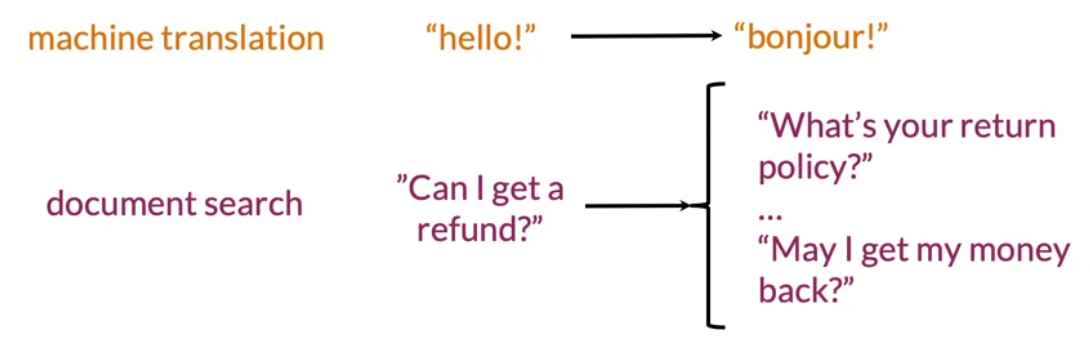
What we can apply ?



Transform vector: such as a word vector.

K nearest neighbours which is a way of seraching for similar items.

Hash table: which will help you assign your word vectors into subsets

Divide vector space into regions.

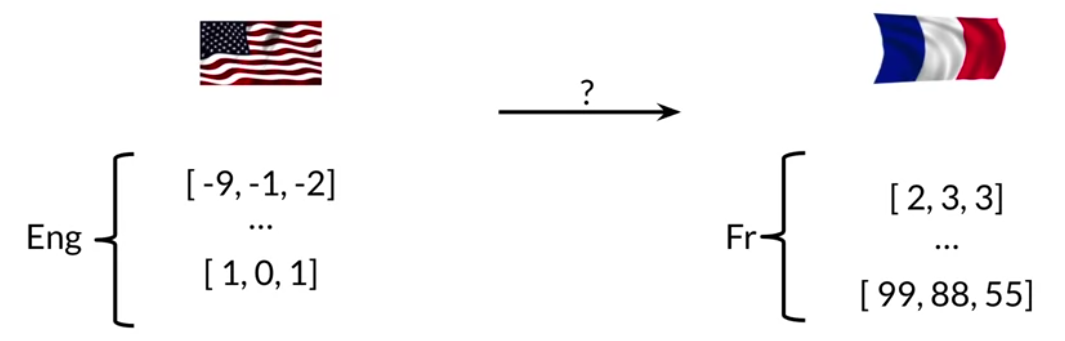
Locality sensitive hasing : which helps you perform approximate k nearest neighbours, an efficient way of searching for similar word vector.

**Transforming word vectors**

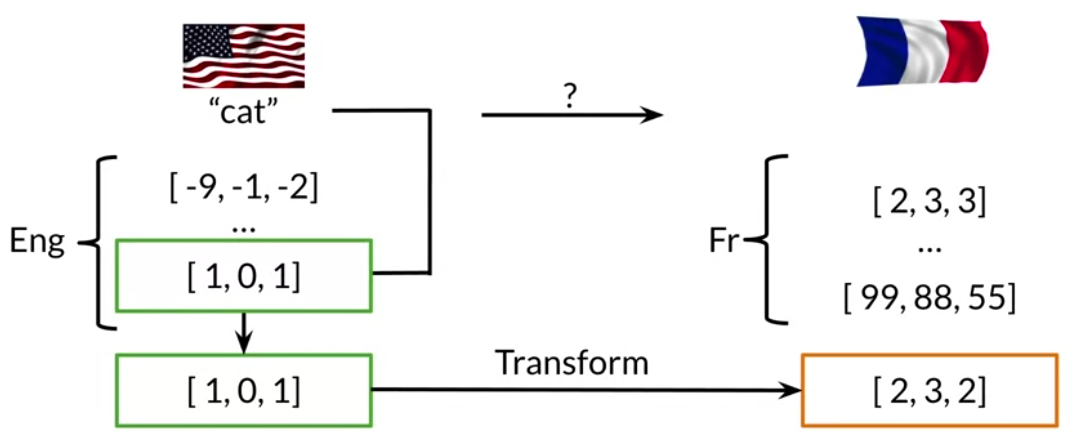
In order to translate an english word to a frensh word, one way would be to generate an extensive list of english words and their associated frensh word.

If you ask human to do this, you would find someone who knows both languages to start making a list.

If you want to a machine to learn how to do this, you would calcualte word embeddings associated with english and word embeddings associated with french.



Next, reterive the english word embeding of a particular english word such as cats, then find some way to transform the english word embeding into word embeding that has a meaning in the frensh word vector space.



* We will see how to convert from the english word vector space to the frensh word vector space in a moment.
* Next, you will take the transformed word vector and search for word vectors in the frensh word vector spcae that are most similar to it.

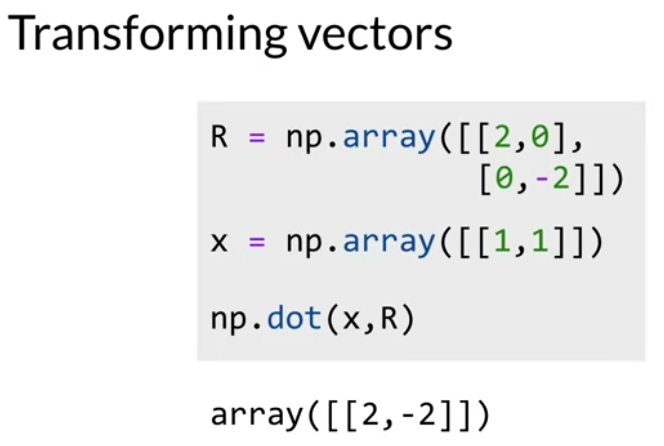
Candidat مرشحين =

The most similar words are Candidates words for you translations.

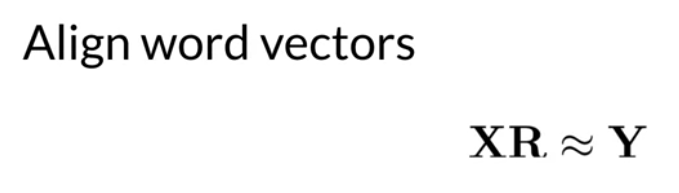
You want find the matrix that can do this transformation for you.

To create transforming vector: Define the matrix and Deifne vector

Muliply matrix and vecotr.



There can be matrix that transforms oru english word vectors into relevant French word vectors,

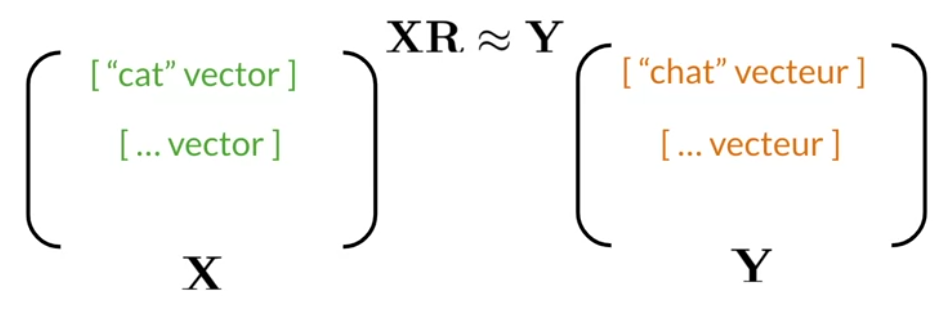


Which we will dentoe as R?

We can start with a randomly selected matrix R and then see how it performs when you try to translate the english word in matrix x and compare that to the actual French word vectors, which is in the matrix Y.

1. You will first to get subset of english words and differnent equivlance get the respective word vectors, and stack the word vectors in their respective matrics X and Y.

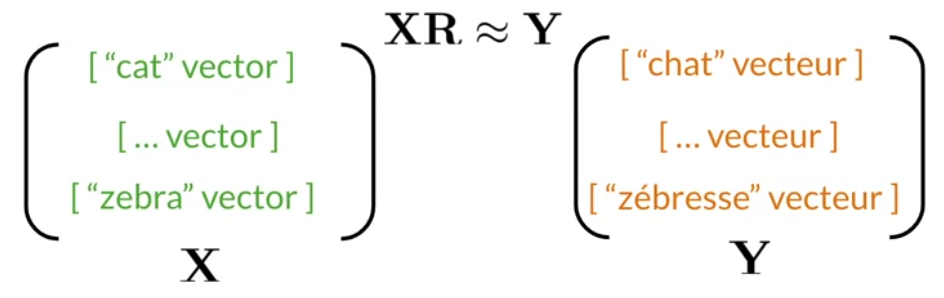
“ the key here is to keep the rows lined up or to align the word vectors.”



This means that if the first word row of matrix X contains the word cat, then the first row of matrix Y should contain the french word for cats,which is chat.

Note

If I already have the english words and their french translations, why I do o need train model to do this?

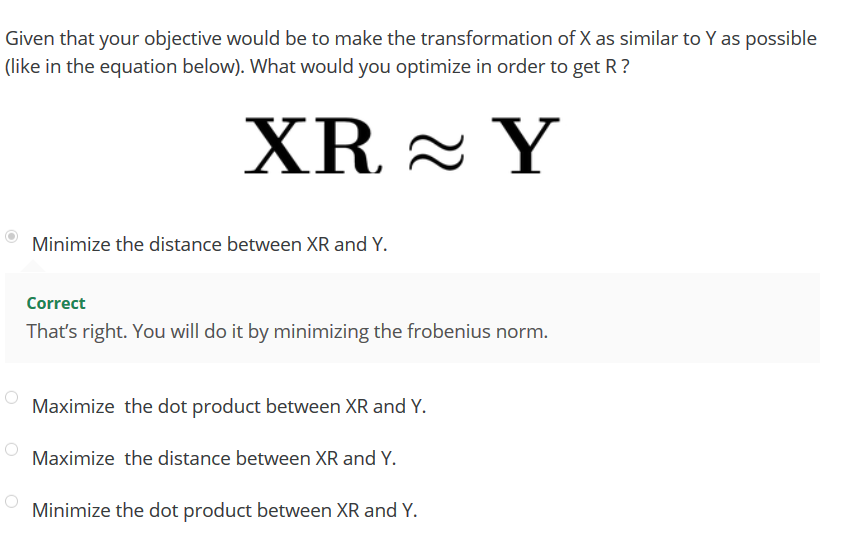


لية منحطش دة python diectionary ?

The collect a subset of these words to find your transformation matrix. And if it words well , then the model can be used to translate words that are not part of your orginal training set.

So you need to **train on a subset of the english french vocabulary and not the entire vocabulary.**

**Quiz**



Sovling for R.

First, we compare the transalation X times R with the actual French word embedding in Y.



Now, think of it as a measure of how far apart the attempt to tanslation **and the actual French vectors are**.

1. If we start with a random matrix R, you can gradually improve this matrix R in a loop
   1. First, compute the gradient by taking the derviative of the loss function with repsect to the matrix R.



* 1. **Next, update the matrix R by subtracting the gradient, but note that it`s the gradient rated by the learning grade alpha.**



**You can either pick a fixed number of times to go through the loop or check the loss at each iteration and break out of the loop when the loss falls between certain threshold.**

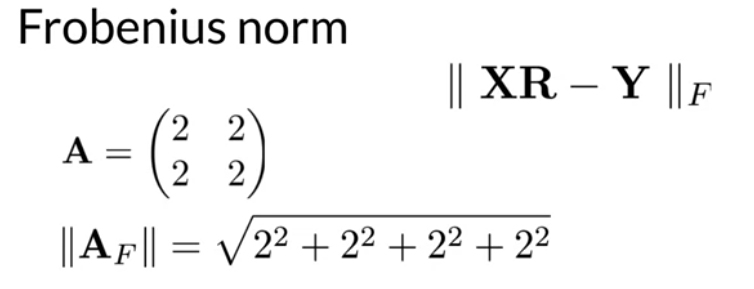
**Frobenius norm: this is measureing the magntiude or the norm of a matrix.**

**Example:**

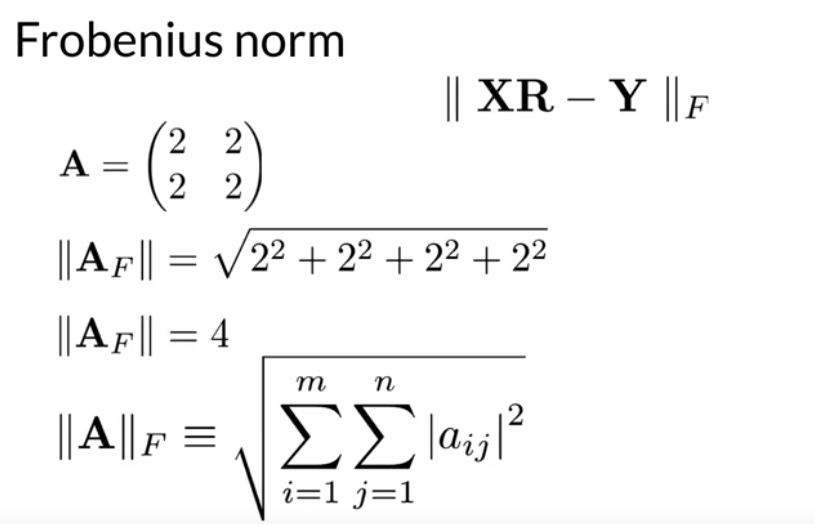
Let`s see an exmpale of calcuating this norm and then see the genearl formula.

