Universidad Nacional de Rosario

Introducción al Aprendizaje Automatizado

Trabajo Práctico III

Villagra Martín

8 de junio de 2017

Introducción

Este trabajo se centra sobre la familias de algoritmos Naive Bayes para resolver los problemas de clasificación. Se analiza su desempeño con diferentes variaciones del mismo bajo diferentes datasets y se explican los resultados.

Ejercicio 1

Las modificaciones pertinentes se encuentran en el archivo *nb_n.cpp* (el cual contiene también las del Ejercicio 4).

Ejercicio 2

Como se puede apreciar en la Figura 1, obtuvimos un error significativamente menor que con los otros métodos. Lo que sucede es que el algoritmo aproxima cada clase como una distribución normal en cada feature y para este dataset en concreto es justamente lo que se tiene, la aproximación no es tanto una aproximación. Era de esperar que bajo estas condiciones Naive Bayes funcione mucho mejor que los otros.

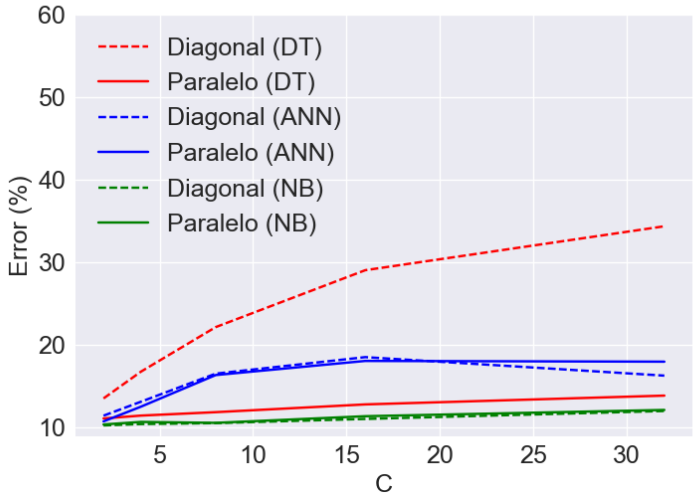


Figura 1: Error porcentual discreto en test en función de la cantidad de dimensiones.

También se destaca que tanto el paralelo como el diagonal tienen el mismo error, que es muy cercano al del clasificador de Bayes. Para el diagonal todas la features serán un buen discriminante puesto que la media calculada para la clase serán bien diferentes. Para el paralelo sólo la primer dimensión aporta probabilidades altas,

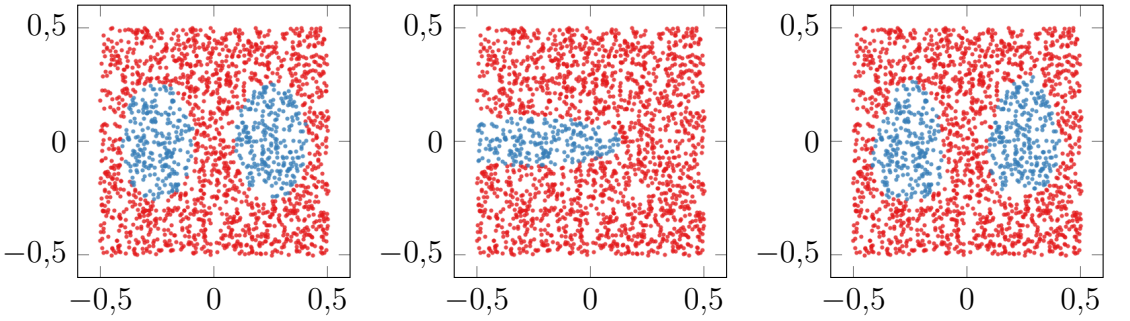
mientras que las demás al tener las medias de las clases centradas en el 0 aportan la misma probabilidad para toda las clases, sin embargo esta diferencia en la primer dimensión es suficiente para elegir entre una clase y la otra.

El error aumenta ligeramente con las dimensiones, lo que puede deberse al sobre-ajuste.

