Reconocimiento de Patrones
Proyecto estandar

Diciembre 2017

Integrantes: Franciso De Izaguirre, Felipe Tambasco

Tutor: Pablo Musé

Resumen

En el presente documento se presenta el trabajo el realizado por el grupo, el mismo se realizó a partir de los conocimientos obtenidos durante el curso aplicados a la base de datos **Fasion MNIST (F-MNIST)**.

Dentro de los métodos de selección se utlizaron PCA y Kernel PCA (K-PCA), mientras que para clasificación se utlizaron, Redes Neuronales Multicapa (NN-MLP), Maquinas de Vectores de Estado (SVM), y finalmente como método de Bagging/Boosting se incursionó con RandomForest.

1. Introducción

Este proyecto trabajó sobre la propuesta estándar del curso. El cual tiene como pauta el desarrollo de un sistema de Reconocimiento de Patrones, donde se trabaje la extracción y selección de características y la clasificación. En particular buscar una mejor separación de las clases que más se confunden según se pueda observar en las matrices de confusión de los métodos.

Se trabajó con la base de datos Fashion Mnist, y al final se corroboraron los resultados obtenidos validando contra un conjunto de test brindado para éste fin.

1.1. Objetivo

El objetivo es conseguir un error inferior a 10 % con respecto al conjunto de Test brindado por el plantel docente. En lo académico profundizar y mejorar la comprensión de los temas del curso, ganando intuición en el transcurso del proyecto.

1.2. Descripción del problema

Se tiene la base de datos Fashion MNIST con sus 60.000 muestras (patrones) y 784 características por patrón. El problema de mejorar la clasificación de la base de datos brindada suple el clasico problema de clasificación que proponía la base de datos MNIST (contenía numeros manuscritos), siendo ahora el nuevo desafío para el área conocida como Machine Learning.

1.3. Antecedentes

Como se dijo en la descripción, el problema que se busca abarcar es mundialmente conocido, por lo que es fácil encontrar Benchmarks de pruebas realizadas por otras personas con distintos metodos de clasificación, los cuales fueron tomados como puntos de partida[1], principalmente para saber qué resultados de accuracy esperar. A continuación se listan algunos:

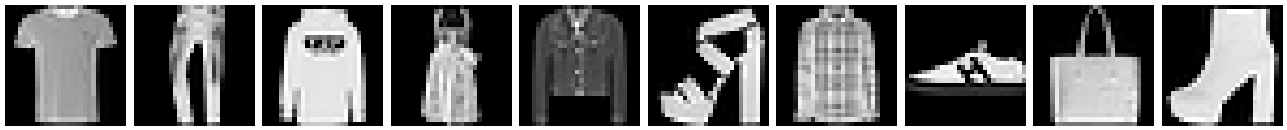
2. Datos

El conjunto de datos es el brindado por el equipo docente, siendo el mismo la base de datos fashion-mnist ya mencionada. La misma cuenta con 60,000 patrones, los cuales están igualmente distribuidos en 10 clases, también se contó al final del proyecto con una base de datos para Test con 10,000 muestras. Estos patrones son imágenes de

[1] Los Benchmarks utilizados son los que figuran en la siguiente URL: **https://github.com/zalandoresearch/fashion-mnist**

Clasificador	Parámetros	Accuracy
SVC	C:10,"kernel":"poly"	89,7 %
Multilayer Perceptron	256-128-64	90 %
Random Forest	çriterion":."entropy","max_depth":50,"n_estimators:100	87,9 %
Ser Humano[2] (3 Categorías posibles)	1000 muestras al azar para personas sin conocimiento de moda	83 %

ropa con un tamaño de 28x28 pixeles, lo cual brinda 784 características. La base de datos está distribuida en las siguientes clases :



(a) Camiseta (b) Pantalon (c) Pulover (d) Vestido (e) Saco (f) Sandalia (g) Camisa (h) Zapatilla (i) Bolso (j) Bota

Figura 1: Muestras de la base de las Catergorias de datos

3. Herramientas

Las herramientas de programación utlizadas fueron varias, se destacan especialmente la librería scitkitLearn para Python, Weka, y las funciones "kmeansz "kmedoids"de Matlab. En lo que respecta a Hardware, fueron utilizadas los ordenadores de los componenetes del grupo y el cluster de Facultad Ingeniería de la Universidad de la República.

4. Rendimientos

Esta sección está destinada a expresar los rendimientos obtenidos mediante la implementación de los distintos métodos. Así como también una Introducción a los métodos utilizados tanto en selección como en extracción y en clasificación. Se espera también que ilustre el porque de los resultados obtenidos. En las siguientes subsecciones también se analizarán resultados mediante matrices de confusión.

4.1. Selección y extracción de características

En principio el conjunto de datos no cumple la regla empírica conocida como "La maldición de la dimensionalidad" (Curse of dimensionality). Dicha regla enuncia que la relación existente entre la cantidad de muestras disponibles y la cantidad de features debe ser mayor a diez veces la cantidad de clases. Por lo cual en este caso, se debería contar con al menor 78400 muestras, cuando se cuenta con 60000. Este simple análisis nos alentó en búsqueda de selectores y extractores de características, y con estos lograr una reducción de características que permita trabajar con el conjunto de datos sin ser afectado por "la maldición".

Se trabajó con dos extractores clásicos, PCA y KernelPCA, y se realizó Wrapper Selection para la selección de características. Como clasificadores se utilizó Random Forest principalmente por su gran velocidad y simpleza, redes neuronales multicapa y SVM.

4.1.1. PCA

En la figura 2 se puede apreciar el aporte a la varianza proporcionado por los valores propios de la matriz de varianza de las distintas características.

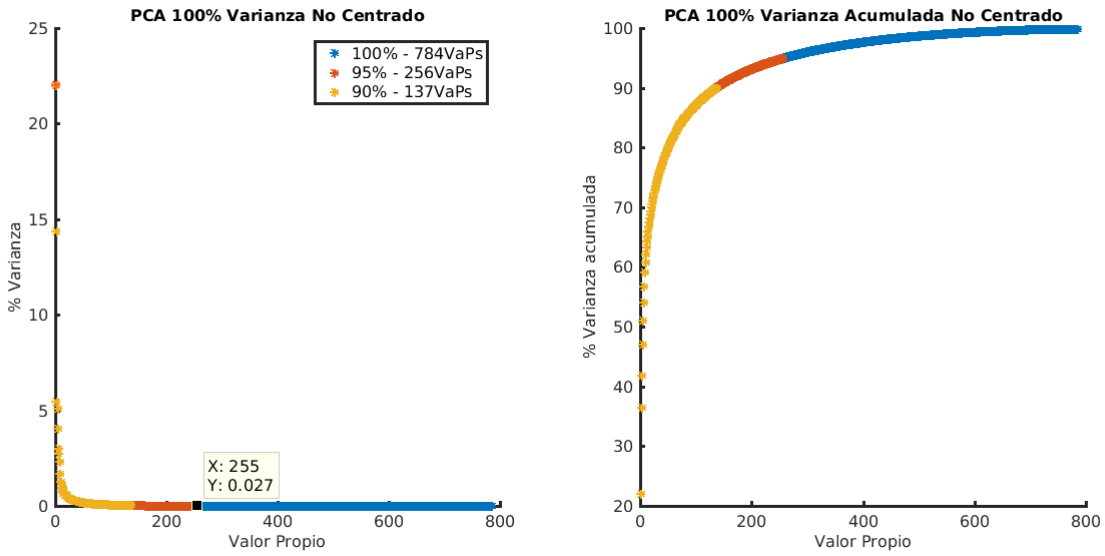


Figura 2: Varianza que aporta cada vector propio y varianza acumulada

Aquí vemos cómo éste método permitiría una reducción significante de muestras. En caso que se quisiera acumular el 90 % de la varianza, alcanza con quedarse con los primeros 137 vectores própios de la matriz, e incluso si se quisiera ser más exigente y acumular el 95 % de la varianza alcanza con los primeros 256 vectores propios como base vectorial.

