DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nº 2

"Reconocimiento de dígitos"

Segundo cuatrimestre de 2017 — Métodos Numéricos

Integrante	LU	Correo electrónico
Luce, Nicolás	294/14	nicolas.p.luce@gmail.com
Cortés Conde Titó, Javier María	252/15	javiercortescondetito@gmail.com
Gomez, Horacio	756/13	horaciogomez.1993@gmail.com
Fernandez, Ignacio Manuel	047/14	nachofernandez.1995@hotmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Resumen

En este trabajo analizamos técnicas que se utilizan para el reconocimiento de dígitos manuscritos. Los algoritmos empleados para esto fueron KNN y PCA. KNN fue utilizado para clasificar los dígitos y PCA para agilizar la ejecucución de KNN en instancias grandes. Con el fin de medir el rendimiento del modelo, hemos realizado pruebas utilizando la base de dígitos manuscritos MNIST[1]. Como resultado, obtuvimos métricas que compiten con las publicadas por otros autores respecto a la misma base de datos[2]. Por último, propusimos una modificación a KNN y comparamos las métricas obtenidas con las resultantes del método convencional.

Palabras clave: Reconocimiento de dígitos manuscritos, KNN, PCA.

[1]Modified National Institute of Standards and Technology - `yann.lecun.com/exdb/mnist/`
[2]Kaggle Digit Recognizer Competition - `kaggle.com/c/digit-recognizer/leaderboard`

Índice

1. Introducción **3**

2. Desarrollo **4**
2.1. Métodos numéricos utilizados 4
2.1.1. PCA 4
2.1.2. *K-fold cross validation* 4
2.1.3. kNN-Modificado 5

3. Experimentación **6**
3.1. Resultados de la experimentación *cuantiativa* 6
3.2. Resultados de la experimentación *cualitativa* 6
3.2.1. Hiperparámetro k 7
3.2.2. Hiperparámetro *alpha* 8
3.2.3. Variación del tamaño del conjunto de entrenamiento 10
3.2.4. Elección del mejor método 10
3.3. Comportamiento del Método seleccionado 12

4. Conclusiones **14**
4.1. Métodos 14
4.2. Knn Modificado 14
4.3. Confusión Simétrica 14
4.4. Misceláneo 14

1. Introducción

Se puede entender que el problema que se busca resolver en este trabajo —reconocimiento de dígitos manuscritos— es una reducción de un problema más general, *Reconocimiento óptico de Caracteres* (OCR[3]).

Para obtener una clasificación eficiente se cuenta con una base de datos de N imágenes previamente etiquetadas con el dígito al que representan, la cual utilizamos como conjunto de entrenamiento para nuestro modelo. Asumiremos que el etiquetado no contiene errores. El objetivo del trabajo consiste en desarrollar un algoritmo supervisado de clasificación que, dadas nuevas instancias, pueda etiquetarlas correctamente.

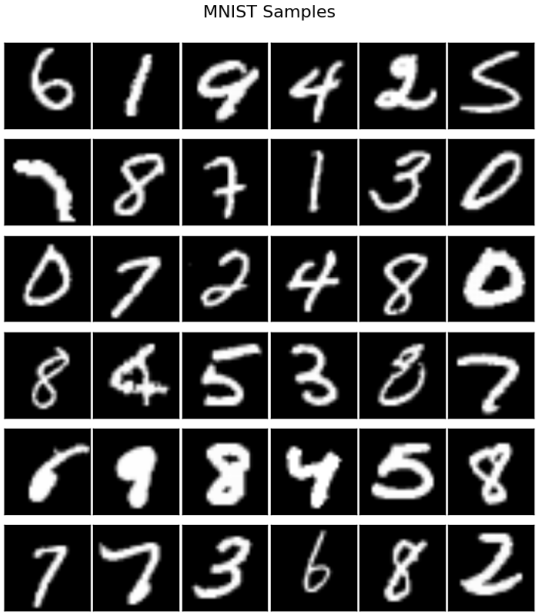


Figura 1: Muestra de la base de entrenamiento.

El algoritmo que usaremos para clasificar es kNN (k-nearest neighbors). Éste considera a cada objeto de la base de entrenamiento como un punto en el espacio euclídeo m-dimensional. Gracias a que conocemos a qué clase pertenece cada uno, podemos clasificar un nuevo objeto según la moda de las clases de los k puntos más cercanos al mismo.

