Knn y Naive Bayes (clasificaión)

Edmond Géraud

El algoritmo Knn

KNN (K-Nearest Neighbors) es un algoritmo de aprendizaje automático utilizado para clasificación y regresión.

El algoritmo KNN se basa en la idea de que los puntos de datos cercanos en el espacio de características tienen etiquetas similares. En otras palabras, si dos puntos son cercanos en el espacio de características, es más probable que pertenezcan a la misma clase o tengan valores de salida similares.

Fuerzas del algoritmo KNN:

- Fácil de entender e implementar.
- Funciona bien con datos no lineales y no paramétricos.
- El modelo se puede actualizar fácilmente con nuevos datos.
- Es un buen punto de partida para la construcción de modelos más complejos.

Debilidades del algoritmo KNN:

- Requiere una gran cantidad de memoria para almacenar todos los datos de entrenamiento.
- La velocidad de predicción puede ser lenta en grandes conjuntos de datos.
- Sensible a la maldición de la dimensionalidad, lo que significa que se vuelve menos efectivo en altas dimensiones.

Cuándo utilizar KNN:

- Cuando se dispone de una cantidad moderada de datos etiquetados y no etiquetados.
- Cuando los datos tienen una estructura no lineal y no paramétrica.

- Cuando se desea construir un modelo de referencia inicial para comparar con modelos más complejos.

Cuándo no utilizar KNN:

- Cuando se tiene una gran cantidad de datos y/o una alta dimensionalidad.
- Cuando los datos tienen estructuras lineales y paramétricas.
- Cuando se desea un modelo rápido y eficiente para la predicción en tiempo real.

En resumen, el algoritmo KNN es una buena opción para conjuntos de datos más pequeños y no lineales, pero puede ser menos efectivo en grandes conjuntos de datos con alta dimensionalidad. Como con cualquier algoritmo, la elección del algoritmo adecuado depende del problema específico que se está abordando y de las características de los datos disponibles.

Teorema de Bayes y Naive Bayes

El teorema de Bayes es una herramienta matemática fundamental en la teoría de la probabilidad que se utiliza para calcular la probabilidad condicional de un evento, dado que otro evento ya ha ocurrido.

El teorema de Bayes establece que la probabilidad de un evento A dado que ha ocurrido el evento B se puede calcular a través de la siguiente fórmula:

$P(A|B) = P(B|A)P(A)/P(B)$

Donde:

- $P(A|B)$ es la probabilidad de que el evento A ocurra dado que ha ocurrido el evento B.
- $P(B|A)$ es la probabilidad de que el evento B ocurra dado que ha ocurrido el evento A.
- $P(A)$ es la probabilidad del evento A.
- $P(B)$ es la probabilidad del evento B.

El algoritmo Naive Bayes utiliza el teorema de Bayes para clasificar nuevas instancias en función de las probabilidades condicionales de las características en cada clase. Se basa en la suposición de que las características son independientes entre sí y que cada una tiene igual importancia en la clasificación.

Fuerzas del algoritmo Naive Bayes:

- Es un algoritmo rápido y eficiente.
- Puede ser utilizado en conjuntos de datos grandes y de alta dimensionalidad.

- Requiere menos datos de entrenamiento en comparación con otros algoritmos de aprendizaje automático.
- A menudo funciona bien en problemas de clasificación binaria y multiclase.

Debilidades del algoritmo Naive Bayes:

- La suposición de independencia entre las características puede no ser verdadera en todos los casos.
- No es adecuado para problemas de regresión.
- El rendimiento puede ser inferior cuando los datos tienen características numéricas continuas en lugar de categóricas o discretas.

En cuanto a la regresión, el algoritmo Naive Bayes no es adecuado ya que está diseñado para problemas de clasificación

En el contexto de Naive Bayes, el parámetro lambda a menudo se refiere a un parámetro de suavizado utilizado en el cálculo de la probabilidad condicional de una característica dada una clase.

La probabilidad condicional $P(x_i|c)$ se puede calcular como la frecuencia observada de la característica x_i en la clase c, dividida por la frecuencia total de la clase c. Sin embargo, en algunos casos, puede haber características que no se observan en una clase determinada, lo que resulta en una probabilidad condicional igual a cero. Esto a su vez puede conducir a problemas durante la clasificación.

El suavizado de Laplace o suavizado de adición-uno se utiliza a menudo para solucionar este problema. Este suavizado se realiza mediante la adición de un valor constante (usualmente 1) al numerador y al denominador al calcular la probabilidad condicional de una característica. Esto asegura que todas las características tengan al menos una frecuencia observada de 1, incluso si no aparecen en una clase determinada.

Sin embargo, en algunos casos, el suavizado de Laplace puede ser demasiado agresivo y puede resultar en una clasificación incorrecta. Es aquí donde el parámetro lambda entra en juego. Lambda se utiliza para controlar la cantidad de suavizado aplicado en el cálculo de la probabilidad condicional de una característica dada una clase. Cuanto mayor sea el valor de lambda, menor será la cantidad de suavizado aplicado.

