$d = 2$	
oil	2.30	21.2
gas	1.56	19.3
city	13.4	34.1
town	15.6	29.8

Visualization of word vectors



Principal Component Analysis



Summary

- Original Space → Uncorrelated features → Dimension reduction
- Visualization to see words relationships in the vector space

Principal component analysis is an unsupervised learning algorithm which can be used to reduce the dimension of your data. As a result, it allows you to visualize your data. It tries to combine variances across features. Here is a concrete example of PCA:

Visualization of word vectors

$d > 2$

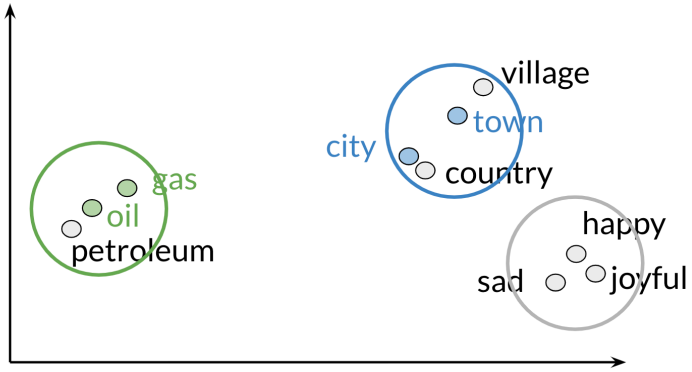
oil	0.20	...	0.10
gas	2.10	...	3.40
city	9.30	...	52.1
town	6.20	...	34.3

How can you visualize if your representation captures these relationships?

oil & gas

town & city

Note that when doing PCA on this data, you will see that oil & gas are close to one another and town & city are also close to one another. To plot the data you can use PCA to go from $d > 2$ dimensions to $d = 2$.



Those are the results of plotting a couple of vectors in two dimensions. Note that words with similar part of speech (POS) tags are next to one another. This is because many of the training algorithms learn words by identifying the neighboring words. Thus, words with similar POS tags tend to be found in similar locations. An interesting insight is that synonyms and antonyms tend to be found next to each other in the plot. Why is that the case?

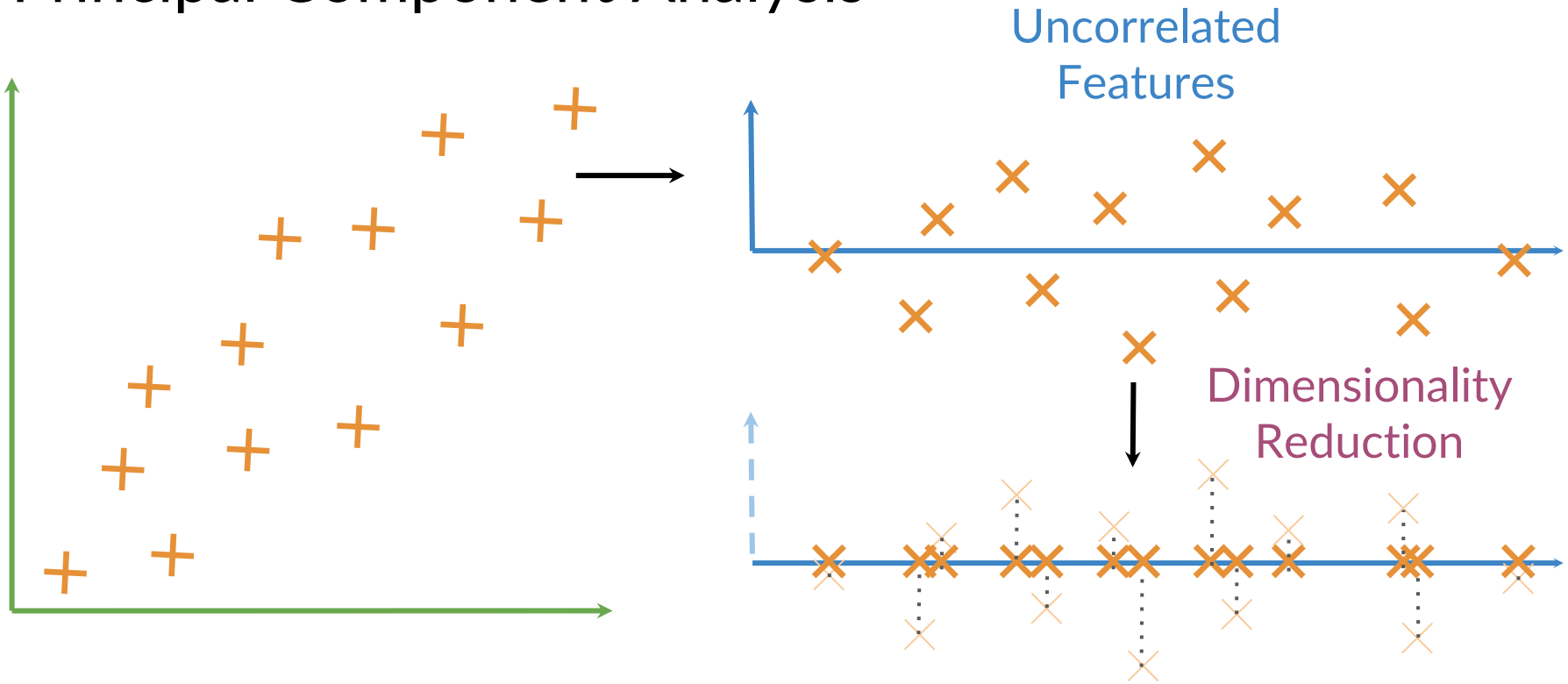
deeplearning.ai

PCA Algorithm

Outline

- How to get uncorrelated features
- How to reduce dimensions while retaining as much information as possible

Principal Component Analysis

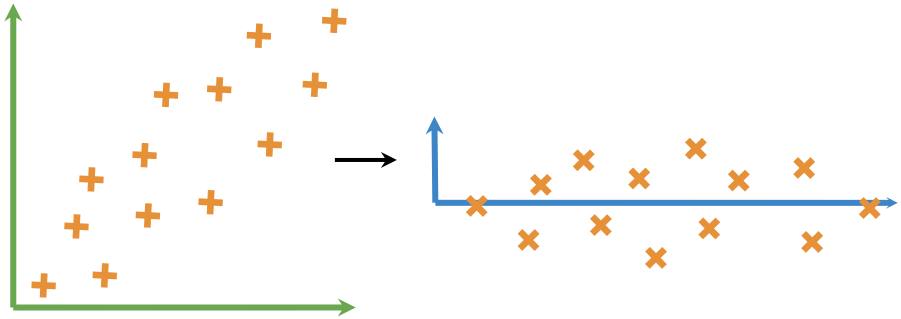


PCA algorithm

Eigenvector: Uncorrelated features for your data

Eigenvalue: the amount of information retained by each feature

PCA algorithm



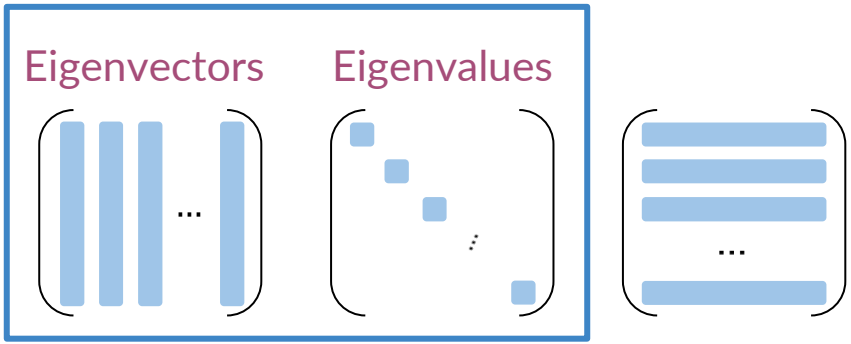
Mean Normalize Data $x_i = \frac{x_i - \mu_{x_i}}{\sigma_{x_i}}$

↓

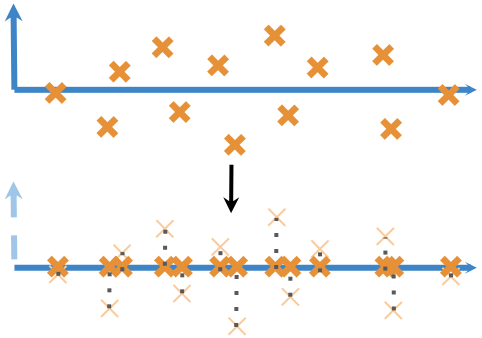
Get Covariance Matrix Σ

↓

Perform SVD $\mathrm{SVD}(\Sigma)$



PCA algorithm



Eigenvectors Eigenvalues

$$U \quad S$$

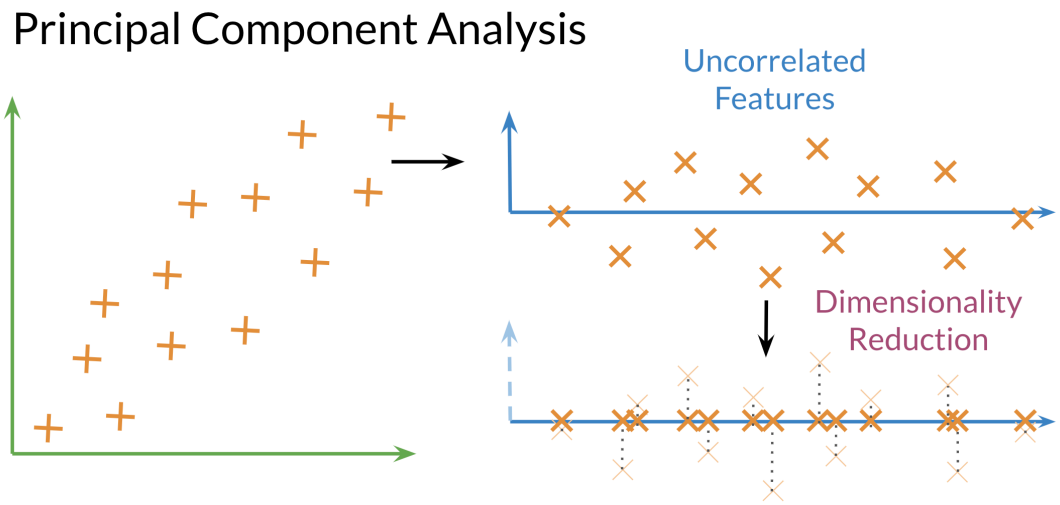
Dot Product to Project Data $X' = XU[:, 0:2]$

Percentage of Retained Variance $\frac{\sum_{i=0}^{1} S_{ii}}{\sum_{j=0}^{d} S_{jj}}$

Summary

- Eigenvectors give the direction of uncorrelated features
- Eigenvalues are the variance of the new features
- Dot product gives the projection on uncorrelated features

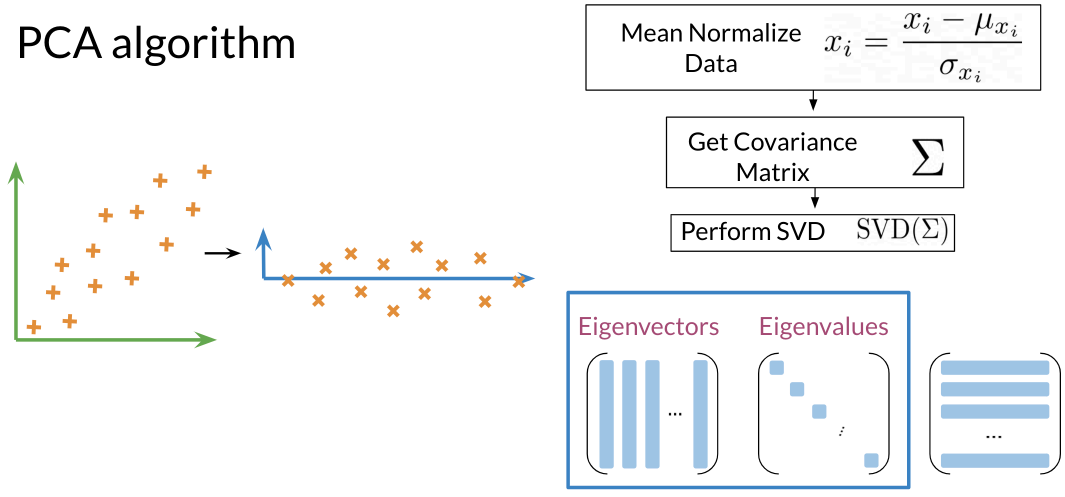
PCA is commonly used to reduce the dimension of your data. Intuitively the model collapses the data across principal components. You can think of the first principal component (in a 2D dataset) as the line where there is the most amount of variance. You can then collapse the data points on that line. Hence you went from 2D to 1D. You can generalize this intuition to several dimensions.