El problema de este algoritmo es que su costo temporal aumenta drásticamente a medida que aumenta la cantidad de dimensiones. Por este motivo se utilizará otro método numérico: PCA (*Principal Component Analysis*). Éste reduce la cantidad de dimensiones de las instancias al quedarse sólo con las que más información aporten, logrando que el tiempo de cómputo de kNN no sea tan elevado y que la instancia no tenga tanta información redundante.

En la sección de experimentación se buscará analizar la *calidad* de las soluciones para diferentes valores de k comparando los resultados de instancias que no hayan sido preprocesadas.

[3] *Optical Character Recognition*

2. Desarrollo

2.1. Métodos numéricos utilizados

2.1.1. PCA

La implementación de PCA en el trabajo surge de la necesidad de reducir la cantidad de dimensiones de las muestras ya que nos permite descartar las variables que aportan poca información. A grandes rasgos el algoritmo de PCA consiste en dos partes:

- Reescribir los datos en una base en la que las variables no se encuentren correlacionadas. Esta transformación se define de manera tal que el orden de las coordenadas se corresponde con el grado de información que aportan (la primer coordenada aporta más información que la segunda y así sucesivamente).
- Reducir la cantidad de dimensiones en las que se expresan los datos. Esto es posible debido a que, en la nueva base, las últimas coordenadas aportan poca información y podrían ser eventualmente descartadas. Por eso al agarrar las primeras *alpha* coordenadas reducimos la cantidad de dimensiones. Esto lleva a una potencial perdida de información, por lo que debemos encontrar un equilibrio entre la optimización temporal y la pérdida de información.

Para calcular la base se utilizó el método de potencias con deflación. Este algoritmo tiene un parámetro *iter* que indica la cantidad de iteraciones que se deben hacer para aproximar el autovalor asociado al autovector.

La relación de éste método con nuestro trabajo es que lo vamos a utilizar para cambiar de base nuestras imágenes y tratar de obtener un clasificador más eficiente en términos temporales.

Algoritmo 1: PCA

Datos: $X \in \mathbb{R}^{n \times m}$ matriz donde cada fila es una imagen del training , μ_i que contiene el promedio de la columna i de X

Resultado: $\bar{X}$, $\in \mathbb{R}^{n \times m}$, donde $\bar{X}$ es la matriz de cambio de base

para $i \leftarrow 1$ **a** n **hacer**
  $X^{(i)} \leftarrow \left(X^{(i)} - \mu_i\right)/\sqrt{n-1}$
fin
$Mx \leftarrow X^t X$
$\bar{X} \leftarrow$ baseAutovectores(Mx)

2.1.2. *K-fold cross validation*

Con el fin de validar y comparar el desempeño de nuestros modelos utilizamos *k-fold cross validation*. Este método de validación se encarga de particionar el conjunto de entrenamiento en k subconjuntos de igual tamaño para luego clasificar cada uno de ellos utilizando a los restantes como conjunto de entrenamiento. La manera en la que particionamos el conjunto mantiene la distribución de los dígitos de la base original con el fin de mantener la proporción de representantes de cada clase.

Como mencionamos, este método es crucial para comparar modelos, por eso hay que tener un cuidado especial y usar las mismas particiones para todos los modelos ya que queremos tener cierto control sobre el desempeño.
Al obtener métricas provenientes de diferentes conjuntos de entrenamiento y test podemos predecir con mayor confianza cómo se comportará el modelo.

2.1.3. *k*NN-Modificado

La heurística original de kNN toma los k vecinos más cercanos a la instancia. Donde k es un hiper parámetro del algoritmo, dejando en manos del programador realizar los experimentos necesarios para concluir un valor "óptimo" de k. Si k es demasiado permisivo, se corre el riesgo de incluir instancias demasiado alejadas que afecten al resultado final. No estando a gusto con este escenario decidimos modificar kNN de forma tal que deje de considerar un número arbitrario de vecinos.

La modificación planteada es la siguiente: sea s_i la distancia máxima entre dos instancias de clase i de la base de entrenamiento, definimos $S = \max_{0 \leq i \leq 9}(s_i)$.

Luego, $\forall v \in V$, $distancia(I, v) \leq \frac{S}{2}$.

