Reconocimiento de Patrones

Proyecto Final

Bruno Benedetti
Francesco Franzoni

3 de diciembre de 2017

Índice

1. Descripción del problema

El objetivo del trabajo es lograr clasificar la base de datos Fashion MNIST. Esta base consiste en 60000 imágenes de 28×28 píxeles en escala de grises (784 características), en las que se representan figuras que corresponden a 10 tipos de prendas de vestir distintas.

1. camiseta
2. pantalón
3. pulóver
4. vestido
5. saco
6. sandalia
7. camisa
8. zapatilla
9. bolso
10. bota

Se trata entonces de un problema de clasificación supervisado, dado que se tiene en la base de datos cada imagen con su etiqueta de la clase a la que corresponde. En la Figura 1 se muestran ejemplos de cada clase.

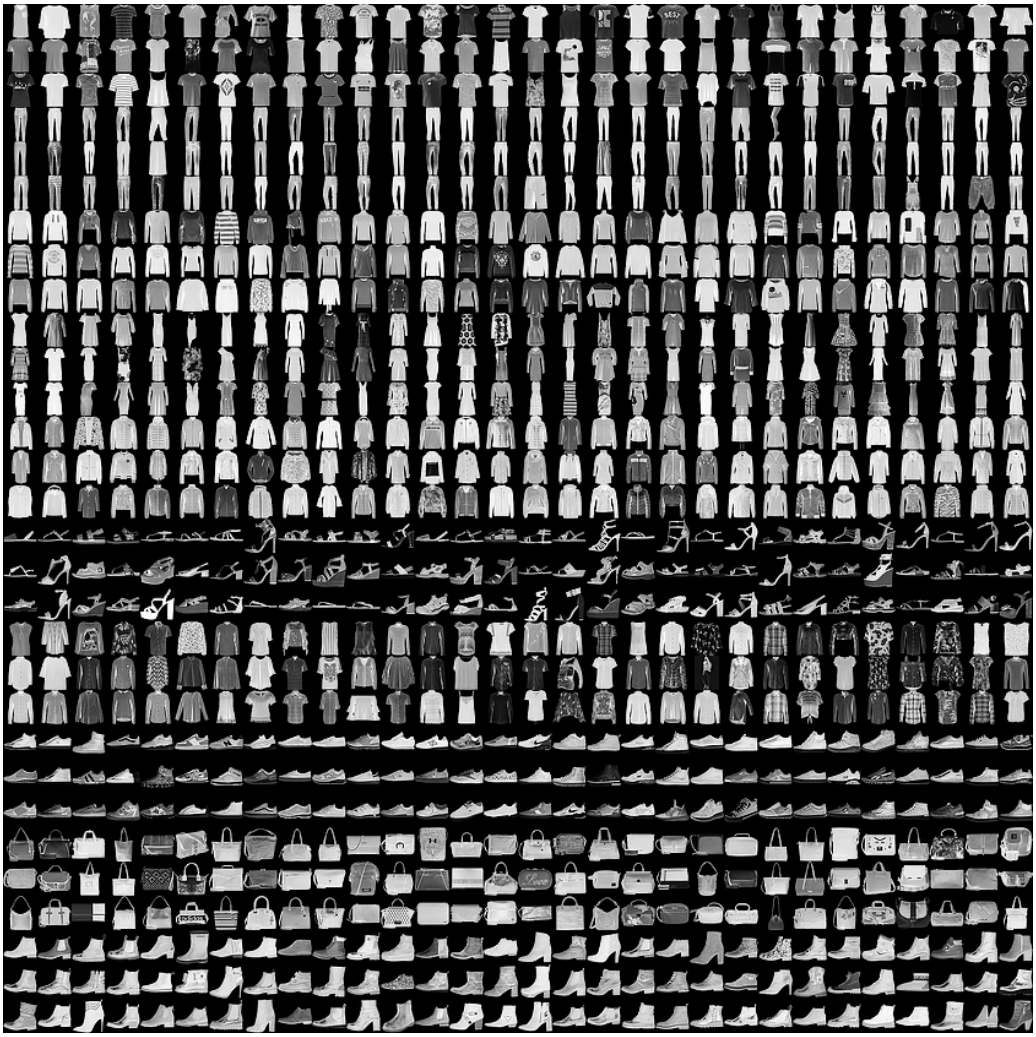


Figura 1: Ejemplos de imágenes de la base de datos Fashion MNIST.

A priori no se tiene información sobre la importancia de cada clase, por lo que se asume que no hay diferencias de costos en errores de clasificación de cada clase. En la práctica podría ser distinto clasificar una camisa como una camiseta, pues la primera es más cara que la segunda, y podría por ejemplo llevar a pérdida de dinero. Dicho esto, en el trabajo se asumirá que las clases tienen todas el mismo peso en la clasificación.

Por otro lado, en los datos disponibles se tiene la misma cantidad de datos en todas las clases, por lo que se trata de un problema balanceado y en el que se asume que las probabilidades a priori de las clases son iguales. Es decir, a la hora de clasificar una nueva muestra, es tan probable que lo que llegue sea una camiseta, como que sea un pantalón.

Como consecuencia de lo anterior, no se le asignará una función de costo o de riesgo al clasificador, mientras que se tomará el accuracy global de todas las clases como medida de éxito.

Es importante mencionar que en este ejercicio en particular se trabaja con dos conjuntos, uno para entrenamiento con 60000 muestras, y otro para test, con 10000 datos.

2. Primeros clasificadores

2.1. Neural Networks

Las redes neuronales consisten en una clasificación en etapas, recibiendo su nombre en analogía con el funcionamiento de las neuronas. Consisten en una capa de entrada con tantos nodos como características se tengan, mientras que la capa de salida presenta la misma cantidad de nodos que de clases posibles.

En lugar de hacer un mappeo directo entre las características y las etiquetas, se utilizan capas intermedias, denominadas capas ocultas, con varios nodos cada una, los cuales aplican una funciónd de activación a una combinación lineal de los nodos de la capa anterior:

$$y_j = f\left(\sum_i w_{ij} x_i + b_j\right) \tag{1}$$

En la Figura 2 se muestra un ejemplo de estructura de una red neuronal.

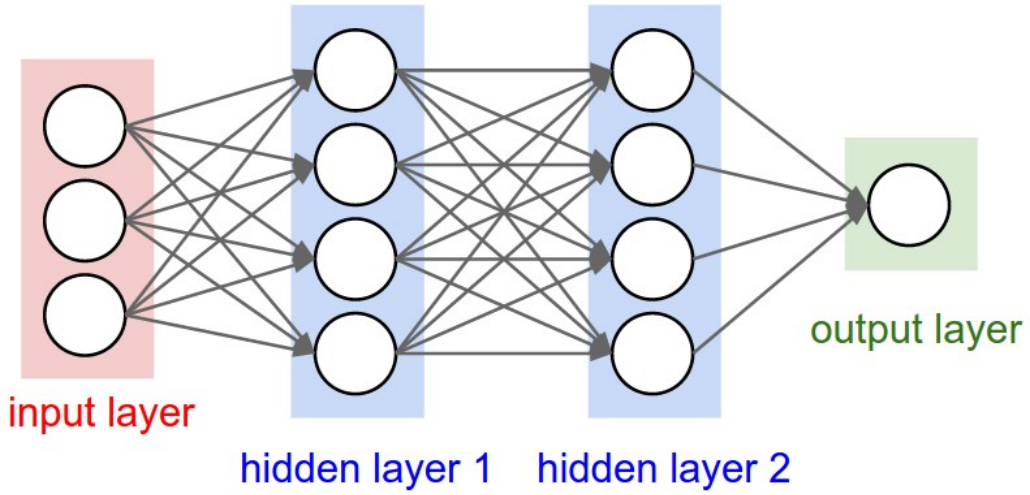


Figura 2: Ejemplo de estructura de una red neuronal.

La ventaja de trabajar con este tipo de clasificador es que se puden utilizar las características iniciales sin hacer un pre-procesamiento de las mismas en un trabajo de selección y/o extracción de features. De hecho, se puede ver la capa oculta como un proceso de extracción de características.

Los datos con los que se realizará las clasificación corresponden a los niveles de gris de las imágenes. De por sí solos cada uno no aporta mucha información en una comparación compleja que permita distinguir entre prendas de vestir, por lo que resulta razonable que sea necesario un pre-procesamiento de los datos. Utilizar como clasificador una red neuronal permite entonces trabajar directamente con los datos de entrada y que en lugar de un pre-procesamiento, se tenga un aprendizaje de las características durante el entrenamiento del clasificador.

Se han obtenido buenos resultados con redes neuronales simples,[1] por lo que este tipo de clasificador resulta una buena opción para probar.

[1] `https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a`

Se elaboró un script en python que utiliza la clase `neural_network.MLPClassifier` de sklearn. Esta clase corresponde a un clasificador de varias capaz de tipo perceptrón (Multi Layer Percepton Classifier), donde cada nodo actúa como un perceptrón con una función de activación, lo que es equivalente a lo que plantea la Ecuación 1.

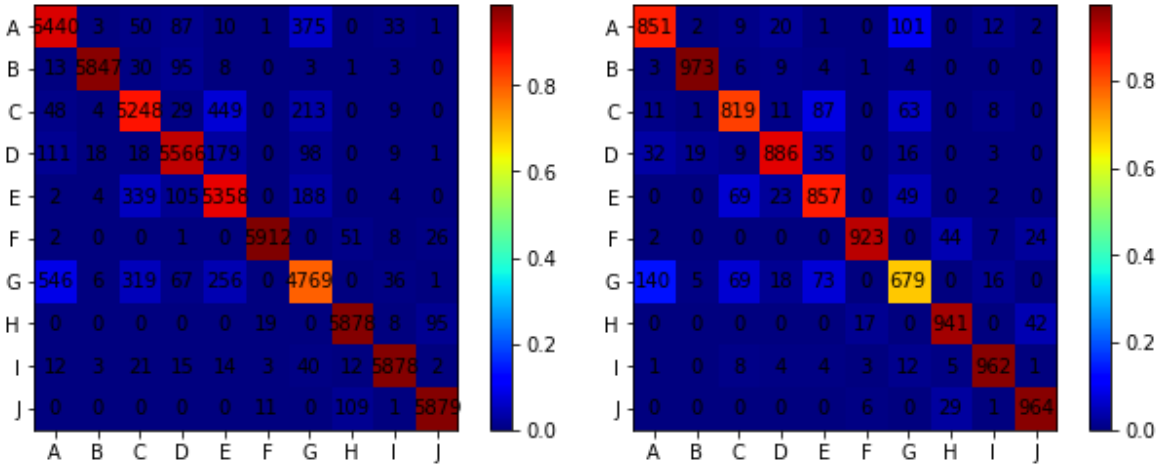
El entrenamiento de los parámetros se realizó a partir de backwards propagation, con el método de stochastic gradient descent (SGD). En cada iteración (epoch) del algoritmo se utiliza un subconjunto de muestras (minibatch) para entrenamiento. La idea consiste en obtener un conjunto de parámetros que minimice una función de costo basada en los errores cometidos. A partir del método de desceneso de gradiente se tiene un algoritmo iterativo que tiende al punto óptimo.

Los principales parámetros del metódo consisten entonces en:

- `hidden_layer_sizes`: cantidades de nodos en cada capa oculta.
- `activation`: función de activación. Se utilizó una función logística.
- `minibatch_size`: cantidad de muestras del mini-batch en cada iteración.
- `learning_rate`: factor de ponderación del gradiente en el aprendizaje. Cuanto más grande más peso se da a la derivada, siendo más *rápido* el aprendizaje. Como contrapartida si es muy alto puede dificultar que el algoritmo llegue al punto óptimo, modificando mucho los parámetros en cada iteración.
- `alpha`: coeficiente de penalización a que los parámetros tomen valores muy altos. Permite reducir la complejidad de la solución obtenida, teniendo como objetivo reducir el overfitting.
- `max_iter`: cantidad máxima de epochs. Evitar overfitting.

Además de los parámetros anteriores, se utilizó una opción de la clase que permite que el *learning rate* sea adaptivo. Esto significa que a partir del valor inicial elegido, si el algoritmo nota que el descenso en la función de costo está por debajo de un cierto umbral, entonces disminuye el *learning rate* a la mitad. Esto permite un aprendizaje más rápido al comienzo del método, pero al momento en que el algoritmo deja de tener un decrecimiento significante en la función de costo, se adapte a variaciones más chicas disminuyendo el coeficiente de aprendizaje.

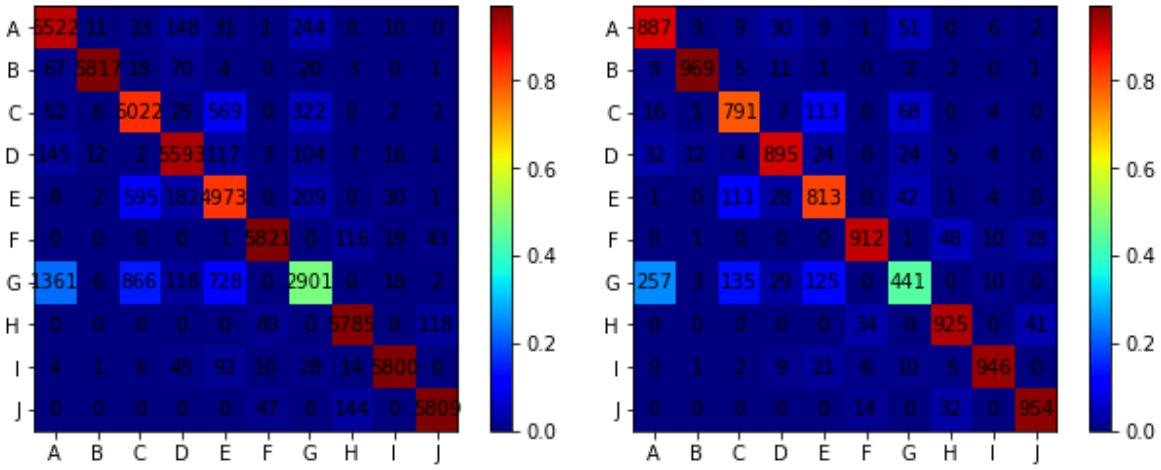
Se probaron distintas combinaciones de los parámetros mencionados anteriormente, para redes neuronales de dos y tres capas. En la Figura 3 se muestran los resultados obtenidos para red neuronal de dos capas de 512 y 128, respectivamente. Se obtuvo un accuracy de 92,96 % sobre el conjunto de train y 88,57 % sobre el de test.



(a) Conjunto de train. (b) Conjunto de test.

Figura 3: Resultados obtenidos con una red neuronal de dos capas. `hidden_layer_sizes=(512,128); activation='logistic'; batch_size=256; learning_rate_init=0.004; alpha=2; max_iter=200`.

Por otro lado, en la Figura 4 se muestran las matrices de confusión para red neuronal de tres capas de 512, 128 y 64, respectivamente. Se obtuvo un accuracy de 88,41 % sobre el conjunto de train y 85,35 % sobre el de test.



(a) Conjunto de train. (b) Conjunto de test.

Figura 4: Resultados obtenidos con una red neuronal de tres capas. `hidden_layer_sizes=(512,128,64); activation='logistic'; batch_size=256; learning_rate_init=0.02; alpha=4; max_iter=400`.

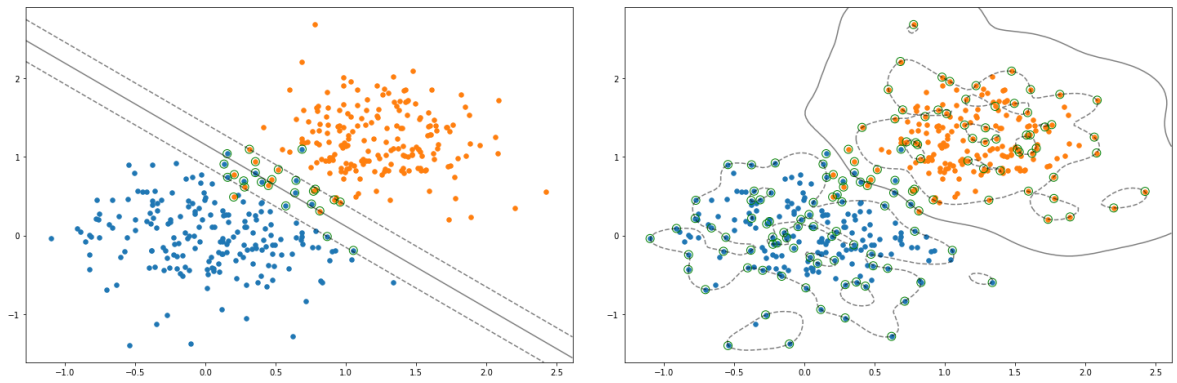
2.2. Support Vector Machines

El algoritmo de Support Vector Machines (SVM) busca una frontera que maximice las distancias entre los puntos de ambas clases. Por lo tanto, tiene una función de costo que considera las distancias entre cada punto y la frontera obtenida. Si bien el método trabaja con fronteras lineales, permite utilizar distintos kernels que transforman un sub-espacio en que los datos no son linealmente separables a uno en que sí lo sean.