Two word in this dictionary:

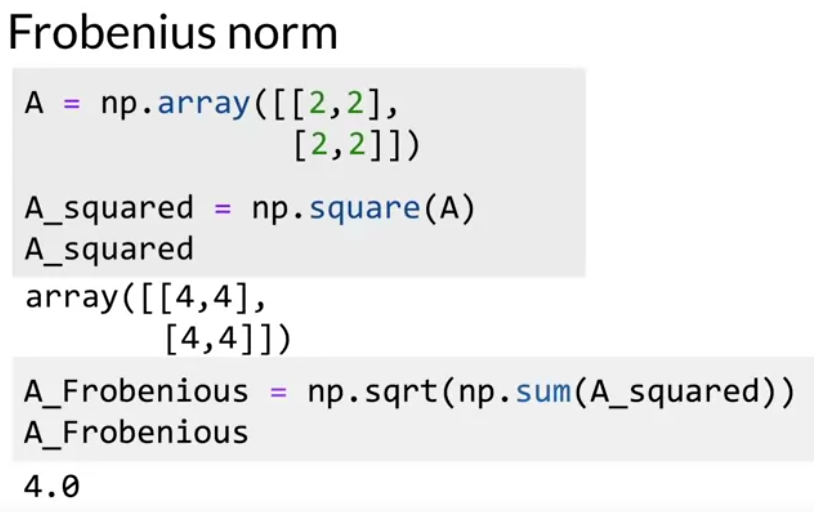
Which is the number of rows in the matrix and the word embeddings have two dimensions. So, that`s the number of columns in the matrix.



If the matrix A looks like this, then to calculate its norm.,



Here`s the actual formula , you just take all the elements in the matrix, square them, and add them up.



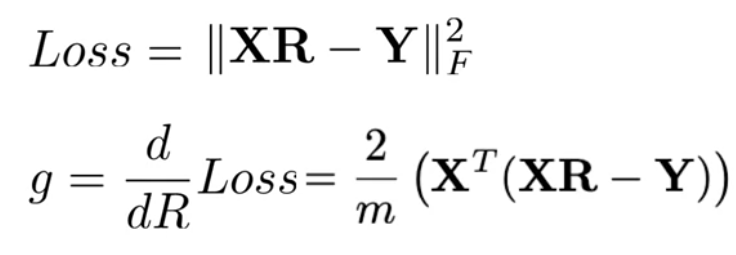
In practice, it`s easier to minimize the square of the Forbnius norm.

If we go back our example with matrix A, the square root of forbenius norm.

How to calcualte the gradient of the loss funciton.

1. The loss is defined as the square of the forbenius norm.
2. The Gradient is the derviative of the loss with respect to the matrix R.

if it looks like this, the scaler m is the number of rows or words in the subset that we are using for training.



If you remember from calcuals, this may look familer to if you if you pretend that R is a single variable instead of a matrix and X and Y are constants.

Y, it helps to use the square of the frobenius norm.

Rather than dealing with te square root that`s in the frobenius norm.

Transform

There are three main vector transformations:

\* Scaling

\* Translation

\* Rotation

The rotation operation changes the direction of a vector, letting unaffected its dimensionality and its norm. Let us explain with some examples.

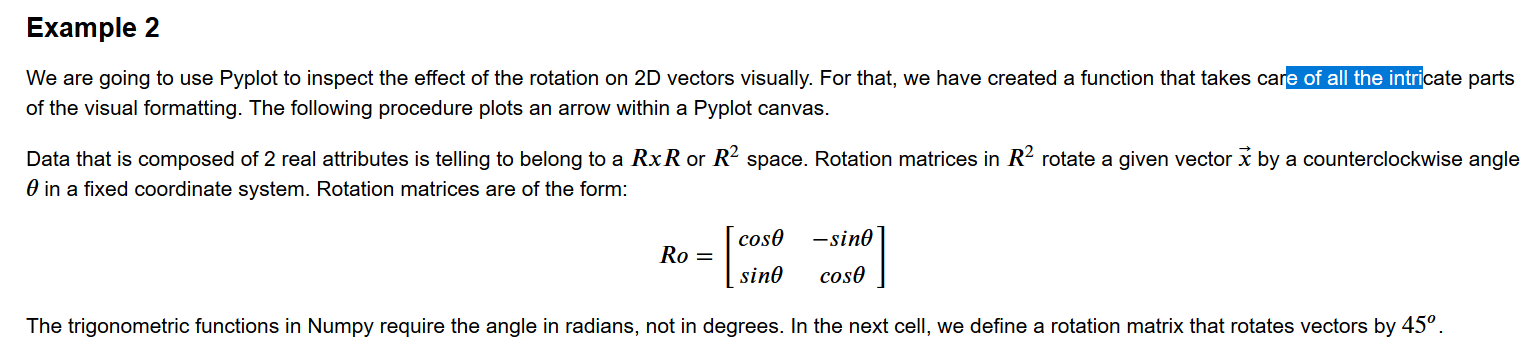
The dot product between a vector and a square matrix produces a rotation and a scaling of the original vector.

We are going to use Pyplot to inspect the effect of the rotation on 2D vectors visually.

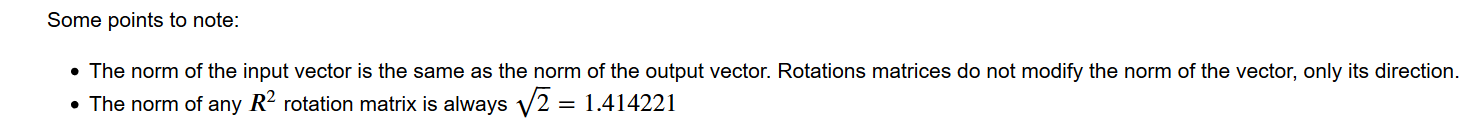
For that, we have created a function `plot\_vectors()` that takes care of all the intricate parts of the visual formatting. The code for this function is inside the `utils\_nb.py` file.

intricate parts اجزاء معقدة :

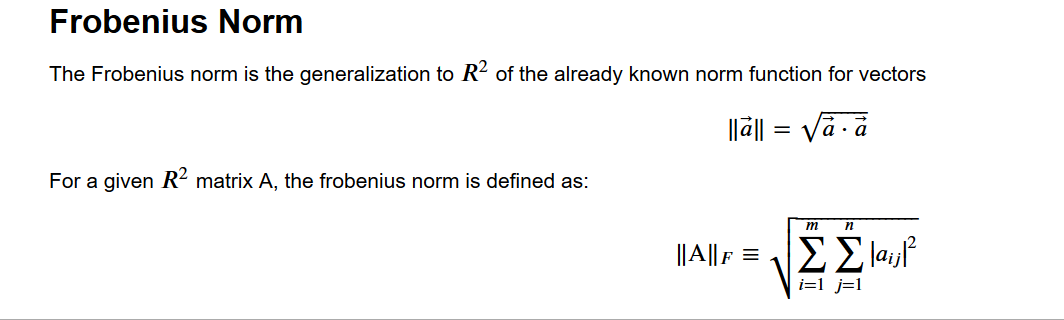
Now we can plot the vector $\vec x = [1, 1]$ in a cartesian plane. The cartesian plane will be centered at `[0,0]` and its x and y limits will be between `[-4, +4]`



Rotation matrix



Equation



`np.square()` is a way to square each element of a matrix. It must be equivalent to use the \* operator in Numpy arrays.

K-nearst neighbors :

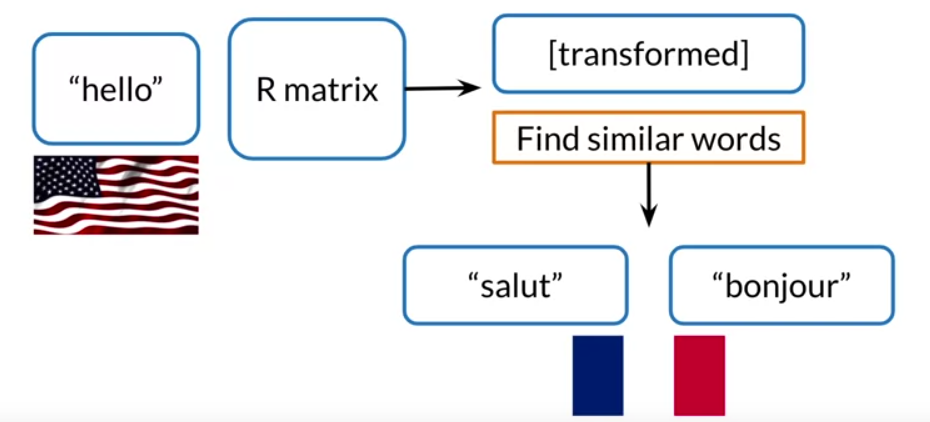
One key operation needed to find a matching word. Was finding the k-nearst neighbours of a vector.

It`s basic building block for many NLP techniques.

Notic: that it transformed word vector after the transformation it`s embedding through an R matrix would be in the French word vector space.

But it is not going to be necessarily identical to any of word vectors the french word vecotr space.

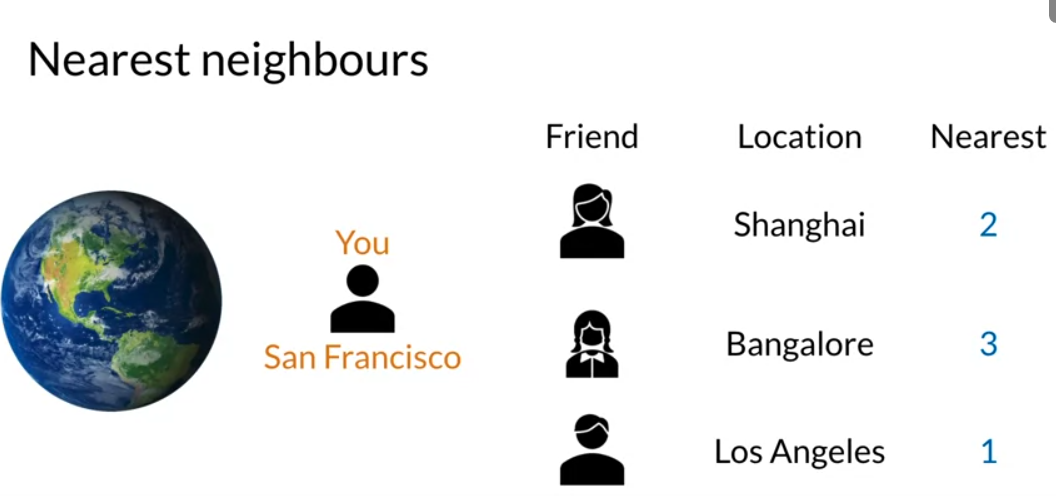
You need to search through the actual French word vectors to find a French word that is similar to the one that you created from the transformaiton.



You may find words such as sault or bonjour which can return as the french translation of the word hello.

* How to find similar word vectors?

Example : how do you find your friends who are living nearby ?



Note: if you have a lot of friends, which i`m sure you do this is very time intensive process.

You have efficient way to do that ?

Note that two of these friends live in another continet

Could you have just searched for a subset of friends who live in the united states.

When you think about organzing subsets of a dataset effiently, you may think about placing your data into buckets.

Hash tables are useful tools for any kind of work involving data,

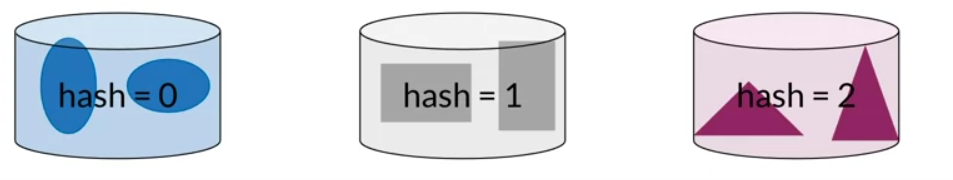
Yes .Correct, if you search in a subset of your entire list of friends there is not guarantee that you will  be able to determine  who are the absolute closest to you.

k-nearst : you could translate a word even if it`s transormation dosen`t exactly match the word embedding in the desired language.

Hash table : a usaful data structure

Let us say you have serveral data items and you want to group them into buckets by some kind of similartiy.

1. One bucket can hold more than one item and each item is always assigned to the same bucket.



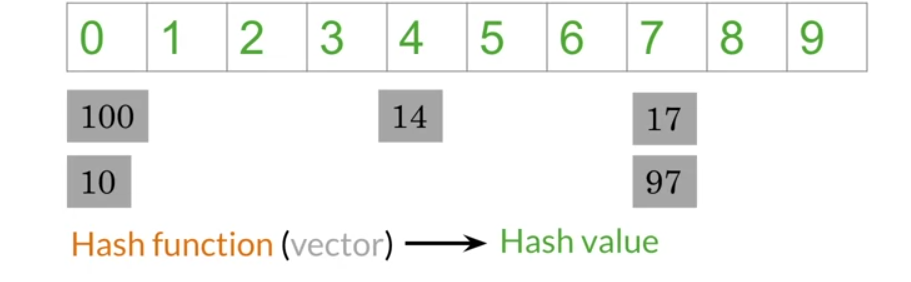
Let`s think about how we would like to do this with word vectors

1. Let`s assume that the word vectors have just 1 dim instead the 300 dim.
2. Each word is represented by a single number such as 100,14,17,10 and 97.
3. You need to find a way to give each vector a hash value which is a key that tells it which bucket it`s assigned to.

“ A Function that assigns a hash value is called a hash function.”

Hash table which is a set of buckets.

In this case hash table has ten buckets



Notic: how the word vectors 100 and 10 are assigned to buckets 0

Hach vlaue = vector % number of buckets.

The reminder I the hash value that tells us the where the word vector should be stroed.

Here is a defination of a funciton that takes in a list of values.

You can think of each value as a one-dim vector.

