Métodos Numéricos

Primer Cuatrimestre - 2015

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico II

Reconocimiento de Dígitos: Métodos Numéricos para la clasificación de datos

Integrante	LU	Correo electrónico
Cuneo, Christian	755/13	chriscuneo93@gmail.com

En este informe voy a desarrollar como solucione el problema planteado por la materia, en el cual se presentaba una base de datos, con diferentes muestras clasificadas, y al presentarse una nueva muestra, uno tenia que clasificara lo mas correctamente posible. En este caso vamos a tratar imágenes de dígitos manuscritos como datos, y el dígito que representan como la clasificación

k nearest neighbors **Análisis de componentes principales** **Método de la Potencia**

Índice

1. Introducción Teórica

La resolución de este trabajo practico tiene como cimientos dos métodos muy importantes del álgebra lineal utilizados para resolver sistemas de ecuaciones, el primero, y el mas conocido, es el método de eliminación gaussiana; el segundo es el de la Factorizacion LU

1.1. K nearest neighbors

El método kNN (k vecinos mas cercanos en ingles) es un un método de clasificación de datos, en este metodo ves a cada muestra, de n atributos, como un elemento en el espacio $\Re^n$. Se tienen un conjunto de muestras ya clasificadas, llamado grupo de entrenamiento, y, para clasificar una muestra nueva uno calcula la distancia euclidiana: $d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$ con cada muestra del grupo de entrenamiento. Luego uno almacena la clasificación de los k elementos mas cercanos y asigna a la muestra desconocida la clasificación predominante del grupo de los mas cercanos.

1.2. Análisis de Componentes Principales

El análisis de componentes principales (PCA en español) tiene este objetivo: dadas n observaciones de p variables, se analiza si es posible representar adecuadamente esta información con un número menor de variables construidas como combinaciones lineales de las originales.

El procedimiento utiliza una transformación ortogonal para convertir un conjunto de muestras con variables posiblemente correlacionadas, en un conjunto de muestras con variables no correlacionadas (no dependientes entre si), a estas variables las llamamos componentes principales (la cantidad de estos componentes siempre es menor o igual a la cantidad de variables originales). Esta tranformación esta definida de una forma que el primer componente principal sea el provoque mayor variabilidad en los datos, quiere decir el de mayor varianza, el segundo sea el siguiente mayor, y así sucesivamente.

2. Desarrollo

En el presente informe se presenta la implementación, desarrollo y análisis de resultados usados en la resolución de un problema práctico, con algunas herramientas estudiadas y provistas por la materia, algunos conceptos vistos en probabilidad y estadística, y por ultimo conceptos aprendidos a medida que se iba avanzando con el trabajo. El problema principal planteado consiste en lograr reconocer una imagen de un dígito manuscrito, quiere decir, poder predecir a que dígito corresponde una imagen dada.

Para resolver este problema se te presenta una base de datos imágenes de dígitos manuscritos que ya se saben a que dígito corresponde cada una (grupo de imágenes "train"), y por supuesto un grupo de imágenes a las que deberás asignar un dígito utilizando alguna forma de reconocimiento (grupo de imágenes "test"). Para hacer el proceso mas simple todas las imágenes son cuadradas de 28x28 pixeles y en escala de grises.

El problema se aborda desde dos puntos de vista, el probabilista y el del álgebra lineal. El álgebra lineal se utiliza para llegar a las soluciones, y la probabilidad se utiliza para hacer este proceso de una forma mas eficiente y rendidora.

A través de la generalización uno puede ver a una imagen como una coordenada en un espacio $\mathbf{R}^n$, siendo n la cantidad de pixeles de la imagen (en este caso 784), y como cada imagen de la base de datos de imágenes conocidas (train) es entonces una coordenada con un una clasificación conocida (su dígito); luego, una aproximación a la solución valida seria, dada una imagen desconocida, buscar que puntos mas cercanos tiene en el espacio $\mathbf{R}^{784}$, y asignarle un resultado acorde. Ya existe un método que realiza estos pasos, y se llama K-vecinos más cercanos (Knn por su ingles "K nearest neighbors") y consiste en buscar, utilizando la distancia euclidiana, los K puntos mas cercanos a cierto punto, en nuestro caso, la imagen desconocida.

