THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE CLASSIFICATION OF DOCUMENTS USING THE SUBSPACE RELEVANCE MODEL

PATRICK HENRY
FALL 2013

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Daniel Kifer
Assistant Professor, Department of Computer Science
Thesis Supervisor

Jesse Barlow
Professor, Department of Computer Science
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The document classification problem is one that identifies how we can efficiently take documents about various subjects and automatically assign them to a specific category. This paper considers the subspace relevance model as a way of classifying documents. Using the steepest descent method over the Stiefel and Grassmannian manifolds, we are able to find the local minimum of a specified function. Our goal is to create an algorithm that will perform document classification efficiently and effectively.

TABLE OF CONTENTS

List of Figures iii

Acknowledgements iv

Chapter 1

Introduction.....1

Chapter 2

The Stiefel and Grassmann Manifolds.....4

Chapter 3

Steepest Descent Algorithm.....6

Chapter 4

The Objective Function.....11

Chapter 5

Conclusion.....12

Appendix A Codes and Algorithms.....13

A.1 The Steepest Descent Method on the Stiefel Manifold.....13

A.2 Python Code for Steepest Descent Method on the Stiefel Manifold (stiefel_descent.py).....13

A.3 Code for the scaling experiments (time_compute.py).....16

A.4 Accuracy test of stiefel_descent.py (accuracyTest.py).....19

Bibliography.....22

Academic Vita.....23

LIST OF FIGURES

Figure 3-1: Running time of time_compute.py vs. n...7

Figure 3-2: Running time of time_compute.py vs. p...8

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Daniel Kifer, for his guidance throughout the process of writing this thesis. He has been a great teacher of the complicated concepts involved in the topic of my thesis, and an understanding mentor as I juggled my coursework with this assignment. I would also like to thank my honors adviser, Jesse Barlow, for his insights throughout my 4 and a half years here at Penn State. It is with his help that I am able to move on in this world and graduate with honors.

Chapter 1

Introduction

Every day, billions of emails and newsfeeds exist across the world. Helping users be able to sort through large amounts of information is what motivates the subject of document clustering and classification. What powers features such as Rich Site Summary (RSS) feeds and the sorting of emails is the ability to cluster documents into disjoint sets. For example, an email client should be able to send a message from a relative and place it into your inbox, but send harmful and/or relatively useless messages to your spam folder. Likewise, an RSS feed should classify what is relevant to the user in order to avoid spending time wading through unnecessary or unwanted information. Relevance can involve a number of factors: for example, the number of times a keyword appears in a document or website, or whether or not the keyword appears in the title. What our research looks into is an algorithm that can perform document clustering, and how we can potentially make it faster.

The algorithm will use an objective function which will take in many documents as input. Its output will be the cluster to which each document would belong. One way to represent the documents could be to create vectors that represent how many times a word appears in each document. Clusters or categories would be represented as vector subspaces. A document's relevance to a cluster or category is measured as the length of the document vector when projected onto the subspace representing the cluster/category.

A document is assigned to the cluster or category for which it has the highest relevance. These vectors would be used to assign a document to an appropriate cluster. By doing this, we can represent each category of documents and use it in a model for classification. What we would hope to accomplish is to improve the speed of the current algorithm used to identify clusters. As a result, our research will involve testing optimization approaches using the Stiefel and Grassmann manifolds. These two manifolds reduce the dimension of the optimization problem and allow for faster convergence (Manton 2001).

The following chapter will explain the significant properties of each manifold, as well as how they will be used in our research. Though the papers that form the basis of our research deal with the complex manifolds, we have kept our focus in the real plane since document classification has no place in the complex world (Manton 2001). What will be explained is how this paradigm does not change the direction of our research very much.

Chapter 3 will explore the optimizations used in a variation of the steepest descent algorithm using each manifold, and compare them with the original performance. Relatively simple Linear Algebraic manipulation allows the running speed of our algorithm to rely less on the number of unique words that appear in a document and more on the number of documents involved, which will almost always be significantly lower.

