Aprendizaje Automático
Trabajo final

Guillermo Gómez Trenado

June 4, 2018

Contents

1 Clasificación de dígitos manuscritos

Nos enfrentamos al problema de clasificación de dígitos manuscritos con tres modelos, uno lineal y dos no lineales. He separado el trabajo en distintos apartados que parcelan cada una de las etapas del análisis y la construcción del modelo.

1.1 Observación y comprensión de los datos

Vamos a cargar los datos para poder analizarlos respetando en todo momento el conjunto de test para no contaminar la evaluación.

```
dtrain = read.csv("data/pendigits.tra",header = FALSE)
dtest = read.csv("data/pendigits.tes",header = FALSE)
dtrain[,ncol(dtrain)]= factor(dtrain[,ncol(dtrain)])
dtest[,ncol(dtest)] = factor(dtest[,ncol(dtest)])
dtrainX = dtrain[,1:(ncol(dtrain)-1)]
dtrainY = dtrain[,ncol(dtrain)]
dtestX = dtest[,1:(ncol(dtest)-1)]
dtestY = dtest[,ncol(dtest)]
paste("[Train] Número de muestras: ",nrow(dtrain))
```

```
## [1] "[Train] Número de muestras:  7494"
```

```
paste("[Test] Número de muestras: ",nrow(dtest))
```

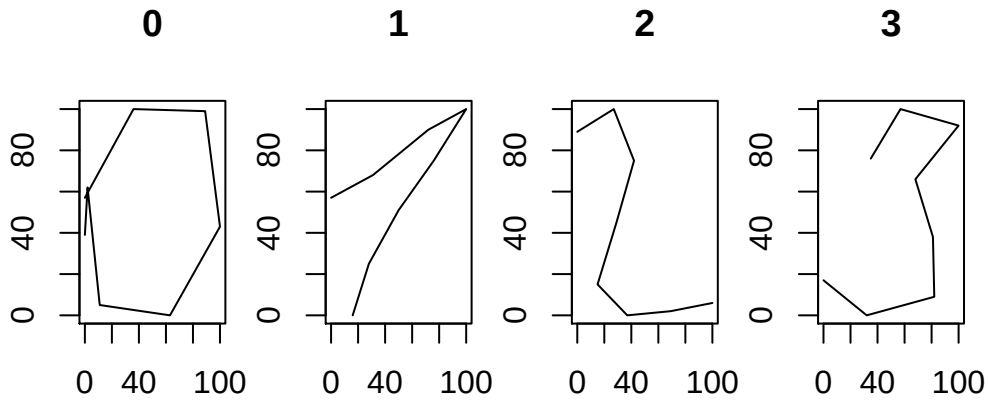
```
## [1] "[Test] Número de muestras:  3498"
```

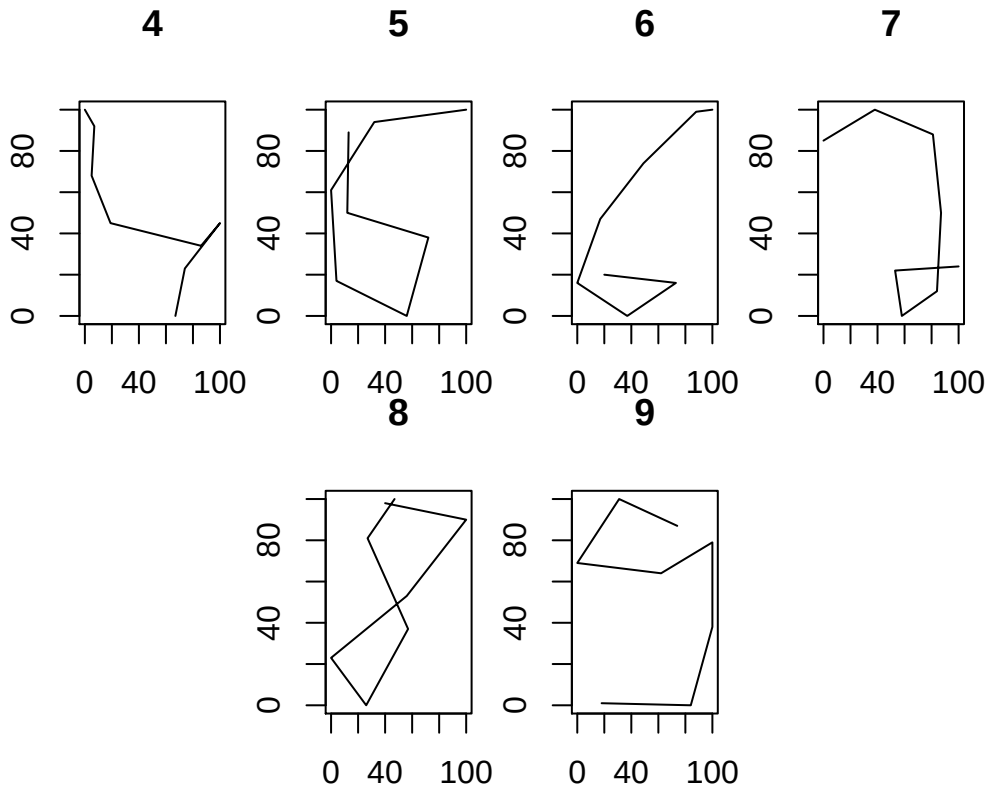
Tenemos ante nosotros 10992 muestras de dígitos manuscritos representados como una sucesión de 8 coordenadas cartesianas enteras en un plano acotado entre 0 y 100 en ambos ejes. Los valores se han normalizado entre 0 y 100 estirando los valores vertical y horizontalmente, por lo general sólo horizontalmente pues los números suelen ser más altos que anchos.

