Machine Learning de la A a la Z

Traducido al castellano por Juan Gabriel Gomila

Hadelin de Ponteves y Kirill Ermenko

2021-04-10

Machine Learning de la A a la Z

Preguntas y Respuestas del Curso

Traducido por Juan Gabriel Gomila

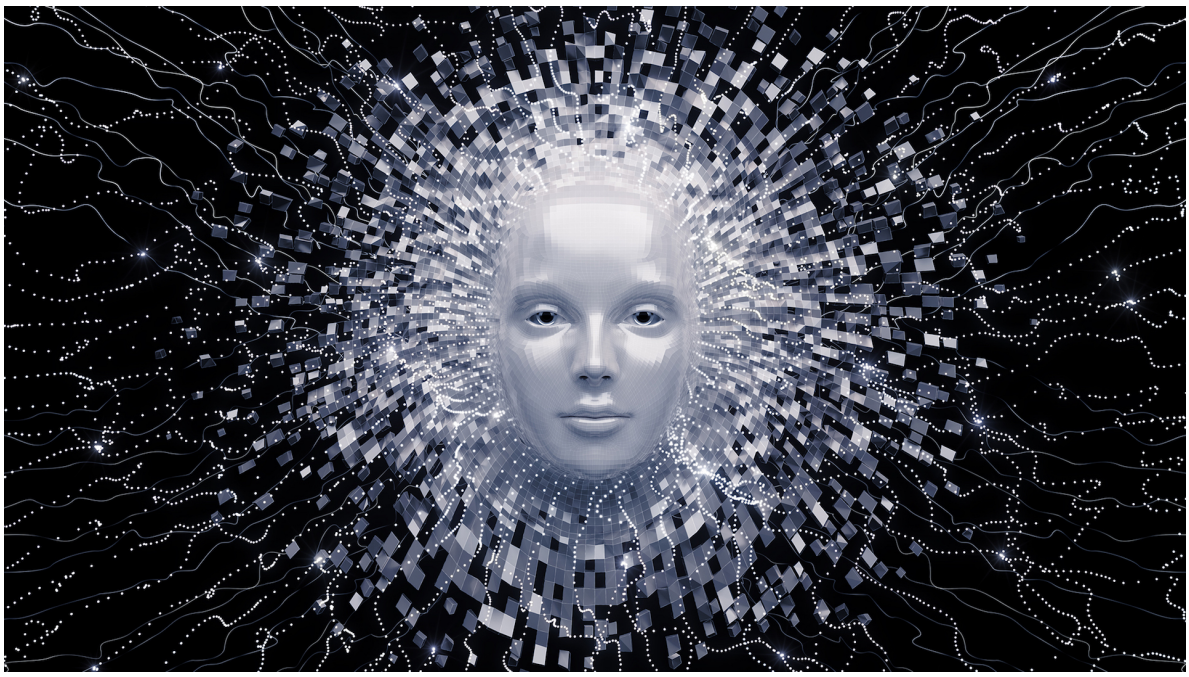
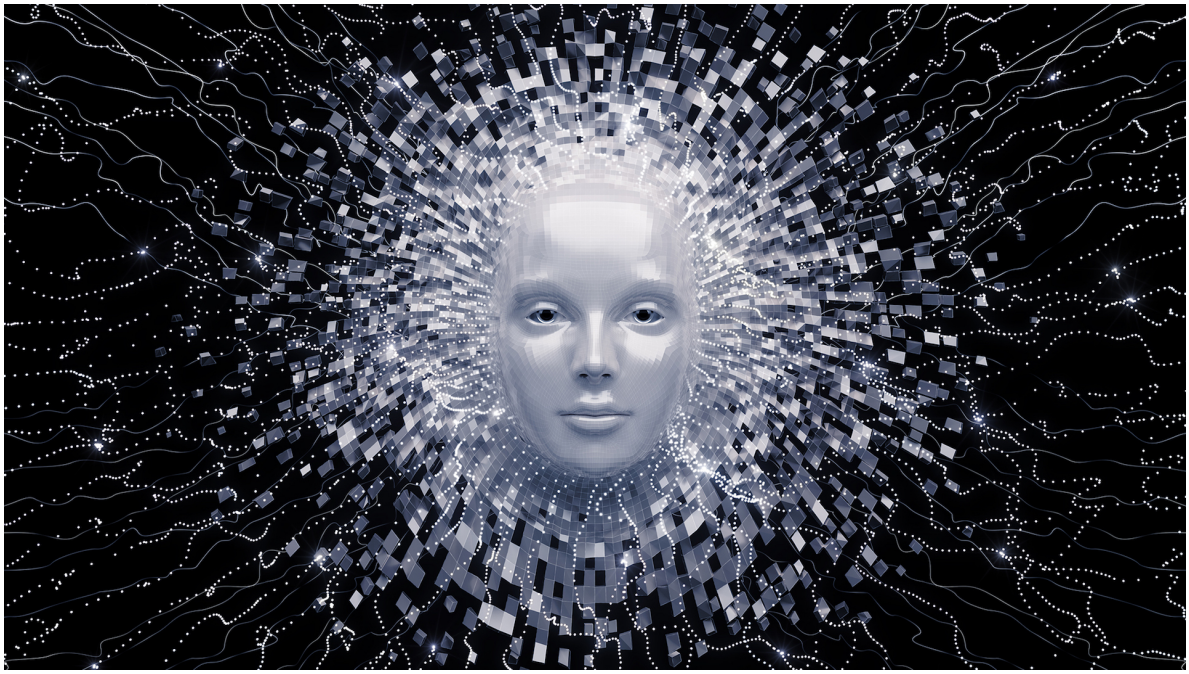


Table of contents

Table of contents

Tabla de Contenidos

Introducción



Este libro es el complemento de nuestro curso online *Machine Learning de A a la Z: R y Python para Data Science* disponible en Udemy. Cubre todas las dudas, recomendaciones y trucos que se derivan del mismo curso, para que las dudas de otros, se conviertan en tus herramientas para defenderte en este campo.

Un aviso antes de empezar, este curso necesita que ya tengas unos conocimientos básicos de matemáticas y de programación, por eso te recomiendo que sigas antes los cursos tal cual y como los recomiendo en el itinerario de aprendizaje de Frogames. En particular, este curso forma parte de las rutas de análisis de datos y de inteligencia artificial, ¡así que échales un ojo!

