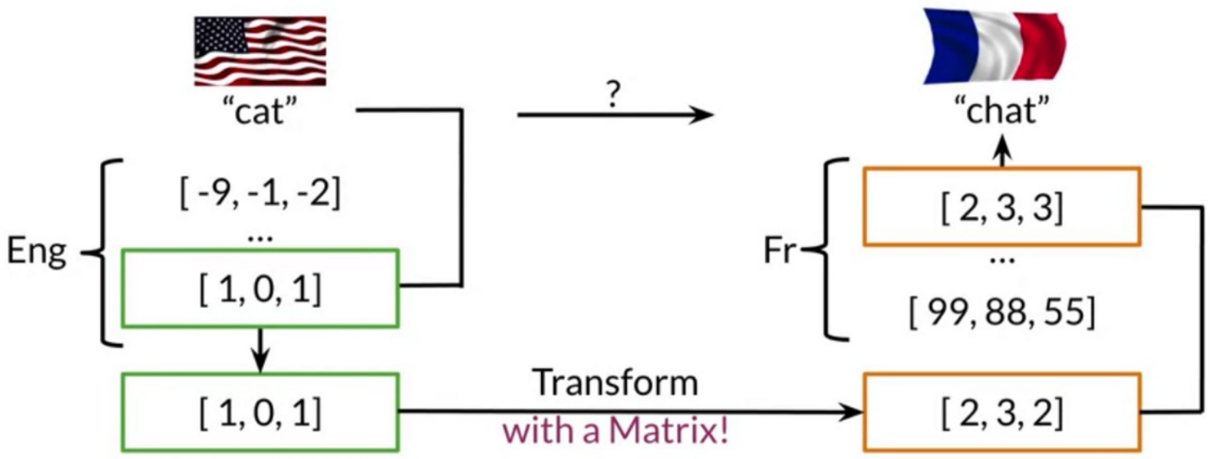
Natural Language Processing with Classification and Vector Spaces || Week — 04 (Machine Translation and Document Search)

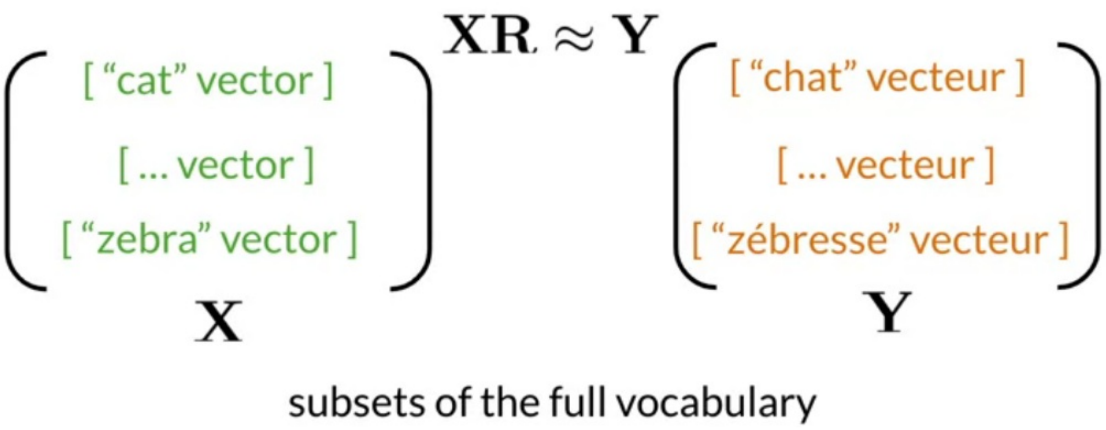
medium.com/@aakashgoel12/natural-language-processing-with-classification-and-vector-spaces-week-04-machine-39f5dcaf647c