Ejercicio 3

Dos elipses

El algoritmo produce un error del 22.5 %. En la Figura 2 vemos que la predicción es muy mala. La media de la clase 0 (en rojo) tiende a estar en el centro y la de la clase 1 (en azul) también pero ligeramente más corrida hacía un lado (esto es porque la cantidad de muestras de la clase azul es menor, por lo que es esperable que su media se aleje de la real, que es también 0). El error producido en test es pura coincidencia pues en este caso una aproximación por Gaussianas es totalmente incorrecto para este dataset. No es de sorprender que en este caso las redes neuronales tengan un resultado mucho mejor. Por último notar en la Figura 2(b) que la elipse azul está alargada, debido a que la varianza en x de la clase 1 es alta por la separación entre las dos elipses.



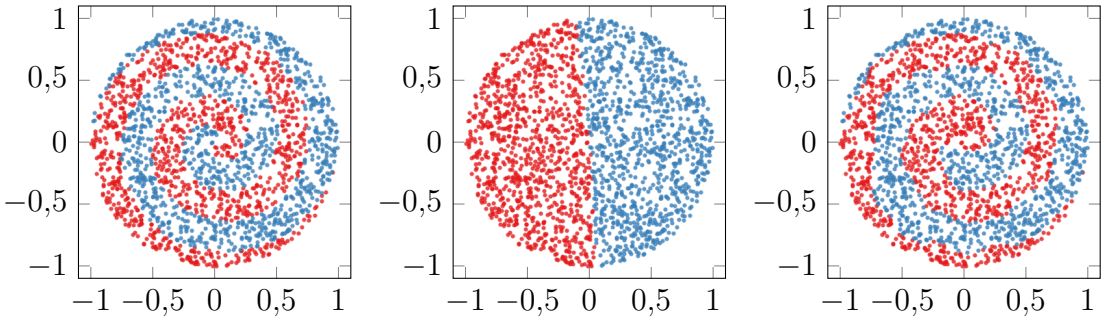
(a) Conjunto de Test. (b) Naive Bayes. (c) Redes Neuronales.

Figura 2: Predicciones sobre el conjunto de test del dataset dos_elipses.

Espirales

El algoritmo produce un error del 43 %. En la Figura 3 se puede apreciar gráficamente este pésimo resultado. El problema es que ambas clases tienen medias y varianzas

similares, por lo que es imposible discernir entre ellas utilizando este criterio. Pequeñas variaciones en la media de las clases llevan a crear esa predicción y mejorar en un 7 % la solución trivial.



(a) Conjunto de Test. (b) Predicción de Naive Bayes. (c) Redes Neuronales.

Figura 3: Predicciones sobre el conjunto de test del dataset espirales.

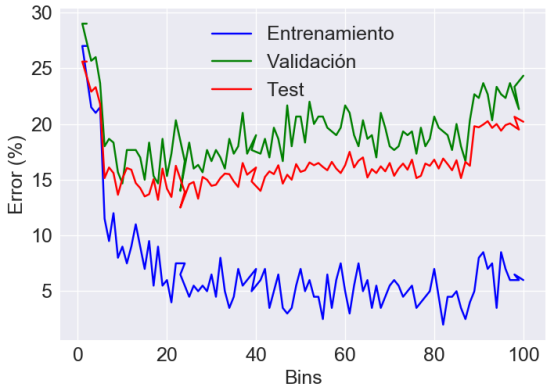
Una vez más las redes neuronales llevan la ventaja pues poseen la capacidad de relacionar los atributos (especialmente en este caso donde las clases dependen totalmente de la relación entre x e y), mientras que este modelo asume que los atributos son independientes.