Con el objetivo de hacer un estudio cuantitativo sobre qué tan bien representan las componentes principales al problema original, se entrenó un clasificador RandomForest sobre k componentes principales, y luego se midió el rendimiento de clasificación al conjunto test proyectado sobre esta base de k componentes. En la figura 3 se muestran estos resultados, con la cantidad de componentes principales en escala logarítmica para una mejor visualización. Se incluyen también los resultados obtenidos en la clasificación del conjunto de validación, que refieren a un 20 % de datos del conjunto original.

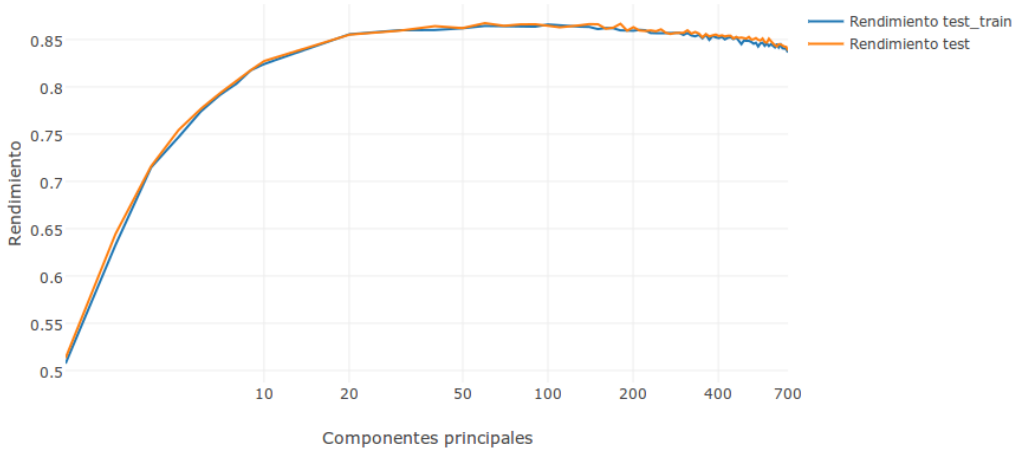


Figura 3: Rendimiento alcanzado en función de cantidad de componentes principales para RandomForest.

Aquí se ve que los mejores rendimientos se alcanzan entre 20 y 200 características. Para una menor cantidad el

rendimiento es notoriamente malo, que es consistente con el hecho de que los datos no son bien representados con tan poca cantidad de características. Para una cantidad mayor vemos que el rendimiento empieza a bajar lentamente, mostrando también un comportamiento oscilatorio, como de "ruido". Esto es contra-intuitivo, y puede deberse a que una cantidad mayor de características en realidad "confunden" al clasificador, aunque estas efectivamente represnten mejor al problema original. También puede deberse a que para estos experimentos se fijaron los parámetros del clasificador, tanto la cantidad de árboles como la profundidad máxima fueron fijadas en 100, por lo que no se descarta que existan otra combinación de parámetros que aumenten el rendimiento en esta parte de la gráfica. Por otra parte, las diferencias en la representación también pueden deberse a efectos de muestreo, dado que existe una cantidad mucho mayor de puntos entre 200 y 700 características, que entre 0 y 200.

También es interesante notar que los rendimientos en validación y test son prácticamente idénticos.

Luego, en la figura 4 concentramos la información aportada por las figuras 2 y 3 al mostrar el rendimiento en función de la varianza acumulada por la cantidad de características.

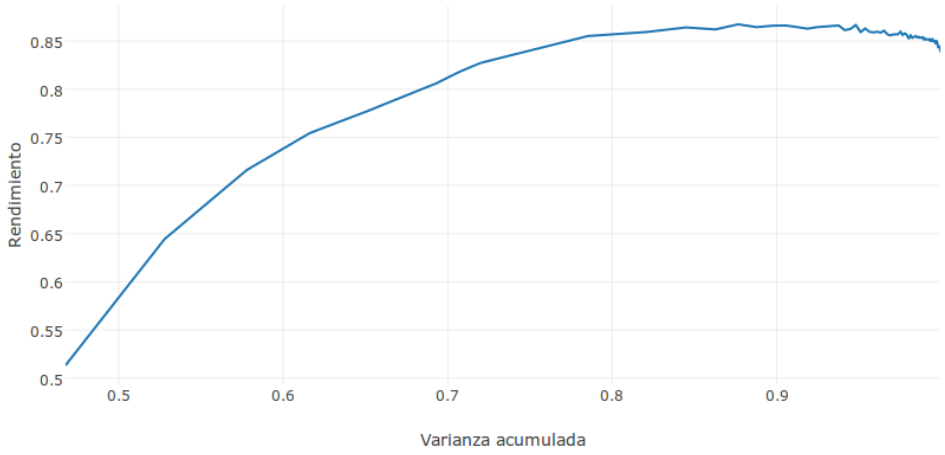


Figura 4: Rendimiento alcanzado en función de la varianza acumulada por los componentes principales para RandomForest.

Aquí vemos que el rendimiento es creciente hasta acumular un poco más del 90 % de la varianza, y luego muestra ese comportamiento ruidoso y decreciente visto antes.

Hubiera sido bueno ver resultados similares para otros clasificadores, como SVM y redes neuronales, pero no se hicieron por el alto tiempo de cómputo que implican.

4.1.2. Kernel PCA

Kernel PCA ofrece la ventaja que brinda trabajar en un espacio de dimensión mayor, pero sin trabajar en un espacio de dimensión mayor gracias a ser un kernel "trick". Permitiendo una versión no lineal de PCA, pudiendo tener tantos vectores própios como la cantidad de muestras. Durante el desarrollo del proyecto terminó siendo la piedra angular del viraje que tuvo el proyecto, producto de que procesar un kernel PCA requiere de mucho cómputo cuando la cantidad de muestras son tantas, pues en éste caso requiere de trabajar temporalmente en un espacio vectorial de 60000x60000 al hallar la matriz de Gram para el kernel, trabajando con float32 esto implica que la memoria ram para ésta matriz es **60000*60000*32=115,2 GB**. Ésto llevo a un empecinamiento por intentar utilizar el algoritmo, derivando primero en la sugerencia de utilizar los centroides[3], y posteriormente en la utilización del cluster de la Universidad como herramienta principal de procesamiento.

En búsqueda de éste procesamiento se pidio en el cluster el nodo con mayor cantidad de memoria RAM, el mismo es el **nodo 71**, el cual cuenta con **128 GB de RAM y un procesador Intel(R) Xeon(R) CPU E5-2680 v3**

[3]El trabajo realizado con los mismos se puede ver en la sección correspondiente

@ 2.50GHz con 24 núcleos

En los ensayos hechos con los centroides se destaca el uso del kernel polinómico

Desafortunadamente, aún reservando todo el potencial del nodo71, no se pudo procesar el algorítmo.

4.1.3. Wrapper Selection

Con el objetivo de reducir la dimensionalidad de los datos, se probó eligiendo k características al azar del total. Reduciendo de esta forma la dimensión del espacio de características, de 784 a k.

Luego de esta selección, con el fin de evaluar si estas características son representativas del total, se uso el 80 % de estos datos para entrenar un clasificador RandomForest, y se validó con el 20 % restante. Se hicieron pruebas con $k = 20, 50, 100$ y 200. La elección de RandomForest como clasificador se debe a los buenos resultados alcanzables por este, y que su simplicidad hace que ejecute en muy poco tiempo.