Chapter 4 will examine the objective function used that we want to apply to the relevance model. This function will take into account the vectors that represent the documents and a matrix that represents a cluster in order to calculate a scalar signifying the relevance.

Document clustering is an important problem in both managing documents whose subject matter can vary and being able to recommend documents that relate to any particular one. While our attempts are not guaranteed success, part of our learning process will be experimenting with our tuned optimizations and analyzing the results in order to pick a meaningful direction going forward.

Chapter 2

The Stiefel and Grassmann Manifolds

Our main source, a paper written by Jonathan H. Manton, defines the Stiefel manifold as follows:

$$St(n,p) = \{ X \in \mathbb{R}^{nxp} : X^T X = I_p \}$$

That is, it is the set of n x p matrices with orthonormal columns. We use this manifold to minimize our objective function by way of a method known as steepest descent (Manton 2001). First, we take any matrix $X \in \mathbb{R}^{nxp}$ with rank *p*. We then define an operator π which takes a point from the same n x p real plane and projects it onto the Stiefel manifold. It can be evaluated as the minimum value of $\|X - Q\|^2$ as given by some matrix Q that lies on the manifold. Take a point X that lies on the manifold. We call the tangent space at X the set of directions tangent to the manifold at that point. Part of the algorithm described by Manton involves choosing a point and a direction from the tangent space. The point is perturbed in the direction chosen before projecting back onto the manifold (Manton 2001). This tangent space will become the parameter used to create a faster convergence.

To optimize on the manifold, following Manton, we write our problem in terms of a parameterization for each iteration of the algorithm. In our case, one takes a point X and map it from a subset of the tangent space to a point on the Stiefel manifold with one condition: any point $Y \in St(n, p)$ that is close to X can be written as Y = h(Z), where h is the mapping. While there are various ways to define h, following Manton we will use the projection operator π. So, h(Z) = π(X + Z). Therefore the objective function is defined as:

$$g(Z) = f(h(Z)) = f(\pi(X + Z)).$$

The *p* columns of any X on the Stiefel manifold will form an orthonormal basis for a subspace of the same dimension, denoted by $\lfloor X \rfloor$. It is called the Grassmann manifold. For any two points $X, Y \in St(n, p)$, they are equivalent if a unitary matrix $S \in \mathbb{R}^{pxp}$ exists that allows the following statement to be true: Y = XS. As a result, a one-to-one relationship exists between points on the Grassmann manifold and an equivalence class on the Stiefel manifold (Manton 2001).

Both of these manifolds hold tremendous significance in making the algorithm converge much faster, because it is able to parametrize them in ways that are both simple and efficient. By taking advantage of this and simple Linear Algebra, we will implement a Steepest descent method and tune it to reduce the number of operations needed, in order to scale to high-dimensional spaces (Manton 2001).

Chapter 3

Steepest Descent Algorithm

Steepest descent is a first-order optimization algorithm. Below, Manton's adaptation of steepest descent is described in psuedocode:

```
function Steepest_Descent
Choose X on the manifold
Set step size $\gamma$ to 1
Compute $D_X$, the derivative of f (a given function)
Compute Z, the descent direction
Calculate the inner product <Z, Z>
    If $\sqrt{\langle Z, Z\rangle}$ is sufficiently small then
        stop
While $f(X) - f(\pi(X + 2\gamma Z)) \geq \gamma\langle Z, Z\rangle$ do
    Set $\gamma$ to $2\gamma$
While $f(X) - f(\pi(X + \gamma Z)) < \frac{1}{2}\gamma\langle Z, Z\rangle$ do
    Set $\gamma$ to $\frac{1}{2}\gamma$
Set X equal to the projection of X + sZ. Return to the line
computing the derivative of X.
```

In the context of the algorithm we are using, the direction of the descent is defined as follows:

$$Z^{(sd)} = XD_X^H X - D_X$$

where D is the derivative of our function f at X (Manton 2001). See Appendix A for the full algorithm.

