Choosing features and metrics for nearest neighbor search

When exploring a large set of documents -- such as Wikipedia, news articles, StackOverflow, etc. -- it can be useful to get a list of related material. To find relevant documents you typically

- Decide on a notion of similarity
- Find the documents that are most similar

In the assignment you will

- Gain intuition for different notions of similarity and practice finding similar documents.
- Explore the tradeoffs with representing documents using raw word counts and TF-IDF
- Explore the behavior of different distance metrics by looking at the Wikipedia pages most similar to President Obama's page.

If you are using GraphLab Create

An IPython Notebook has been provided below to you for this assignment. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code.

- Download the Wikipedia people dataset in SFrame format: people_wiki.gl.zip
- Download the companion IPython notebook: 0_nearest-neighbors-features-and-metrics_blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.

Open the companion IPython notebook and follow the instructions in the notebook. The instructions below do not apply to users of GraphLab Create.

If you are not using GraphLab Create

It is possible to complete this assignment without using GraphLab Create. The instructions below are geared towards Python users, but you are free to adapt them to your specific environment.

Disclaimer. We have tested the assessment using the standard Python installation (with access to scikit-learn). However, the assessment may not be compatible with other tools (e.g. Matlab, R).

Download the dataset

- Download the Wikipedia people dataset in SFrame format: people_wiki.gl.zip. (Those experimenting with other tools, get people_wiki.csv.zip instead.)

- Download the mapping between words and integer indices: people_wiki_map_index_to_word.gl.zip (or alternatively, people_wiki_map_index_to_word.json.zip)
- Download the pre-processed set of word counts:people_wiki_word_count.npz
- Download the pre-processed set of TF-IDF scores:people_wiki_tf_idf.npz

Import packages

```
import sframe                                   # see below for install instruction
import matplotlib.pyplot as plt                 # plotting
import numpy as np                              # dense matrices
from scipy.sparse import csr_matrix             # sparse matrices
%matplotlib inline
```

About SFrame. SFrame is a dataframe library Dato has released free-of-charge. Its source code is available here. You may install SFrame via pip:

```
pip install --upgrade sframe
```

Load in the dataset

We will be using the same dataset of Wikipedia pages that we used in the Machine Learning Foundations course (Course 1). Each element of the dataset consists of a link to the wikipedia article, the name of the person, and the text of the article (in lowercase).

```
wiki = sframe.SFrame('people_wiki.gl/')
wiki = wiki.add_row_number()                    # add row number, starting at 0
```

Extract word count vectors

For your convenience, we extracted the word count vectors from the dataset. The vectors are packaged in a sparse matrix, where the i-th row gives the word count vectors for the i-th document. Each column corresponds to a unique word appearing in the dataset. The mapping between words and integer indices are given in people_wiki_map_index_to_word.gl.

To load in the word count vectors, define the function

```
def load_sparse_csr(filename):
    loader = np.load(filename)
    data = loader['data']
    indices = loader['indices']
    indptr = loader['indptr']
    shape = loader['shape']

    return csr_matrix( (data, indices, indptr), shape)
```

and then run

```
word_count = load_sparse_csr('people_wiki_word_count.npz')
```

The word-to-index mapping is given by

```
map_index_to_word = sframe.SFrame('people_wiki_map_index_to_word.gl/')
```

(Optional) Extracting word count vectors yourself. We provide the pre-computed word count vectors to minimize potential compatibility issues. You are free to experiment with other tools to compute the word count vectors yourself. A good place to start is sklearn.CountVectorizer.

Find nearest neighbors using word count vectors

Let's start by finding the nearest neighbors of the Barack Obama page using the word count vectors to represent the articles and Euclidean distance to measure distance. For this, we will use scikit-learn's implementation of k-nearest neighbors. We first create an instance of the NearestNeighbor class, specifying the model parameters. Then we call the fit() method to attach the training set.

```
from sklearn.neighbors import NearestNeighbors

model = NearestNeighbors(metric='euclidean', algorithm='brute')
model.fit(word_count)
```

Run the following cell to obtain the row number for Obama's article:

```
print wiki[wiki['name'] == 'Barack Obama']
```

The output should resemble the following:

```
+-------+-------------------------------+--------------+
|   id  |              URI              |     name     |
+-------+-------------------------------+--------------+
| 35817 | <http://dbpedia.org/resour... | Barack Obama |
+-------+-------------------------------+--------------+
+-------------------------------+-------------------------------+
|              text             |           word_count          |
+-------------------------------+-------------------------------+
| barack hussein obama ii br... | {'operations': 1, 'represe... |
+-------------------------------+-------------------------------+
```

which locates Obama's article at index 35817.

