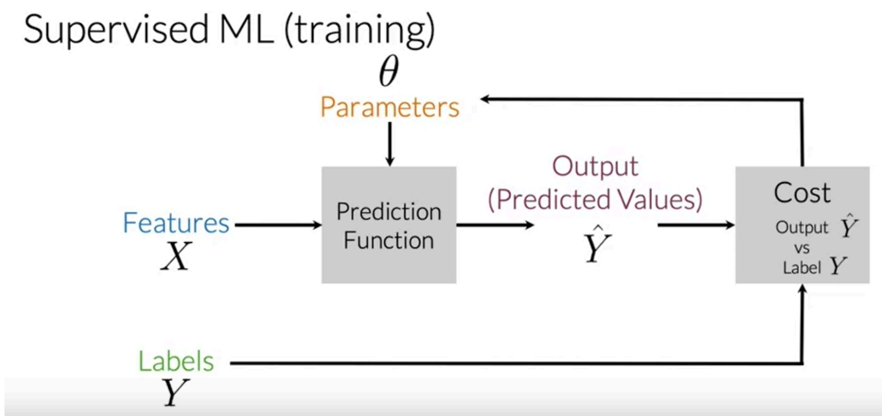
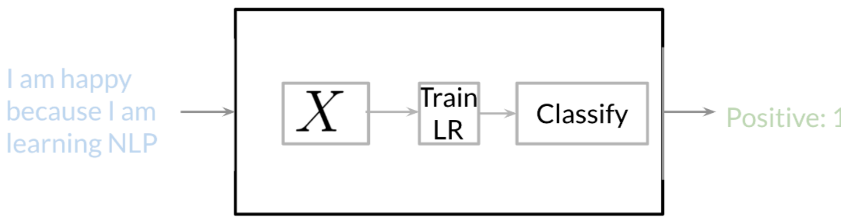
Classification and Vector Spaces:

Supervised ML:





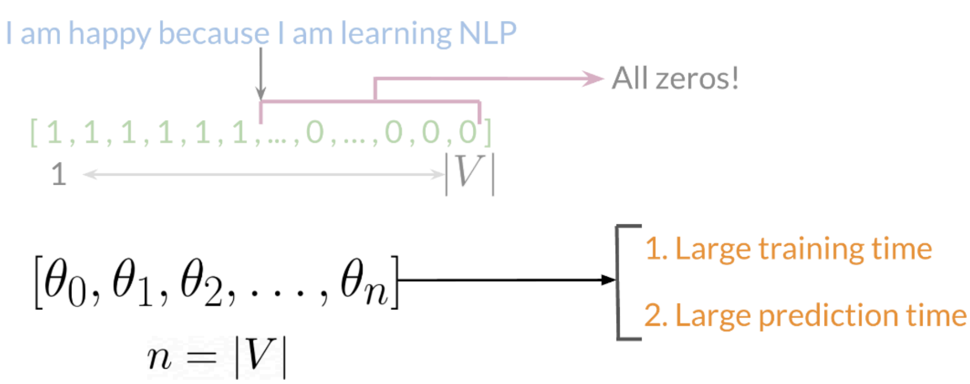
Vocabulary (V):

Set of all possible words in the corpus

Feature Extraction:

Sparse Representation:

For each word, assign 1 if present in vocab, else assign 0



Positive and Negative Frequencies:

For each word in the vocabulary, count how many times that word appear in the corpus of that particular class (positive or negative).

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

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

Feature Extraction with Frequencies:

I am sad, I am not learning NLP

$$X_m = [1, \underbrace{\sum_w freqs(w,1)}_{\boxed{8}}, \sum_w freqs(w,0)]$$

I am sad, I am not learning NLP

$$X_m = [1, \sum_w freqs(w,1), \underbrace{\sum_w freqs(w,0)}_{\boxed{11}}]$$

Here summation is the sum of freq of "set of words" for that sentence.

Preprocessing:

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.