Algoritmo 2: kNN-Modificado

Datos: $A \in \mathbb{R}^m$, $traning_set \in \mathbb{R}^{n \times m}$, $S \in \mathbb{R}$
Resultado: $etiqueta \in \{0, ... ,9\}$
Complejidad: $\mathcal{O}(n \cdot log(n) \cdot m)$, donde m es el tamaño de las instancias.

```
1  Heap ← Vacío
2  para cada I ∈ traning_set hacer
3  |  d ← distancia(A, I)
4  |  Heap.push(< etiqueta(I), d >)
5  fin
6  Etiquetas ∈ {0, ... ,9}^10
7  mientras Π2(Heap.top) ≤ S/2 hacer
8  |  e ← Π1(Heap.top)
9  |  Etiquetas[e] ← Etiquetas[e] + 1
10 |  Heap.pop()
11 fin
12 etiqueta ← max(etiquetas)
```

3. Experimentación

En esta sección analizaremos los resultados obtenidos por medio de cada método implementado —kNN, kNN + PCA, kNN-Mod— proponiendo diferentes hipótesis acerca de cómo debería comportarse esta información. En base a diferentes análisis buscaremos responder las siguientes preguntas:

- ¿Cómo afectan los hiperparámetros de cada método a la calidad de los resultados? ¿Qué configuración resulta óptima?
- ¿Hasta qué punto reducir o ampliar la base de entrenamiento mejora la calidad de los resultados?
- ¿Podemos generalizar una fórmula que resulte óptima para todas las clases?

La calidad de los resultados obtenidos será analizada mediante el *accuracy*, la cantidad de dígitos correctamente clasificados respecto al total; y el *F1-Score*, utilizado para abarcar las medidas de *precission* y *recall* de los tests. A modo de presentar la mayor cantidad de información posible para cada configuración, y por la gran cantidad de parámetros que probamos, consideramos que la mejor opción para mostrar los resultados es simplemente graficar los promedios de cada métrica por cada K-fold.

3.1. Resultados de la experimentación *cuantiativa*

Para la experimentación cuantitativa tomamos un conjunto de entrenamiento fijo, es decir fijamos los parámetros K de K-fold en 6 y K de KNN en 3 y variamos *alpha* entre 30 y 50. El objetivo es medir el tiempo que tarda en obtener las componentes principales y la clasificación de las imágenes. No se tuvieron en cuenta los tiempos de lectura de datos y de separación de datos para entrenar y para clasificar.

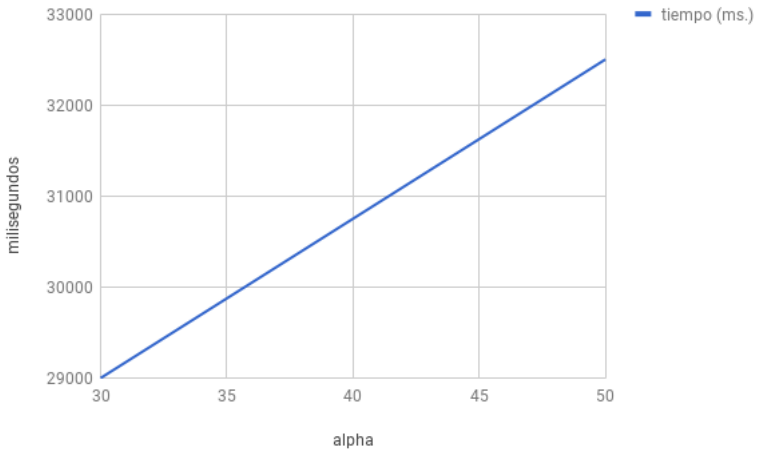


Figura 2: Resultados cuantitativos variando **alpha**

Era esperable que a medida que aumentáse el hiperparámetro *alpha* aumentara la cantidad de tiempo que toma realizar las clasificaciones ya que al tomar más componentes principales, aumenta la cantidad de dimensiones que representan a una imagen.

3.2. Resultados de la experimentación *cualitativa*

Debido a la cantidad de configuraciones posibles, resulta poco práctico mostrar la totalidad de los resultados obtenidos. Adjuntamos al trabajo un *Jupyter Notebook* en el que se

pueden encontrar dichos resultados. Los gráficos presentados en el informe son aquellos que consideramos más representativos.

3.2.1. Hiperparámetro *k*

El hiperparámetro con el cual experimentamos primero fue k de kNN sin transformar los datos con PCA. Decidir qué cantidad de vecinos es la óptima es crucial para clasificar las instancias. Consideramos que k no debería ser demasiado grande, ya que añadiríamos instancias demasiado alejadas al conjunto de vecinos. Sin embargo, con un valor demasiado bajo se corre el riesgo de que el conjunto de vecinos esté conformado principalmente por vecinos no representativos.

Desgraciadamente, debido al extenso tiempo de ejecución de la experimentación, nos vimos forzados a usar valores bajos de k. Analizamos en estos la fluctuación del *accuracy* en los resultados.

