Week 1 - Logistic Regression

A General Framework for Supervised ML

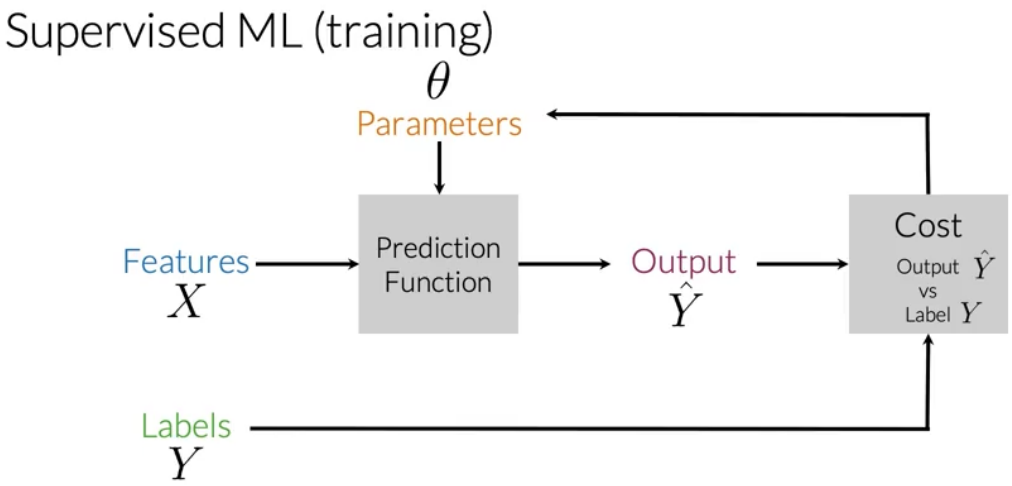

Figure 1

Sentiment analysis is a task in which a statement (tweet in our case) is predicted as a positive or negative one!

This can be done using logistic regression classifier.

Steps

(A) Extract Features

(B) Train Models

(C) Make Predictions

(A) Extract Features

Consider our corpus as the following texts

```
I am happy because I am learning NLP
I am happy
I am sad, I am not learning NLP
I am sad
```

We can build a list of unique words (V).

To express our tweets in the for of numbers, we can make a sparse matrix that indicates whether the word in the list V is present or not a.k.a sparse representation

Problems with this representation

- ➤ Too slow
- ➤ too much memory

So we skip this method, and go for the method suggested next!

1. Preprocess the tweets
   - ➤ remove stop words (done using the nltk library)
   - ➤ remove punctuation (done using regular expressions)
   - ➤ remove tweet handles and urls (done using regular expressions)
   - ➤ stemming
   - ➤ lowercase all words and tokenize (.lower() or nltk.tokenize.TweetTokenizer)

tokenization refers to the process of dividing a string/corpus into tokens. Here, we divide into words (so the terms tokens and words will be used interchangeably)

Stemming refers to the process of reducing every word to its base. e.g. the words learning, learn, and learned will all be stemmed to the base word 'learn'. For stemming, here we have used the Porter Stemmer algorithm

2. Build a list of (V) unique words

   In our case , it will be

Vocabulary
I
am
happy
because
learning
NLP
sad
not

   Here, V = 8.

3. Build a word frequency vector table (freqs) that counts the occurence of each word for each class. This is like a correlation matrix for words and 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

Table 1

   This table is known as a frequency dictionary

4. Build a feature vector for each tweet

$$X_m = [1,\ \Sigma C(w,1),\ \Sigma C(w,0)]$$

   where 1 is the bias, $\Sigma C(w,1)$ is the sum of positive frequencies and $\Sigma C(w,0)$ is the sum of negative frequencies

   e.g. For the third tweet, we will ignore words in the dictionary that don't appear in the tweet

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

   X_3 = [1 8 11]

Your feature matrix will be of size (m,3), where m is the number of tweets

Pseudo-code

The pseudo-code is given below

```
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
```

Figure 2

(B) Train Model

General steps for training.

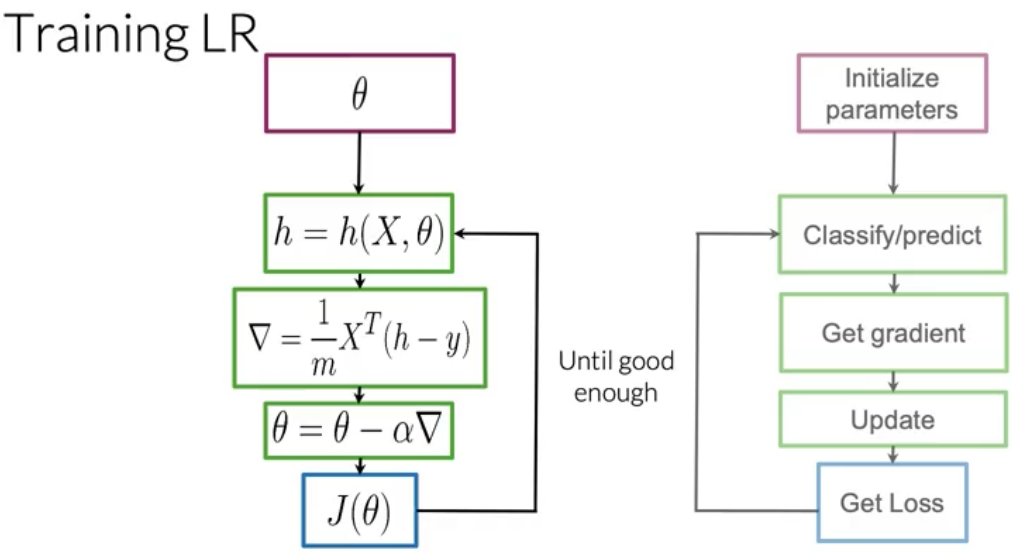

Figure 3

(C) Make Predictions

For each number in the output matrix, compare with a threshold (e.g. 0.5). If it is higher than 0.5, convert it to 1. Else, 0.

Calculate accuracy using

$$ACC = \sum_{i=1}^{m} \frac{1}{m} [pred^{(i)} == y^{(i)}]$$

Week 2 - Naive Bayes

Bayes' Rule

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A \cap B)}{P(A)}$$

$$P(A|B)P(B) = P(B|A)P(A)$$

$$P(A|B) = P(B|A)\frac{P(A)}{P(B)}$$

Naive Bayes is a supervised ML algorithm which assumes that all features are independent of each other (hence the naivety)

We will build upon table 1. For this week, the corpus (and hence the table!) has been modified.

```
I am happy because I am learning NLP
I am happy, not sad
I am sad, I am not learning NLP
I am sad, not happy
```

Find the sum of all words in each class.

Vocabulary	PosFreq (1)	NegFreq (0)
I	3	3
am	3	3
happy	2	1
because	1	0
learning	1	1
NLP	1	1
sad	1	2
not	1	2
Nclass	13	13

Instead of counts, we need probabilities. Divide each term with the corresponding Nclass. For the both classes, divide by 13 (coincident.)

Vocabulary	Pos (1)	Neg (0)
I	0.24	0.24
am	0.24	0.24
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.15
not	0.08	0.15
Sum!	1	1

As a side note, words with similar probabilities don't add meaning to the sentiment.

After building probabilities table, to find the sentiment, calculate for each tweet the probability product.

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

given that there are m words in the tweet.

Laplacian Smoothing

Some words might appear in one class only. If so, P(w_i|0 or 1) = 0 and taking logarithm or dividing be 0 will give infinity. By using LS, all probabilities will be non-zero

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

N_{class} is the freq. of all words in the class
V is the number of unique words in the vocabulary

e.g.

$$P(I|1) = \frac{3+1}{13+8} = 0.19$$

$$P(because|0) = \frac{0+1}{13+8} = 0.05$$

and so on...

Vocabulary	Pos (1)	Neg (0)
I	0.19	0.19
am	0.19	0.19
happy	0.14	0.10
because	0.10	0.05
learning	0.10	0.10
NLP	0.10	0.10
sad	0.10	0.14
not	0.10	0.14
Sum!	1	1

e.g. for a given tweet (which may have unknown words since we are testing; such words are ignored!) : I am happy today I am learning

$$Product = \frac{0.19}{0.19} * \frac{0.19}{0.19} * \frac{0.14}{0.10} * \frac{0.19}{0.19} * \frac{0.19}{0.19} * \frac{0.10}{0.10} = \frac{0.14}{0.10} = 1.4$$

Since product > 1, it is a positive tweet

Ratio of Probabilities

Vocabulary	Pos (1)	Neg (0)	ratio
I	0.19	0.19	1
am	0.19	0.19	1
happy	0.14	0.10	1.4
because	0.10	0.05	2
learning	0.10	0.10	1
NLP	0.10	0.10	1
sad	0.10	0.14	0.6
not	0.10	0.14	0.6

All words with ratio greater than 1 convey more sense of positivity and vice versa.

As a last step, in order to account for a misbalanced dataset (if any), we multiply with $\frac{P(pos)}{P(neg)}$. The log os this fraction is known as the logprior.

Finally, we take log to fix large and small numbers.

$$\log\left[\frac{P(pos)}{P(neg)}\prod_{i=i}^{m}\frac{P(w_i|pos)}{P(w_i|neg)}\right] = \text{logprior} + \sum_{i=1}^{m}\lambda(w_i)$$

where

$$\text{logprior} = \log\frac{P(pos)}{P(neg)}$$

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

Once you take logs, you need to compare with 0.

Vocabulary	Pos (1)	Neg (0)	λ
I	0.19	0.19	0
am	0.19	0.19	0
happy	0.14	0.10	0.33
because	0.10	0.05	0.69
learning	0.10	0.10	0
NLP	0.10	0.10	0
sad	0.10	0.14	-0.51
not	0.10	0.14	-0.51

Now, for a given tweet 'I am happy because I am learning', we need to find the log-likelihood

log-likelihood = 0 + 0 + 0.33 + 0.69+ 0 +0 0 = 1.02

Since it is greater than 0, it is a positive tweet.

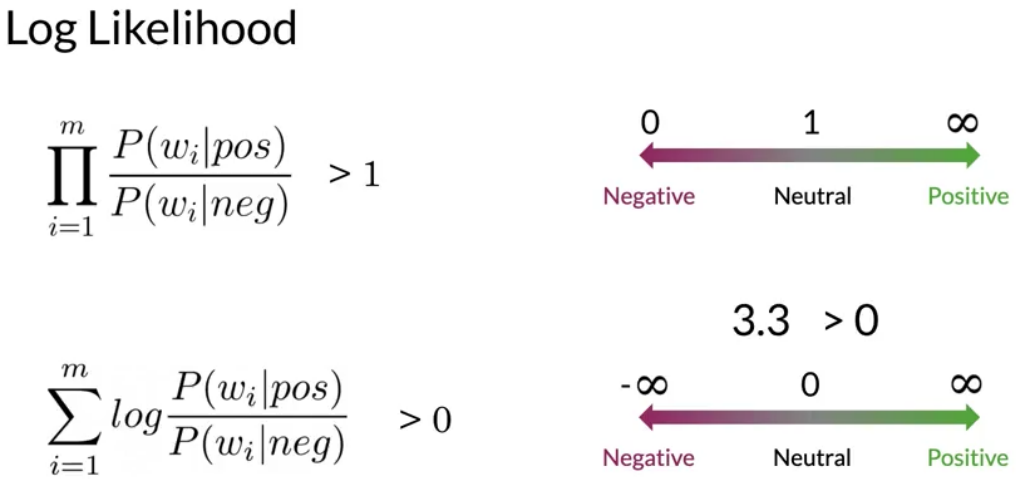

Figure 4

Sources of Errors

➤ Punctuations and stopwords (e.g. emojis) may carry meaning

➤ word order is important and cannot be ignored

➤ irony and sarcasm can flip the sentiment of the sentence

Week 3 - Vector Spaces

Unlike previous models, vector space models allow to incorporate meanings while representing sentences. Sentences with similar meanings will be 'closer' even though they might be made of different words, while sentences with different meanings will be 'farther' even though they are made of similar words.

e.g.

Different	Similar
Where are you going?	What is your age?
Where are you from?	How old are you?

It will also allow us to capture dependencies between words. i.e. some words appear more often together

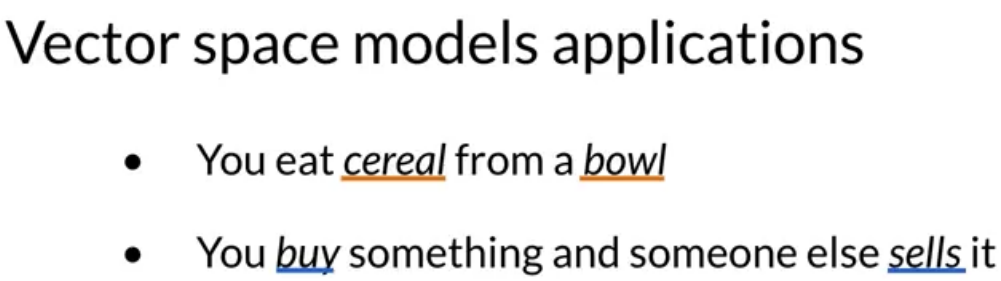

Figure 5

Word by Word Design

Co-occurence of two words is the number of times they occur together within a certain distance k.

Consider the following corpus

```
I like simple data
I prefer simple raw data
```

For k =2, our co-occurence matrix will be.

	simple	raw	like	I
data	2	1	1	0

Table 2

Note that this requires the vector to be equal to the size of the vocabulary.

Word By Document Design

The length of the vector can be reduced by choosing the vertical columns as categories. i.e. count how many times each word appears in a given category.

e.g. if the three categories in the document are entertainment, economy, and ML, our matrix might look like.

	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

Table 3

If we draw them on a graph, it will show that economy and ml are more alike than entertainment.

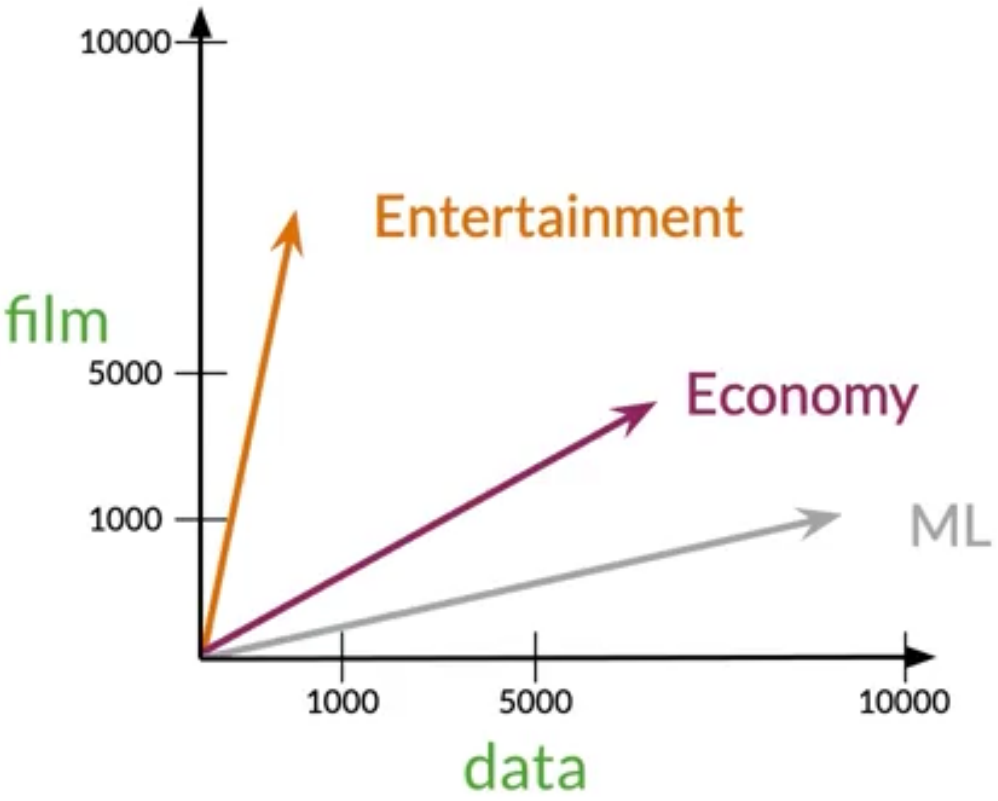

Figure 6

Two functions are used to determine the similarity between vectors

- Euclidean distance
- Cosine similarity

Euclidean distance is simply the distance between the two points. It is the norm of the difference of the vector

