UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales
Departamento de Computación
Métodos Numéricos

TRABAJO PRÁCTICO NÚMERO 3

Reconocimiento óptico de caracteres.

Alumnos:

Izcovich, Sabrina | sizcovich@gmail.com
Otero, Fernando | fergabot@gmail.com
Vita, Sebastián | sebastian_vita@yahoo.com.ar

Palabras Clave:

OCR - Método de Potencia - SVD - QR

Resumen:

Este trabajo consiste en un análisis sobre la implementación de un método de reconocimiento de dígitos manuscritos basado en la descomposición en valores singulares. Dicha implementación consiste en un programa encargado de leer imágenes que, a partir de la descomposición en valores singulares, logra determinar de qué dígito se trata. Para que las pruebas fueran posibles, utilizamos la base de datos MNIST que nos proporcionó una gran cantidad de imágenes de cada dígito que luego serían comparadas con el dígito a definir. Para alcanzar nuestro objetivo debíamos hallar los autovalores y autovectores de la matriz de covarianza de la base de datos. Para ello, utilizamos, en primer lugar, el Método de Potencia y, debido a que éste generaba complicaciones, optamos por QR. El estudio realizado se focalizó en los parámetros elegidos para aplicar el método que resultaron determinantes a la hora de hallar efectividad del mismo.

Índice

1. Introducción teórica

Para la realización del estudio de señales, utilizamos los siguientes conceptos:

- **Reconocimiento óptico de caracteres:** Es un proceso dirigido a la digitalización de textos. Dicho proceso consiste en identificar automáticamente símbolos o caracteres pertenecientes a un determinado alfabeto a partir de una imagen. Existen OCR muy diversos según los tipos de problemas que abordan y las funcionalidades que ofrecen: para designar el reconocimiento de caracteres manuscritos se utiliza el ICR (intelligent character recognition), para la verificación de contenidos previamente conocidos se utiliza OCV (optical character verification) y, por último, para el reconocimiento de marcas se utiliza OMR (optical mark recognition). Nuestro trabajo estará implícitamente enfocado en el ICR.
- **Covarianza:** Consiste en una medida de dispersión conjunta a un par de variables. En otras palabras, es el dato básico para determinar si existe una dependencia entre ambas variables y, además, es el dato necesario para estimar otros parámetros básicos, como el coeficiente de correlación lineal o la recta de regresión. Se encuentra representada de la siguiente manera:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{n}$$

- **MNIST:** Consiste en una base de datos de dígitos manuscritos. Dichos dígitos tienen un tamaño normalizado y se encuentran centrados en una imagen de tamaño fijo. Es utilizada para aprender técnicas y patrones de reconocimiento de datos del mundo real. En nuestro caso, utilizamos las imágenes y las etiquetas para evaluar nuestro código y lograr la obtención de resultados.
- **Método de Potencia:** Consiste en una técnica iterativa que permite determinar el valor característico dominante de una matriz, es decir, el valor característico con mayor magnitud. Dicho método es utilizado para hallar tanto autovalores como autovectores. Para aplicarlo, supusimos que la matriz A de nxn tiene n valores característicos λ_1, λ_2,...,λ_n con un conjunto asociado de vectores característicos linealmente independientes $v_1, v_2, ..., v_n$. Más aún, supusimos que A tiene exactamente un valor característico, λ_1, cuya magnitud es la mayor, por lo que $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq ... \geq |\lambda_n| \geq 0$.
- **Factorización QR:** Consiste en una factorización matricial siguiendo la siguiente propiedad: Para toda matriz A de tamaño mxn cuyas columnas forman un conjunto linealmente independiente, existe una matriz Q de tamaño mxn, cuyas columnas forman un conjunto ortonormal y una matriz triangular superior R de tamaño nxn tales que A = QR.
- **Descomposición en valores singulares:** Siendo uno de los más grandes desarrollos aportados por el álgebra lineal moderna con importantes aplicaciones en diversos campos, la descomposición en valores singulares se encuentra definida de la siguiente manera:
  La SVD de una matriz A es una factorización del tipo $A = U\Sigma V^t$ con U $\in \mathbb{R}^{mxm}$, V $\in \mathbb{R}^{nxn}$ ortogonales y $\Sigma \in \mathbb{R}^{mxn}$ una matriz formada por los valores singulares de A en su diagonal principal ordenados de mayor a menor.

2. Desarrollo

Análisis previo: En primer lugar, decidimos procesar la base de datos MNIST en *Matlab* dado que el procesamiento de matrices en dicho programa resulta trivial y preferimos asegurarnos de que los datos con los que compararíamos nuestra imagen fueran correctos. Para ello, pasamos la matriz de MNIST de 3 dimensiones de 28x28x60000 a una de 60000x784 con el fin de obtener una matriz cuyas filas fueran las 60000 imágenes. Dado que la matriz resultante contenía demasiados

datos, provocando grandes demoras a la hora de realizar cálculos, decidimos utilizar únicamente las primeras 10000 de las 60000 imágenes, con lo cual la dimensión de la matriz terminó siendo 10000x784. La matriz en cuestión fue la utilizada como base de datos para realizar las comparaciones necesarias.
Por otra parte, de manera tal a poder definir la matriz V conformada por los autovectores de la matriz de covarianza X^tX puestos como columnas, decidimos utilizar el Método de Potencia + Deflación para hallar los mismos. Dicho método está basado en la siguiente propiedad:

Sea $A \in R^{nxn}$ *una matriz con autovalores distintos* $\lambda_1 > \lambda_2 > ... > \lambda_n$ *y una base ortonormal de autovectores* $\Rightarrow A - \lambda_1 v_1 v_1^t$ *tiene autovalores* $0, \lambda_2, ..., \lambda_n$*, con autovectores asociados* $v_1, ..., v_n$*.*

y sigue el siguiente algoritmo:

1: **for** $i = 1 \rightarrow k$ **do**
2: $v_i \leftarrow MetodoPotencia(A)$
3: $\lambda i \leftarrow (v_i^t A v_i)/(v_i^t v_i)$
4: $A \leftarrow A - \lambda_i v_i v_i^t$
5: **end for**

Decidimos utilizar dicho método pues su implementación resultó simple y corta (contra el Algoritmo QR) permitiéndonos enfocar nuestro trabajo en los resultados y experimentaciones.
El problema presentado por el Método de Potencia + Deflación fue que, para que éste funcionara, los autovalores en módulo debían ser todos distintos pues es la única manera de asegurar convergencia en el Método de Deflación. Si bien esta restricción nos podría traer problemas, supusimos que no nos toparíamos con éstos. Al intentar calcular V hallamos Nan's por lo que decidimos verificar el resultado de realizar SVD a la matriz de covarianza en $Matlab$. Luego de tal experimentación, nos encontramos con resultados muy distintos por lo que nos dispusimos a hallar los autovalores de la matriz en cuestión. Una vez éstos calculados, encontramos que varios de ellos eran iguales (con valor 0), por lo que las iteraciones del Método de Deflación ya no aseguraban convergencia a una combinación lineal de los correspondientes vectores propios. De este modo, el Método de Potencia+Deflación quedó inutilizado.