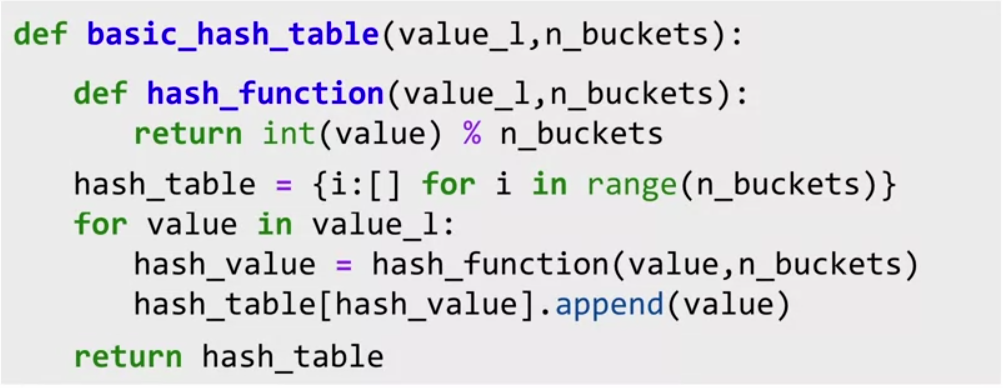
It also take in the number of pockets

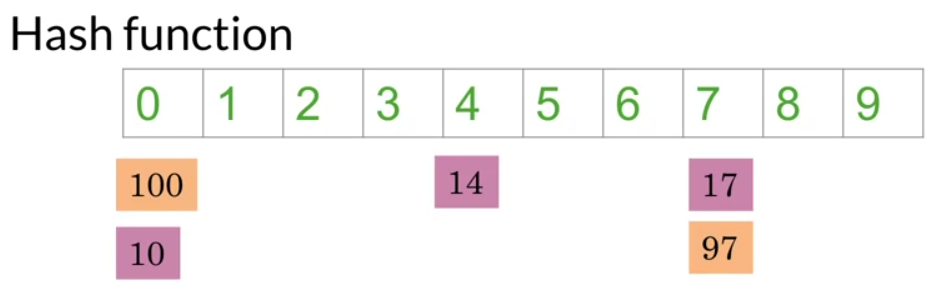
Define the hash function used in the moduler operator.

Then you create a hash table, notic that this I a dictonary comperhension.

The key is an integer and he value is an empty list, which you will use as bucket for storae

For each word vector, calcualte its hash value, that append it to the appropirate list.





Recall that your orginal goal was to put similar word vectors into the same bukets,but here it doesn`t look like numbers that are close to each other

Ideally, you want to have a hash function that puts similar word vectors in the same bucket like this.

To do that using “ locality sensitve hashing” locality is another word for location, senstive is another word for caring

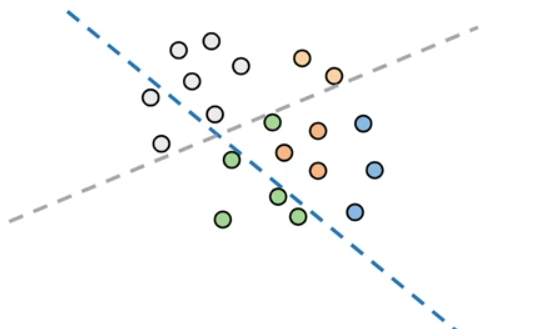
**So locality sensitive hashing is a hashing method that`s cares very deeply about assigning items based on where they are located in vectors space.**

That you will use key method reduce the computational cost of finding k-nearest neighbour in high dim spaces is locality -senstive hashing.

Let`s assume that you are using word vectors with just 2-dim

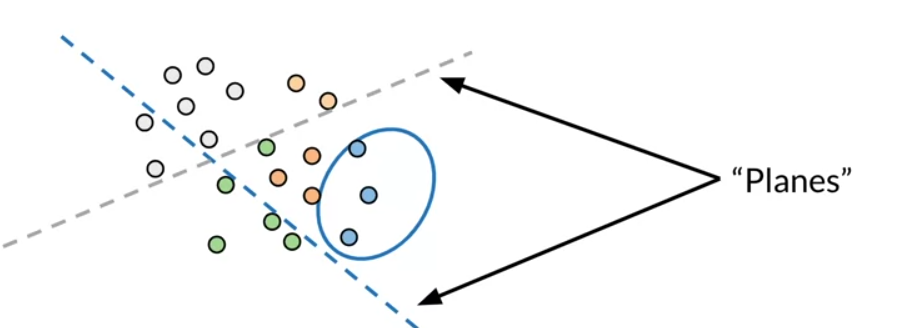
I will decipt each vector as a circle instead of arrows.

Let`s say you want to find a way to know that these blue dots are somehow close to each other, and these gray dots are also related to each other.



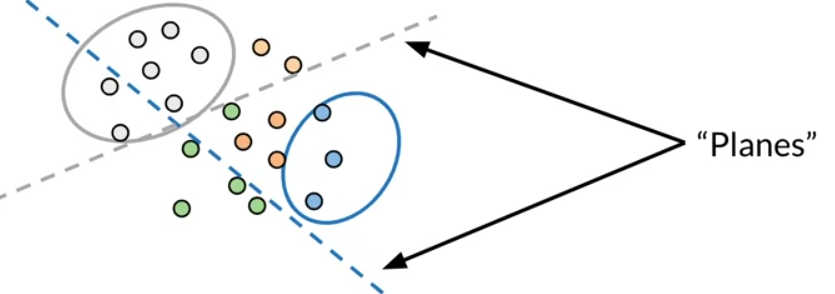
1. First, divide space using these dashed lines,which I will call planes

Notice how the blue plane slices up the space into vectors that are above it or below it.



The blue vectors all happen to be on the same side of the blue plane.

Similarly, the gray vectors happen to be above the gray plane.



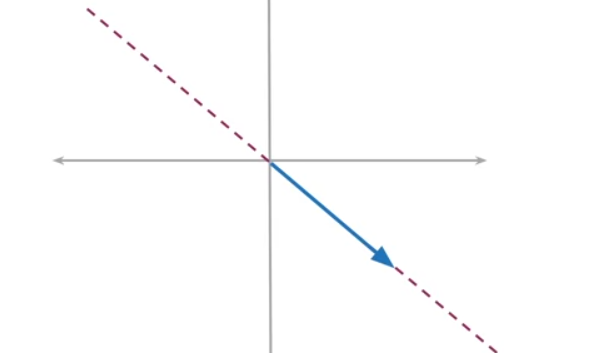
It looks like the planes can help us bucket the vectors into subsets based on their location. This is exactly what you want.

A hashing function that is sensitive to the location of the items that it`s assigning into buckets.

Now, let`s see why i`m calling these dashed lines planes.

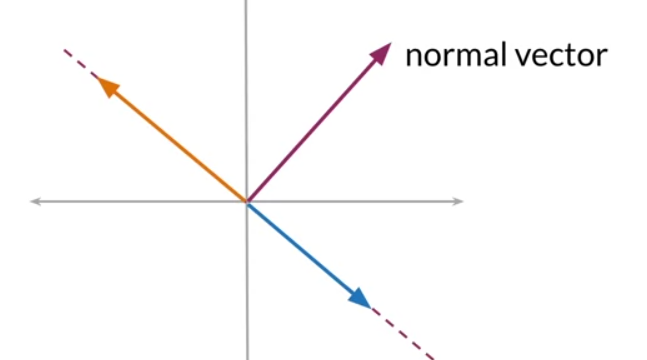
A plane would be this magenta line into 2-dim space, and actually represents all the possible vectors that would be sitting on that plane.

In other words, they would be parallel to the plane, such as blue vector or this orange vector.



Or the orange vector you can define a plane with a single vector.

This magenta vector is prependicular to the plane, and it`s called the normal vector to that plane.



The normal vector is prependicular to any vectors that lie on the plane.

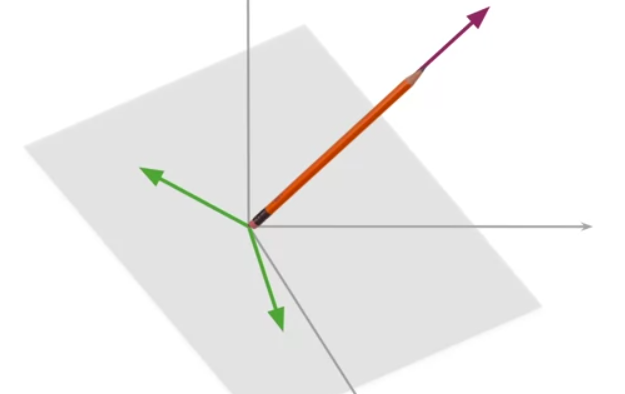
It might help to think about this in 3-dim

Find a sheet of paper and find a pencil.

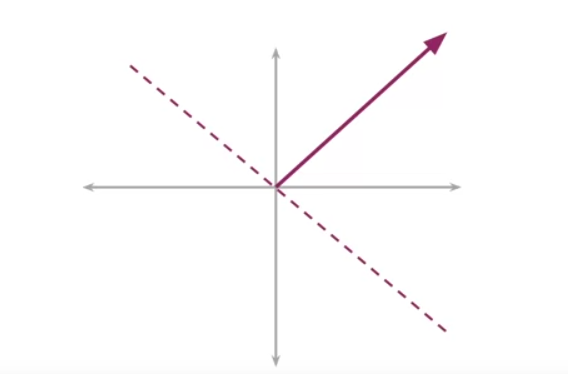


On the table and draw some vectors on it,then hold the pencil vertically over the paper.

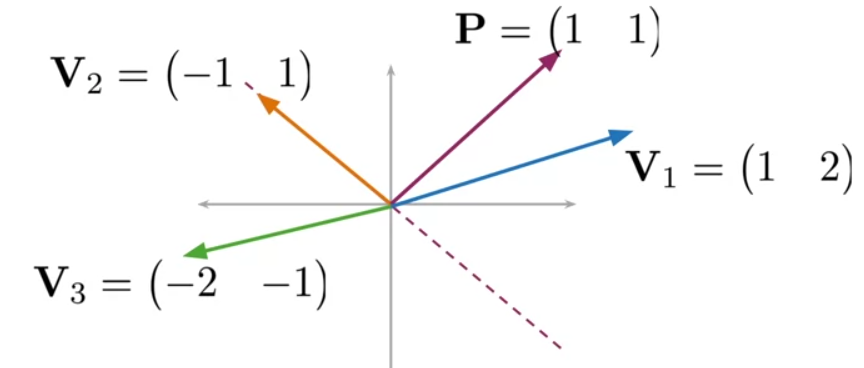
Any vector on the paper are prependicular to the pencil.



The vector is on one side of the plane or the other, but how do you do this mathmetically ?

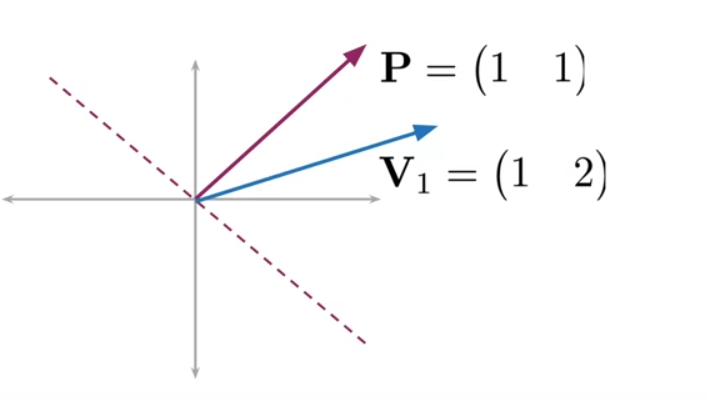


Here, the sample vectors in blue, orange and green.



The normal vector to the plane is labeled P.

Let`s focus on v1.

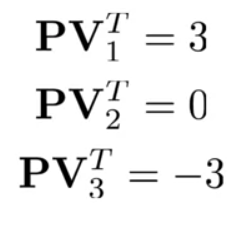


What if you take dot product of P with v1. You got 3.

Now let`s look at the v2.

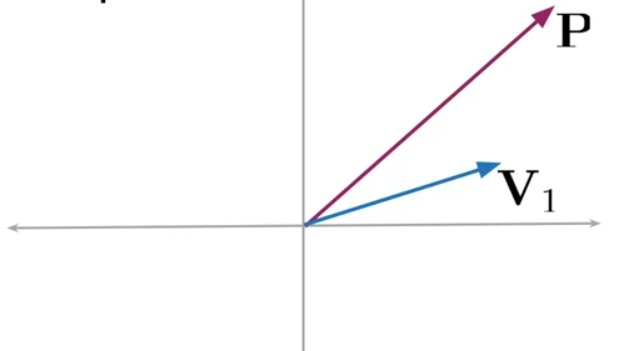
If you take dot product of P with v2 you get 0

Finally, let`s look at v3. If you get dot prodcut of p with v3 you get the -3.



Do you know notice something about the signs and how they related to their position relative to the red plane.

1. When is the dot prodcut os positive, the vector is on one side of the plane.
2. If the dot product is negative, the vector is on the opposite side of the plane.
3. If the dot product is zero, the vecotr is on the plane.

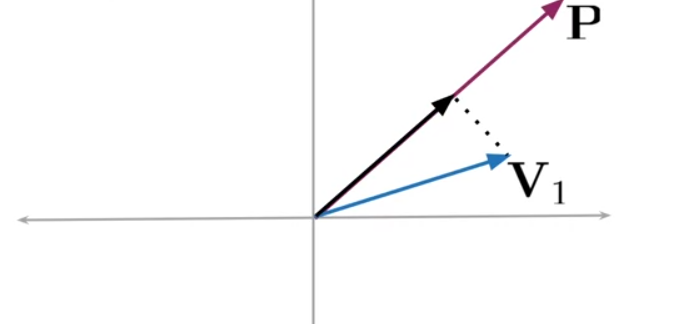


To visualize dot product, imagine one of the vectors such as P, as if it`s surface of the earth

Gravity pulls all objects straight down towards the surface of the earth.

Next, pretend you are standing at the end of the vector you tie a string to a rock and let gravity pull the rock to the surface of vector P.

The string is prependicular to vector P.