Este curso es divertido y ameno pero al mismo tiempo todo un reto pues tenemos mucho de Machine Learning por aprender. Lo hemos estructurado del siguiente modo:

- Parte 1 - Preprocesamiento de datos
- Parte 2 - Regresión: Regresión Lineal Simple, Regresión Lineal Múltiple, Regresión Polinomial, SVR, Regresión en Árboles de Decisión y Regresión con Bosques Aleatorios

- Parte 3 - Clasificación: Regresión Logística, K-NN, SVM, Kernel SVM, Naive Bayes, Clasificación con Árboles de Decisión y Clasificación con Bosques Aleatorios
- Parte 4 - Clustering: K-Means, Clustering Jerárquico
- Parte 5 - Aprendizaje por Reglas de Asociación: Apriori, Eclat
- Parte 6 - Reinforcement Learning: Límite de Confianza Superior, Muestreo Thompson
- Parte 7 - Procesamiento Natural del Lenguaje: Modelo de Bag-of-words y algoritmos de NLP
- Parte 8 - Deep Learning: Redes Neuronales Artificiales y Redes Neuronales Convolucionales
- Parte 9 - Reducción de la dimensión: ACP, LDA, Kernel ACP
- Parte 10 - Selección de Modelos & Boosting: k-fold Cross Validation, Ajuste de Parámetros, Grid Search, XGBoost

Además, el curso está relleno de ejercicios prácticos basados en ejemplos de la vida real, de modo que no solo aprenderás teoría, si no también pondrás en práctica tus propios modelos con ejemplos guiados.

Y como bonus, este curso incluye todo el código en Python y R para que lo descargues y uses en tus propios proyectos.

¡Muchas gracias a todos por unirte a nosotros en este curso, te deseo sin duda un emocionante viaje al mundo del machine learning con nosotros!

Capítulo 1

Preprocesado de Datos

1.1. Importar el Dataset

No puedo importar el conjunto de datos. Dice que no se encuentra el archivo. ¿Qué tengo que hacer?

- **Python**: Asegúrate de que en el Explorador de archivos, estés en la carpeta que contiene el archivo `'Data.csv'` que deseas importar. Esta carpeta se denomina `Çarpeta del directorio de trabajo"`.
- **R**: Asegúrate de configurar la carpeta de directorio de trabajo correctamente como lo hacemos en la Lección y que esta carpeta de directorio de trabajo contiene el archivo `'Data.csv'`.

¿Cuál es la diferencia entre las variables independientes y la variable dependiente?

Las variables independientes son los datos de entrada que tenemos, y cada una de ellas la queremos utilizar para predecir algo. Ese algo a predecir es la variable dependiente.

En Python, ¿por qué creamos `X` e `y` por separado?

Porque queremos trabajar con matrices de `Numpy`, en lugar de data frames de `Pandas`. Los arrays de `Numpy` son el formato más conveniente para trabajar al realizar el preprocesamiento de datos y la creación de modelos de aprendizaje automático. Entonces creamos dos matrices separadas, una que contiene nuestras variables independientes (también llamadas características de entrada) y otra que contiene nuestra variable dependiente (lo que queremos predecir).

En Python, ¿qué hace exactamente `'iloc'`?

Localiza la columna por su índice, la posición que esta ocupa. En otras palabras, usar `'iloc'` nos permite tomar columnas simplemente tomando su índice.

En Python, ¿qué hace exactamente `'.values'`?

Devuelve los valores de las columnas que estamos tomando (por su índice) dentro de una matriz de `Numpy`. Así es básicamente como `X` e `y`se convierten en matrices de `Numpy`.

En R, ¿por qué no tenemos que crear matrices o arrays?

Porque R funciona de manera muy diferente a Python. R contiene excelentes herramientas que permiten trabajar directa y fácilmente con data frames.

1.2. Datos Faltantes y NAs

En Python, ¿cuál es la diferencia entre `fit` y `transform`?

La parte de `fit` se utiliza para extraer información de los datos sobre los que se aplica el objeto (aquí, `Imputer` detectará los valores faltantes y obtendrá la media de la columna). Luego, la parte de `transform` se usa para aplicar alguna transformación (aquí, `Imputer` reemplazará el valor faltante por la media).

En R, ¿por qué usamos la función `ave()` cuando reemplazamos los valores faltantes por una media de la columna cuando podemos hacerlo mucho más simple de esta manera?

```
dataset$Age[is.na(dataset$Age)] = mean(dataset$Age, na.rm = T)
```

Usamos la función `ave()` con el parámetro `FUN` porque nos permite agregar algunas opciones en el cálculo de la media, como por ejemplo calcular la media de observaciones agrupadas por otra característica, o calcular la media de subconjuntos, etc. Esta función puede ser muy útil si deseamos ser más precisos al reemplazar los valores faltantes por la media.

¿Reemplazar por la media es la mejor estrategia para manejar los valores faltantes o NAs?

Es una buena estrategia pero no siempre la mejor. Depende de nuestro problema comercial, de la forma en que se distribuyen sus datos y del número de valores faltantes. Si, por ejemplo, tenemos muchos valores faltantes, la sustitución de medias no es lo mejor. Otras estrategias incluyen la imputación por la "mediana", la imputación por el valor "más frecuente" o la imputación de predicción. La imputación de predicciones es en realidad otra gran estrategia que se recomienda, pero que no cubrimos en la Parte 1 porque es demasiado avanzada para hacerlo en la Parte 1. Esta estrategia de hecho requiere comprender la Parte 3: Clasificación. Entonces, si completas la Parte 3, esta es la estrategia que es incluso mejor que la imputación media: tomamos la columna de características que contiene los valores faltantes y establecemos esta columna de características como la variable dependiente, mientras establecemos las otras columnas como las variables independientes. Luego, dividimos nuestro conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba donde el conjunto de entrenamiento contiene todas las observaciones (las filas) donde la columna de características que acabamos de establecer como la variable dependiente no tiene ningún valor faltante y el conjunto de prueba contiene todas las observaciones donde la columna de la variable dependiente contiene los valores faltantes. Luego, realizamos un modelo de clasificación (uno bueno para esta situación es k-NN) para predecir los valores faltantes en el conjunto de prueba. Y finalmente reemplazamos los valores faltantes del conjunto de prueba por las predicciones. ¡Una gran estrategia!

1.3. Datos Categóricos

En Python, ¿qué hacen los dos métodos `'fit_transform'`?

Cuando se llama al método `'fit_transform()'` desde la clase `LabelEncoder()`, transforma las categorías de formato string a formato de número entero Por ejemplo, transforma Francia, España y Alemania

en 0, 1 y 2. Luego, cuando se llama al método `'fit_transform()'` desde la clase `OneHotEncoder()`, crea columnas separadas para cada etiqueta diferente con valores binarios 0 y 1. Esas columnas separadas son las variables ficticias o variables dummy.

En R, ¿por qué no creamos manualmente las variables ficticias como lo hacemos en Python?

Porque se crean automáticamente cuando se usa la función `'factor()'` como lo hacemos en la Lección. Lo veremos visualmente al iniciar los problemas de regresión y la clasificación.

