Cosine Similarity

Learn about the cosine similarity metric and how it's used.

Chapter Goals:

- Understand how the cosine similarity metric measures the similarity between two data observations

A. What defines similarity?

To find similarities between data observations, we first need to understand how to actually measure similarity. The most common measurement of similarity is the cosine similarity metric.

A data observation with numeric features is essentially just a vector of real numbers. Cosine similarity is used in mathematics as a similarity metric for real-valued vectors, so it makes sense to use it as a similarity metric for data observations. The cosine similarity for two data observations is a number between -1 and 1. It specifically measures the *proportional* similarity of the feature values between the two data observations (i.e. the ratio between feature columns).

Cosine similarity values closer to 1 represent greater similarity between the observations, while values closer to -1 represent more divergence. A value of 0 means that the two data observations have no correlation (neither similar nor dissimilar).

B. Calculating cosine similarity

The cosine similarity for two vectors, u and v, is calculated as the dot product between the L2-normalization of the vectors. The exact formula for cosine similarity is:

$$\text{cossim}(u, v) = \frac{u}{||u||_2} \cdot \frac{v}{||v||_2}$$

where $||u||_2$ represents the L2 norm of u and $||v||_2$ represents the L2 norm of v.
In scikit-learn, cosine similarity is implemented via the `cosine_similarity` function (which is part of the `metrics.pairwise` module). It calculates the cosine similarities for pairs of data observations in a single dataset, or pairs of data observations between two datasets.

The code below computes cosine similarities between pairs of observations in a 2-D dataset.

```
from sklearn.metrics.pairwise import cosine_similarity
data = np.array([
  [ 1.1,  0.3],
  [ 2.1,  0.6],
  [-1.1, -0.4],
  [ 0. , -3.2]])
cos_sims = cosine_similarity(data)
print('{}\n'.format(repr(cos_sims)))
```

When we only pass in one dataset into `cosine_similarity`, the function will compute cosine similarities between pairs of observations within the dataset. In the code above, we passed in `data` (which contains 4 data observations), so the output of `cosine_similarity` is a 4x4 array of cosine similarity values.

The value at index (i, j) of `cos_sims` is the cosine similarity between data observations i and j in `data`. Since cosine similarity is symmetric, the `cos_sims` array contains the same values at index (i, j) and (j, i).

Note that the cosine similarity between a data observation and itself is 1, unless the data observation contains only 0's as feature values (in which case the cosine similarity is 0).

If we decide to pass in two datasets (with equal numbers of columns) into `cosine_similarity`, the function will compute the cosine similarities for pairs of data observations between the two datasets.

```
from sklearn.metrics.pairwise import cosine_similarity
data = np.array([
  [ 1.1,  0.3],
  [ 2.1,  0.6],
  [-1.1, -0.4],
```

```
  [ 0. , -3.2]])
data2 = np.array([
  [ 1.7,  0.4],
  [ 4.2, 1.25],
  [-8.1,  1.2]])
cos_sims = cosine_similarity(data, data2)
print('{}\n'.format(repr(cos_sims)))
```

In the code above, the value at index (i, j) of `cos_sims` is the cosine similarity between data observation i in `data` and data observation j in `data2`. Note that `cos_sims` is a 4x3 array, since `data` contains 4 data observations and `data2` contains 3.

Time to Code!

The code exercise for this chapter will be to use the `cosine_similarity` function to compute the most similar data observations for each data observation in `data`. Both `cosine_similarity` and `data` are imported/initialized in the backend.

First, we need to calculate the pairwise cosine similarities for each data observation.

Set `cos_sims` equal to `cosine_similarity` applied to `data`.

```
# CODE HERE
```

The diagonal in `cos_sims` represents the similarities between each observation and itself. We'll substitute each diagonal value with 0, so that for each data observation we find the most similar observation *besides itself*.

In NumPy, the function `fill_diagonal` allows us to fill the diagonal of an array with a specified value.

Call `np.fill_diagonal` with `cos_sims` as the first argument and `0` as the second argument.

```
# CODE HERE
```

For each row of `cos_sims`, the column containing the largest cosine similarity score represents the most similar data observation. We can find the column indexes with the largest value by using the `argmax` function of `cos_sims`.

We set the keyword argument `axis` equal to 1 to specify the largest column indexes for each row.

Set `similar_indexes` equal to the output of `cos_sims.argmax` with the `axis` keyword argument equal to `1`.

```
# CODE HERE
```