July 26, 2020

Machine Translation

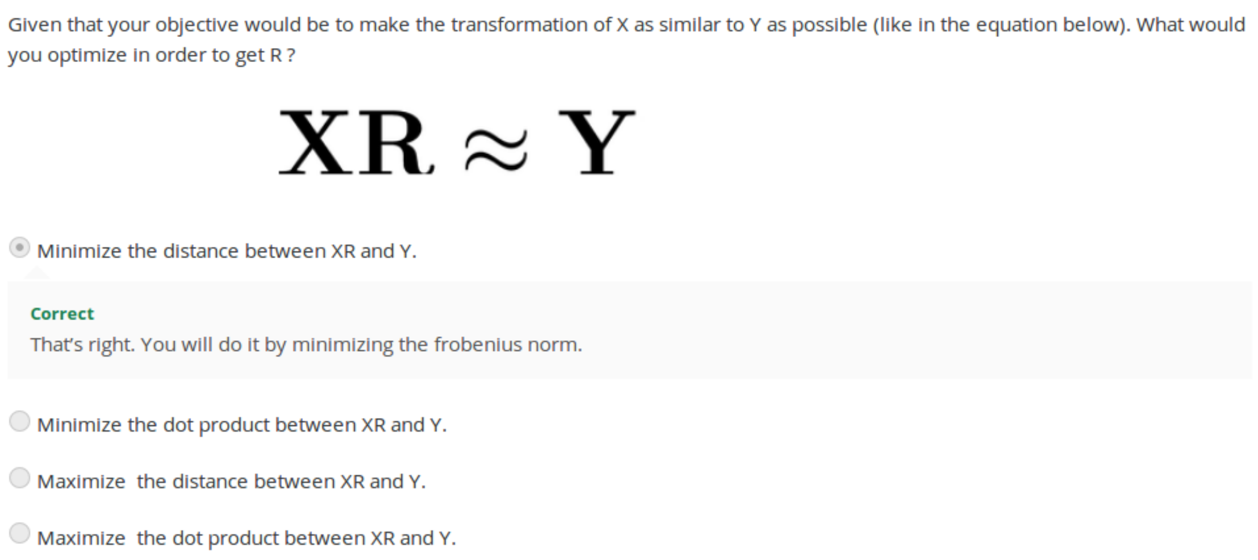


Transformation from English to French word vector



Using pair of english, french word, Learn**R**

R need to learn as part of training.



Given that your objective would be to make the transformation of X as similar to Y as possible (like in the equation below). What would you optimize in order to get R ?

$$\mathbf{XR} \approx \mathbf{Y}$$

[✓ Minimize the distance between XR and Y.]

Correct
That's right. You will do it by minimizing the frobenius norm.

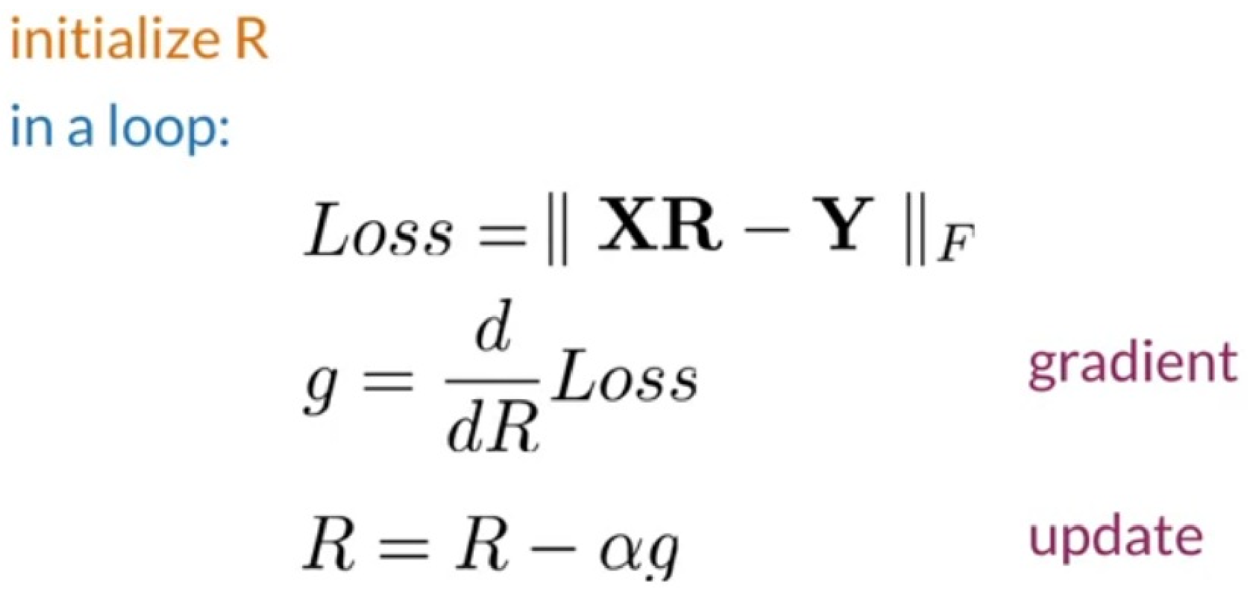
Minimize the dot product between XR and Y.

Maximize the distance between XR and Y.

Maximize the dot product between XR and Y.