We simplify some of the multiplications of matrices made in evaluating the inner product as a way to tune the algorithm. The inner product, calculated in step 4, involves the initial quantity $Z^T(I_p - \frac{1}{2}XX^T)Z$. Using the distributive law, we obtain the quantity $(Z^T Z - \frac{1}{2}Z^T X X^T Z)$. Since our X is such that $X^T X = I_p$, we can simply further to $(Z^T Z - \frac{1}{2}Z^T Z) = (\frac{1}{2}Z^T Z)$. As a result, we are able to reduce the operations from $O(p^2n)$ to $O(pn)$. We ran scaling experiments that would create randomized matrices with varying dimensions of *n* and *p* and recorded the amount of time taken to calculate the derivative and direction of each one. See Figures 1 and 2 below for the running times matched against the number of rows and the number of columns, respectively.

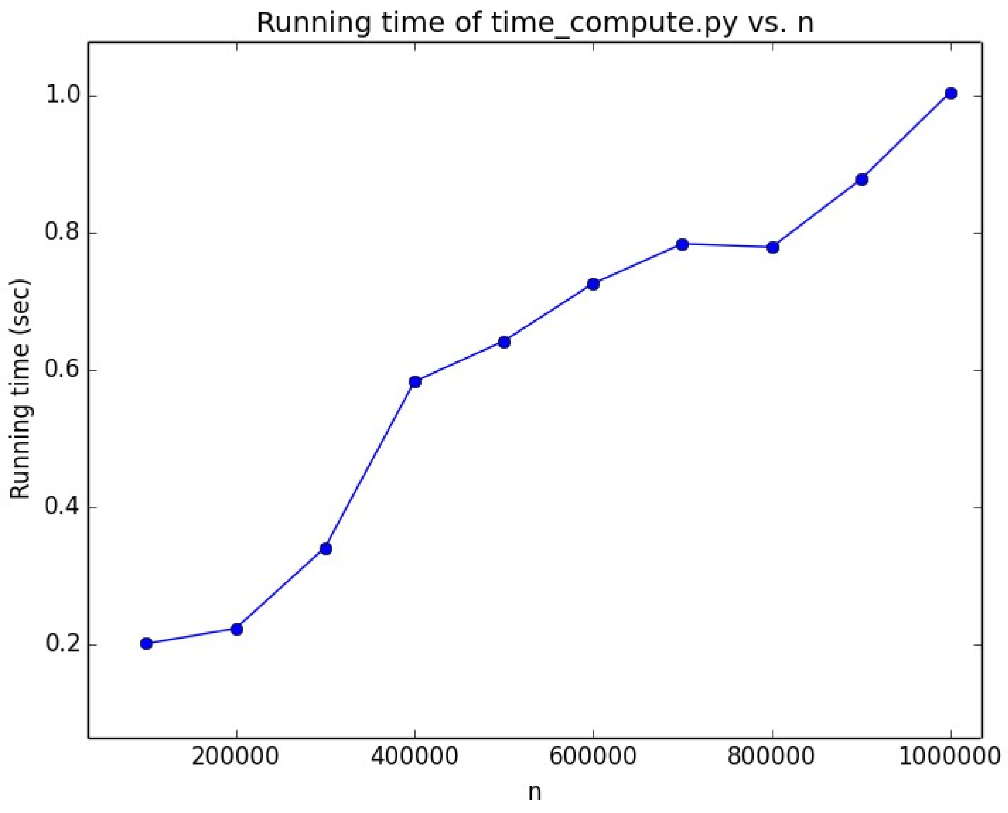

Figure 3-1: Running time of time_compute.py vs. n

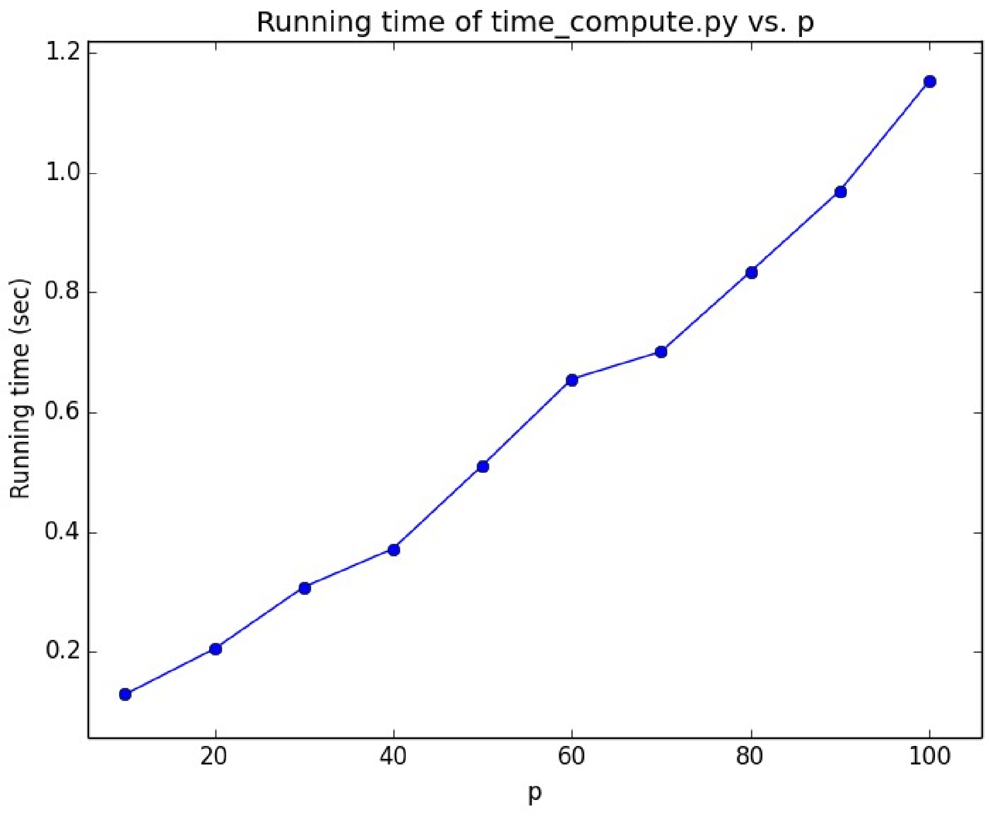

Figure 3-2: Running time of time_compute.py vs. p

For Figure 3-1, the values of n begin at 100,000 and increase by increments of 100,000 up to 1,000,000. The value of p is held constant at 10. In Figure 3-2, the values of p begin at 10 and increment by 10 up to 100 while n is held constant at 100,000. The code calculates the derivative and direction for each pair of p and n once. Both figures depict a single run of the code. The polynomial used for taking the derivative was $f(x)=x^2+12\text{x}+36$, and was arbitrarily chosen.

The code was written initially in MATLAB, but was later ported to Python, using the robust libraries numpy and matplotlib. Using this code would help us accomplish our goal, but would first require sample data to further test its efficiency and accuracy. Consult Appendix A for the Python code.

Chapter 4

The Objective Function

Our objective function will be the following:

$$\sum \log(\|v_i^T W\|^2)$$

where v_i^T is the transpose of the ith document's vector, and W is a matrix that represents a cluster. We want to apply this function to the relevance model (van Rijsbergen).

$\|v_i^T W\|^2$ represents the square of the 2-norm of the product inside. Our algorithm for the stiefel descent is used to find the corresponding W that fits our vector v_i .

Chapter 5

Conclusion

After examining the steepest descent method in the Stiefel manifold, we have found that our tuned optimizations of Manton's original algorithm converged faster. This is a great boon considering that document handling would usually deal with millions upon millions of documents, which would result in just as many vectors. If one were to again consider the number of Google searches per day and what would happen if each search had to take several minutes, Google would not be anywhere near as popular as it is.

A next step would be to apply this algorithm to sample data while using specific document clustering algorithms. From what we have evaluated in our current research, we can analyze a clustering algorithm anew, with fewer instructions and faster convergence. This would allow for greater control over automated document management and recommendation.