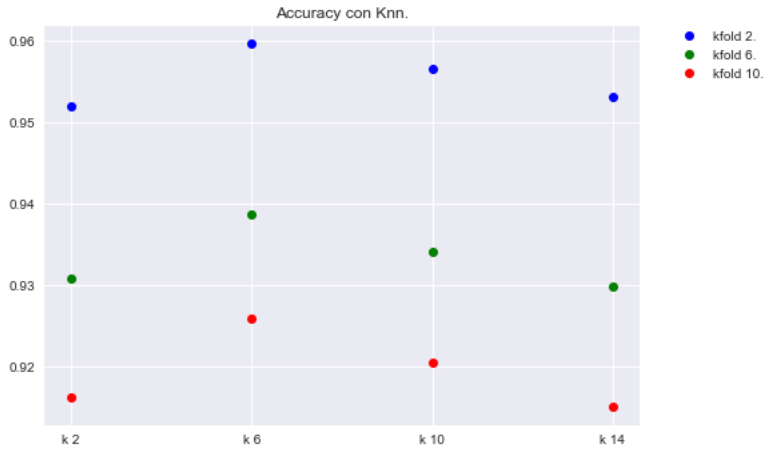


Figura 3: Resultados cualitativos de kNN

Dentro de los valores analizados podemos observar en la figura 2, para distintos valores de K-fold, $k = 6$ resulta ser aquel que clasifica con mayor *accuracy*. A partir de estos datos decidimos aumentar la granularidad en la etapa de experimentación con PCA para tener los antes vistos y todos los impares menores a 10.

3.2.2. Hiperparámetro *alpha*

Para comprender como podrían afectar los distintos valores de *alpha*, resulta más sencillo considerar los casos extremos. Si el valor de *alpha* fuera mínimo, se estaría descartando información relevante, perdiendo así la representatividad de las instancias. Por el contrario, si *alpha* fuese máximo, se estarían teniendo en cuenta variables que aportarían más ruido que información, por lo que estaríamos introduciendo errores en nuestro modelo.

Definimos a $f : \mathbb{N} \to [0, 1] \ / \ f(\alpha)$ como la función que determina el *accuracy* dado un *alpha*.

Hipótesis: f tiene un único máximo.

Esta hipótesis se basa en la idea intuitiva de que, como PCA ordena las componentes para maximizar la varianza, al agregar más a la base es probable que en cierto punto dejemos de importar información crucial para clasificar y simplemente agreguemos ruido. Puede ser que existan dos máximos, pero suponemos que estos van a ser alphas aledaños (por ejemplo el 35 y el 36).

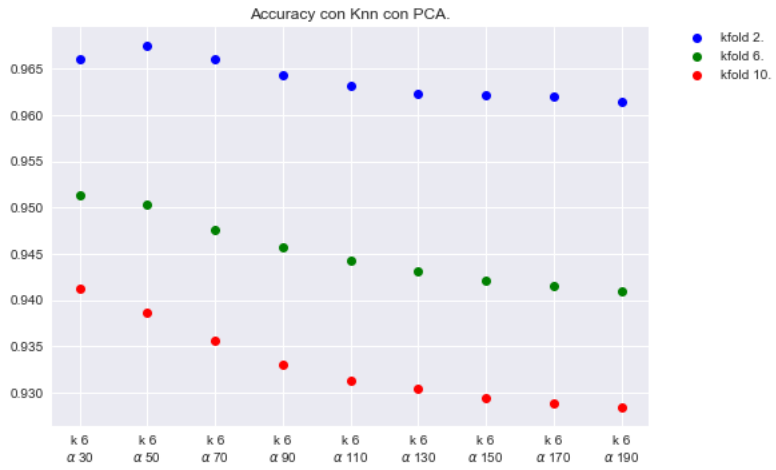


Figura 4: Resultados cualitativos de kNN con PCA, $k = 6$

En primer lugar podemos observar que para diferentes valores de k y distintos tamaños de folds los resultados se comportan de la misma manera —ver *Jupyter Notebook* para los distintos valores de k. Por otro lado, vemos que nuestra hipótesis parece ser correcta ya que existe $\alpha \in (30, 190)$ que es máximo de f. Si bien no probamos con valores de α que recubran todo el rango posible, decidimos experimentar utilizando una granularidad mayor dentro del intervalo (30, 70) para ver como se comporta en el intervalo en el que parece encontrarse el máximo. A continuación mostraremos el gráfico con esta nueva granularidad para poder analizar en más detalle el comportamiento del hiperparámetro alpha.