Por lo tanto, el parámetro lambda en Naive Bayes se utiliza para controlar el nivel de suavizado aplicado en el cálculo de la probabilidad condicional de una característica dada una clase. Un valor adecuado de lambda puede ayudar a evitar la sobreajuste o subajuste del modelo y mejorar la precisión de la clasificación.

Las curvas ROC (Receiver Operating Characteristic) son una herramienta comúnmente utilizada para evaluar el rendimiento de los modelos de clasificación, como KNN y Naive Bayes, en la tarea de clasificación binaria.

Una curva ROC es una representación gráfica de la tasa de verdaderos positivos (sensibilidad) frente a la tasa de falsos positivos (1-especificidad) para diferentes valores umbral de clasificación. En general, cuanto más cerca esté la curva ROC del extremo superior izquierdo del gráfico, mejor será el rendimiento del modelo.

En el caso de KNN, la curva ROC se puede utilizar para encontrar el valor óptimo del hiperparámetro k, que representa el número de vecinos más cercanos utilizados para la clasificación. El valor óptimo de k es aquel que produce la curva ROC más cercana al extremo superior izquierdo del gráfico.

Por otro lado, en el caso de Naive Bayes, la curva ROC se puede utilizar para ajustar el umbral de clasificación y maximizar el rendimiento del modelo en función del costo de los falsos positivos y falsos negativos. Un ejemplo común es el análisis de diagnóstico médico, donde los falsos positivos y falsos negativos tienen diferentes consecuencias clínicas y pueden requerir diferentes umbrales de clasificación.

En resumen, las curvas ROC son una herramienta útil para evaluar el rendimiento de los modelos de clasificación binaria, incluidos KNN y Naive Bayes. En el caso de KNN, la curva ROC se puede utilizar para encontrar el valor óptimo de k, mientras que en el caso de Naive Bayes, se puede utilizar para ajustar el umbral de clasificación y maximizar el rendimiento del modelo en función del costo de los falsos positivos y falsos negativos.

Práctica

0. Explicad y extended la información anterior del libro Machine Learning with R. Estudiad las fortalezas y deblilidades en una tabla de Markdown. Entended el problema subyacente de cuándo utilizar y cuándo no cada algoritmo.

Datos

Este conjunto de datos corresponde al análisis de varias características útiles para determinar la localización subcelular de proteínas. En concreto, la descripción de los atributos es:

Sequence Name: Accession number for the SWISS-PROT database mcg: McGeoch's method for signal sequence recognition. gvh: von Heijne's method for signal sequence recognition. alm: Score of the ALOM membrane spanning region prediction program. mit: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins. erl: Presence of "HDEL" substring (thought to act as a signal for retention in the endoplasmic reticulum lumen). Binary ttribute. pox: Peroxisomal targeting signal in the C-terminus. vac: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins. nuc: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins. Class Distribution. The class is the localization site. * CYT (cytosolic or cytoskeletal) 463 * NUC (nuclear) 429

* MIT (mitochondrial) 244 * ME3 (membrane protein, no N-terminal signal) 163 * ME2 (membrane protein, uncleaved signal) 51 * ME1 (membrane protein, cleaved signal) 44 * EXC (extracellular) 35 * VAC (vacuolar) 30 * POX (peroxisomal) 20 * ERL (endoplasmic reticulum lumen) 5

Estadística descriptiva

NOTA: En cualquier momento podemos ayudarnos en el R picando `? funcion`

Para cargar paquetes `install.packages(paquete,dependencies=T)`

Para obtener ayuda, el mejor sitio es https://stackoverflow.com/ en inglés

Para crear gráficos en ggplot2 boxplots:

http://www.sthda.com/english/wiki/ggplot2-box-plot-quick-start-guide-r-software-and-data-visualization

Para crear graficos de barras

http://www.sthda.com/english/wiki/ggplot2-barplots-quick-start-guide-r-software-and-data-visualization

Para crear gráficos de densidad

```
# ggplot(datos,aes(y=mivariable,fill=miclase))+geom_density()
```

EXISTE UNA LIBRERÍA LLAMADA `ggstatsplot`

https://indrajeetpatil.github.io/ggstatsplot/

La cual a parte de graficarnos los gráficos que necesitamos, a parte nos realiza las estadísticas básicas que necesitaremos para comprender los resultados en calidad de artículo.