Loss fn. to optimize **R**

Solving R



initialize R

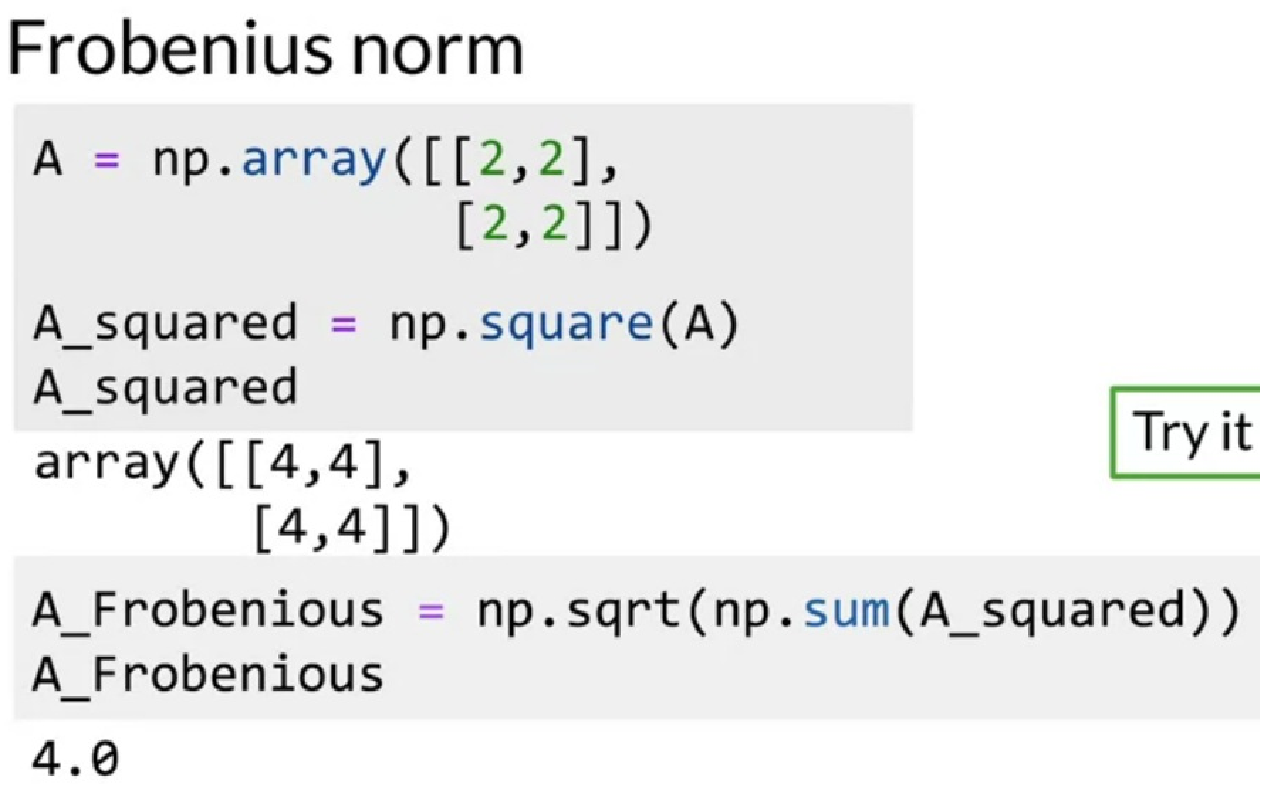
in a loop:

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

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

$$R = R - \alpha g \quad \text{update}$$

|| → Magnitude or Norm of matrix, F denote Frobenius form



Frobenius norm

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

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

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

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

```
4.0
```

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

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

→The gradient is a matrix that encodes how much a small change in `R` affect the change in the loss function.

→ The gradient gives us the direction in which we should decrease `R` to minimize the loss

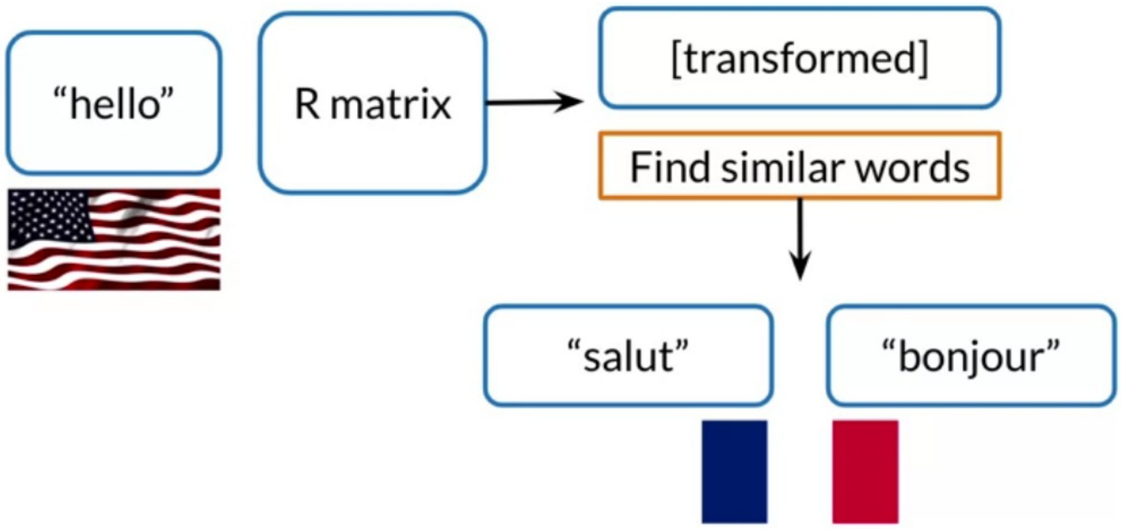
→ The gradient of the loss with respect to the matrix encodes how much a tiny change in some coordinate of that matrix affect the change of loss function.