Let us run the k-nearest neighbor algorithm with Obama's article. Since the NearestNeighbor class expects a vector, we pass the 35817th row of word_count vector.

```
distances, indices = model.kneighbors(word_count[35817], n_neighbors=10) # 1st arg: word count
    vector
```

The query returns the indices of and distances to the 10 nearest neighbors. To display the indices and distances together with the article name, run

```
neighbors = sframe.SFrame({'distance':distances.flatten(), 'id':indices.flatten()})
print wiki.join(neighbors, on='id').sort('distance')[['id','name','distance']]
```

which should give us an output in the form of the following:

```
+-------+---------------------------+---------------+
|   id  |            name           |    distance   |
+-------+---------------------------+---------------+
| 35817 |        Barack Obama       |      0.0      |
| 24478 |         Joe Biden         | 33.0756708171 |
| 28447 |       George W. Bush      | 34.3947670438 |
| 35357 |      Lawrence Summers     | 36.1524549651 |
| 14754 |        Mitt Romney        | 36.1662826401 |
| 13229 |      Francisco Barrio     | 36.3318042492 |
| 31423 |       Walter Mondale      | 36.4005494464 |
| 22745 | Wynn Normington Hugh-Jones | 36.4965751818 |
| 36364 |         Don Bonker        |  36.633318168 |
|  9210 |        Andy Anstett       | 36.9594372252 |
+-------+---------------------------+---------------+
```

For now, treat this piece of code as a black box; we will revisit the **join** operation in the following sections.

Interpreting the nearest neighbors

All of the 10 people are politicians, but about half of them have rather tenuous connections with Obama, other than the fact that they are politicians.

- Francisco Barrio is a Mexican politician, and a former governor of Chihuahua.
- Walter Mondale and Don Bonker are Democrats who made their career in late 1970s.
- Wynn Normington Hugh-Jones is a former British diplomat and Liberal Party official.
- Andy Anstett is a former politician in Manitoba, Canada.

Nearest neighbors with raw word counts got some things right, showing all politicians in the query result, but missed finer and important details.

For instance, let's find out why Francisco Barrio was considered a close neighbor of Obama. To do this, let's look at the most frequently used words in each of Barack Obama and Francisco Barrio's pages.

First, run the following cell to obtain the word_count column, which represents the word count vectors in the dictionary form. This way, we can quickly recognize words of great importance.

```
def unpack_dict(matrix, map_index_to_word):
    table = list(map_index_to_word.sort('index')['category'])
    data = matrix.data
    indices = matrix.indices
    indptr = matrix.indptr

    num_doc = matrix.shape[0]

    return [{k:v for k,v in zip([table[word_id] for word_id in indices[indptr[i]:indptr[i+1]]
                                 data[indptr[i]:indptr[i+1]].tolist())} \
               for i in xrange(num_doc) ]

wiki['word_count'] = unpack_dict(word_count, map_index_to_word)
```

Now printing the SFrame produces

```
+----+-------------------------------+---------------------+
| id |              URI              |         name        |
+----+-------------------------------+---------------------+
| 0  | <http://dbpedia.org/resour... |    Digby Morrell    |
| 1  | <http://dbpedia.org/resour... |    Alfred J. Lewy   |
| 2  | <http://dbpedia.org/resour... |    Harpdog Brown    |
| 3  | <http://dbpedia.org/resour... | Franz Rottensteiner |
| 4  | <http://dbpedia.org/resour... |        G-Enka       |
| 5  | <http://dbpedia.org/resour... |    Sam Henderson    |
| 6  | <http://dbpedia.org/resour... |    Aaron LaCrate    |
| 7  | <http://dbpedia.org/resour... |   Trevor Ferguson   |
| 8  | <http://dbpedia.org/resour... |     Grant Nelson    |
| 9  | <http://dbpedia.org/resour... |     Cathy Caruth    |
+----+-------------------------------+---------------------+
+-------------------------------+-------------------------------+
|              text             |           word_count          |
+-------------------------------+-------------------------------+
| digby morrell born 10 octo... | {'selection': 1, 'carltons... |
| alfred j lewy aka sandy le... | {'precise': 1, 'thomas': 1... |
| harpdog brown is a singer ... | {'just': 1, 'issued': 1, '... |
| franz rottensteiner born i... | {'englishreading': 1, 'all... |
| henry krvits born 30 decem... | {'they': 1, 'gangstergenka... |
| sam henderson born october... | {'now': 1, 'currently': 1,... |
| aaron lacrate is an americ... | {'exclusive': 2, 'producer... |
| trevor ferguson aka john f... | {'taxi': 1, 'salon': 1, 'g... |
| grant nelson born 27 april... | {'houston': 1, 'frankie': ... |
| cathy caruth born 1955 is ... | {'phenomenon': 1, 'deboras... |
+-------------------------------+-------------------------------+
```