Trabajad con lo que más os acomode

Crear las funciones de normalización

```
###
# X.numericos <- apply(X,2,function(x) mean(x)/(max(x)-min(x)))
```

- $$Y = \mu_x/(max(x) - min(x))$$
- $$Y = median(x)/(max(x) - min(x))$$

- $$Y = (x - min(x))/(max(x) - min(x))$$
- $$Y = x - \mu_x/sd(x)$$
- $$x/\sqrt{(\sum x^2)}$$

1. **Para los datos crudos y cada tipo de normalización haced 25 gráficos, agrupados de 5 en 5. De manera que en cada conjunto de gráficos tengamos cada variable y el tipo de normalización. Podemos agruparlos con `ggarange`**

```
# p1 <- migrafico1 de ggplot2
# p2 <- migrafico2 de ggplot2
# p3 <- migrafico3 de ggplot2
# p4 <- migrafico4 de ggplot2
# p5 <- migrafico5 de ggplot2
#
# ggarange(p1,p2,p3,p4,p5,nrow=3,ncol=2)
```

2. **De los datos en cada tipo de normalización partir los datos en entrenamiento y prueba. Podemos realizar lo siguiente:**

   Fijad la semilla de aleatorizacion en 123456789 en cada chunk de aqui en adelante, cada vez que exista un algortimo de ML, o cuando se entrenan los datos. Todo el alumnado debe tener los mismos resultados.

```
#set.seed(123456789)
# n <- nrow(datos)
# idx <- sample(1:n,size=n*0.7,replace = F)
# entrenamiento <- datos[idx,]
# test <- datos[idx,]
```

3. **NOTA: DE AQUÍ EN ADELANTE, LOS PASOS SON ORIENTATIVOS, RECOMENDARÍA EMPEZAR POR LA REGRESIÓN LOGÍSTICA NORMAL, LA RIDGE, LASSO, KNN Y FINALMENTE NAIVE BAYES.**

   http://www.sthda.com/english/articles/36-classification-methods-essentials/149-penalized-logistic-regression-essentials-in-r-ridge-lasso-and-elastic-net/

   PASOS A TENER EN CUENTA:

1. EN CADA CHUNK QUE HAYA UN ALGORITMO COMO LA PARTICIÓN DE DATOS, O EL ENTRENAMIENTO O EL TESTEO, FIJAD LA SEMILLA 123456789
2. DIVIDIR EL CONJUNTO DE DATOS EN ENTRENAMIENTO 70 X Y TESTEO 30 X
3. ENTRENAR CADA MODELO
4. PREDICCION
5. MEJORA
6. PRUEBA
7. MÉTRICAS

4. **Entrenar el algoritmo de Knn y Naive Bayes con cada tipo de normalización y datos crudos con el paquete `caret`**

   http://topepo.github.io/caret/train-models-by-tag.html#bayesian-model

   http://topepo.github.io/caret/using-your-own-model-in-train.html

   NOTA podemos realizar una función para naive bayes y otra para Knn que nos retorne el objeto modelo. Y de entrada al algoritmo tenga por un lado la clasificación de los datos y por otro lado el tipo de normalización algo por el estilo:

   Naive Bayes

   https://rpubs.com/maulikpatel/224581

   Knn

   https://rpubs.com/njvijay/16444

```
###
# funcion_knn <- function(X_transformados.train,clase.train){
#   ### completar
#   ctrl <- trainControl(method = "repeatedcv", repeats = 3, number = 10)
#  knnFit <- train(clase.train ~ ., data = X_transformados.train, method = "knn",trCo
#   return(knn_fit)
# }
```

   Y podríamos realizar un loop sobre cada tipo de normalizacion

```
# lista.datos  <- list(mean_norm=X1,
#                      min_max=X2,
#                      etc..)
```

```
# lista.resultados <- vector("list",length=numero_de_noramlizaciones)
# for(l in 1:lista.datos){
#   lista.resultados[[l]] <- funcion_knn(lista.datos[[l]],clase.train)
# }
```

5. **Graficad cual es el mejor parámetro de Lambda para Naive Bayes, y el mejor para Knn. En el segundo realizad una curva ROC**

https://stackoverflow.com/questions/34169774/plot-roc-for-multiclass-roc-in-proc-package

https://stackoverflow.com/questions/36631054/roc-curves-for-multiclass-classification-in-r

5. **Realizad lo mismo pero con la regresión logística de Ridge, y Lasso**

   http://www.sthda.com/english/articles/36-classification-methods-essentials/149-penalized-logistic-regression-essentials-in-r-ridge-lasso-and-elastic-net/

6. Al final debemos de tener 3 tablas, con tantas columnas como tipos de normalización tengamos y tantas filas como algoritmos hayamos utilizado. Es decir, necesitaríamos 6 columnas, y 5 filas, por Knn con el mejor número de vecinos, Naive Bayes con el mejor Lambda, logística normal, Ridge y Lasso. La primera tabla indicaríamos el Accuracy, la segunda la sensibilidad del método, y la tercera la especificidad.

7. Finalmente, explicad, cuál es el mejor método, y porqué, que fortaleza le conceden los datos al método, y qué método de normalización es el mejor