Eigenvector : the resulting vectors, also known as the uncorrelated features of your data

Eigenvalue: the amount of information retained by each new feature. You can think of it as the variance in the eigenvector.

Also each **eigenvalue** has a corresponding eigenvector. The eigenvalue tells you how much variance there is in the eigenvector. Here are the steps required to compute PCA:



Steps to Compute PCA:

- Mean normalize your data

- Compute the covariance matrix
- Compute SVD on your covariance matrix. This returns $[USV] = svd(\Sigma)$. The three matrices U, S, V are drawn above. U is labelled with eigenvectors, and S is labelled with eigenvalues.
- You can then use the first n columns of vector U, to get your new data by multiplying $XU[:, 0:n]$.

Counterclockwise Rotation

If you want to rotate a vector r with coordinates (x, y) and angle α counterclockwise over an angle β to get vector r' with coordinates (x', y') then the following holds:

$$x = r * cos(\alpha)$$

$$y = r * sin(\alpha)$$

$$x' = r' * cos(\alpha + \beta)$$

$$y' = r' * sin(\alpha + \beta)$$

Trigonometric addition gives us:

$$cos(\alpha + \beta) = cos(\alpha)cos(\beta) - sin(\alpha)sin(\beta)$$

$$sin(\alpha + \beta) = cos(\alpha)sin(\beta) + sin(\alpha)cos(\beta)$$

For proof, see this Wikipedia page section.

As the length of the vector stays the same,

$$x' = r * cos(\alpha)cos(\beta) - r * sin(\alpha)sin(\beta)$$

$$y' = r * cos(\alpha)sin(\beta) + r * sin(\alpha)cos(\beta)$$

This equates to:

$$x' = x * cos(\beta) - y * sin(\beta)$$

$$y' = x * sin(\beta) + y * cos(\beta)$$

Written as matrix multiplication with row vectors, this becomes,

$$[x', y'] = [x, y] \cdot \begin{bmatrix} cos(\beta) & sin(\beta) \\ -sin(\beta) & cos(\beta) \end{bmatrix}$$

with the rotation matrix equal to,

$$R = \begin{bmatrix} cos(\beta) & sin(\beta) \\ -sin(\beta) & cos(\beta) \end{bmatrix}$$

Written as matrix multiplication with column vectors, this becomes,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\beta) & -sin(\beta) \\ sin(\beta) & cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

with the rotation matrix equal to,

$$R = \begin{bmatrix} cos(\beta) & -sin(\beta) \\ sin(\beta) & cos(\beta) \end{bmatrix}$$

Note that the position of $-sin(\beta)$ in the rotation matrix has changed.

Clockwise Rotation

If rotation is clockwise, then the rotation matrix for multiplication with row vectors becomes,

$$R = \begin{bmatrix} cos(-\beta) & sin(-\beta) \\ -sin(-\beta) & cos(-\beta) \end{bmatrix}$$

As $sin(-\beta) = -sin(\beta)$ and $cos(-\beta) = cos(\beta)$

this equates to

$$R = \begin{bmatrix} cos(\beta) & -sin(\beta) \\ sin(\beta) & cos(\beta) \end{bmatrix}$$

So clockwise rotation of a vector $[x, y]$ can be expressed as,

$$[x', y'] = [x, y] \cdot \begin{bmatrix} cos(\beta) & -sin(\beta) \\ sin(\beta) & cos(\beta) \end{bmatrix}$$

The rotation matrix for multiplication with column vectors becomes,

$$R = \begin{bmatrix} cos(-\beta) & -sin(-\beta) \\ sin(-\beta) & cos(-\beta) \end{bmatrix}$$

which equates to,

$$R = \begin{bmatrix} cos(\beta) & sin(\beta) \\ -sin(\beta) & cos(\beta) \end{bmatrix}$$

So clockwise rotation of a vector $\begin{bmatrix} x \\ y \end{bmatrix}$ can be expressed as,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\beta) & sin(\beta) \\ -sin(\beta) & cos(\beta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Author: Reinoud Bosch

deeplearning.ai

Overview

What you'll be able to do!

machine translation

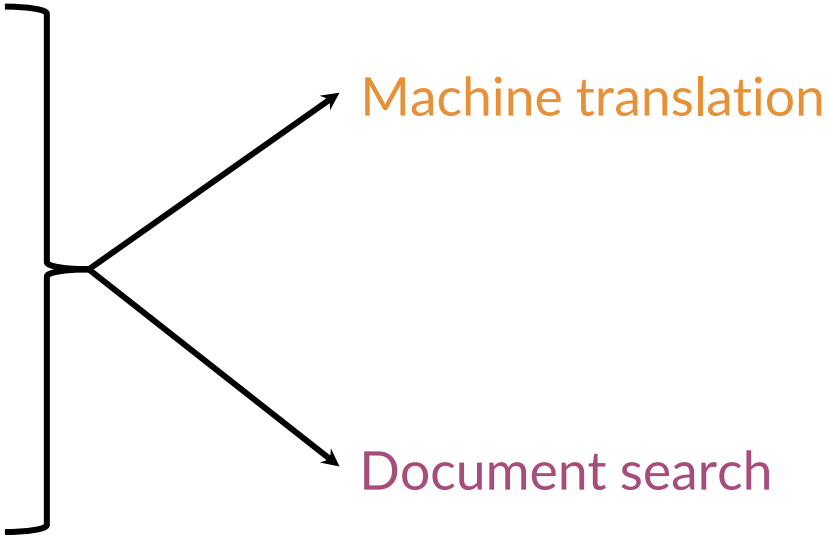
"hello!" → "bonjour!"

document search

"Can I get a refund?" → "What's your return policy?" … "May I get my money back?"

Learning Objectives

- Transform vector
- “K nearest neighbors”
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors



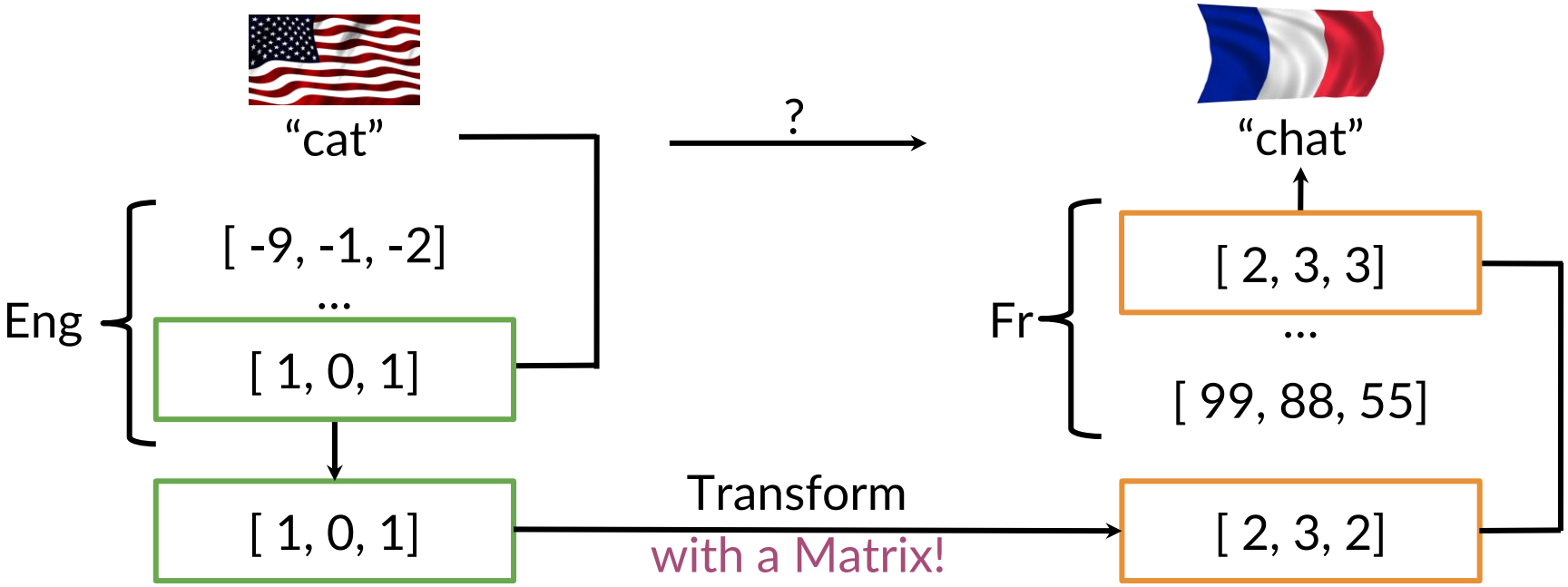
deeplearning.ai

Transforming word vectors

Outline

- Translation = Transformation
- How to get a good transformation

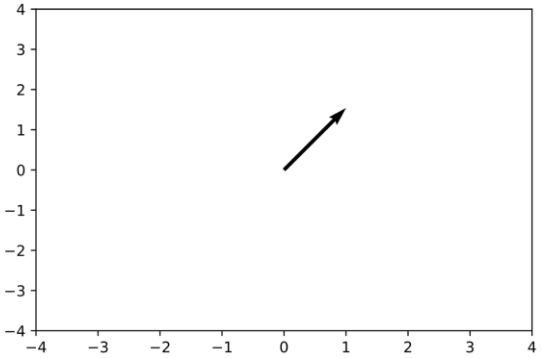
Overview of Translation

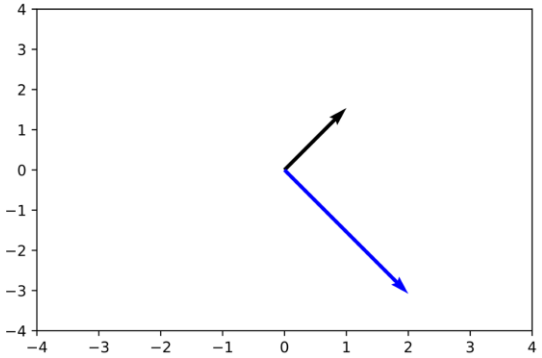


Transforming vectors

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} = \begin{pmatrix} 2 & -2 \end{pmatrix}$$

X R Y





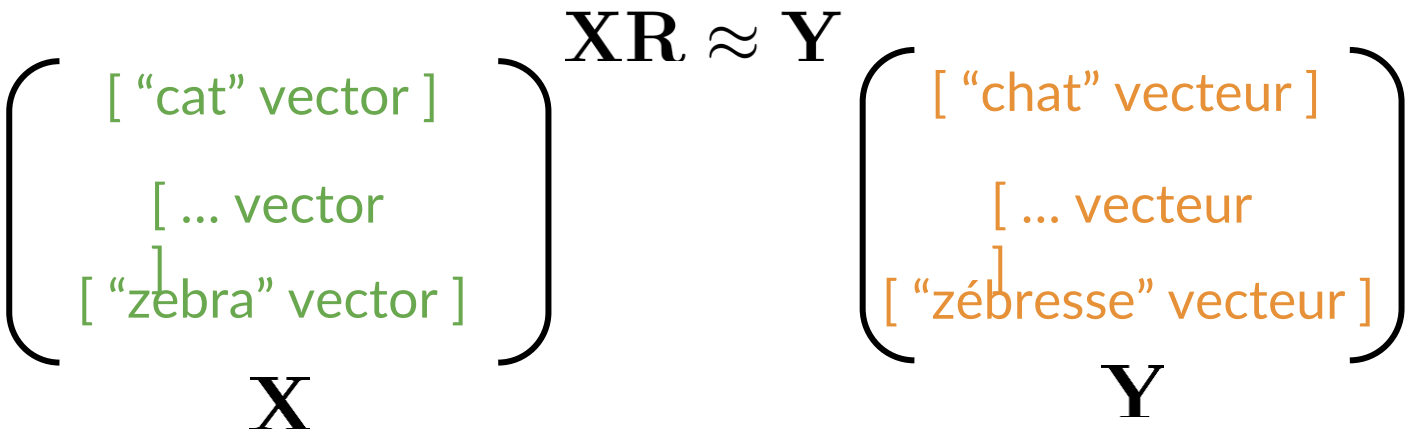
Transforming vectors

```
R = np.array([[2,0],
              [0,-2]])

x = np.array([[1,1]])

np.dot(x,R)
```

```
array([[2,-2]])
```

Try it yourself!

Align word vectors



subsets of the full vocabulary

Solving for R

initialize R

in a loop:

$$Loss = \| \mathbf{XR} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss$$ gradient

$$R = R - \alpha g$$ update

Frobenius norm

$$\| \mathbf{XR} - \mathbf{Y} \|_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Frobenius norm

```
A = np.array([[2,2],

- --
A_squared = np.square(A)
A_squared
```

```
array([[4,4],
```

Try it yourself!

```
A_Frobenious = np.sqrt(np.sum(A_squared))
A_Frobenious
```

```
4.0
```

Frobenius norm squared

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

$$\|\mathbf{A}\|_F^2 = 16$$

Gradient

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR} Loss = \frac{2}{m} \left(\mathbf{X}^T(\mathbf{XR} - \mathbf{Y})\right)$$

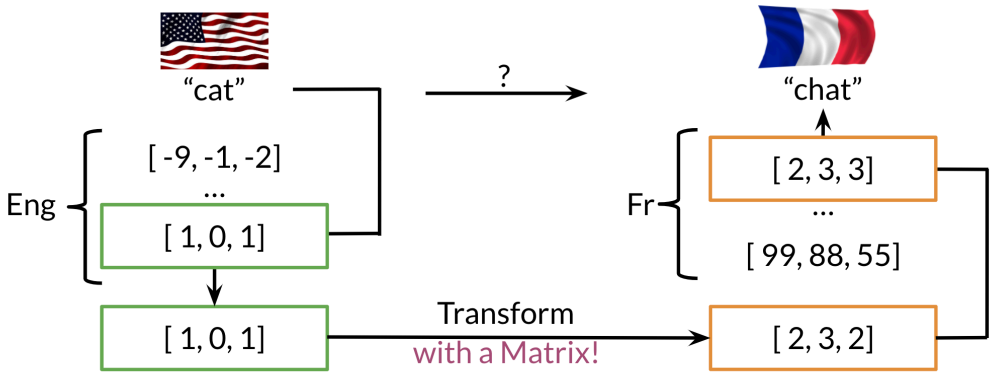
number of rows (or words in the subset)

Implement in the assignment!

Summary

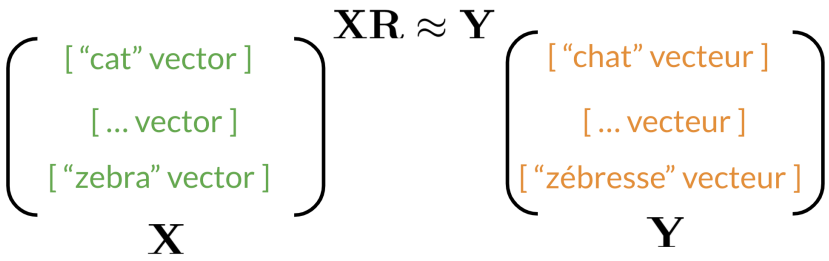
- $\mathbf{XR} \approx \mathbf{Y}$
- minimize $\|\mathbf{XR} - \mathbf{Y}\|_F^2$

In the previous week, I showed you how we can plot word vectors. Now, you will see how you can take a word vector and learn a mapping that will allow you to translate words by learning a "transformation matrix". Here is a visualization:



Note that the word "chat" in french means cat. You can learn that by taking the vector corresponding to "cat" in english, multiplying it by a matrix that you learn and then you can use cosine similarity between the output and all the french vectors. You should see that the closest result is the vector which corresponds to "chat".

Here is a visualization of that showing you the aligned vectors:



subsets of the full vocabulary

Note that X corresponds to the matrix of english word vectors and Y corresponds to the matrix of french word vectors. R is the mapping matrix.

Steps required to learn *R*:

- Initialize R
- For loop

$$Loss = \|XR - Y\|_F$$

$$g = \frac{d}{dR} Loss$$

$$R = R - \alpha * g$$

Here is an example to show you how the frobenius norm works.

$\|\mathbf{XR} - \mathbf{Y}\|_F$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \left| a_{ij} \right|^2}$$

In summary you are making use of the following:

· XR ≈ Y

· minimize $\|XR - Y\|_F^2$

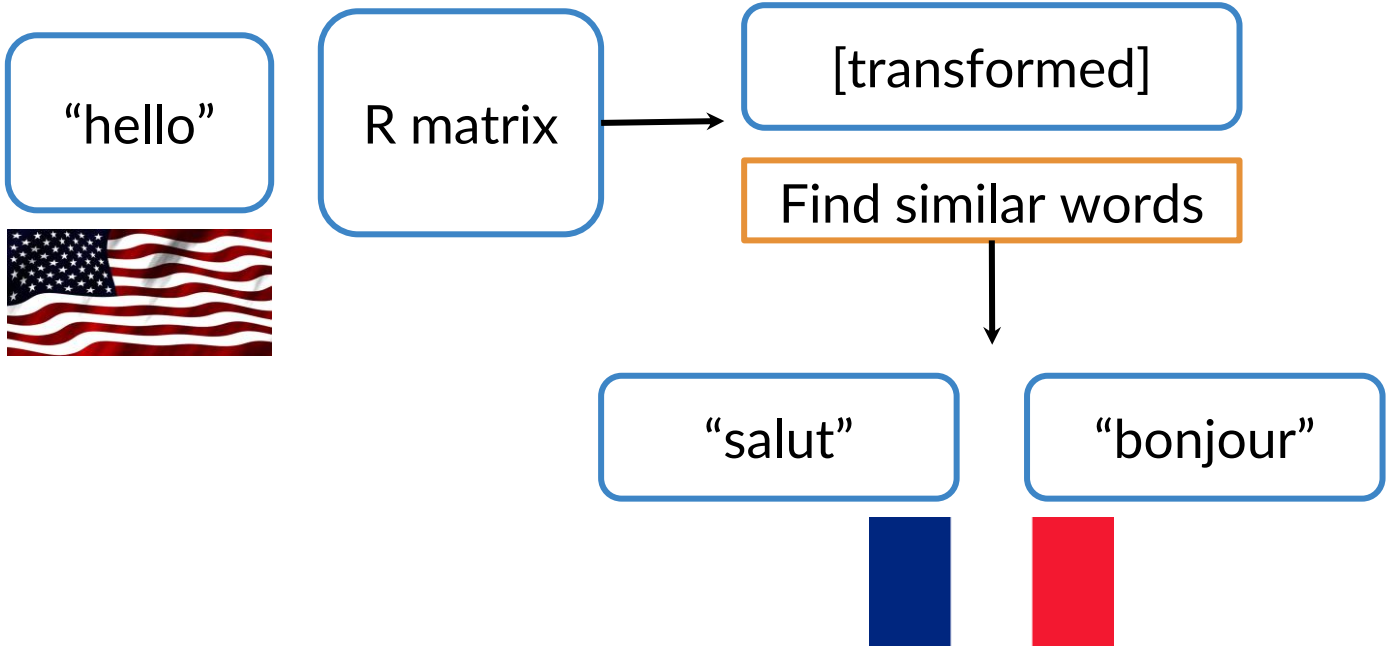
deeplearning.ai

K-nearest neighbors

Outline

- K closest matches ⟶ K-nearest neighbors

Finding the translation



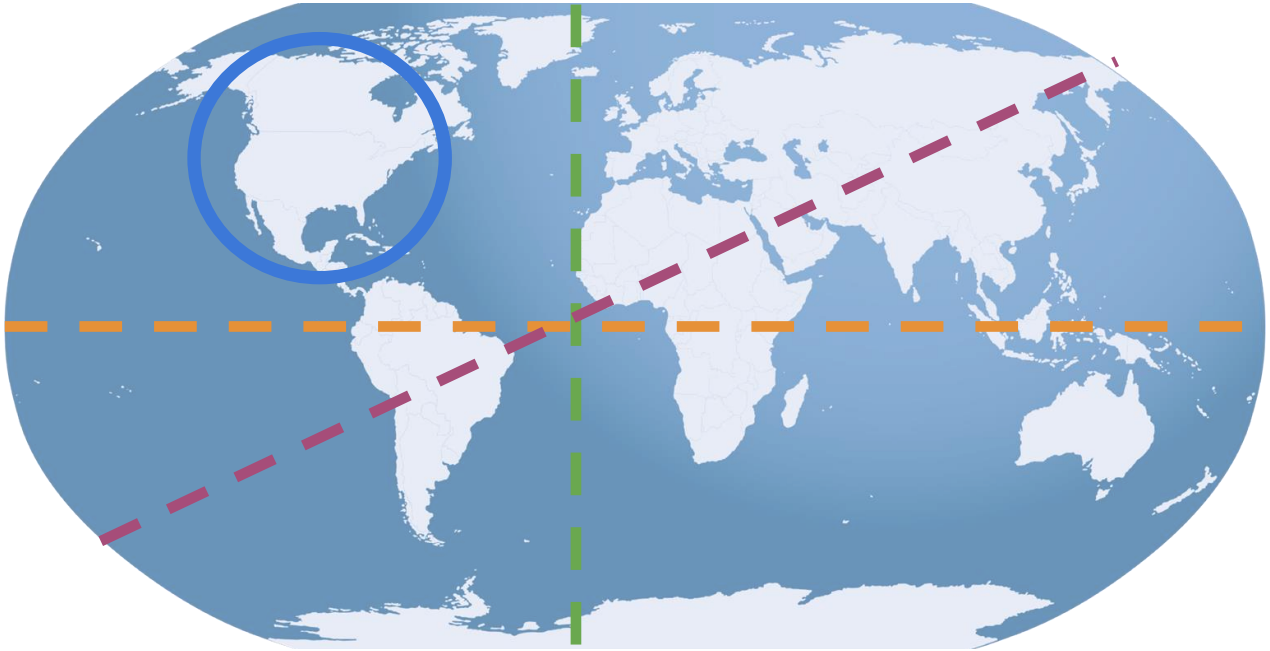
Nearest neighbours

You

San Francisco

Friend	Location	Nearest
	Shanghai	2
	Bangalore	3
	Los Angeles	1

Nearest neighbors

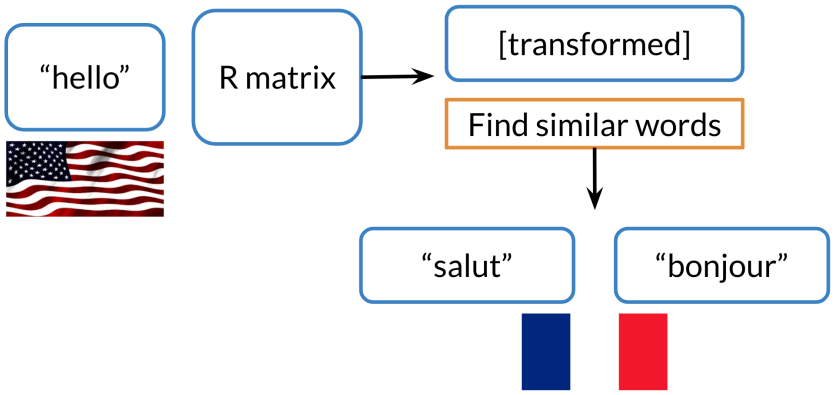


Hash tables!

Summary

- K-nearest neighbors, for closest matches
- Hash tables

After you have computed the output of XR you get a vector. You then need to find the most similar vectors to your output. Here is a visual example:



In the video, we mentioned if you were in San Francisco, and you had friends all over the world, you would want to find the nearest neighbors. To do that it might be expensive to go over all the countries one at a time. So we will introduce hashing to show you how you can do a look up much faster.

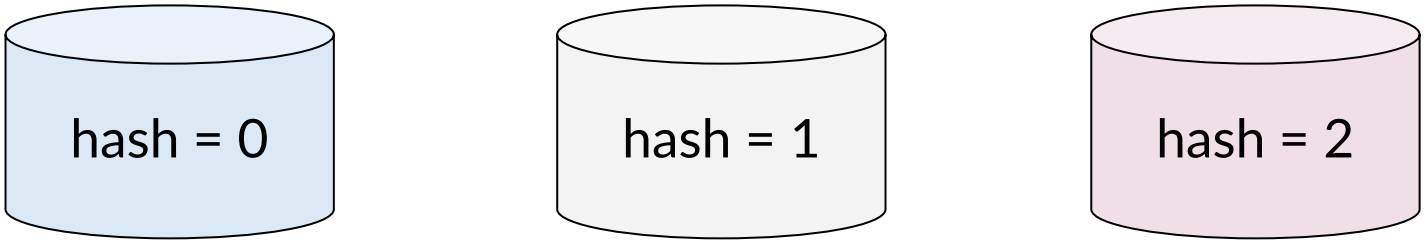
deeplearning.ai

Hash tables and hash functions

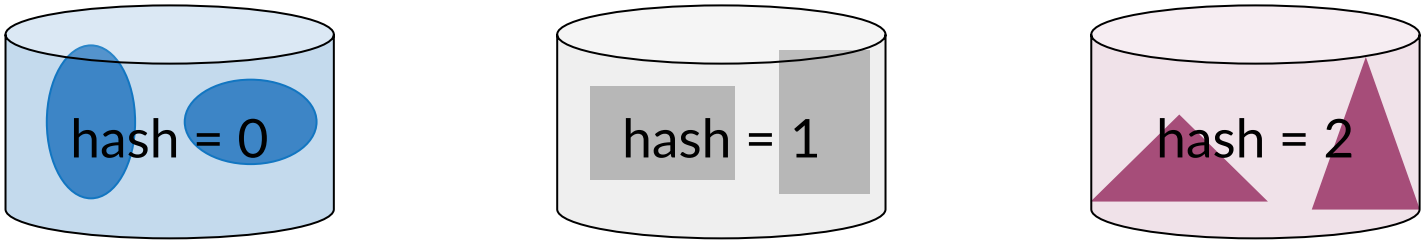
Outline

- Hash values
- Hash functions

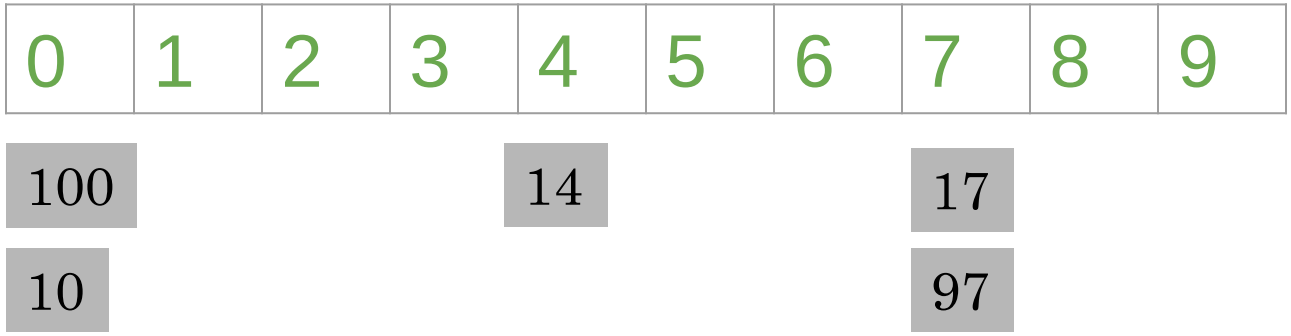
Hash tables



Hash tables



Hash function



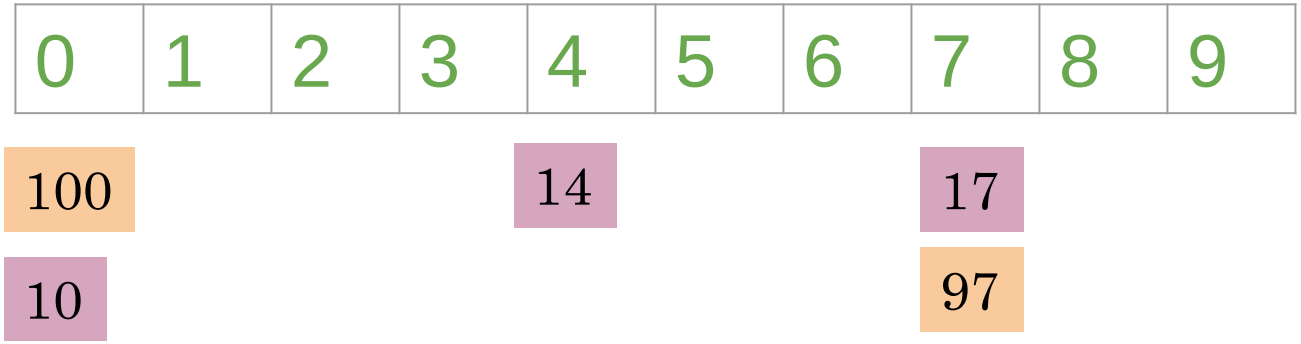
Hash function (vector) ⟶ Hash value

Hash value = vector % number of buckets

Create a basic hash table

```
def basic_hash_table(value_l,n_buckets):

    def hash_function(value_l,n_buckets):
            return int(value_l) % n_buckets

    hash_table = {i:[] for i in range(n_buckets)}
    for value in value_l:
            hash_value =
hash_function(value,n_buckets)

    return hash_table
```

Hash function



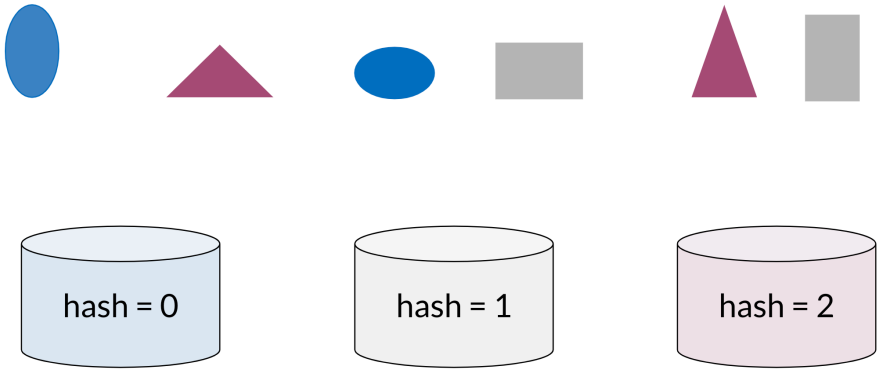
Hash function by location?

Locality sensitive hashing, next!

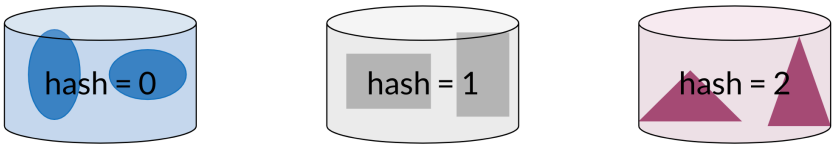
Summary

Hash function (vector) ⟶ Hash value

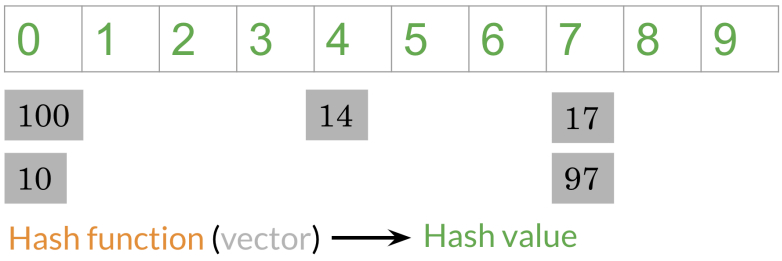
Imagine you had to cluster the following figures into different buckets:



Note that the figures blue, red, and gray ones would each be clustered with each other



You can think of hash function as a function that takes data of arbitrary sizes and maps it to a fixed value. The values returned are known as *hash values* or even *hashes* .



Hash value = vector % number of buckets

The diagram above shows a concrete example of a hash function which takes a vector and returns a value. Then you can mod that value by the number of buckets and put that number in its corresponding bucket. For example, 14 is in the 4th bucker, 17 & 97 are in the 7th bucket. Let's take a look at how you can do it using some code.

```
def basic_hash_table(value_l,n_buckets):

    def hash_function(value_l,n_buckets):
        return int(value_l) % n_buckets

    hash_table = {i:[] for i in range(n_buckets)}
    for value in value_l:
        hash_value = hash_function(value,n_buckets)
        hash_table[hash_value].append(value)

    return hash_table
```

The code snippet above creates a basic hash table which consists of hashed values inside their buckets. **hash_function** takes in *value_l* (a list of values to be hashed) and *n_buckets* and mods the value by the buckets. Now to create the *hash_table,* you first initialize a list to be of dimension *n_buckets (* each value will go to a bucket) . For each value in your list of values, you will feed it into your **hash_function,** get the *hash_value,* and append it to the list of values in the corresponding bucket.