Recordamos que, dado n elementos, la cantidad de combinaciones de elegir k de ellos, en cualquier orden, está dada por:

$$C_k^n = \frac{n!}{k!(n-k)!}$$

Con $n = 784$, puede verse que la cantidad de combinaciones posibles son demasiadas para los valores de k sugeridos, del orden de 10^{39} para el caso de $k = 20$, y de 10^{191} para $k = 200$. Lo que vuelve imposible una búsqueda exhaustiva.

Por lo anterior, resulta interesante ver la distribución de los rendimientos alcanzados en estas pruebas, para estimar la probabilidad de que existan combinaciones que alcancen rendimientos tan buenos o mejores como los alcanzados usando el total de características.

En la figura 5 se adjuntan estas distribuciones. En primer lugar, es bueno notar que las cuatro curvas tienen la misma forma, que se asemeja a una distribución normal con una "cola" más alargada para la izquierda.

Luego, vemos que para los casos de 20 y 50 características, es muy poco probable encontrar un rendimiento que ronde el 90 %. En cambio, para $k = 100$ y $k = 200$, no parece tan improbable encontrar combinaciones de características que sí alcancen estos resultados.

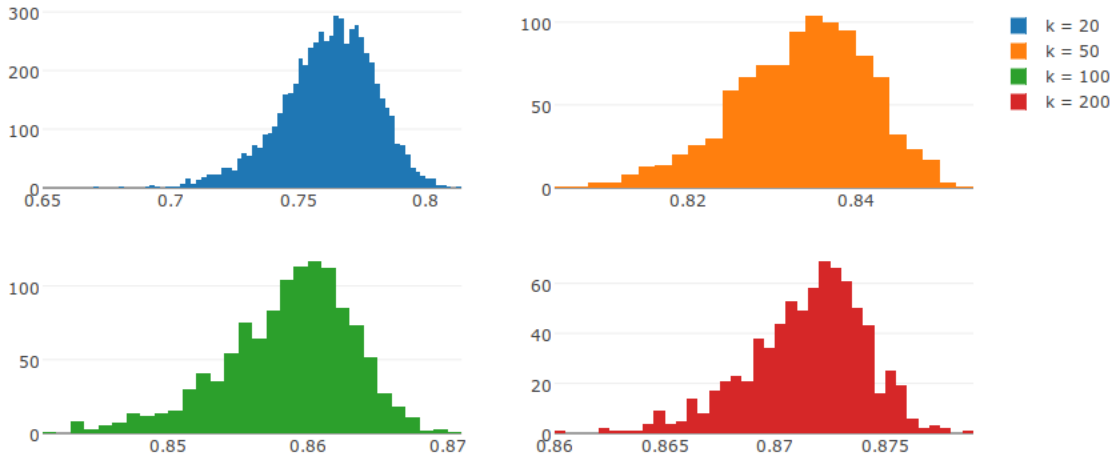


Figura 5: Distribución rendimientos para distintos valores de k.

Por último, encontramos este método muy útil para explorar cómo se distribuyen los rendimientos para distintos valores de k. Y también vemos que por la forma de estas distribuciones, es posible encontrar una combinación de

características que representen el problema, y de esa forma reducir la dimensión del mismo. Y para hacerlo debe implementarse un algoritmo que converja a una buena solución en un tiempo razonable, por ejemplo un algoritmo genético.

4.2. Elección de Clasificadores

En el curso se vio un amplio abanico de Clasificadores, por lo cual el grupo decidió decantar por el uso de Redes Neuronales, RandomForest y Maquinas de Vectores de Estado. Los resultados obtenidos y un breve resumen de cada clasificador es lo que se presenta en ésta sección.

Las pruebas realizadas y los resultados aquí mostrados refieren a usar el total de características, sin métodos de preprocesamiento, y tomando como referencia los resultados alcanzados en [1] y [2].

En la etapa de entrenamiento del clasificador se dividió el conjunto de 60000 muestras en un conjunto de entrenamiento y otro de validación. Pudiendo variar en cada caso cómo se hizo esta división. Si en la descripción de resultados de un clasificador no se especifican estos porcentajes reservados para entrenamiento y validación, es porque se considera que esto no es relevante para los resultados alcanzados.

4.3. RandomForest

Fue el clasificador con el que se hicieron más pruebas debido al bajo tiempo de cómputo que implica, y que sus resultados suelen ser comparables redes neuronales y SVMs, alcanzando los mejores resultados.

Se trabajó con bosques de 100 árboles, variando la profundidad máxima y el porcentaje de datos reservado para training y validación. Se reunen los resultados obtenidos en la figura 6.

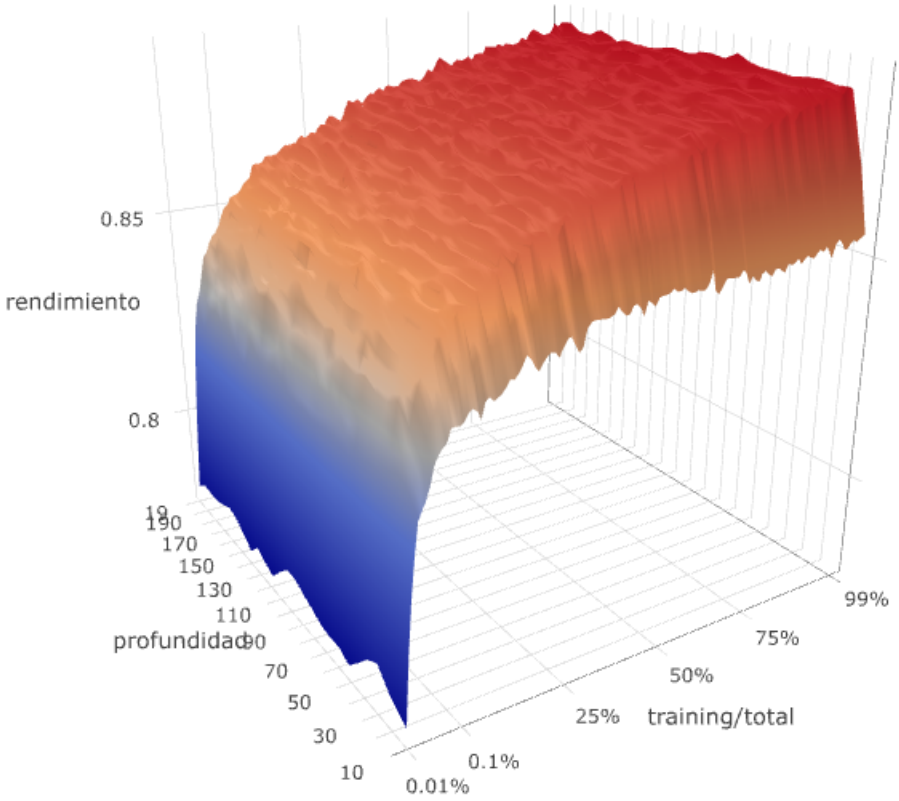


Figura 6: Rendimiento RandomForest para el conjunto test.

Aquí se ve que para profundidades mayores a 30 el rendimiento es casi constante, y para profundidades menores a este valor cae ligeramente. Por otro lado el rendimiento siempre aumenta al aumentar la proporción de datos

reservada para entrenamiento, lo cual sorprende un poco, ya que no se evidencia sobre-entrenamiento. Esto puede deberse a que cada clase tiene una distribución uniforme, y no existen muchos patrones que sean "muy distintos al resto". Por otro lado, en el otro extremo también sorprende la velocidad de aprendizaje del clasificador, alcanzando un rendimiento de 84 % solo con el 0.1 % del total de datos reservado para entrenamiento. Es decir, que alcanza dicho rendimiento ajustando a solamente 6000 muestras, esto es incluso menos de lo que luego usa para clasificar el conjunto test. Ademas, más sorpredente aún, con el 0.01 % (solo 600 patrones) alcanza un rendimiento cercano al 75 %. Esto también podría explicarse si la distribución en cada clase es muy uniforme.