En el caso en que no se logre que los datos sean separables, la variante SVM-C permite aplicar el mismo algoritmo incluyendo en la función de costo una penalización a las muestras mal clasificadas. A diferencia de únicamente considerar el margen de separación entre dos clases que no son separables, se define una nueva funciónde costo:

$$\begin{cases} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} & \begin{cases} y_i(\boldsymbol{w}^T\phi(x_i) + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \quad \forall i \end{cases} \end{cases} \tag{2}$$

En la Figura 5 se muestran un par de ejemplos de clasificadores con SVM-C.



(a) Kernel lineal.

(b) Kernel *rbf*.

Figura 5: Ejemplos de clasificadores con SVM-C.

A diferencia de lo descrito para una red neuronal, es necesario un pre-procesamiento de las características iniciales. Para ello se utilizó el algoritmo extracción de características PCA: Primary Component Analysis.

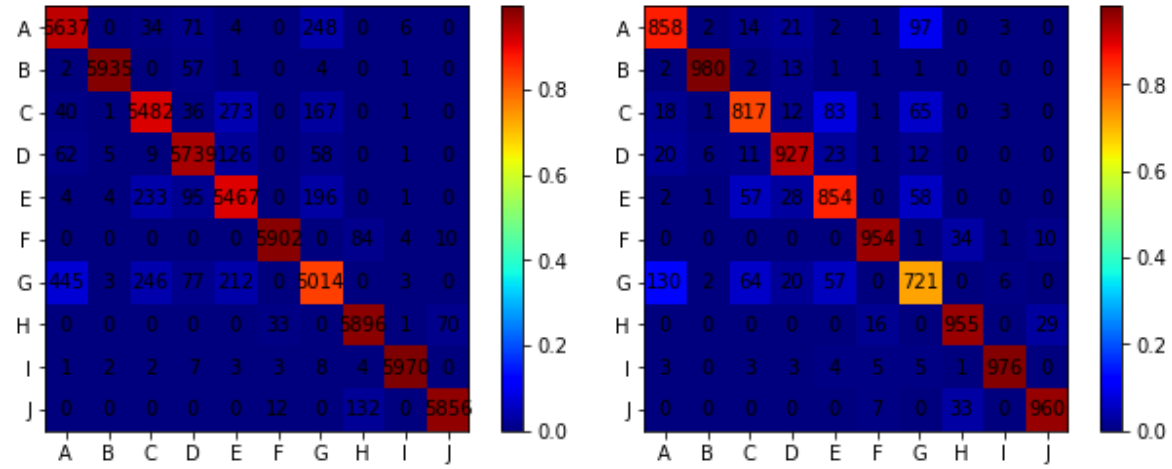
El algortimo PCA funciona a través del cálculo de la varianza de los datos, buscando las combinaciones lineales de los atributos iniciales que llevan a una varianza mayor y por tanto más separación de los datos. De esta manera, se consideran nuevas características a partir de aquellas que llevan a mayor varianza. Es importante notar que se trata de un método no supervisado. Es decir, no considera las etiquetas de las clases al momento del cálculo, por lo que no se tiene ningún *bias* en la elección de las características.

Se elaboró un script en python que utiliza las clases `descomposition.PCA` y `svm` de sklearn para realizar la extracción de características y aprendizaje, respectivamente.

Se probaron distintas combinaciones de los siguientes parámetros:

- `n_components`: cantidad de características a extraer con PCA.
- `C`: peso en la función de costo de errores cometidos en entrenamiento.
- `kernel`: se probó con los kernel `linear`, `rbf` y `sigmoid`.
- `gamma`: parámetro para los kernel `rbf` y `sigmoid`. Cuanto mayor es el valor de gamma más se penaliza la distancia entre las muestras. Esto genera un overfitting en que cada muestra pasa a ser un vector de soporte, por lo que se debe ajustar el parámetro con precaución.

En la Figura 6 se muestran las matrices de confusión obtenidas con este clasificador. Se obtuvo un accuracy de 94,83 % sobre el conjunto de entrenamiento y 90,04 % en el de test.



(a) Conjunto de train.

(b) Conjunto de test.

Figura 6: Resultados obtenidos con SVM-C.
`n_components=50; C=10; gamma=1e-02`.

