3.3.3: Defining your similarity metric

Topic Progress: (https://gurus.pyimagesearch.com/topic/defining-your-image-descriptor/) (https://gurus.pyimagesearch.com/topic/indexing-your-dataset/) (https://gurus.pyimagesearch.com/topic/defining-your-similarity-metric/) (https://gurus.pyimagesearch.com/topic/searching/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/the-4-steps-of-building-any-image-search-engine/)

Feedback

Before we dive into this lesson, let's have a quick review.

Two lessons ago, we explored the first step of building an image search engine: **defining your image descriptor (https://gurus.pyimagesearch.com/topic/defining-your-image-descriptor/)**. We explored three aspects of an image that can easily be described: color, texture, and shape.

From there, we moved on to step two: **feature extraction and indexing (https://gurus.pyimagesearch.com/topic/indexing-your-dataset/)**. Feature extraction is the process of quantifying our dataset by applying an image descriptor to extract features from every image in our dataset, whereas indexing takes these feature vectors and applies specialized data structures to facilitate search speed.

Feature extraction is also a task that can easily be made parallel — if our dataset is large, we can easily speedup the feature extraction process by using multiple cores/processors on our machine.

Finally, regardless if we are using serial or parallel processing, we need to write our resulting feature vectors to disk for later use.

Now, it's time to move onto the third step of building an image search engine: **Defining your similarity metric**.

Objectives:

In this lesson, we will:

- Review the four conditions required for a function to be considered *distance metric*.
- Review popular distance functions/similarity metrics, including:
  - Euclidean
  - Manhattan/City Block
  - Intersection
  - χ^2
  - Cosine
  - Hamming

This lesson is meant to be a high-level overview of distance metrics. We won't be performing anything super practical in this lesson, other than discovering what Python packages contain which distance metrics. As we get further into this module, we'll start applying these similarity functions to compare feature vectors (and thus images for similarity). However, for the time being, let's just focus on how we use them to compute the distance between two vectors.

Feedback

Defining your similarity metric

Today, we are going to perform a cursory review of popular distance and similarity metrics we can use to compare two feature vectors.

Note: Depending on which distance function you use, there are many, many "gotchas" you need to look out for along the way. We'll be using many of these distance functions inside this course, and I'll be sure to provide examples on how to properly use them. But don't blindly apply a distance function to feature vectors without first understanding how the feature vectors should be scaled, normalized, etc., otherwise you might get strange results.

So what's the difference between a distance metric and a similarity metric?

In order to answer this question, we first need to define some variables.

Let d be our distance function, and x, y, and z be real-valued feature vectors, then the following conditions must hold:

1. **Non-negativity:** $d(x, y) >= 0$. This simply means that our distance must be non-negative and greater than or equal to zero.
2. **Coincidence Axiom:** $d(x, y) = 0$ if and only if $x = y$. A distance of zero (meaning the vectors are identical) is only possible of the two vectors have the same value.
3. **Symmetry:** $d(x, y) = d(y, x)$. To consider our distance function as a distance metric, the order of the parameters in the function should not matter. Specifying $d(x, y)$ instead of $d(y, x)$ should not matter to our distance metric — both function calls should return the same value.
4. **Triangle Inequality:** $d(x, z) <= d(x, y) + d(y, z)$. Do you remember back to your high school trigonometry class? This condition simply states that the sum of the lengths of any two sides must be greater than the remaining side.

If all four conditions hold, then our distance function can be considered a distance metric.

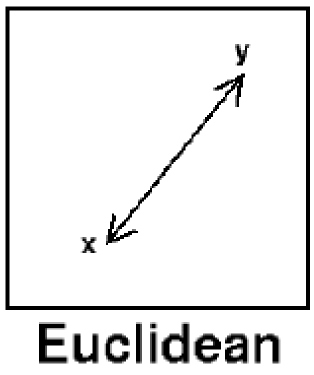
So does this mean that we should *only* use distance metrics and ignore other types of similarity metrics?

Feedback

Of course not!

But it's important to understand the terminology, especially when you go off to build image search engines on your own.