1.4. Dividir el conjunto de datos en el conjunto de entrenamiento y el conjunto de prueba

¿Cuál es la diferencia entre el conjunto de entrenamiento y el conjunto de prueba?

El conjunto de entrenamiento es un subconjunto de nuestros datos en el que su modelo aprenderá a predecir la variable dependiente a partir de las variables independientes. El conjunto de prueba es el subconjunto complementario del conjunto de entrenamiento, en el que evaluará nuestro modelo para ver si logra predecir correctamente la variable dependiente utilizando para ello las variables independientes.

¿Por qué dividimos el dataset utilizando el valor de la variable dependiente?

Porque queremos tener valores bien distribuidos de la variable dependiente en el conjunto de entrenamiento y prueba. Por ejemplo, si solo tuviéramos los mismos valores de la variable dependiente en el conjunto de entrenamiento que en el de prueba, nuestro modelo no podría aprender ninguna correlación entre las variables independientes y dependientes.

1.5. Escalado de Características

¿Realmente tenemos que aplicar Escalado de Características en las variables dummy?

Sí, si deseas optimizar la precisión de las predicciones de tu modelo. No, si deseas mantener la mayor interpretación posible en tu modelo.

¿Cuándo deberíamos utilizar la estandarización y la normalización?

Por lo general, debemos normalizar (normalización) cuando los datos se distribuyen normalmente, segun una campana de Gauss, y escalar (estandarizar) cuando los datos no se distribuyen normalmente. En caso de duda, debemos optar por la estandarización. Sin embargo, lo que se suele hacer es probar los dos métodos de escalado y evaluar así cual de los dos nos da el mejor resultado.

Capítulo 2

Regressión

2.1. Regresión Lineal Simple

2.1.1. Intuición de la Regresión Lineal Simple

¿Qué significanson exactamente los coeficientes b_0 y b_1 en la ecuación de regresión lineal simple?

$$\text{Sueldo} = b_0 + b_1 \times \text{Experiencia}$$

b_0 es el sueldo que recibe sin experiencia alguna y b_1 es el aumento de salario por año de experiencia adicional.

¿Por qué tomamos las diferencias al cuadrado y simplemente no las diferencias en valor absoluto?

Porque las diferencias al cuadrado facilitan la deducción de la ecuación de una recta de regresión a través de sus derivadas. De hecho, para encontrar esa recta, necesitamos calcular la primera derivada de la función de error de pérdida, y es mucho más difícil calcular la derivada de valores absolutos que los valores al cuadrado.

2.1.2. Regresión Lineal Simple en Python

¿Por qué no aplicamos el escalado de características en nuestro modelo de regresión lineal simple?

Es simplemente porque dado que la variable y es una combinación lineal de las variables independientes, los coeficientes pueden adaptar su escala para poner todo en la misma escala. Por ejemplo, si tenemos dos variables independientes x_1 y x_2 y si y toma valores entre 0 y 1, x_1 toma valores entre 1 y 10 y x_2 toma valores entre 10 y 100, entonces b_1 se puede multiplicar por 0.1 y b_2 se puede multiplicar por 0.01 para que y, b_1x_1 y b_2x_2 estén todos en la misma escala.

¿Qué hace exactamente la instrucción `regressor.fit(X_train, y_train)`?

El método `fit` tomará los valores de X_train y y_train y luego calculará los coeficientes b_0 y b_1 de la ecuación de regresión lineal simple ($y = b_0 + b_1 x$) como se ve en la clase de intuición sobre el método en el curso. Ese es todo el propósito de este método `fit`.

2.1.3. Regresión lineal simple en `R`

¿Qué es el p-valor?

Para comprender el p-valor, debemos comenzar por comprender la hipótesis nula: la hipótesis nula es la suposición de que los parámetros asociados a nuestras variables independientes son iguales a cero. Por lo tanto, bajo esta hipótesis, nuestras observaciones son totalmente aleatorias y no siguen un patrón determinado. El p-valor es la probabilidad de que los parámetros asociados a nuestras variables independientes tengan ciertos valores distintos de cero, suponiedo que la hipótesis nula es verdadera. Lo más importante a tener en cuenta sobre el p-valor es que es una métrica estadística: cuanto menor es el p-valor, más significativa es estadísticamente una variable independiente, que quiere decir que será el mejor predictor.

En la última clase de `R` vemos un área gris alrededor de la línea azul. ¿Qué significa y cómo podemos obtenerla?

El área gris alrededor de la recta de regreesión es el intervalo de confianza. Puede agregarse con la instrucción `+ geom_smooth(method ='lm')` justo después de `geom_line(...)` para obtener estos límites de confianza.

¿Cómo obtenemos los p-valores en Python como lo hacemos en R?

Aprenderemos cómo hacerlo en la siguiente sección sobre Regresión lineal múltiple.

2.2. Regresión lineal múltiple

2.2.1. Intuición de la regresión lineal múltiple

¿Cuáles son las hipótsis que debe verificarse en la regresión lineal múltiple?

Como vemos en detalle en el curso de estaddística descriptiva en Udemy deben cumplirse:

- Linealidad: debe existir una relación lineal entre la variable dependiente y las variables independientes. Los diagramas de dispersión pueden mostrar si existe una relación lineal o curvilínea.
- Homoscedasticidad: esta suposición establece que la varianza de los términos de error es similar entre los valores de las variables independientes. Un gráfico de residuos estandarizados versus valores predichos puede mostrar si los puntos están distribuidos por igual en todos los valores de las variables independientes.
- Normalidad multivariante: La regresión lineal múltiple asume que los residuos (las diferencias entre el valor observado de la variable dependiente y y el valor predicho $\hat{y}$ se distribuyen normalmente.
- Independencia de errores: la regresión lineal múltiple asume que los residuos (las diferencias entre el valor observado de la variable dependiente y y el valor predicho $\hat{y}$ son independientes.

- Falta de multicolinealidad: la regresión lineal múltiple asume que las variables independientes no están altamente correlacionadas entre sí. Esta suposición se prueba utilizando valores del factor de inflación de varianza (VIF).

¿Cómo se relaciona el coeficiente b_0 con la trampa de las variables ficticias?

Como $D_2 = 1 - D_1$ entonces si calculamos tanto D_1 como D_2 obtenmos:

$$\begin{aligned} y &= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4D_1 + b_5D_2 \\ &= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4D_1 + b_5(1 - D_1) \\ &= b_0 + b_5 + b_1x_1 + b_2x_2 + b_3x_3 + (b_4 - b_5)D_1 \\ &= b_0^* + b_1x_1 + b_2x_2 + b_3x_3 + b_4^*D_1 \end{aligned}$$

con $b_0^* = b_0 + b_5$ y $b_4^* = b_4 - b_5$

Por lo tanto, la información de la variable ficticia redundante D_2 entra a formar parte de la constante b_0.