A partir de esto, decidimos utilizar el algoritmo QR para encontrar los autovectores necesarios para el cálculo de TC. Dicho algoritmo sigue el siguiente pseudocódigo:

Algorithm 2.1 function [V] = RQ(Ak)

1: **repeat**
2: $V = identidad(784, 784)$
3: $sumaArriba = 1005$
4: **while** $sumaArriba >= 1000$ **do**
5: $sumaArriba = 0$
6: $[Q_k, R_k] = qr(A_k)$
7: $A_k = R_k * Q_k$
8: $V = Q_k * V$
9: **for** $i = 2 \rightarrow 784$ **do**
10: **for** $j = 1 \rightarrow i - 1$ **do**
11: $sumaArriba = sumaArriba + abs(A_k(i, j))$
12: **end for**
13: **end for**
14: **end while**
15: **until** A_{k+1} sea triangular inferior

Para que nuestros resultados fueran lo más precisos posibles, decidimos evitar todo tipo de error posible dado a outliers. Para lograr esto, preferimos no comparar cada transformada característica de la base de datos con la del dígito a determinar sino que elegimos calcular un promedio del conjunto de transformadas de cada dígito (del 0 al 9) y, éste sí, compararlo con el dígito a definir. La comparación mencionada anteriormente hace, en realidad, referencia al cálculo de la norma2. A partir de dicha comparación, el menor valor de las normas2 calculadas resultó ser el determinante a la hora de concluir a qué dígito se refería el parámetro ingresado.

Para realizar nuestras experimentaciones, alteramos k; siendo éste la cantidad de elementos del vector de la transformada característica a comparar. De esta manera, los valores 'extraordinarios' se vieron excluidos de la experimentación, dejando únicamente los valores principales y relevantes para la comparación.

Implementación: El paso siguiente consistió en programar en C++ la clase matriz conformada por las funciones requeridas para realizar los cálculos debidos. Estas funciones consistieron en las básicas necesarias para manipular matrices, como por ejemplo, suma, multiplicación, norma y producto interno, entre otras. Para facilitar la ejecución del programa, optamos por agregar a nuestra clase la función encargada de comparar la transformada característica del dígito a determinar con las correspondientes a los dígitos de la base de datos. Dicha clase se encuentra definida en $Matriz.cpp$

El *main.cpp* consistió en un menú necesario para procesar las matrices de entrada que serían, más tarde, utilizadas por las funciones encargadas de hallar la transformada más semejante a la recibida por parámetro. Para realizarlo, nos limitamos a utilizar herramientas conocidas (if, while, etc.) y $printf$'s. Los archivos requeridos para correr el programa son, entonces, la matriz de la base de datos junto con la matriz de etiquetas de las imágenes. Dichas matrices deben encontrarse en archivos separados. El programa genera automáticamente los archivos conteniendo la matriz TC y la matriz V^t indispensables para la ejecución del programa. De manera general, los pasos seguidos por nuestro programa son los siguientes: primero, se verifica si se tiene V^t. Si ésta se encuentra, se busca TC. Caso contrario, la matriz de la base de datos es buscada. Una vez encontrada dicha matriz, las matrices V^t y TC son generadas. A partir de aquí, el programa busca el dígito a analizar y las etiquetas de la base de datos para continuar llamándose a sí mismo de manera tal a realizar las ejecuciones necesarias con el material almacenado.

Modo de uso del programa: Nuestro programa corre en Windows 7 y Ubuntu. Para compilarlo, recomendamos usar g++. Para ejecutarlo es necesario compilar $Matriz.cpp$ y luego, *main.cpp*. Luego, se debe abrir el ejecutable del main. Si los archivos de texto anunciados anteriormente se encuentran dentro de la carpeta del ejecutable, el resultado obtenido aparecerá en pantalla en cuestión de minutos.

Recuperación de resultados: Para verificar la correctitud de nuestro programa comparamos los resultados obtenidos con la solución del mismo proceso realizado en Matlab. De esta manera, corroboramos los algoritmos realizados y descartamos experimentos innecesarios. Al realizar varias iteraciones llegamos a la conclusión de que los resultados de nuestro programa resultaban satisfactorios para la mayoría de las pruebas realizadas. Dicha verificación fue exclusivamente necesaria para el cálculo de autovalores dado que el Método de Potencia puede otorgar resultados incorrectos en el caso en el que se repitan autovalores y éstos pueden ser no evidentes.

3. Resultados

A partir de una continuidad de experimentaciones, logramos hallar distintos resultados que nos permitieron sacar diversas conclusiones:
Tal como puede observarse en el gráfico a continuación, pudimos constatar que, cuanto mayor es k, más alta resulta la cantidad de aciertos (7371 aciertos en 10000 imágenes con k = 784). Al ver detalladamente los dígitos que más problemas causaban, notamos que algunos de ellos son altamente reconocidos contra otros no tan evidentes. Por ejemplo, la cantidad de aciertos para el dígito 2 fue de 893 (contra los 991 presentes en la prueba) siendo ésta una muy buena aproximación. Lo mismo ocurrió con el número 6, dando 1010 aciertos contra los 1014 presentes en la base testeada. Sin embargo, algunos dígitos como por ejemplo el 4, difirieron mucho en cuanto a lo obtenido ya que fueron 1386 los reconocidos contra los 980 contenidos en la base de datos. Hallamos un sentido a esto al notar que el dígito 9 obtuvo muy pocos aciertos dándonos lugar a concluir que la mayor parte de los 9's fueron reconocidos como 4's. Esto nos permitió confirmar que realizar el promedio de las TC's de la base de datos puede generar problemas en muchos casos de reconocimiento de dígitos debido a algún posible outlier.

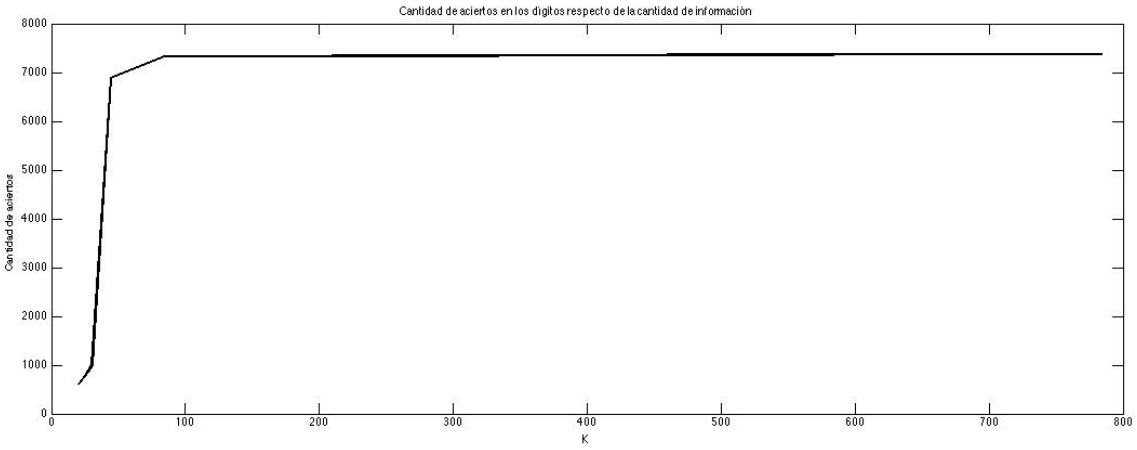

Figura 1: Cantidad de Aciertos respecto de K a partir de 10000 imágenes.