Euclidean



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/sim_metric_euclidean.png)

FIGURE 1: THE EUCLIDEAN DISTANCE BETWEEN TWO POINTS (SOURCE (HTTP://WWW.IMPROVEDOUTCOMES.COM/DOCS/WEBSITEDOCS/CLUSTERING/CLUSTERING_PARAI

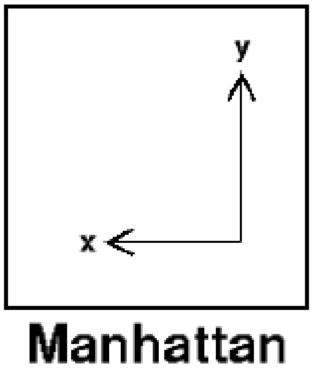
$d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{\sum_{i-1}^{N}(q_i - p_i)^2}$

The Euclidean Distance is arguably the most well known and most used distance metric. It is normally described as the distance between two points, “as the crow flies”.

Euclidean Python

```python
>>> import numpy as np
>>> np.random.seed(42)
>>> A = np.random.uniform(size=(128,))
>>> B = np.random.uniform(size=(128,))
>>> from scipy.spatial import distance as dists
>>> dists.euclidean(A, B)
4.718639454176824
>>> dists.euclidean(A, A)
0.0
```

Manhattan/City Block



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/sim_metric_manhattan.png)

FIGURE 2: THE MANHATTAN DISTANCE BETWEEN TWO POINTS (SOURCE (HTTP://WWW.IMPROVEDOUTCOMES.COM/DOCS/WEBSITEDOCS/CLUSTERING/CLUSTERING_PARAI

$d(\boldsymbol{p},\boldsymbol{q}) = \sum_{i-1}^{N} |q_i - p_i|$

The Manhattan Distance, or sometimes called City Block Distance, can be described as taking a drive in a taxicab along the city blocks until you reach your destination. The Manhattan Distance is much more forgiving than the Euclidean Distance, as the “penalty” is only the absolute value of the difference versus the square of the difference.

Manhattan/Cityblock Python

```python
>>> dists.cityblock(A, B)
42.855594992006971
>>> dists.cityblock(B, B)
0.0
```

Histogram Intersection

$d(\boldsymbol{H_1},\boldsymbol{H_2}) = \sum_{i-1}^{N} min(H_{1i}, H_{2i})$

As the name suggests, the Histogram Intersection metric is used to compare histogram distributions. The final similarity is the sum of the minimum entries between histograms H_1 and H_2.

Histogram intersection Python

```python
>>> def histogram_intersection(H1, H2):
...     return np.sum(np.minimum(H1, H2))
...
>>> histogram_intersection(A, B)
41.161048784578654
>>> histogram_intersection(A, A)
61.718585096884347
```

χ^2 Distance

$d(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{2} \sum_{i-1}^{N} (q_i - p_i)^2 / (q_i + p_i)$

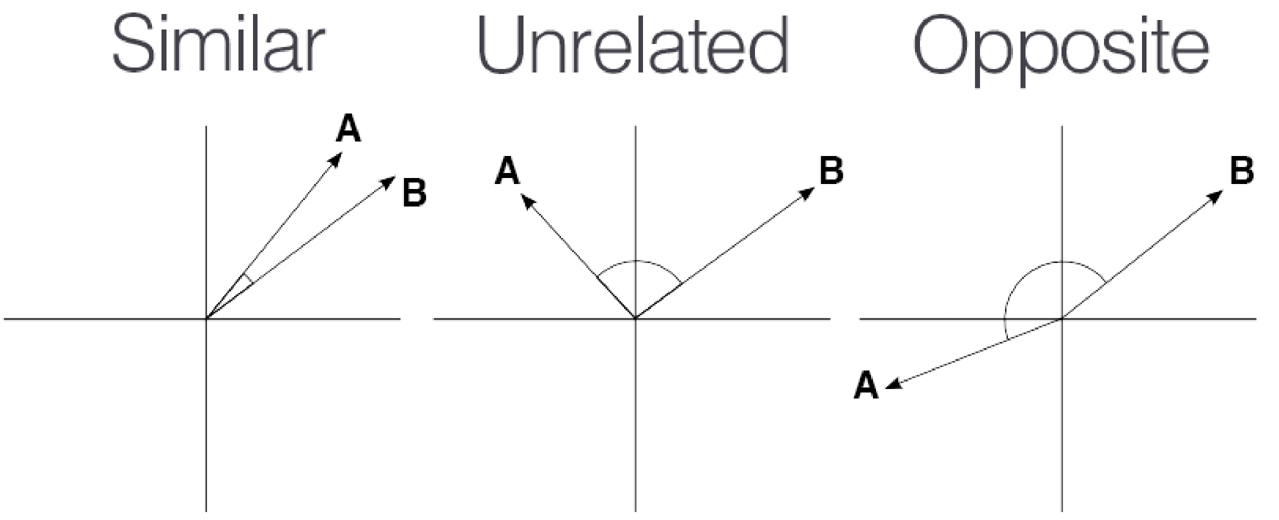
A more popular method to compare histogram distributions is the χ^2. Most problems in computer vision can be represented as histograms/probability distributions, so we'll make heavy use of the χ^2 Distance. And as we'll see later in this module, we can actually ***manipulate feature vectors*** in such a way that we can compare them using the Euclidean Distance, but they are *actually* being compared using χ^2 Distance!