Now given an input, you don't have to compare it to all the other examples, you can just compare it to all the values in the same *hash_bucket* that input has been hashed to.

When hashing you sometimes want similar words or similar numbers to be hashed to the same bucket. To do this, you will use "locality sensitive hashing." Locality is another word for "location". So locality sensitive hashing is a hashing method that cares very deeply about assigning items based on where they're located in vector space.

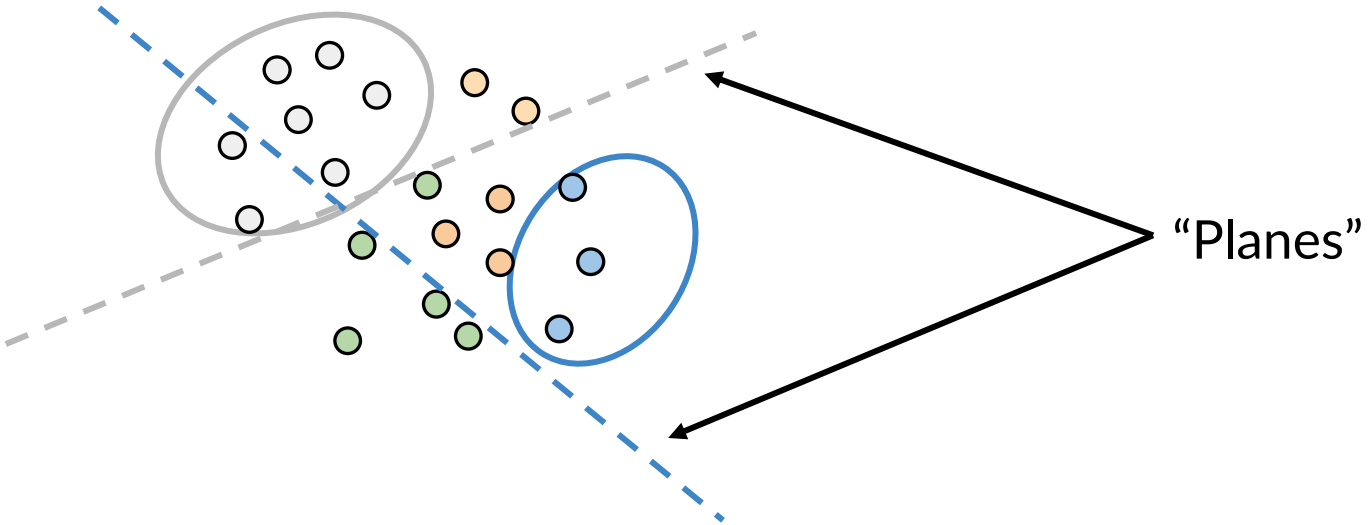
deeplearning.ai

Locality sensitive hashing

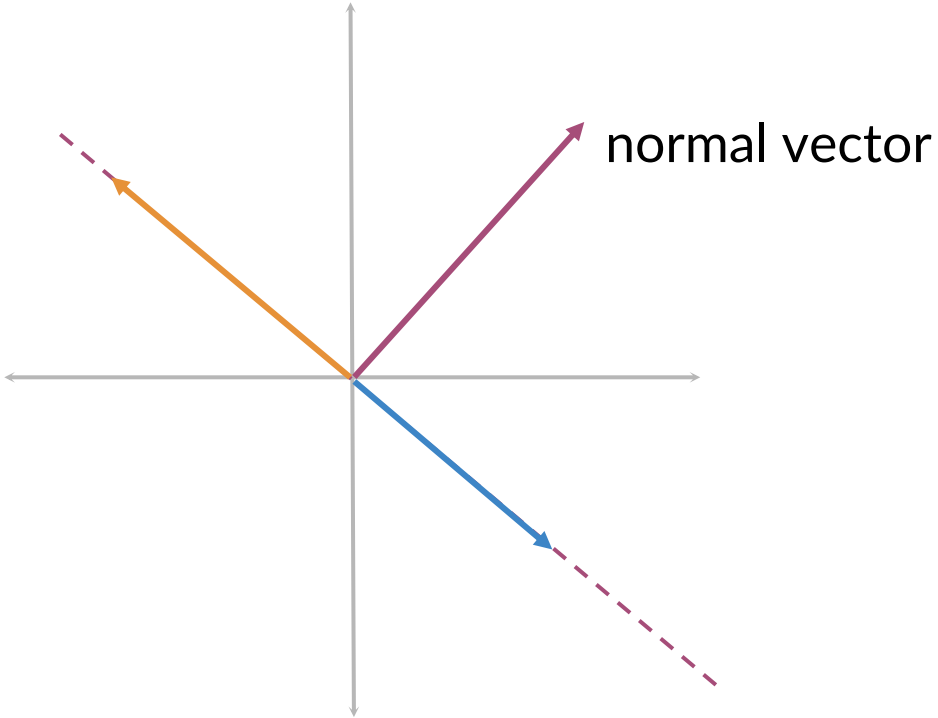
Outline

- Locality sensitive hashing with planes in vector spaces

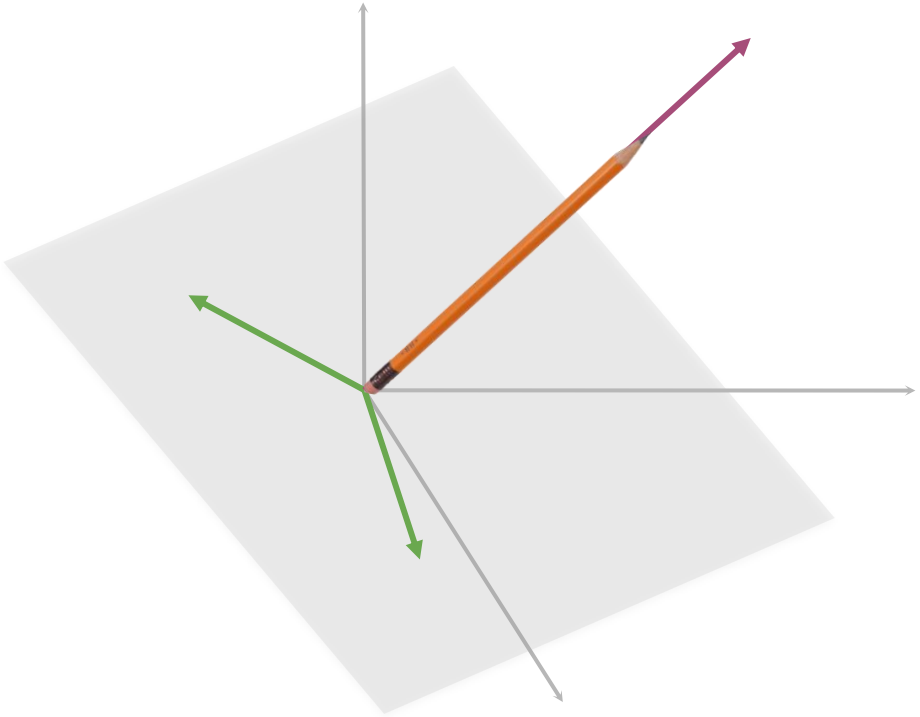
Locality Sensitive Hashing



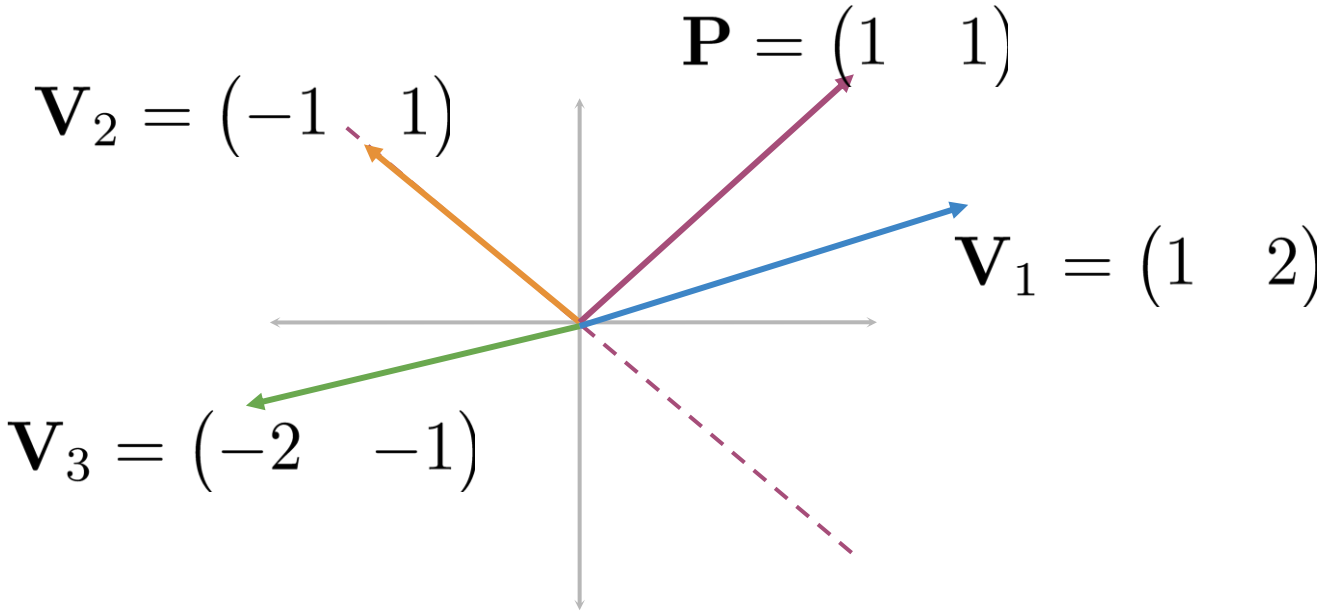
Planes



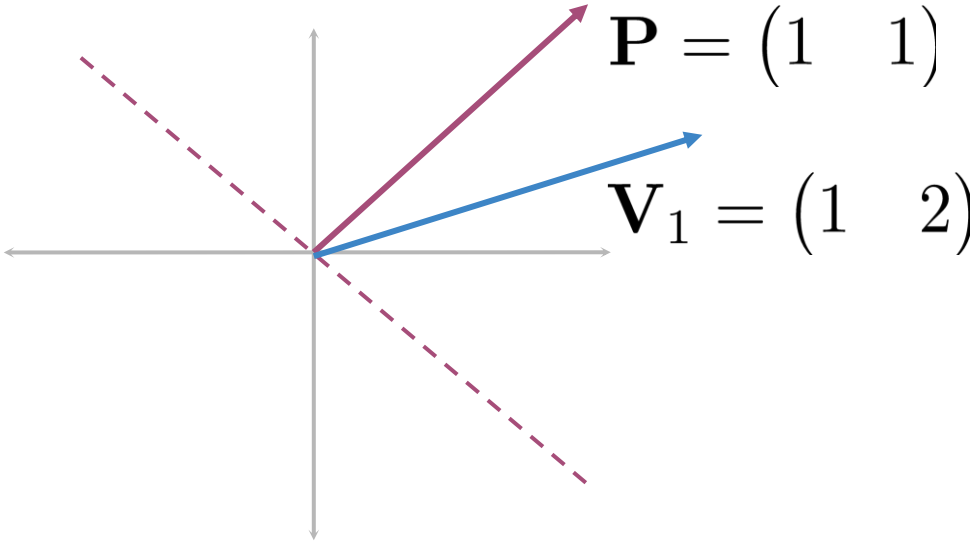
Planes



Which side of the plane?