To make things even easier, we provide a utility function that displays a dictionary in tabular form:

```
def top_words(name):
    """
    Get a table of the most frequent words in the given person's wikipedia page.
    """
    row = wiki[wiki['name'] == name]
    word_count_table = row[['word_count']].stack('word_count', new_column_name=['word','count
    return word_count_table.sort('count', ascending=False)

obama_words = top_words('Barack Obama')
print obama_words

barrio_words = top_words('Francisco Barrio')
print barrio_words
```

Let's extract the list of most frequent words that appear in both Obama's and Barrio's documents. We've so far sorted all words from Obama and Barrio's articles by their word frequencies. We will now use a dataframe operation known as **join**. The **join** operation is very useful when it comes to playing around with data: it lets you combine the content of two tables using a shared column (in this case, the word column). See the documentation for more details.

For instance, running

```
combined_words = obama_words.join(barrio_words, on='word')
```

will extract the rows from both tables that correspond to the common words.

Since both tables contained the column named count, SFrame automatically renamed one of them to prevent confusion. Let's rename the columns to tell which one is for which. By inspection, we see that the first column (count) is for Obama and the second (count.1) for Barrio.

```
combined_words = combined_words.rename({'count':'Obama', 'count.1':'Barrio'})
```

Note. The **join** operation does not enforce any particular ordering on the shared column. So to obtain, say, the five common words that appear most often in Obama's article, sort the combined table by the Obama column. Don't forget ascending=False to display largest counts first.

```
combined_words.sort('Obama', ascending=False)
```

Quiz Question. Among the words that appear in both Barack Obama and Francisco Barrio, take the 5 that appear most frequently in Obama. How many of the articles in the Wikipedia dataset contain all of those 5 words?

Hint:

- Refer to the previous paragraph for finding the words that appear in both articles. Sort the common words by their frequencies in Obama's article and take the largest five.
- Each word count vector is a Python dictionary. For each word count vector in SFrame, you'd have to check if the set of the 5 common words is a subset of the keys of the word count vector. Complete the function has_top_words to accomplish the task.
- Convert the list of top 5 words into set using the syntax "set(common_words)", where common_words is a Python list. See this link if you're curious about Python sets.
- Extract the list of keys of the word count dictionary by calling the keys() method.
- Convert the list of keys into a set as well.
- Use issubset() method to check if all 5 words are among the keys.
- Now apply the has_top_words function on every row of the SFrame.
- Compute the sum of the result column to obtain the number of articles containing all the 5 top words.

```
common_words = ...  # YOUR CODE HERE
```

```

def has_top_words(word_count_vector):
    # extract the keys of word_count_vector and convert it to a set
    unique_words = ...   # YOUR CODE HERE
    # return True if common_words is a subset of unique_words
    # return False otherwise
    return ...  # YOUR CODE HERE

wiki['has_top_words'] = wiki['word_count'].apply(has_top_words)

# use has_top_words column to answer the quiz question
... # YOUR CODE HERE
```

Checkpoint. Check your has_top_words function on two random articles:

```
print 'Output from your function:', has_top_words(wiki[32]['word_count'])
print 'Correct output: True'
print 'Also check the length of unique_words. It should be 167'

print 'Output from your function:', has_top_words(wiki[33]['word_count'])
print 'Correct output: False'
print 'Also check the length of unique_words. It should be 188'
```

Quiz Question. Measure the pairwise distance between the Wikipedia pages of Barack Obama, George W. Bush, and Joe Biden. Which of the three pairs has the smallest distance?