En el gráfico anterior puede observarse que, si bien los resultados son más precisos, alcanza con guardar el 6 % de los datos de cada imagen para alcanzar un 70 % de correctitud en la deducción de caracteres. Esto nos da lugar a pensar que, a pesar de ser mejor lograr una precisión del 73 %, no es realmente necesario almacenar toda la transformada característica de cada dígito para deducirlo. De esta manera, se evita ocupar memoria de manera excesiva y se logra mejorar el tiempo de ejecución de los programas encargados de realizar dichas comparaciones facilitando el reconocimiento de dígitos.

4. Discusión

Una vez obtenidos los resultados presentados previamente, nos realizamos cuestionamientos que nos parecieron interesantes para analizar. En primer lugar, pensamos qué podría haber pasado si en vez de realizar el promedio de cada dígito dentro de la base de datos hubiéramos comparado uno a uno con el dígito a determinar. Creemos que los resultados hubieran dependido básicamente de la cantidad de datos mal taggeados. Esto se debe a que la comparación entre la transformada de la imagen a determinar y la transformada de una imagen perteneciente a la base de datos puede resultar muy similar pero el resultante será siempre el de la etiqueta de dicha imagen. El

inconveniente que pudo haberse presentado por dígitos mal taggeados fue la probabilidad de que el promedio de éstos fuera desplazado de su centro. Sin embargo, al saber que la mayoría se encuentra correctamente etiquetado, supimos que no afectaría a gran escala los resultados. Por otra parte, si existiese la posibilidad de obtener una base de datos sin problemas de etiquetamiento, pensamos que la comparación con cada elemento de la base de datos daría mejores resultados en la mayoría de los casos (siempre que la cantidad de outliers no fuera demasiado grande). Esto se debe a que la comparación resultaría más exacta y la probabilidad de equivocarse se reduciría enormemente. Del mismo modo, se evitaría correr el riesgo de que dos promedios resulten altamente semejantes por tener representaciones parecidas dando lugar a errores en muchos de sus dígitos (como por ejemplo nuestro caso con los dígitos 4 y 9).

Por otra parte, pensamos que si no hubiese habido autovalores repetidos, probablemente el Método de Potencia hubiese sido más eficiente que el Método QR pues la cantidad de operaciones y de ciclos a realizar son mucho menores, disminuyendo enormemente el tiempo de ejecución del programa.

5. Conclusiones

Luego de las experimentaciones mencionadas anteriormente, pudimos extraer ciertas conclusiones. En primer lugar, podemos afirmar que cuanto mayor es la cantidad de datos en la transformada característica $(X_i * V^t)$ de cada imagen utilizada para compararlas entre sí con el fin de reconocer el dígito de una de ellas, más preciso es el resultado obtenido pues más límpida resulta la transformada característica de cada dígito. Esto se debe a que cuanto mayor es la cantidad de información, mayor precisión se obtiene.
Por otro lado, pudimos concluir que el Método de Potencia es más simple y claro que el Método QR pero la falta de certeza sobre su convergencia puede aportar grandes problemas a tal punto de dejarlo inutilizable en muchos casos.
Por otra parte, pudimos observar que no es esencial utilizar 60000 datos para realizar reconocimiento de dígitos manuscritos, sino que 10000 imágenes nos alcanzaron perfectamente para obtener muy buenos resultados evitando demorar demasiado el tiempo de ejecución de los mismos.
Por último, podemos afirmar que la utilización de SVD es muy efectiva para el reconocimiento de dígitos, superando ampliamente nuestras espectativas sobre los resultados que esperábamos obtener.

6. Apéndices

6.1. A

Laboratorio de Métodos Numéricos - Primer Cuatrimestre 2013
Trabajo Práctico Número 3: OCR+SVD

Introducción

El reconocimiento óptico de caracteres (OCR, por sus siglas en inglés) es el proceso por el cual se traducen o convierten imágenes de dígitos o caracteres (sean éstos manuscritos o de alguna tipografía especial) a un formato representable en nuestra computadora (por ejemplo, ASCII). Esta tarea puede ser más sencilla (por ejemplo, cuando tratamos de determinar el texto escrito en una versión escaneada a buena resolución de un libro) o tornarse casi imposible (recetas indescifrables de médicos, algunos parciales manuscritos de alumnos de métodos numéricos, etc).

El objetivo del trabajo práctico es implementar un método de reconocimiento de dígitos manuscritos basado en la descomposición en valores singulares, y analizar empíricamente los parámetros principales del método.

Como instancias de entrenamiento, se tiene un conjunto de n imágenes de dígitos manuscritos en escala de grises del mismo tamaño y resolución (varias imágenes de cada dígito). Cada una de estas imágenes sabemos a qué dígito se corresponde. En este trabajo consideraremos la popular base de datos MNIST, utilizada como referencia en esta área de investigación[1].

Para $i = 1, \ldots, n$, sea $x_i \in \mathbb{R}^m$ la i-ésima imagen de nuestra base de datos almacenada por filas en un vector, y sea $\mu = (x_1 + \ldots + x_n)/n$ el promedio de las imágenes. Definimos $X \in \mathbb{R}^{n \times m}$ como la matriz que contiene en la i-ésima fila al vector $(x_i - \mu)^t/\sqrt{n-1}$, y

$$X = U \Sigma V^t$$

a su descomposición en valores singulares, con $U \in \mathbb{R}^{n \times n}$ y $V \in \mathbb{R}^{m \times m}$ matrices ortogonales, y $\Sigma \in \mathbb{R}^{n \times m}$ la matriz diagonal conteniendo en la posición (i, i) al i-ésimo valor singular σ_i. Siendo v_i la columna i de V, definimos para $i = 1, \ldots, n$ la *transformación característica* del dígito x_i como el vector $\mathbf{tc}(x_i) = (v_1^t\, x_i, v_2^t\, x_i, \ldots, v_k^t\, x_i) \in \mathbb{R}^k$, donde $k \in \{1, \ldots, m\}$ es un parámetro de la implementación. Este proceso corresponde a extraer las k primeras *componentes principales* de cada imagen. La intención es que $\mathbf{tc}(x_i)$ resuma la información más relevante de la imagen, descartando los detalles o las zonas que no aportan rasgos distintivos.