There are three main vector transformations:

- Scaling
- Translation
- Rotation →It changes the direction of a vector, letting unaffected its dimensionality and its norm. The dot product between a vector and a square matrix produces a rotation and a scaling of the original vector. A

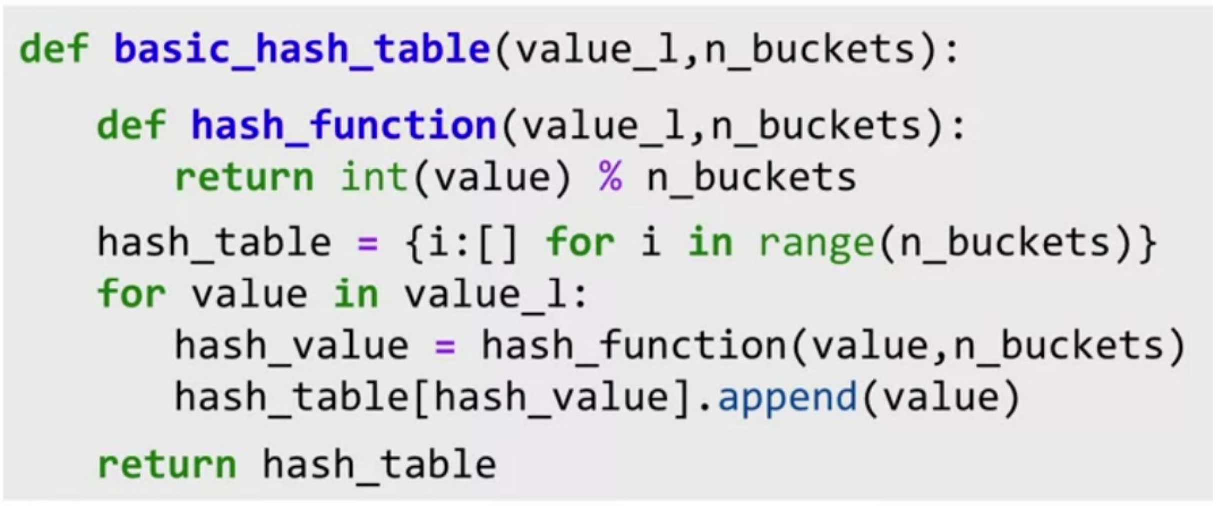
Nearest Neighbours

Finding the translation



Finding similar words can be very expensive as search space is all set of words in French vocabulary. We may search in subset of space instead of complete vocab.

Create a basic hash table



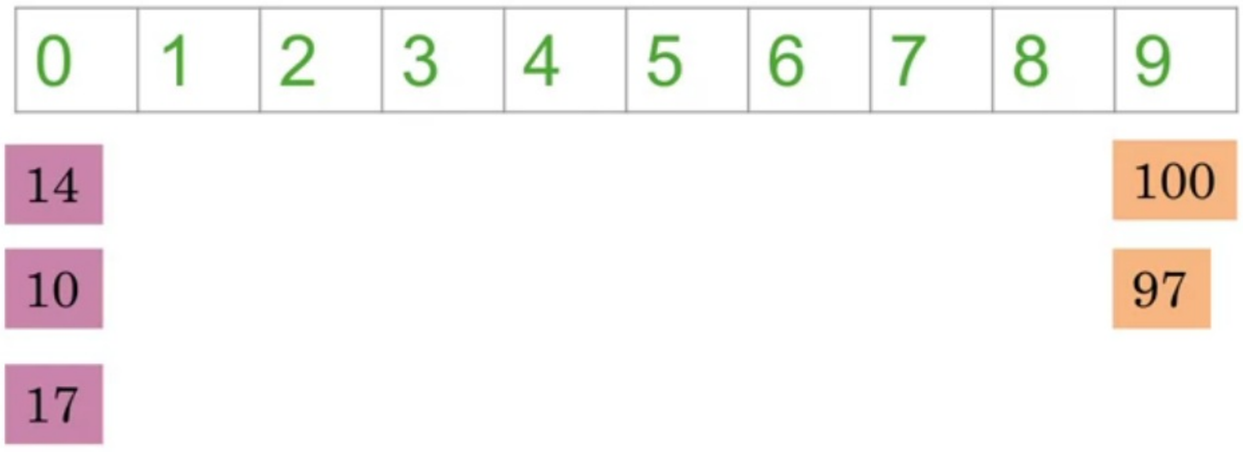
```
def basic_hash_table(value_l,n_buckets):

    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
```