Hint: To compute the Euclidean distance between two dictionaries, usesklearn.metrics.pairwise.euclidean_distances.

Quiz Question. Collect all words that appear both in Barack Obama and George W. Bush pages. Out of those words, find the 10 words that show up most often in Obama's page.

Note. Even though common words are swamping out important subtle differences, commonalities in rarer political words still matter on the margin. This is why politicians are being listed in the query result instead of musicians, for example. In the next subsection, we will introduce a different metric that will place greater emphasis on those rarer words.

Extract the TF-IDF vectors

Much of the perceived commonalities between Obama and Barrio were due to occurrences of extremely frequent words, such as "the", "and", and "his". So the nearest neighbors algorithm is recommending plausible results sometimes for the wrong reasons.

To retrieve articles that are more relevant, we should focus more on rare words that don't happen in every article. **TF-IDF** (term frequency–inverse document frequency) is a feature representation that penalizes words that are too common. Let us load in the TF-IDF vectors and repeat the nearest neighbor search.

For your convenience, we extracted the TF-IDF vectors from the dataset. The vectors are packaged in a sparse matrix, where the i-th row gives the TF-IDF vectors for the i-th document. Each column corresponds to a unique word appearing in the dataset. The mapping between words and integer indices are given in people_wiki_map_index_to_word.gl.

To load in the TF-IDF vectors, run

```
tf_idf = load_sparse_csr('people_wiki_tf_idf.npz')
```

In addition to the sparse matrix, we also store the TF-IDF vectors in dictionary form as well, to allow for easy interpretation.

```
wiki['tf_idf'] = unpack_dict(tf_idf, map_index_to_word)
```

(Optional) Extracting TF-IDF vectors yourself. We provide the pre-computed TF-IDF vectors to minimize potential compatibility issues. You are free to experiment with other tools to compute the word count vectors yourself. A good place to start is sklearn.TfidfVectorizer.

Find nearest neighbors using TF-IDF vectors

Since we are now using a different set of features, we should create a new nearest neighbor model. Create another instance of the NearestNeighbor class as follows. Then call the fit() method to associate it with the TF-IDF vectors.

```
model_tf_idf = NearestNeighbors(metric='euclidean', algorithm='brute')
model_tf_idf.fit(tf_idf)
```

Perform the nearest neighbor search by running

```
distances, indices = model_tf_idf.kneighbors(tf_idf[35817], n_neighbors=10)
```

To print the names of the articles, we perform join using the indices:

```
neighbors = sframe.SFrame({'distance':distances.flatten(), 'id':indices.flatten()})
print wiki.join(neighbors, on='id').sort('distance')[['id', 'name', 'distance']]
```

which produces

```
+-------+-------------------------+---------------+
|   id  |           name          |    distance   |
+-------+-------------------------+---------------+
| 35817 |       Barack Obama      |      0.0      |
|  7914 |      Phil Schiliro      | 106.861013691 |
| 46811 |      Jeff Sessions      | 108.871674216 |
| 44681 |  Jesse Lee (politician) | 109.045697909 |
| 38376 |      Samantha Power     | 109.108106165 |
|  6507 |       Bob Menendez      | 109.781867105 |
| 38714 | Eric Stern (politician) |  109.95778808 |
| 44825 |      James A. Guest     | 110.413888718 |
| 44368 |   Roland Grossenbacher  |  110.4706087  |
| 33417 |      Tulsi Gabbard      | 110.696997999 |
```

Let's determine whether this list makes sense.

- With a notable exception of Roland Grossenbacher, the other 8 are all American politicians who are contemporaries of Barack Obama.
- Phil Schiliro, Jesse Lee, Samantha Power, and Eric Stern worked for Obama.

Clearly, the results are more plausible with the use of TF-IDF. Let's take a look at the word vector for Obama and Schilirio's pages. Notice that TF-IDF representation assigns a weight to each word. This weight captures relative importance of that word in the document. Let us sort the words in Obama's article by their TF-IDF weights; we do the same for Schiliro's article as well.

```
def top_words_tf_idf(name):
    row = wiki[wiki['name'] == name]
    word_count_table = row[['tf_idf']].stack('tf_idf', new_column_name=['word','weight'])
    return word_count_table.sort('weight', ascending=False)

obama_tf_idf = top_words_tf_idf('Barack Obama')
print obama_tf_idf

schiliro_tf_idf = top_words_tf_idf('Phil Schiliro')
print schiliro_tf_idf
```

Using the **join** operation we learned earlier, try your hands at computing the common words shared by Obama's and Schiliro's articles. Sort the common words by their TF-IDF weights in Obama's document. The first 10 words should say: Obama, law, democratic, Senate, presidential, president, policy, states, office, 2011.