Ejercicio 4

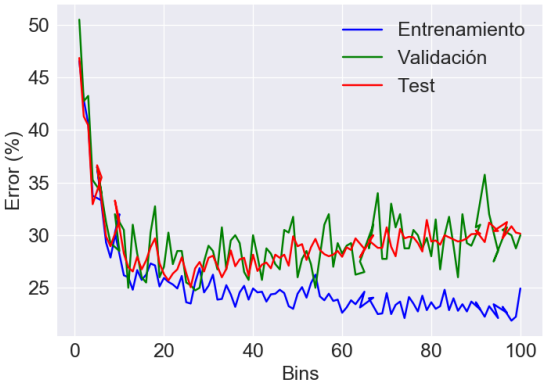
Se implementó la clasificación con histogramas en el archivo *nb_n.cpp*. Se suple un parámetro adicional en el archivo de configuración que indica la cantidad de bins, si este es 0 o no existe se utiliza la aproximación Gaussiana.

En la Figura 4 se observan los errores para diferentes cantidades de bins. En ambos gráficos se ve sobreajuste, aunque para dos_elipses es más notorio probablemente debido a la diferencia en la cantidad de puntos. En espirales se comienza con un error mucho más alto pues en este dataset las clases están entrelazadas y no se puede discernir entre ellas con una poca cantidad de bins. El sobreajuste se produce porque los bins se vuelven demasiado pequeños (estos bins se especializan indebidamente en clasificar muestras individuales del conjunto data).

En la Figura 5 se puede ver que las diferentes clases están delimitadas por rectas: los límites de los bins. En algunos prevalecen los horizontales, y en otros los vérticales. Aunque se aumente la cantidad de bins, no es posible lograr una mejor clasificación, porque este modelo asume que los features son independientes cuando en realidad no lo son. En consecuencia al aumentar demasiado la cantidad de bins sólo se logra sobreajuste.

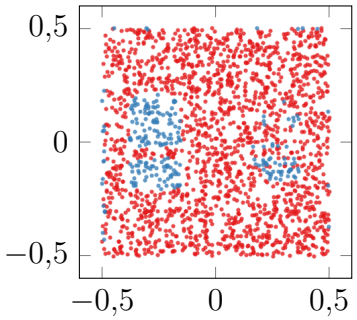


(a) Dataset dos_elipses.

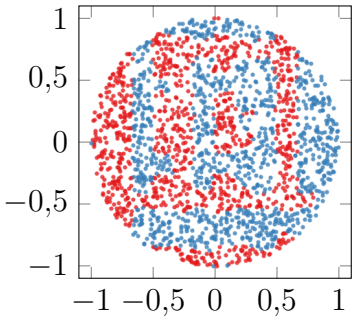


(b) Dataset espirales.

Figura 4: Errores vs cantidad de bins.



(a) Dataset dos_elipses, bins=12, Error=15 %.



(b) Dataset espirales, bins=21. Error=26 %

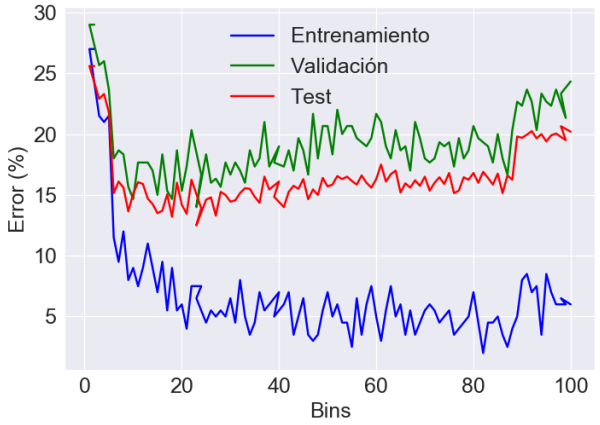
Figura 5: Predicciones con menor error en validación de Naive Bayes con Histogramas.