También es bueno ver el resultado logrado para el conjunto de validación. Con el objetivo de ver si estos rendimientos son distintos a los obtenidos para el conjunto test, en la figura 7 se muestra la diferencia de rendimientos en ambos conjuntos.

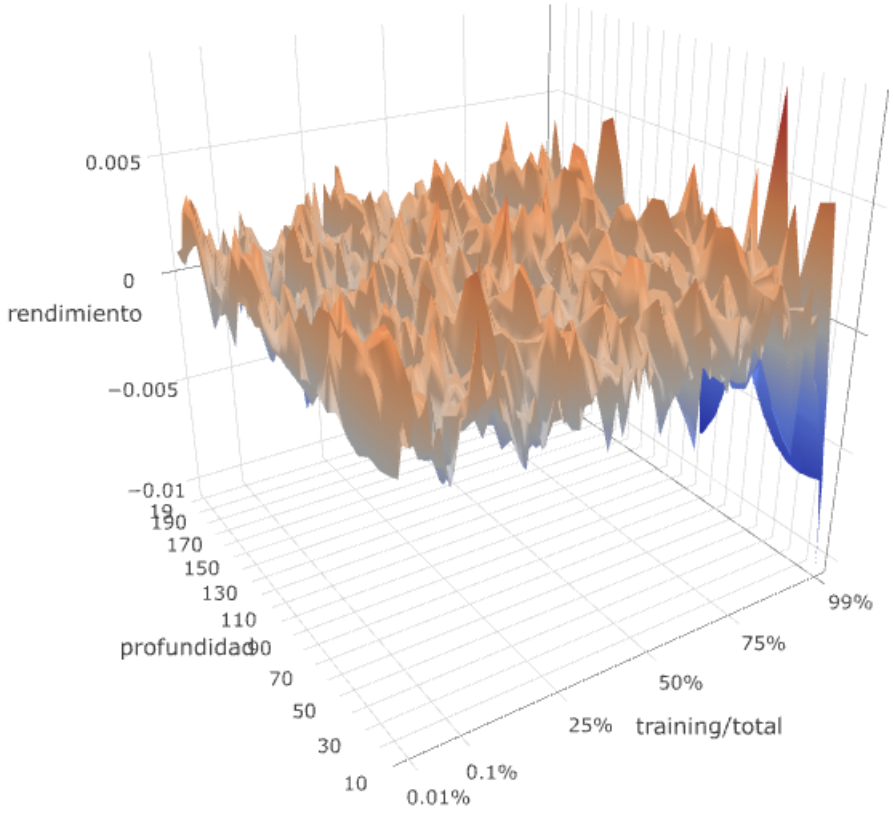


Figura 7: Diferencia de rendimientos para RandomForest entre conjuntos validación y test.

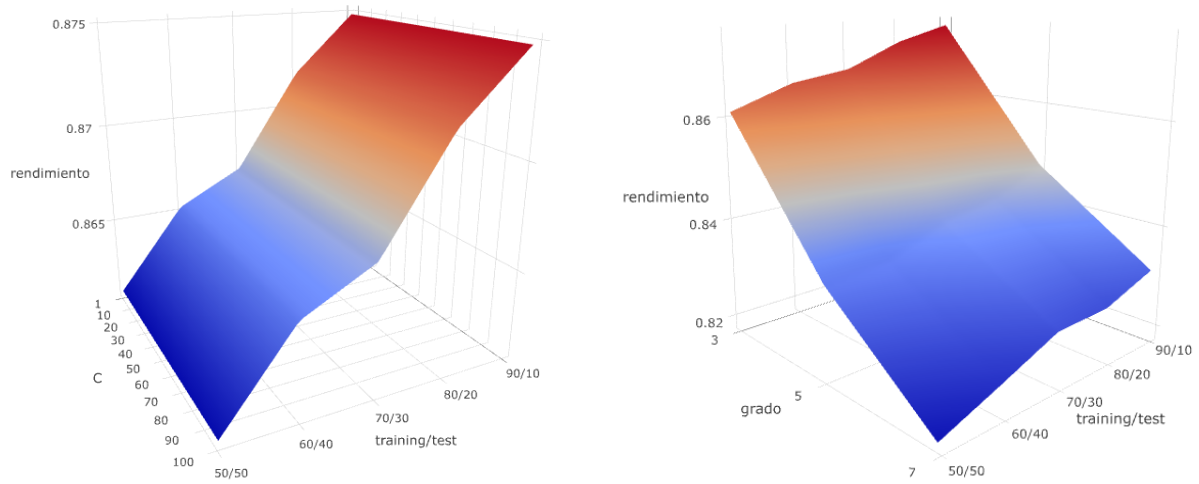
Puede verse que esta es prácticamente nula, lo que refuerza la idea de que la distribución de patrones es bastante uniforme, y es de alguna forma indistinta la cantidad de muestras usadas en cada etapa de clasificación. La mayor diferencia de rendimientos entre conjuntos fue en el extremo de usar más el 99 % de datos para entrenamiento, dejando un conjunto de validación ya muy chico.

Nuevamente estos son resultados obtenidos con el clasificador RandomForest, y sería interesante ver similares para otros clasificadores.

4.3.1. SVM

Se utilizaron SVM con kernel gausianos y polinomial, por tener los mejores rendimientos en los Benchmarks mencionados. Se vió el efecto al variar el valor de C, y para el kernel polinomial variando también el grado del polinomio. También se varió el porcentaje de muestras que componen el conjunto de entrenamiento y el de validación, aunque no de forma tan exahustiva como el caso de RandomForest.

En la figura 8 vemos los rendimientos en función de estos parámetros para un kernel polinomial. En esta, vemos a la izquierda los resultados al dejar fijo el grado del polinomio en 3. Aquí puede verse que el rendimiento es insensible a cambios en C, y que mejora al aumentar el tamaño del conjunto reservado para entrenar. En la derecha, fijamos C, y vemos que el rendimiento mejora al bajar el grado del polinomio, y que para cualquiera de estos valores también mejora al aumentar la proporción de datos reservada para entrenar.



(a) Rendimientos con grado = 3

(b) Rendimientos con C fijo

Figura 8:

Sobre la proporción de datos reservada para entrenamiento y validación no es posible extraer conclusiones como en el caso de RandomForest, ya que el muestreo es muy escaso, y solo se tomaron 5 casos.

La insensibilidad en C puede deberse a que el conjunto de datos usado para entrenar es muy grande, lo que hace que esencialmente puedan despreciarse los efectos de este parámetro.

El mejor resultado obtenido con estas pruebas fue usando un kernel polinomial, de grado 3, con un 87.3 % de rendimiento en el conjunto test.

4.3.2. Redes Neuronales

Las Redes Neuronales utilizadas en este proyectos fueron basadas en capas ocultas de perceptrones, (Multilayer Perceptron Neural Networks). Se realizaron pruebas con un gran número de combinaciones, teniendo como centro una configuración que se encontró como aceptable en la documentación de la base de datos Fashion MNIST. En la misma se encontraba una red de tres capas (256-128-64) con un rendimiento del 90 %.

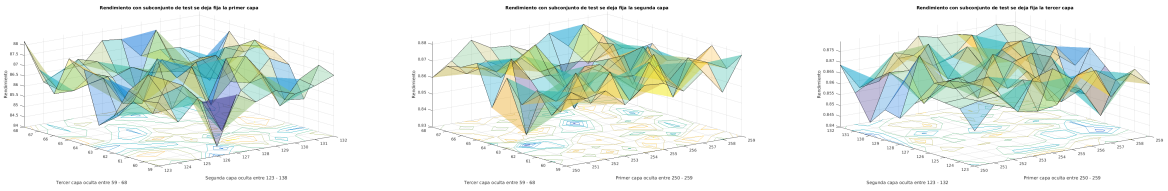
La salida normalizada de las redes neuronales multi capas se puede pensar como una probabilidad. Y en el caso donde las muestras tienden a "infinito" éstas se aproximan en mínimos cuadrados a las probabilidades a posteriori. Con esta base de datos se puede considerar que así es, pues 60000 muestras es un conjunto muy grande.