Dada una nueva imagen x de un dígito manuscrito, que no se encuentra en el conjunto inicial de imágenes de entrenamiento, el problema de reconocimiento consiste en determinar a qué dígito corresponde. Para esto, se calcula $\mathbf{tc}(x)$ y se compara con $\mathbf{tc}(x_i)$, para $i = 1, \ldots, n$.

Enunciado

Se pide implementar un programa que lea desde archivos las imágenes de entrenamiento de distintos dígitos manuscritos y que, utilizando la descomposición en valores singulares, se calcule la transformación característica de acuerdo con la descripción anterior. Para ello se deberá implementar algún método de estimación de autovalores/autovectores. Dada una nueva imagen de un dígito manuscrito, el programa deberá determinar a qué dígito corresponde. El formato de los archivos de entrada y salida queda a elección del grupo. Si no usan un entorno de desarrollo que incluya bibliotecas para la lectura de archivos de imágenes, sugerimos que utilicen imágenes en formato RAW.

Se deberán realizar experimentos para medir la efectividad del reconocimiento, analizando tanto la influencia de la cantidad k de componentes principales seleccionadas como la influencia de la precisión en el cálculo de los autovalores.

Fecha de entrega

- *Formato electrónico:* viernes 21 de junio de 2013, hasta las 23:59 hs., enviando el trabajo (informe+código) a `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP3]` seguido de la lista de apellidos de los integrantes del grupo.
- *Formato físico:* lunes 24 de junio de 2013, de 18 a 20hs (en la clase de la práctica).

6.2. B

Implementación de nuestro programa:

Matriz.cpp

```
/*#include <iostream>

#include <cmath>

#include <limits>
```

[1] http://yann.lecun.com/exdb/mnist/

```
#include "Matriz.h"


#define N 60000
#define M 784
#define K 5


using namespace std;



Matriz::Matriz()
{
    p = NULL;
    rows = 0;
    cols = 0;
}


Matriz::Matriz(const int row_count, const int column_count)
{
    try
    {
        p = NULL;


        if (row_count > 0 && column_count > 0)
        {
            rows = row_count;
            cols = column_count;


            p = new double*[rows];
            for (int r = 0; r < rows; r++)
```

```
            {
                p[r] = new double[cols];

                for (int c = 0; c < cols; c++)
                {
                    p[r][c] = 0;
                }
            }
        }
    }
    catch(std::bad_alloc&)
    {
        cout << "Error al crear una Matriz. No hay memoria suficiente" << endl;
        p = NULL;
        rows = 0;
        cols = 0;
    }
}

Matriz::Matriz(const Matriz& a)
{
    rows = a.rows;
    cols = a.cols;
    p = new double*[a.rows];
    for (int r = 0; r < a.rows; r++)
    {
        p[r] = new double[a.cols];

        for (int c = 0; c < a.cols; c++)
        {
```

```
            p[r][c] = a.p[r][c];
        }
    }
}

Matriz::~Matriz()
{
    for (int r = 0; r < rows; r++)
    {
        delete[] p[r];
    }
    delete[] p;
    p = NULL;
}

void Matriz::Cero()
{
    for(int i = 0; i < this->rows; i++)
    {
        for(int j = 0; j < this->cols; j++)
        {
            this->p[i][j] = 0;
        }
    }
}

void Matriz::identidad()
{
    for(int i = 0; i < this->rows; i++)
```

```
    {
        for(int j = 0; j< this->cols; j++)
        {
            if (i == j)
                this->p[i][j] = 1;
            else
                this->p[i][j] = 0;
        }
    }
}

Matriz& Matriz::operator= (const Matriz& a)
{
    rows = a.rows;
    cols = a.cols;
    p = new double*[a.rows];
    for (int r = 0; r < a.rows; r++)
    {
        p[r] = new double[a.cols];

        // copia los valores de la matriz a
        for (int c = 0; c < a.cols; c++)
        {
            p[r][c] = a.p[r][c];
        }
    }
    return *this;
}

// le suma un valor double a la matriz a
```

```
Matriz& Matriz::sumar(const double v)
{
    for (int r = 0; r < rows; r++)
    {
        for (int c = 0; c < cols; c++)
        {
            p[r][c] += v;
        }
    }
    return *this;
}

// resta un double a la matriz a
Matriz& Matriz::restar(const double v)
{
    return sumar(-v);
}

// multiplica un double a la matriz a(elements wise)
Matriz& Matriz::multiplicar(const double v)
{
    for (int r = 0; r < rows; r++)
    {
        for (int c = 0; c < cols; c++)
        {
            p[r][c] *= v;
        }
    }
    return *this;
```

```
}

Matriz& Matriz::dividir(const double v)
{
    for (int r = 0; r < rows; r++)
    {
        for (int c = 0; c < cols; c++)
        {
            p[r][c] /= v;
        }
    }
    return *this;
}

Matriz& Matriz:: operator+(const Matriz& b)
{
    // me fijo si las dimensiones son las mismas
    if (this->rows == b.rows && this->cols == b.cols)
    {
        Matriz* res = new Matriz(this->rows, this->cols);

        for (int r = 0; r < this->rows; r++)
        {
            for (int c = 0; c < this->cols; c++)
            {
                res->p[r][c] = this->p[r][c] + b.p[r][c];
            }
        }
        return *res;
    }
```

```
    else
    {
        cout << "Error de dimensiones (+)";
    }
    Matriz* res = new Matriz();
    return *res;
}

void Matriz::operator+=(double b)
{
    this->sumar(b);
}

void Matriz::operator/=(double b)
{
    this->dividir(b);
}

Matriz& Matriz::multiplicar(const Matriz& b)
{
    if (this->cols == b.rows)
    {
      Matriz* res = new Matriz(this->rows, b.cols);

      for (int r = 0; r < this->rows; r++)
      {
        for (int c_res = 0; c_res < b.cols; c_res++)
        {
          for (int c = 0; c < this->cols; c++)
```

```
            {
              res->p[r][c_res] += this->p[r][c] * b.p[c][c_res];
            }
          }
        }
        return *res;
    }
    else
    {
        cout << "Error de dimensiones (multiplicar)" << endl << "n = "
        << this->rows << " m1 = "
        << this->cols << " m2 = " << b.rows << " k = " << b.cols;
        cout << "Error de dimensiones";
        Matriz* res = new Matriz(this->rows, this->cols);
        return *res;
    }
}

Matriz& Matriz::operator* (const Matriz& b)
{
    if (this->cols == b.rows)
    {
        Matriz* res = new Matriz(this->rows, b.cols);

        for (int r = 0; r < this->rows; r++)
        {
          for (int c_res = 0; c_res < b.cols; c_res++)
          {
            for (int c = 0; c < this->cols; c++)
            {
```

```
            res->p[r][c_res] += this->p[r][c] * b.p[c][c_res];
          }
        }
        //cout << "r = " << r << " de " << this->rows << endl;
      }
      return *res;
    }
    else
    {
      cout << "Error de dimensiones (*)" << endl << "n = " << this->rows
      << " m1 = " << this->cols
      << " m2 = " << b.rows << " k = " << b.cols;
      cout << "Error de dimensiones";
      Matriz* res = new Matriz(this->rows, this->cols);
      return *res;
    }
}

void Matriz::operator-=(const double b)
{
    for(int i = 0; i < this->rows; i++)
    {
        for(int j = 0; j < this->cols; j++)
        {
            this->p[i][j] = this->p[i][j] - b;
        }
    }
}
```

```
void Matriz::operator-=(const Matriz& b)
{
    for(int i = 0; i < this->rows; i++)
    {
        for(int j = 0; j < this->cols; j++)
        {
            this->p[i][j] = this->p[i][j] - b.p[i][j];
        }
    }
}

Matriz& Matriz::operator-(const Matriz& b)
{
        Matriz* res = new Matriz(this->rows, this->cols);
    for(int i = 0; i < this->rows; i++)
    {
        for(int j = 0; j < this->cols; j++)
        {
            res->p[i][j] = this->p[i][j] - b.p[i][j];
        }
    }
    return *res;
}

void Matriz::imprimir()
{
    for(int i = 0; i < this->rows; i++)
    {
        cout << "[ ";
        for(int j = 0; j < this->cols; j++)
```

```
        {
            cout << this->p[i][j];
            if(j != this->cols-1)
                cout << " ";
        }
        cout << "]" <<  endl;
    }
    cout << ";"<< endl << endl;
}

Matriz Matriz::traspuesta()
{
    Matriz res(this->cols, this->rows);
    for(int i=0; i<this->rows; i++)
        for (int j = 0; j < this->cols; j++)
            res.p[j][i] = this->p[i][j];

    return res;
}

void Matriz::mediaCero()
{
    Matriz *media = new Matriz(1, this->cols);
    media->Cero();
    for(int i = 0; i < this->rows; i++)
    {
        for(int j = 0; j < this->cols; j++)
        {
            media->p[0][j] += this->p[i][j];
```

```
        }
    }
    //tengo en el vector media la sumatoria de los vectores, me
    falta dividir cada uno de los elementos por n y restarle eso a la
    matriz original
    for(int j = 0; j < this->cols; j++)
    {
        media->p[0][j]/=this->rows;
        for(int i = 0; i < this->rows; i++)
        {
            this->p[i][j] -= (media->p[0][j]);
        }
    }
    delete media;
}

void Matriz::set(const int row, const int col, double valor)
{
    this->p[row][col] = valor;
}

double Matriz::get(const int fila, const int col)
{
    return this->p[fila][col];
}

const int Matriz::filas()
{
    return this->rows;
}
```

```
const int Matriz::columnas()
{
    return this->cols;
}

double Matriz::norma2()
{
    double res = 0;
    for(int i = 0; i < this->rows; i++)
        for(int j = 0; j < this->cols; j++)
            res+= pow(this->p[i][j],2);
    res = sqrt(res);
    return res;
}

double Matriz::normaInf()
{
    double res;
    for(int i = 0; i < this->rows; i++)
        for(int j = 0; j < this->cols; j++)
        {
            if(res < abs(this->p[i][j]))
                res = abs(this->p[i][j]);
        }

    return res;
}
```

```
double Matriz::prodInterno() //para vectores
{
    double sum = 0;
    for(int i = 0; i < this->rows; i++)
    {
        sum += pow(this->p[i][0], 2);
    }
    return sum;
}

double Matriz::potencia(Matriz* E, int Max, float epsilon, Matriz* X) //En X devuelvo
el autovector, y la funcion devuelve el autovalor
{
    // X es m x 1
    Matriz res(E->rows, E->cols);
    *X/= X->norma2();
    double lambda = 100;
    double normaDeX;

    for(int i = 0; i < Max; i++)
    {
        normaDeX = X->normaInf();
        *X = *E * *X;
        lambda = X->normaInf() / normaDeX;
        *X/= X->norma2();

        if(abs(lambda) < epsilon)
            break;
    }
    return lambda;
```

```
}

Matriz* Matriz::potenciaYDeflacion(int cantAutovectores, double errorMax,
                                   int iterMax, Matriz* v)
{
    //v == Vi en el pseudocodigo
    // this = E
    /*
        N = 60000
        M = 784
        double Vt[K][M];
        double TC[K][M];
        double A[N][M];
    */
    Matriz *lambda = new Matriz();