Luego de tener estos K puntos cercanos, uno puede clasificar a que grupo pertenece la imagen desconocida de distintas formas, en nuestro caso se utiliza la votación, básicamente teniendo estos K vecinos, buscamos al grupo dominante (la clasificación que mas veces aparece) y asignamos esta clasificación a nuestra imagen desconocida.

Al trabajar en $\mathbf{R}^{784}$ este proceso se vuelve muy lento, ya que este método se ve afectado directamente por la dimensión de los puntos que procesa, específicamente al calcular la distancia euclidiana, por esto lo que se va a buscar es reducir la dimensión de los puntos, tratando de perder la menor cantidad de características posibles de las imágenes. Otro problema es que este método se ve tocado por la maldición de la dimensionalidad, que en este caso afecta al momento de decidir que puntos son los mas cercanos al punto a clasificar, ya que si la dimensión de las muestras es muy grande, la mayoría de los vectores van a ser equidistantes, por lo tanto no se puede confiar en los resultados de la 'votacion'.

Es por esto que vamos a buscar alguna forma de reducir la dimensión de los datos a procesar. Aquí es donde entra en juego la probabilidad y estadística.

Vamos a buscar que pixeles de las imágenes son mas importantes y descartar los menos importantes, haciendo un análisis de componentes principales reduciendo así la dimensión en la que trabajamos. Para esto vamos a acudir al termino de la covarianza, que, aplicada a dos variables, muestra la tendencia de estas dos variables a variar juntas. Esto, aplicado a estas imágenes, vamos a buscar que puntos de las imágenes son los que mas varían entre todas las imágenes de train, obteniendo así información para saber que características (pixeles) de las imágenes son las mas importantes a analizar, las que relacionan o distinguen a una imagen de otra. Para esto vamos a calcular una matriz de covarianzas de los datos de train, ahi veremos la covarianza que tiene cada pixel con los demás pixeles de las muestras (hay que tener en cuenta que esta matriz sera simetrica, ya que la covarianza entre las variables a y b es igual a la covarianza entre b y a, por definición). Lógicamente esta matriz va a pertenecer a $\mathbf{R}^{784x784}$. Para calcular esta matriz primero generaremos la matriz X con una muestra de train por fila y haciendo calculos para que la media de estas sea cero, de esta forma:

$$x^{(i)} = (x^{(i)} - \mu)/\sqrt{n-1}$$

Donde $x^{(i)}$ es la fila i de X y μ es un vector con la media de cada variable de las muestras y n es la cantidad de muestras. Luego la matriz de covarianzas M_X sera:

$$M_X = X^t X$$

Como resultado M_X sera:

$$M_{X(j,k)} = \frac{1}{n-1}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

Como se ve, esta es la covarianza entre la variable j y k, y, en el caso $j = k$ el producto sera entre dos multiplicandos iguales, que serán la desviación estándar de la variable, y la desviación estándar de una variable al cuadrado es la varianza de esta. El siguiente paso sera encontrar los autovalores mas grandes de esta matriz y sus autovectores asociados, para lograr esto se utilizara el Método de la Potencia tantas veces como autovalores se quieran encontrar. Ya teniendo estos autovectores (que tienen a los mas grandes autovalores asociados), uno puede formar una base de autovectores para poder realizar una transformación característica aplicando un cambio de base y logrando un cambio de coordenadas de las muestras, de esta forma lograremos expresar las muestras de una forma que los datos principales de la muestra original se encuentren en cada coordenada de la muestra transformada. La base de autovectores de M_x sera:

$$V = (v_1, v_2, \ldots, v_\alpha)$$

Siendo α la cantidad de autovalores y autovectores buscados (la cantidad de componentes principales). Y la transformación característica sobre una muestra x se aplicara de esta forma:

$$tc(x) = V^t x = (v_1^t x, v_2^t x, \ldots, v_\alpha^t x)$$

Este transformación característica se aplicará a cada muestra de la base 'train', logrando, como vimos antes, tener en cuenta solo la información mas importante y disminuyendo la dimensión de las muestras. Esta misma transformación se aplicara luego a la muestra a clasificar, y ahora, al buscar los k vecinos mas cercanos, se tardara menos en calcular las distancias euclidianas (por la reducción de dimensión) y ademas se sabrá que la clasificación sera mas acertada (por el descarte de información innecesaria y redundante).