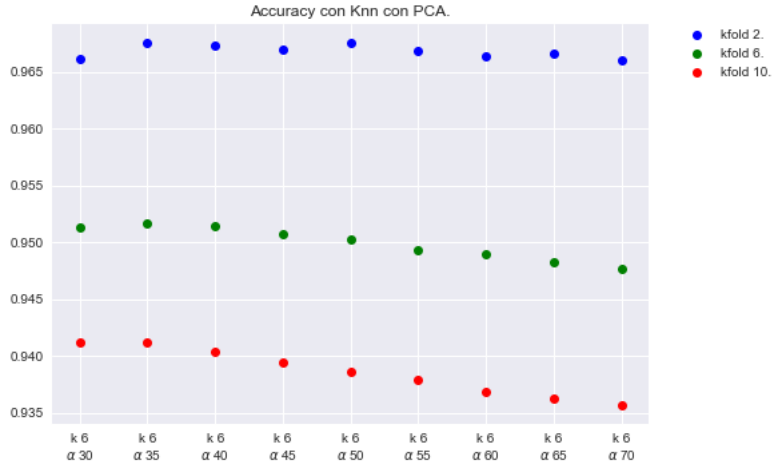


Figura 5: Resultados cualitativos de kNN con PCA, $k = 6$

Aunque es muy difícil distinguir, pero en los folds distintos a dos, se ve claramente que el máximo es alpha 35, en cambio el fold dos tiene dos posibles máximos, el 35 y el 50, al analizar esos dos casos por separados los valores son extremadamente similares, esto puede refutar nuestra teoría, pero también tenemos que tener en consideración que solo son dos medidas por cada alpha.

También vale aclarar que para distintos folds o distintos k el máximo alpha no tiene por que ser el mismo, aunque en los datos que analizamos encontramos los mismos resultados que lo mostrado en (ver jupyter notebook).

3.2.3. Variación del tamaño del conjunto de entrenamiento

Con el fin de poder entender qué tan útil es reducir o ampliar la base de entrenamiento, variamos la cantidad de folds con los que realizamos k-*fold cross validation*. Esperábamos que al aumentar el tamaño de la base de entrenamiento mejoraran nuestros resultados, pero no fue así. Como podemos observar en los gráficos 2, 3 y 4, empeoraron.

Como esto no era lo que esperábamos hicimos un análisis exhaustivo de los algoritmos para encontrar errores, el primer error que nos hicieron notar fue que PCA cambiaba de base con una matriz que fue creada sobre todos los datos y no sobre los datos de entrenamiento, al cambiar el código corrimos nuevamente los experimentos y nos topamos con la misma relación. Cuando miramos el gráfico 2 es el algoritmo Knn sin PCA, entonces el error no podía encontrarse ahí, otro eje en el cual decidimos investigar fue en la creación de los folds y si había solapamiento entre los datos de entrenamiento y los de testeo, lamentablemente esta rama no resultó fructífera para aclarar el dilema. La última rama que se nos ocurrió chequear fue que estuviéramos graficando o calculando mal el *accuracy* aunque, luego de analizar el código, vimos que este no tiene problemas para hacer lo antes mencionado.
Para validar nuestros resultados decidimos correr el algoritmo de knn con los datos de entrenamiento siendo cada fold y los de testeo los no clasificados dados por la competencia de kaggle y que la pagina se encargara de analizar el accuracy, con esto nos aseguramos de no tener solapamiento de datos entre los conjuntos y, de no tener problemas al cambiar de base los datos y calcular mal el accuracy.

Cuadro 1: Accuracy según Kaggle con Knn de dos vecinos.

fold	accuracy
2	0.95442
6	0.92971
10	0.91400

Como se puede observar Kaggle nos da los mismo resultados que los obtenidos en la experimentación. Si bien no era lo que esperábamos que sucediera, logramos encontrar una explicación para este *"fenómeno"*. Al variar el parámetro k de k-*fold cross validation*, lo que hacemos es (a medida que éste aumenta) agrandar la base de entrenamiento y disminuir a su vez el conjunto de testeo. Esto último puede llevarnos a que al tener pocos datos sobre los cuales testear (y a partir de los cuales generar las métricas), una misma cantidad de errores en las clasificaciones empeore aún más nuestras métricas. Nos queda pendiente realizar pruebas similares en las que se mantenga el tamaño del conjunto de testeo con el fin de entender si existe o no una cantidad excesiva de datos que empeore las clasificaciones.