2.3. Random Forest

La idea escencial del método de clasificación Random Forest(RF) es construir distintos árboles de decision donde cada uno de ellos evalúa la muestra y realiza una prediccion. El resultado final corresponde a la etiqueta que tenga mayor cantidad de predicciones. Dado que los árboles de decisión son sensibles al ruido, este procedimiento ayuda a independizar el resultado del ruido. Cada árbol es construido usando el siguiente algoritmo:

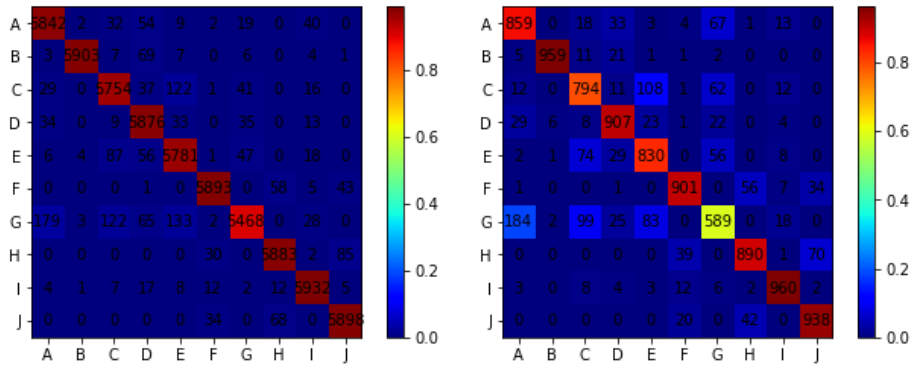
- Sea N el número de casos de prueba, M es el número de variables en el clasificador.
- Sea N el número de casos de prueba, M es el número de variables en el clasificador.
- Sea m el número de variables de entrada a ser usado para determinar la decisión en un nodo dado; m debe ser mucho menor que M.
- Elegir un conjunto de entrenamiento para este árbol y usar el resto de los casos de prueba para estimar el error.
- Para cada nodo del árbol, elegir aleatoriamente m variables en las cuales basar la decisión. Calcular la mejor partición del conjunto de entrenamiento a partir de las m variables.

Al igual que en el caso de SVM, se utilizó PCA para extracción de características.

Random Forest permite controlar el crecimiento de los árboles, y cómo se combinan los mismos. Se utilizó un script de python que utiliza la clase `RandomForestClassifier` de sklearn, probando varias combinaciones de los siguientes parámetros buscando la que diera mejores resultados:

- `n_estimators`: cantidad de árboles de decisión.
- `max_depth`: profunidad máxima de cada árbol entrenado.
- `min_samples_leaf`: cantidad mínima de muestras en una hoja.
- `criterion`: criterio utilizado en la evaluación de cada partición en el árbol.

En la Figura 7 se muestran los resultados obtenidos.



(a) RF sobreentrenado para el conjunto train.

(b) RF aplicado al conjunto test

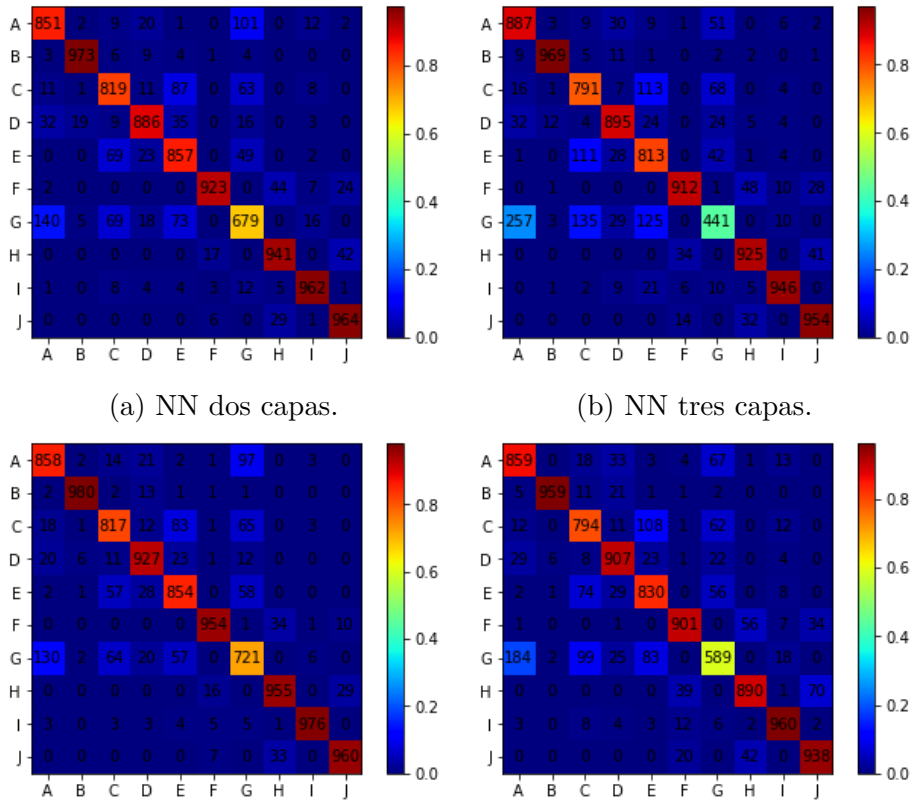
Figura 7: Resultados obtenidos con Random Forest.
`n_components=50; n_estimators=200; max_depth=30; min_samples_leaf=5`.

En la figura 7a se observa como el conjunto queda sobre entrenado, por lo que su desempeño baja considerablemente al evaluar el conjunto de test 7b. Sin embargo, se tiene un resultado global de 86,51 %, el cual es levemente inferior a los obtenidos con SVM-C y NN pero con una clasificación menos costosa computacionalmente.

3. Análisis inicial de resultados

Con los tres métodos probados se obtuvieron buenos resultados, por encima de 87 % con los tres algoritmos. De todas formas, se considera que hay lugar a mejoras en el método y se busca mejorar el nivel de accuracy obtenido.

Al observar las matrices de confusión se puede notar que hay una clase en particular que tiene un porcentaje más bajo de éxito respecto al resto. La misma corresponde a la clase G, de camisas. Además, se puede notar que en los casos de error, se clasifica en general como clases A (camisetas), C (pulóveres) o E (sacos).



(a) NN dos capas.

(b) NN tres capas.