2.1. Implementación

2.2. Experimentación

Los objetivos principales de la experimentación en este trabajo es averiguar cual es la mejor forma de usar las técnicas implementadas para tener las mayores posibilidades de aciertos a la hora de clasificar datos desconocidos.
Para poder medir la efectividad de cualquiera de las técnicas de implementación implementadas uno tiene que aplicar los métodos para tratar de clasificar datos ya conocidos, en este punto entra en juego el concepto de Cross-Validation. Este método se basa en particionar la base de training que usamos en dos partes, una para usar de training, y otra de la cual tomaremos cada muestra y usaremos la implementación para clasificarla, para luego comparar la clasificación dada por el programa con la clasificación original de la muestra.
Cross-Validation plantea realizar este procedimiento para diferentes particiones de la base de training original y analizar cual es el porcentaje de acierto de la técnica (con sus respectivos parámetros), y usando estos datos, elegir cual es la mejor técnica a utilizar, y con que parámetros, balanceando el tiempo de ejecución y con el porcentaje de aciertos.

Para mis experimentos use dos juegos de Cross-Validation, uno en el que particiono la base de training en un 10 % para usar de base de test y un 90 % para usar de base de train; y otro en el que particiono en un 20 % para la base de test y un 80 % para la base de train. Para el primer juego hice 10 particiones distintas y para el segundo 5, todas generadas aleatoriamente utilizando python, usando $int(random.uniform(1, 42000))$ para generar 4200 números en el primer juego, y 8400 en el segundo (diferentes) para seleccionar las muestras para la partición de test, y el resto quedaba para usar de train.

2.2.1. Experimentación 1: kNN

Aquí vamos a analizar cual es el rendimiento temporal y la eficiencia del modelo de clasificación kNN. Para esto realicé un análisis utilizando Cross-Validation y midiendo los resultados variando el único

parámetro de esta implementación: cuantos elementos cercanos elegir para luego hacer la votación, es decir, el k de kNN.

La hipótesis principal es que el tiempo no va a variar por la cantidad de elementos que tome para hacer la votación, ya que simplemente para ver que elementos de la base de training están mas cerca de la muestra a clasificar implica medir la distancia euclidiana de cada elemento con la muestra, y este proceso es el que lleva mas tiempo, ya que luego la distancia se guardara en una cola de prioridad, y luego, la cantidad k de elementos cercanos que vamos a utilizar se van a 'popear' de esta cola, que es un proceso que tarda un tiempo insignificante, ya que lo que mas tarda es 'pushearlos'. En resumen, la diferencia de tiempo entre elegir un k y otro se vera en la cantidad de 'pops' a hacer para la votación, el cual es insignificante comparado con el calculo de las distancias euclidianas, que es de orden $O(n^2)$.

Los k que tome de parámetro para pruebas los pensé luego de contar cuantas muestras de cada dígito hay en la base de training, de estas cuentas tome la mínima, y utilice la regla de tres simple para ver, si hay x dígitos iguales en la base de training original, cuantos hay en la partición de training del cross-validation. Teniendo este numero uno piensa que, si usa un k mayor a este numero, es seguro que se van a tener en cuenta elementos distintos al elemento a clasificar, sea cual sea. A continuación se encontraran los resultados en gráficos, de estos experimentos. Hay que tener en cuenta que la cantidad de aciertos se calcula promediando los aciertos de cada $folding$ de cross-validation:

2.2.2. Experimentación 2: kNN + PCA

Esta experimentación va a analizar el rendimiento temporal y la eficiencia del modelo PCA junto al posterior uso de kNN. Para este experimento utilice como previamente dicho la técnica de Cross-Validation para analizar datos confiables, disminuyendo la probabilidad de que la elección de muestras para la base de train y la base de test afecten el resultado.

En este caso lo que hice fue usar la implementación de PCA + kNN variando k de 0 a 1000 (ya que previamente vimos que no cuesta nada mas, temporalmente hablando) y α de 1 a 60 (30 a 60 para el segundo juego de Cross-Validation, por cuestiones de tiempo); y luego para cada instancia:

- Medir el tiempo de ejecución del cambio de base al grupo de muestras de train, utilizando la transformación ortogonal obtenida gracias al análisis de componentes principales.
- Medir el tiempo de ejecución del procedimiento de kNN con el nuevo set de train de menores dimensiones.
- Medir la eficiencia de la técnica y sus parámetros checkeando cuantas calificaciones correctas acertó.

