Distance Metrics

Almost all machine learning algorithms use metrics while building their models. In general, there are two kinds of metrics: those for model evaluation and those for data evaluation.

Model Evaluation

These metrics help evaluate and compare different models. It could be different versions of the same algorithm (e.g. hyper-parameter tuning) or completely different algorithms (neural network model vs a Bayesian model). These metrics also help evaluate the "cost" during the building of a model. Each epoch (iteration of a model build) needs to be evaluated as well. There's no need to continue the search for a better model, if the current one is 'close enough'.

You may have heard of **loss functions**, **cost functions**, **fitness**, **optimizing the objective function** can refer to metrics at this stage. The objective function is the function you want to minimize (or maximize). Some reserve the term loss function for a single training example and cost function for the entire training set.

Specific metrics to evaluate a model will be discussed in detail for the lesson dedicated to the algorithm.

Data Relationship Evaluation: *distance*

These metrics help determine the relationship between different instances in a dataset. For example, in tf•idf, each word had a value to reflect how 'important' it was to a document.

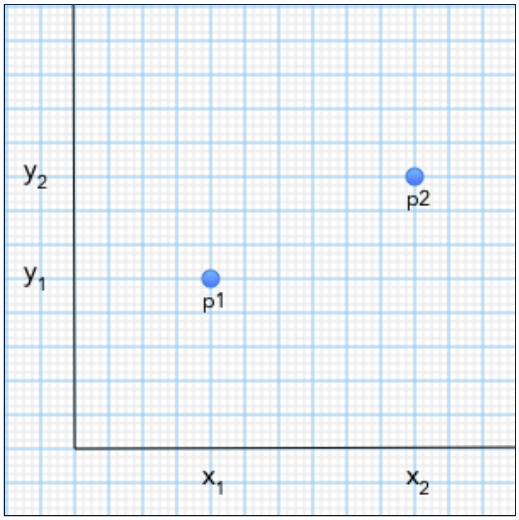
This lesson is focusing on the latter: where we want to know which metrics can be used to measure the 'distance' between elements/points in the dataset.

Algorithms for classification (K-Nearest Neighbors (KNN)), clustering (K-Means), and document retrieval all rely on some sort of metric to help determine distance or similarity between instances/data points.

Mathematical Primer

The cost of learning about distance metrics involves taking the time to understand some of the math behind the metrics. Sometimes the biggest barrier is just being able to read the nomenclature and symbols of the underlying formulas.

Let's start by looking at the familiar two dimensional graph (a.k.a. euclidean, Cartesian coordinates) for 2 points. Each point has two (hence two-dimensional) components: an `x` value and a `y` value.



Symbolically we can write $p_1 = \{x_1, y_1\}$ and $p_2 = \{x_2, y_2\}$

Both `p1` and `p2` can also be called vectors (we'll discuss the meaning soon) and can be written using 'vector' notation:

$$\overrightarrow{p_1} = \{x_1, y_1\}$$

Each 'dimension' is usually written using the same symbol. So rather than having an `x` attribute and a `y` attribute, a common sub-scripted letter (usually `u`, `v`, `w`) is used:

$$p_1 = \overrightarrow{u} = \{u_1, u_2\}$$
$$p_2 = \overrightarrow{v} = \{v_1, v_2\}$$

Sometimes the ^ (i.e. 'hat') symbol is used to signal the value as a vector as well. Also take note that u_1 and v_1 are the `x` attributes and u_2 and v_2 are the `y` attributes. Each subscript is the same dimension across all the vectors.

Multiple Dimensions

The nice thing about this notation is that it easily extends into multiple dimensions. We don't have to worry about using different letters (x,y,z). Each dimension is indicated by a number instead.

$$\overrightarrow{u} = \hat{u} = \{u_1, u_2, u_3, ..., u_{n-1}, u_n\}$$
$$\overrightarrow{v} = \hat{v} = \{v_1, v_2, v_3, ..., v_{n-1}, v_n\}$$

Both `u` and `v` are n-dimensional vectors. You can think of a dimension as simply an attribute rather than trying to visualize a high-dimensional space.