(c) SVM-C.

(d) RF.

Figura 8: Comparación de matrices de confusión obtenidas en test. Se observa en general que la clase G (camisas) es la más problemática.

Con el objetivo de analizar los resultados obtenidos, en la Figura 9 se muestran las imágenes que resultan de tomar para cada píxel el promedio de cada clase.

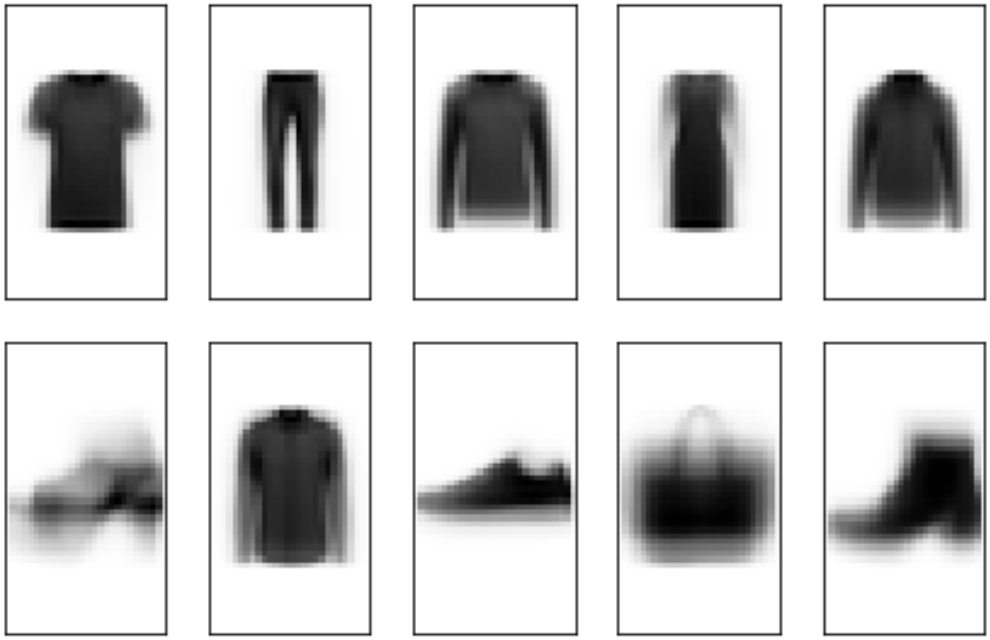


Figura 9: Imágenes promedio de cada clase.

Se puede ver en la Figura anterior que hay una similitud importante entre la clase de camisas y la de camisetas, pulóveres o sacos. Además, se puede observar que un problema particular de esta clase es el largo de las mangas. Se puede ver que en la parte extendida de las mismas en la imagen, los niveles de gris son más difusos, por lo que no se puede asumir que hay proporciones comparables de camisas con y sin mangas. Aquellas que no tienen mangas se pareceran a la clase de A de camisetas, como se ve en la figura promedio, mientras que las de mangas largas se confundirán con las clases C y E.

La Figura 10 muestra para una clasificación de ejemplo (se tomó en este caso la de SVM-C), muestras con las que el clasificador cometió errores. En línea con lo mencionado en el párrafo anterior, algunos casos de camisas de manga corta son clasificados como camisetas, mientras que algunos casos de camisas de manga larga son clasificados como pulóveres o como sacos.

(a) G clasificada como A. (b) G clasificada como C. (c) G clasificada como E.

Figura 10: Errores cometidos para la clase G en el conjunto de test con clasificador SVM-C.

4. Clasificación por votación

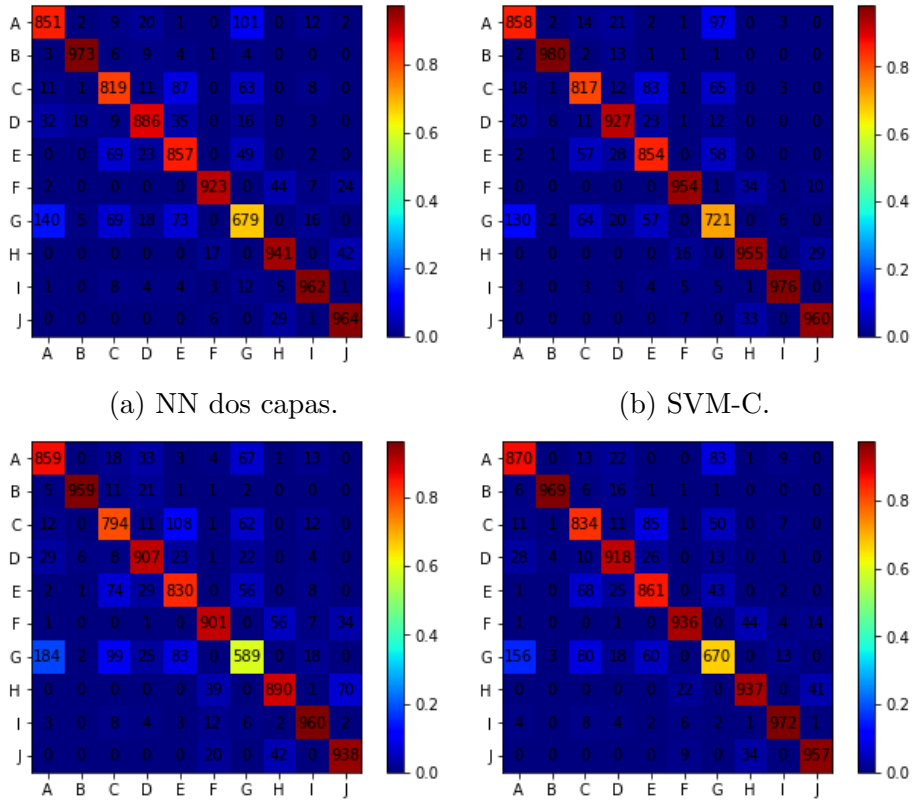
En busca de mejorar la clasificación obtenida individualmente con los métodos descriptos se implementó un sistema de votación por mayoría. Intuitivamente distintos clasificadores se equivocan en muestras distintas, por lo tanto, para mejorar el desempeño del sistema se clasifica una nueva muestra de acuerdo a la moda del resultado de los tres clasificadores operando de forma individual. No se realiza voto ponderado ya que todas las clases tienen el mismo costo, y como se observó en 3 los métodos presentan dificultades en la misma clase.