6. Removing Punctuations

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

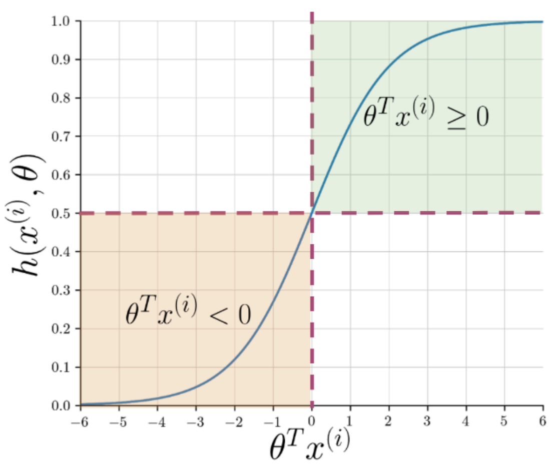
Preprocessed tweet:
[tun, great, ai, model]

Complete Implementation:

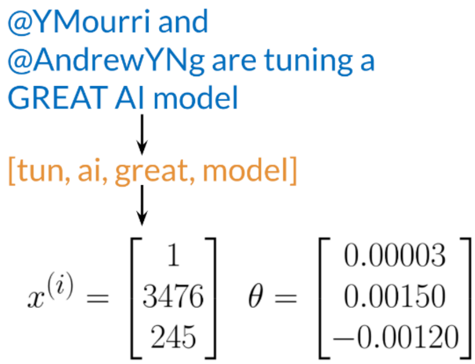
```
freqs = build_freqs(tweets,labels) #Build frequencies dictionary

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
```

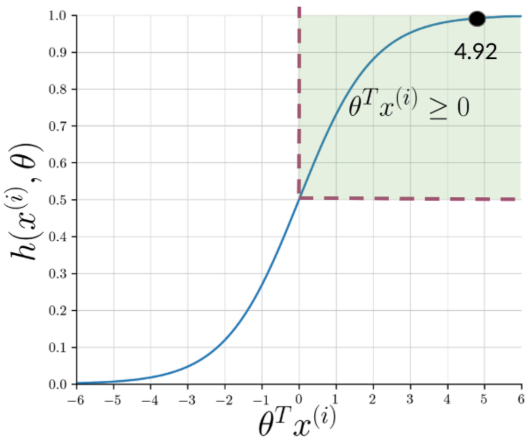
Logistic Regression:

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

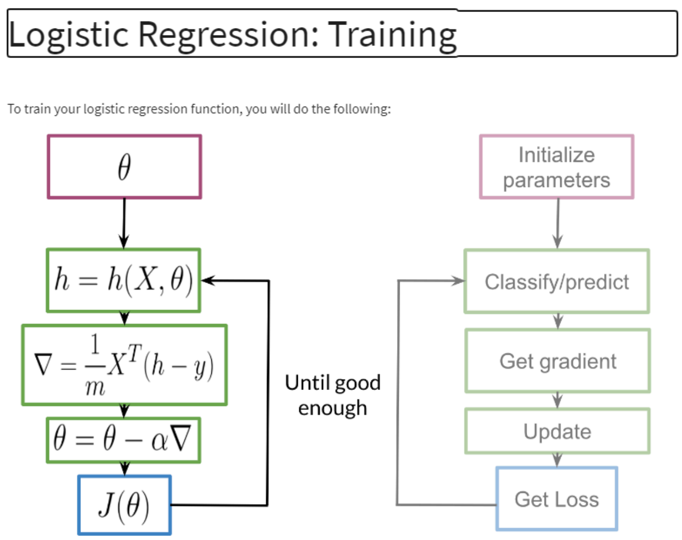


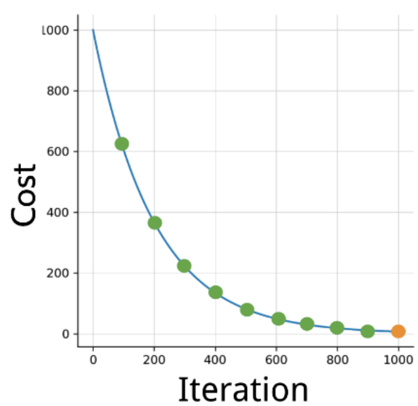
It uses sigmoid function





Training Classifier:





Testing Classifier:

- X_{val} Y_{val} θ

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

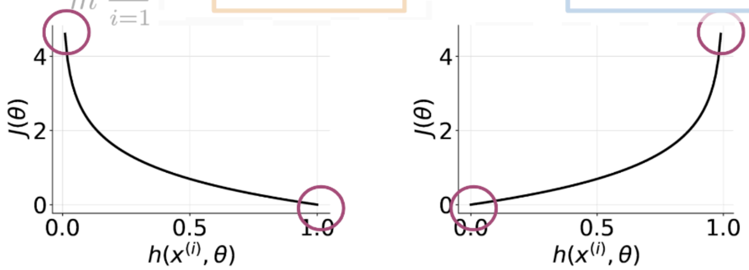
$$pred = h(X_{val}, \theta) \geq 0.5 \qquad \begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 = \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

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

Cost Function:

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

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



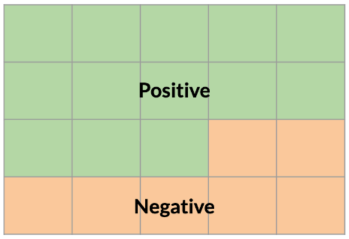
Y = 1 **Y = 0**

Derivation of loss function:
https://www.coursera.org/learn/classification-vector-spaces-in-nlp/supplement/b3fHH/optional-logistic-regression-cost-function

Derivation of gradient:
https://www.coursera.org/learn/classification-vector-spaces-in-nlp/supplement/afcaR/optional-logistic-regression-gradient

Naive Bayes:
Probability:

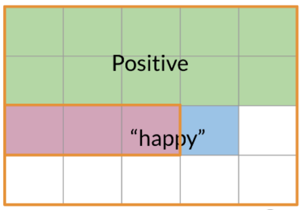
Corpus of tweets



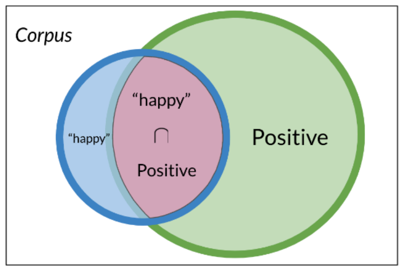
A → Positive tweet

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

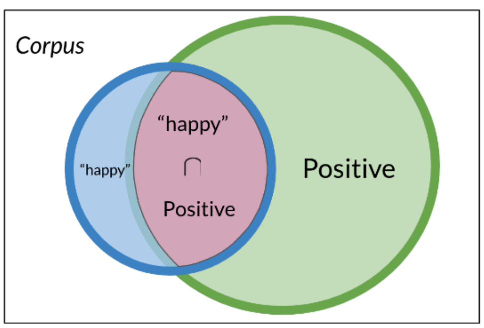
P(Negative) = 1 - P(Positive) = 0.35



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



Conditional Probabilities:



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

Bayes Rule:

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

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

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

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

Naive Bayes:

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

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

P(word|pos_class) or P(word|neg_class)

Tweet: I am happy today; I am learning.

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

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

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

Naive Bayes

Laplacian Smoothing:

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

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

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

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

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

Log Likelihood:

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

The first component is called the log prior and the second component is the log likelihood.

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	

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

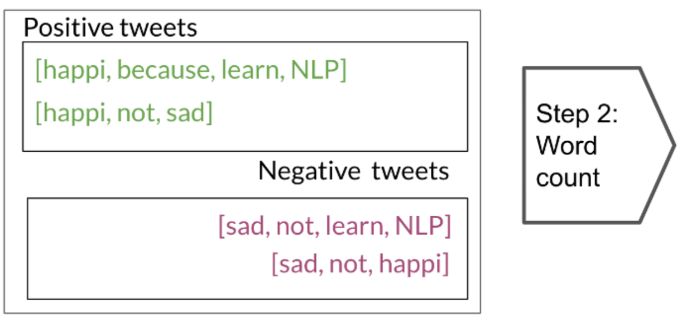
Training Naive Bayes:

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.