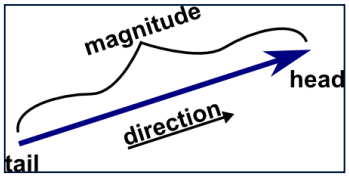
Each vector represents the same attribute across all data instances. It is also possible to have a vector represent all the attributes of a single instance. It's important to know what the vector represents -- and it usually depends on the problem or context.

Math on Vectors

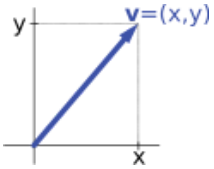
Once our data is expressed in vector form, we can do math on them (specifically linear algebra). The one caveat is that all the data (as vectors) needs to have the same number of dimensions. If you want to *add* `u` and `v`, they both need to have the same number of dimensions (components, attributes).

Physics, Math, and Data Vectors

Euclidean Vectors



One stumbling point is that different fields will use the word vector differently. In mathematics, vector represents direction/orientation and magnitude (and 'position' isn't a quantity).



In physics, the word vector usually refers to a physical observation. Examples might be velocity, torque, and acceleration. These vectors are called Euclidean vectors. Since position isn't considered, these vectors are always anchored at the origin (0,0).

Data Vectors

In computer science the word vector may be used to refer to a data type (e.g. arrays) or a type of graphics. Even in Python:

```
some_list = [0,1,2,3,4]
```

The variable `some_list` can be called (a one dimensional list with 5 elements) or a vector with 5 attributes (e.g. 5-dimensional; one row).

In data science the word **feature vector** is used to discuss instances with `n` dimensional attributes.

It indeed can be confusing. Vector based mathematics is deep and wide -- there are several courses that are centered on defining and describing properties of using vectors. Rather than going deep into the properties of vector math, we will try to stay applicable to our needs. If you have time, taking a course in linear algebra will be well worth the investment.

Too many normals 🤔

We have discussed *normalizing* data, and you may have heard of using an *L1 norm*, *L2 norm*, *normalizing* a vector, and even calculating a *vector norm*. It's all so confusing! The word normal is indeed multi-dimensional. Let's start with some simple definitions.

L1 Norm

L1 Single Vector Norm

The L1 Norm for a single vector is just the sum of the absolute values of the components.

$$L1 = \| u \|_1 = \sum_{i=1}^{n} |u_i|$$

L2 Norm

L2 Single Vector Norm

The L2 Norm for a single vector is just the square root of the summed squared component values.

$$L2 = \| u \|_2 = \sqrt{\sum_{i=1}^{n} (u_i)^2}$$

The L2 Norm is also called the ***length*** or ***magnitude*** of a vector.

Sometimes it's seen as $|u|$ or $\| u \|$

We can use numpy (and scipy) to demonstrate these norms

```
import numpy as np
from numpy import linalg as LA

values = np.array([x for x in range(0,10)])
l1_norm = LA.norm(values, ord=1)
l2_norm = LA.norm(values, ord=2)
ld_norm = LA.norm(values)  # default is ord==2
print('L1', l1_norm)
print('L2', l2_norm, ld_norm)
$_
```

For completeness sake the `L0` norm is the number of non-zero components in a vector. Both L1 and the L2 norms are used in their respective distance formulas

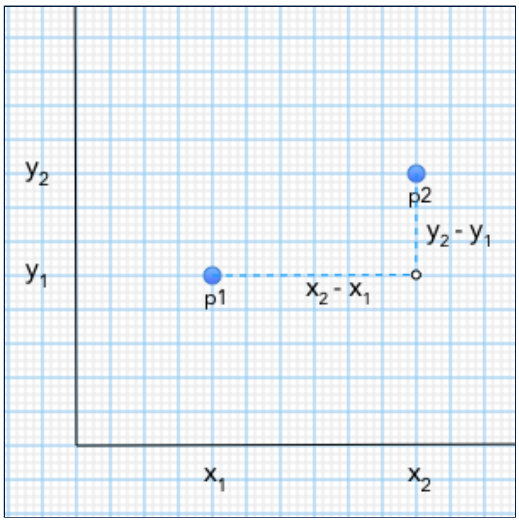
Distance Formulas

With both the L1 and L2 Norms defined, we also have our first two distance formulas (which work for any number of dimensions). In these examples, we are assuming that each vector contains the relevent attributes of one instance.

L1/Taxicab Distance

L1 Distance Norm

The L1 *distance* norm is also called Manhattan (or Taxicab) norm. It is the sum of the absolute difference of the components of the vector. In the image below, the distance is simply $|x_2 - x_1| + |y_2 - y_1|$



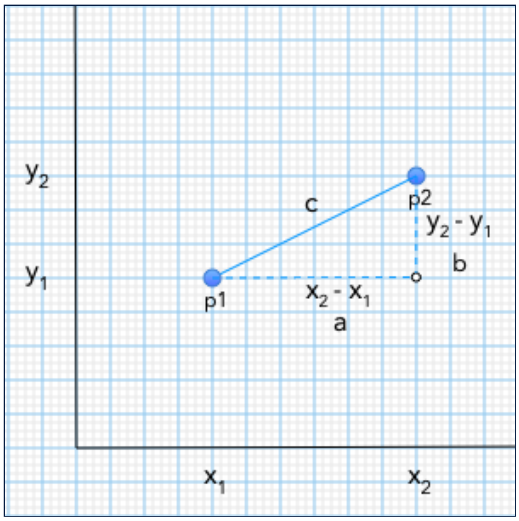
We can express this in vector notation as

$$L1_{norm} = \| D \|_1 = \sum_{i=1}^{n} |u_i - v_i|$$

L2/Euclidean Distance

L2 Distance Norm

The L2 *distance* norm (also called the Euclidean norm) is the square root of sum of the squared differences:



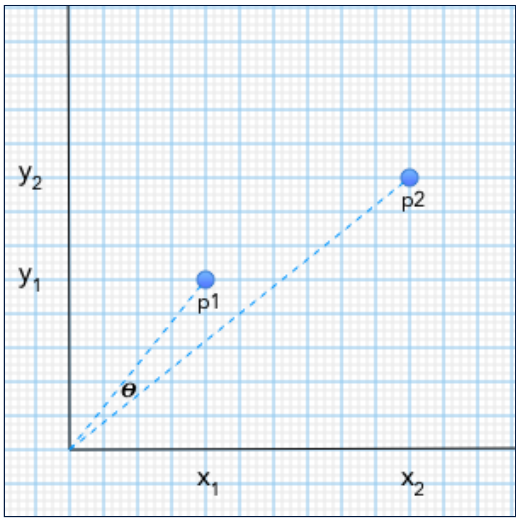
This is the familiar $a^2 + b^2 = c^2$ formula. We can express this in vector notation as

$$L2 = \| D \|_2 = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}$$

Note that these quantities are defined for *any number* of dimensions.

Cosine Distance

Although we have explained the cosine metric before (e.g. tf•idf), it's a good time to discuss it in greater detail. The cosine distance (or similarity) metric uses the L2 Norm in its formula. In the image below, the angle between the two vectors (θ) is the measurement we are interested in.



By using some geometric properties (and who doesn't long for their high-school geometry class), we can find the `cos(θ)` using the the formula below:

$$cos(\theta) = \frac{\sum_{i=1}^{n} (u_i \times v_i)}{\sqrt{\sum_{i=1}^{n} u_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}}$$

If you look carefully, you will notice the denominator is just the L2norm for u an v

$$cos(\theta) = \frac{\sum_{i=1}^{n}(u_i \times v_i)}{\| u \|_2 \| v \|_2}$$

Dot Product

In linear algebra, the dot product (also called the *inner product*) of two vectors is the result of summing the component-by-component multiplication:

$$u \cdot v = (u_1, .., u_n) \cdot (v_1, .., v_n) = u_1 v_1 + .. + u_n v_n$$

```
u = [2, 4, 6,]
v = [3, 5, 7,]
u•v = (2*3) + (4*5) + (6*7)
u•v = 6 + 20 + 42 = 68
```

So now we can rewrite the cos formula as

$$cos(\theta) = \frac{u \cdot v}{\| u \| \| v \|}$$

In fact, looking even closer, we can replace everything with dot products:

$$\frac{u \bullet v}{\sqrt{u \bullet u}\sqrt{v \bullet v}}$$

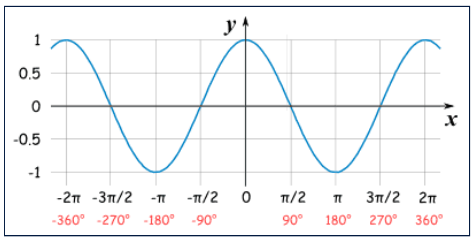
The following code demonstrates this (not a proof :)

```
import numpy as np
from numpy import linalg as LA
from scipy.spatial import distance

def cosine_similarity_v1(x, y):
    return np.dot(x,y)/(LA.norm(x) * LA.norm(y))

def cosine_similarity_v2(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))

def cosine_similarity_v3(x, y):
    return 1 - distance.cosine(x,y)

u = np.array([2, 4, 6,])
v = np.array([3, 5, 7,])
d1 = cosine_similarity_v1(u,v)
d2 = cosine_similarity_v2(u,v)
d3 = cosine_similarity_v3(u,v)
print(d1, d2, d3)

from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(u.reshape(1,3),v.reshape(1,3)))
$_
```

The Range of Values

Although we want the value of Θ, we take the cosine of that angle to map the values to the [-1, 1] range. This provides the benefit of being able to restrict the range of output.

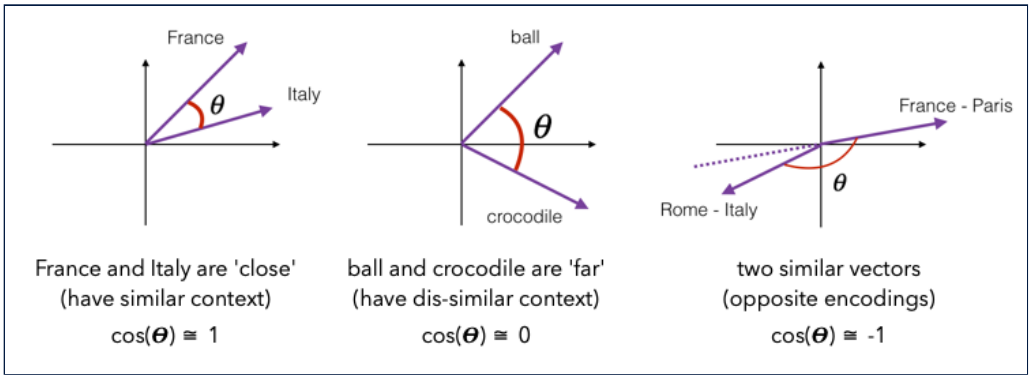
Take a look at the cosine graph to re-familiarize yourself with the range of the cosine function:



A value of 1 (Θ=0), means the vectors are completely aligned. When the dot product is 0, it means the vectors are perpendicular to each other (geometrically speaking).

Word Similarity

When we were discussing word similarity in spaCy (words with 'similar' embeddings), those close to 0 were very similar words (fri and fries) or words that co-occurred often (Italy and coffee).



Vector Normalization

In tf•idf scoring, there was an option to 'normalize' the document vectors. This was done so that longer documents (which will have more words) didn't become 'more important' just because of their length.

In vector normalization, we take the vector (the components of the vector) and divide each by the 'length' of the vector. However, length in this context *is* the L2 norm. This comes from terminology of vectors having both length and magnitude.

```
def norm_demo():
    # normalizing a vector
    u = np.array([1,2,3])
    l2_norm = LA.norm(u)  # sometimes called the 'length' of the vector
    unit_vector = u/l2_norm
    return unit_vector

print(norm_demo())
$_
```

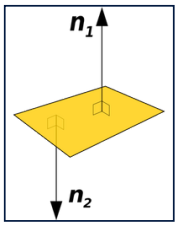
When you do vector normalization, you are essentially creating a *unit* vector. A unit vector has length/magnitude (i.e. L2 norm) of 1.0:

```
unit_vector = norm_demo()
# a unit vector has length 1
print(LA.norm(unit_vector))
$_
```

You can even use sklearn's preprocessing module to do vector normalization:

```
import sklearn.preprocessing as pre

v = np.array([0,3,-4])
unit_norm = pre.Normalizer(norm='l2').fit_transform([v])
print(unit_norm)

print(pre.normalize([v], norm='l2'))
$_
```

The "normal" vector



Just when you thought there's enough normal to go around, the phrase 'vector normal' adds yet another path. A normal vector is one that is perpendicular to a surface (sometimes called orthogonal). Many image shading algorithms rely on vector normals to understand orientation.