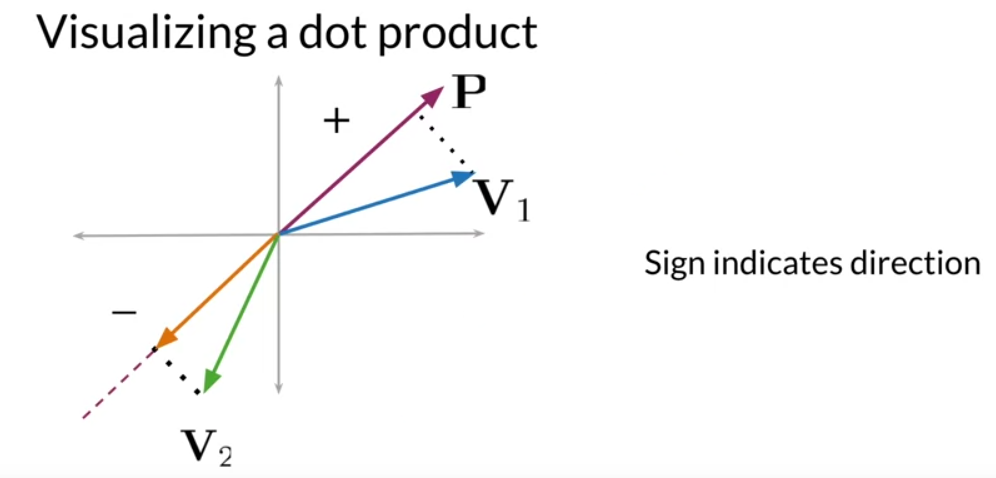
Now, if draw a vector that`s in the same direction of P but ends up at the end the rock.

You will have the what`s called the projection of v1 onto vector P.

* The magntiude or length of that vector is equal the dot product of v1 and P.

If you had this other green vector and projected it onto vector P, the prohected vector would be pointing in the parallel but opposite direction of P

The dot product would be negative number.



This means that the sign of the dot product indicates the direction of projection with repsect of the purple normal vector.

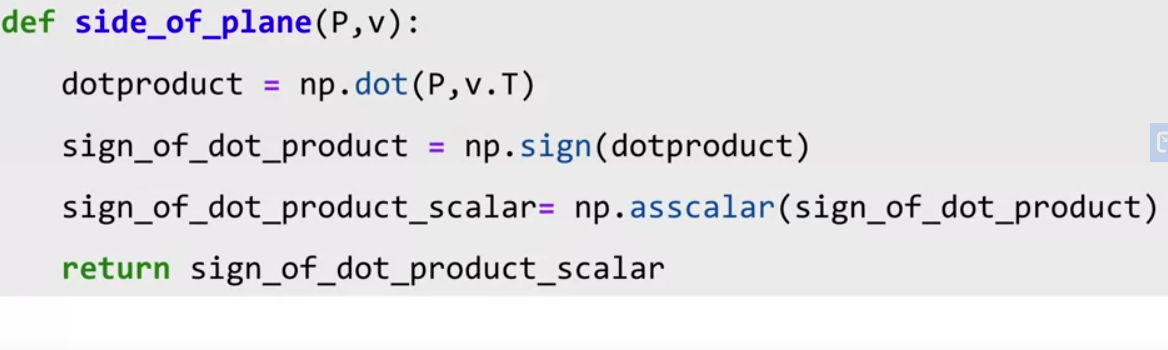
The dot product is + or – can tell you whether the v1 or v2 are on one side of the plane or the other.

Which side of the plane.

1. The fnciton side of plane takes in the normal vector P, and the vector v.
2. Using numpy dot to take dot products.
3. Using numpy.sign to get a +1 if the dot produt is positive,
4. -1 if the dot product is negative.
5. 0 if the dot prdouct is zero.
6. I`m using numpy.asscalar.

Notice: the pronunciation of the funtion.

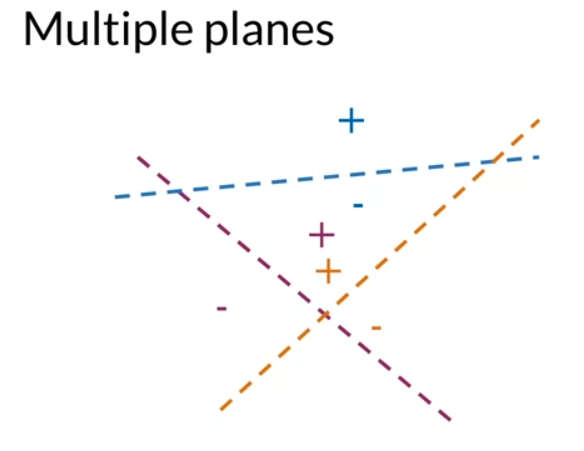
If the vector can represented as a single scalar, this function retrives the scalar and that`s it.



Muliple plane.

How to **use information for multiple** to get a hash value for your data in your vector space.

In order to divide your vector space into manangeable regions, you will want to use more than one plane.



For each plane, can find out whether a vector is on the + or – side of the plane.

So, you will get muliple signals, one for each plane and you want to find a way to combine all of those signals into a single hash value.

This has value will define a particular region within the vector space.

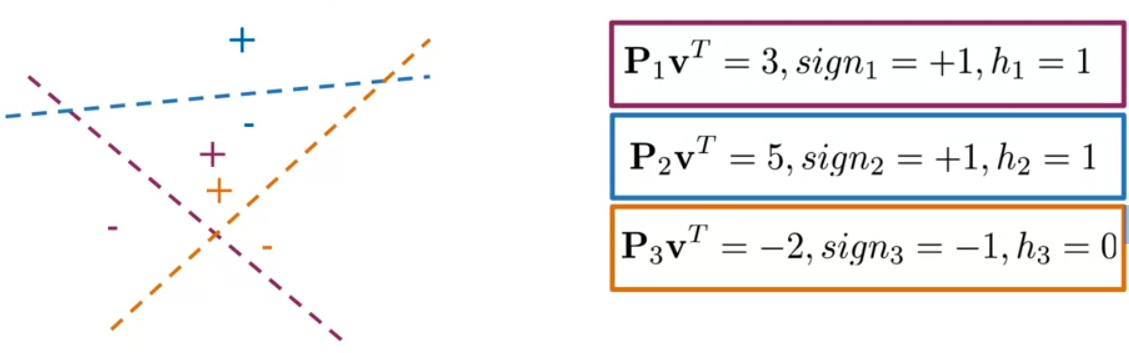
Example

Will see the general formula for combing signals from multiple plaens.

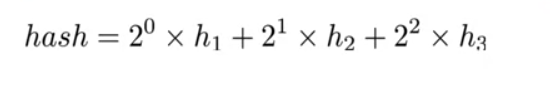
For a signle vector, let`s say that it`s dot product with plane 1 is 3, so the sign is positive, and the hash value is set to 1 to indicate that the sign is positive.

For second plane, the dot product is 5, so the sign is again is positive and the hash value is 1.

For thrid plane, the dot product is -2, so the sign is -1 and the hash value is set to 0 to indicate that the vector v is on the negative side of plane 3.



To combine these intermediate hash values into a single hash value, you will do the following



So just a reminder you hasve muliple planes and it helps us to divide the vector space into smaller sub regions.

But you want have a single hash value so you will know which bucket to assign the vectoring.

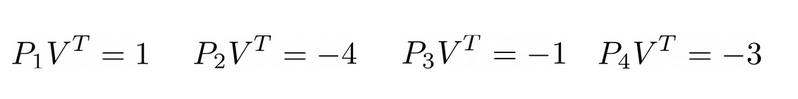
You do this by combing the signals from all the planes into a single hash value.

Here, the rules you applied written out if he sign of the dot product is greaher than or equal to 0, assign the intermediate has value of 1.

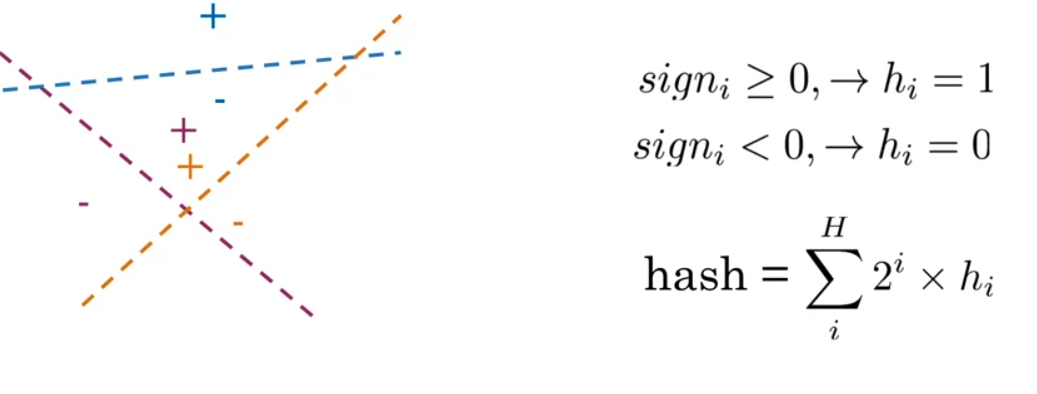
Otherwise if the dot prodcut is less than 0,assign the intermediate hash value of 0.

To combine the intermediate as value use this formula, this how you get locality sensitve hashing.

Given the following dot products between a vector and four different planes, compute the vector’s hash value.

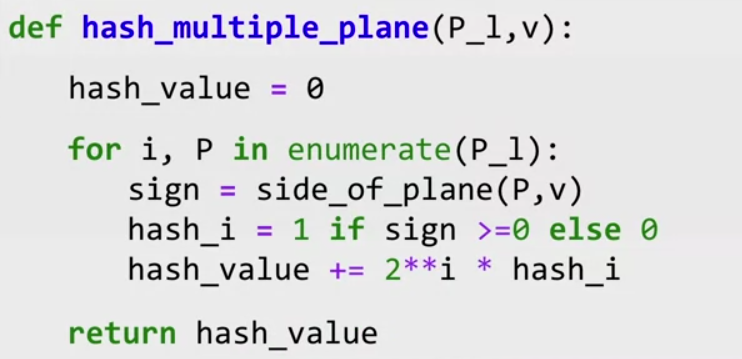


=1



Psudocode

1. Given a list of plane and vector starts with a hash\_value of 0, which you will use to accumulate the sum of intermediate hash values.
2. Then for each plane, you want to calculate the sign of the dot product.
3. Set the intermediate hash\_value to 1 if the sign is greater than or equal 0.
4. Else I will set to 0 then muliply the intermidate add value by 2 power and added to hash\_value.
5. Finally, return the hash.

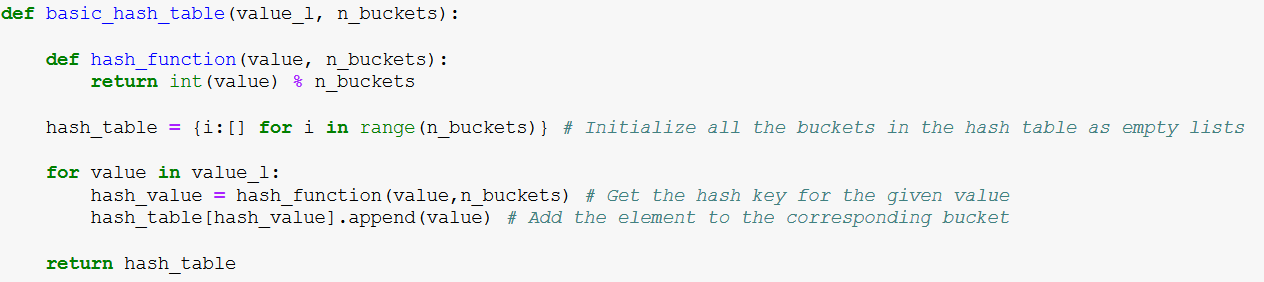


A key point for the lookup using hash functions is the calculation of the hash key or bucket id that we assign for a given entry. In this notebook, we will cover:

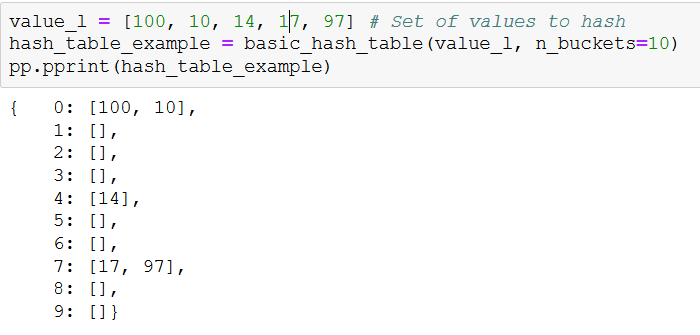
* Basic hash tables
* Multiplanes
* Random planes

In the next cell, we will define a straightforward hash function for integer numbers. The function will receive a list of integer numbers and the desired amount of buckets. The function will produce a hash table stored as a dictionary, where keys contain the hash keys, and the values will provide the hashed elements of the input list.

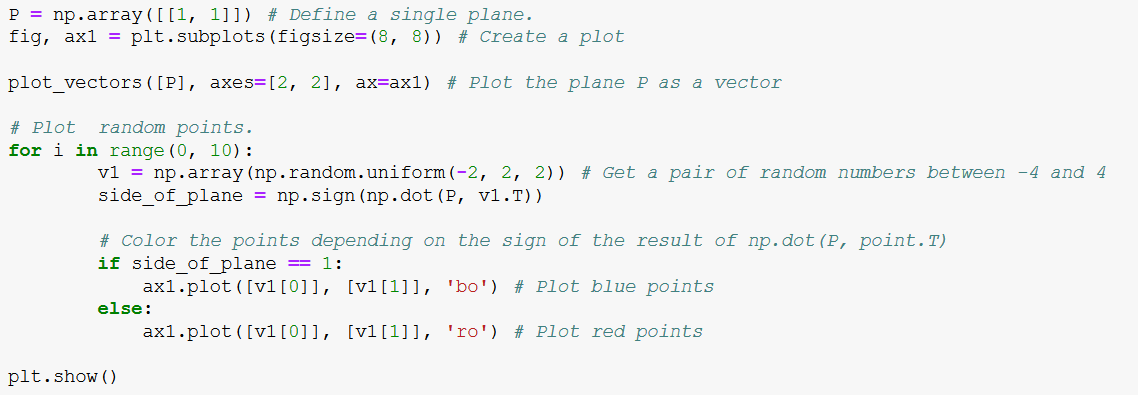
The hash function is just the remainder of the integer division between each element and the desired number of buckets.