La muestra está ya separada en dos conjuntos, uno de entrenamiento y otro de test, lo interesante de esta separación es que encontramos 250 muestras por cada escritor, 44 escritores en el conjunto de entrenamiento y 14 distintos para el test. Lo que estamos evaluando en definitiva es si podemos extraer características suficientes de un conjunto reducido de escritores para predecir los dígitos de escritores no conocidos.

Vamos a visualizar alguno de las muestras de entrenamiento para hacernos una mejor idea.

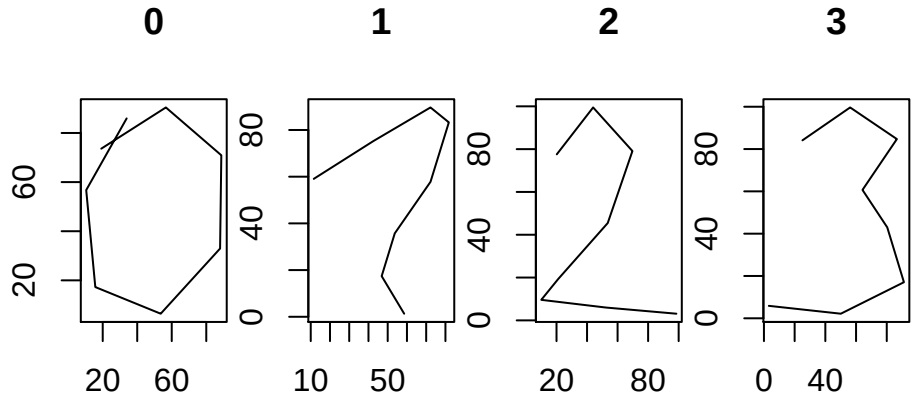
```
paintNumber = function(r,c){
  xp = unlist(r[seq(1,length.out = 8, by = 2)])
  yp = unlist(r[seq(2,length.out = 8, by = 2)])
  plot(x = xp,y= yp,type="l",main = c, xlab = "", ylab = "")
}

for(i in 0:9){
    paintNumber(dtrainX[dtrainY == i,][1,],i)
}
```