Minkowski Distance

Another distance metric is the generalization of the Euclidean metric. Rather than taking the square root, we take the `n`th root. It's named the Minkowski distance and it serves well for managing points in high dimensional space.

$$\sqrt[p]{(u_1 - v_1)^p + (u_2 - v_2)^p + .. + (u_n - v_n)^p}$$

In the KNN lesson, this was the `p` value for the model:

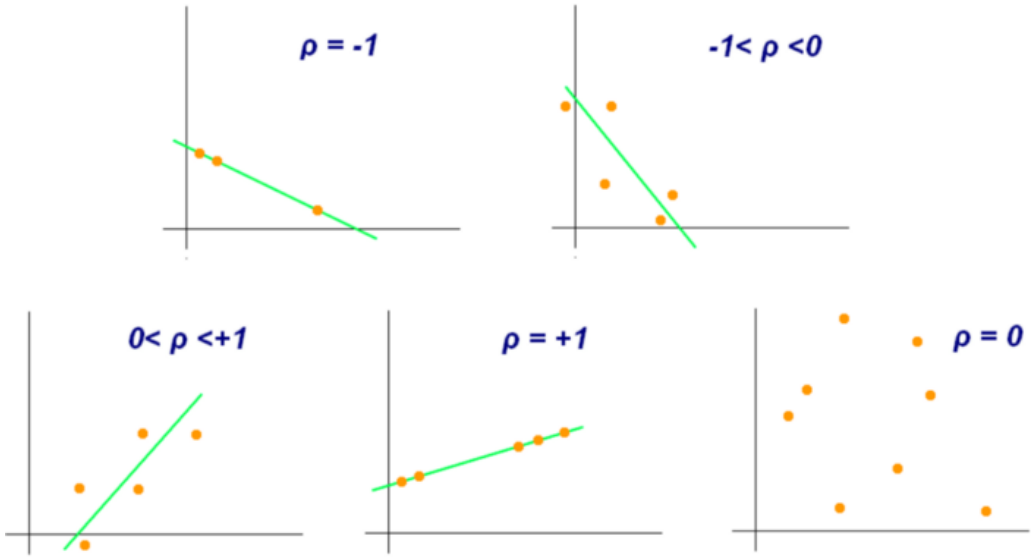
```
knn = KNeighborsClassifier(n_neighbors=11, p=2) # p == 2 euclidean
```

Pearson's correlation similarity

Person's correlation coefficient, a common statistic for linear regression, can also be used to measure how two items are correlated.

$$r_{uv} = \frac{\sum_{i=1}^{n}(u_i - \overline{u})(v_i - \overline{v})}{\sqrt{\sum_{i=1}^{n}(u_i - \overline{u})^2}\sqrt{\sum_{i=1}^{n}(v_i - \overline{v})^2}}$$

It's range is from -1 (negative correlation) to 0 (no correlation) to +1 (positive correlation). The higher the correlation, the higher the similarity.



Distance Metrics with Scipy

The distance metrics discussed (and a few more) are all available in scipy.

```
from scipy.spatial import distance
import scipy.stats

# defining the points
point_1 = (1, 2, 3)
point_2 = (4, 5, 6)

l1_dist = distance.cityblock(point_1, point_2)
print('Manhattan Distance {}, {} = {:.4f}'.format(point_1, point_2, l1_dist))

l2_dist = distance.euclidean(point_1, point_2)
print('Euclidean Distance {}, {} = {:.4f}'.format(point_1, point_2, l2_dist))

ln_dist = distance.minkowski(point_1, point_2, p=3)
print('Minkowski Distance {}, {} = {:.4f}'.format(point_1, point_2, ln_dist))

r, p_value = scipy.stats.pearsonr(point_1, point_2)
print('Pearson r {}, {} = {:.4f}'.format(point_1, point_2, r))
$_
```

Vector Math

We'll keep this section short, but it's important to know what's happening when you are working with vectors.