Quiz Question. Among the words that appear in both Barack Obama and Phil Schiliro, take the 5 that have largest weights in Obama. How many of the articles in the Wikipedia dataset contain all of those 5 words?

```
common_words = ...  # YOUR CODE HERE

def has_top_words(word_count_vector):
    # extract the keys of word_count_vector and convert it to a set
    unique_words = ...   # YOUR CODE HERE
    # return True if common_words is a subset of unique_words
    # return False otherwise
    return ...  # YOUR CODE HERE

wiki['has_top_words'] = wiki['word_count'].apply(has_top_words)

# use has_top_words column to answer the quiz question
...  # YOUR CODE HERE
```

Notice the huge difference in this calculation using TF-IDF scores instead of raw word counts. We've eliminated noise arising from extremely common words.

Choosing metrics

You may wonder why Joe Biden, Obama's running mate in two presidential elections, is missing from the query results of model_tf_idf. Let's find out why. First, compute the distance between TF-IDF features of Obama and Biden.

Quiz Question. Compute the Euclidean distance between TF-IDF features of Obama and Biden.

The distance is larger than the distances we found for the 10 nearest neighbors. But one may wonder, is Biden's article that different from Obama's, more so than, say, Schiliro's? It turns out that, when we compute nearest neighbors using the Euclidean distances, we unwittingly favor short articles over long ones. Let us compute the length of each Wikipedia document, and examine the document lengths for the 100 nearest neighbors to Obama's page.

```
# Comptue length of all documents
def compute_length(row):
    return len(row['text'])
wiki['length'] = wiki.apply(compute_length)

# Compute 100 nearest neighbors and display their lengths
distances, indices = model_tf_idf.kneighbors(tf_idf[35817], n_neighbors=100)
neighbors = sframe.SFrame({'distance':distances.flatten(), 'id':indices.flatten()})
nearest_neighbors_euclidean = wiki.join(neighbors, on='id')[['id', 'name', 'length',
    'distance']].sort('distance')
print nearest_neighbors_euclidean
```