Como uno esta haciendo un análisis de componentes principales, al variar α uno decide cuantos componentes principales va a tener en cuenta para hacer la reducción de dimensiones, siempre empezando por el componente principal que abarca la información mas importante de los datos, hacia la que abarca detalles que no van a servir al tratar de distinguir un dato de otro. Sabiendo esto, uno espera que, al ir aumentando el α, el hit rate del método vaya aumentando mientras se esté considerando nueva información relevante, y, una vez que se llegue a abarcar toda la información relevante, aumentar el α no hará mas que considerar datos irrelevantes o demasiado detallistas, entonces el hit rate se estabilizara; si uno sigue aumentando el α, uno supondría que, al estar tomando en cuenta mas información irrelevante que relevante, el hit rate se vera disminuido.

Teniendo en cuenta esto, me pareció lógico aplicar el método para distintos α hasta que el hit rate se estabilice, y ahí sabremos (no con 100 % certeza) que hemos llegado al punto en el que la eficiencia dejara de seguir aumentando positivamente.

Decidí no medir el tiempo de computo que requiere generar la matriz de covarianzas, ya que esta depende unicamente del tamaño de la base de train, que es fija para cada Cross-Validation distinto, por lo tanto no me parecía un dato interesante.

Veamos los resultados de esta experimentación:

Para estos gráficos promedie los hit-rates de todas las particiones de cada juego de Cross-Validation por separado Ahora veremos cual fue el promedio de todos los hits-rates para cada alpha:

Y por ultimo los tiempos de cambio de base de la base de train, y del que tarda kNN en decidir la clasificación:

De estos resultados se pueden sacar las siguientes conclusiones:

- El tiempo de computo del cambio de base aumenta linealmente a medida que aumenta el α, esto sucede porque va a haber que multiplicar cada muestra de la base de train por un autevector mas, para agregar una dimensión mas a la muestra a usar luego. El tiempo que lleva hacer el producto entre la muestra y el autovector va a ser siempre el mismo, ya que es el producto entre un vector de 784 con otro de 784, por lo tanto tomara tan solo 784*784 operaciones en todos los casos, si este tiempo también se viera afectado por el cambio de α, entonces estaríamos hablando de un orden de computo $O(n^2)$ pero como no es así, el orden de complejidad temporal es de $O(n)$ como muestran los resultados de tiempo. Ademas como se ve, el resultado es prácticamente igual en para ambos juegos de Cross-Validation.

- El tiempo de computo de kNN también aumenta linealmente a medida que aumenta el α, esto es porque el procedimiento de calculo de distancias se va a hacer entre la muestra y la misma cantidad de muestras de train anterior, lo único que cambia es que cada muestra de train, al igual que la muestra desconocida, al aplicarles el cambio de base tendrán una dimensión mas que en el caso interior, esto hará que calcular las distancias euclidianas cueste una iteración mas, al verse afectado solo ese tiempo de computo, el orden de complejidad temporal sera de $O(n)$. En este caso también los resultados fueron consistentes entre los dos juegos de Cross-Validation.

- La efectividad del método de clasificación al variar α se vio afectada de una forma mas fuerte desde el $\alpha = 1$, en el cual la efectividad era muy baja, al $\alpha = 10$ donde se empieza a estabilizar, luego sigue subiendo mas lentamente hasta llegar al $\alpha = 18$, donde se estabiliza casi pareciendo constante. Desde este α la tasa de aciertos va aumentando muy levemente hasta encontrar su pico máximo en $\alpha = 39$ para el primer juego de CV y en $\alpha = 36$ para el segundo, luego de estos valores, la tasa de efectividad baja muy levemente y se queda oscilando, pero no logra superar el pico anterior, al menos en este rango de α. Al llegar al pico, el hit-rate es del 97,54 % y del 97,40 % para cada CV respectivamente. Lo que hay que destacar es que desde el $\alpha = 20$ aproximadamente ya se esta dentro del 97 % de efectividad, entonces se ve que el método es bastante efectivo desde un α bastante mas chico, esto va a ser útil a la hora de balancear entre efectividad y tiempo de computo.

