Tim Healy

# Python Pseudo-Code

The overarching idea for my project is to use nonlinear dimensionality reduction, via diffusion maps, to uncover semantic relationships between different articles of text data. I'm interested in applying this idea to twitter data to understand what Politicians are talking about. More specifically, I will cluster dimensionally reduced data to see if semantically linked topics emerge.

STEP 1 - Get Tweets

Using the Twitter API, download csvs of tweets and metadata for each politician to a local directory. Construct Term-Document (TD) Matrix to represent the tweets in a Matrix -> https://en.wikipedia.org/wiki/Document-term_matrix

```
class ParseTweets():
    def __init__():
        '''
        Initialize object with a Twitter API key
        TODO: Initialize an object for every Politician?
        '''
    def get_politicians(self):
        '''
        This method will return a list of politicians
        returns list object
        '''
    def get_tweets(self, politician):
        '''
        Gets tweets for inputted politician
        TODO: figure out if date range should be implemented
        returns list object
        '''
    def preprocess_(self, text):
        '''
        Some preprocessing will most likely need to be done to
        remove stop words, stemming, perhaps even by using synonyms
        TODO: NLTK library
        '''
    def construct_td_matrix(self, tweets):
        '''
        Construct a TD Matrix from inputted tweets data
        TODO: Implement term frequency - inverse document frequency
              (tf-idf) statistic
        '''
    def main(self):
        '''
        Executes the program
        TODO: Explore efficiency of using loop vs. generator object
```

```
        '''
        self.politicians = get_politicians()
        for politician in self.politicians:
            tweets = get_tweets(politician)
            tweets_td = construct_td_matrix(tweets)
```

STEP 2 - Dimensionality Reduction

Using Diffusion Maps or PCA, reduce the dimensionality of the Term-Document Matrix generated from the `ParseTweets` object.

```
class PCA():
    def __init__():
        '''
        Initialize object with a TD Matrix
        TODO: could potentially use library since this isn't the project focus
        '''
    def SVD():
        '''
        Conduct Singular Value Decomposition (SVD) on TD Matrix
        TODO: decide whether SVD needs it's own object
              find efficient SVD library
        return U, sigma, V
        '''
    def reduce(self, n):
        '''
        Reconstruct TD Matrix using n principal components
        return reconstructed TD Matrix
        '''
    def main():
        svd = SVD(td_matrix)
        svd.SVD()
        svd.reduce(n)
```

```
class DiffusionMap():
    def __init__():
        '''
        Initialize object with a TD Matrix
        '''
    def calculate_similarity(self):
        '''
        Calculate stochastic / markov similarity matrix which measures the
        distance between points
        TODO: implement cosine similarity as distance metric
        '''
    def normalize(self):
        '''
        Normalize and transform the matrix
        '''
    def decomposition(self, n):
        '''
        Calculate eigenvalues and reduce normalized similarity matrix
        return diffusion map
        TODO: SVD?
        '''
    def reduce(self):
        '''
        Use diffusion map to calculate the lower dimensional manifold
        '''
    def main():
        dm = DiffusionMap(td_matrix)
        dm.calculate_similarity()
        dm.normalize()
        dm.decomposition(n)
        dm.reduce()
```

STEP 3 - Cluster and Visualize the Data

The dimensionally reduced data will then be clustered, most likely using the K-means algorithm, to see if any topic clusters emerge. Will also explore deriving important terms from groups of text.

```
class Cluster():
    def __init__:
        '''
        Initialize object with a dimensionally reduced matrix
        '''
    def k_means(self):
        '''
        Implement k-means clustering algorithmn
        '''
```