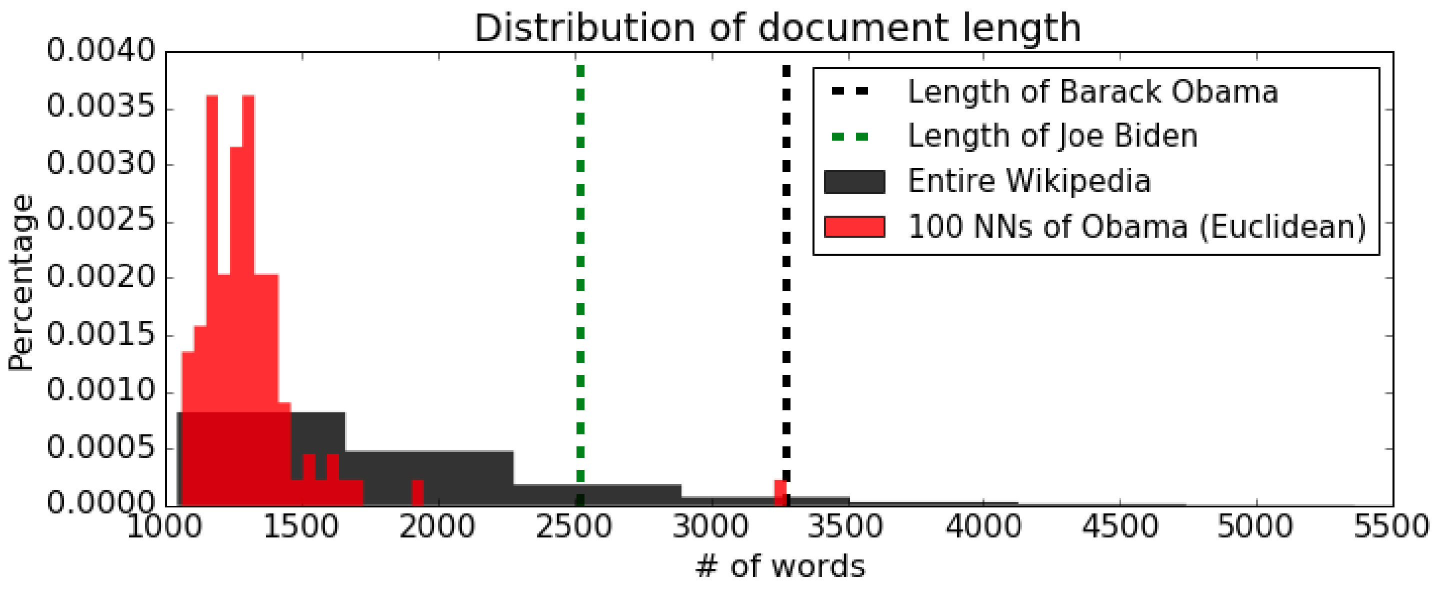
The output should be in the form

```
+-------+-------------------------+--------+---------------+
|   id  |           name          | length |    distance   |
+-------+-------------------------+--------+---------------+
| 35817 |       Barack Obama      |  3278  |      0.0      |
|  7914 |      Phil Schiliro      |  1288  | 106.861013691 |
| 46811 |      Jeff Sessions      |  1398  | 108.871674216 |
| 44681 |  Jesse Lee (politician) |  1374  | 109.045697909 |
| 38376 |      Samantha Power     |  1911  | 109.108106165 |
|  6507 |       Bob Menendez      |  1222  | 109.781867105 |
| 38714 | Eric Stern (politician) |  1589  |  109.95778808 |
| 44825 |      James A. Guest     |  1251  | 110.413888718 |
| 44368 |   Roland Grossenbacher  |  1099  |  110.4706087  |
| 33417 |      Tulsi Gabbard      |  1352  | 110.696997999 |
+-------+-------------------------+--------+---------------+
```

To see how these document lengths compare to the lengths of other documents in the corpus, let's make a histogram of the document lengths of Obama's 100 nearest neighbors and compare to a histogram of document lengths for all documents.

```
plt.figure(figsize=(10.5,4.5))
plt.hist(wiki['length'], 50, color='k', edgecolor='None', histtype='stepfilled', normed=True,
         label='Entire Wikipedia', zorder=3, alpha=0.8)
plt.hist(nearest_neighbors_euclidean['length'], 50, color='r', edgecolor='None', histtype
    ='stepfilled', normed=True,
         label='100 NNs of Obama (Euclidean)', zorder=10, alpha=0.8)
plt.axvline(x=wiki['length'][wiki['name'] == 'Barack Obama'][0], color='k', linestyle='--',
    linewidth=4,
           label='Length of Barack Obama', zorder=2)
plt.axvline(x=wiki['length'][wiki['name'] == 'Joe Biden'][0], color='g', linestyle='--',
    linewidth=4,
           label='Length of Joe Biden', zorder=1)
plt.axis([1000, 5500, 0, 0.004])

plt.legend(loc='best', prop={'size':15})
plt.title('Distribution of document length')
plt.xlabel('# of words')
plt.ylabel('Percentage')
plt.rcParams.update({'font.size':16})
```



Relative to the rest of Wikipedia, nearest neighbors of Obama are overwhelmingly short, most of them being shorter than 2000 words. The bias towards short articles is not appropriate in this application as there is really no reason to favor short articles over long articles (they are all Wikipedia articles, after all). Many Wikipedia articles are 2500 words or more, and both Obama and Biden are over 2500 words long.

Note: Both word-count features and TF-IDF are proportional to word frequencies. While TF-IDF penalizes very common words, longer articles tend to have longer TF-IDF vectors simply because they have more words in them.

To remove this bias, we turn to **cosine distances**:

d(\mathbf{x},\mathbf{y}) = 1 - \dfrac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}

Cosine distances let us compare word distributions of two articles of varying lengths.