- La efectividad del método al variar k es notable, se ve por los resultados que el mejor k a utilizar es el $k = 3$. Es muy interesante ver que, con $k = 1$ el método ya es bastante efectivo, pero al elegir $k = 2$ la efectividad se ve reducida notoriamente, esto tiene que ser ya que, al hacer la votación uno mide el máximo en cantidades de 'votos', y al haber solo dos, va a, o haber dos votos del mismo, en ese caso se elige ese, o haber uno para una clasificación y otro para otra clasificación, y en este caso, al estar uno tomando un simple máximo, va a elegirse el primero que se llegue con la iteración del algoritmo de 'máximo', entonces va a haber una probabilidad del 50 % de elegir la clasificación correcta. Luego al elegir $k = 3$ siempre va a haber un desempate, al menos que sean tres votos diferentes, que seria de esperar que fuera un caso raro.

3. Conclusiones

4. Referencias

Los conceptos los tome de:

- Las clases teóricas y practicas de la cátedra.
- Matrix Computations - Gene H. Golub, Charles F. Van Loan - JHU Press, Oct 15, 1996
- Numerical analysis - Richard L. Burden J. Douglas Faires - 2011

5. Apéndice: A

Métodos Numéricos
Primer Cuatrimestre 2015
Trabajo Práctico 2

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Si nos organizamos aprobamos todos...

Introducción

Uno de los momentos de mayor tensión durante un cuatrimestre se genera en épocas de parciales. El día del examen, luego de entregar la resolución del mismo, suele aparecer esa inmanejable incertidumbre en los alumnos: *¿entregué todas las hojas?*, *¿consideré el caso particular de $\beta = 0$?*, *¿por qué mi compañero puso verdadero, si yo encontré un contraejemplo?* Desde el lado del docente, la dificultad radica en corregir un gran volumen de ejercicios técnicamente complejos en un tiempo acotado, buscando deducir en base a información limitada si el alumno comprendió o no el tema.

Estos dos factores llevaron a la conformación de una comisión interclaustro de docentes y alumnos, para quienes resguardamos su identidad, a proponer como solución el desarrollo de una herramienta automática de corrección de exámenes, con el fin de reducir los tiempos en la devolución de los mismos. Para evitar suspicacias respecto de las correcciones, se propuso dividir el problema en varias etapas y que el sistema sea desarrollado por alumnos, contando desde ya con la validación de los docentes. La primera etapa del proyecto consiste en desarrollar una herramienta que permita automáticamente identificar dígitos manuscritos, a ser aplicada a una digitalización de cada examen, y cuyo post-procesamiento involucrará darle semántica y reglas a la información obtenida.

Como es de esperar, el éxito de la herramienta depende inevitablemente de disponer de un buen reconocedor de dígitos manuscritos. Para semejante responsabilidad, dado el gran nivel mostrado durante el comienzo del cuatrimestre, la comisión asesora ha decidido elegir a los alumnos de Métodos Numéricos para llevar adelante el desarrollo.

Objetivos y Metodología

Como instancias de entrenamiento, se tiene una base de datos de n imágenes de $M \times M$ píxeles, las cuales se encuentran etiquetadas con el dígito, 0-9, al que corresponden. Definimos como $m = M \times M$ al número total de píxeles de una imagen. Asumiremos también que las imágenes se encuentran en escala de grises (cada pixel corresponde a un valor entre 0 y 255, indicando la intensidad del mismo) y que el etiquetado no contiene errores. El objetivo del trabajo consiste en utilizar la información de la base de datos para, dada una nueva imagen de un dígito sin etiquetar, determinar a cuál corresponde teniendo en cuenta factores de calidad y tiempo de ejecución requeridos.

Una primera aproximación es utilizar el conocido algoritmo de k *Vecinos más Cercanos* (kNN, por su nombre en inglés). En su versión más simple, este algoritmo considera a cada objeto de la base de entrenamiento como un punto en el espacio, para el cual se conoce a qué clase corresponde (en nuestro caso, qué dígito es). Luego, al obtener un nuevo objeto que se busca clasificar simplemente se buscan los k vecinos más cercanos y se le asigna la clase que posea el mayor número de repeticiones dentro de ese subconjunto, es decir, la moda. Con este objetivo, podemos representar a cada imagen de nuestra base de datos como un vector $x_i \in \mathbb{R}^m$, con $i = 1, \ldots, n$, y de forma análoga interpretar las imágenes a clasificar mediante el algoritmo

kNN.