Which side of the plane?



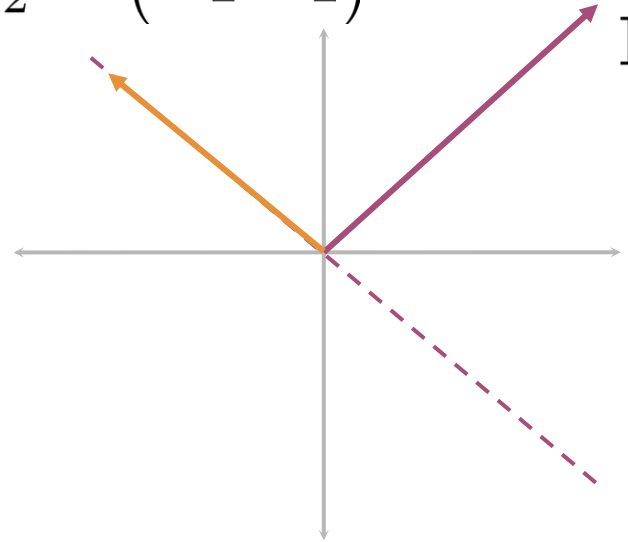
$$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

$$\mathbf{V}_1 = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

$$\mathbf{P}\mathbf{V}_1^T = 3$$

Which side of the plane?

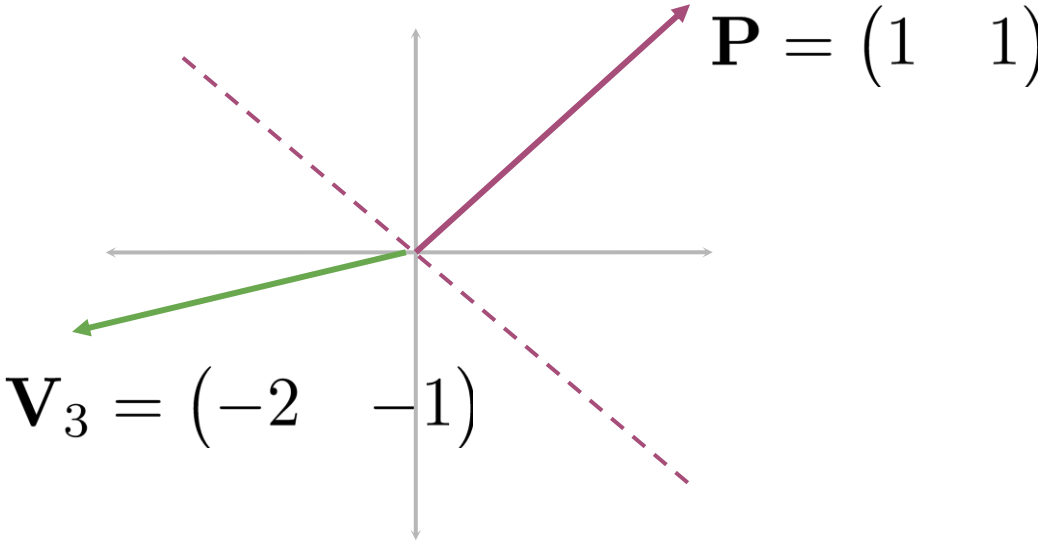
$\mathbf{V}_2 = \begin{pmatrix} -1 & 1 \end{pmatrix}$



$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$

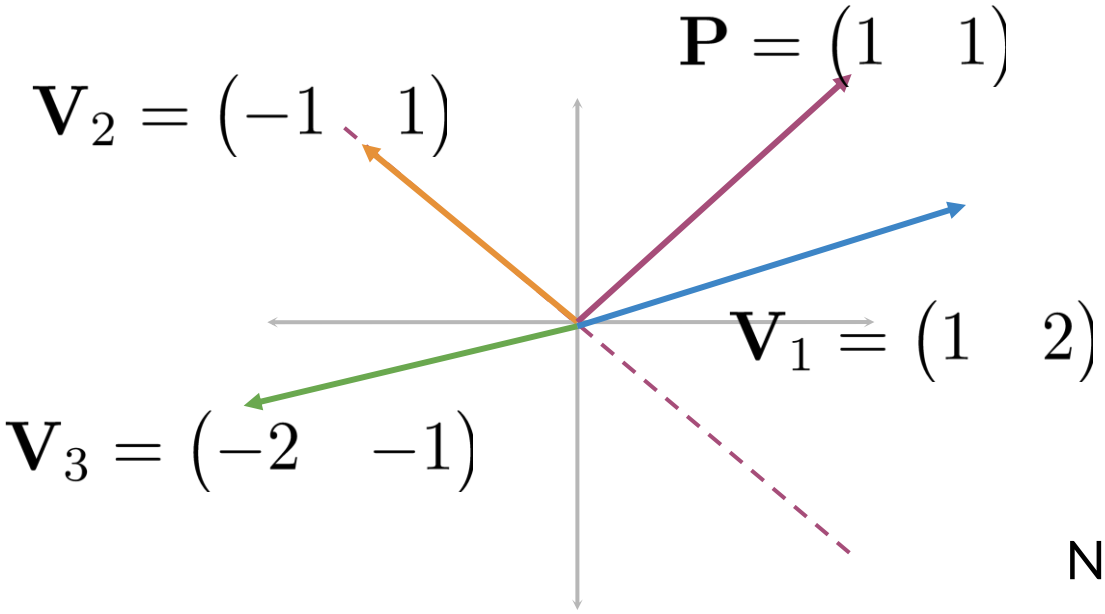
$$\mathbf{P}\mathbf{V}_2^T = 0$$

Which side of the plane?



$$\mathbf{P}\mathbf{V}_3^T = -3$$

Which side of the plane?



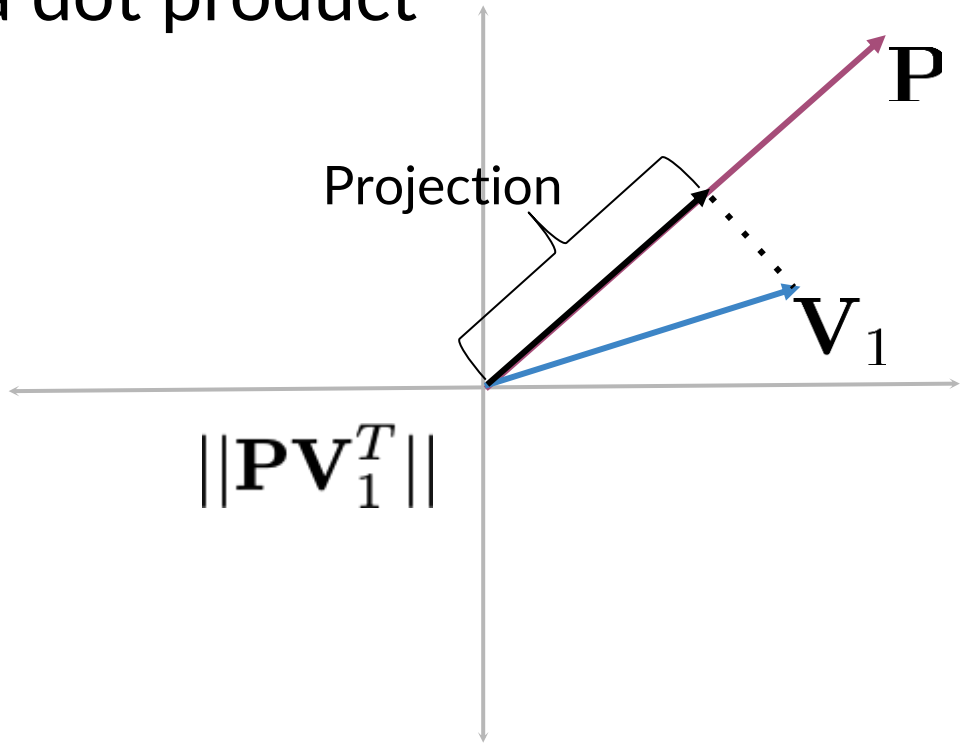
$$\mathbf{P}\mathbf{V}_1^T = 3$$

$$\mathbf{P}\mathbf{V}_2^T = 0$$

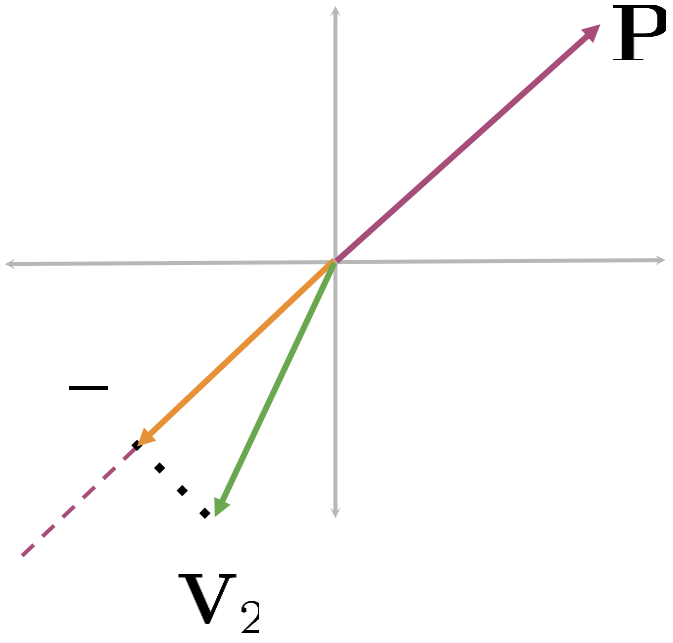
$$\mathbf{P}\mathbf{V}_3^T = -3$$

Notice the signs?

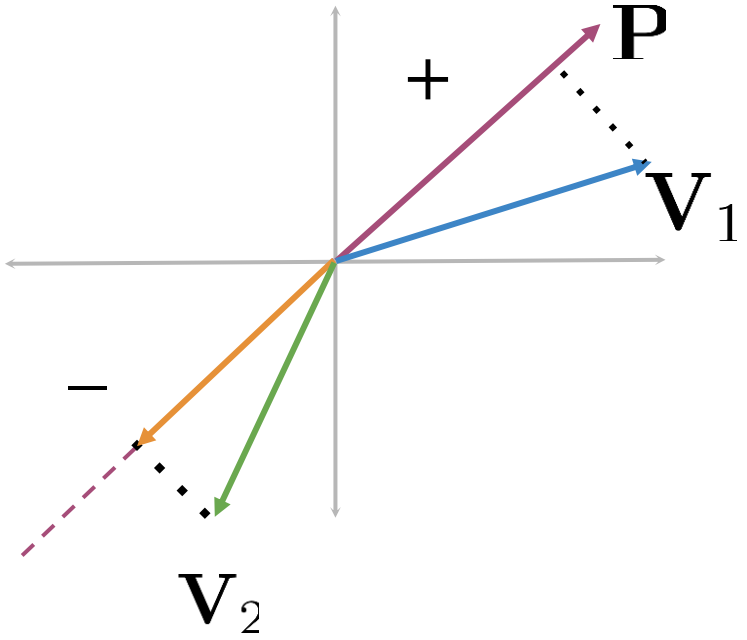
Visualizing a dot product



Visualizing a dot product



Visualizing a dot product



Sign indicates direction

Which side of the plane?