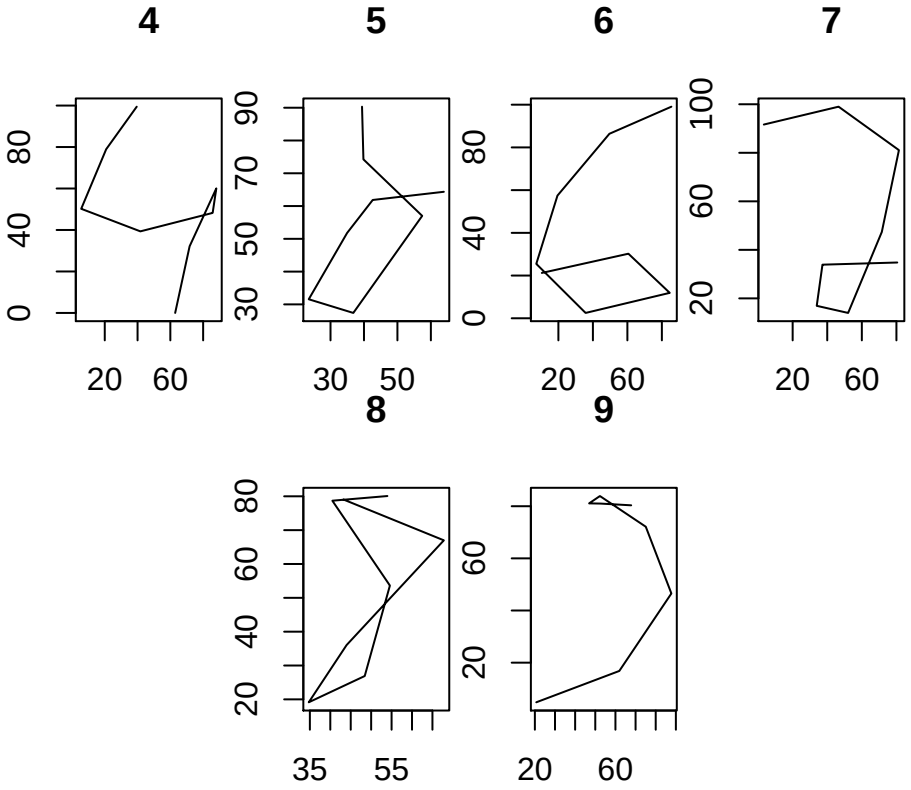




Y los valores medios

```
for(i in 0:9){
    paintNumber(apply(dtrainX[dtrainY == i,],c(2),mean),i)
}
```





Se aprecia de forma inmediata cómo parece que hay clases que serán más fáciles de clasificar por métodos lineales como el 2 o el 4, donde el valor medio conserva características representativas de la clase mientras que otros seguramente necesitarán modelos no lineales que generen abstracciones sobre la relación entre distintas características y puedan separar el espacio de decisión en fronteras no lineales.

1.2 Preprocesado de los datos

La única transformación que vamos a aplicar a los datos, pues la reducción de características a priori no parece necesaria —más aún con sólo 16 características— es normalizar los valores entre -1 y 1.

```
ndtrainX = apply(dtrainX, c(2), function(c){c/50-1})
ndtestX = apply(dtestX, c(2), function(c){c/50-1})
```

1.3 Definición de conjuntos de training, validación y test

En este problema el conjunto de train y test está ya acertadamente separado en dos conjuntos distintos primando la independencia entre los dos conjuntos con número equilibrado de representación de cada clase en cada subconjunto, unas 750 muestras por clase en el conjunto de entrenamiento y unas 350 en el test.

Para la validación vamos a utilizar validación cruzada 5-fold que nos permitirá por un lado el cálculo de los hiperparámetros de cada uno de los modelos elegidos y por otro lado comparar los métodos entre ellos en base a una medida de error que será el número de clasificaciones incorrectas.

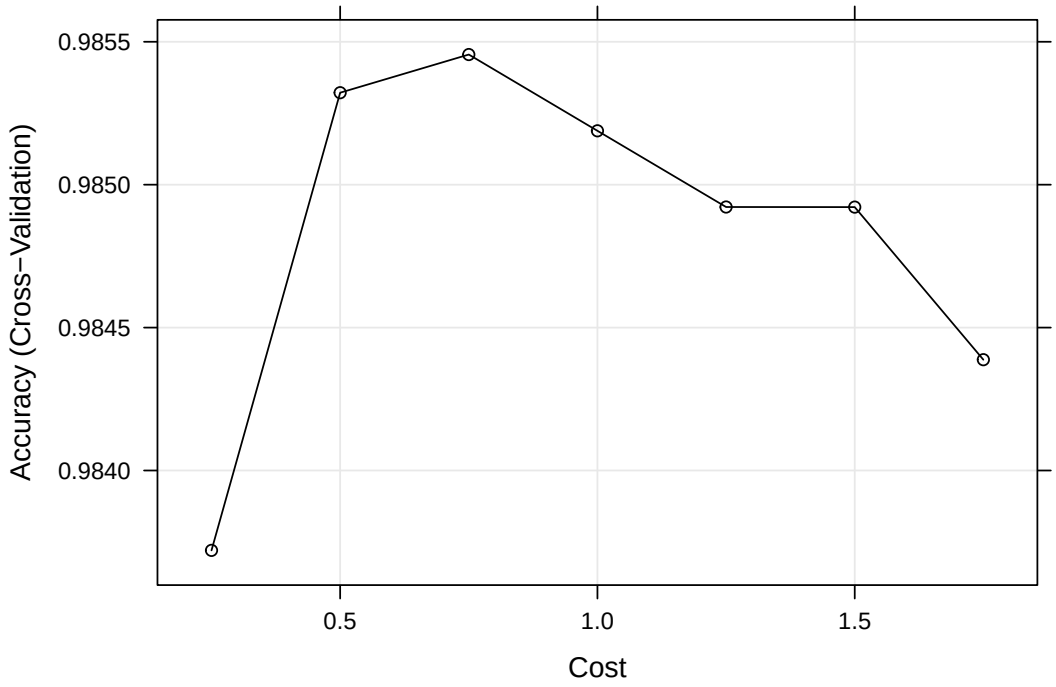
1.4 Definición de modelos a usar y estimación de hiperparámetros