Multiplying Vectors

As mentioned, the dot product (or inner product) is when you multiply two vectors together. It is defined as

$$u \cdot v = (u_1, ..., u_n) \cdot (v_1, ..., v_n) = u_1 v_1 + ... + u_n v_n$$

Column x Row

The dot product whose value is a scalar (1 dimensional) can be thought of as a row vector multiplied by a column vector.

$$\begin{bmatrix} 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \\ 3 \end{bmatrix} = 36$$

```
u = np.array([2,3,4])
v = np.array([6,4,3])
print(np.dot(u,v))   # output is scalar
print(np.inner(u,v)) # same as dot
$_
```

Transposing Vectors

When you transpose a vector, you change it from 'row' form to 'column' form (and vica-versa). Each numpy vector has a `.T` attribute which represents the transpose:

$$\begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

```
u = np.array([[2,3,4]])
print(u)
print(u.T)
row = u.reshape(1,-1)
col = u.reshape(-1,1)
print(row)
print(col)
$_
```

You can now see that a vector multiplied by it's transpose is the square of the L2 norm. So using the transpose is sometimes included in both the definitions of dot product and L2 definitions:

```
def all_the_same():
 u = np.array([2,3,4])
 row = u.reshape(1,-1)
 print(row)
 print(row.T)

 print(np.sqrt(np.dot(row, row.T)))
 print(np.sqrt(np.dot(u, u.T)))
 print(LA.norm(u))
 print(np.sqrt(np.sum(u * u.T)))
$_
```

Row x Column

Taking a column vector and multiplying it by a row vector, actually produces a matrix (rows and columns)

$$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \times \begin{bmatrix} 6 & 4 & 3 \end{bmatrix} = \begin{bmatrix} 12 & 8 & 6 \\ 18 & 12 & 9 \\ 24 & 16 & 12 \end{bmatrix}$$

```
u = np.array([2,3,4])
v = np.array([6,4,3])

row = u.reshape(1,-1)
col = v.reshape(-1,1)
print(row)
print(col)

print(np.multiply(col, row)) # output is a matrix
print(np.outer(v,u))         # same but with vectors
```

Matrix Math

Until we start working on more complex data, we'll keep the review on matrix math even shorter.

Matrix Multiplication

The rules for multiplying matrices are similar (but there are restrictions on the shapes of what matrices can be multiplied together.

$$\underset{(2x3)}{\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}}\underset{(3x2)}{\begin{bmatrix}6 & 3\\ 5 & 2\\ 4 & 1\end{bmatrix}} = \underset{(2x2)}{\begin{bmatrix}1*6+2*5+3*4 & 1*3+2*2+3*1\\ 4*6+5*5+6*4 & 4*3+5*2+6*1\end{bmatrix}}$$

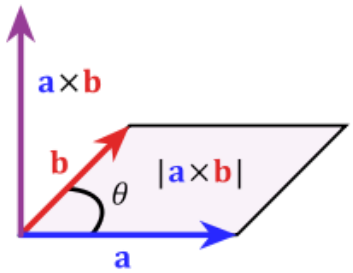
Transposing Matrices

Similarly, you can transpose a matrix as well:

$$\begin{bmatrix}6 & 4 & 24\\ 1 & -9 & 8\end{bmatrix}^T = \begin{bmatrix}6 & 1\\ 4 & -9\\ 24 & 8\end{bmatrix}$$

Cross Product ... (and so much more)

Another *kind* of vector multiplication is the *cross product*. The result of a cross product is another vector (sometimes called a vector product) that is at right angles to both.



You can solve for the perpendicular distance from a line to a point (useful in linear regression), by calculating a cross product between the different line segments.

There's certainly more to discuss when you dabble or dive in linear algebra. But we will re-visit more topics as they become necessary to understand a situation.

Review

Before you go, you should **know**:

- What is the L1 norm? ⊕
- What is the L2 norm? ⊕
- What does normalizing a vector mean? ⊕

What is the range of the cosine distance?

Lesson Assignment

We'll be using a few of these metrics in other lessons. Be sure you are comfortable with the material. You do not have to submit anything for this lesson.

The End!