Let us train a new nearest neighbor model, this time with cosine distances. We then repeat the search for Obama's 100 nearest neighbors.

```
model2_tf_idf = NearestNeighbors(algorithm='brute', metric='cosine')
model2_tf_idf.fit(tf_idf)
distances, indices = model2_tf_idf.kneighbors(tf_idf[35817], n_neighbors=100)
neighbors = sframe.SFrame({'distance':distances.flatten(), 'id':indices.flatten()})
nearest_neighbors_cosine = wiki.join(neighbors, on='id')[['id', 'name', 'length', 'distance']]
    .sort('distance')
print nearest_neighbors_cosine
```

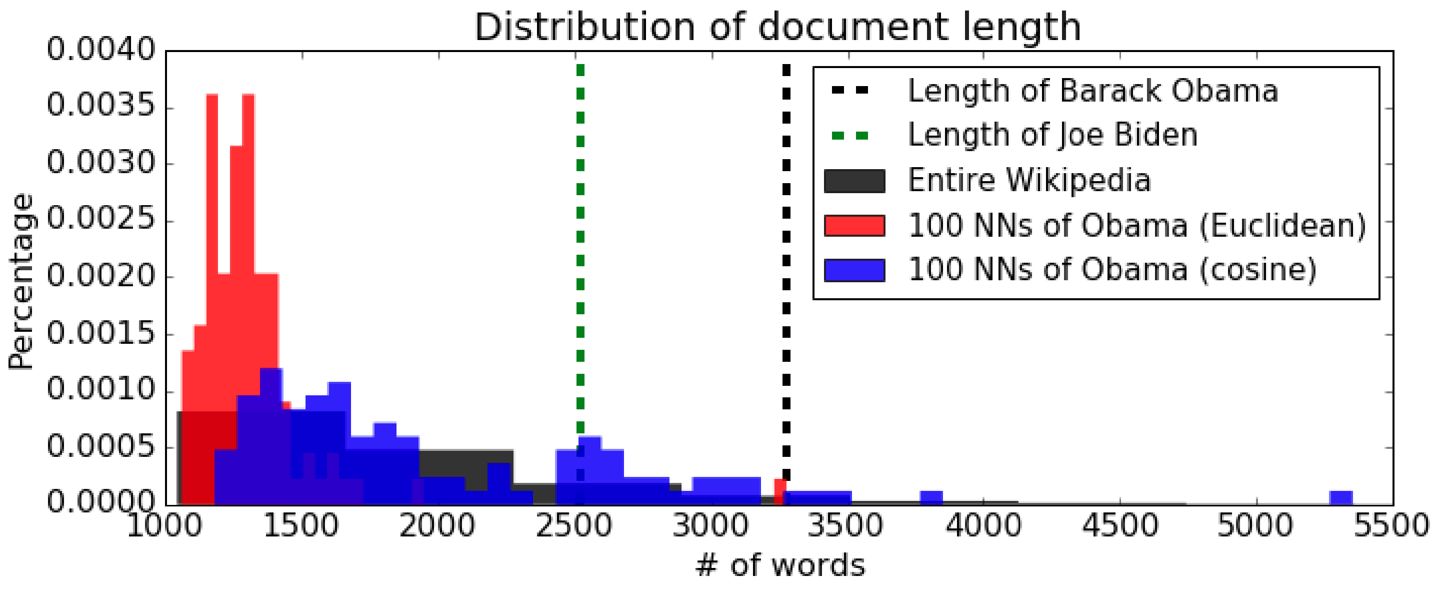
which prints

```
+-------+------------------------+--------+-------------------+
|   id  |          name          | length |      distance     |
+-------+-------------------------+--------+---------------------+
| 35817 |       Barack Obama      |  3278  | -1.11022302463e-15 |
| 24478 |        Joe Biden        |  2523  |   0.703138676734   |
| 38376 |      Samantha Power     |  1911  |   0.742981902328   |
| 57108 |  Hillary Rodham Clinton |  3472  |   0.758358397887   |
| 38714 | Eric Stern (politician) |  1589  |   0.770561227601   |
| 46140 |       Robert Gibbs      |  1572  |   0.784677504751   |
|  6796 |       Eric Holder       |  1430  |   0.788039072943   |
| 44681 |  Jesse Lee (politician) |  1374  |   0.790926415366   |
| 18827 |       Henry Waxman      |  1607  |   0.798322602893   |
|  2412 |     Joe the Plumber     |  1422  |   0.799466360042   |
```

From a glance at the above table, things look better. For example, we now see Joe Biden as Barack Obama's nearest neighbor! We also see Hillary Clinton on the list. This list looks even more plausible as nearest neighbors of Barack Obama.