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

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
```

Figure 7

Now consider the following example where the size of the corpus is different

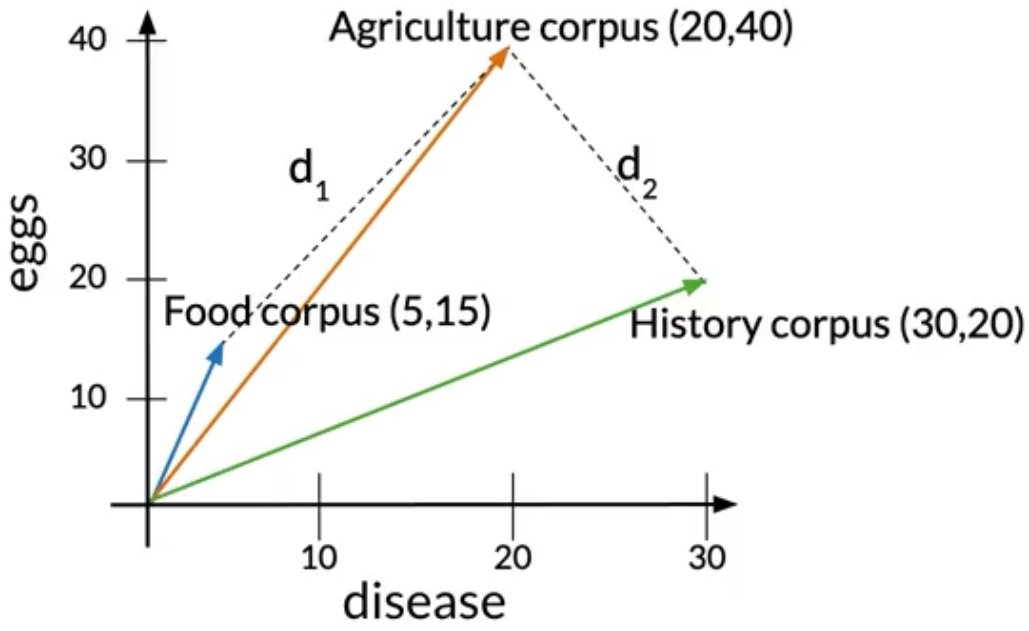

Figure 8

In such cases, the Euclidean will mostly give the wrong answers. If the corpus were balanced (i.e. there were more words in the Food corpus, its vector would have been longer and closer to Agriculture)

Cosine similarity is more useful if the corpora size is different. You can see that the angle between food and agriculture is smaller than the angle between agriculture and history.

The norm of a vector is given by

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

and the dot product between two vectors is given by

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

Now,

$$\vec{v} \cdot \vec{w} = \|\vec{v}\|\|\vec{w}\| \cos\beta$$

$$\cos\beta = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\|\|\vec{w}\|}$$

How to use these vectors?

Figure 9

We will find the vector that is closest to the point (10,4), which in our case is Moscow.

Unfortunately, since the vectors are in a higher dimension, we cannot visualize them like in the figure 9.

But to do so, we can use an algorithm called PCA that can reduce the dimensions.

PCA Algorithm

Recall that eigenvector represent the uncorrelated features for your data. Eigenvalues represent the amount of information retained by each feature.

Steps

➤ Mean normalize data

➤ Get the covariance matrix

➤ Perform SVD (or eigendecomposition ???)

Once you get the eigenvectors (U) and eigenvalues(S) (with the latter arranged in descending order), we can reduce dimensions (to n) as follows.

$$X' = XU[:, 0 : n-1]$$

The percentage of information retained is

$$\frac{\sum_{i=0}^{n-1} S_{ii}}{\sum_{j=0}^{d-1} S_{jj}}$$

where d is the actual number of dimensions.

Week 4 - Machine Translation

Consider words given in two different languages e.g. English and French

For each language, the rows represent corresponding words. The dictionary is stored in the form of two matrices, E and F.

We need another matrix to transform a corresponding row of E into F. Fortunately, we need only a subset of E and F to train our model and not the whole set.

$$ER = F$$

Our predictions might have some error. So our steps look as follows

➤ Insitialize a matrix R (random maybe?)

➤ calculate loss

$$\text{Loss} = \|ER - F\|_F^2$$

➤ calculate gradient

$$g = \frac{d}{dR}\text{Loss} = \frac{2}{m}(E^T(ER - F))$$

➤ update R

$$R = R - \alpha g$$

➤ Repeat the above three steps until necessary.

Once our model is trained, and a suitable value of R is found, the error will be minimized. Still, when we try to test our model, the result will not perfectly match the answer. We need to find the nearest neighbor!

K-Nearest Neighbor Algorithm

We divide our list of possibilities into regions. And we will search for closest answer in the same region.

Definitions:

A hash function takes in a vector and outputs a hash value.

A hash value is an indication of the region (or 'bucket').

A simple Example

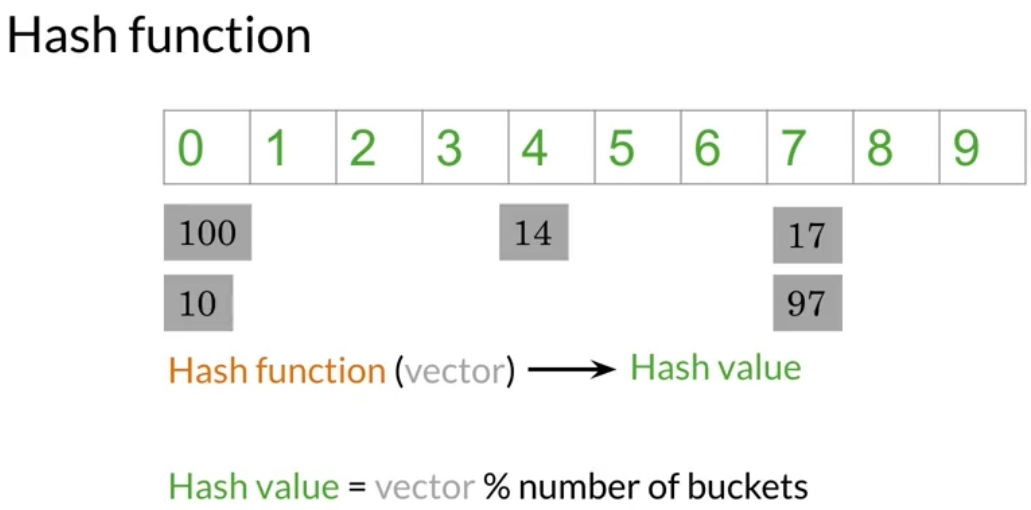

Figure 10

And the corresponding code:

Create a basic hash table