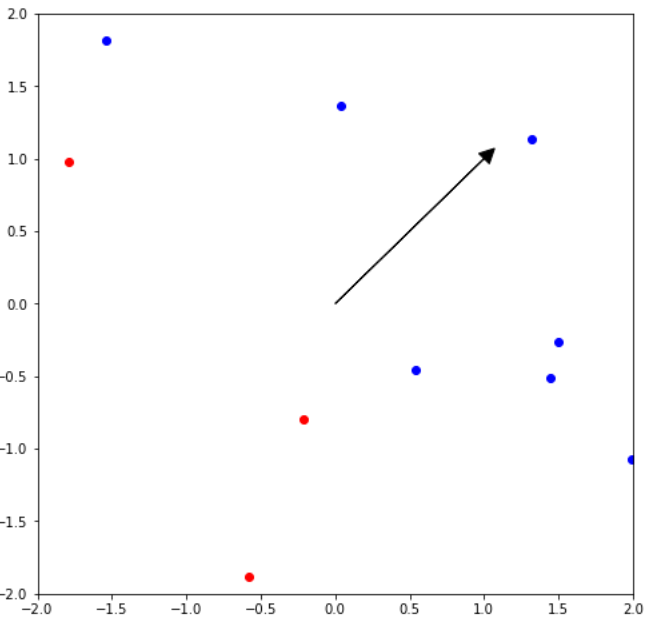
Print function



Multiplanes hash functions **are other types of hash functions**. Multiplanes hash functions are based on the idea of numbering every single region that is formed by the intersection of n planes. In the following code, we show the most basic forms of the multiplanes principle. First, with a single plane:



Ploting



The first thing to note is that the vector that defines the plane does not mark the boundary between the two sides of the plane. It marks the direction in which you find the 'positive' side of the plane. Not intuitive at all!

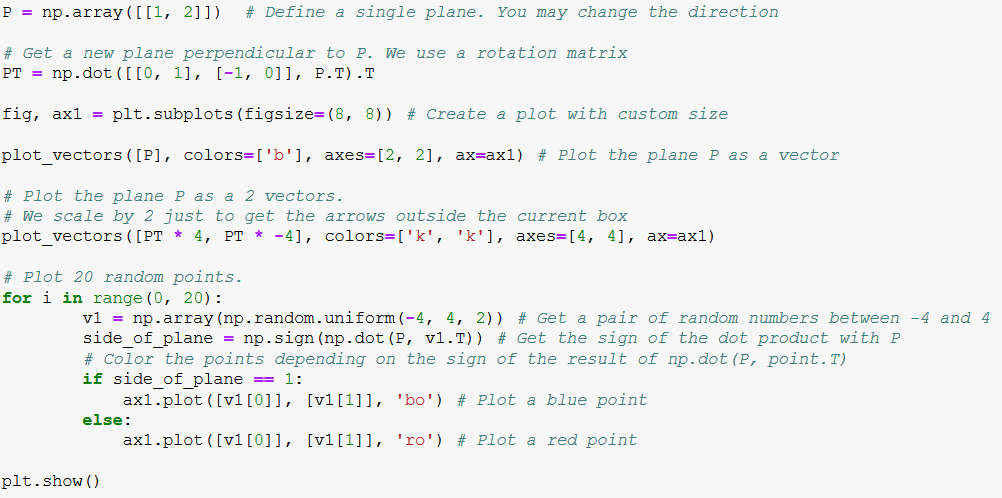
-----------------------

If we want to plot the separation plane, we need to plot a line that is perpendicular to our vector `P`. We can get such a line using a $90^o$ rotation matrix.

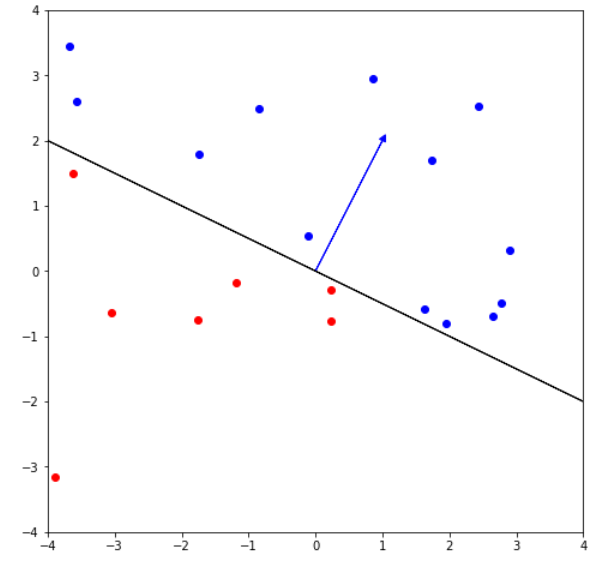
-----------------------

Feel free to change the direction of the plane `P`.

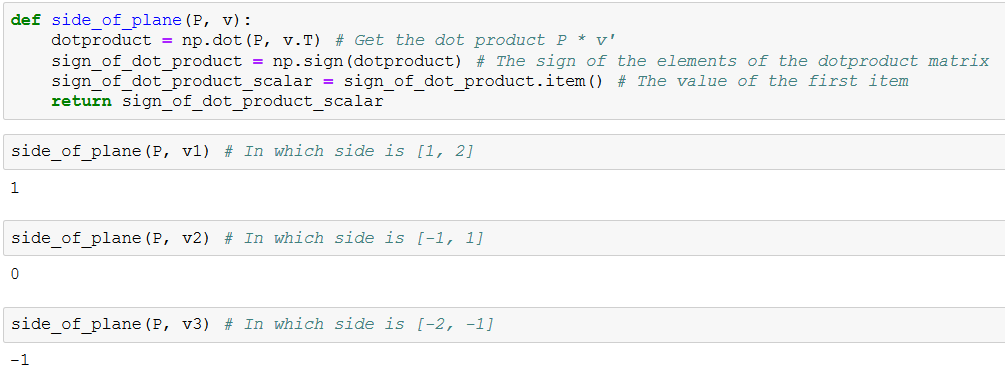
Cac..

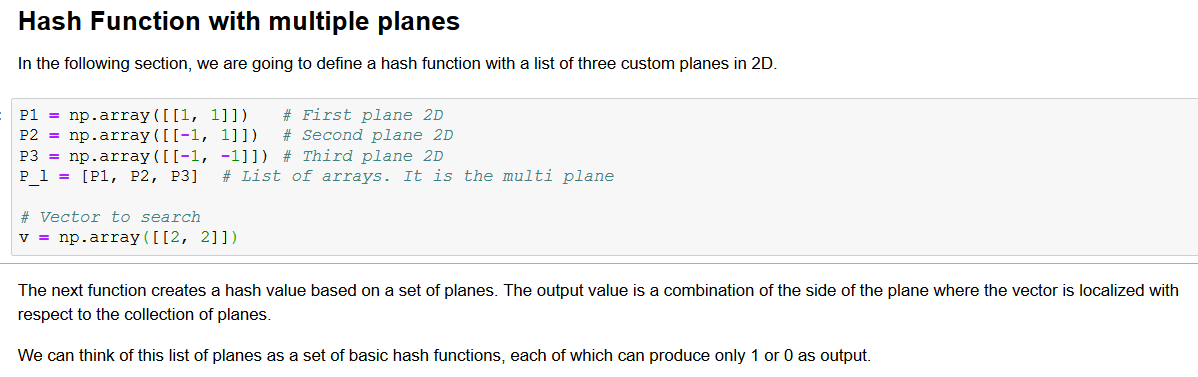


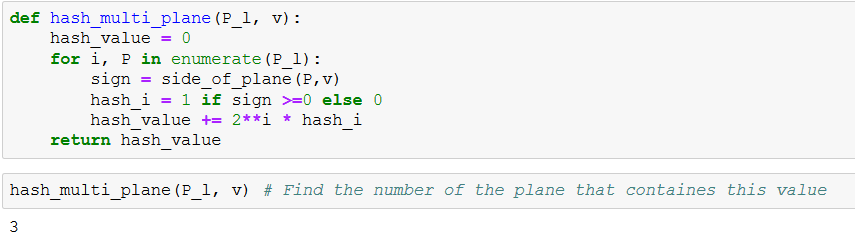
Ploting



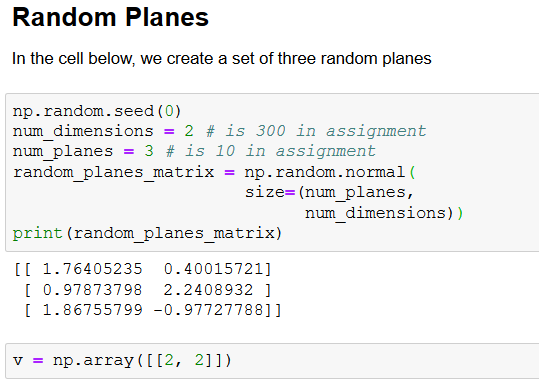
The function below checks in which side of the plane P is located the vector `v`



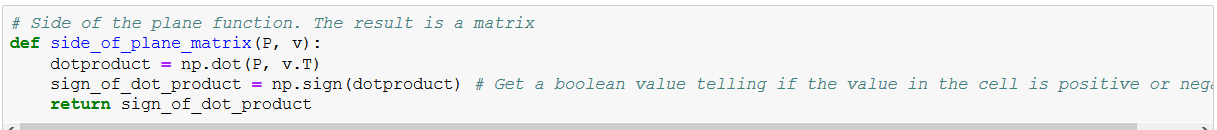


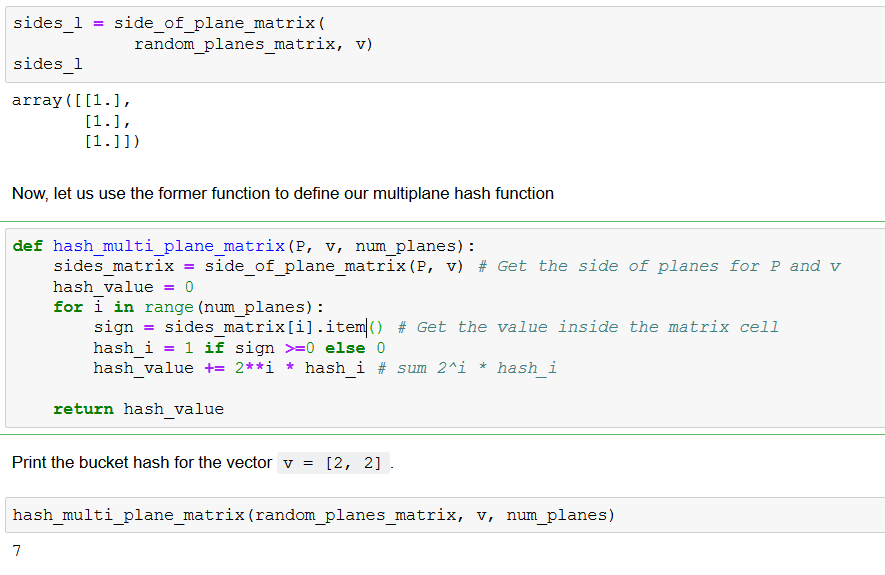


Random



The next function is similar to the `side\_of\_plane()` function, but it evaluates more than a plane each time. The result is an array with the side of the plane of `v`, for the set of planes `P`





Document vectors

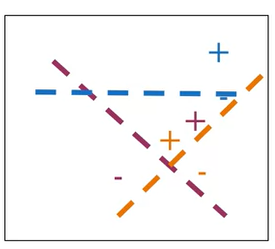
Before we finish this lab, remember that you can represent a document as a vector by adding up the word vectors for the words inside the document. In this example, our embedding contains only three words, each represented by a 3D array.



Approximate nearest neightbours:

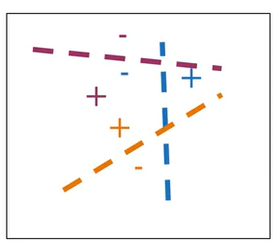
Localility-senstivty hashing : will make algorthim k-nearest neightbours much faster than brute search.

Random planes such as these three



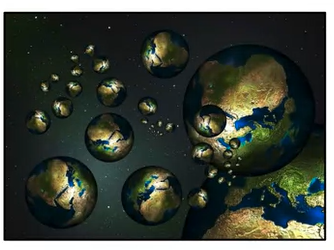
You can divide the vector space into regions : these planes the best way to divide up the vector space.

What if insteaded, you divided the vector space like this?



In fact, you can`t know for sure which sets of planes is the best way to divide up the vector space, so why not create muliple sets random planes so that you can divide up the vector space into muliple, independent sets of hash tables.

You can think that create muliple copies of the universe, or a multiverse , if you will.



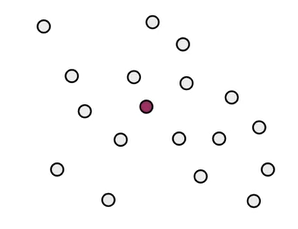
You can make of all differnet sets of random planes in order yo help us find a good set of friendely neighbourhood vectors,

I mean a set of k-nearest neighbours.

Muliple sets of random planes:

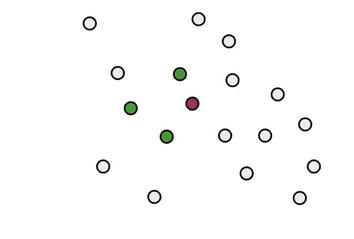
Back to our muliple sets of random planes.

**Let`s say you have a vector space, and this magneta dot in the middle represents the transformation of english word into a French word vecotr.**

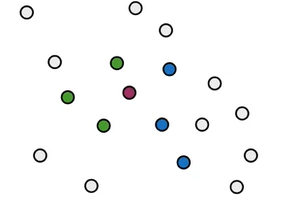


you are trying to find other french word vectors that may be smiliar.

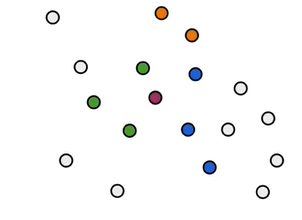
So maybe one universe of random planes helped us to determine that this megenda vector and these green vectors are all assigned to same hash bucket.