Ejercicio 5

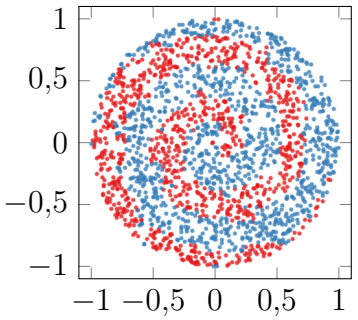
Para hacer los atributos dependientes, se optó por seguir la idea de los histogramas. Sólo que en lugar de un histograma con b bins por cada cada atributo, se dividió el espacio de todos los a inputs en b^a hiperregiones de igual volumen. Por ejemplo, para el caso $2D$ el plano xy se dividió en $b \times b$ rectángulos. Cada hiperregión representaría un bin y por cada clase se genera un único histograma. La implementación se encuentra en el archivo *nb_n_cubes.cpp*.

Para el dataset espirales se logró un error del 10 %. En la Figura 6 se muestran los resultados. Está técnica genera sobreajuste aún más rápido, pues si los hiperbins son muy pequeños se especializan en reconocer sólo una muestra del conjunto de train.

Sin embargo estos efectos se mitigan al utilizar el conjunto de validación, llevando a una predicción bastante aceptable.



(a) Error vs Cantidad de Bins.



(b) Predicción en test con menor error en validación.

Figura 6: Resultados de Naive Bayes con features dependientes.

Ejercicio 6

En los ejercicios anteriores se utilizarons bins con un tamaño constante para cada feature, y a su vez la cantidad de bins era pre-establecida y común a todos los atributos. Los métodos de discretización como este proveen alternativas a esta restricción.

Implementación

El código se encuentra en el archivo *nb_n_entropy.cpp*. Para implementar eficientemente el algoritmo la primera observación que se hizo fue que si el conjunto de entrenamiento se ordena por la feature que se quiere discretizar, entonces las particiones están representadas por intervalos en el arreglo de las muestras. Esto es útil porque tanto el estado de la recursiva como las subparticiones a probar pueden ser representadas más eficientemente por intervalos y no como subconjuntos. A su vez el calculo de la entropía puede ser optimizado teniendo en cuenta esta observación utilizando la técnica de prefix sums: con un precalculo linear, la entropía de un subarreglo de muestras se calcula en $O(a)$, donde a es la cantidad de atributos. Por último la representación con intervalos (dos ints en lugar de un subconjunto) produce a un código relativamente más simple.

Resultados

Se obtuvo un error muy satisfactorio del 5.65 % en test y 3.5 % en train. Las predicciones se ilustran en la Figura 7. Este bajo error está explicado por la excelente elección de límites entre los bins: para el eje x se utilizaron 4 y para el y 2, que justamente delimitan los rectángulos circunscritos de las dos elipses.

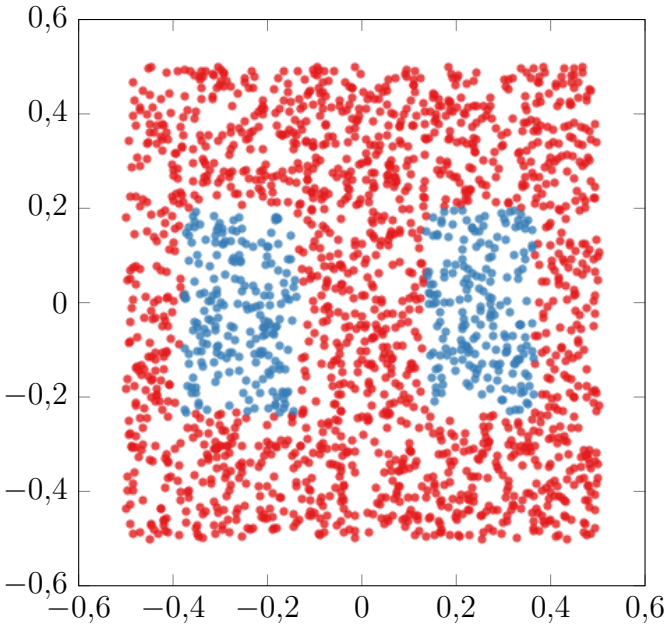


Figura 7: Predicción en test de Naive Bayes con discretización recursiva para el dataset dos_elipses.