Para ésta prueba otro de los parámetros que se utilizaron fue Adam como solver ya que está considerado el mejor para bases con gran cantidad de datos, y fue corroborado en pruebas las cuales no forman parte del informe. La librería que brinda scitkitLearn para python utiliza el esquema de aprendizaje estocástico, y la configuración por defecto que se utilizó usa batchs de 200 patrones y "relu" como función de activación definida como $f(x) = \text{máx}(0, x)$

Los resultados obtenidos con ésta grilla que recorre combinaciones cercanas de cantidad de nodos por capa. Lo cual nos levanta la sospecha de haber quedado en un mínimo local

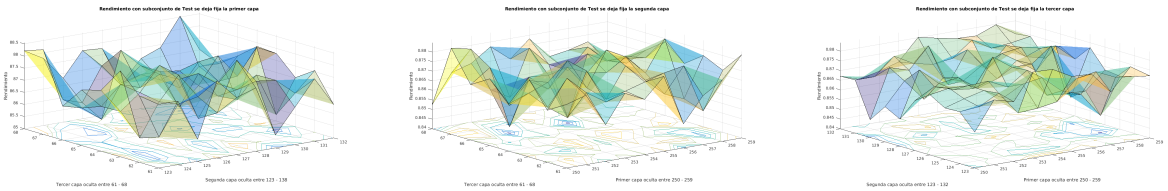
El caso con mejor porcentaje de acierto observando el resultado para el subconjunto de test es para la red con 259-123-68 perceptrones en cada una de las tres capas . Veamos cuales fueron los porcentajes:

Sub conjunto Test	88,5 %
Conjunto de Test	88,48 %
Sub conjunto de Train	94,2 %



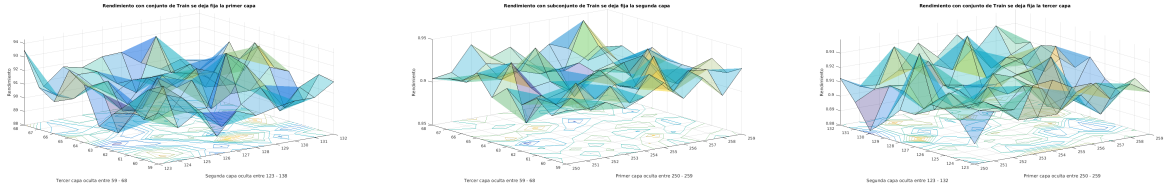
(a) Primer capa fija en 256 (b) Segunda capa fija en 128 (c) Tercera capa fija en 64

Figura 9: Resultados 3 capas sub conjunto de test



(a) Primer capa fija en 256 (b) Segunda capa fija en 128 (c) Tercera capa fija en 64

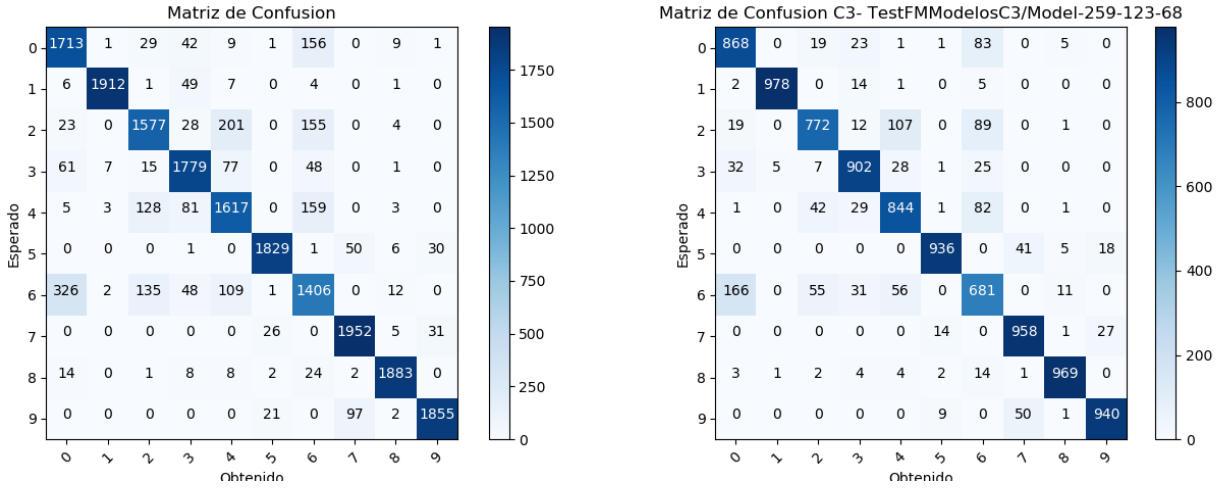
Figura 10: Resultados 3 capas conjunto de test



(a) Primer capa fija en 256 (b) Segunda capa fija en 128 (c) Tercera capa fija en 64

Figura 11: Resultados 3 capas sub conjunto de train

Como se puede ver no pareciera que el clasificador esté sobre entrenado, pues para dos conjuntos totalmente distintos, como los son el subconjunto de test y el conjunto de test, el porcentaje de acierto son muy próximos. Veamos las matrices de confusión para ambos conjuntos:



(a) Subconjunto de Test (b) Conjunto de Test

Figura 12: Matriz de confusión redes neuronales de tres capas 259-123-68

Este clasificador no llegó al porcentaje esperado de 90 % de acierto al contrastar con el conjunto de test. Quedará a futuro analizar los resultados si se varían parámetros tales como la cantidad de épocas (epoch).

En otras pruebas realizadas se probó con el sigmoide como función de activación ya que esta hace una distribución mas suave de los pesos. Pero los resultados fueron magros. El porcentaje de acierto en training no alcanzó el 80 % producto de ello no se cuenta con matrices de confusión ya que el grupo consideró como minimo aceptable para obtenerlas un score superior a 87, 7 % valor.

4.4. Matriz de confusión

Con el fin de estudiar mejor los resultados obtenidos, y ver qué patrones son los más confundidos, vemos ahora la matriz de confusión del problema.

En la figura 13 vemos esta matriz para un caso genérico. Aquí vemos que la prenda que más se confunde en la número 6, que es confundida con las 0, 2 y 4.

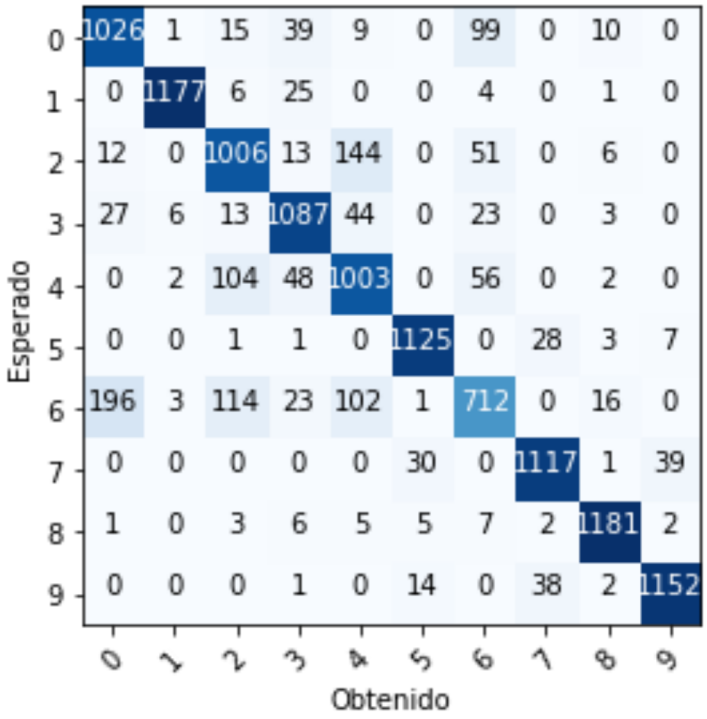


Figura 13: Matriz de confusión para el total de clases.