En este caso, los métodos SVM-C y RF se encuentran entrenados bajo el mismo conjunto de características obtenidas de aplicar PCA al conjunto de entrenamiento. Sin embargo, la red neuronal se encuentra entrenada con las 784 características iniciales.

Los resultados para los conjuntos de train y test son los siguientes:

Método	**Train Acc**	**Test Acc**	**Class G Train Acc**	**Class G Test Acc**
Neural Networks	92,96 %	88,57 %	79,48 %	67,90 %
SVM-C	94,83 %	90,04 %	83,57 %	72,10 %
Random Forest	97, 05 %	86,29 %	91,13 %	58,90 %
Votación	-	89,38 %	-	67,20 %

Tabla 1: Porcentajes de clasificación



(a) NN dos capas.

(b) SVM-C.

(c) Random Forest

(d) Votación

Figura 11: Comparación de matrices de confusión obtenidas donde los parámetros utilizados son los descriptos en la sección de cada método.

Como puede observarse en la tabla 1 y la matriz de confusión de votación 11c, el resultado mejora respecto a NN y Random Forest, sin embargo no logra alcanzar el rendimiento del sistem SVM-C, por lo que la clasificación por votación no es suficiente para mejorar el desempeño de la clase G.

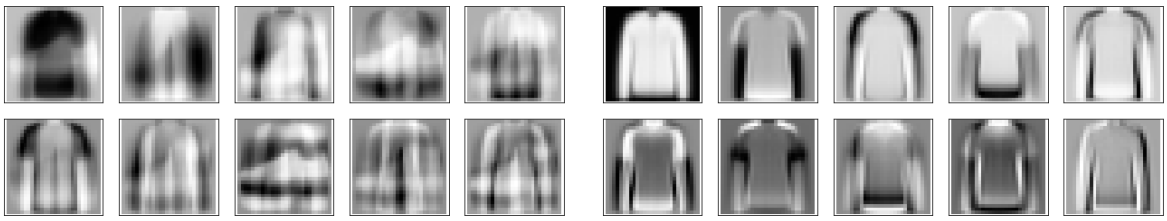
Esto se ve reflejado en la cantidad de votos que recibió la clase G, siendo que de un total de 1000 datos correspondientes a dicha clase, sólo 513 obtuvieron su correcta clasificación por decisión unánime, mientras que los restantes 159 datos correctamente clasificados son resultado de una votación no unánime. Esto implica que hay 328 datos que son mal clasificados por los distintos métodos, y que además coinciden en el error, dando como resultado una votación correspondiente a otra clase.

Este comportamiento va en contra de uno de los pilares de la clasificación por votación que establece que distintos clasificadores se equivocan en distintas muestras, lo cual explica por qué la clasificación por votación no mejoró los resultados obtenidos individualmente y reafirma el parecido entre las clases de camisas, camisetas, pulóver y sacos, reivindicando la dificultad de clasificar este tipo de datos.

5. Clasificación en cadena

Además de la votación, otra opción que se probó fue la de realizar un nuevo clasificador que se encargue de etiquetar las clases A C E y G. Para ello se probaron los mismos tipos de clasificadores, pero utilizando únicamente las muestras de estas clases. La idea está fundamentada en que los algoritmos de redes neuronales o de PCA pueden realizar una búsqueda de características considerando únicamente estas clases en particular, ignorando el resto.

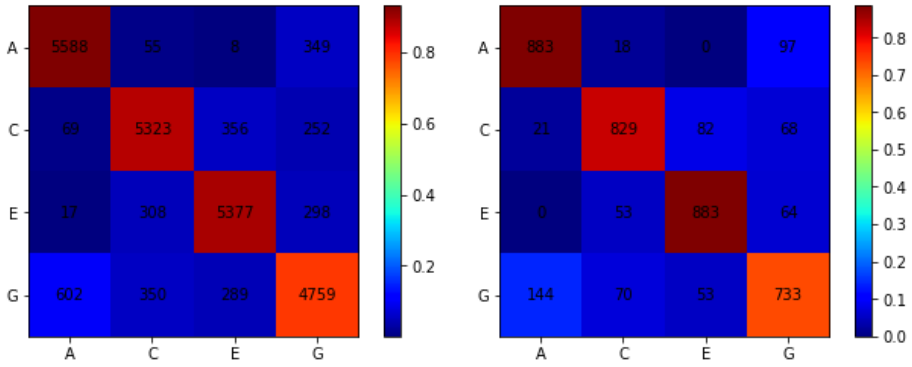
A modo de ejemplo, en la Figura 12 se muestran las diferencias entre aplicar PCA al conjunto completo y aplicar PCA al conjunto formado por las muestras de las clases A C E y G. Se puede observar que en las primeras componentes del conjunto completo aparecen zapatos y pantalones, mientras que en las componentes del conjunto reducido se ven imágenes a nivel de torso. De esta manera al aplicar PCA al nuevo conjunto se pueden encontrar nuevas características, que permitan discriminar mejor las clases de este subconjunto.