3.2.4. Elección del mejor método

Teniendo 10 *F1-Score* por cada método (1 por clase). Con el fin de decidir cual es el mejor método, tomamos el promedio de cada *F1-score* y los ordenamos de menor a mayor. Vamos a hacer un *ranking* para determinar el mejor método. Cada método recibe un puntaje entre 1 y 5 por cada clase. Como vimos que algunas métricas variaban muy poco entre distintos métodos para algunas clases, pensamos que había clases con mayor y menor importancia. Debido a esto es que decidimos asignar un peso a cada clase. Esto se logró tomando el promedio más chico y el más grande de cada clase y restándolos. Como esto nos da un valor decimal, lo multiplicamos por mil y nos quedamos con la parte entera. Multiplicamos los puntajes de esa clase por ese valor dejándonos para cada método un puntaje por clase que fue modificado según cuanto varió el *F1-score* en esa clase. Al sumarlo obtenemos a nuestro ganador.

La razón por la cual decidimos asignar un peso por cada clase es por la diferencia que hay entre el máximo y mínimo del valor de *F1-Score* en cada una. Una primer intuición nos dice que si una clase tiene poca diferencia quiere decir que la clase tiene poco "peso". Entonces, la asignación del puntaje para cada método va a ser baja ya que para la clase le es lo mismo elegir cual método es el mejor debido a su poca diferencia entre ellos. Sin embargo , si nos basamos en esta diferencia, podríamos tener outliers que podrían estropear un sistema de puntaje balanceado. Por este motivo tomamos el promedio de cada método y calculamos la mayor diferencia. Eso nos determina el peso de cada clase. En el siguiente gráfico se muestran los resultados para los distintos métodos de la clase 0 con K-fold = 6, que además es la clase que menor "peso". Los puntos blancos en el centro de cada método representan el promedio que nos servirá para calcular el peso. Notar que el peso esta multiplicado por 1000 para obtener valores enteros.

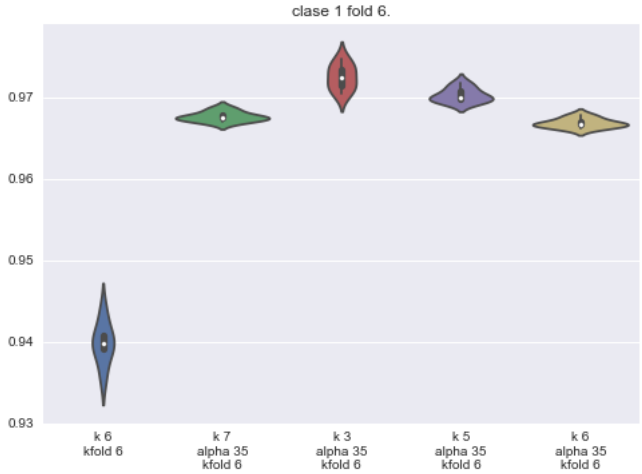


Figura 6: Comparación de métodos para la clase 1, utilizando *F1-Score*.

Con la siguiente tabla podremos ver como quedó la elección del mejor método basándonos en el sistema de votación que propusimos. Elegimos 5 algoritmos distintos, el mejor knn sin usar PCA (que teníamos según nuestro análisis anterior para comparar como funcionaba el algoritmo con y sin PCA), y los 4 mejores knn usando PCA. Decidimos mostrarla para K-fold = 6. Por columnas tenemos los 5 métodos que son:

1. ***k*NN**: 6;
2. ***k*NN**: 3; ***alpha***: 35;
3. ***k*NN**: 5; ***alpha***: 35;
4. ***k*NN**: 6; ***alpha***: 35;
5. ***k*NN**: 7; ***alpha***: 35;

En la posición (i, j) de la tabla se puede ver el puntaje que le asignó la clase i al método j. De la tabla anterior se pueden extraer los pesos totales de cada método:

El resultado que se puede extraer del tercer cuadro es que el método (kNN = 5 y *alpha* = 35) para K-fold = 6 es el que más puntaje recibió y por ende es la mejor combinación de parámetros según nuestro criterio.

Si analizamos en detalle el cuadro 2 podemos observar que varios métodos tienen una clase con la que se desempeñan mejo. El algoritmo que se desempeñó mejor solo gan+o en

Cuadro 2: Puntaje de los métodos

	KNN 6	KNN 7 alpha 35	KNN 3 alpha 35	KNN 5 alpha 35	KNN 6 alpha 35
0	8	24	16	40	32
1	32	96	160	128	64
2	17	34	51	85	68
3	8	32	16	40	24
4	6	18	12	24	30
5	15	45	30	75	60
6	4	8	20	16	12
7	17	34	85	68	51
8	25	50	125	100	75
9	10	20	40	50	30

Cuadro 3: Puntaje de cada método

Método	Puntaje total
KNN 6	142
KNN 7 alpha 35	361
KNN 3 alpha 35	555
KNN 5 alpha 35	626
KNN 6 alpha 35	446

5 categorías. Al ver este cuadro podemos entender que si cambiamos el objetivo de nuestro algoritmo y solo analizamos un subconjunto de dígitos, nuestra elección podría no ser la óptima.