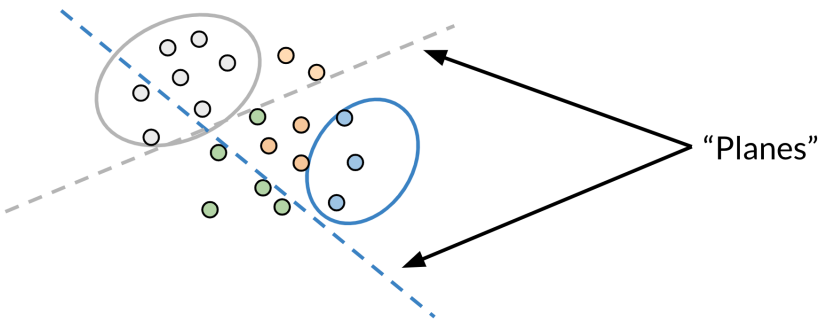
```
def side_of_plane(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    sign_of_dot_product_scalar= np.asscalar(sign_of_dot_product)
    return sign_of_dot_product_scalar
```

Try it!

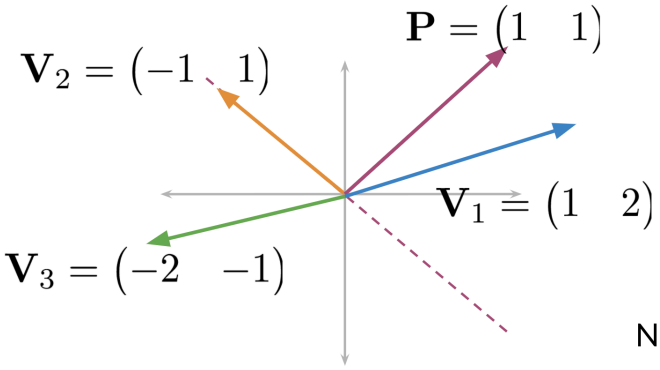
Summary

- Sign of dot product ⟶ Hash values

Locality sensitive hashing is a technique that allows you to hash similar inputs into the same buckets with high probability.



Instead of the typical buckets we have been using, you can think of clustering the points by deciding whether they are above or below the line. Now as we go to higher dimensions (say n-dimensional vectors), you would be using planes instead of lines. Let's look at a concrete example:

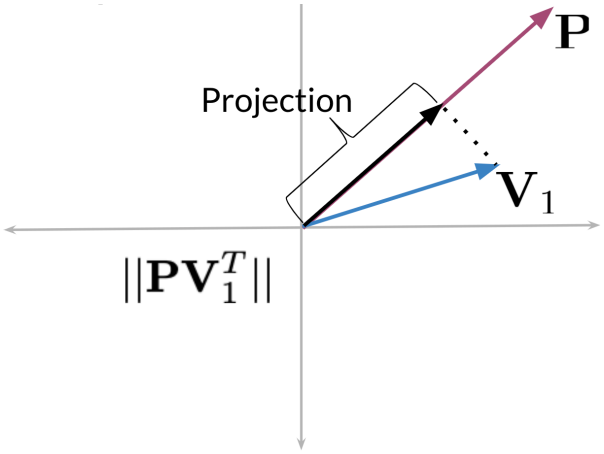


$$\mathbf{P}\mathbf{V}_1^T = 3$$
$$\mathbf{P}\mathbf{V}_2^T = 0$$
$$\mathbf{P}\mathbf{V}_3^T = -3$$

Notice the signs?

Given some point located at (1,1) and three vectors
$V_1 = (1, 2), V_2 = (-1, 1), V_3 = (-2, -1)$ you will see what happens when we take the dot product. First note that the dashed line is our plane. The vector with point $P = (1, 1)$ is perpendicular to that line (plane). Now any vector above the dashed line that is multiplied by $(1, 1)$ would have a positive number. Any vector below the dashed line when dotted with $(1, 1)$ will have a negative number. Any vector on the dashed line multiplied by $(1, 1)$ will give you a dot product of 0.

Here is how to visualize a projection (i.e. a dot product between two vectors):



When you take the dot product of a vector V_1 and a P, then you take the magnitude or length of that vector, you get the black line (labelled as Projection). The sign indicates on which side of the plane the projection vector lies.

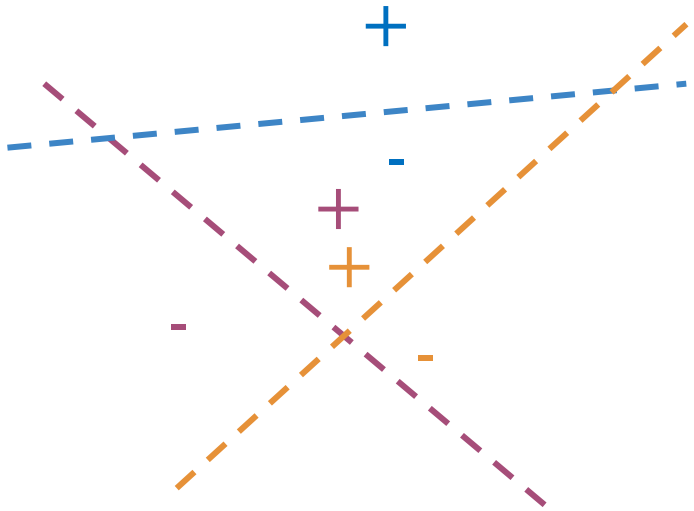
deeplearning.ai

Multiple Planes

Outline

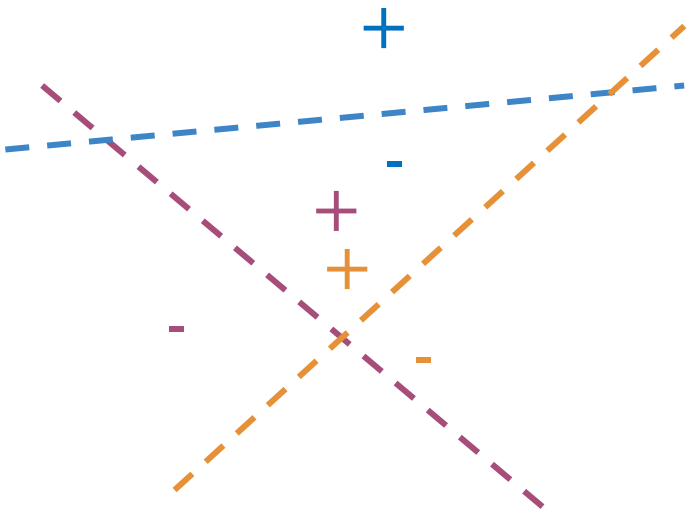
- Multiple planes ⟶ Dot products ⟶ Hash values

Multiple planes



single hash value?

Multiple planes, single hash value?



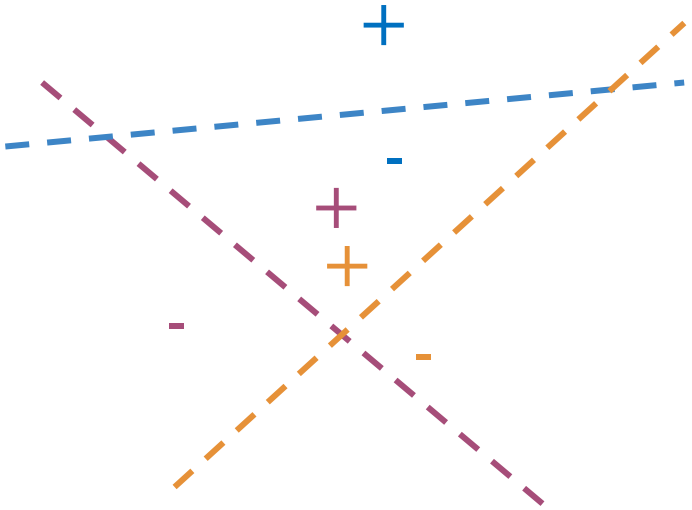
$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

$$\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$
$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$
$$= 3$$

Multiple planes, single hash value!



$$sign_i \geq 0, \rightarrow h_i = 1$$

$$sign_i < 0, \rightarrow h_i = 0$$

$$\text{hash} = \sum_{i}^{H} 2^i \times h_i$$

Multiple planes, single hash value!!

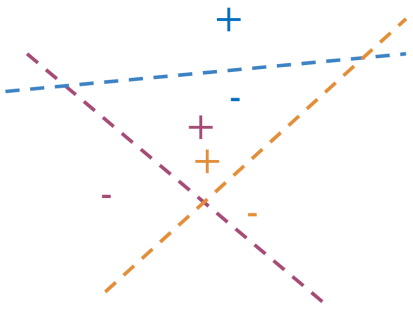
```
def hash_multiple_plane(P_l,v):

    hash_value = 0

    for i, P in enumerate(P_l):
        sign = side_of_plane(P,v)
        hash_i = 1 if sign >=0 else 0
        hash_value += 2**i * hash_i

    return hash_value
```

Try it!

Summary

- Planes ⟶ Sign of dot product ⟶ Hash values

You can use multiple planes to get a single hash value. Let's take a look at the following example:



$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

$$\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1$$

$$\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$
$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$
$$= 3$$

Given some point denoted by **v,** you can run it through several projections P_1, P_2, P_3 to get one hash value. If you compute $P_1 v^T$ you get a positive number, so you set $h_1 = 1$. $P_2 v^T$ gives you a positive number so you get $h_2 = 1$. $P_3 v^T$ is a negative number so you set h_3 to be 0. You can then compute the hash value as follows.

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$

$$= 1 \times 1 + 2 \times 1 + 4 \times 0 = 3$$

Another way to think of it, is at each time you are asking the plane to which side will you find the point (i.e. 1 or 0) until you find your point bounded by the surrounding planes. The hash value is then defined as:

$$hash_value = \sum_i^H 2^i \times h_i$$

Here is how you can code it up:

```
def hash_multiple_plane(P_l,v):

    hash_value = 0

    for i, P in enumerate(P_l):
        sign = side_of_plane(P,v)
        hash_i = 1 if sign >=0 else 0
        hash_value += 2**i * hash_i

    return hash_value
```

P_l is the list of planes. You initialize the value to 0, and then you iterate over all the planes (P), and you keep track of the index. You get the sign by finding the sign of the dot product between v and your plane P. If it is positive you set it equal to 1, otherwise you set it equal to 0. You then add the score for the ith plane to the hash value by computing $2^i \times h_i$.