(a) Conjunto completo. (b) Conjunto con clases A C E y G.

Figura 12: Primeras 10 componentes resultantes al aplicar PCA.

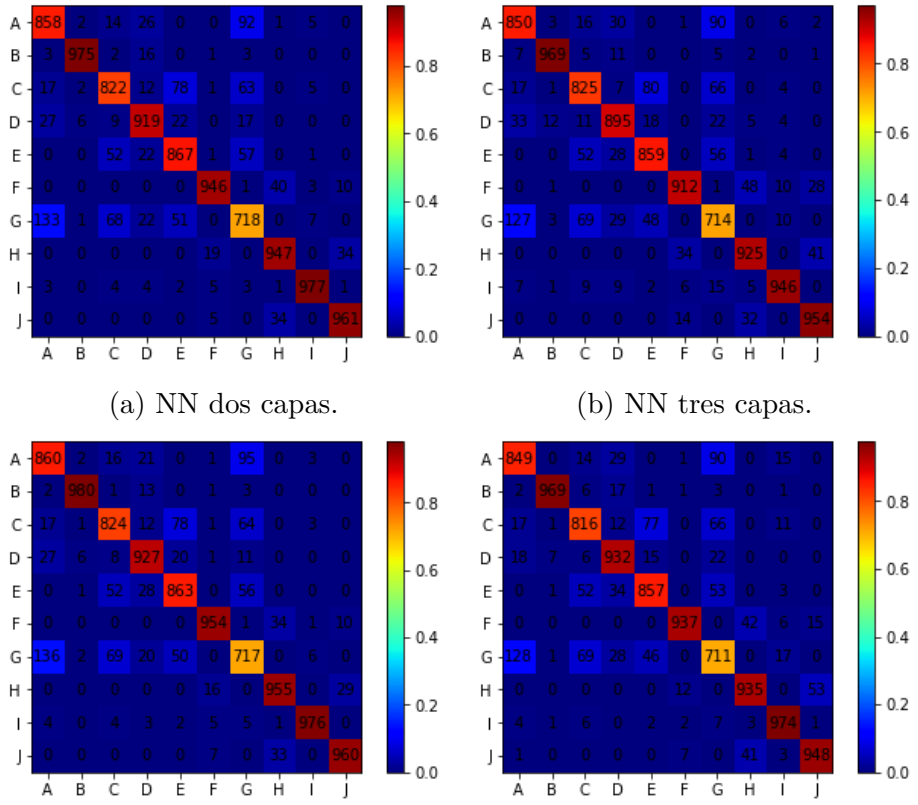
Para el subconjunto formado por las cuatro clases mencionadas se consiguieron mejores resultados con PCA y SVM-C, obteniendo un accuracy de 83,24 % en el conjunto de test.



(a) Conjunto de train. (b) Conjunto de test.

Figura 13: Resultados obtenidos con SVM-C.
`n_components=60; C=100; gamma=1e-03`.

Una vez obtenido un clasificador para estas clases en particular, se realizó un script en python que considere el clasificador inicial junto con el nuevo clasificador, realizando entonces una evaluación en cadena. Aquellas muestras que con el primer clasificador eran etiquetadas como de las clases A C E o G, pasan luego por el segundo clasificador encargado de etiquetar en estas cuatro clases.



(a) NN dos capas.

(b) NN tres capas.

(c) SVM-C.

(d) RF

Figura 14: Resultados obtenidos en el conjunto de test con la clasificación en cadena.

Método	Test Acc. – Solo	Test Acc. – Cadena
NN 2 hidden layers	88,57 %	89,92 %
NN 3 hidden layers	85,35 %	88,51 %
SVM-C	90,04 %	90,18 %
RF	86,76 %	89,30 %

Tabla 2: Porcentajes de clasificación

Se pueden comparar los resultados obtenidos con lo presentado en la Figura 8, observando en general una mejora en los clasificadores.

6. Conclusiones

Se trabajó con el conjunto de datos Fashion MNIST 2.0, y se aplicaron diversos clasificadores estudiados en el curso como lo son las Redes Neuronales, Support Vector Machines y Random Forest.

Al implementarlos se alcanzó un rendimiento del sistema de entre 88 % y 90 % para los tres métodos, logrando resultados similares a los reportados en el benchmark[2].

Se encontró que la principal dificultad en la clasificación es para todos los métodos evaluados la clase G correpondiente a camisas. Revisando el tipo de datos se observa que son muy similares a las camisetas, pulóveres y sacos, clases con la cual los algoritmos confunden a las camisas.

Para abordar este problema se implementaron distintas técnicas. En primer lugar se desarrolló un algoritmo de votación por mayoría entre la clasificación de los tres métodos utilizados, sin embargo los resultados obtenidos presentan una leve mejoría para dicha clase y muestran que los algoritmos cometen prácticamente los mismos errores, reafirmando la dificultad de clasificación de las camisas.

Luego se entrenó utilizando solo datos correspondientes a las clases de camisas, camisetas, pulóveres y sacos, en búsqueda de obtener modelos que ajusten mejor a dichas clases y así poder implementar un sistema de clasificación en cascada. Con este método se obtuvieron leves mejoras.

Durante el trabajo se consideró el caso en que todas las clases tenían el mismo costo de clasificación. Como trabajo a futuro se puede considerar distintos pesos sobre cada clase, por ejemplo evaluando el costo promedio en dinero de cada prenda de vestir, y realizar un entrenamiento que tenga en cuenta dicha ponderación.

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