2.2.2. Regresión lineal múltiple en Python

Normalmente predecimos los resultados de un conjunto de observaciones (el conjunto de testing). ¿Cómo hacemos lo mismo cuando solamente tenemos una observación?

Digamos que esta observación tiene las siguientes características: `Feature Value 1`, `Feature Value 2`, … y `Feature Value m` (m variables independientes). Entonces, el código para obtener el resultado previsto sería el siguiente:

```
y_pred = regressor.predict(np.array([[Feature Value 1, ..., Feature Value m]]))
```

¿Cómo implementar la eliminación automática hacia atrás en Python?

Se puede implementar utilizando el siguiente código:

```
import statsmodels.formula.api as sm
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
    regressor_OLS.summary()
    return x
SL = 0.05
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
X_Modeled = backwardElimination(X_opt, SL)
```

2.2.3. Regresión lineal múltiple en R

Hemos predicho los resultados de un conjunto de observaciones (el conjunto de testing). ¿Cómo hacemos lo mismo cuando solo tenemos una observación?

Primero debemos crear una nueva variable `single_observation` con los valores de entrada y configurar esta variable como un data frame:

```
single_observation = data.frame(R.D.Spend = 10000,
                                Administration = 20000,
                                Marketing.Spend = 30000,
                                State = 1)
```

Luego, simplemente podemos predecir el resultado de esta única observación exactamente como lo hicimos con el conjunto de teesting, simplemente reemplazando `test_set` por `single_observation`:

```
y_pred = predict(regressor, newdata = single_observation)
```

¿Cómo implementar la eliminación automática hacia atrás en R?

Se puede implementar usando el siguiente código:

```
backwardElimination <- function(x, sl) {
    numVars = length(x)
    for (i in c(1:numVars) ){
      regressor = lm(formula = Profit ~ ., data = x )
      maxVar = max(coef(summary(regressor))[c(2:numVars), "Pr(>|t|)"])
      if (maxVar > sl){
        j = which(coef(summary(regressor))[c(2:numVars),"Pr(>|t|)"]==maxVar)
        x = x[,-j]
      }
      numVars = numVars - 1
    }
    return(summary(regressor))
  }

  SL = 0.05
  dataset = dataset[,c(1,2,3,4,5)]
  backwardElimination(dataset, SL)
```

2.3. Regresión Polinómica

2.3.1. Intuición sobre la Regresión Polinómica

¿La regresión polinómica es un modelo lineal o no lineal?

Eso depende de a qué te refieras. La regresión polinomial es lineal en los coeficientes ya que no tenemos ninguna potencia de los coeficientes (todos los coeficientes se elevan a la potencia de 1: b_0, b_1,..., b_n). Sin embargo, la regresión polinomial es una función no lineal de la entrada x, ya que tenemos las entradas elevadas a varias potencias: x (potencia 1), x^2 (potencia 2),..., x^n (potencia n). Así es como

también podemos ver la Regresión polinomial como un modelo no lineal. Además, de hecho, la regresión polinomial es apropiada cuando los datos no están distribuidos linealmente (lo que significa que no se puede ajustar una línea recta entre y y x).

2.3.2. Regresión Polinomial en Python

¿Por qué no hemos aplicado escalado de características en nuestro modeleo de regresión polinomial?

Es simplemente porque, dado que y es una combinación lineal de x y x^2, los coeficientes pueden adaptar su escala para poner todo en la misma escala. Por ejemplo, si y toma valores entre 0 y 1, x toma valores entre 1 y 10 y x^2 toma valores entre 1 y 100, entonces b_1 puede multiplicarse por 0.1 y b_2 puede ser multiplicado por 0.01 para que y, b_1x_1 y b_2x_2 estén todos en la misma escala.

¿Cómo elegimos el mejor grado para la regresión?

La forma principal de encontrar un buen ajuste es representar gráficamente el modelo y ver cómo se ve visualmente. Simplemente prueba varios grados y verás cuál te da el mejor ajuste. La otra opción es encontrar la raíz del error cuadrático medio (RMSE) más bajo para tu modelo, pero en ese caso ten cuidado de no sobreajustar los datos.

¿Por qué hemos tenido que crear un segundo modelo de regresión lineal `lin_reg_2`? ¿No podríamos haber usado `lin_reg` directamente?

No, porque `lin_reg` ya está ajustado a X e y y ahora queremos ajustar un nuevo modelo lineal a X_{poly} e y. Entonces tenemos que crear un nuevo objeto regresor. Es importante que el método de ajuste aquí encuentre el coeficiente entre las variables independientes y la variable dependiente. Por lo tanto, dado que `lin_reg` ya obtuvo los coeficientes de correlación entre X y y, se debe crear `lin_reg_2` para obtener algunos nuevos coeficientes de correlación entre X_{poly} e y.

2.3.3. Regresión polinomial en R

En R, ¿tengo que crear manualmente las diferentes columnas para cada característica polinomial? ¿Qué pasa si hay muchas características polinomiales?

Rara vez tendrás que crear más de 4 características polinomiales, de lo contrario se sobreajustará el modelo, lo cual debe evitarse absolutamente en Machine Learning. Entonces, incluso si tienes que crearlos manualmente, nunca tomará demasiado tiempo. Además puedes usar la plantilla que te damos en el curso.

¿Cómo encontramos el mejor grado de la regresión polinomial? (Haciendo esta pregunta nuevamente en caso de que algunos estudiantes solo hagan R)

La forma principal de encontrar un buen ajuste es representar gráficamente el modelo y ver cómo se ve visualmente. Simplemente prueba varios grados y verás cuál te da el mejor ajuste. La otra opción es encontrar la raíz del error cuadrático medio (RMSE) más bajo para tu modelo, pero en ese caso ten cuidado de no sobreajustar los datos.

2.4. Support Vector Regression

2.4.1. Intuición de SVR

¿Cuando debemos usar SVR?

Debemos usar SVR si un modelo lineal como la regresión lineal no se ajusta muy bien a los datos. Esto significaría que estamos lidiando con un problema no lineal, donde sus datos no están distribuidos linealmente. Por lo tanto, en ese caso, SVR podría ser una solución mucho mejor.

No he entendido la clase de intuición del algoritmo. ¿Es un problema para seguir el curso?

De ningún modo. SVR es un modelo bastante abstracto y, además, no es de uso común. Lo que tienes que comprender es el modelo SVM, que verás más adelante en la Parte 3 - Clasificación. Luego, una vez que comprendas el modelo SVM, obtendrás una mejor comprensión del modelo SVR, ya que SVR es simplemente el SVM para Regresión. Sin embargo, queríamos incluir SVR en este curso para brindarte una opción adicional en tu kit de herramientas de aprendizaje automático.