Ideally, we want a hash function that puts similar word vectors in same bucket. Use **Locality Sensitive Hashing**.

→ Locality is another word for location

→ Sensitive is another word for caring

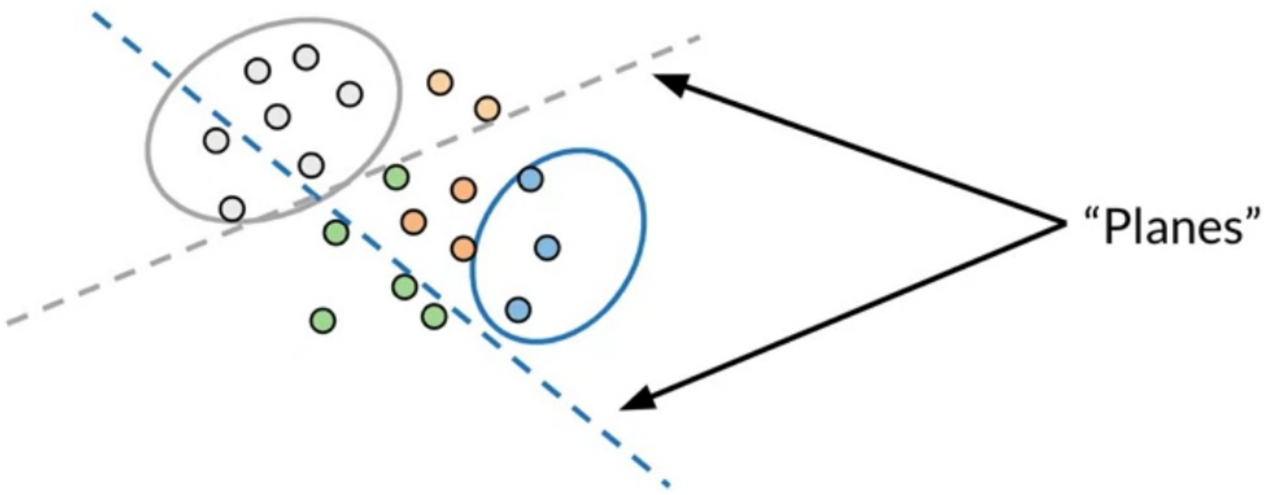
LSH is a hashing method that cares very deeply about assigning items based on where they are located in vector space.