Testing Naive 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

Application:

- Author identification
- Spam filtering
- Information retrieval
- Word disambiguation

Assumptions:

- Independence
- Relative frequency in corpus

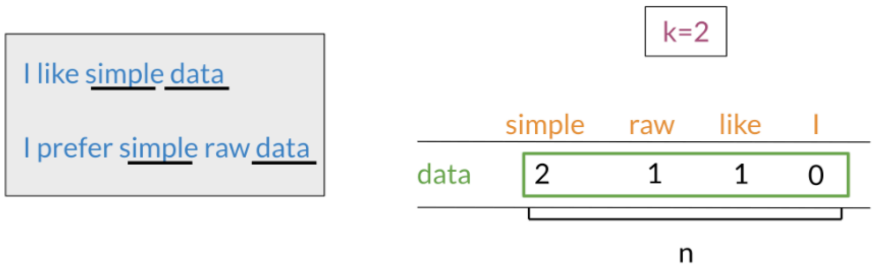
Means unequal frequencies of classes in the dataset

Errors:

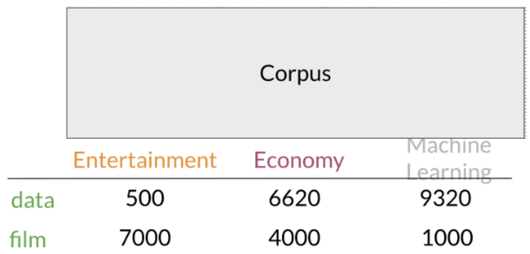
- Removing punctuation and stop words
- Word order
- Adversarial attacks

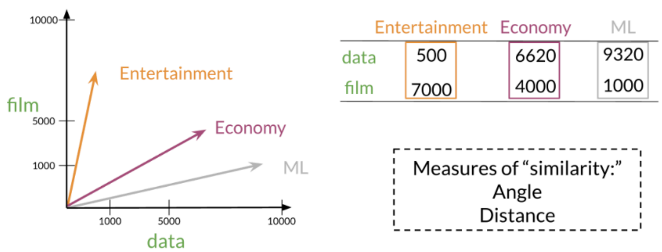
Vector Space:

Word by Word Design:



Word by document design:



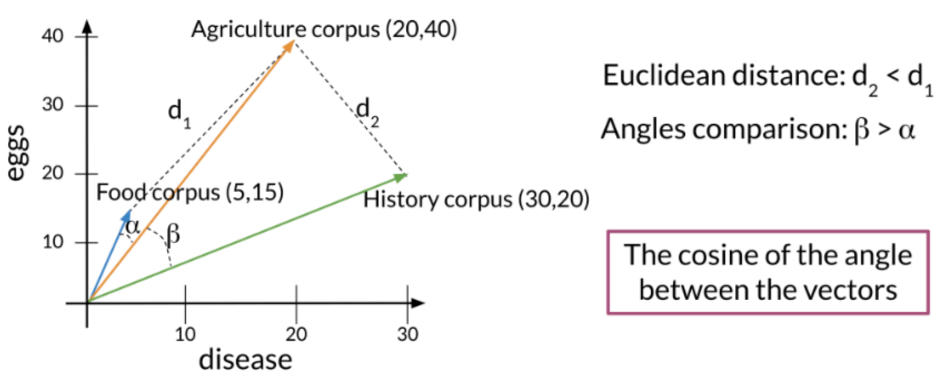


Euclidean Distance:

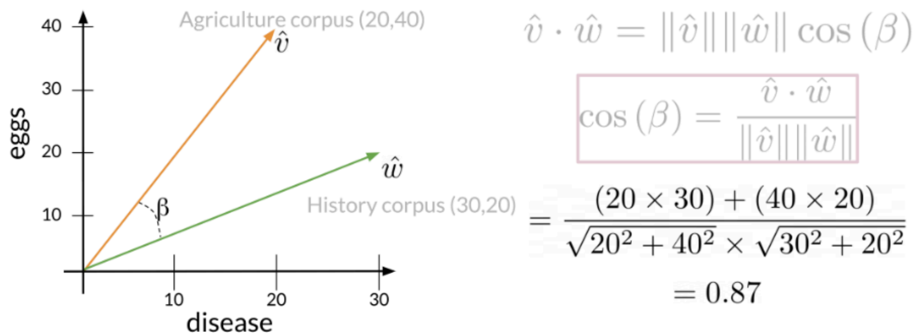
$$d(B, A) = \sqrt{((B_1 - A_1)^2 + (B_2 - A_2)^2)}$$

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

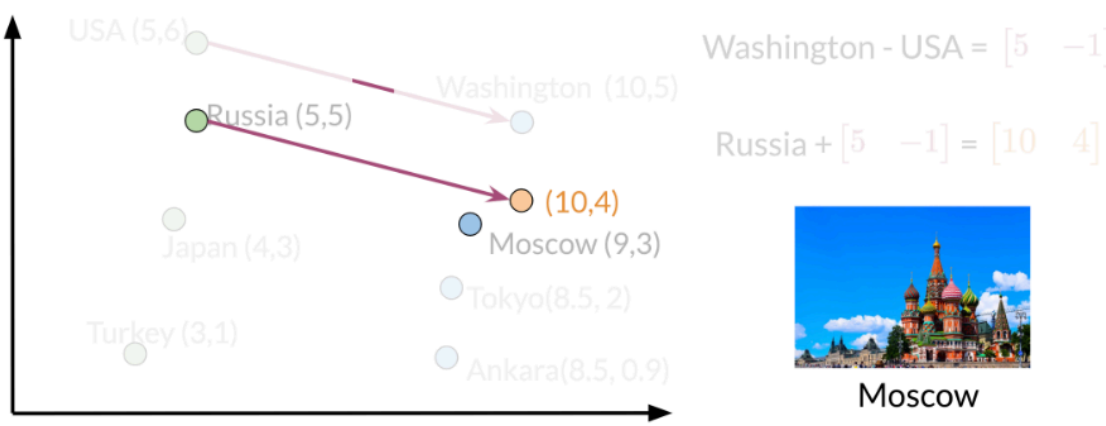
Cosine Similarity:



If corpus are of different sizes, then cosine similarity is a better metric.