    Matriz *Vt = new Matriz(cantAutovectores,M);
    Matriz *vt = new Matriz(); //esto es Vt[i], un vector
    for (int i = 0; i < cantAutovectores; i++)
    {
        potencia(this,iterMax,errorMax, v);
        for(int j = 0; j < M; j++)
        {
            Vt->p[i][j] = v->p[j][0]; //k x m, m x 1
        }
        cout << "Obtuve autovector Nro: " << i+1 << endl;
        *vt = v->traspuesta();
        *lambda = vt->multiplicar(*this);
        *lambda = *lambda * *v;
        *lambda/=v->prodInterno();
```

```
        double L = lambda->matrizAEscalar();
        *v = v->multiplicar(L);
        *v = *v * *vt;
        *this = *this - *v;
    }
    delete vt;
    return Vt;
}

double Matriz::matrizAEscalar()
{
    return this->p[0][0];
}

//esta funcion va a sumar en cada indice la transformada de cada matriz y
despues las divide por la cantidad para quedarnos con el promedio de cada una
Matriz& Matriz::juntarMatrices(Matriz& X, Matriz& label){
    Matriz *res = new Matriz(M, 10);
    res->Cero();
        for(int j=0; j < label.cols; ++j){
        if(label.p[0][j] == 0){
            for(int i = 0; i < res->rows; ++i){
                                res->p[i][0] =  res->p[i][0] + X.p[i][0];
            }
        }else if(label.p[0][j] == 1){
                for(int i = 0; i < res->rows; ++i){
                                res->p[i][1] =  res->p[i][1] + X.p[i][1];
                }
        }else if(label.p[0][j] == 2){
            for(int i = 0; i < res->rows; ++i){
```

```
                        res->p[i][2] =  res->p[i][2] + X.p[i][2];
    }
}else if(label.p[0][j] == 3){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][3] =  res->p[i][3] + X.p[i][3];
    }
}else if(label.p[0][j] == 4){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][4] =  res->p[i][4] + X.p[i][4];
    }
}else if(label.p[0][j] == 5){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][5] =  res->p[i][5] + X.p[i][5];
    }
}else if(label.p[0][j] == 6){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][6] =  res->p[i][6] + X.p[i][6];
    }
}else if(label.p[0][j] == 7){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][7] =  res->p[i][7] + X.p[i][7];
    }
}else if(label.p[0][j] == 8){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][8] =  res->p[i][8] + X.p[i][8];
    }
}else if(label.p[0][j] == 9){
    for(int i = 0; i < res->rows; ++i){
                        res->p[i][9] =  res->p[i][9] + X.p[i][9];
```

```
        }
    }


}
for(int j=0; j < res->cols; ++j){
    if(j == 0){
                    for(int i=0; i < res->rows; ++i){
           res->p[i][j] = res->p[i][j]/5923;
        }
    }
    if(j == 1){
                    for(int i=0; i < res->rows; ++i){
         res->p[i][j] = res->p[i][j]/6742;
        }
    }
    if(j == 2){
                    for(int i=0; i < res->rows; ++i){
         res->p[i][j] = res->p[i][j]/5958;
        }
    }
    if(j == 3){
                    for(int i=0; i < res->rows; ++i){
         res->p[i][j] = res->p[i][j]/6131;
        }
    }
    if(j == 4){
                    for(int i=0; i < res->rows; ++i){
         res->p[i][j] = res->p[i][j]/5842;
        }
    }
```

```
        if(j == 5){
                        for(int i=0; i < res->rows; ++i){
                res->p[i][j] = res->p[i][j]/5421;
            }
        }
        if(j == 6){
                        for(int i=0; i < res->rows; ++i){
                res->p[i][j] = res->p[i][j]/5918;
            }
        }
        if(j == 7){
                        for(int i=0; i < res->rows; ++i){
                res->p[i][j] = res->p[i][j]/6265;
            }
        }
        if(j == 8){
                        for(int i=0; i < res->rows; ++i){
                res->p[i][j] = res->p[i][j]/5851;
            }
        }
        if(j == 9){
                        for(int i=0; i < res->rows; ++i){
                res->p[i][j] = res->p[i][j]/5949;
            }
        }
    }
    return *res;
}
```

```
int Matriz::averiguarDigito(Matriz& Vt, Matriz& Xi, Matriz& lab, int opcion) //
this = TC

{

    int numero;

    int res;

        double menorDistancia = numeric_limits<double>::max( );

        Matriz *tcXi = new Matriz();

        Matriz *vectorTc = new Matriz(M,1);

        double distancia;

        // Xi es un vector columna. tc(Xi) = Vt * Xi

        *tcXi = Vt * Xi;


    if(opcion == 1)

    {

        for(int j=0; j < this->cols; ++j)

        {

            for(int i=0; i < this->rows; ++i)

            {

                //en la i-esima columna de TC estan los k valores que describen

                la i-esima imagen

                vectorTc->p[i][0] = this->p[i][j];

            }

            distancia = (*vectorTc-Xi).norma2();

            if(distancia < menorDistancia)

            {

                numero = j;

                menorDistancia = distancia;

            }

        }

        res = lab.p[numero][0];
```

```
    }
    else
    {
        //opcion 2
        Matriz *newTC = new Matriz();
        *newTC = this->juntarMatrices(*this, lab);

        for(int j=0; j < newTC->cols; ++j)
        {
            for(int i=0; i < newTC->rows; ++i)
            {
                //en la i-esima columna de TC estan los k valores que describen
                la i-esima imagen
                vectorTc->p[i][0] = newTC->p[i][j];
            }
            distancia = (*vectorTc-Xi).norma2();
            if(distancia < menorDistancia)
            {
                numero = j;
                menorDistancia = distancia;
            }
        }
        res = numero;
    }
    delete tcXi;
    delete vectorTc;
        return res;
}

void Matriz::compararTC(Matriz& tcMatlab)
```

```
{
    double mayorDirefencia = numeric_limits<double>::min( );
    int mDi, mDj;
    double normaVectorMasLejano = numeric_limits<double>::min( );
    double norma;
    int VML;
    Matriz *comparador1 = new Matriz(M, 1);
    Matriz *comparador2 = new Matriz(M, 1);

    for(int j = 0; j < this->cols; j++)
    {
        for(int i = 0; i < this->rows; i++)
        {
            if(abs(tcMatlab.p[i][j] - this->p[i][j]) > mayorDirefencia)
            {
                mayorDirefencia = tcMatlab.p[i][j];
                mDi = i;
                mDj = j;
            }

            comparador1->p[i][0] = tcMatlab.p[i][j];
            comparador2->p[i][0] = this->p[i][j];
        }
        norma = (*comparador1 - *comparador2).norma2();
        if(norma > normaVectorMasLejano)
        {
            normaVectorMasLejano = norma;
            VML = j;
        }
```

```
    }

    cout << endl << "Resultados del analisis:" << endl << endl;
    cout << "El elemento mas alejado del aportado es el TC[" << mDi << "]["
    << mDj <<"], por " <<
        mayorDirefencia << " de diferencia" << endl;
    cout << "Por otro lado, el vector mas alejado del aportado es el " << VML <<
     "-esimo vector, por una diferencia de " << normaVectorMasLejano <<
     " en norma 2" << endl;
}

void  Matriz::rotacion(Matriz& Q, int i, int j)    //R = this
{
        double coseno = this->p[j][j]/sqrt(this->p[j][j]*this->p[j][j]+
        this->p[i][j]*this->p[i][j]);
        double seno = this->p[i][j]/sqrt(this->p[j][j]*this->p[j][j]+
        this->p[i][j]*this->p[i][j]);

        double aux;
        for(int w=j; w < this->rows; w ++){
                aux = this->p[j][w];
                this->p[j][w] = aux*coseno + seno * this->p[i][w];
                this->p[i][w] = aux*(-seno) +   coseno * this->p[i][w];
        }
        this->p[i][j]=0;

        for(int w=0; w < this->rows; w ++){
                aux = Q.p[j][w];
                Q.p[j][w] = aux*coseno + seno * Q.p[i][w];
                Q.p[i][w] = aux*(-seno) +       coseno * Q.p[i][w];
```

```
        }
}


void Matriz::factorizacionQR (Matriz Q, double toleranciaPorElemento) //return Q,
R = this
{
    if(this->rows != this->cols)
    {
        cout << "Error de dimensiones. QR opera sobre matriz cuadrada." << endl;
    }
        Q.identidad();

        for (int j=0; j<this->rows -1;j++){
                for(int i=j+1;i<this->rows;i++){
                        if(fabs(this->p[i][j])>=toleranciaPorElemento){
                                this->rotacion(Q,i,j);
                        }
                }
        }
}


void Matriz::algoritmoQR(Matriz Q, double tolMatrizA, double tolPorElemento)
{
    if(this->rows != this->cols)
    {
        cout << "Error de dimensiones. QR opera sobre matriz cuadrada." << endl;
    }
    while(! (this->paradaQR(tolMatrizA)))
    {
        this->factorizacionQR(Q, tolPorElemento);
```

```
        *this = *this * Q;
    }
}

bool Matriz::paradaQR(double tol)
{
    double aux = 0;
    for(int i = 1; i < this->rows; i++)
    {
        for(int j = 0; j < i-1; j++)
        {
            aux += abs(this->p[i][j]);
        }
    }
    return (tol > aux);
}

void Matriz::cortar(int nuevaCantFilas)
{
    for(int a = nuevaCantFilas; a < this->rows; a++)
    {
        delete[] p[a];
    }
}
```