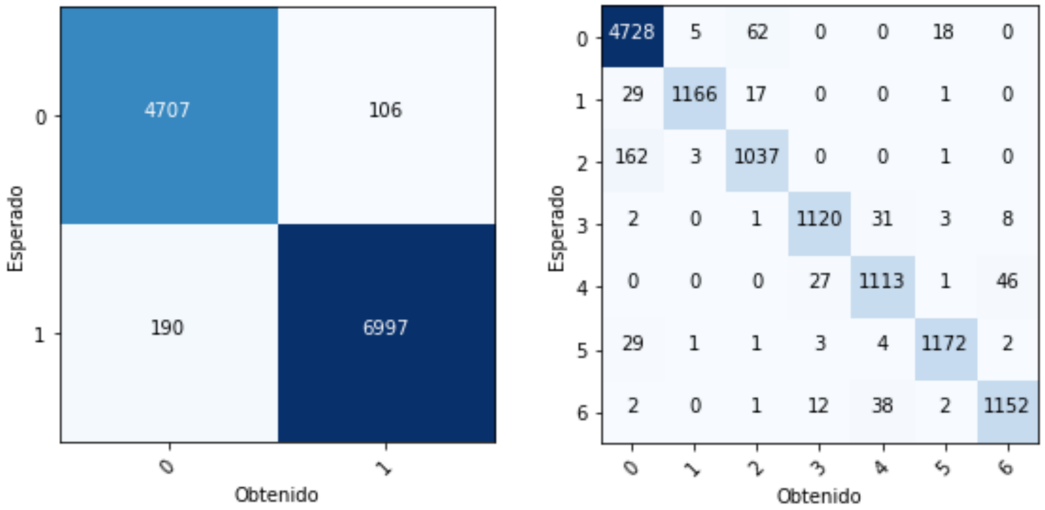
Basándonos en esto, y como forma de mejorar la clasificación, se prueba agrupando las clases 0, 2, 4 y 6 en una sola clase, y luego hacer una clasificación interna en este subgrupo. Esta forma de abordar el problema es debido a que estas clases son muy parecidas entre sí, pero muy distintas al resto. Por lo que entrenar un clasificador solo para estas 4 clases puede dar mejores resultados y capturar mejor las pequeñas diferencias entre prendas para discriminarlas.

Con respecto a las 6 clases restantes, se probó concentrándolas todas en una sola clase para luego hacer una clasificación interna, o dejándolas por separado. En la figura 14 vemos las dos matrices de confusión que resultan de esto, y en la figura 15 las que resultan de la clasificación interna en cada subgrupo de clases.

Si bien, a priori esto pareciera dar mejores resultados y un mayor número de prendas clasificadas correctamente, en la tabla 1 se listan los rendimientos de cada caso, para el conjunto de validación.

Aquí vemos que en realidad no se reportan beneficios reales de esta práctica, sino que se alcanza un rendimiento comparable con la clasificación de las 10 clases de una sola vez. A continuación se tabulan los resultados para el conjunto test, donde, como es de esperarse, se ve un comportamiento similar.

Por otra parte, quedó pendiente estudiar si existen otras formas de agrupar las clases, u otros clasificadores, que aporten mejores resultados.



(a) Caso 4 (1) + 6 (1)

(b) Caso 4 (1) + 6 (6)

Figura 14:

Método	Accuracy
10 clases	88.2 %
4 (1) + 6 (6)	87.9
4 (1) + 6 (1)	87.8

Cuadro 1: Rendimientos para conjunto validación.

Método	Accuracy
10 clases	882 %
4 (1) + 6 (6)	88 %
4 (1) + 6 (1)	87.9 %

Cuadro 2: Rendimientos para conjunto test.

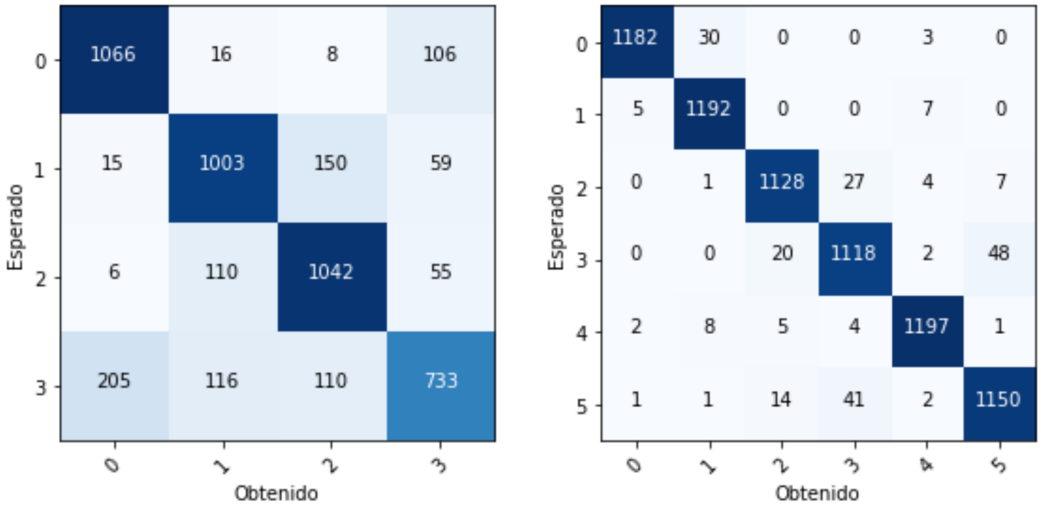
5. Pruebas con Centroides

Como se dijo en la sección de KernelPCA, al intentar procesar KernelPCA se encontró que no era posible con el Hardware disponible por el grupo ya que éste procesamiento requería de mayor cantidad de memoria RAM, en búsqueda de una solución a éste problema fue que se consideró utilizar el Cluster disponible para los usuarios de la Universidad, el mismo fue finalmente una gran herramienta para el proyecto. Lamentablemente con los recursos asignados a la tarea por parte del cluster fue igualmente infructífera la campaña.

Una alternativa fue la de disminuir la cantidad de muestras pero de una forma más eficiente que un sorteo, o al menos que intentase ser más representativa.

Con esta premisa, junto a los docentes del curso, fue que surgió la idea de utilizar los Centroides generados al correr el algoritmo de K-means. En dicho algoritmo los centroides tienden a establecerse en las zonas más densas del conjunto, con lo cual generaría que los centroides fuesen muestras representativas del conjunto.

La cantidad de centroides a obtener se fijó en base a una búsqueda binaria de cuantas muestras eran capaces de procesar las computadores personales de los integrantes del grupo (el resultado es en el entorno de las quince mil muestras). Originalmente se consideró que fueran los quince mil centroides en todo el conjunto de datos, lo cual fue descartado ya que el algoritmo K-Means podría no considerar centroides de alguna clase o que fuera muy dispar la selección. Fue por este motivo que se corrió el algoritmo buscando mil quinientos centroides en cada clase. Surge un nuevo inconveniente, éstos centroides se ven como promedios de las muestras que los circundan, pero no son muestras que realmente pertenezcan al conjunto, lo cual podría generar que un centroide de remeras tuviese más similitud con un pulover (además de las similitudes entra algunas clases ya mencionadas), por lo cual se sumó



(a) Clasificación clases 0,2,4,6.