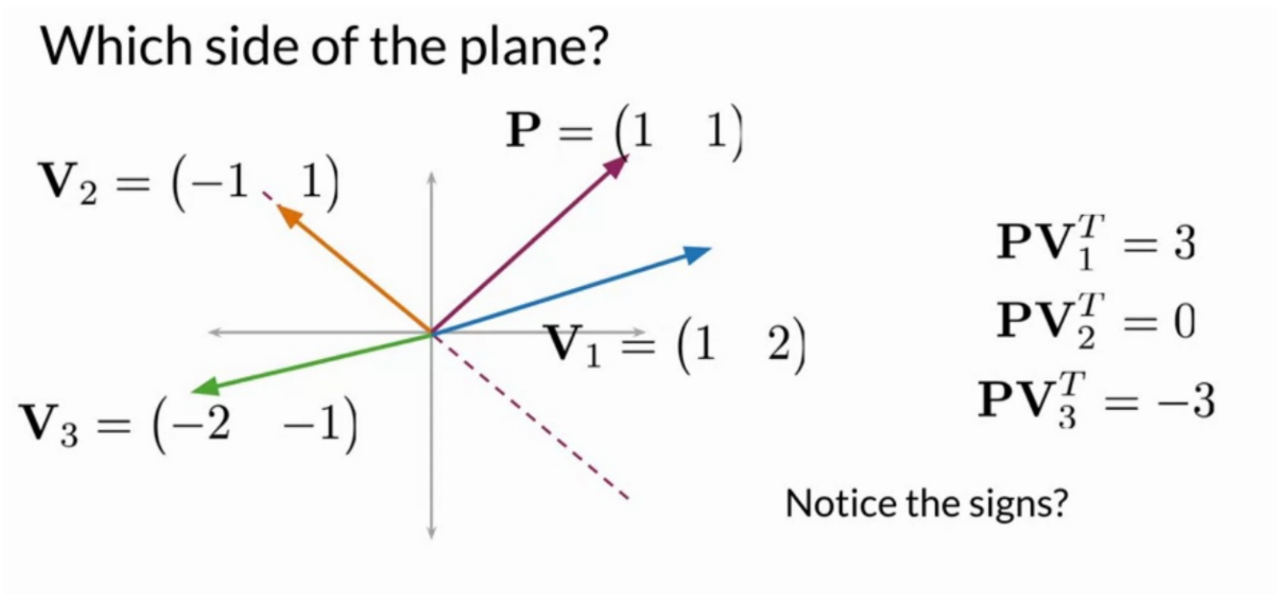
Locality Sensitive Hashing

→ **Planes** can help us bucket the vectors into subsets based on their location.

Each plane divides the space to 2 parts. So n planes divide the space into 2^n hash buckets.



→Normal vector is perpendicular to any vectors that lie on plane.

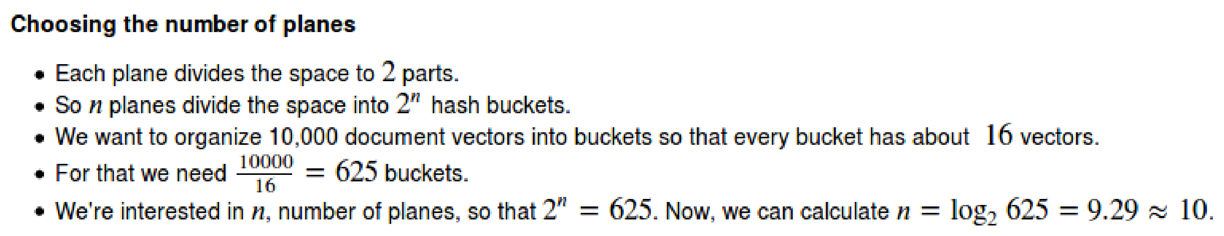

Which side of the plane?
$\mathbf{P} = (1 \quad 1)$
$\mathbf{V}_2 = (-1 \quad 1)$
$\mathbf{V}_1 = (1 \quad 2)$
$\mathbf{V}_3 = (-2 \quad -1)$
$\mathbf{P}\mathbf{V}_1^T = 3$
$\mathbf{P}\mathbf{V}_2^T = 0$
$\mathbf{P}\mathbf{V}_3^T = -3$
Notice the signs?

Sign of dot product → if +ve (same side of plane), -ve (opposite side of plane), 0 (On the plane).

```python
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
```

P → Normal vector of plane (Perpendicular to plane)