Vamos a usar tres modelos distintos, un modelo lineal que será SVM con un kernel lineal, y dos modelos no lineales, SVM con kernel RBF y una red neuronal. Se trata de SVM con soft-margin así que tenemos que definir el parámetro de coste de clasificación incorrecta, resultando esta decisión en el equilibrio entre máximo margen y clasificiaciones incorrectas.

```
tC = trainControl(method="cv",number=5)
```

1.4.1 SVM con kernel lineal

```
svmLGrid = expand.grid(C=seq(0.25,1.75,by = 0.25))
svmLRes = train(x=ndtrainX,y=dtrainY,method="svmLinear",trControl = tC,
                tuneGrid = svmLGrid)

gtThan = svmLRes$results$Accuracy > 0.5

plot(svmLRes)
```



```
paste("Mejor coste: ", svmLRes$bestTune)
```

```
## [1] "Mejor coste:  0.75"
```

```
paste("Precisión: ", svmLRes$results[svmLRes$results$C ==
                                       svmLRes$bestTune[[1]],]$Accuracy)
```

```
## [1] "Precisión:  0.985455276073383"
```

```
svmLTune = svmLRes$bestTune
```

Por lo que he podido comprobar el mejor coste depende altamente de la aleatoriedad de la selección de los subconjuntos para **5-fold CV** y cualquier valor entre 1 y 1.5 da los mejores resultados, para el kernel RBF fijaremos el coste a 1 para poder invertir la capacidad de cómputo en el cálculo del sigma más ajustado.

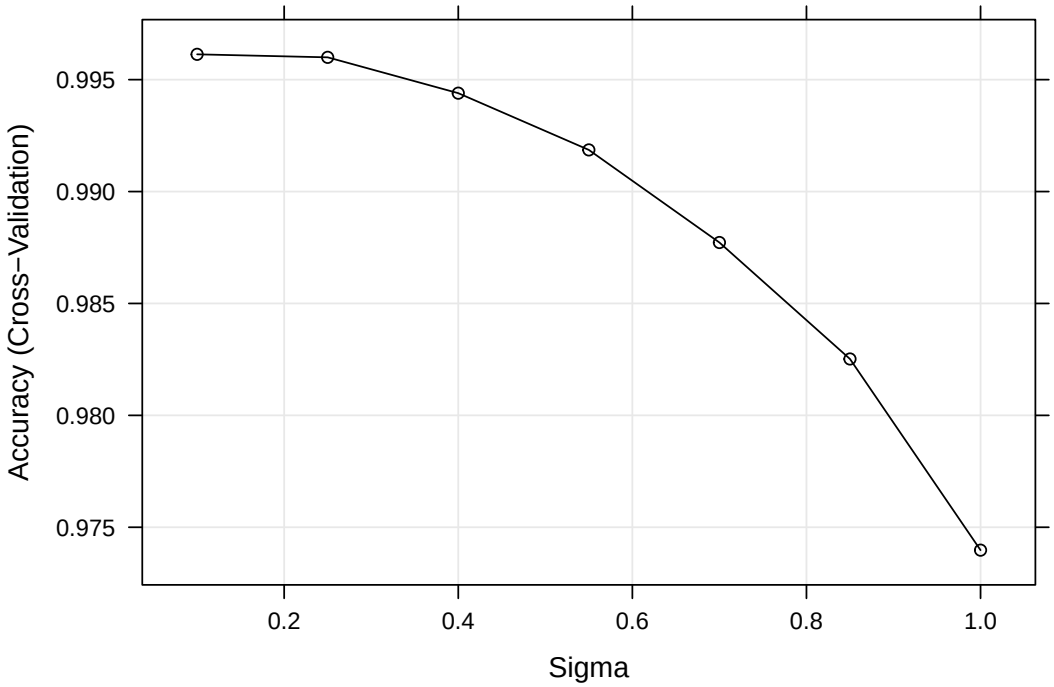
1.4.2 SVM con kernel RBF

Usamos SVM con el kernel **radial basis function** (ver Karatzoglou, Smola, and Hornik 2018) cuyo parámetro configurable es el coste y σ en:

$$K(\mathbf{x}, \mathbf{x}') = exp\left(-\frac{\|\mathbf{x}, \mathbf{x}'\|^2}{2\sigma^2}\right)$$

```
svmRBFGrid = expand.grid(C=1,sigma=seq(0.1,1,by = 0.9/6))
svmRBFRes = train(x=ndtrainX,y=dtrainY,method = "svmRadial",
                  trControl = tC, tuneGrid = svmRBFGrid)
```

```
gtThan = svmRBFRes$results$Accuracy > 0.5
plot(svmRBFRes)
```




```
paste("Mejor sigma: ", toString(svmRBFRes$bestTune$sigma))
```

```
## [1] "Mejor sigma:  0.1"
```

```
paste("Precisión: ", svmRBFRes$results[svmRBFRes$results$sigma ==
                                       svmRBFRes$bestTune$sigma,]$Accuracy)
```

```
## [1] "Precisión:  0.996131019136627"
```

```
svmRBFTune = svmRBFRes$bestTune
```