3.3. Comportamiento del Método seleccionado

Dentro del análisis del método, al ser un clasificador multi-clase sobre dígitos, definimos un concepto llamado confusión simétrica. Sean X e Y clases del clasificador, la confusión simétrica$_{x,y}$ es qué tanto se parece la predicción de la clase Y cuando en realidad es la clase X con la predicción de la clase X cuando en realidad es la clase Y. Normalmente el ser humano se confunde tantas veces el 4 con el 9 como el 9 con el 4, al fin y al cabo lo que queremos ver es que tanto se asemeja el comportamiento de nuestro clasificador a este error humano.

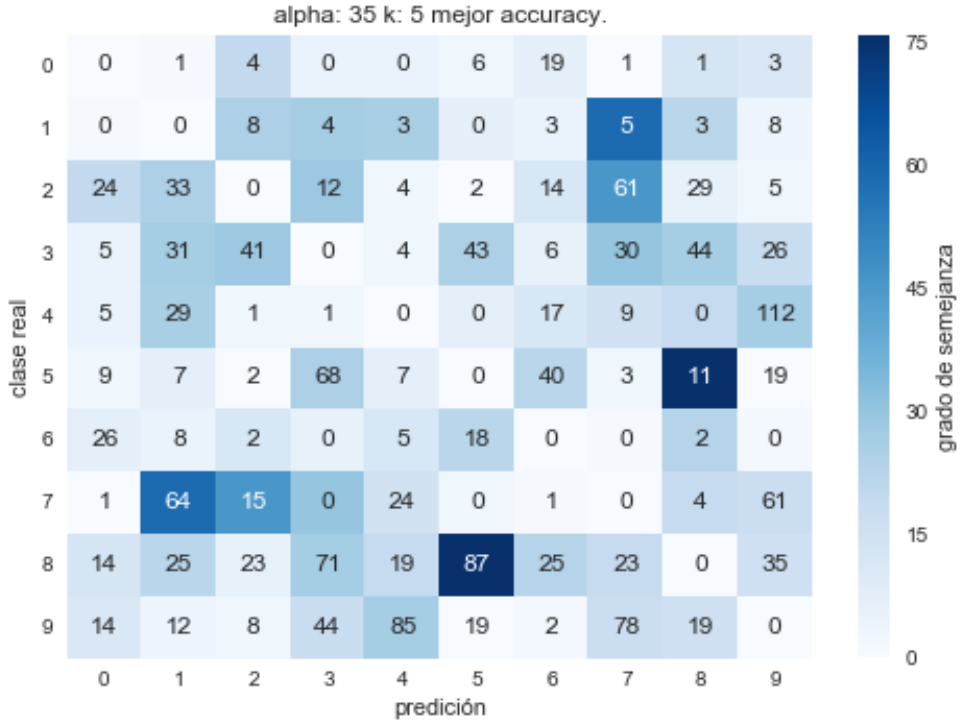
Al tener la matriz de confusión M, la forma naïve de ver esto es hacer M - M^t y graficar un heatmap con el resultado. Esto tiene muchas falencias.

- El heatmap nos va a dar valores que indican la diferencia de la resta. Consideremos dos ejemplos, en el primer heatmap la posición (X, Y) tiene un valor de 10 (es decir X predijo 10 veces mas a Y que Y a X), pero Y nunca predijo X. En el segundo heatmap la posición (X, Y) tiene un valor de 20 (es decir X predijo 20 veces mas a Y que Y a X), pero Y predijo 100 veces a X. Se puede argumentar que el segundo heatmap tiene una fuerte confusión simétrica$_{x,y}$ y la primera una muy baja. Para subsanar el problema, las diferencias son representadas en color, como cualquier heatmap y en un cuadrado (X, Y) se escribe cuantas veces el clasificador predijo la clase Y cuando en realidad era la clase X, es decir la matriz de confusión (exceptuando la diagonal, ya que no queremos añadir información que no vamos a analizar).