2.4.2. SVR en Python

¿Para qué necesitamos hacer `sc_Y.inverse_transform`?

Necesitamos el método `inverse_transform` para volver a la escala original. De hecho, hemos aplicado el escalado de variables, por lo que obtenemos para cada variable, una escala de valores alrededor de 0 y si hacemos una predicción sin invertir la escala, obtendremos el salario previsto escalado. Y, por supuesto, queremos el salario real, no el escalado, por lo que tenemos que usar `sc_Y.inverse_transform`. También lo que es importante entender es que `transform` e `inverse_transform` son métodos que van emparejados de la mano.

2.4.3. SVR en R

¿Por qué no aplicamos el escalado de características como lo hicimos explícitamente en Python?

Eso es porque en la función `svm()` de R, los valores se escalan automáticamente.

¿Podemos seleccionar las variables más significativas gracias al p-valor como hicimos antes en R?

No se puede utilizar el p-valor porque SVR no es un modelo lineal y los p-valores se aplican solo a los modelos lineales. Por lo tanto, la selección de características está fuera de discusión. Pero puedes realizar la extracción de características, que verá en la Parte 9: Reducción de dimensionalidad. Eso se puede aplicar a los SVR y reducirá el número de características

2.5. Árboles de Decisión para la Regresión

2.5.1. Intuición sobre Árboles de Decisión para la Regresión

¿Cómo divide el algoritmo los puntos de datos?

Utiliza la reducción de la desviación estándar de las predicciones. En otras palabras, la desviación estándar se reduce inmediatamente después de una división. Por lo tanto, la construcción de un árbol de decisión se trata de encontrar el atributo que devuelve la reducción de la desviación estándar más alta (es decir, las ramas más homogéneas).

¿Qué es la ganancia de información y cómo funciona en los árboles de decisión?

La ganancia de información en la regresión del árbol de decisión es exactamente la reducción de la desviación estándar que buscamos alcanzar. Calculamos cuánto disminuye la desviación estándar después de cada división. Debido a que cuanto más se reduce la desviación estándar después de una división, más homogéneos serán los nodos secundarios.

¿Qué es la entropía y cómo funciona en los árboles de decisión

La Entropía mide el desorden en un conjunto, aquí en una parte resultante de una división. Entonces, cuanto más homogéneos sean los datos en una parte, menor será la entropía. Cuantas más divisiones tengamos, más posibilidades tendremos de encontrar partes en las que los datos sean homogéneos y, por lo tanto, menor será la entropía (cercana a 0) en estas partes. Sin embargo, es posible que todavía econtremos algunos nodos donde los datos no sean homogéneos y, por lo tanto, la entropía no sería necesariamente tan pequeña.

2.5.2. Árboles de Decisión para Regresión en Python

¿Tiene mucho sentido un árbol de decisiones en 1D?

En realidad no, como vimos en la parte práctica de esta sección. En 1D (es decir, una variable independiente), el árbol de decisiones tiende claramente a sobreajustarse a los datos. El árbol de decisiones sería mucho más relevante en una dimensión superior, pero hay que tener en cuenta que la implementación que hicimos aquí en 1D sería exactamente la misma en una dimensión superior. Por lo tanto, es posible que desees mantener este modelo en tu kit de herramientas en caso de que estés tratando con un espacio de mayor dimensión. Este será realmente el caso en la Parte 3 - Clasificación, donde usaremos el Árbol de decisión para la clasificación en 2D, que veremos que resulta ser más relevante.

2.5.3. Árboles de Decisión para Regresión en R

¿Por qué obtenemos resultados diferentes entre Python y R?

Es probable que la diferencia se deba a la división aleatoria de datos. Si hiciéramos una validación cruzada (consulta la Parte 10) en todos los modelos en ambos lenguajes de programación, probablemente obtendríamos una precisión media similar. Dicho esto, recomendaríamos más el uso de Python para árboles de decisión, ya que el modelo está un poco mejor implementado en Python.

¿Es apropiado el árbol de decisiones aquí?

Aquí, en este ejemplo, podemos ver claramente que la curva de ajuste es una escalera con grandes huecos en las discontinuidades. Por lo tanto, ese modelo de regresión de árbol de decisión no es el más apropiado, y eso se debe a que solo tenemos una variable independiente que toma valores discretos. Entonces, lo que sucedió es que la predicción se hizo en la parte inferior de la brecha en Python, y se hizo en la parte superior de la brecha en R. Y como la brecha es grande, eso hace una gran diferencia.

Si tuviéramos muchas más observaciones, tomando valores con más continuidad (como con un paso de 0.1), las brechas serían menores y por lo tanto las predicciones en Python y R muy alejadas entre sí.

¿Podemos seleccionar las variables más significativas gracias al valor p como hicimos antes en R?

No se puede utilizar el valor p porque el árbol de decisión no es un modelo lineal y los valores p se aplican solo a los modelos lineales. Por lo tanto, la selección de características está fuera de discusión. Pero puedes realizar la extracción de características, que verá en la Parte 9: Reducción de dimensionalidad. Eso se puede aplicar a los árboles de decisión y reducirá el número de características.

2.6. Regresión con Bosques Aleatorios

2.6.1. Intuición sobre la Regresión con Bosques Aleatorios

¿Cuál es la ventaja y el inconveniente de los bosques aleatorios en comparación con los árboles de decisión?

- Ventaja: los bosques aleatorios pueden brindarte un mayor poder de predicción que los árboles de decisión.
- Inconveniente: el árbol de decisión te dará más interpretabilidad que los bosques aleatorios, porque puedes represetar el gráfico de un árbol de decisión para ver las diferentes divisiones que conducen a la predicción, como se ve en la clase de intuición. Esto es algo que no puedes hacer con Random Forest.

¿Cuándo usar Random Forest y cuándo usar los otros modelos

La mejor respuesta a esta pregunta es: ¡pruébalos todos!

De hecho, gracias a las plantillas del curso, solo te llevará 10 minutos probar todos los modelos, lo que es muy poco en comparación con el tiempo dedicado a las otras partes de un proyecto de ciencia de datos (como el preprocesamiento de datos, por ejemplo). Así que no tengas miedo de probar todos los modelos de regresión y comparar los resultados (a través de la validación cruzada que veremos en la Parte 10). Esto quiere decir que te hemos dado el máximo número de modelos en este curso para que los tengas en tu kit de herramientas y aumentes tus posibilidades de obtener mejores resultados.