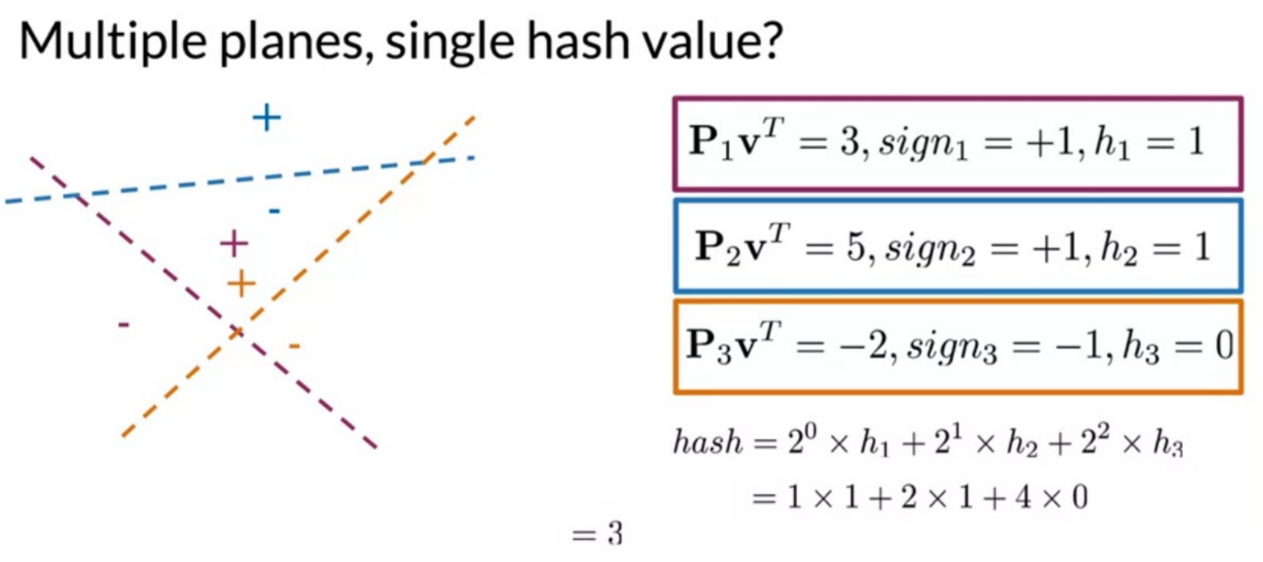
Multiple Planes



Choosing the number of planes

- Each plane divides the space to 2 parts.
- So n planes divide the space into 2^n hash buckets.
- We want to organize 10,000 document vectors into buckets so that every bucket has about 16 vectors.
- For that we need $\frac{10000}{16} = 625$ buckets.
- We're interested in n, number of planes, so that $2^n = 625$. Now, we can calculate $n = \log_2 625 = 9.29 \approx 10$.

No. of Planes



Multiple planes, single hash value?

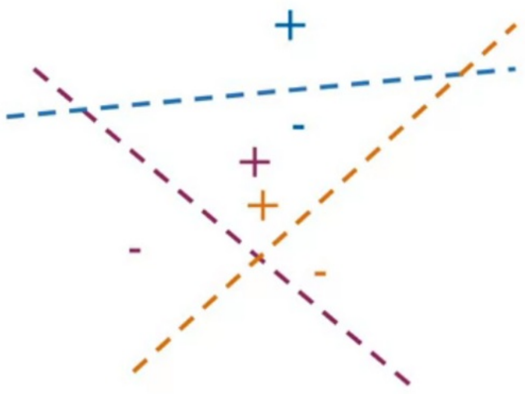
$$\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1$$

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

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

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

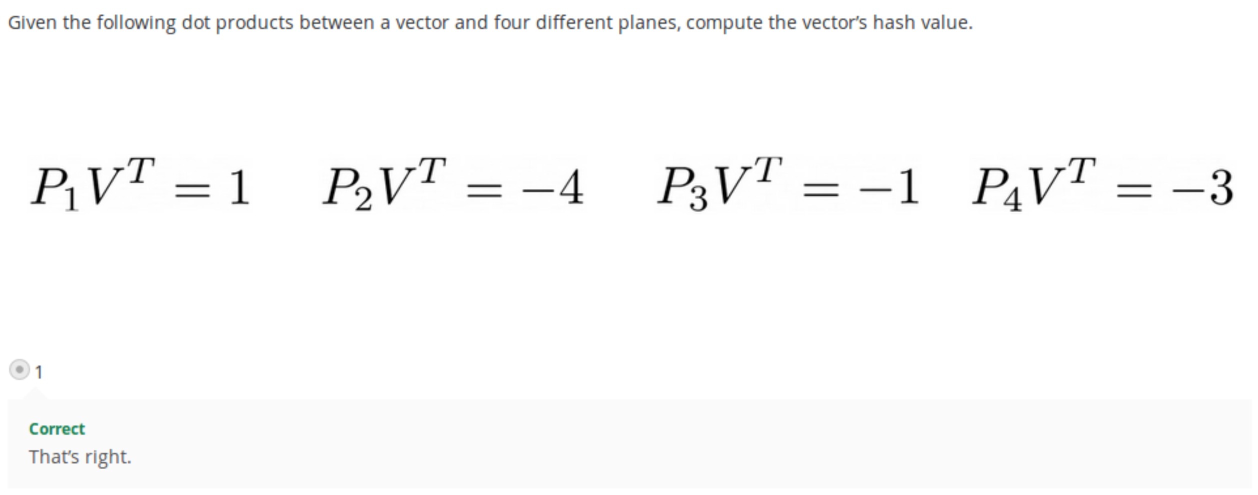
Multiple planes, single hash value!



