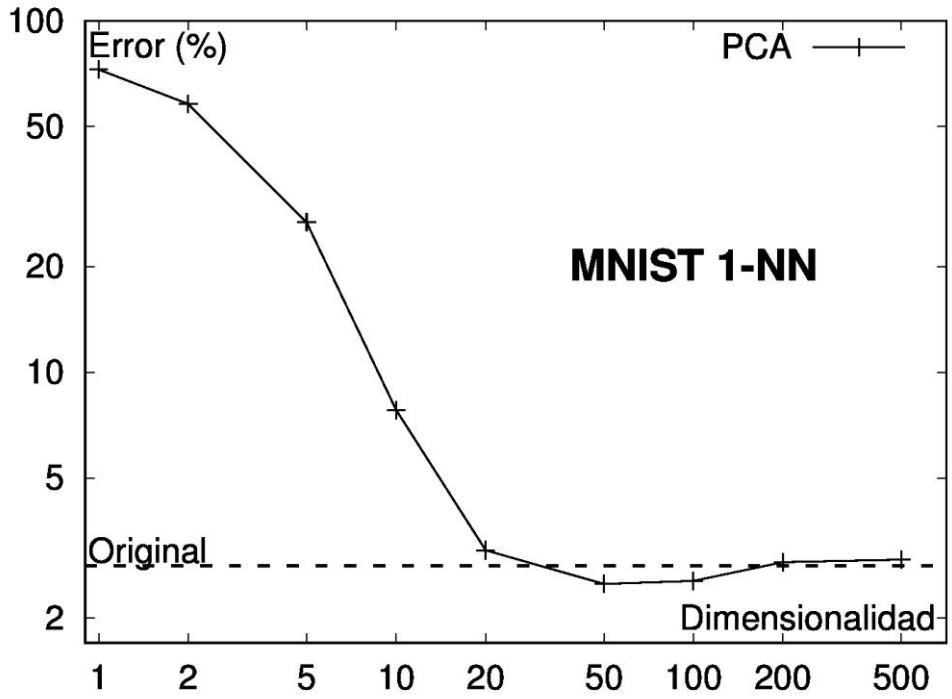
Memoria Entrega 1 (Opcional) Proyecto PER

José Miguel Acosta Triana



Esta es la gráfica que he obtenido que relaciona el número de dimensiones proyectadas con el porcentaje de error al clasificar del conjunto de entrenamiento de MNIST a través de PCA, aplicando el algoritmo de edición de prototipos de Wilson y luego aplicando 1-NN para clasificar.

Al igual que en el caso base sin usar el algoritmo Wilson, se puede observar que cuando se tienen pocas dimensiones es difícil clasificar los datos ya que perdemos dimensiones con una varianza significativa. Pero a medida que añadimos dimensiones se hace más sencillo.

Tras haber probado anteriormente con el conjunto de entrenamiento, decidí que 50 dimensiones era lo más óptimo dentro de lo que pude probar. Así que lo apliqué al conjunto de entrenamiento de MNIST y luego comprobé con el conjunto de test. El resultado al aplicar PCA a 50 dimensiones, luego Wilson con 1-NN y luego vecinos más cercanos con k = 1 fue de una tasa de error del 2,65% al clasificar las muestras, esta tasa de error es inferior a la que obtuve aplicando PCA a 89 dimensiones, que eran las más óptimas, ya que había una tasa de error del 3,09%. Por lo tanto, podemos observar que al aplicar el algoritmo de Wilson podemos reducir la tasa de error al hacer que las fronteras de decisión sean simplemente conexas y quitar confusión al clasificar. Si comparamos con los resultados de la web de MNIST, observamos que esa tasa de fallo de 2,65% es menor que cualquier tasa presente en la web para un clasificador k-NN con distancia euclídea al que no se le haya aplicado ningún preprocesado.

A la hora de implementar el algoritmo de Wilson en GNU Octave he seguido bastante de cerca las recomendaciones hechas en el boletín del proyecto de prácticas, he implementado los métodos auxiliares nombrados en el boletín también.

```
function [ind] = Wilson (X,xl,k)
ind = [1:rows(X)]; #Vector horizontal de índices
N = rows(X); #nº de datos
V = mnn(X,xl,100);
error = true;
while (error)
  error = false;
  for i=1:N
    if(ismember(i,ind))
      c = knnV(V(:,i),ind,xl,k);
      if(c != xl(i,1))
        ind = setdiff(ind,i);
        error = true;
      endif
    endif
  endfor
endwhile
endfunction
```

Esta función recibe las muestras, etiquetas de clase y k a aplicar para k-NN; tras ello, crea un índice con las muestras presentes y llama a *mnn*, que es una función que proporciona un vector columna con los 100 vecinos más cercanos para cada muestra, luego tenemos un bucle que nos permite ver si se clasifican correctamente las muestras, pero solo se ejecuta la parte de dentro si la muestra sigue perteneciendo al índice, si es así, con el método auxiliar *knnV* comprobamos los k vecinos más cercanos de la muestra y clasificamos, si esta clase no coincide con la etiqueta de clase de la muestra, la quitamos del vector de índices y seguimos clasificando muestras. El algoritmo acabará cuando ya no se produzcan errores al clasificar las muestras.