Sin embargo, si deseas algunos atajos, aquí hay algunas reglas generales que te ayudarán a decidir qué modelo usar:

- Primero, debes averiguar si tu problema es lineal o no lineal. Aprenderá cómo hacerlo en la Parte 10: Selección de modelos. Entonces:
  - Si tu problema es lineal, debes optar por Regresión lineal simple si solo tienes una característica, y Regresión lineal múltiple si tienes varias características.
  - Si tu problema no es lineal, debes optar por Regresión polinomial, SVR, Árbol de decisión o Bosque aleatorio. Entonces, ¿cuál debemos elegir entre estos cuatro? Esto lo aprenderás en la Parte 10: Selección de modelos. El método consiste en utilizar una técnica muy relevante que evalúa el rendimiento de tus modelos, llamada k-Fold Cross Validation, y luego seleccionar el modelo que muestra los mejores resultados. Siéntete libre de saltar directamente a la Parte 10 si ya quieres aprender cómo hacerlo ahora mismo.

2.6.2. Regresión con Bosques Aleatorios en Python

¿Cómo sé cuántos árboles debo usar?

Primero, recomendaría elegir el número de árboles experimentando. Por lo general, toma menos tiempo del que pensamos encontrar el mejor valor ajustando y ajustando el modelo manualmente. En realidad, eso es lo que hacemos en general cuando construimos un modelo de Machine Learning: lo hacemos en varias tomas, experimentando varios valores de hiperparámetros como el número de árboles. Sin embargo, también debes saber que en la Parte 10 cubriremos la Validación cruzada en k-Folds y la Búsqueda de cuadrícula, que son técnicas poderosas que puede usar para encontrar el valor óptimo de un hiperparámetro, como aquí el número de árboles.

2.6.3. Regresión con Bosques Aleatorios en R

¿Cómo decidimos cuántos árboles serían suficientes para obtener un resultado relativamente preciso?

Al igual que en Python, puedes encontrar un número relevante de árboles a través de la experimentación y el ajuste manual. Debes usar suficientes árboles para obtener una buena precisión, pero no debes usar demasiados árboles porque eso podría causar un ajuste excesivo. Aprenderás a encontrar el número óptimo de árboles en la primera sección de la Parte 10: Selección del modelo. Se realiza con una técnica llamada Ajuste de parámetros (búsqueda por cuadrícula con validación cruzada de k-Fold).

¿Por qué obtenemos resultados diferentes entre Python y R?

Es probable que la diferencia se deba a la división aleatoria de datos. Si hiciéramos una validación cruzada (consulta la Parte 10) en todos los modelos en ambos idiomas, probablemente obtendríamos una precisión media similar.

¿Podemos seleccionar las variables más significativas gracias al p-valor como hicimos antes en R?

No se puede usar el p-valor porque los bosques aleatorios no son modelos lineales y los valores p se aplican solo a los modelos lineales. Por lo tanto, la selección de características está fuera de discusión. Pero puedes realizar la extracción de características, que verá en la Parte 9: Reducción de dimensionalidad. Eso se puede aplicar a Random Forests y reducirá el número de características.

2.7. Evaluar la eficacia de los modelos de regresión

Entiendo que para evaluar múltiples modelos lineales puedo usar el R-cuadrado ajustado o la matriz de Pearson. Pero, ¿cómo puedo evaluar los modelos de regresión polinomial o los modelos de regresión con bosques aleatorios que no son lineales?

Se pueden evaluar modelos de regresión polinomial y modelos de regresión con bosques aleatorios, calculando el Error cuadrático medio (la media de los errores al cuadrado). Se puede calcular esta métrica fácilmente con una función de suma o usando un bucle for, calculando la suma de las diferencias al cuadrado entre los resultados predichos y los resultados reales, sobre todas las observaciones del conjunto de prueba.

Realmente no se podría aplicar la eliminación hacia atrás a los modelos de regresión polinomial y regresión aleatoria de bosque porque estos modelos no tienen coeficientes combinados en una ecuación de regresión lineal y, por lo tanto, no tienen p-valores p.

Sin embargo, en la Parte 9 - Reducción de dimensionalidad, veremos algunas técnicas como Eliminación hacia atrás, para reducir el número de características, basadas en Selección de características y Extracción de características.

¿Qué son los sesgos o varianzas bajos / altos en el aprendizaje automático?

- Sesgo bajo es cuando las predicciones de tu modelo están muy cerca de los valores reales.
- Sesgo alto es cuando las predicciones de tu modelo están lejos de los valores reales.
- Varianza baja: cuando ejecutamos el modelo varias veces, las diferentes predicciones de los puntos de observación no varian mucho.
- Varianza alta: cuando ejecutamos el modelo varias veces, las diferentes predicciones de los puntos de observación varian mucho.

Lo que deseamos obtener cuando creamos un modelo es: Bajo Sesgo y Baja Varianza.

En Python, ¿cómo implementar la eliminación hacia atrás automatizada con R-Cuadrado ajustado?

Retomemos el problema de la regresión lineal múltiple, con 5 variables independientes. La eliminación automática hacia atrás, incluido el R cuadrado ajustado, se puede implementar de esta manera:

```
import statsmodels.formula.api as sm
def backwardElimination(x, SL):
    numVars = len(x[0])
    temp = np.zeros((50,6)).astype(int)
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        adjR_before = regressor_OLS.rsquared_adj.astype(float)
        if maxVar > SL:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    temp[:,j] = x[:, j]
                    x = np.delete(x, j, 1)
                    tmp_regressor = sm.OLS(y, x).fit()
                    adjR_after = tmp_regressor.rsquared_adj.astype(float)
                    if (adjR_before >= adjR_after):
                        x_rollback = np.hstack((x, temp[:,[0,j]]))
                        x_rollback = np.delete(x_rollback, j, 1)
                        print (regressor_OLS.summary())
                        return x_rollback
                    else:
                        continue
    regressor_OLS.summary()
    return x

SL = 0.05
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
X_Modeled = backwardElimination(X_opt, SL)
```

¿Cómo obtener el R-cuadrado ajustado en R?

Denotemos la variable dependiente por DV y las variables independientes por IV_1, IV_2, etc. Entonces,

el código R para obtener el R-Cuadrado ajustado es el siguiente:

```
summary(lm(DV ~ IV1 + IV2 + ..., dataset))$adj.r.squared
```

Capítulo 3

Clasificación

3.1. Regresión Logística

3.1.1. Intuición de la Regresión Logística

¿Es la regresión logística un modelo lineal o no lineal?

Es un modelo lineal. Visualizarás esto al final de la sección al ver que el separador del clasificador es una línea recta.

¿Cuales son las hipótesis del modelo de regresión logística ?

Primero, la regresión logística binaria requiere que la variable dependiente sea binaria y la regresión logística ordinal requiere que la variable dependiente sea ordinal.