```
def basic_hash_table(value_l,n_buckets):
    def hash_function(value_l,n_buckets):
        return int(value) % 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
```

Figure 11

But this hash method does not place nearby neighbors together!

So we use locality sensitive hashing, instead of simple hashing.

Steps

Consider the following region

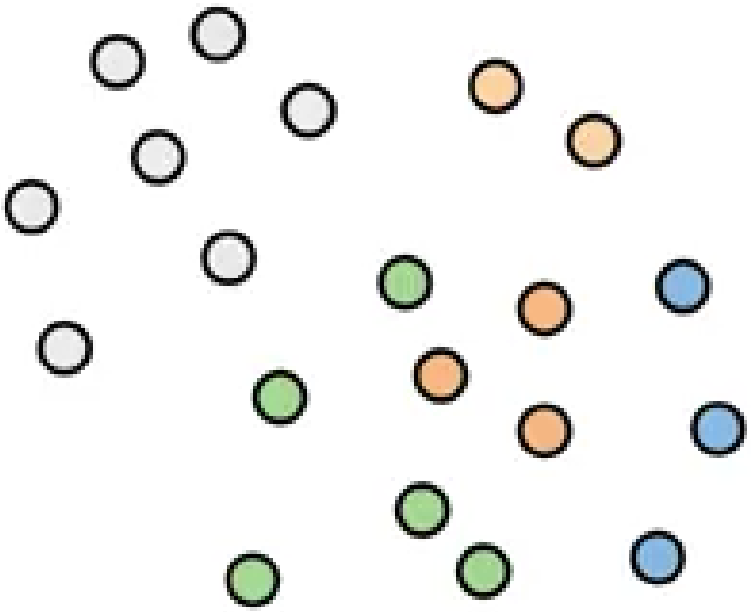

Figure 12

➤ Divide region into planes

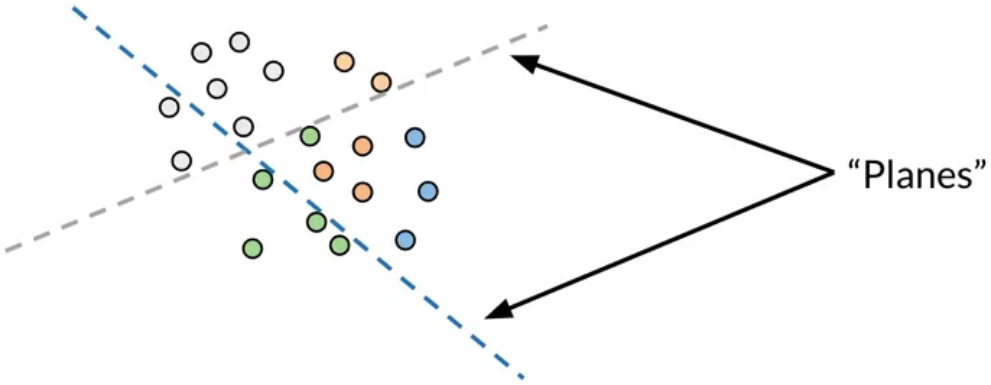