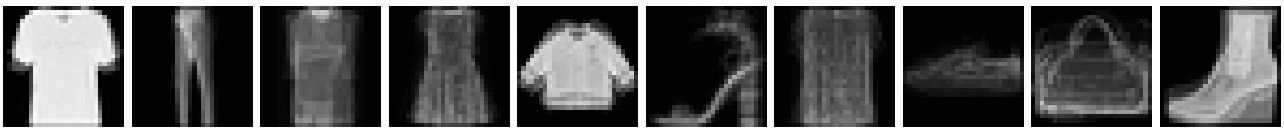
(b) Clasificación clases 1,3,5,7,8,9.

Figura 15:

un nuevo conjunto para pruebas, los centroides del algoritmo K-Medoids (también ejecutado en Matlab). El cual además es más robusto que K-Means. A continuación se describen las pruebas realizadas con ambos conjuntos, sus similitudes y sus diferencias.

5.1. K-Means

A continuación se muestran algunos centroides en el afán de poder apreciar el promedio ya mencionado en las imágenes.



(a) Camiseta (b) Pantalón (c) Pulover (d) Vestido (e) Saco (f) Sandalia (g) Camisa (h) Zapatilla (i) Bolso (j) Bota

Figura 16: Muestras de la base de datos las Categorías

Si bien se encuentran imágenes como las anteriores, en su gran mayoría las 15000 muestras parecieran ser del conjunto original, lo cual se puede apreciar cuando se comparan los resultados obtenidos con el conjunto de obtenido a través de K-Means y el obtenido mediante K-Medoids.

5.2. K-Medoids

En esta base de datos no es necesario presentar imágenes de muestra, ya que son en esencia las mismas que se pueden apreciar en la figura 1

Como se dijo, ésta base de datos es creada en el afán de mejorar los resultados otenidos con el Subconjunto generado mediante K-Means ya que éste tiene muestras que son promedios de las otras.

En las siguientes subsecciones se verán las comparaciones de los resultados.

5.3. kernel-PCA + RandomForest

De correr KernelPCA con el kernel Poly sobre el conjunto, seleccionando las 150 componentes principales que brinda el método, y posteriormente un clasificador RandomForest, se tiene los siguientes resultados para el subconjunto generado con K-Means

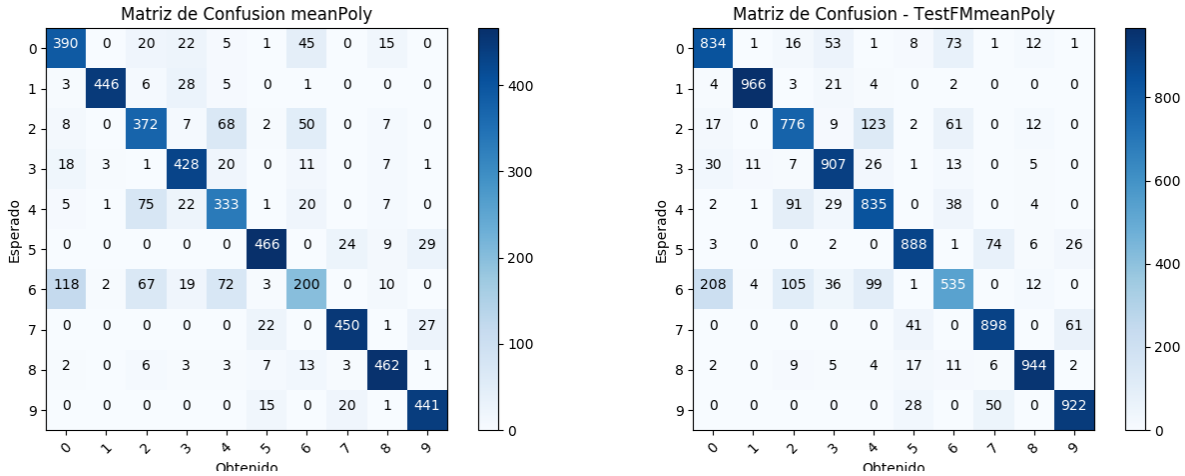
Subconjunto train	100 %
Subconjunto test	80, 6 %
Conjunto test	85,1 %

Cuadro 3: Conjunto generado por K-Means

La elección del kernel viene por los resultados obtenidos con el mismo para SVC.

Como se puede ver los resultados en los tres conjuntos son un poco dispares, sin embargo no parecería estar sobre ajustado, o al menos no mucho, ya que si bien hay diferencia sobre todo con el subconjunto de train, pareciera que para conjuntos externos el resultado es similar.

Observando la matriz de confusión tanto para el conjunto como para el subconjunto de test, se aprecia a simple vista que las clases con las que se confunde el método siguen siendo las mismas, y en una proporción similar en lo que respecta a la mal clasificados en la clase 6.



(a) Subconjunto de test

(b) Conjunto de test

Figura 17: Matrices de confusión kernel=poly $n\_components = 150$ y clasificador RandomForest (Mean)

Veamos ahora que ocurrió con el conjunto seleccionado mediante K-Medoids.

La siguiente tabla de rendimientos y las siguientes matrices de confusión, expresan claramente que el resultado no varía significativamente habiendo utilizado un subconjunto o el otro.

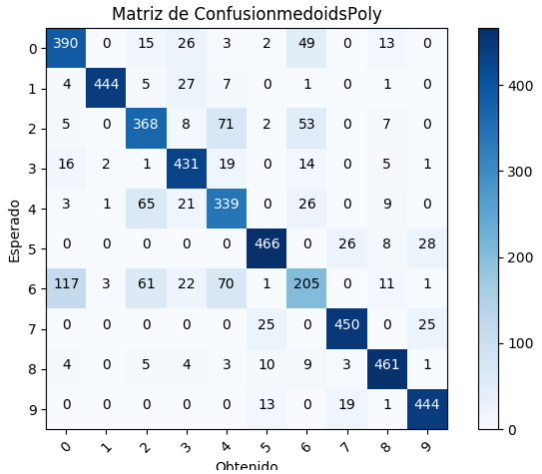
Subconjunto train	100 %
Subconjunto test	80, 8 %
Conjunto test	84,7 %

Cuadro 4: Conjunto generado por K-Medoids

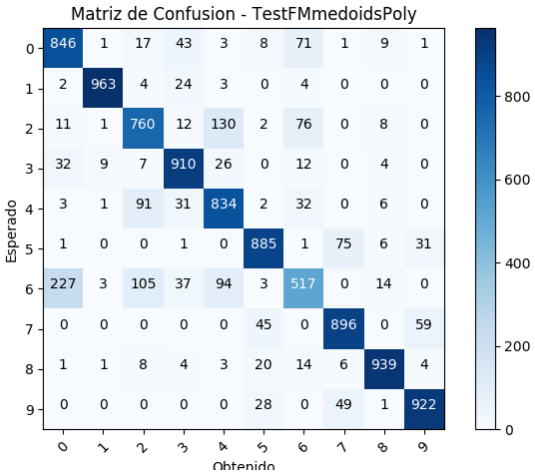
5.4. Redes neuronales

Por lo observado en la sección anterior, para entrenar la red neuronal se utilizó unicamente el conjunto obtenido mediante K-Means obteniendo la siguiente matriz de confusión. Confirmando nuevamente la dificultad para separar los conjuntos 0, 4 y 6. Sólo que en este caso la clase peor clasificada es la 4 en lugar de la 6.

Si bien los resultados no fueron tan buenos como los obtenidos con todo el conjunto, es destacable que con un cuarto de la cantidad de muestras original el porcentaje de aciertos es muy bueno, superior al 80 %. Para otras configuraciones se vio que la clase mas confundida fue la 6.