1.4.3 Neural network

Usamos el perceptrón multicapa con backpropagation, (ver Bergmeir 2017). He observado con experimentaciones paralelas que el uso de **weight decay** no mejora el resultado en CV y sólo ralentiza el aprendizaje, probablemente por el reducido número de iteraciones que trabajamos en ordenadores personales. Por otro lado, la configuración de parámetros que mejores resultados da para backpropagation estándar (ver "SNSS - Parameters of the Learning Functions" 2017) es $\eta = 0.9$ y $d_{max} = 0.3$, éste último es el umbral mínimo de salida para propagar un resultado, lo cual previene el sobreentrenamiento. Por otro lado he probado con incrementar el número de iteraciones pero no consigue mejorar el resultado.

```
nodes_c = floor(seq(0,by = 50/3,length.out = 4))
nnGrid = expand.grid(layer1=nodes_c,layer2=nodes_c,layer3=nodes_c)
nnRes = train(x=ndtrainX,y=dtrainY,method = "mlpML", trControl = tC,
              tuneGrid = nnGrid, learnFunc = "Std_Backpropagation",
              learnFuncParams = c(0.9,0.3), maxit = 100)
```

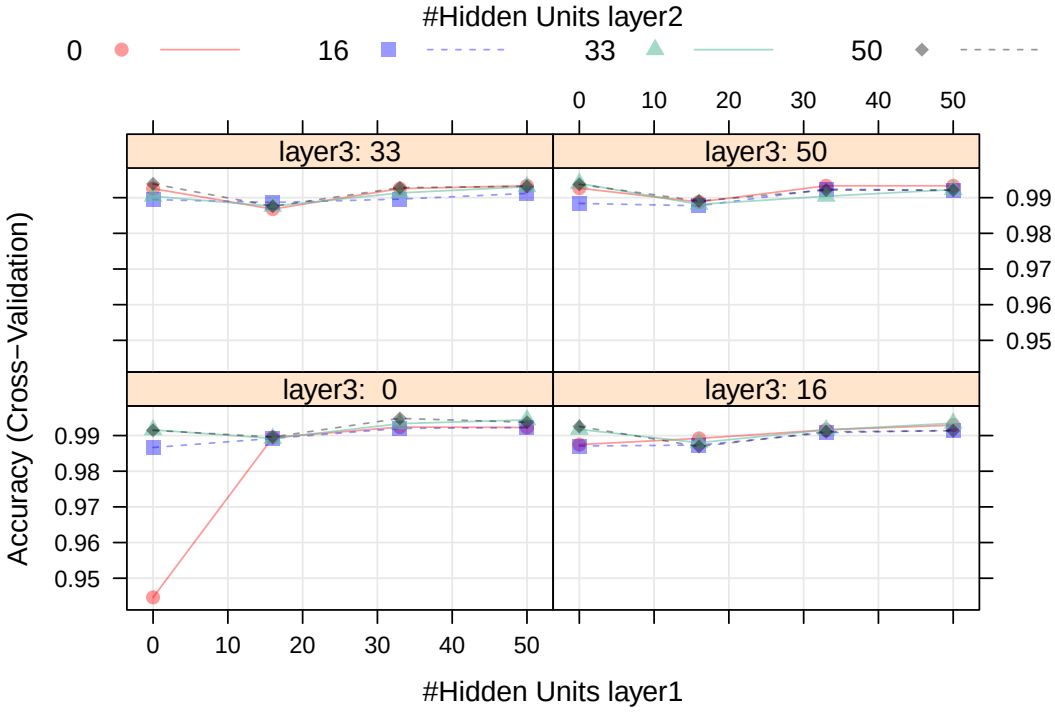
```
nnTune = nnRes$bestTune
```

```
nnRes
```

```
## Multi-Layer Perceptron, with multiple layers
##
## 7494 samples
##   16 predictors
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5995, 5996, 5994, 5995, 5996
## Resampling results across tuning parameters:
##
##   layer1  layer2  layer3  Accuracy   Kappa
##    0       0       0      0.9446234  0.9384644
##    0       0      16      0.9874570  0.9860609
##    0       0      33      0.9925278  0.9916961
##    0       0      50      0.9926619  0.9918451
##    0      16       0      0.9866570  0.9851718
##    0      16      16      0.9870560  0.9856153
##    0      16      33      0.9894595  0.9882860
##    0      16      50      0.9883916  0.9870995
##    0      33       0      0.9915949  0.9906592
##    0      33      16      0.9917273  0.9908065
##    0      33      33      0.9903935  0.9893242
##    0      33      50      0.9939954  0.9933270
##    0      50       0      0.9914604  0.9905097
##    0      50      16      0.9925284  0.9916967
##    0      50      33      0.9938626  0.9931795
##    0      50      50      0.9937288  0.9930307
##   16       0       0      0.9894585  0.9882852
##   16       0      16      0.9891930  0.9879901
##   16       0      33      0.9867901  0.9853197
##   16       0      50      0.9887915  0.9875439
##   16      16       0      0.9891913  0.9879882
##   16      16      16      0.9873238  0.9859127
##   16      16      33      0.9886589  0.9873967
##   16      16      50      0.9877247  0.9863582
##   16      33       0      0.9891923  0.9879893
##   16      33      16      0.9879900  0.9866534
##   16      33      33      0.9875905  0.9862093
##   16      33      50      0.9881249  0.9868032
##   16      50       0      0.9895915  0.9884329
##   16      50      16      0.9869235  0.9854681
##   16      50      33      0.9875912  0.9862100
##   16      50      50      0.9890578  0.9878399
```

```
##   33       0       0      0.9923944  0.9915479
##   33       0      16      0.9915947  0.9906588
##   33       0      33      0.9925283  0.9916965
##   33       0      50      0.9933289  0.9925864
##   33      16       0      0.9919947  0.9911035
##   33      16      16      0.9907931  0.9897682
##   33      16      33      0.9895923  0.9884338
##   33      16      50      0.9923951  0.9915487
##   33      33       0      0.9933285  0.9925860
##   33      33      16      0.9915939  0.9906582
##   33      33      33      0.9913275  0.9903622
##   33      33      50      0.9903933  0.9893240
##   33      50       0      0.9947966  0.9942175
##   33      50      16      0.9910612  0.9900664
##   33      50      33      0.9927950  0.9919930
##   33      50      50      0.9921275  0.9912511
##   50       0       0      0.9922609  0.9913992
##   50       0      16      0.9929280  0.9921408
##   50       0      33      0.9933288  0.9925863
##   50       0      50      0.9933287  0.9925862
##   50      16       0      0.9922608  0.9913994
##   50      16      16      0.9914606  0.9905102
##   50      16      33      0.9911938  0.9902135
##   50      16      50      0.9919940  0.9911030
##   50      33       0      0.9943959  0.9937722
##   50      33      16      0.9934625  0.9927349
##   50      33      33      0.9930619  0.9922896
##   50      33      50      0.9922611  0.9913998
##   50      50       0      0.9937290  0.9930310
##   50      50      16      0.9913276  0.9903623
##   50      50      33      0.9931951  0.9924376
##   50      50      50      0.9921272  0.9912508
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were layer1 = 33, layer2 = 50
##  and layer3 = 0.
```

```
plot(nnRes)
```



Por lo que observamos el uso de más de una capa no mejora el error en la validación.

1.4.4 Reflexión sobre los resultados

Hemos planteado tres modelos, cada uno con suficiente reputación experimental en distintos campos y los resultados son interesantes. En primer lugar SVM con el kernel lineal ha conseguido dar resultados competitivos, a lo largo de las distintas ejecuciones que he realizado suele dar un valor sobre 0.985. Por otro lado SVM con kernel RBF y NN dan resultados prácticamente idénticos, con la mejor configuración de ambos el límite experimental en la capacidad de clasificación suele estar entre 0.995 y 0.996.

El problema surge al tener decidir sobre qué modelo elegir, aunque los dos contendientes son claros, SVM-RBF o NN. Lo analizo atendiendo a las particularidades del problema en el siguiente punto.

1.5 Selección y ajuste del modelo final

El primer problema que tenemos que atajar es la función que intentamos aprender, la dificultad radica en que estamos hipotetizando que seremos capaces de adivinar la escritura de individuos desconocidos en base a la escritura de individuos conocidos suficientemente grande, sin embargo, para el entrenamiento con **5-fold CV** estamos aprendiendo la escritura de los mismos individuos que intentamos clasificar posteriormente, estimar el comportamiento en E_{test} —y consecuentemente en E_{out}— a partir del error en la validación es osado y no tenemos garantías teóricas de que esto suceda, tampoco tenemos en los datos del train la

información sobre a qué usuario pertenece un dígito para hacer CV sobre distintos usuarios entrenando con unos y estimando con otros.

Por otro lado, y aún en relación a la función de clasificación que estamos aprendiendo, y como se infiere de lo expuesto anteriormente no tenemos garantías para definir la muestra como independiente e idénticamente distribuida sobre una distribución de probabilidad desconocida —y tenemos motivos de sobra para sospechar que no es así—. Por esto no podemos utilizar la dimensión de **Vapnik-Chervonenkis** para hacer una estimación de error fuera de la muestra con la desigualdad de **Hoeffding**. Aún así analicemos la dimensión VC para cada uno de los modelos, para cruzar los dedos y esperar que nos pueda decir algo sobre la capacidad de generalización.

El principal problema entre los dos modelos respecto a la dimensionalidad es el siguiente, mientras que teóricamente —lo siguiente no es cierto para la representación de números con coma flotante en ordenadores de $n$bits— la d_{VC} de NN es finita limitada para la función de activación sigmoide por $O(|E|^2 \text{ů} |V|^2)$, la d_{VC} para SVM-RBF es infinita, aunque está limitada por $\left\lceil \frac{4R^2}{\gamma^2} \right\rceil$, siendo R el radio de la mínima hiperesfera que incluye todos los puntos de la muestra y γ el márgen de la frontera, esto quiere decir que para márgenes altos del SVM la complejidad de la clase es pequeña y consecuente la generalización es buena.

Con todo lo anterior dicho, y abandonando el respaldo teórico pues no podemos extraer garantías estadísticas sobre la esperanza de error fuera de la muestra tenemos que atenernos a los resultados experimentales de otros autores sobre este problema y problemas similares.

Versiones más refinadas de ambos modelos —CNN(ver F. Alimoglu and Alpaydin 1997) y SVM-RBF(ver Barbuzzi, Impedovo, and Pirlo 2012)— han conseguido muy buenos resultados en la versión por píxeles de este problema —dejando a un lado versiones refinadas del KNN que consiguen muy buenos resultados aunque con una huella de computo y memoria enorme— , aunque SVM aparentemente mejores, por esto, por el resultado ligeramente superior sobre **5-fold CV** y por la reconocida capacidad de generalización de este modelo para γ altos —si es que es posible generalizar, que no lo sabemos— vamos a elegir el SVM con el kernel radial para ajustar el modelo y ver qué tal lo hace.

```
tr = train(x = ndtrainX, y = dtrainY, method = "svmRadial",
           trControl = trainControl(method = "none"),
           tuneGrid =  svmRBFTune)
```

```
pred = predict(tr,ndtestX)
etest = length(which(pred != dtestY))/nrow(ndtestX)
paste("Error sobre el test: ", etest)
```

```
## [1] "Error sobre el test:  0.0194396798170383"
```

```
paste("Tasa de aciertos: ", 1-etest)
```

```
## [1] "Tasa de aciertos:  0.980560320182962"
```

Luego analizaremos este resultado, vamos a ver ahora, sólo por curiosidad, qué habría pasado si hubieramos elegido la red neuronal, antes de nada hay que advertir, que si en este paso el resultado de NN fuera significativamente mejor no podríamos elegirla de todos modos, pues no podemos ajustar el modelo basándonos en el test, tendríamos que conseguir nuevos datos

y repetir el experimento sobre la red neuronal para comprobar la validez del resultado. Aun así puede ser interesante ver qué sucede.

```
trnn = train(x = ndtrainX, y = dtrainY, method = "mlpML",
             trControl = trainControl(method = "none"),
             tuneGrid =  nnTune, maxit=100)
```

```
prednn = predict(trnn,ndtestX)
etestnn = length(which(prednn != dtestY))/nrow(ndtestX)
paste("Error sobre el test: ", etestnn)
```

```
## [1] "Error sobre el test:  0.0714694110920526"
```

```
paste("Tasa de aciertos: ", 1-etestnn)
```

```
## [1] "Tasa de aciertos:  0.928530588907947"
```

Aunque esto no sirva de nada parece que este resultado aislado apoya nuestra decisión.

1.6 Idoneidad de la métrica y estimación del error fuera de la muestra

```
confusionMatrix(pred,dtestY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 351   0   0   0   0   0   0   0   0   0
##          1   0 356   2   1   0   0   0  14   0   2
##          2   0   7 362   0   0   0   0   2   0   0
##          3   0   0   0 333   0   5   0   0   0   0
##          4   0   1   0   0 359   0   0   1   0   0
##          5   0   0   0   0   4 329   0   0   1   1
##          6   0   0   0   0   1   0 336   0   0   0
##          7   0   0   0   0   0   0   0 340   0   3
##          8  12   0   0   0   0   0   0   0 335   1
##          9   0   0   0   2   0   1   0   7   0 329
##
## Overall Statistics
##
##                Accuracy : 0.9806
##                  95% CI : (0.9754, 0.9849)
##     No Information Rate : 0.1041
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9784
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9669   0.9780   0.9945  0.99107   0.9863  0.98209
## Specificity            1.0000   0.9939   0.9971  0.99842   0.9994  0.99810
## Pos Pred Value         1.0000   0.9493   0.9757  0.98521   0.9945  0.98209
## Neg Pred Value         0.9962   0.9974   0.9994  0.99905   0.9984  0.99810
## Prevalence             0.1038   0.1041   0.1041  0.09605   0.1041  0.09577
## Detection Rate         0.1003   0.1018   0.1035  0.09520   0.1026  0.09405
## Detection Prevalence   0.1003   0.1072   0.1061  0.09663   0.1032  0.09577
## Balanced Accuracy      0.9835   0.9860   0.9958  0.99475   0.9928  0.99010
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity           1.00000  0.93407  0.99702  0.97917
## Specificity           0.99968  0.99904  0.99589  0.99684
## Pos Pred Value        0.99703  0.99125  0.96264  0.97050
## Neg Pred Value        1.00000  0.99239  0.99968  0.99778
## Prevalence            0.09605  0.10406  0.09605  0.09605
## Detection Rate        0.09605  0.09720  0.09577  0.09405
## Detection Prevalence  0.09634  0.09806  0.09949  0.09691
## Balanced Accuracy     0.99984  0.96655  0.99646  0.98800
```

Sobre el ajuste del modelo podemos observar que la especificidad es aparentemente muy buena, siendo capaz de discriminar con éxito los casos negativos y por otro lado, la sensibilidad depende más de la clase concreta. El mayor motivo de error parece el 1 y el 7 —parece razonable— y el 8 y el 0, que mientras parecía razonable en el problema basado en píxeles en éste me ha sorprendido, especialmente porque al dibujar los números —tanto el ejemplo como el centroide de la clase— parecían distinguirse exitosamente, sin embargo, las transformaciones no lineales que realizar el kérnel RBF no nos permiten ver a qué se debe esta situación.

La calidad del modelo ajustado depende de la terea destino de éste, y los resultados son similares a los obtenidos por Alpaydin y Alimoglu (ver 1998) aplicando el KNN; sin embargo el SVM-RBF una vez entrenado es mucho más rápido y la huella de memoria es mucho menor, sin embargo añadir más datos supone el entrenamiento completo del modelo otra vez, no como el KNN que por su funcionamiento puede consumir más datos y dar mejores resultados sin tener que pasar por una etapa de entrenamiento

1.7 Conclusiones

Nos hemos enfrentado a un problema sobre el que no teníamos garantías de generalización y experimentalmente hemos conseguido obtener un resultado satisfactorio —al menos en la similitud del error en la validación con el del test—. No hemos conseguido aparentemente mejorar al KNN pero tenemos un modelo mucho más eficiente para la misma tarea.

Por lo que he podido leer, se está avanzando en el uso de CNN y combinaciones de varias NN más sencillas para resolver el problema con éxito —la versión con píxeles—, pero sin embargo en mi experimentación no conseguí encontrar indicios que apoyaran el uso del MLP, es posible que sea debido a que el ajuste de los parámetros del MLP es una tarea computacionalmente más exigente que el cálculo de los parámetros de un SVM con la

misma precisión, sencillamente porque el entrenamiento consume más tiempo con iteracione suficientes y porque son más parámetros.

Queda pendiente afinar aún más los parámetros del SVM para poder quizás rascar unas centésimas al error y para el futuro experimentar con la red neuronal con **weigth-decay** y con inercia, con suficiente tiempo y capacidad de computación para ajustar los parámetros y ver si consigo mejorar el resultado, también creo que debido a que es un problema con tan pocas características de entrada —sólo 16 variables— el uso de muchas capas ocultas no está justificado, pues una sóla capa suficientemente grande debería ser capaz de ajustar la función con la mejor de sus capacidades.

Bibliografía

Alimoglu, F., and E. Alpaydin. 1997. "Combining Multiple Representations and Classifiers for Pen-Based Handwritten Digit Recognition." In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, 2:637–40 vol.2. doi:10.1109/ICDAR.1997.620583.

Alpaydin, E., and Fevzi. Alimoglu. 1998. "Pen-Based Recognition of Handwritten Digits." Bogazici University. https://archive.ics.uci.edu/ml/machine-learning-databases/pendigits/pendigits.names.

Barbuzzi, D., D. Impedovo, and G. Pirlo. 2012. "Benchmarking of Update Learning Strategies on Digit Classifier Systems." In *2012 International Conference on Frontiers in Handwriting Recognition*, 35–40. doi:10.1109/ICFHR.2012.186.

Bergmeir, Christoph. 2017. "Neural Networks Using the Stuttgart Neural Network Simulator." R Project. https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf.

Karatzoglou, Alexandros, Alex Smola, and Kurt Hornik. 2018. "Kernel-Based Machine Learning Lab." R Project. https://cran.r-project.org/web/packages/kernlab/kernlab.pdf.

"SNSS - Parameters of the Learning Functions." 2017. Universität Tübingen. http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node52.html#SECTION00540000000000000000.