Another entirely differnet set of random planes helped us determine that these blue vectors are in the same hash bucket as the red vector.



A third set of random planes helped us determine that these orange vectors are in the same hash bucket as the magenda vector.



By using muliple sets of random planes for localilty-sensitive hashing, you hae more robust way of searching the vector space for a set of vectors that are possible candiates to nearest neightbours This is called

Approximate nearest neighbours because you are not searching the entire vector space, but just a subset of it.

So it`s not the absolute k-nearest neightbours, but the approxmiately the k-nearest neightbours.

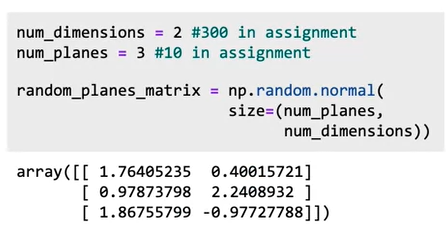
you scarifice some percision in orde to gain effciency in you serach.

**Make one set of random planes**

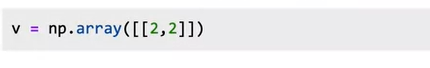
Random planes in code assuming that you word vectors have 2-dims

And you want to generate three random planes.

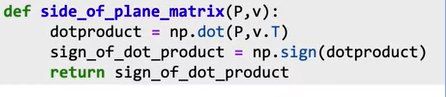
You will use numpy.random.normal to generate a matrix of three rows and two 2 col.



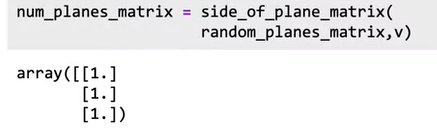
1. You will ceate vector v, and for each random plane, see which side of the plane the vector is on.



1. So you will find out whether the vector v is on the positive or negative side of each of these three planes.



1. Notice that instead of using a for loop to work on one plane at a time you can use numpy.dot to do this in one step
2. The result is vector v is on the positive side of each of the three random planes.



1. How to combine these intermediate hash values into a single hash value, but please , do check out the lecture notebook to see all the code

Localitiy-senstive hashing allows to compute k-nearest much faster than naïve search.

This powerful tool can be used for many tasks related to our vectors.

Seaching Documents

How can use fast k-nearest neighbor to search for pieces of text related to a query in a large collection of documents.

1. You simply create vectors for both find the nearest neighbors.

To get ready to perform document search, first,

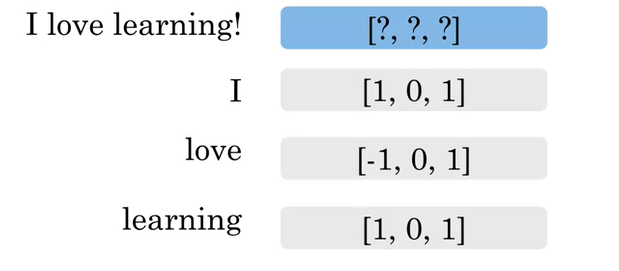
1. How represent documents as vectors instead of just words as vectors.

Let`s say you hav document composed of three words.

“ I love learning .”

How can represnet this entire documents as a vector?

1. You can find the word vectors for each indvidual word.

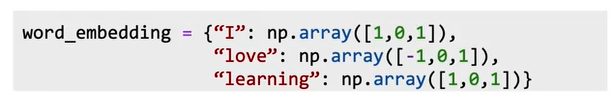


Then just add them together, so sum of these words vectors become a document vector, where the same dim as the word vectors.

In this case, 3-dims



1. You can apply document seach by using k-nearst neighbors.
2. Create a mini dictionay for word embeddings.



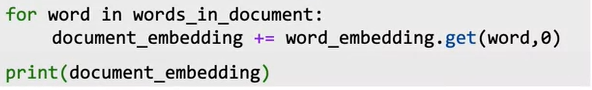
1. Here`s the list of what`s contained in the document.



1. You are going to initilaize the documens embedding as an array of zeros.



1. Now for each word in a document, you will get the word vector if the word exists in the dictionary els 0.
2. You add these all up and return the documents embedding.



This is example of a very general method that text can be embedded into vector space so that nearest neighbors refer to text with similar meaning.

Embed text in advanced way.

**Gensim** is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing ...

**import** numpy as np print("I like ", np.pi). For testing the **SciPy** library and Matplotlib, here's a fun Easter egg: from **scipy import** misc **import** matplotlib.pyplot as plt ...

**SciPy** is a library that **uses NumPy** for more mathematical functions. **SciPy uses NumPy** arrays as the basic data structure, and comes with modules for various commonly **used** tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving, and signal processing.

Resources

* English embeddings from Google code archive word2vec [look for GoogleNews-vectors-negative300.bin.gz](https://code.google.com/archive/p/word2vec/)
  + You'll need to unzip the file first.
* and the French embeddings from [cross\_lingual\_text\_classification](https://github.com/vjstark/crosslingual_text_classification).
  + in the terminal, type (in one line) curl -o ./wiki.multi.fr.vec https://dl.fbaipublicfiles.com/arrival/vectors/wiki.multi.fr.vec

from gensim.models import KeyedVectors

en\_embeddings = KeyedVectors.load\_word2vec\_format('./GoogleNews-vectors-negative300.bin', binary = True)

fr\_embeddings = KeyedVectors.load\_word2vec\_format('./wiki.multi.fr.vec')

# loading the english to french dictionaries

en\_fr\_train = get\_dict('en-fr.train.txt')

print('The length of the english to french training dictionary is', len(en\_fr\_train))

en\_fr\_test = get\_dict('en-fr.test.txt')

print('The length of the english to french test dictionary is', len(en\_fr\_train))

english\_set = set(en\_embeddings.vocab)

french\_set = set(fr\_embeddings.vocab)

en\_embeddings\_subset = {}

fr\_embeddings\_subset = {}

french\_words = set(en\_fr\_train.values())

for en\_word in en\_fr\_train.keys():

fr\_word = en\_fr\_train[en\_word]

if fr\_word in french\_set and en\_word in english\_set:

en\_embeddings\_subset[en\_word] = en\_embeddings[en\_word]

fr\_embeddings\_subset[fr\_word] = fr\_embeddings[fr\_word]

for en\_word in en\_fr\_test.keys():

fr\_word = en\_fr\_test[en\_word]

if fr\_word in french\_set and en\_word in english\_set:

en\_embeddings\_subset[en\_word] = en\_embeddings[en\_word]

fr\_embeddings\_subset[fr\_word] = fr\_embeddings[fr\_word]

pickle.dump( en\_embeddings\_subset, open( "en\_embeddings.p", "wb" ) )

pickle.dump( fr\_embeddings\_subset, open( "fr\_embeddings.p", "wb" ) )

