Shivank Goel
NetId : sg2359

Homework 1

CUR Decomposition

CUR decomposition is one of the ways of dimensionality reduction of a given matrix A (of m objects and n features) just like svd. The goal is to get a compressed representation of A that may be easier to analyze or interpret using the field specific knowledge. This method is especially useful for sparse matrices A. We choose set of columns C and a set of rows R, which play the role of U and V in SVD. We can pick any number of rows and columns. The choice of rows and columns is made using the algorithm as described in [1]. CUR is always an approximate approximation of matrix A, measured in in terms of forbenius norm, however CUR serves some advantages over svd in situations like:

1. Interpretation : SVD is difficult to interpret - jth column of matrix A is a linear combination of left singular vectors and singular values. We can estimate A as linear combination of top k left singular vectors. However, interpreting the singular vectors is difficult as they themselves may lack any meaning in terms of the field from which data is drwan. CUR decomposition is interpretable to the extend the data is interpretable by the practitioner of the fields.

2. Sparsity : If matrix A is very sparse, still the U, V matrices obtained via svd of $A = U\Sigma V^T$ can be dense. Although Σ is sparse but it's usually small. However, CUR decomposition picks up the exact columns and rows of matrix A and hence maintains the sparsity in C and R, whereas U is dense and small. Thus overall we can store A's approximation using lesser space than svd.

Python code for ColSelect

```
import numpy as np
import random
from numpy.linalg import pinv as inv

def decision(probability):
    return random.random() < probability

def colselect(A,k,row = False,eps=1):
    c = (k*np.log(k)) / (eps*eps)
    m,n = A.shape[0], A.shape[1]
    u,s,vh = np.linalg.svd(A, full_matrices = False)
    vh = vh[:k,:]
    probs = (1/k)*(vh**2).sum(axis=0)
    probs = [min(1,c*p) for p in probs]
    idxs = [decision(p) for p in probs]
    cols  = A[:,idxs]
    included_idx = [i for i in range(n) if idxs[i]]
    if row:
            return cols.T,included_idx
    return cols,included_idx
```

Python code for CUR decomposition

```
def cur_decompose(A,k,e=1, return_idx = False):
    m,n = A.shape[0], A.shape[1]
    if k>min(m,n):
            return [],[],[]
    C, included_cols = colselect(A,k,False,eps=e)
    R, included_rows = colselect(A.T,k,True,eps=e)
    U = inv(C) @ A @ inv(R)
    if return_idx:
            return C,U,R,included_cols,included_rows
    return C,U,R
```

Dataset

For demonstrating the usefulness of the CUR decomposition we will be using the movie ratings dataset from the https://movielens.org/ website provided at https://grouplens.org/datasets/movielens/100k/. The dataset contains 100,000 ratings from 943 users on 1682 movies. The 'usermovies' matrix of dimension 943 x 1682 is quite sparse since it contains only 6.3% (100,000/1,586,126) non zero entries. Each rating is an integer from 1 to 5. The code to create 'usermovies' matrix from above source is as follows:

```
def getdata():
    file = open('u.data','r')
    usermovies = np.zeros((943,1682))
    entries = [[int(t) for t in k.split('\t')] for k in file.readlines()]
    for e in entries:
            usermovies[e[0]-1,e[1]-1] = e[2]
    return usermovies

usermovies = getdata()
```

SVD Analysis

Below is the code for getting forbenius norm reconstruction errors for all the first k rank approximations of the given matrix upto a given rank. We will use it to compare reconstruction error of CUR decomposition. The Figure : 1 shows the best rank-k approximation errors on our 'usermovies' data for k values ranging from 1 to 10 for both CUR and SVD. The svd forbenius norm error decreases from 980.96 for k = 1 to 854.98 for k = 10.

```
def give_error(A,B):
    return np.sqrt(((A-B)**2).sum())

def give_svd_results(A,upto=10):
    errors = []
    ks = [i for i in range(1,upto+1)]
    a,b,c = np.linalg.svd(A)
    for k in ks:
            errors.append(give_error(A,a[:,:k]@np.diag(b[:k])@c[:k,:]))
    return errors
```

CUR Analysis

Here we will analyse the results after CUR decomposition. This decomposition ensures that the error $\|A - CUR\|_F$ is at most $(2+\epsilon)$ times the error of the best rank k approximation. For the sake of simplicity I will choose $\epsilon = 1$ for all of my CUR decomposition. Since, CUR is a randomized algorithm, I will run the CUR algorithm multiple times (N=10) for same value of k, to try not to get unlucky and get bad results.