Figure 13

➤ Find on which side of the plane a specific region is present. To do this, draw the normal vector.

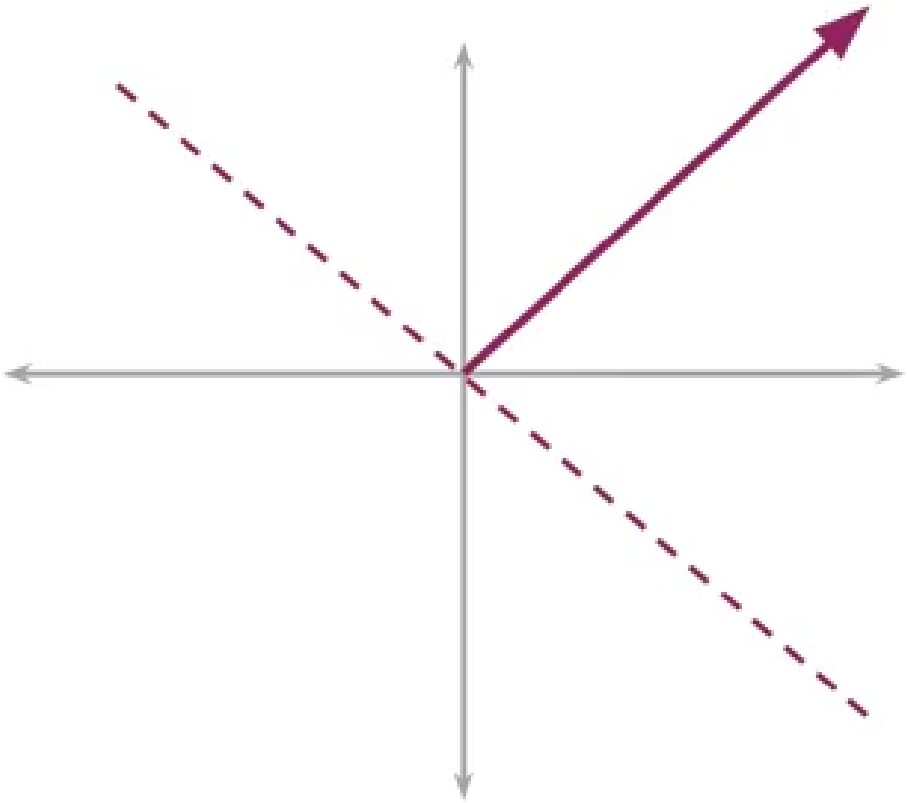

Figure 14

➤ Take the dot product. If it is on the same side, the product will be positive

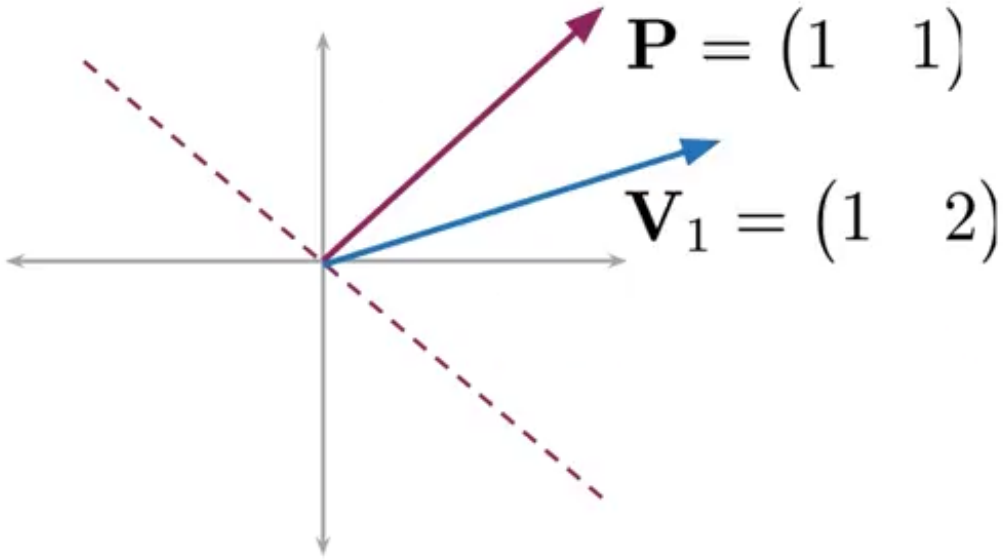

Figure 15

➤ Else, it will be negative

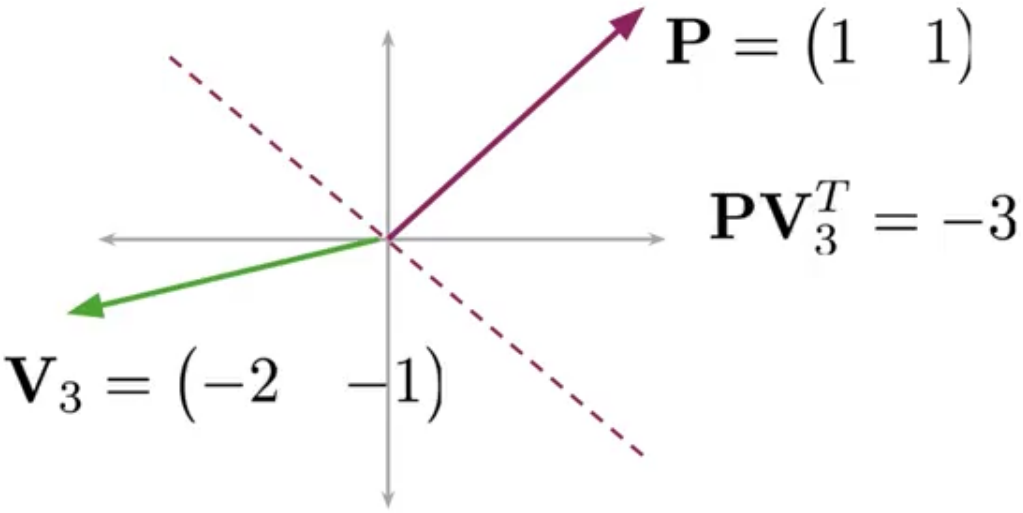

Figure 16

➤ Summary

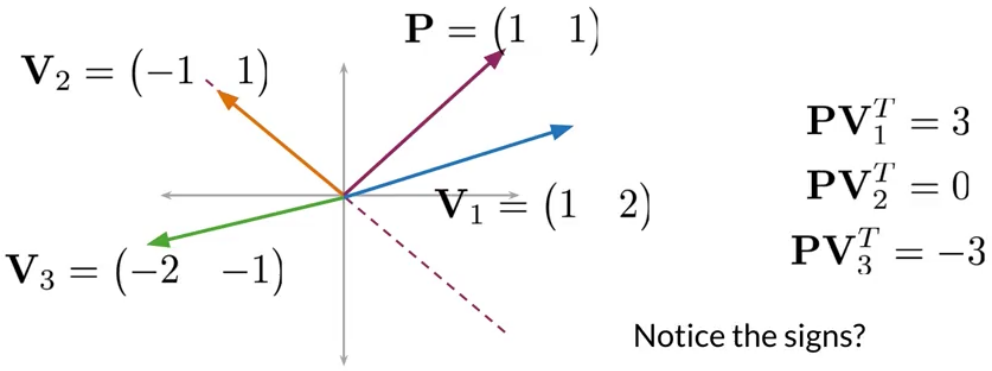

Figure 17

Code