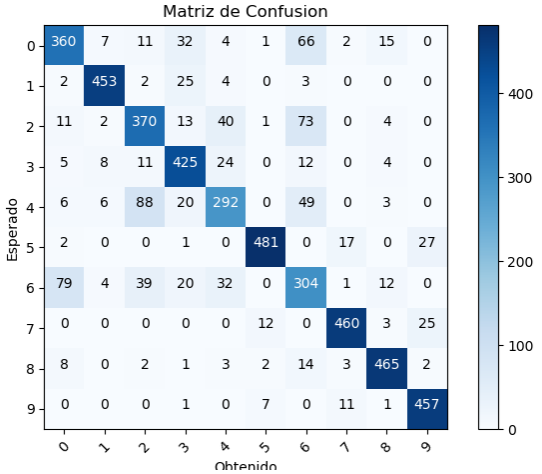


(a) Sub conjunto de test

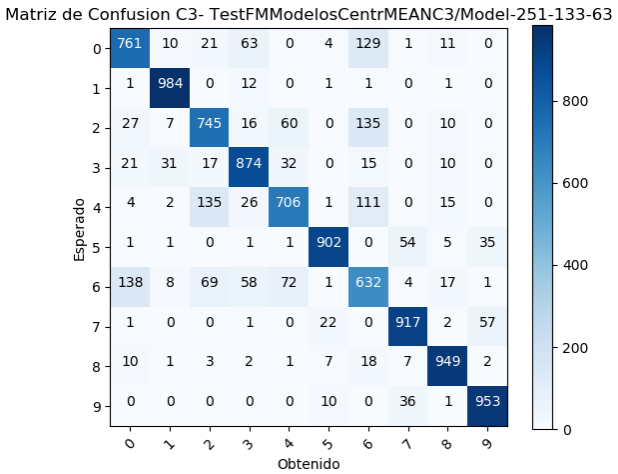


(b) Conjunto de test

Figura 18: Matrices de confusión kernel=poly $n\_components = 150$ y clasificador RandomForest (Medoides)



(a) Sub conjunto de test



(b) Conjunto de test

Figura 19: Matrices de confusión Red neuronal 251-133-63 subconjunt K-Means

6. Conclusiones

El desarrollo del proyecto promovió un buen trabajo en grupo, llevándonos hacia lugares inesperados como lo es haber trabajado con el Cluster de la Universidad, una herramienta tan potente y que se encuentra desconocida por los integrantes de la Universidad, tanto docentes como estudiantes. Nos permitió familiarizarnos con distintas herramientas de programación para Reconocimiento de patrones, experimentar distintas formas de utilizarlas, y conocer en la búsqueda otros métodos por fuera del alcance del curso. Pero sobre todo nos permitió encarar un trabajo de principio a fin donde nos enfrentamos a un problema vanilla y estándar en el mundo del Reconocimiento de patrones, pero que nos abrió la cabeza para enfrentar con las herramientas del curso situaciones inesperadas, lo cual conllevó a mejorar el conocimiento sobre los métodos y ganar intuición sobre los mismos.

Respecto al problema en si mismo, creemos que se mostró que con la capacidad de procesamiento suficiente, o en su defecto el tiempo suficiente, utilizando algoritmos básicos, de complejidad media, se pueden obtener buenos resultados al clasificar esta base de datos.

Cabe destacar que en la base de datos hay imágenes que a simple vista parecieran estar generando ruido en la base, a éstas les dedicamos una sección en el anexo donde se muestran algunos ejemplos.

Lo dicho anteriormente puede ser un colorario del tamaño de la base de datos, al ser ésta tan grande permite explorarla de muchas formas, realizando validación cruzada o simplemente partiendo el conjunto en dos uno de train y otro de test, efectos de ésto se aprecia en el trabajo con los centroides donde se desminuyó considerablemente la cantidad de muestras, y por ende la cantidad de vectores propios de kernel-PCA, y aún así obtener resultados similares a los obtenidos por ejemplo en lo que son las redes neuronales Multilayer utilizando todo el conjunto.

El efecto que produce tener tantos datos también se observan en la poca sensibilidad que tienen los factores de regularización de los métodos como lo es el caso del parámetro C para SVC.

7. Referencias

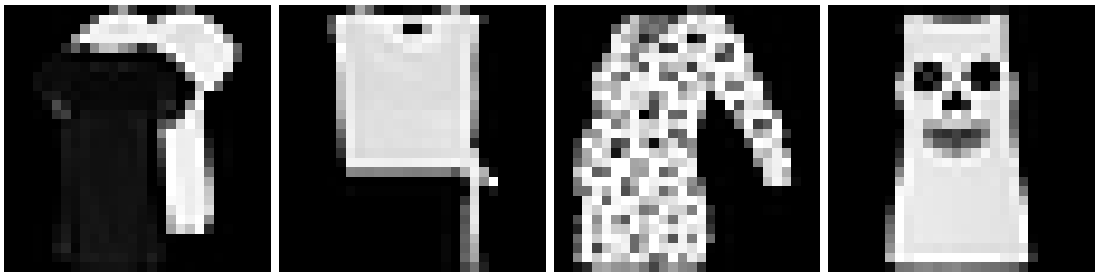
[1] **https://github.com/zalandoresearch/fashion-mnist**

[2] **http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/**.

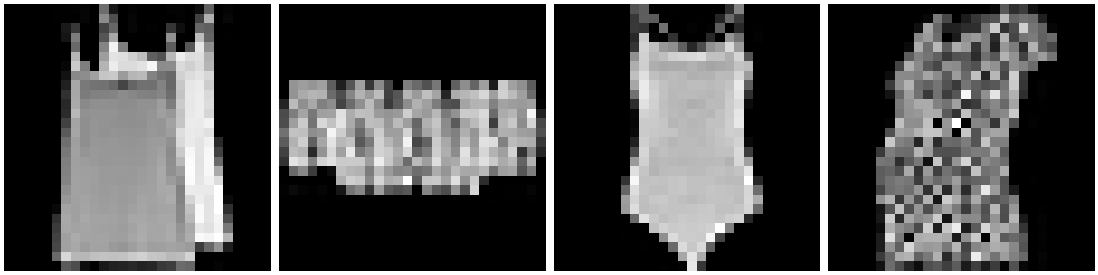
7.1. Anexos

7.2. Muestras ruidosas

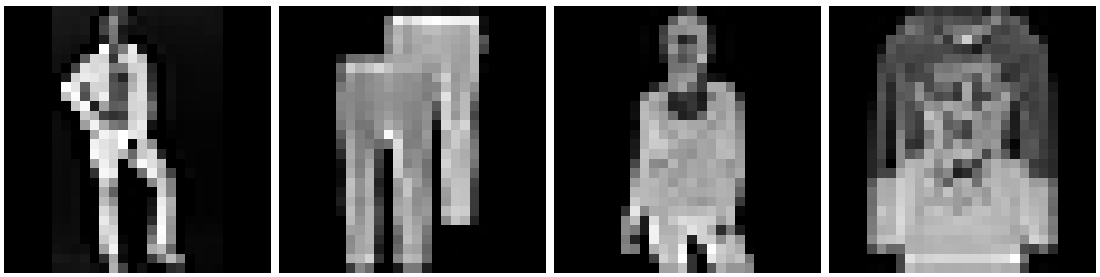
En este anexo se muestran algunas de los patrones que tiene la base de datos que no son exactamente una prenda de ropa. En algunos casos son modelos enteros que tienen más de una prenda, hay imágenes que tienen más de una imagen de la misma prenda, o algunas prendas al menos llamativas



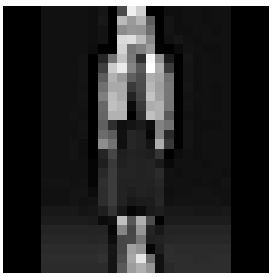
(a) Camisa (b) Camiseta (c) Camiseta (d) Camiseta



(e) Camiseta (f) Camiseta (g) Camiseta (h) Camiseta



(i) Pantalon (j) Pantalon (k) Pulover (l) Pulover



(m) Saco

Figura 20: Muestras que generan ruido