- Con la solución previa, generamos un heatmap con mucha información y difícil de leer. si la posición (X, Y) tiene un valor N, la posición (Y, X) tiene un valor $-N$, generando

así dos colores distintos. Eso sumado a que cada cuadrado tiene un número distinto al valor que representa el color del heatmap, lleva a un gráfico extremadamente difícil de interpretar. Para tratar de hacer un gráfico más simple, al hacer M-M^t nos quedamos con los valores absolutos, generando generando un solo color para las posiciones (X, Y) y (Y, X).

Para ver en más detalle el comportamiento de la confusión simétrica para el método elegido, utilizamos el $kfold = 6$ y buscamos los folds que brinden el mejor y peor *accuracy*. Como la diferencia no es sustancial, mostramos sólo uno en el informe (aunque dejamos ambos en el *Jupyter Notebook* adjunto).



En el gráfico, aunque sigue siendo bastante confuso, se pueden seleccionar dos pares, el (9,4) que tiene un alto grado de confusión simétrica ya que las predicciones fueron altas pero el grado de disparidad fue bajo; y el (8,5) que tiene un bajo grado de confusión simétrica ya que el grado de disparidad es alto y los valores son lo suficientemente altos como para poder analizarlos.

4. Conclusiones

4.1. Métodos

Al evaluar tantos métodos distintos podemos concluir que, de los pre-seleccionados, no hay uno que sea mejor que todos para todas las clases. Por esto fue necesario generar distintas métricas, el *F1-score* promedio y el sistema de puntajes antes mencionado para quedarnos con uno. El método seleccionado fue *KNN* con *PCA* con los hiperparámetros k: 6 y α: 60.

4.2. Knn Modificado

Esta modificación no dio ninguna mejora, aunque la idea parecía tener sentido nos quedamos con un clasificador que daba un F1-score por debajo del 0.5 para todas las clases (exceptuando la del 0). Aunque no mostramos los resultados en el infrome, ante un análisis exhaustivo del algoritmo notamos que esto sucede debido a que toma en cuenta muchos vecinos (aproximadamente 2000), lo que era contrario a la idea del algoritmo propuesto. Como no se tuvo tiempo para utilizar otras medidas de distancia o analizar los clusters de clases (para ver si cada clase tenía un único cluster o si eran distintos), quedará como un trabajo para realizar a futuro. Esta heurística tiene muchísimas variaciones distintas posibles, algunas de las interesantes son:

- Poner como condición de corte el k de knn y la distancia a la misma vez.
- En vez de seleccionar una distancia, usar varias a la vez, es decir, tener tantos candidatos como cantidad de distancias y que estos candidatos sean los que clasifican realmente a la imagen. Es decir agregar una fase mas de votación. A diferencia del Meta Knn (mencionado más adelante), esto no tiene una complejidad temporal mayor, ya que se sigue recorriendo la misma pila.

4.3. Confusión Simétrica

Al definir una nueva métrica nos encontramos con dificultades:

- Generar gráficos simples e informativos
- Encontrar una forma sencilla de definir el grado de confusión simétrica para un par (X, Y)
- Encontrar una forma de normalizar los datos para poder comparar distintas matrices de confusión generadas con distinta cantidad de datos de testeo y a la misma vez poder encontrar un único valor que resuma la confusión simétrica total para un método.

Como trabajo a futuro, nos gustaría poder superar al menos alguna de las dificultades antes mencionadas.

4.4. Misceláneo

Por cuestiones de tiempo no logramos realizar toda la experimentación que consideramos interesante para la realización del trabajo práctico. En varias etapas nos encontramos con preguntas que, para ser respondidas se requería generar una gran cantidad de resultados nuevos. Como esto no resultó viable, presentamos dichos cuestionamientos en la siguiente lista.

- PCA: ver como cambia la calidad de las respuestas en base a la cantidad de iteraciones que hace el método de potencias, o cambiar el enfoque y decidir frenar cuando la

diferencia entre el vector previo y el calculado sea lo suficientemente chica (un hiperparámetro). Esta última parece tener más sentido que usar una cantidad de iteraciones arbitraria ya que podríamos manejar mejor cuál es el error que aceptamos.

- Métodos:
- Meta Knn: Como mencionamos en la conclusión, no hay un método que sea mejor para todas las clases, eso nos deja la idea de tener varios knn que clasifiquen y luego voten por cuál sería la clasificación final. Esto tiene sus problemáticas de tiempos de ejecución si no se paralelizan las clasificaciones.
- Knn: darle un peso a los vecinos más cercanos, con alguna métrica. Esto parece seguir la idea del algoritmo knn.