It is worth noting again that our research does not guarantee success, but rather the ability to either rule out incorrect methodologies or improve on what we already know. While our testing has dictated a faster convergence, there could be possible errors down the line in the algorithm. Regardless, the only way to know would be to continue down the path we have already carved, and choose the way that works best.

Appendix A

Code and Algorithms

This appendix will hold both the code that I have personally written as well as any algorithms originally conceived both in research and practice.

A.1 The Steepest Descent Method on the Stiefel Manifold (Manton 2001)

1. Choose $X \in \mathbb{R}^{nxp}$ such that $X^H X = I$. Set step size γ := 1.
2. Compute D_X, the derivative of f at X.
3. Compute the descent direction Z := $X D_X^H X - D_X$.
4. Evaluate the inner product <Z, Z> = tr{Z^H (I - $\frac{1}{2}$ XX^H) Z}. If $\sqrt{\langle Z, Z \rangle}$ is sufficiently small, then stop.
5. If $f(X) - f(\pi(X + 2\gamma Z)) \geq \gamma \langle Z, Z \rangle$ then set γ := 2γ and repeat Step 5.
6. If $f(X) - f(\pi(X + \gamma Z)) < \frac{1}{2}\gamma \langle Z, Z \rangle$ then set γ := $\frac{1}{2}\gamma$ and repeat Step 6.
7. Set X := $\pi(X + \gamma Z)$. Go to Step 2.

A.2 Python Code for Steepest Descent Method on the Stiefel Manifold (stiefel_descent.py)

Note: This code follows A.1's algorithm, but also includes optimizations used to improve the running time. It also records the amount of time taken to perform Steps 2 and 3 of the steepest descent.

```
#Programmed by Patrick Henry

#! /usr/bin/python

import numpy

import math

import time

def stiefel_descent(X, deriv, func):

      #Step 1 of algorithm

      step = 1

      #Get sizes of matrix

      y = X.shape

      while true:

            time1 = time.time()

            #Step 2 - calculate derivative of func at X

            D = deriv(func, X)


            #Step 3

            A = transpose(D)

            Z = X*(A*X) - D

            time2 = time.time() - time1


            #Step 4
```

```
#Original matrix: Z'*(I - 0.5*X*X')*Z
B = transpose(Z)
trMatrix = 0.5*B*Z
outerProduct = numpy.trace(trMatrix)
f = outerProduct

# If <Z,Z> is sufficiently small, then stop
if math.sqrt(outerProduct) > 10^(-4):
      #Calculating projection of (X + 2*gamma*Z)
      U, s, V = np.linalg.svd(X + 2*step*Z)
      proj1 = U*np.eye(y[0],y[1])*transpose(V)

      #Step 5 of the algorithm
      while (func(X) - func(proj1)) >= step*outerProduct:
            step = 2*step;

      # Calculating projection of (X + gamma*Z)
      U, s, V = np.linalg.svd(X + step*Z)
      proj2 = U*np.eye(y[0],y[1])*transpose(V)

      # Step 6
      while (func(X) - func(proj2)) < .5*step*outerProduct:
            step = .5*step;
            # Step 7
            U, s, V = np.linalg.svd(X + step*Z)
            X = U*np.eye(y[0],y[1])*transpose(V)
        else:
            break
    return f
```

A.3 Code for the scaling experiments (time_compute.py)