En segundo lugar, la regresión logística requiere que las observaciones sean independientes entre sí. En otras palabras, las observaciones no deben provenir de medidas repetidas o datos coincidentes.

En tercer lugar, la regresión logística requiere que haya poca o ninguna multicolinealidad entre las variables independientes. Esto significa que las variables independientes no deben estar muy correlacionadas entre sí.

Cuarto, la regresión logística asume la linealidad de las variables independientes y las probabilidades logarítmicas. aunque este análisis no requiere que las variables dependientes e independientes estén relacionadas linealmente, requiere que las variables independientes estén relacionadas linealmente con las probabilidades logarítmicas.

¿Se puede utilizar también la regresión logística para muchas variables independientes?

Sí, la regresión logística se puede utilizar para tantas variables independientes como se desee. Sin embargo, hay que tener en cuenta que no podrás visualizar los resultados en más de 3 dimensiones.

3.1.2. Regresión Logística en Python

¿Qué hace el método `fit`?

El método `fit` básicamente entrena el modelo de regresión logística utilizando los datos de entrenamiento. Por lo tanto, calculará y obtendrá los pesos (coeficientes) del modelo de regresión logística (ver la lección de intuición) para ese conjunto particular de datos de entrenamiento compuesto por `X_train` y `y_train`. Luego, justo después de recopilar los pesos / coeficientes, tendremos un modelo de regresión logística completamente entrenado en los datos de entrenamiento y listo para predecir nuevos resultados gracias al método de predicción.

Hemos predicho los resultados de un conjunto de observaciones (el conjunto de prueba). ¿Cómo hacemos lo mismo para una sola observación, para predecir un solo resultado?

Digamos que esta observación tiene las siguientes características: Edad = 30, Salario estimado = 50000. Entonces, el código para obtener el resultado predicho sería el siguiente (observemos cómo no debemos olvidar escalar esa única observación primero):

```
y_pred = classifier.predict(sc_X.transform(np.array([[20, 50000]])))
```

¿Es la Matriz de confusión la forma óptima de evaluar el desempeño del modelo?

No, solo nos da una idea de qué tan bien puede funcionar nuestro modelo. Si obtenemos una buena matriz de confusión con pocos errores de predicción en el conjunto de prueba, existe la posibilidad de que el modelo tenga un buen poder predictivo. Sin embargo, la forma más relevante de evaluar el modelo es a través de la Validación cruzada de K-Fold, que veremos en la Parte 10. Consiste en evaluar nuestro modelo en varios conjuntos de prueba (llamados conjuntos de validación), para que podamos asegurarnos de que no tengamos suerte en un solo conjunto de prueba. Hoy en día, la mayoría de los científicos de datos o desarrolladores de inteligencia artificial evalúan su modelo a través de la validación cruzada de K-Fold. Sin embargo, la técnica es un tema diferente, así que hemos preferido dejarlo para la Parte 10 después de que cubramos todos los diferentes modelos.

¿Cómo volver a transformar la edad y el salario a su escala original?

Podemos volver a transformar Edad y Salario utilizando el método `inverse_transform()`. Para ello, tomamos nuestro objeto escalador `sc` y aplicamos el método `inverse_transform()` de esta manera:

```
X_train = sc.inverse_transform(X_train)
X_test = sc.inverse_transform(X_test)
```

¿Cómo hacer la misma clasificación cuando las variables dependientes tienen más de dos categorías?

Expliquemos cómo hacerlo con tres clases.

Primero, sigue siendo el mismo código para hacer las predicciones. Además, aunque se basa en el método Uno contra Todos, ni siquiera necesitamos crear variables ficticias de nuestra variable dependiente.

En cuanto a las variables independientes, solo necesitamos aplicar `LabelEncoder` seguido de `OneHotEncoder`. Crearemos nuestras variables ficticias (dummies), independientemente del número de categorías que tengan nuestras variables categóricas.

Y eventualmente, simplemente necesitaremos agregar un color de esta manera:

```
# Visualización de los resultados del conjunto de entrenamiento
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1,
                               step = 0.01),
```

```
                    np.arange(start = X_set[:, 1].min() - 1,
                              stop = X_set[:, 1].max() + 1,
                              step = 0.01))
plt.contourf(X1,
             X2,
             classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75,
             cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0],
                X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Regresión Logística (Conjunto de Entrenamiento)')
plt.xlabel('Edad')
plt.ylabel('Salario Estimado')
plt.legend()
plt.show()
```

Como puedes ver en este código, solamente hemos agregado el color 'azul', lo cual lo tuvimos que hacer porque como ahora tenemos tres clases, entonces `np.unique(y_set)` se convierte en 3.

¿Puedes explicar el método `meshgrid()` con más detalles y algunos de sus usos?

En el método `meshgrid()`, tenemos dos argumentos:

- El primer argumento son los valores de rango de las coordenadas x en nuestra cuadrícula.
- En segundo lugar, están los valores de rango de las coordenadas y en nuestra cuadrícula.

Entonces, digamos que estos argumentos primero y segundo son respectivamente $[-1, +1]$ y $[0, 10]$, entonces obtendremos una cuadrícula donde los valores irán desde $[-1, +1]$ en el eje x y $[0, 10]$ en el eje y.

¿Puede explicar el método `contourf()` con más detalles y algunos de sus usos?

Antes de usar el método `contourf` (contorno), se necesita construir una cuadrícula. Eso es lo que hacemos en la línea de arriba cuando construimos `X1` y `X2`.

Entonces, el método `contourf()` toma varios argumentos:

1. Los valores de rango de las coordenadas x de la cuadrícula,
2. Los valores de rango de las coordenadas y de la cuadrícula,
3. Una línea de ajuste (o curva) que se trazará en esta cuadrícula (trazamos esta línea de ajuste usando la función de predicción porque esta línea son las predicciones continuas de nuestro modelo),
4. Luego, el resto son argumentos opcionales como los colores para trazar regiones de diferentes colores.

Las regiones estarán separadas por esta línea de ajuste, que es de hecho la línea de contorno.

3.1.3. Regresión Logística en R

Hemos predicho los resultados de un conjunto de observaciones (el conjunto de prueba). ¿Cómo hacemos lo mismo para una sola observación?