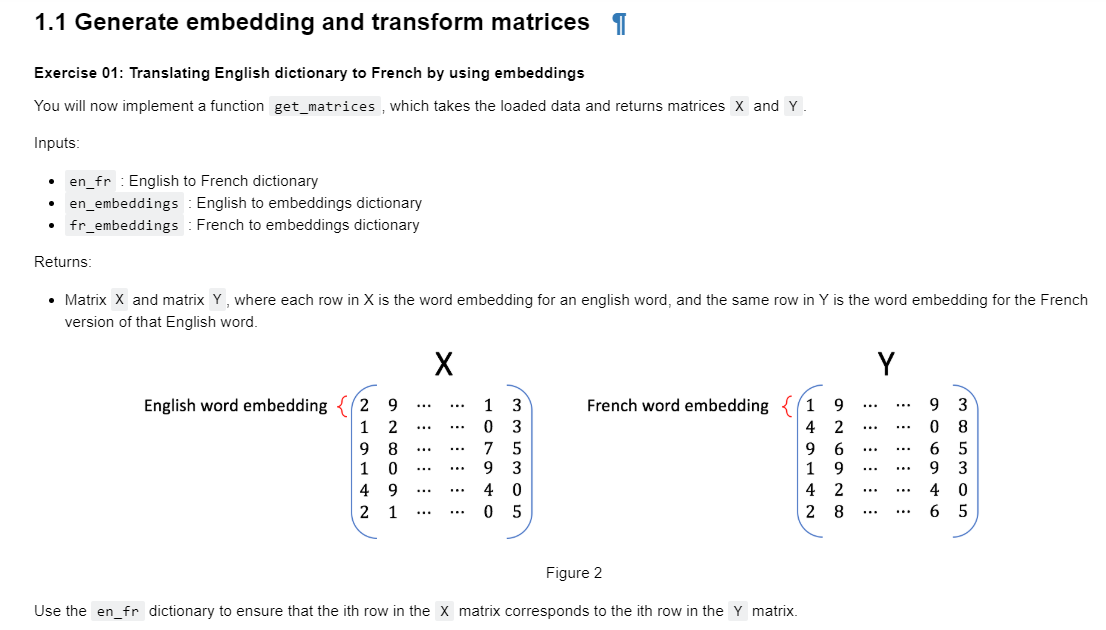
```

#### Looking at the English French dictionary

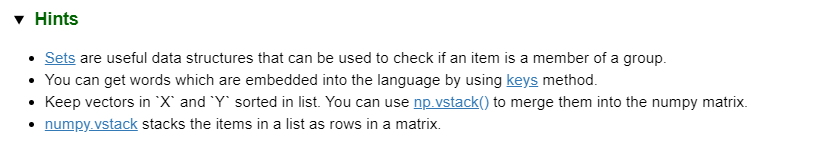
* **en\_fr\_train** : is a dictionary where the key is the English word and the value is the French translation of that English word.
* {'the': 'la',
* 'and': 'et',
* 'was': 'était',

'for': 'pour',

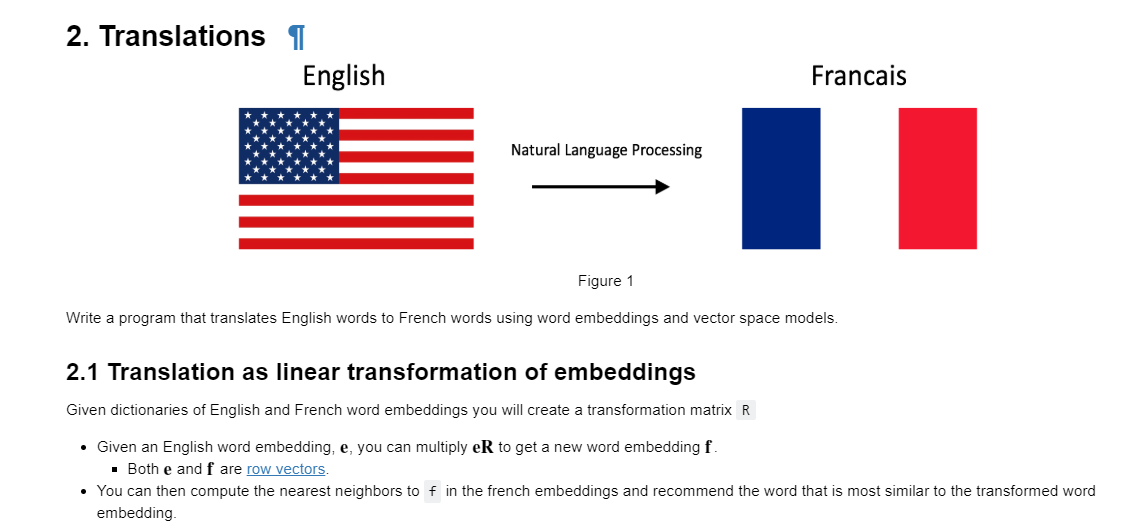
* en\_fr\_test is similar to en\_fr\_train, but is a test set. We won't look at it until we get to testing.



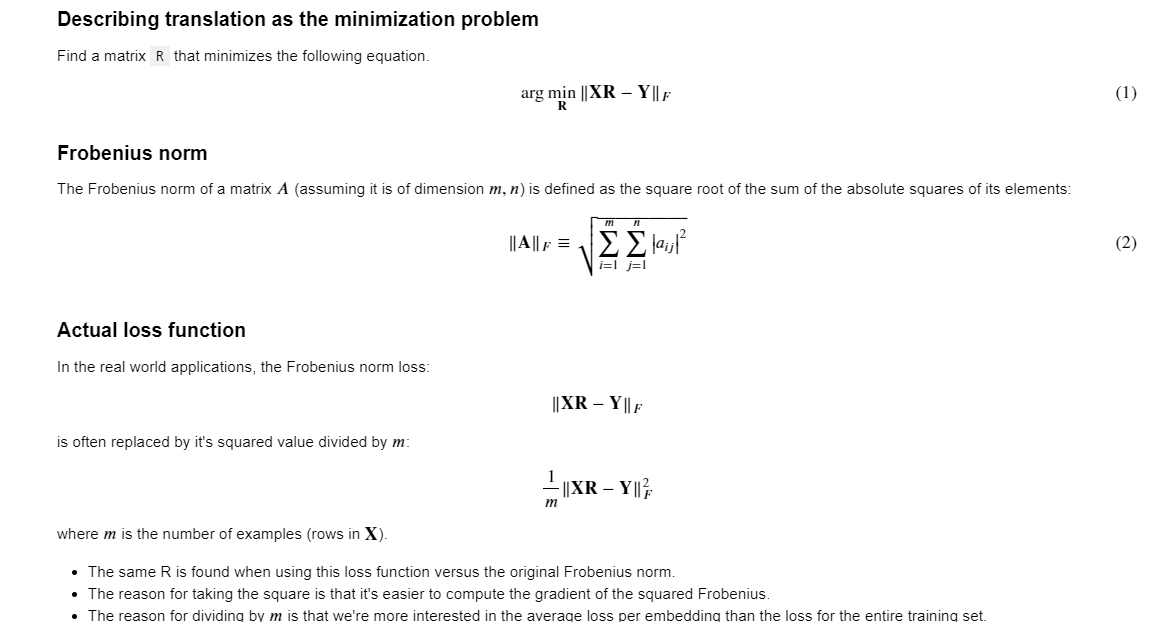
Note

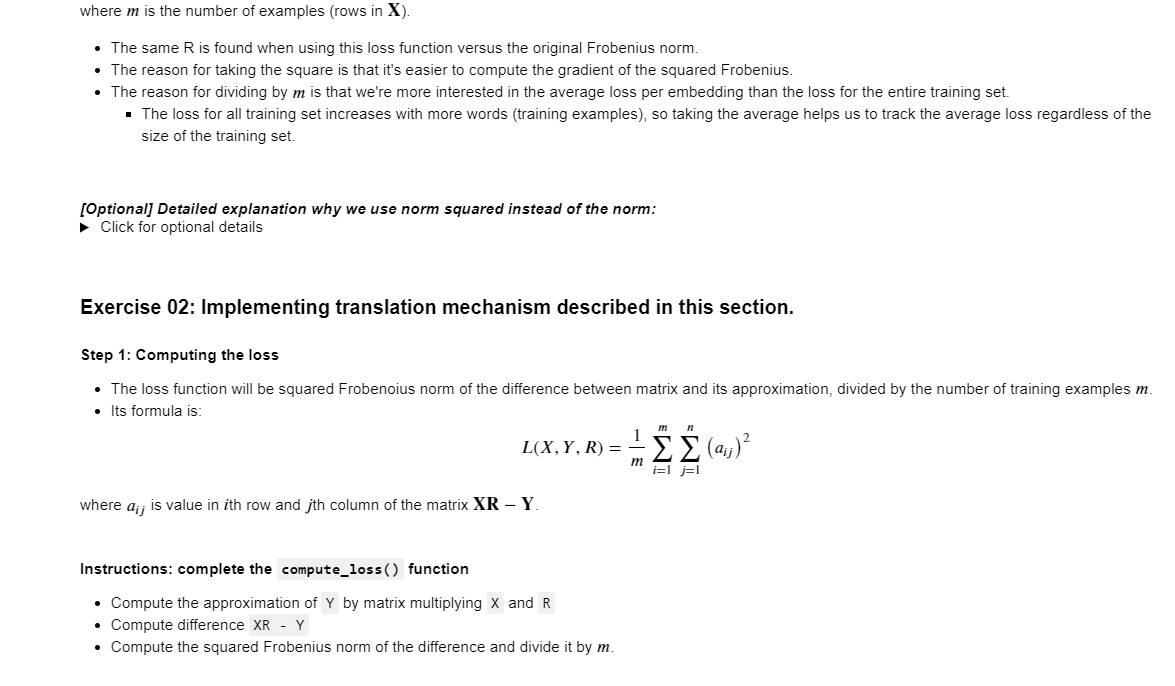


**Vector space model** or term **vector model** is an algebraic **model** for representing text documents (and any objects, in general) as **vectors** of identifiers (such as index terms). It is used in information filtering, information retrieval, indexing and relevancy rankings.

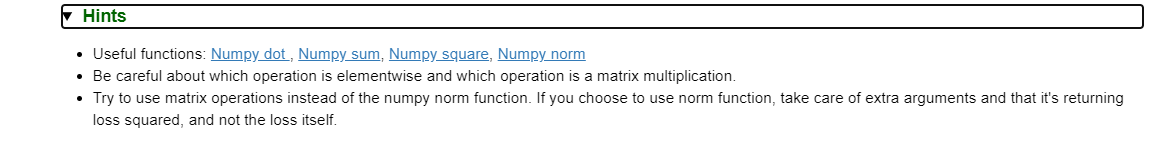


Describing matrix



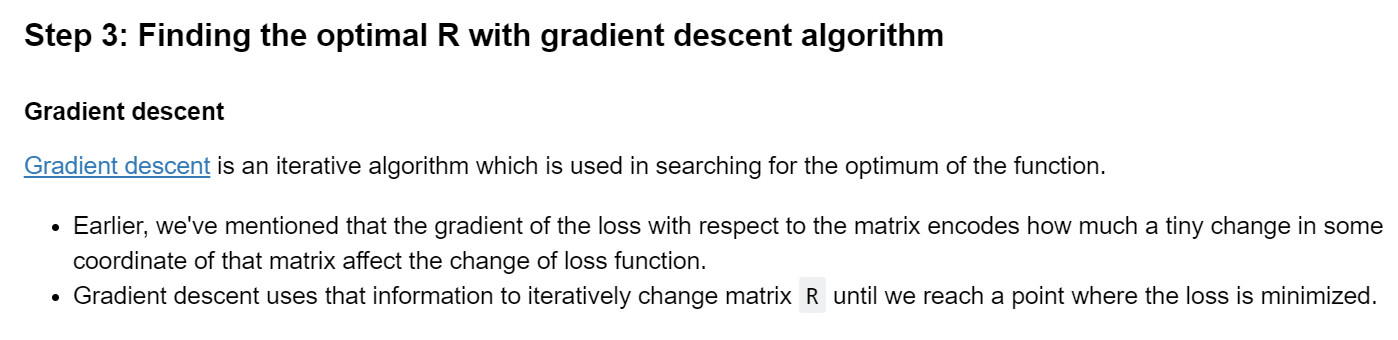


Note

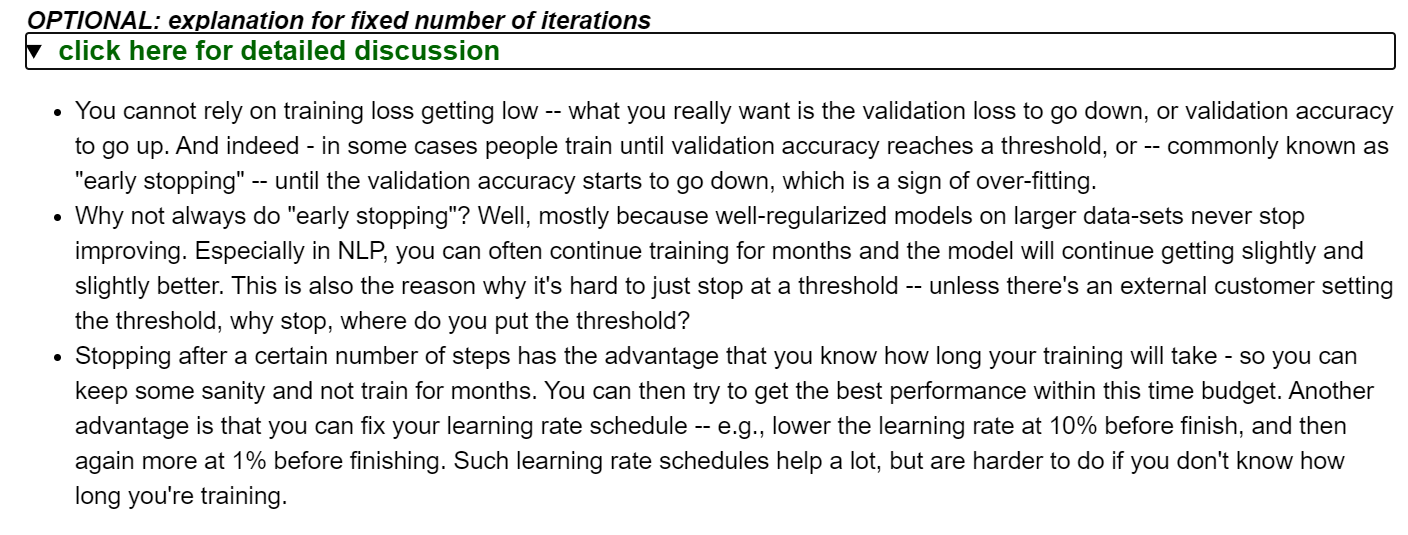


##### *Detailed explanation why we use norm squared instead of the norm*

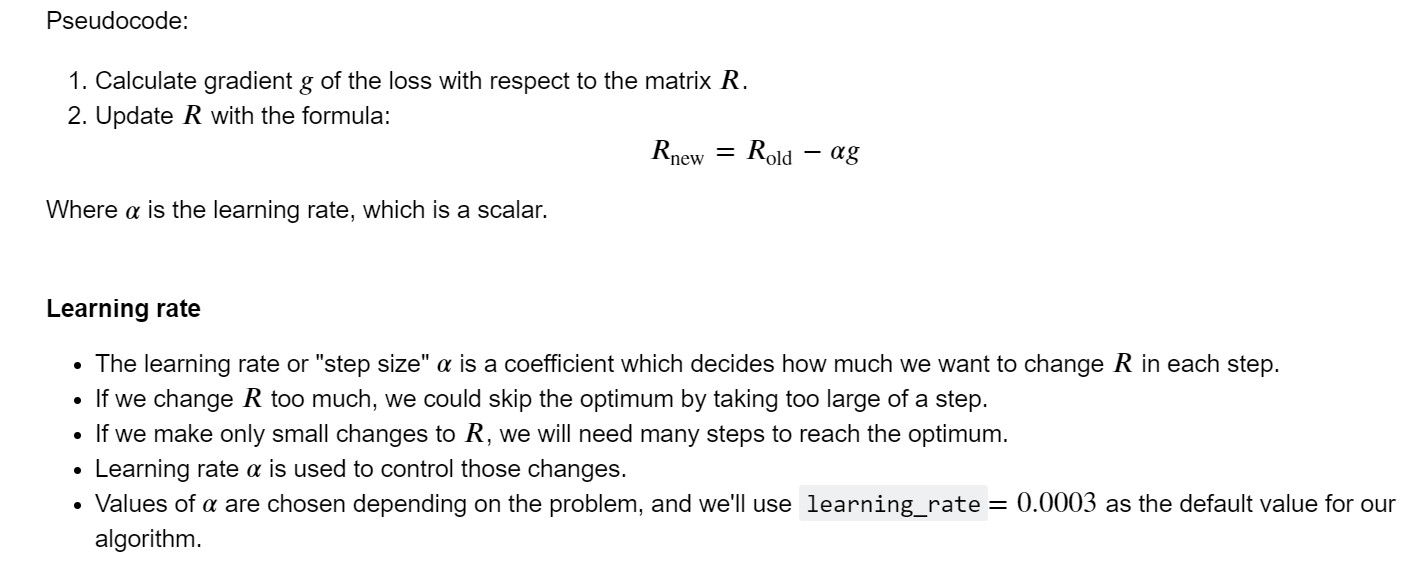
* The norm is always nonnegative (we're summing up absolute values), and so is the square.
* When we take the square of all non-negative (positive or zero) numbers, the order of the data is preserved.