```
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
```

Figure 18

But a single plane will not help! So we combine multiple planes. e.g. 3. We will calculate the side of the vector w.r.t all planes. And then put the three numbers inside a single value - the hash value.

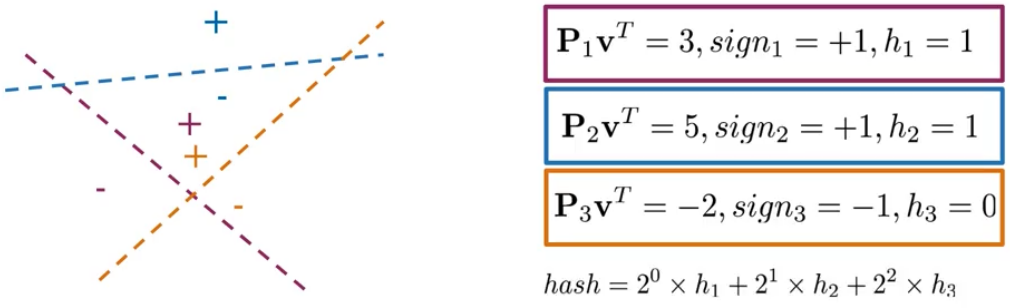

Figure 19

Corresponding code:

```
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
```

Figure 20