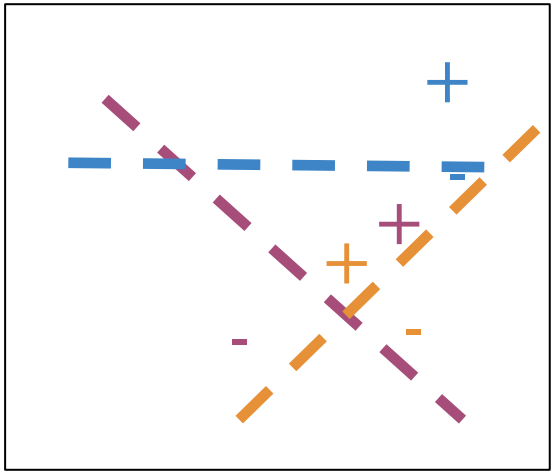
deeplearning.ai

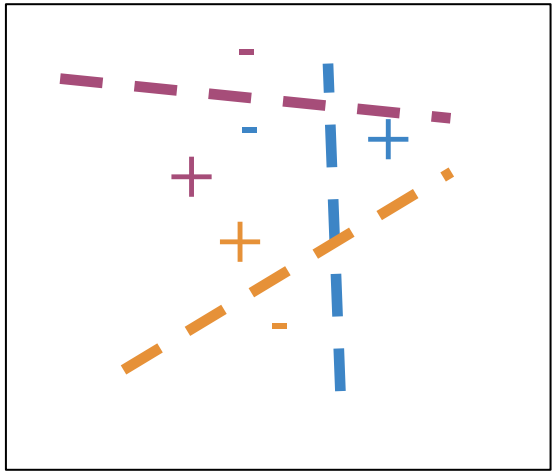
Approximate nearest neighbors

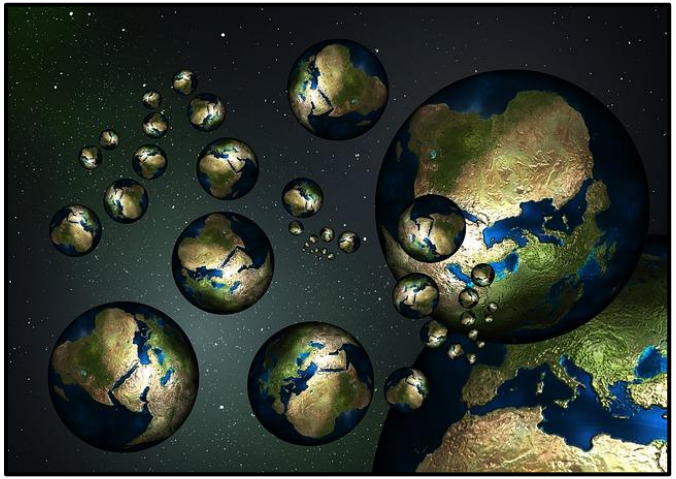
Outline

- Multiple sets of planes for approximate K-nearest neighbors

Random planes

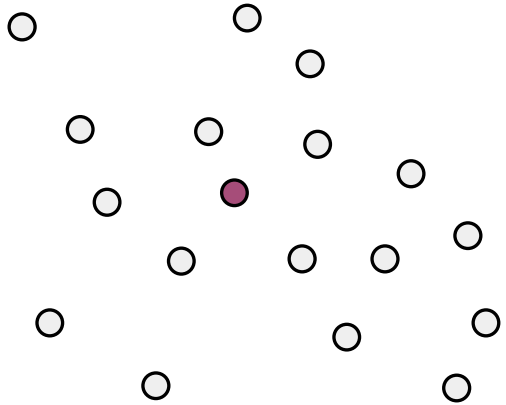




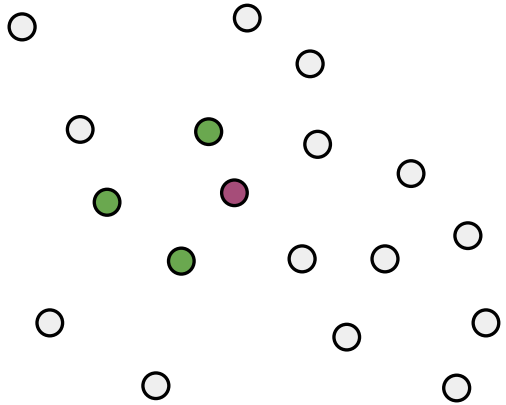


Cultural reference: Spider-Man: Into the Spider-Verse

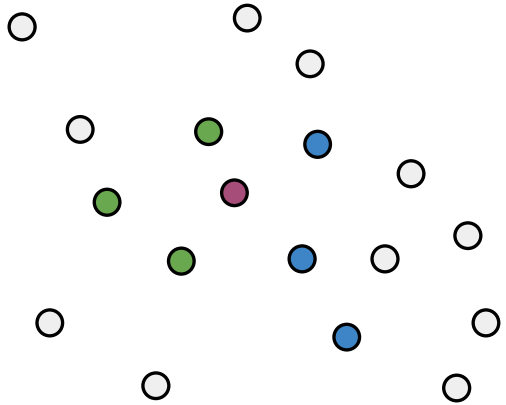
Multiple sets of random planes



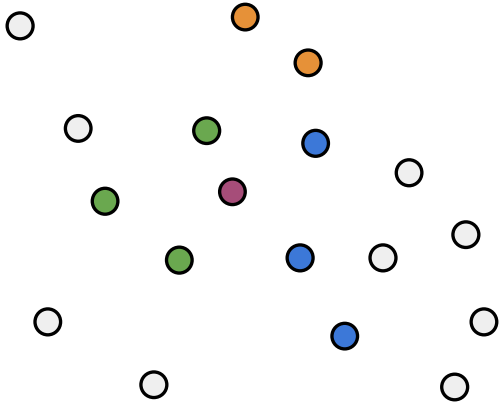
Multiple sets of random planes



Multiple sets of random planes



Multiple sets of random planes



Approximate nearest
(friendly) neighbors

Make one set of random planes

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

```
def side_of_plane_matrix(P,v):
          dotproduct = np.dot(P,v.T)
          sign_of_dot_product =
np.sign(dotproduct)

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

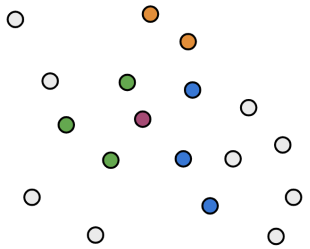
```
array([[1.]
       [1.]
       [1.])
```

See notebook for calculating the hash value!

Summary

- Multiple universes → Locality sensitive hashing → A. K-NN

Approximate nearest neighbors does not give you the full nearest neighbors but gives you an approximation of the nearest neighbors. It usually trades off accuracy for efficiency. Look at the following plot:



Approximate nearest
(friendly) neighbors

You are trying to find the nearest neighbor for the red vector (point). The first time, the plane gave you green points. You then ran it a second time, but this time you got the blue points. The third time you got the orange points to be the neighbors. So you can see as you do it more times, you are likely to get all the neighbors. Here is the code for one set of random planes. Make sure you understand what is going on.

```
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```
v = np.array([[2,2]])
```

```
def side_of_plane_matrix(P,v):
     dotproduct = np.dot(P,v.T)
     sign_of_dot_product = np.sign(dotproduct)
     return sign_of_dot_product

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

```
array([[1.]
       [1.]
       [1.])
```

See notebook for calculating the hash value!

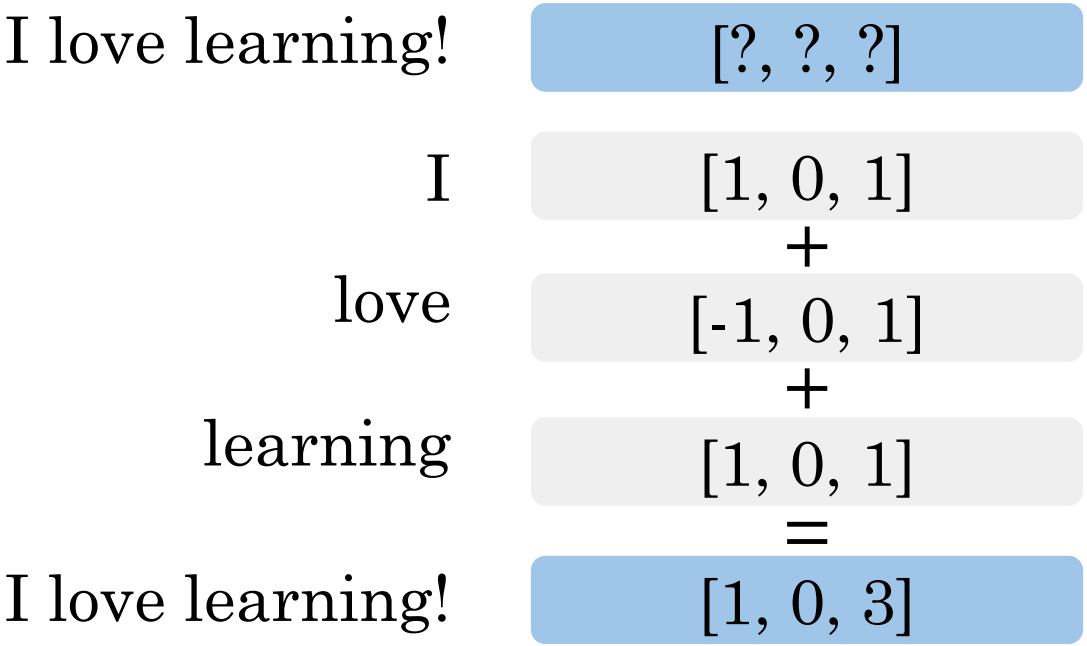
deeplearning.ai

Searching documents

Outline

- Representation for documents
- Document search with K-nearest neighbors

Document representation



Document Search

K-NN!

Document vectors

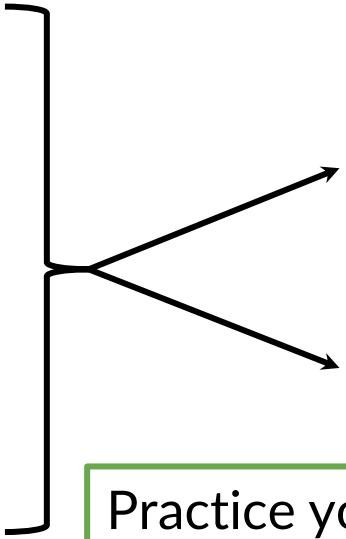
```
word_embedding = {“I”: np.array([1,0,1]),
                  “love”: np.array([-1,0,1]),
                  “learning”: np.array([1,0,1])}

words_in_document = [‘I’, ‘love’, ‘learning’]

document_embedding = np.array([0,0,0])

for word in words_in_document:
        document_embedding +=

print(document_embedding)
```

```
array([1 0 3])
```

Try it!

Revisit Learning Objectives

- Transform vector
- “K nearest neighbors”
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors



Machine translation

Document search

Practice your skills in the assignment!

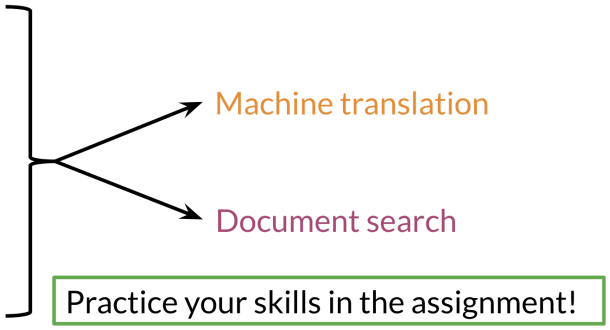
The previous video shows you a toy example of how you can actually represent a document as a vector.

```
word_embedding = {"I": np.array([1,0,1]),
                  "love": np.array([-1,0,1]),
                  "learning": np.array([1,0,1])}

words_in_document = ['I', 'love', 'learning']

document_embedding = np.array([0,0,0])

for word in words_in_document:
    document_embedding += word_embedding.get(word,0)

print(document_embedding)
```

```
array([1 0 3])
```

In this example, you just add the word vectors of a document to get the document vector. So in summary you should now be familiar with the following concepts:

- Transform vector
- "K nearest neighbors"
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors



Good luck with the programming assignment!