Chi-squared Python

```python
>>> def chi2_distance(histA, histB, eps=1e-10):
...     return 0.5 * np.sum(((histA - histB) ** 2) / (histA + histB + eps))
...
>>> chi2_distance(A, B)
12.4605831086939
>>> chi2_distance(B, B)
0.0
```

Feedback

Cosine



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/06/sim_metric_cosine.png)

FIGURE 3: VISUALIZING THE COSINE DISTANCE.

$d(\boldsymbol{p}, \boldsymbol{q}) = cos(\theta) = (q \cdot p) / (||q||_2 ||p||_2)$

Where the $||\cdot||_2$ indicates the L_2 norm. The cosine distance returns a value between *-1* and *1*, where *1* indicates perfect similarity, *0* indicates orthogonality (decorrelation), and *-1* corresponds to "exactly opposite". Conceptually, we can think of the cosine distance as the angle between two vectors. We won't be using this similarity function as much until we get into the vector space model, tf-idf weighting, and high dimensional positive spaces, but the Cosine similarity function is extremely important.

It is also worth noting that the Cosine similarity function is not a proper distance metric — it violates both the triangle inequality and the coincidence axiom. To prove this to yourself, generate a few random feature vectors and apply these rules.

It's important to keep in mind that when using the SciPy implementation (http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.distance.cosine.html#scipy.spatial.distance.cosine) of the cosine distance, we are actually computing:

$$d(\boldsymbol{p}, \boldsymbol{q}) = 1 - (q \cdot p)/(||q||_2||p||_2)$$

Implying that *0* is perfect similarity and larger values indicate less similarity.

Cosine   Python
```python
>>> dists.cosine(A, B)
0.26637808522927109
>>> dists.cosine(A, A)
0.0
```

Feedback

Hamming

Given two binary vectors, the Hamming Distance measures the number of "disagreements" between the two vectors, then divides by the length. Two identical vectors would have zero disagreements, and thus perfect similarity.

Hamming   Python
```python
>>> A = np.random.randint(0, high=2, size=(8,))
>>> A
array([0, 1, 0, 0, 1, 1, 0, 1])
>>> B = np.random.randint(0, high=2, size=(8,))
>>> B
array([0, 1, 0, 1, 0, 0, 0, 1])
>>> dists.hamming(A, B)
0.375
>>> dists.hamming(A, A)
0.0
```

Summary

In this lesson, we performed a cursory exploration of distance and similarity functions that can be used to measure how "similar" two feature vectors are.

Popular distance functions and similarity functions include (but are certainly not limited to): Euclidean Distance, Manhattan (City Block) Distance, χ^2 Distance, Cosine Distance, and Hamming Distance.

This list is by no means exhaustive. There are an incredible amount of distance and similarity functions. But before we start diving into details and exploring when and where to use each function, I simply wanted to provide a 10,000 ft overview of what goes into building an image search engine.

At this point, you should have a basic idea of what it takes to build an image search engine. Start extracting features from images using simple color histograms (https://gurus.pyimagesearch.com/lessons/color-histograms/). Then, compare them using the distance functions discussed above. Be sure to note your findings, especially which distance functions worked best on which datasets.

Quizzes		Status
1	Defining your Similarity Metric Quiz (https://gurus.pyimagesearch.com/quizzes/defining-your-similarity-metric-quiz/)	

Feedback

← Previous Topic (https://gurus.pyimagesearch.com/topic/indexing-your-dataset/) Next Topic → (https://gurus.pyimagesearch.com/topic/searching/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)

- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback