Blog post 5: Singular Value Decomposition

Link to using PCA for dimensionality reduction:
https://codingmachinelearning.wordpress.com/2016/06/14/principal-component-analysis-pca/

This post, we will see how to use SVD for reducing dimension. Before that we need to understand the problems associated with PCA so as to appreciate SVD. PCA computes eigenvalues to capture variance. But one cannot compute eigenvalues for non-square matrix.

In SVD we do not have eigenvectors. Instead we have left and right singular vectors and SVD is possible even on non-square matrices. The only bottleneck is the fact that it is a quadratic time algorithm.

The latest thing in NLP is using word vectors for representing word such that the semantics and the context of the word is preserved. The notion of inner product in the vector space triggers the idea of similarity. But if we use one-hot encoding method, computing inner product does not represent similarity. In one hot encoding we represent each word as a vector of 0's and 1's where 1 occurs when the word is considered.

For example we have the following 5 words in the dictionary/vocabulary: I, like, motel, hotel, enjoy
The one-hot representation for like will be [0,1,0,0,0] and for enjoy will be [0,0,0,0,1] and so on. Here if we take inner product between hotel and motel which is [0,0,0,1,0] * [0,0,1,0,0] it will tell they are not similar but in reality they are. How to capture this notion. Though we may be solving this problem here, we will use SVD to aid us in this process.

Consider the following 3 sentences in our vocabulary:

1. I like deep learning.
2. I like NLP.
3. I enjoy flying.

```
[
 [0, 2, 1, 0, 0, 0, 0, 0],
 [2, 0, 0, 1, 0, 1, 0, 0],
 [1, 0, 0, 0, 0, 0, 1, 0],
 [0, 1, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 1],
 [0, 1, 0, 0, 0, 0, 0, 1],
 [0, 0, 1, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 1, 1, 1, 0]
]
```

We can build a co-occurrence matrix (by taking 1 window context around each word) and to reduce dimension and to capture the word similarity we will use SVD.Code for computing SVD and displaying the plot is also provided in this post.

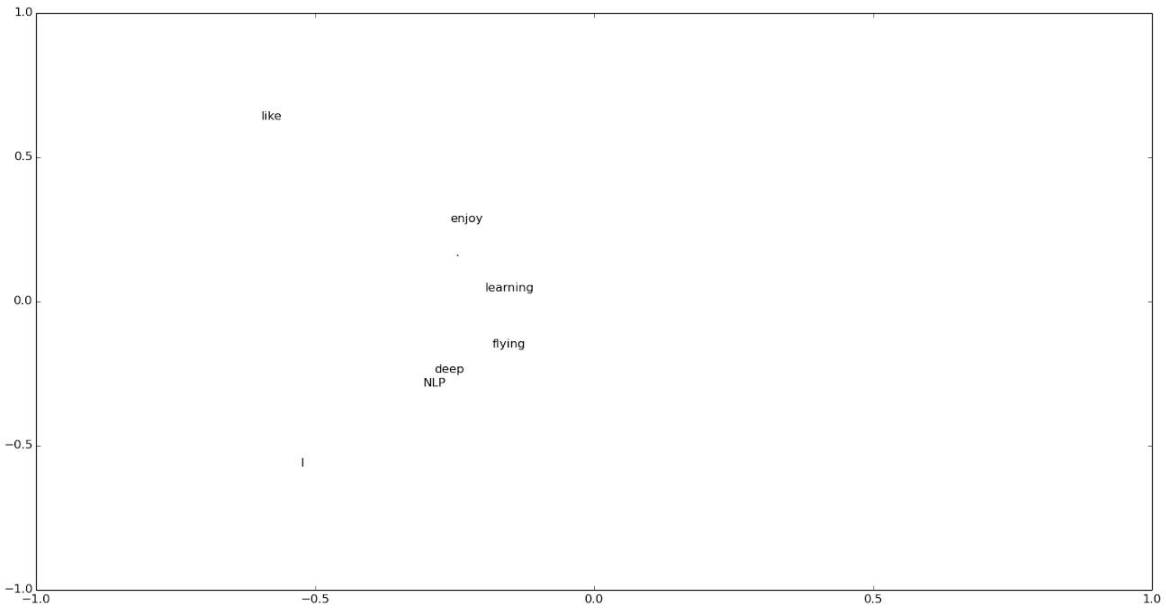
The numpy svd function returns 3 outputs. U, s and v. U contains left singular vectors(need not be a square matrix), s is matrix that contains eigenvalues and v contains the right singular vectors (again, need not be square)

We can decompose the co-occurrence matrix as x=u*s*v . Based on the eigenvalues in s matrix we can reduce dimension by taking k most significant vectors. Thus we can see how SVD can also be used as dimensionality reduction tool.

```
u,s,v=la.svd(x,full_matrices=False)
```

```
for i in xrange(len(words)):
    plt.text( u[i,0], u[i,1], words[i] )
```

When we visualize the word vectors we obtained by taking svd of the co-occurrence matrix we get the following output



We can see that enjoy is closed to like than flying and so on. The dependency is preserved just by choosing the left singular vector and plotting them.

Summary:

- We saw a cool application of SVD in word vector representation.
- SVD is not restricted to square matrices and gives out left and right singular vectors than just eigenvectors
- Using the eigenvalues in the s matrix of u,s,v decomposition and thresholding based on the user's need we can select first k singular vectors which helps in dimensionality reduction
- SVD is quadratic time algorithm $O(N^2)$. On large matrices computational time will increase drastically

Next Post:

In the next series of posts I will be writing about classifiers in general. That is what are the various classifiers we have, and how they behave, what type of decision boundary they represent and so on.

***Code** for executing the above application is made publicly available on github*.Please follow the below link to the code and to the visualization output. Thank you.

Link: https://github.com/vsuriya93/coding-machine-learning/tree/master/SVD