This code was written to verify the running time of the Python code. It creates plots that analyze the amount of time taken to calculate the derivative and descent direction of randomized matrices of varying dimensions.

```
#Programmed by Patrick Henry
#! /usr/bin/python
import numpy as np
import math
import time
import random
import matplotlib.pyplot as plt

def time_compute():
    ns = np.array([100000, 200000, 300000, 400000, 500000, 600000,
700000, 800000, 900000, 1000000])
    nruns = np.zeros((10, 1))
```

```
ps = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

pruns = np.zeros((10, 1))


# coefficients of polynomial, i.e. x^2+12x+36

coeff = np.array([1, 12, 36])


for i in range(0,10):

      Y = np.zeros((ns[i], ps[0]))

      for j in range(0, ns[i]):

            for k in range(0, ps[0]):

                  Y[j,k] = random.random()


      #Record time

      time1 = time.time()


      #Calculate derivative at Y

      p = np.poly1d(coeff)

      D = np.polyval(np.polyder(p), Y)


      #calculate direction

      Z = Y*(D*Y) - D

      nruns[i] = time.time() - time1
```

```
#Plot run time vs. n

plt.plot(ns, nruns, 'bo-')

plt.ylabel('Running time (sec)')

plt.xlabel('n')

plt.axis([50000, 1050000, -0.01, nruns[9]])

plt.title('Running time of time_compute.py vs. n')

plt.show()


for i in range(0,10):

      Y = np.zeros((ns[0], ps[i]))

      for j in range(0, ns[0]):

            for k in range(0, ps[i]):

                  Y[j,k] = random.random()


      #Record time

      time2 = time.time()


      #Calculate derivative at Y

      p = np.poly1d(coeff)

      D = np.polyval(np.polyder(p), Y)


      #calculate direction

      Z = Y*(D*Y) - D
```

```
        pruns[i] = time.time() - time2


    #Plot run time vs. p

    plt.plot(ps, pruns, 'bo-')

    plt.ylabel('Running time (sec)')

    plt.xlabel('p')

    plt.axis([5, 105, -0.01, pruns[9]])

    plt.title('Running time of time_compute.py vs. p')

    plt.show()


time_compute()
```

A.4 Accuracy Test of stiefel_descent.py (accuracyTest.py)

This program tests the accuracy of the steepest descent method by generating an nxn matrix called A, then generating a symmetric positive semidefinite matrix S, and defines a function f(X) that returns the sum of $x_1^T * S * x_1 + ... + x_p^T * S * x_p$, where p is the number of columns of X. Then the steepest descent is used to find the matrix X which will maximize f(X). The solution's columns will be the top *p* eigenvectors of S.

```
    #Programmed by Patrick Henry

    #! /usr/bin/env python

    #! /usr/bin/python

    import numpy

    import math
```

```
import time

import random



def Test(n):

      A = np.zeros(n, n)

      for i in range(1,n):
            for j in range(1,n):
                  A[i,j] = random.random()

      S = transpose(A) * A

      f = stiefel_descent(X, np.polyder, function)

      return f

def function(X, S):
      sum = 0
      y = X.shape
      for i in range(0, y[0] - 1):
            sum += transpose(X[:,i])*S*X[:,i]
```

```
        return sum
```

BIBLIOGRAPHY

"Facts about Google and Competition." *Facts about Google and Competition*. Google, n.d. Web. 9 Nov. 2013.

James, I. M. "The Stiefel Manifolds." *The Topology of Stiefel Manifolds*. Cambridge [Eng.: Cambridge UP, 1976. 13-20. Print.

Manton, Jonathan H. *Modified Steepest Descent and Newton Algorithms for Orthogonally Constrained Optimisation, Part I: The Complex Stiefel Manifold*. Thesis. University of Melbourne, Parkville, Victoria 3010, Australia, 2001. Kuala Lumpur: International Symposium on Signal Processing and Its Applications, 2001. Print.

Manton, Jonathan H. *Modified Steepest Descent and Newton Algorithms for Orthogonally Constrained Optimisation, Part II: The Complex Grassmann Manifold*. Thesis. University of Melbourne, Parkville, Victoria 3010, Australia, 2001. Kuala Lumpur: International Symposium on Signal Processing and Its Applications, 2001. Print.

Van Rijsbergen, C.J. *The Geometry of Information Retrieval*. Cambridge, England: Cambridge UP, 2004. Print.

ACADEMIC VITA

Patrick Henry
pih5051@psu.edu

Education

B.S., Computer Science, 2013, Pennsylvania State University, University Park, PA

Honors and Awards

Dean's List: FA09, SP10, FA10, SP11, FA12

Recipient, Schreyer Honors College Academic Scholarship 2009-Present

Recipient, Bunton Waller Academic Merit Scholarship 2009-2013