Fundamentos de Aprendizaje Automático

Práctica 2

Grupo1461
Mario Valdemaro García Roque
Manuel Reyes Sánchez

Apartado 1 – datos P1

Validación cruzada 5:

	Weka	Naive-Bayes	Naive-Bayes Laplace	Naive-Bayes dist. Normal	Naive-Bayes Laplace y dist. normal
Tic-tac-toe	30,27%	30,89%	29,53%	30,79%	30,05%
crx	22,1%	37,1%	30,14%	40,58%	39,42%
Iris	4%	15,3%	8,0%	8,0%	6,0%

Validación cruzada 10:

	Weka	Naive-Bayes	Naive-Bayes Laplace	Naive-Bayes dist. Normal	Naive-Bayes Laplace y dist. normal
Tic-tac-toe	30,38%	29,95%	30,26%	29,75%	30,16%
crx	22,32%	39,71%	30,14%	40,58%	39,85%
Iris	4%	13,99%	8,66%	7,99%	7,33%

Vamos a analizar los datos obtenidos teniendo en cuento si cuentan los conjuntos de datos con atributos continuos o no.

Como se puede observar los datos observados en crx e Iris muestran que Weka obtiene mejores resultados que nuestros clasificadores. Esto se debe quizá a ciertas mejoras que pueda tener weka implementadas sobre Naive-Bayes o en la lectura de los datos. Por ejemplo, nos hemos dado cuenta de que los datos continuos son discretizados lo que mejora las predicciones. Por ejemplo, si nuestro programa lee 0,001 y 0,002 van a ser considerados diferentes, mientras que weka los considerara iguales (de acuerdo a un criterio que se puede cambiar). No hemos conseguido desactivar esta opción.

Tic-tac-toe tiene atributos solo de tipo nominales y como se puede observar nuestros resultados se aproximan a los valores de weka, ya que en estos datos no se pueden realizar discretizaciones.

Apartado 2 - knn

Datos de nuestra implementación (10 folds):

Vecino	1	3	5	11	21	51
Wine Q.	40,34%	45,84%	47,9%	49,47%	47,4%	48,22%
wdbc	8%	7,56%	6,33%	6,5%	7,38%	8,28%
Heart D.	56,75%	55,11%	54,12%	49,17%	46,86%	46,20%

Datos de weka (10 folds):

Vecino	1	3	5	11	21	51
Wine Q.	35,21%	41,21%	41,84%	41,43%	40,71%	42,21%
wdbc	4,04%	3,16%	2,99%	2,99%	3,16%	4,75%
Heart D.	45,54%	45,87%	45,87%	43,89%	41,91%	42,57%

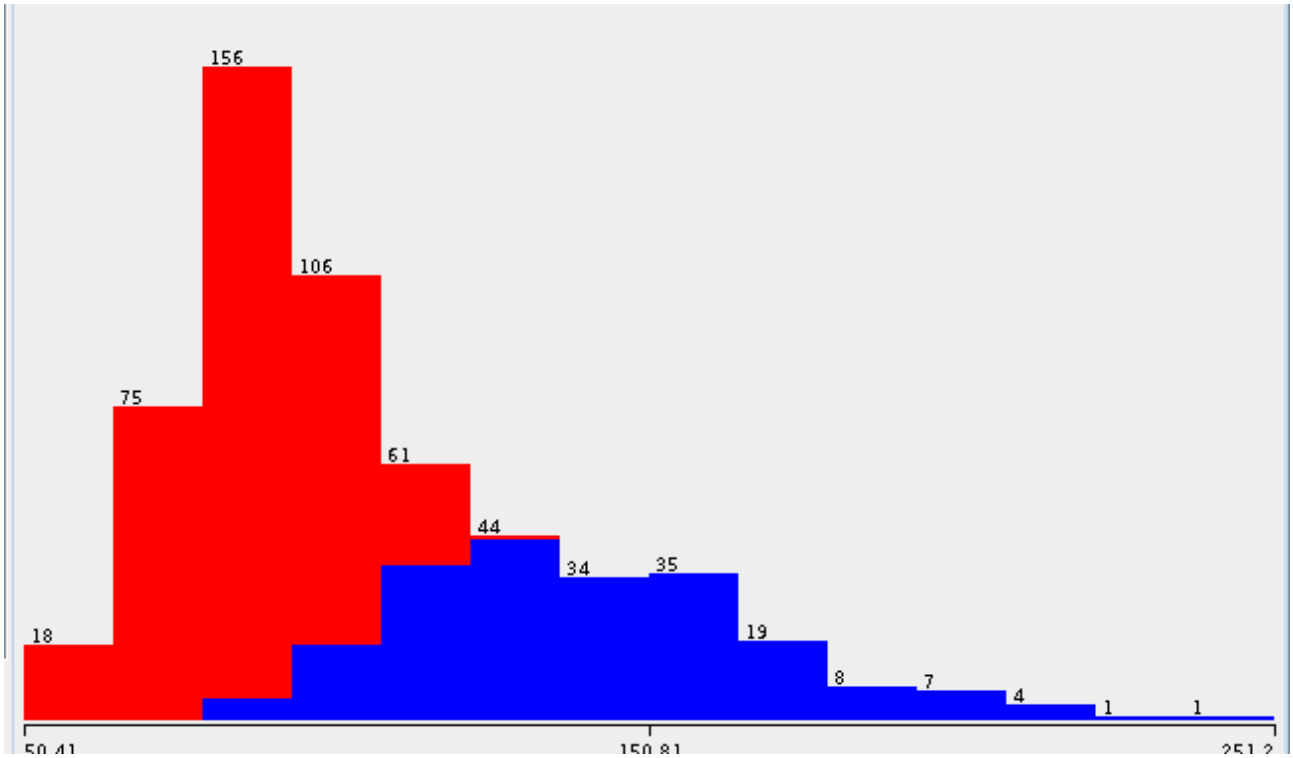
Se puede observar como nuestra implementación y weka arrojan datos similares, siendo ligeramente mejores en weka.

Nuestros datos varían entre cada ejecución ya que desordenamos los datos de forma aleatorio, mientras que weka obtiene siempre el mismo resultado ya que o bien no mezcla o bien realiza siempre la misma desordenación.

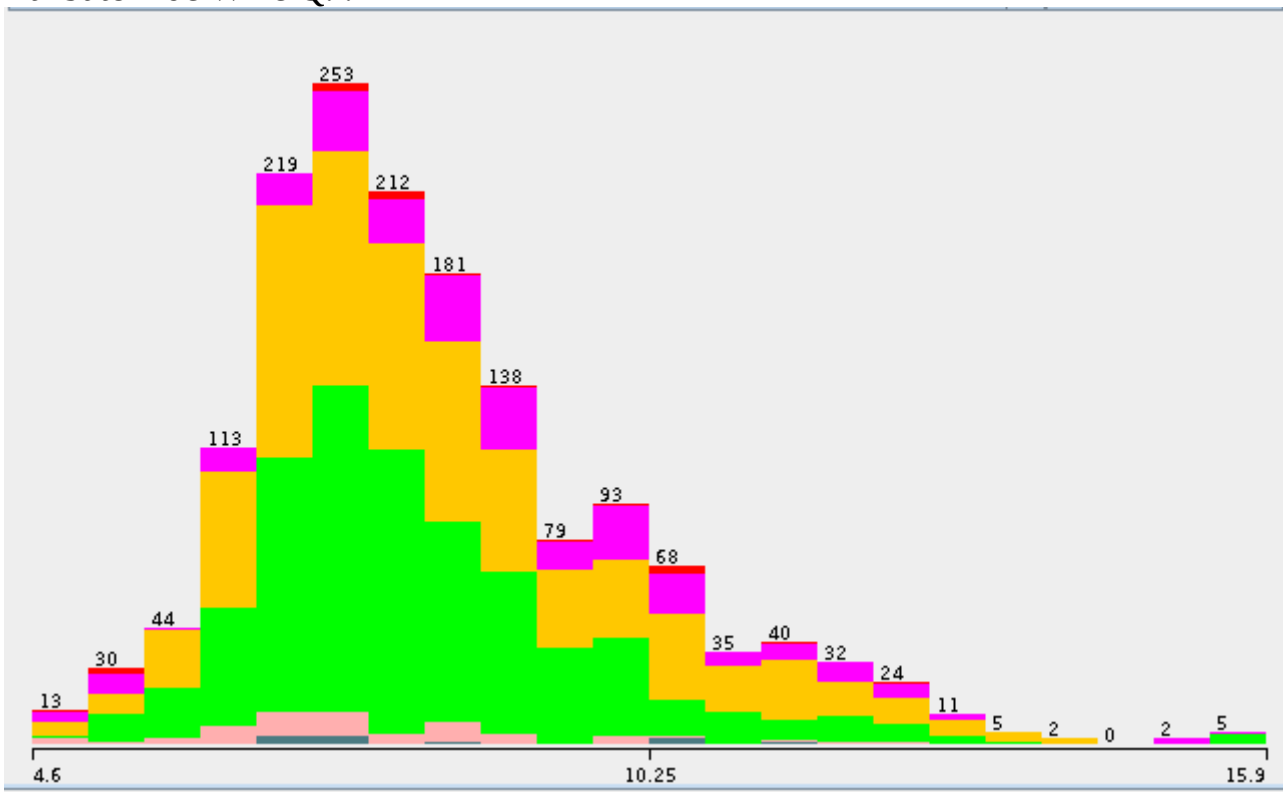
Dependiendo de nuestros datos a analizar el numero de vecino debe variar para obtener el mejor resultado. Por ejemplo en Wine Q. Se observa como de los k evaluados 1 es el mejor, sin embargo en wdbc es mejor 5.

También observamos que los mejores resultados los obtenemos en los datos de wdbc, clasificando con un error entorno a un 6-8% con nuestro programa y un 3-5% con weka, mientras si probamos con los otros datos obtenemos un tasa de error muy significativa. Esto se debe a que al normalizar las zonas donde se separa cada clase se encuentran muy acotadas pudiendo clasificar un dato de una determinada zona como una clase u otra. Sin embargo las fronteras que encontramos en los otros 2 archivos no están tan bien definidas:

Atributo 15 de wdbc:



Atributo 1 de Wine Q. :



Como se observa, las medias de wdbc están mas separadas que en Wine Q.

Apartado 3 – Regresión Logística

Datos: wdbc (10 folds)

	Resultado
Épocas=50; t. aprendizaje= 10^-2	8,78%
Épocas=50; t. aprendizaje= 10^-3	12,82%
Épocas=50; t. aprendizaje= 10^-4	14,75%
Épocas=100; t. aprendizaje= 10^-2	11,24%
Épocas=100; t. aprendizaje= 10^-3	11,77%
Épocas=100; t. aprendizaje= 10^-4	9,5%
Épocas=150; t. aprendizaje= 10^-2	8,44%
Épocas=150; t. aprendizaje= 10^-3	9,31%
Épocas=150; t. aprendizaje= 10^-4	10,89%
Épocas=1000; t. aprendizaje= 10^-2	10,78%
Épocas=1000; t. aprendizaje= 10^-3	10,19%
Épocas=1000; t. aprendizaje= 10^-4	9,13%

Los datos son medias de varias ejecuciones en este caso, ya que dependiendo de como se inicie el vector la recta tardará mas o menos épocas en alcanzar el resultado óptimo. Cuanto mayor sea nuestro numero de épocas menores serán las diferencias entre ejecuciones, pues se consigue corregir la recta por muy mal que se generara al inicio.

Los resultados son similares a los obtenidos por vecinos próximos, siendo ligeramente superiores en el caso de knn. La forma de clasificar en ambos modelos es similar si (como ocurre en este conjunto de datos) cuando realizamos la clasificación knn los datos quedan bien definidos en dos grupos separados de manera similar a como lo haría la recta de regresión. También es destacarle observar que knn es en nuestro caso más rápido ya que el numero de épocas condiciona el tiempo de ejecución.

En weka los datos obtenidos son similares, siendo ligeramente mejores, debido quizá a optimizaciones de weka, como una mejor elección del vector de inicio.

Apartado 4 – Mejorar knn

DKNN (DynamicKNN)

Realizamos una mejora orientada a encontrar el mejor vecino posible de entre un rango de vecinos dado y un incremento.

Cada vez que DKNN es entrenado guarda el mejor vecino de acuerdo al siguiente proceso: se vuelve a realizar una partición de los datos para tener un conjunto de datos de train y otro de validación. Se realiza entrenamiento y validación para todos los vecinos, que estén entre el mínimo y el máximo (si es posible) y los incrementos indicados. El menor error es guardado.

Una vez terminado este proceso podemos llamar a un método que nos devolverá el vecino optimo mas veces obtenido (la moda).

Esto tiene un coste de tiempo muy alto, que podría ser mejorado estableciendo condiciones de parada. Por ejemplo guardando la prueba anterior y comparando con el resultado siguiente, parando el proceso si se empeora el resultado.

KNNDW (KNN Distance Weighted)

Por otro lado hemos cambiado la importancia que se le da a cada vecino de los k seleccionados de acuerdo a su distancia al elemento a clasificarlos.

Esta técnica depende de los datos con los cuales estemos trabajando, por ejemplo, en heart-disease no hay mejora aparente. En el caso, por ejemplo de Wine Q. podemos observar como los datos mejoran:

Vecino	1	3	5	11	21	51
Wine Q. KNN	40,34%	45,84%	47,9%	49,47%	47,4%	48,22%
Wine Q.	40,83%	29,71%	39,33%	38,14%	36,21%	35,27%

KNNDW						

Vemos como con 1 vecino no hay cambio, lo cual no tendría lógica, y que con mas vecinos obtenemos una mejora generalizada de alrededor 10 puntos. Esto es debido a que ahora contamos con la “opinión” de esos k vecinos pero damos mas importancia a los que se encuentran mas cercanos al elemento a clasificar.

Esto podría ser usado junto con el anterior para hallar el mejor k de KNNDQ, obteniendo el método DKNNDQ

KNNNB (KNN Naive-Bayes)

Por último hemos realizado un cambio a KNN para que en vez de simplemente predecir la moda de los K vecinos mas próximos con esos datos se genere un clasificador Naive-Bayes. Una vez realizado este entrenamiento predecimos el valor de la clase para el dato.

Dependiendo de los datos puede producirse mejora o no. También depende de el numero de vecinos que usemos, tendiendo en este caso a ser mejores k algo mayores. Al igual que en el método anterior este método se puede conjugar con el primero para conseguir DKNNNB y conseguir el mejor k.

De nuestros conjuntos de datos se observa una mejora notable en los de wine-quality:

Vecino	1	3	5	11	21	51
Wine Q. KNN	40,34%	45,84%	47,9%	49,47%	47,4%	48,22%
Wine Q. KNNNB	84,4%	59,54%	43,65%	37,96%	36,65%	36,77%

Como se mencionaba arriba las predicciones mejoran con un numero alto de vecinos, pues si bien gracias a las distancias nos quedamos con datos mas próximos naive-bayes necesita una población significativa.