Sin embargo, el algoritmo del vecino más cercano es sensible a la dimensión de los objetos a considerar y, aún aplicando técnicas de preprocesamiento, puede resultar muy costoso de computar. Teniendo en cuenta esto, una alternativa interesante de preprocesamiento que busca reducir el tamaño de los elementos a considerar es el método de *Análisis de Componentes Principales* (PCA, por su sigla en inglés), que consiste en lo siguiente. Para $i = 1, \ldots, n$, recordar que $x_i \in \mathbb{R}^m$ 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

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

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 $\mathbf{tc}(x_i) = (v_1^t\, x_i, v_2^t\, x_i, \ldots, v_\alpha^t\, x_i) \in \mathbb{R}^\alpha$, donde $\alpha \in \{1, \ldots, m\}$ es un parámetro de la implementación. Este proceso corresponde a extraer las α 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, se calcula $\mathbf{tc}(x)$ y se compara con $\mathbf{tc}(x_i)$, para $i = 1, \ldots, n$, utilizando algún criterio de clasificación adecuado, como por ejemplo kNN.

Finalmente, nos concentramos en la evaluación de los métodos. Dado que necesitamos conocer a qué dígito corresponde una imagen para poder verificar la correctitud de la clasificación, una alternativa es particionar la base de entrenamiento en dos, utilizando una parte de ella en forma completa para el entrenamiento y la restante como test, pudiendo así corroborar la predicción realizada. Sin embargo, realizar toda la experimentación sobre una única partición de la base podría resultar en una incorrecta estimación de parámetros, como por ejemplo el conocido *overfitting*. Luego, se considera la técnica de *cross validation*, en particular el *K-fold cross validation*, para realizar una estimación de los parámetros del modelo que resulte estadísticamente más robusta.

Enunciado

Se pide implementar un programa en `C` o `C++` que lea desde archivos las imágenes de entrenamiento correspondientes a distintos dígitos y que, utilizando los métodos descriptos en la sección anterior, dada una nueva imagen de un dígito determine a qué número pertenece. Para ello, el programa deberá implementar el algoritmo de kNN así como también la reducción de dimensión previa utilizando PCA.

Con el objetivo de obtener la descomposición en valores singulares, se deberá implementar el método de la potencia con deflación para la estimación de autovalores/autovectores de la matriz de covarianza. En este contexto, la factibilidad de aplicar este método es particularmente sensible al tamaño de las imágenes de la base de datos. Por ejemplo, considerar imágenes en escala de grises de 100×100 píxeles implicaría trabajar con matrices de tamaño 10000×10000. Se pide tener en cuenta este factor durante el desarrollo y analizar cómo afecta (si es que lo hace) en el desarrollo del trabajo.

Consideramos la base de datos MNIST, en la versión disponible en *Kaggle*. Esta base contiene un conjunto de datos de entrenamiento de 42.000 dígitos manuscritos en escala de grises y con $M = 28$. A su vez, provee un conjunto de datos de test de 28.000 dígitos, de similares

características, pero sin el correspondiente etiquetado. En base a estos datos, se pide separar el procedimiento en dos etapas.

La primera de ellas consiste en realizar un estudio experimental con los métodos propuestos sobre la base de entrenamiento y utilizando la técnica de *K-fold cross validation* mencionada anteriormente. Para esta última tarea en particular, se recomienda leer y utilizar la rutina `cvpartition` provista por MATLAB. Llamamos k a la cantidad de vecinos a considerar en el algoritmo kNN, α a la cantidad de componentes principales a tomar y K a la cantidad de particiones consideradas para el cross-validation. La calidad de los resultados obtenidos serán analizados considerando la tasa de efectivdad lograda, es decir, la cantidad dígitos correctamente clasificados respecto a la cantidad total de casos analizados. En función de la experimentación se piden como mínimo los siguientes experimentos:

- Analizar el comportamiento y la factibilidad de aplicar el algoritmo kNN para un rango razonable de valores distintos de k, analizando el compromiso entre el tiempo de ejecución y la calidad de los resultados obtenidos. Es posible, como experimento opcional, considerar alguna técnica alternativa de preprocesamiento de la información en caso de ser necesario (y, desde ya, que no sea PCA).
- Analizar la calidad de los resultados obtenidos al combinar PCA con kNN, para un rango amplio de combinaciones de valores de k y α. Considerar en el análisis también el tiempo de ejecución.
- Realizar los experimentos de los items anteriores para al menos dos valores distintos de K. Justificar el por qué de la elección de los mismos.
- En base a los resultados obtenidos para ambos métodos, seleccionar aquella combinación de parámetros que se considere la mejor alternativa, con su correspondiente justificación, y compararlas entre sí y sugerir un método para su utilización en la práctica.