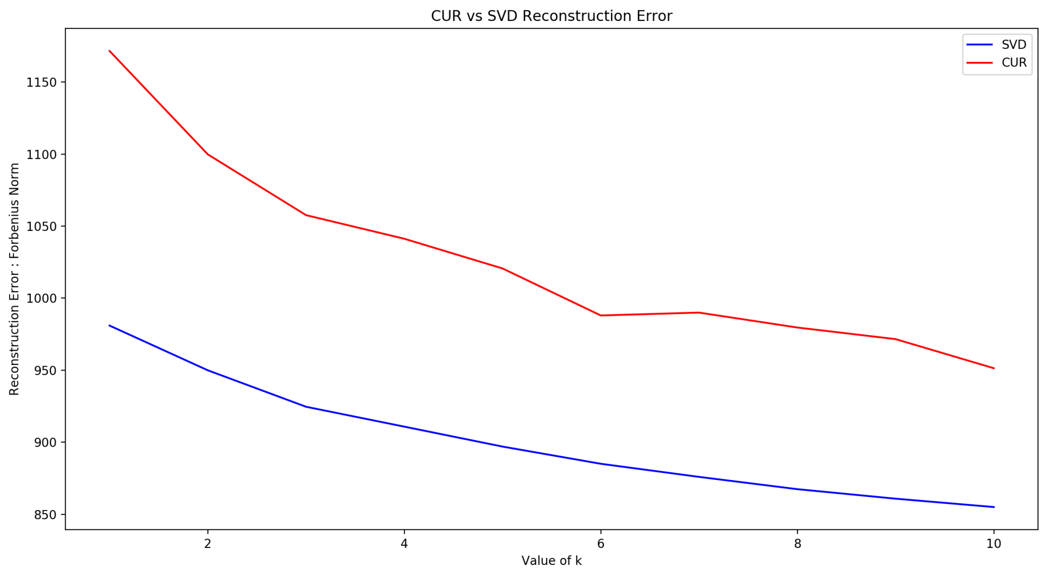
```
def give_cur_vals(A,k,N=10):
    c,u,r,cols,rows  = cur_decompose(A,k,return_idx = True)
    err = give_error(A,c@u@r)
    for i in range(N):
            ctmp,utmp,rtmp,c_tmp,r_tmp=cur_decompose(A,k,return_idx =True)
            err_temp = give_error(A,ctmp@utmp@rtmp)
            if err_temp < err:
                    err = err_temp
                    c = ctmp
                    u = utmp
                    r = rtmp
                    cols = c_tmp
                    rows = r_tmp
    return c,u,r,err,cols,rows

def give_cur_results(A,upto=10):
    errors = []
    ks = [i for i in range(1,upto+1)]
    for k in ks:
            a,b,c,err,rows,cols = give_cur_vals(A,k)
            errors.append(err)
    return errors
```

Python code plotting CUR and SVD reconstruction errors

```
    def plot_cur_and_svd_error(A,upto=10):
        errs1 = give_svd_results(A,upto)
        errs2 = give_cur_results(A,upto)
        x = [i for i in range(1,upto+1)]
        plt.plot(x,errs1,'b-',label='SVD')
        plt.plot(x,errs2,'r-',label='CUR')
        plt.xlabel(" Value of k ")
        plt.ylabel("Reconstruction Error : Forbenius Norm")
        plt.title("CUR vs SVD Reconstruction Error")
        plt.legend(loc = 'best')
        plt.show()
```

Figure 1: Reconstruction Error



Observations

Let's compare the SVD and CUR results for k = 5. For k=5 we get a forbenius error of 1004.67 using CUR decomposition. The C(943 x 16) matrix and R(11 x 1682) matrix obtained are very sparse with 20.24% and 13.2% non zero entries respectively. The U(16*11) matrix obtained is a fully dense matrix with no zero entries. The decomposition is quite interpretable as the C matrix contains columns with indices [9, 21, 54, 56, 178, 190, 227, 279, 293, 299, 332, 345, 464, 506, 525, 674] which corresponds to the movies as described in the appendix A and R matrix contains rows with indices [81, 84, 89, 129, 388, 421, 430, 434, 746, 773, 911] which can be similarly mapped to the users' actual names. The selection of rows and columns are made using the leverage scores as described in [1]. We can see that the movies selected by the algorithm are across various genre like 'Richard III' - Political Drama, 'Braveheart' - Action , 'The Professional' - Mystery , 'Priest' - Fantasy and Science Fiction, 'Up Close and Personal' - Romance. Hence the selected columns can be well representative of the whole data set.

Using the truncated SVD decomposition with k = 5 we get the forbenius norm error of 896.95. For the decomposition, $usermovies = U\Sigma V^T$ The U matrix (943 x 5) and V matrix (5 x 1682) are fully dense matrices. The Σ (5 x 5) matrix is a diagonal matrix consisting of five entries i.e. 20% non zero entries. Interpreting U and V is difficult since they are some linear combinations of the actual columns and rows, unlike in the case of CUR decomposition.

References

[1] M. W. Mahoney and P. Drineas, "Cur matrix decompositions for improved data analysis," *Proceedings of the National Academy of Sciences*, vol. 106, no. 3, pp. 697–702, 2009.

A CUR Decomposition Selected Movies

1. Richard III (1995)
2. Braveheart (1995)
3. Professional The (1994)
4. Priest (1994)
5. Clockwork Orange A (1971)
6. Amadeus (1984)
7. Star Trek: The Wrath of Khan (1982)
8. Up Close and Personal (1996)
9. Liar Liar (1997)
10. Air Force One (1997)
11. Game, The (1997)
12. Jackie Brown (1997)
13. Jungle Book, The (1994)

14. Streetcar Named Desire, A (1951)
15. Ben-Hur (1959)
16. Nosferatu (Nosferatu, eine Symphonie des Grauens) (1922)