* For example, if 3 > 2, 3^2 > 2^2
* Using the norm or squared norm in gradient descent results in the same *location* of the minimum.
* Squaring cancels the square root in the Frobenius norm formula. Because of the [chain rule](https://en.wikipedia.org/wiki/Chain_rule), we would have to do more calculations if we had a square root in our expression for summation.
* Dividing the function value by the positive number doesn't change the optimum of the function, for the same reason as described above.
* We're interested in transforming English embedding into the French. Thus, it is more important to measure average loss per embedding than the loss for the entire dictionary (which increases as the number of words in the dictionary increases).



Note:



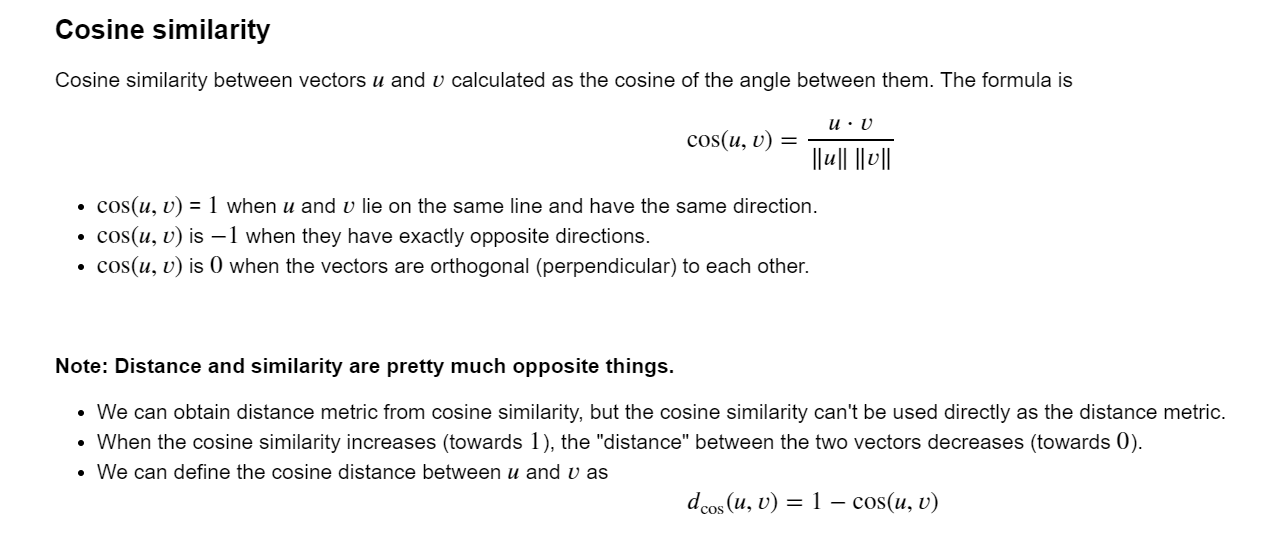
Get more details



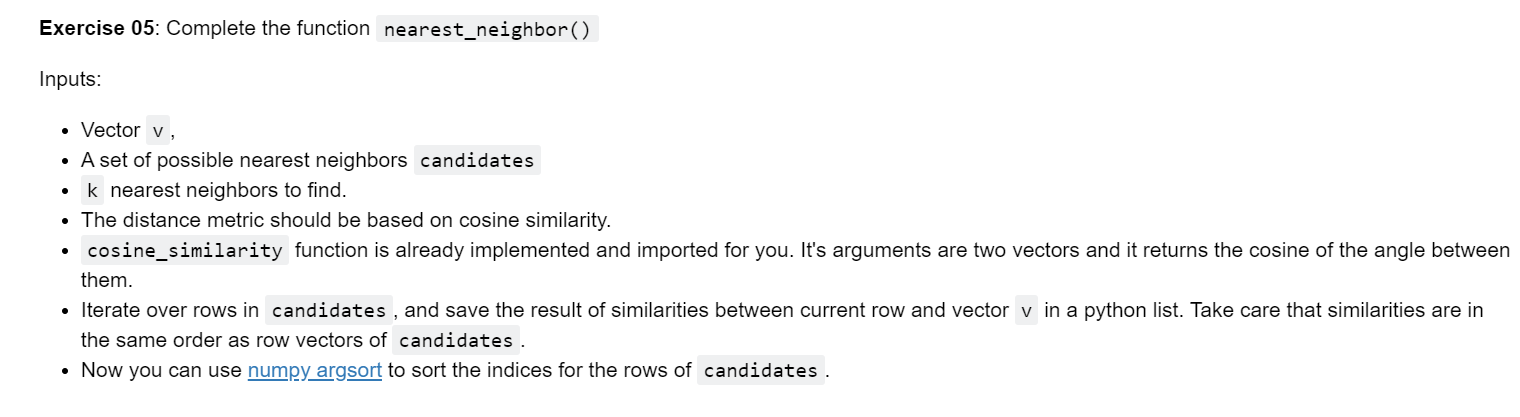
Testing Data



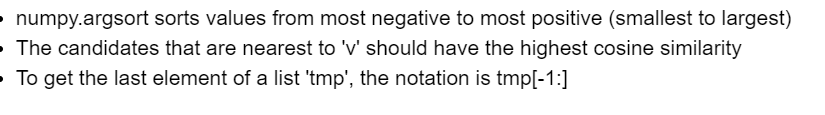
Cosine similarity

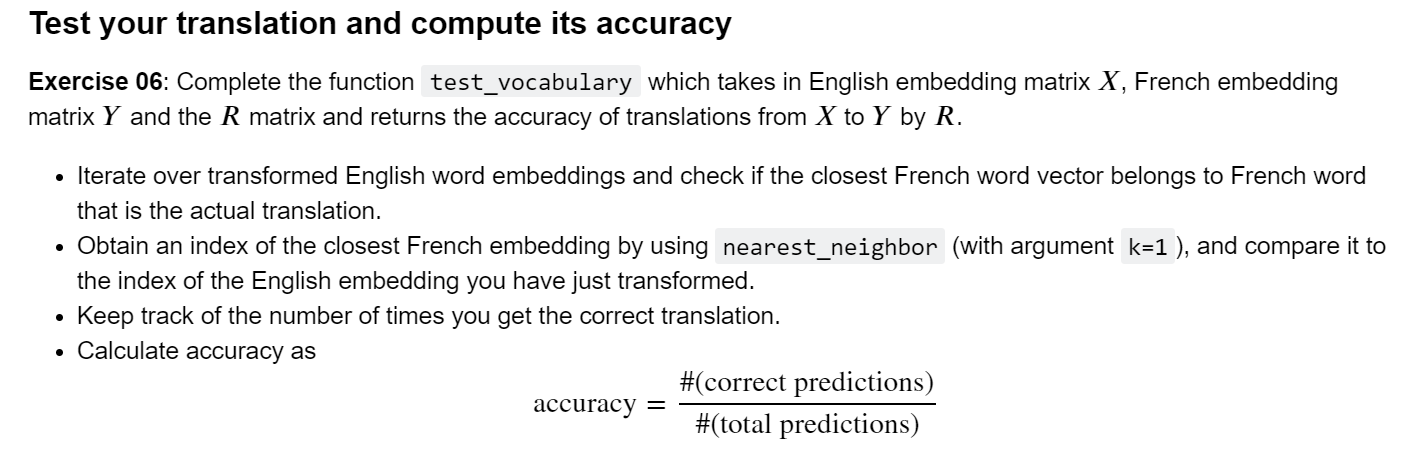


Ex



Hint:

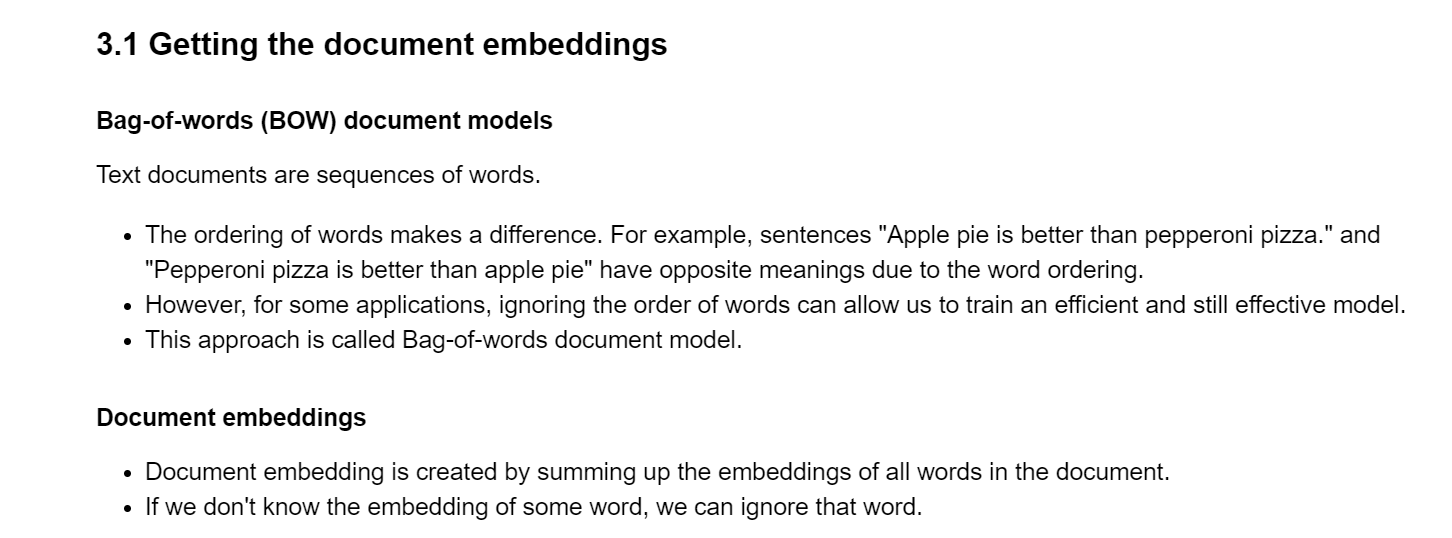




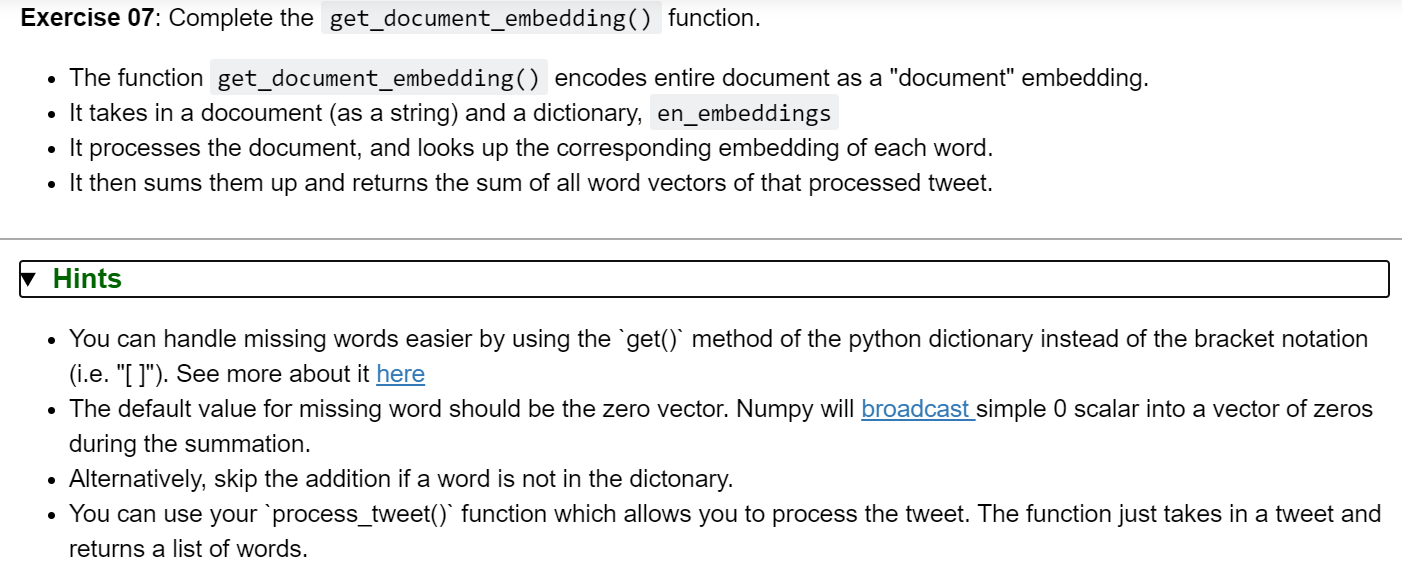
Using Locality sensitivity hash



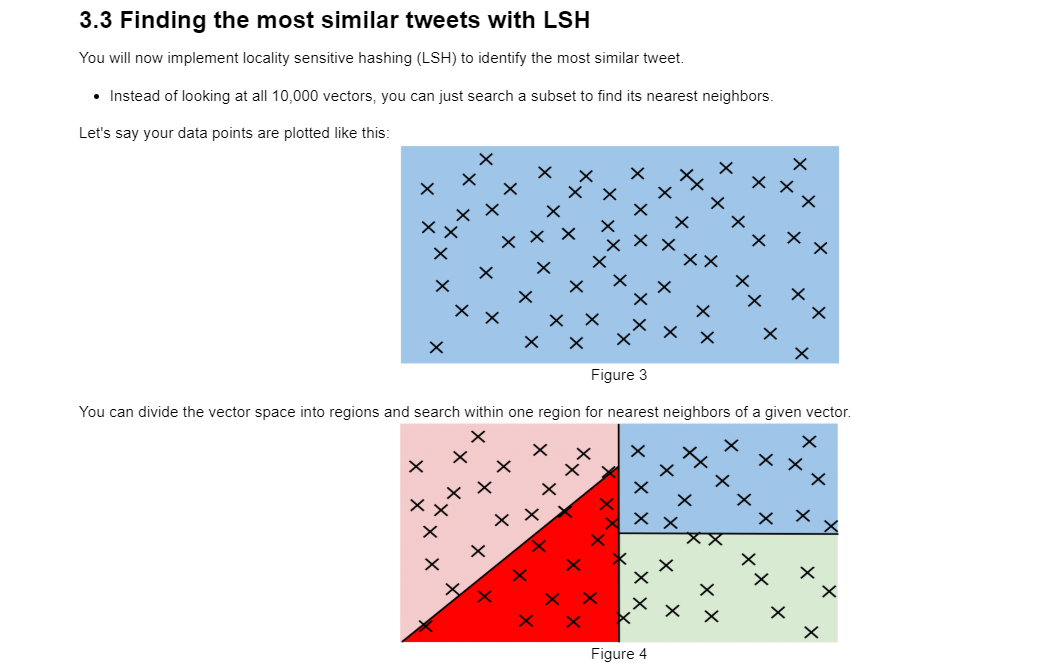
Getting



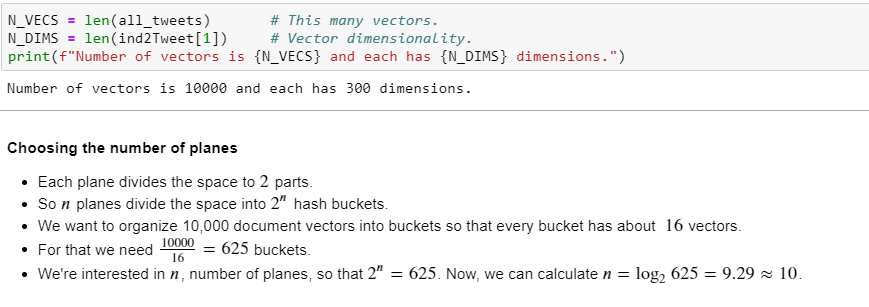
Ex

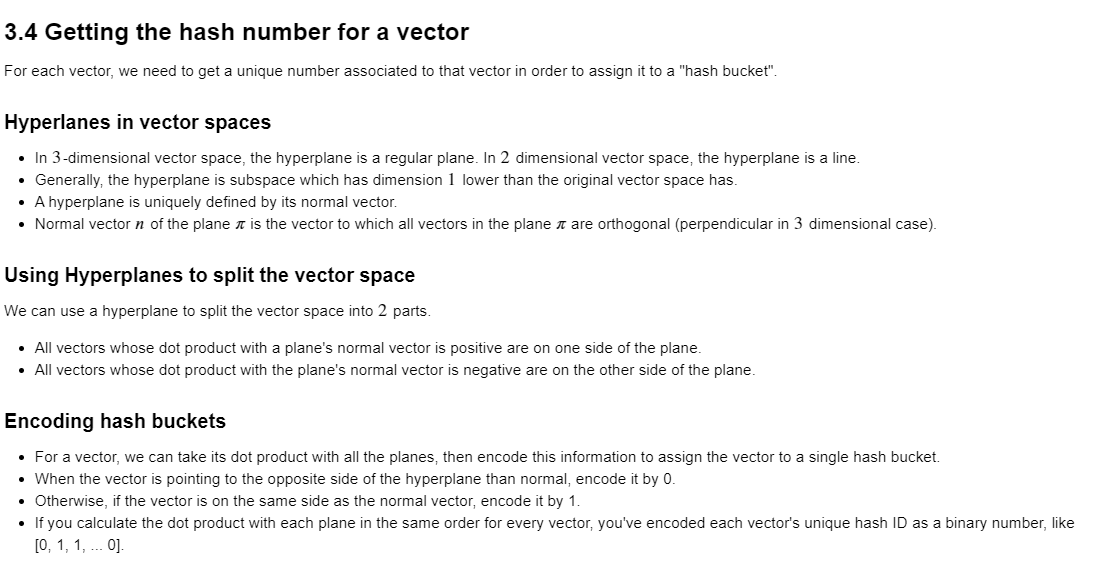


Implement



Divided vector into region





How to imple…