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

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

Question:



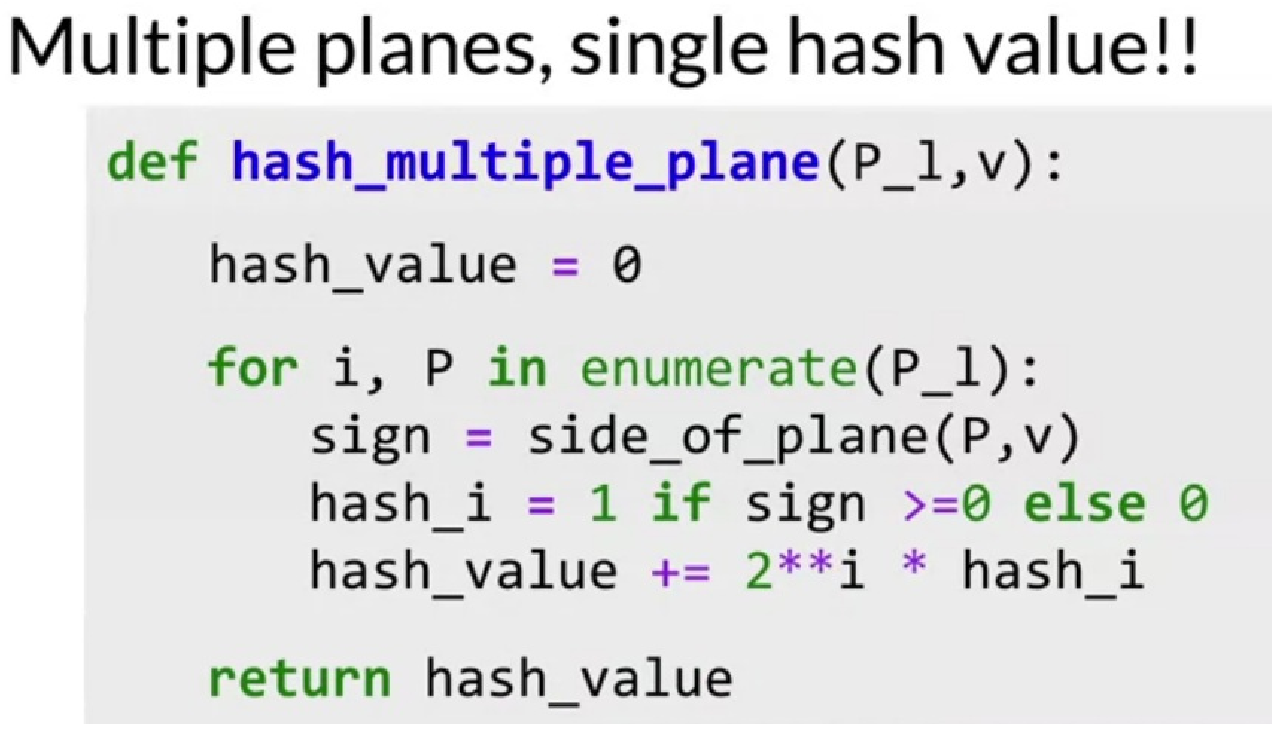
Given the following dot products between a vector and four different planes, compute the vector's hash value.

$$P_1V^T = 1 \quad P_2V^T = -4 \quad P_3V^T = -1 \quad P_4V^T = -3$$

[✓ 1]

Correct
That's right.

Question on Hash value



Multiple planes, single hash value!!

```python
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
```

## Create Random Planes

```
np.random.seed(0)
num_dimensions = 2 # is 300 in assignment
num_planes = 3 # is 10 in assignment
random_planes_matrix = np.random.normal(
                 size=(num_planes,
                     num_dimension
```

## Side of Plane

```
# Side of the plane function. The result is a matrix
def side_of_plane_matrix(P, v):
   dotproduct = np.dot(P, v.T)
   sign_of_dot_product = np.sign(dotproduct) # Get a boolean value telling if the value in the
cell is positive or negative
   return sign_of_dot_product
```

## Get Hash value for given vector and multiplane — > Region No

```
def hash_multi_plane_matrix(P, v, num_planes):
   sides_matrix = side_of_plane_matrix(P, v) # Get the side of planes for P and v
   hash_value = 0
   for i in range(num_planes):
      sign = sides_matrix[i].item() # Get the value inside the matrix cell
      hash_i = 1 if sign >=0 else 0
      hash_value += 2**i * hash_i # sum 2^i * hash_i

         return hash_valuev = [2, 2]hash_multi_plane_matrix(random_planes_matrix, v,
num_planes)
```

Idea of LSH was to divide vector space in to regions using multiple planes say which led to `N` regions and hash function help to tell exact region no given vector of same dimension of vector space (2-D).

But, Are these planes best way to divide vector space ? So, lets create multiple sets of random planes so that you can divide up vector space into multiple independent sets of hash tables.

Now, we can use LSH for K-Nearest neighbour known as ***Approximate KNN.***

> **numpy.squeeze()** removes unused dimensions from an array; for instance, it converts a (10,1) 2D array into a (10,) 1D array