Let's make a plot to better visualize the effect of having used cosine distance in place of Euclidean on our TF-IDF vectors.

```python
plt.figure(figsize=(10.5,4.5))
plt.figure(figsize=(10.5,4.5))
plt.hist(wiki['length'], 50, color='k', edgecolor='None', histtype='stepfilled', normed=True,
         label='Entire Wikipedia', zorder=3, alpha=0.8)
plt.hist(nearest_neighbors_euclidean['length'], 50, color='r', edgecolor='None', histtype
    ='stepfilled', normed=True,
         label='100 NNs of Obama (Euclidean)', zorder=10, alpha=0.8)
plt.hist(nearest_neighbors_cosine['length'], 50, color='b', edgecolor='None', histtype
    ='stepfilled', normed=True,
         label='100 NNs of Obama (cosine)', zorder=11, alpha=0.8)
plt.axvline(x=wiki['length'][wiki['name'] == 'Barack Obama'][0], color='k', linestyle='--',
    linewidth=4,
           label='Length of Barack Obama', zorder=2)
plt.axvline(x=wiki['length'][wiki['name'] == 'Joe Biden'][0], color='g', linestyle='--',
    linewidth=4,
           label='Length of Joe Biden', zorder=1)
plt.axis([1000, 5500, 0, 0.004])
plt.legend(loc='best', prop={'size':15})
plt.title('Distribution of document length')
plt.xlabel('# of words')
plt.ylabel('Percentage')
plt.rcParams.update({'font.size': 16})
plt.tight_layout()
```



Indeed, the 100 nearest neighbors using cosine distance provide a sampling across the range of document lengths, rather than just short articles like Euclidean distance provided.

Moral of the story: In deciding the features and distance measures, check if they produce results that make sense for your particular application.

Problem with cosine distances: tweets vs. long articles

Happily ever after? Not so fast. Cosine distances ignore all document lengths, which may be great in certain situations but not in others. For instance, consider the following (admittedly contrived) example.

```
+--------------------------------------------------------+
|                                            +--------+  |
|  One that shall not be named               | Follow |  |
|  @username                                 +--------+  |
|                                                        |
|  Democratic governments control law in response to     |
|  popular act.                                          |
|                                                        |
|  8:05 AM - 16 May 2016                                 |
|                                                        |
|  Reply   Retweet (1,332)   Like (300)                  |
|                                                        |
+--------------------------------------------------------+
```

How similar is this tweet to Barack Obama's Wikipedia article? Let's transform the tweet into TF-IDF features, using an encoder fit to the Wikipedia dataset. (That is, let's treat this tweet as an article in our Wikipedia dataset and see what happens.)

```
tweet = {'act': 3.4597778278724887,
 'control': 3.721765211295327,
 'democratic': 3.1026721743330414,
 'governments': 4.167571323949673,
 'in': 0.0009654063501214492,
 'law': 2.4538226269605703,
 'popular': 2.764478952022998,
 'response': 4.261461747058352,
 'to': 0.04694493768179923}
```

Let's look at the TF-IDF vectors for this tweet and for Barack Obama's Wikipedia entry, just to visually see their differences.

```
word_indices = [map_index_to_word[map_index_to_word['category']==word][0]['index'] for word in
    tweet.keys()]
tweet_tf_idf = csr_matrix( (tweet.values(), ([0]*len(word_indices), word_indices)),
                          shape=(1, tf_idf.shape[1]) )
```

Now, compute the cosine distance between the Barack Obama article and this tweet:

```
from sklearn.metrics.pairwise import cosine_distances

obama_tf_idf = tf_idf[35817]
print cosine_distances(obama_tf_idf, tweet_tf_idf)
```

Let's compare this distance to the distance between the Barack Obama article and all of its Wikipedia 10 nearest neighbors:

```
distances, indices = model2_tf_idf.kneighbors(obama_tf_idf, n_neighbors=10)
print distances
```

With cosine distances, the tweet is "nearer" to Barack Obama than everyone else, except for Joe Biden! This probably is not something we want. If someone is reading the Barack Obama Wikipedia page, would you want to recommend they read this tweet? Ignoring article lengths completely resulted in nonsensical results. In practice, it is common to enforce maximum or minimum document lengths. After all, when someone is reading a long article from *The Atlantic*, you wouldn't recommend him/her a tweet.