Finalmente, y con los parámetros seleccionados en la fase experimental, ejecutar el método seleccionado sobre el conjunto de datos de test, utilizando como entrenamiento los 42.000 dígitos comprendidos en el conjunto de entrenamiento. Analizar el tiempo de cómputo requerido. Dado que la cátedra no posee la información sobre a qué dígito corresponde cada imagen (y la idea no es graficar uno por uno y obtenerlo a mano), se sugiere a cada grupo participar en la competencia *Digit Recognizer* actualmente activa en *Kaggle* realizando el/los envíos que consideren apropiados y reportar en el informe los resultados obtenidos.

Puntos opcionales (no obligatorios)

- Mostrar que si tenemos la descomposición $A = U\Sigma V^t$, V es la misma matriz que obtenemos al diagonlizar la matriz de covarianzas.
- Realizar experimentos utilizando dígitos manuscritos creados por el grupo, manteniendo el formato propuesto.[1] Reportar resultados y dificultades encontradas.
- Implmenentar alguna mejora al algoritmo de kNN.

Programa y formato de archivos

[1] Notar que en la base original las figuras están rotadas y es blanco sobre negro, y no al revés.

Se deberán entregar los archivos fuentes que contengan la resolución del trabajo práctico. El ejecutable tomará tres parámetros por línea de comando, que serán el nombre del archivo de entrada, el nombre del archivo de salida, y el método a ejectutar (0: kNN, 1: PCA + kNN).

Asumimos que la base de datos de imágenes de entrenamiento se llama `train.csv` y que la base de test `test.csv`, y que ambas siguen el formato establecido por la competencia. Para facilitar la experimentación, el archivo de entrada con la descripción del experimento tendrá la siguiente estructura:

- La primera línea contendrá el path a la(s) base(s) de datos, k, α y K.
- Luego, habra K líneas de 42.000 valores, uno por cada muestra de la base de entrenamiento, donde un 1 indicará que esa muestra se considera parte del entrenamiento, y un 0 que se considera parte del test. Luego, de esta forma se pueden codificar las particiones realizadas por el *K-fold cross validation*.

El archivo de salida obligatorio tendrá para cada partición que figure en el archivo de entrada el vector solución con los α valores singulares de mayor magnitud, con una componente del mismo por línea. Además, el programa deberá generar un archivo, cuyo formato queda a criterio del grupo, indicando la tasa de reconocimiento obtenida para cada partición. Adicionalmente, se generará un archivo que concatene la extension `.csv` al segundo valor recibido como parámetro del programa, que escribirá la predicción realizada sobre la base de test en el formato requerido por la competencia siguiendo el formato establecido por las reglas de la competencia.

Junto con el presente enunciado, se adjunta una serie de scripts hechos en `python` y un conjunto instancias de test que deberán ser utilizados para la compilación y un testeo básico de la implementación. Se recomienda leer el archivo `README.txt` con el detalle sobre su utilización.

Sobre la entrega

- Formato electrónico: Jueves 14 de Mayo de 2015, hasta las 23:59 hs., enviando el trabajo (`informe` + `código`) a `metnum.lab@gmail.com`. El `asunto` del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo. Ejemplo: `[TP2] Acevedo, Miranda, Montero`
- Formato físico: Viernes 15 de Mayo de 2015, de 17:30 a 18:00 hs.

6. Apéndice: B

Scripts Experimentos:

```
1  for x in range(0,PRECISION+1):
2    outp = subprocess.check_output([ bindir +"tp", param1, param2, param3,
         param4]).split(',')
3    if (len(outp) < 3): continue
4    microseconds = int(outp[0])
5    if (microseconds != 0):
6        time.append(microseconds)
7  done += 1
8  print("G="+ param4 + " Set=" + test + " Method=" + param3 + ' | ' + str(done) + "of" +
       str(total) + " | Process " + "{0:.2f}".format(float(done*100)/float(total)) + "%" +
       " | " + str(outp))
9  ou.write(test + ',' + str(method) + ',' + str(granul) + ',' +
       str(int(trimmean(sorted(time), 0.25))) + ',' + outp[1] + ',' + outp[2] + "\n")
10 ou.close()
```