Matriz.h

```
#include <iostream>
#include <cmath>

using namespace std;
```

```
class Matriz
{
public:
    Matriz();
    Matriz(const int row_count, const int column_count);
    Matriz(const Matriz& a);
    ~Matriz();

    void Cero();
    void identidad();
    Matriz& operator= (const Matriz& a);
    Matriz& sumar(const double v);
    Matriz& restar(const double v);
    Matriz& multiplicar(const double v);
    Matriz& multiplicar(const Matriz& b);
    Matriz& dividir(const double v);
    Matriz& operator+(const Matriz& b);
    void operator+= (const double b);
    void operator/= (const double b);
    void operator-=(const double b);
    void operator-=(const Matriz& b);
    Matriz& operator-(const Matriz& b);
    Matriz& operator*(const Matriz& b);
    void imprimir();

    Matriz traspuesta();
    void mediaCero();
    Matriz* potenciaYDeflacion(int cantAutovectores, double errorMax,
```

```
    int iterMax, Matriz* X0);
    void set(const int fila, const int col, double valor);
    double get(const int fila, const int col);
    const int columnas();
    const int filas();
    double norma2();
    double normaInf();
    double prodInterno();
        int averiguarDigito(Matriz& Vt, Matriz& Xi, Matriz& lab, int opcion);
    void compararTC(Matriz& tcMatlab);
    void algoritmoQR(Matriz Q, double tolMatrizA, double tolPorElemento);
    void cortar(int nuevaCantFilas);



private:
    double matrizAEscalar();
    double potencia(Matriz* E, int M, float epsilon, Matriz* X);
    Matriz& juntarMatrices(Matriz& TC, Matriz& label);
    void rotacion(Matriz& Q, int i, int j);
    void factorizacionQR (Matriz Q, double toleranciaPorElemento);
    bool paradaQR(double tol);


    int rows;
    int cols;
    double** p; //el puntero a la matriz
};
```