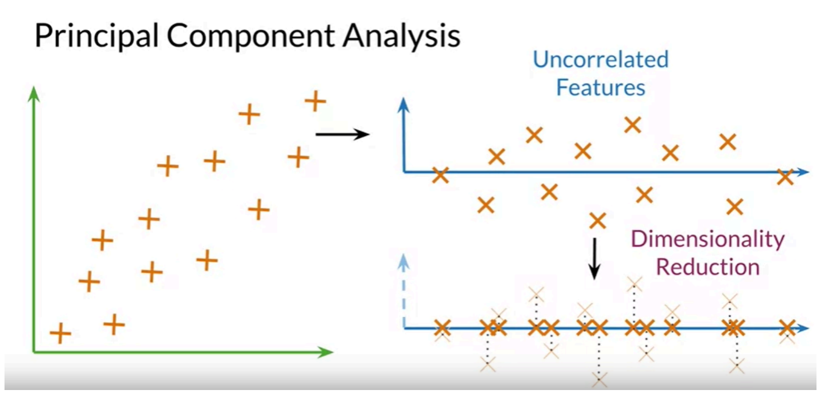


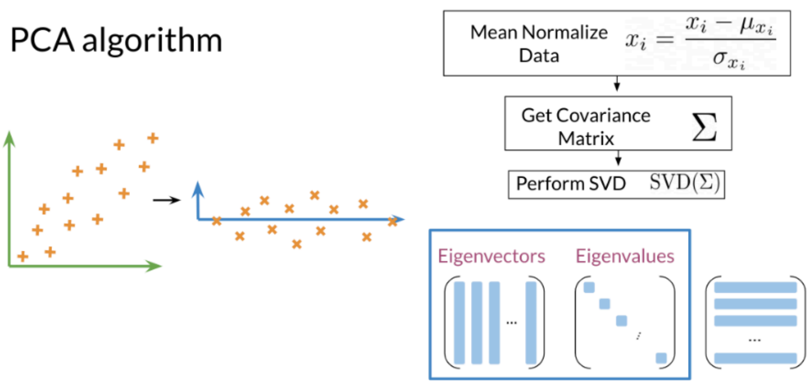
Manipulating vectors:



PCA:

- Dimensionality Reduction
- Unsupervised



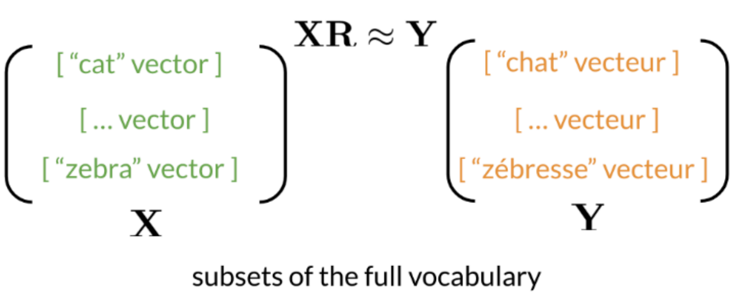


<u>**Steps to Compute PCA:**</u>

- 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 $XU[:, 0:n]$.

Machine Translation:

Transforming word vectors:



Steps required to learn R:

- Initialize R
- For loop

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

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

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

Frobenius Norm:

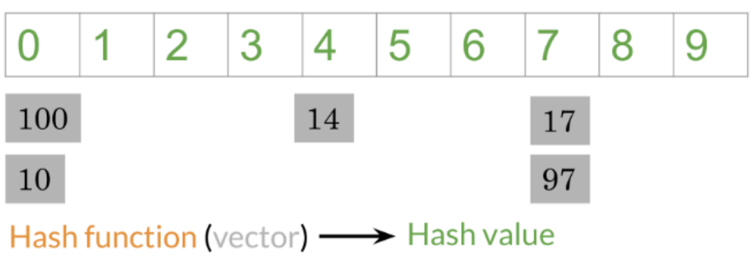
$$\|\mathbf{XR} - \mathbf{Y}\|_F$$
$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$
$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$
$$\|\mathbf{A}_F\| = 4$$
$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Note: For Simplicity, we can minimize the square of frobenius norm.

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

Here, m is the number of words or rows in the R matrix.

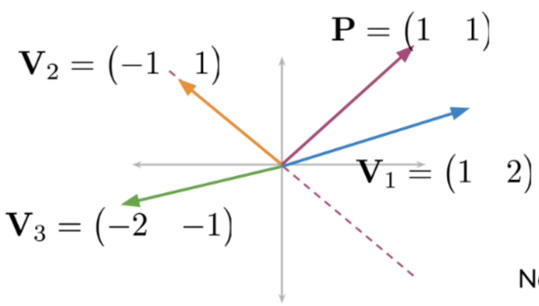
K-nearest neighbours:
Hash Tables:



Hash value = vector % number of buckets

```
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
```

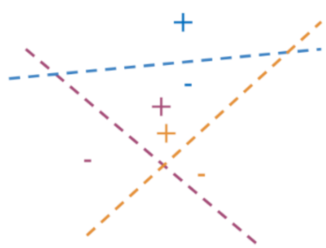
Locality sensitive hashing:



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

Notice the signs?

P is the perpendicular to the plane



$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$
$$\boxed{\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1}$$
$$\boxed{\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0}$$

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

```
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
```

Approximate nearest neighbors:

```
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!

Searching Document:

```
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])
```