main.cpp

```
#include <iostream>
#include <sstream>
#include <cstdio>
#include <cmath>
```

```
#include <fstream>
#include <vector>
#include "Matriz.h"

#define N 60000
#define M 784
#define K 50

using namespace std;

int main(int argc, char **argv)
{
    printf("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
            -*-*-*-*-");
    printf("\nBienvenido al programa del TP3 de MetNum, primer cuatrimestre
            2013.\n");
    cout<<"Alumnos"<<endl;
    cout<<"             Izcovich, Sabrina. LU: 550/11"<<endl;
    cout<<"             Otero, Fernando. LU: 424/11"<<endl;
    cout<<"             Vita, Sebastian. LU: 149/11"<<endl;

    Matriz *Vt = new Matriz(K,M);
    Matriz *TC = new Matriz(K,N);

    /* 0) */
    ifstream archivoVt;
    archivoVt.open("Vt.txt");
    if(archivoVt.good() && archivoVt.is_open())
    {
        cout << "Existe la matriz Vt" << endl;
        cout << "Cargando..." << endl;
        int aux;
        for (int i = 0; i < K; i++)
        {
            for(int j = 0; j < M; j++)
            {
                if((i+1)%6001 == 0)
                    cout << "|"; // 10 barritas = fila cargada

                archivoVt >> aux;
                Vt->set(i,j,aux);
            }
            cout << endl;
        }
        cout << "Matriz Vt cargada" << endl;

        ifstream archivoTC;
        archivoTC.open("TC.txt");
        if(archivoTC.good() && archivoTC.is_open())
        {
            cout << "Existe la matriz TC" << endl;
            cout << "Cargando..." << endl;
            int aux;
            for (int i = 0; i < K; i++)
            {
                for(int j = 0; j < N; j++)
                {
                    archivoTC >> aux;
                    TC->set(i,j,aux);
```

```
        }
        if((i+1)%6001 == 0)
            cout << "|"; // 10 barritas = matriz cargada
    }
    cout << "Matriz TC cargada" << endl;
}
else
{
    cout << "Se produjo algun error. Existe Vt pero no existe TC" << endl;
    ifstream entradaDB;
    entradaDB.open("matriz.txt");
    if(entradaDB.good() && entradaDB.is_open())
    {
        Matriz *A = new Matriz(M,N);
        cout << "Existe la matriz de la base de datos" << endl;
        cout << "Cargando..." << endl;
        int aux;
        for (int i = 0; i < M; i++)
        {
            for(int j = 0; j < N; j++)
            {
                entradaDB >> aux;
                A->set(i,j,aux);
            }
            if((i+1)%6000 == 0)
            cout << "| "; // 10 barritas = matriz cargada
        }
        cout << " Matriz A cargada" << endl;

        //A->imprimir();
        /** 5) */
        *TC = *Vt * *A;
        cout << "Creada TC" << endl;
        delete A;

        ifstream matrizMatlab;
        matrizMatlab.open("matlab.txt");
        Matriz *matlab = new Matriz(N, 1);
        if(matrizMatlab.good() && matrizMatlab.is_open())
        {
            cout << "Existe matriz externa TC para comparar." << endl
            << "Cargando...";

            double elem;
            for(int i = 0; i < K; i++)
            {
                for(int j = 0; j < N; j++)
                matrizMatlab >> elem;
                matlab->set(i, 0, elem);
                if(i+1 % 6000 == 0)
                    cout << "|";
            }
            cout << "Matriz TC externa cargada" << endl;
            TC->compararTC(*matlab);
        }
        else
        {
            cout << "No hay matriz TC externa para comparar.
                        El programa sigue su curso."
```

```
                <<  endl;
            }
            matrizMatlab.close();

            /* ME QUEDO CON TC Y CON Vt PARA TRABAJAR CON EL VECTOR
                    QUE ME PASEN*/
            /** Y ACA ESCRIBO TC EN TC.txt*/

            ofstream escribirTC;
            escribirTC.open ("TC.txt", ofstream::trunc);
            if(escribirTC.good() && escribirTC.is_open())
            {
                for(int i = 0; i < K; i++)
                {
                    for(int j = 0; j < N; j++)
                    {
                        escribirTC << TC->get(i,j);
                    }
                }
            }
            else
            {
                cout << "Error al crear el archivo TC.txt";
            }
            escribirTC.close();
        }
        else
        {
            cout << "No existe el archivo de la base de datos." << endl <<
            "El programa no puede continuar" << endl;
            entradaDB.close();
            archivoTC.close();
            int aux;
            cin >> aux;
            return 0;
        }
        entradaDB.close();
    }
    archivoTC.close();
}
else
{
    cout << "No existe la matriz V" <<  endl << endl;
    ifstream entradaDB;
    entradaDB.open("matriz.txt");
    if(entradaDB.good() && entradaDB.is_open())
    {
        Matriz *A = new Matriz(M,N);
        Matriz *X = new Matriz(N,M);
        cout << "Existe la matriz de la base de datos" << endl;
        cout << "Cargando..." << endl;
        int aux;
        for (int i = 0; i < N; i++)
        {
            for(int j = 0; j < M; j++)
            {
                entradaDB >> aux;
                A->set(i,j,aux);
            }
```

```
    if((i+1)%6000 == 0)
    cout << "| "; // 10 barritas = matriz cargada
}
cout << " Matriz A cargada" << endl;

/** 2) y 3) */
*X = A->traspuesta();
X->mediaCero();
Matriz *Xt = new Matriz();
    *Xt = X->traspuesta();
    cout << "Creada Xt" << endl;
    Matriz *E = new Matriz();
    *E = *Xt * *X;
    *E/=(N-1);
    delete X;
    delete Xt;
    cout << "Creada At * A" << endl;
    /** 4a) Potencia y deflacion */
    //double errorMax = 0.00001;
    //int iterMax = 1000;
    //Matriz *X0 = new Matriz(M, 1);
    //X0->Cero();
//X0->set(0,0, 1); //para q tenga norma 1, nada mas
    //Vt = E->potenciaYDeflacion(K, errorMax, iterMax, X0);
//delete X0;
/** 4b) Algoritmo QR*/
Matriz *Q = new Matriz(E->columnas(), E->filas());
E->algoritmoQR(*Q, 0.1, 0.00001);
delete E;
Vt->cortar(K); //deja la matriz de (K,M)
    cout << "Creada Vt" << endl;
    delete Q;
    /** 5) */
    *TC = *Vt * *A;
    cout << "Creada TC" << endl;
    delete A;

ifstream matrizMatlab;
matrizMatlab.open("matlab.txt");
Matriz *matlab = new Matriz(N, 1);
if(matrizMatlab.good() && matrizMatlab.is_open())
{
    cout << "Existe matriz externa TC para comparar." << endl
     << "Cargando...";

    double elem;
    for(int i = 0; i < K; i++)
    {
        for(int j = 0; j < N; j++)
        matrizMatlab >> elem;
        matlab->set(i, 0, elem);
        if(i+1 % 6000 == 0)
            cout << "|";
    }
    cout << "Matriz TC externa cargada" << endl;
    TC->compararTC(*matlab);
}
else
{
```

```
            cout << "No hay matriz TC externa para comparar. El programa
                            sigue su curso." <<  endl;
        }
        matrizMatlab.close();

            /* ME QUEDO CON TC Y CON Vt PARA TRABAJAR CON EL VECTOR
            QUE ME PASEN*/
        /** Y ACA ESCRIBO VT EN Vt.txt Y TC EN TC.txt*/

        ofstream escribirVt;
        escribirVt.open ("Vt.txt", ofstream::trunc);
        if(escribirVt.good() && escribirVt.is_open())
        {
            // voy a escribir los 2 archivos al mismo tiempo porque Vt
            y TC tienen las mismas dimensiones
            // Vt ES DE K X M Y TC ES DE K X N
            ofstream escribirTC;
            escribirTC.open ("TC.txt", ofstream::trunc);
            if(escribirTC.good() && escribirTC.is_open())
            {
                for(int i = 0; i < K; i++)
                {
                    for(int j = 0; j < M; j++)
                    {
                        escribirVt << Vt->get(i,j);
                        if(j<N)
                            escribirTC << TC->get(i,j);
                    }
                }
            }
            else
            {
                cout << "Error al crear el archivo TC.txt";
            }
            escribirTC.close();
        }
        else
        {
            cout << "Error al crear el archivo Vt.txt";
        }
        escribirVt.close();
    }
    else
    {
        cout << "No existe el archivo de la base de datos."
         << endl << "El programa no puede continuar" << endl;
        entradaDB.close();
        archivoVt.close();
        int aux;
        cin >> aux;
        return 0;
    }
    entradaDB.close();
}
archivoVt.close();



ifstream archivoLabel;
```

```
archivoLabel.open("label.txt");
Matriz *lab = new Matriz(N, 1);
if(archivoLabel.good() && archivoLabel.is_open())
{
    cout << "Existe el vector de etiquetas." << endl
            << "Cargando...";

    double elem;
    for(int i = 0; i < N; i++)
    {
        archivoLabel >> elem;
        lab->set(i, 0, elem);
        if(i+1 % 6000 == 0)
            cout << "|";
    }
    cout << "Matriz Label cargada" << endl;
}
else
{
    cout << "No existe la etiqueta. El programa no puede continuar."
    <<  endl;
    int aux;
    cin >> aux;
    return 0;
}
archivoLabel.close();


/** PARTE B
    1) Leo de un archivo el vector X ( Matriz *X = new Matriz(M,1); )
    2) tc(X) = *TC * *X
    3) Recorro lo que tenga que recorrer (sea TC, un TC alfa-podado, o
    un vector con los 10 promedios) viendo con quien se parece mas tc(X)
    Para ver con quien se parece mas miro (*X-algunVector).norma2()
*/

ifstream archivoVectorX;
archivoVectorX.open("digito.txt");
if(archivoVectorX.good() && archivoVectorX.is_open())
{
    cout << "Existe el vector a analizar." << endl << "Cargando...";
    Matriz *X = new Matriz(M, 1);
    double elem;
    for(int i = 0; i < M; i++)
    {
        archivoVectorX >> elem;
        X->set(i, 0, elem);
        if(i+1 % 78 == 0)
            cout << "|";
    }
    cout << "Matriz X cargada" << endl << endl;

    int opcion = -1;
    cout << "Listo! solo falta que elija la opcion con la que quiere que
    definamos que digito esta buscando" << endl;
    while (opcion < 1 || opcion > 2)
    {

        cout << "Opcion 1: Usar todas las imagenes de la base de datos" << endl <<
```

```
            "Opcion 2: Usar el promedio de
            cada digito" << endl << endl;
            cout << "Ingrese el numero de la opcion que desea:" << endl;
            cin >> opcion;
            if (opcion < 1 || opcion > 2)
                cout << "Error. Ingrese 1 o 2" << endl << endl;
        }
        cout << "El digito que se cargo es el: " << endl;
        int resultado = TC->averiguarDigito(*Vt, *X, *lab, opcion);

        cout << resultado;

        delete lab;
        delete Vt;
        delete TC;
        delete X;

    }
    else
    {
        cout << "Cree un archivo 'digito.txt' y vuelva a intentarlo." << endl;
    }
    archivoVectorX.close();
    return 0;
}
```

7. Referencias

- BURDEN, RICHARD L. ; Análisis numérico, 7ma ed. 2002. México, Thomson Learning.
- http://definicion.de/
- http://www.iti.es/media/about/docs/tic/13/articulo2.pdf
- http://mathworld.wolfram.com/Covariance.html
- http://yann.lecun.com/exdb/mnist/