Primero hay que crear una nueva variable `single_observation` con los valores de entrada (por ejemplo, Edad = 30 y Salario estimado = 50000) y establecer esta variable como un data frame:

```
single_observation = data.frame(Age = 30, EstimatedSalary = 50000)
```

Luego, simplemente se puede predecir el resultado de esta única observación exactamente como lo hicimos con el conjunto de prueba, simplemente reemplazando `test_set` por `single_observation`:

```
prob_pred = predict(classifier, type = 'response', newdata = single_observation)
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

¿Cómo hacemos el escalado inverso en R?

Puedes hacerlo con el siguiente código:

```
training_set[-3] * attr(training_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
test_set[-3] * attr(test_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
```

¿Cómo visualizar la misma clasificación cuando las variables dependientes tienen más de dos clases?

Supongamos que tenemos, por ejemplo, tres categorías (o clases). Entonces el código para la visualización sería el siguiente:

```
# Visualización de los resultados del conjunto de entrenamiento
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('PC1', 'PC2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set,
       pch = '.',
       col = ifelse(y_grid == 2,'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set,
       pch = 21,
       bg = ifelse(set[, 3] == 2,
       'blue3', ifelse(set[, 3] == 1,
       'green4',
       'red3')))
```

3.2. K-Nearest Neighbors (K-NN)

3.2.1. Idea de K-NN

¿Es K-NN un modelo lineal?

No, K-NN es un modelo no lineal, como verás en las secciones prácticas de este curso.

¿Qué pasa si encontramos el mismo número de vecinos? Por ejemplo, si elegimos K = 6 y si encontramos 3 puntos de datos en cada categoría, ¿dónde deberíamos colocar el nuevo punto de datos?

En ese caso, el punto se asigna aleatoriamente a uno de los dos.

¿Qué tipo de problemas comerciales requiere el uso de K-NN?

K-NN puede ser una buena solución para el procesamiento del lenguaje natural. Veremos esto en la Parte 7. Además, K-NN funciona bien para problemas no lineales (cuando los datos no son separables linealmente), como veremos en esta sección (Python o R).

3.2.2. K-NN en Python

¿Cuál es el significado de $n = 5$ al construir el modelo K-NN?

n = 5 significa que se entrenará el modelo K-NN con 5 vecinos. Cuando vemos la clase de intuición para K-NN, n = 5 es el número de vecinos que Juan Gabriel menciona en los videos.

¿Qué número de vecinos debemos elegir?

Cuantos más vecinos tengamos, más posibilidades tendrá el grupo de vecinos de encontrar predicciones correctas y, por lo tanto, más posibilidades tendremos de aumentar la precisión del modelo. Sin embargo, hay que tener cuidado, si tenemos demasiados vecinos, eso provocará un ajuste excesivo en el conjunto de entrenamiento y las predicciones serán malas en las nuevas observaciones en el conjunto de prueba.

¿Cómo puedo invertir la escala para obtener la escala original de las características?

Podemos volver a transformar Edad y Salario utilizando el método `inverse_transform ()`. Se debe tomar el objeto escalador `sc` y aplicar el método `inverse_transform()` de esta manera:

```
X_train = sc.inverse_transform(X_train)
X_test = sc.inverse_transform(X_test)
```

3.2.3. K-NN en R

¿Cómo visualizar la salida en forma de gráfico si el número de predictores es más de 2?

No podríamos porque un predictor corresponde a una dimensión. Entonces, si tenemos más de 2 predictores, sería difícil representar la salida visualmente. Si tenemos tres predictores, aún podríamos hacerlo usando algunos paquetes avanzados y, en ese caso, las regiones de predicción se convertirían en algunos volúmenes de predicción (se tratan de los espacios de región en 3 dimensiones), y el límite de predicción ya no sería una curva de ajuste, pero se convertiría en un plano curvo que separa los

volúmenes de predicción. Sería un poco más difícil ver eso visualmente, pero aún es posible. Pero luego en 4+ dimensiones, eso sería ya muy difícil.

¿Qué número de vecinos debemos elegir?

Cuantos más vecinos tengamos, más posibilidades tendrá el grupo de vecinos de encontrar predicciones correctas y, por lo tanto, más posibilidades tendremos de aumentar la precisión de su modelo. Sin embargo, hay que tener cuidado, si tenemos demasiados vecinos, eso provocará un ajuste excesivo en el conjunto de entrenamiento y las predicciones serán malas en las nuevas observaciones en el conjunto de prueba.

3.3. Support Vector Machine (SVM)

3.3.1. Idea de las SVM

¿Son las SVM un modelo lineal?

Sí, SVM es un modelo lineal. Lo veremos fácilmente en las secciones prácticas de este curso, al visualizar los resultados en el gráfico (notarás que el límite de predicción es una línea recta). Sin embargo, podemos hacer del SVM sea un modelo no lineal, agregando un kernel, que veremos en la siguiente sección.

¿Por qué vemos los vectores de soporte como vectores y no como puntos?

Los vectores son puntos en el espacio 2-D (como en este ejemplo), pero en problemas del mundo real tenemos conjuntos de datos de dimensiones más altas. En un espacio de n dimensiones, los vectores tienen más sentido y es más fácil hacer aritmética vectorial y manipulaciones matriciales en lugar de considerarlos como puntos. Por eso generalizamos los puntos de datos a vectores. Esto también nos permite pensar en ellos en un espacio de N dimensiones.

3.3.2. SVM en Python

¿Qué hace aquí el método `fit`?

Simplemente entrenamos el modelo SVM en `X_train` e `y_train`. Más precisamente, el método `fit` recopilará los datos en `X_train` e `y_train`, y a partir de eso calculará los vectores de soporte. Una vez que se calculan los vectores de soporte, el modelo de clasificador está completamente listo para hacer nuevas predicciones con el método de predicción (porque solo requiere los vectores de soporte para clasificar nuevos datos).

3.3.3. SVM en R

Si mi conjunto de datos tiene más de 2 variables independientes, ¿cómo represento la visualización?

En ese caso, no se puede representar la visualización, porque una variable independiente corresponde a una dimensión. Entonces, si tuvieramos, por ejemplo, 8 variables independientes, necesitaríamos crear una gráfica en 8 dimensiones. Eso sería imposible. Sin embargo, pueden utilizarse técnicas de reducción de dimensionalidad para reducir el número de variables independientes extrayendo las más significativas

desde el punto de vista estadístico. Entonces, si logramos reducir sus 8 variables originales a dos, puede representarse la visualización en 2D.

¿Por qué obtengo resultados diferentes en Python y R?

La diferencia puede provenir de una variedad de factores:

- los factores aleatorios en el modelo y el hecho de que usamos diferentes semillas,
- los diferentes valores predeterminados de los hiperparámetros (los parámetros que no se aprenden) pero lo más normal es elegir los mismos,
- ligeras diferencias en los algoritmos entre Python y R.

3.4. Kernel SVM

3.4.1. Idea del Kernel SVM

¿Por qué exactamente convertimos el espacio de alta dimensión en 3D de nuevo a 2D?

Eso es porque necesitamos volver a nuestro espacio original que contiene nuestras variables independientes. Si nos quedaramos en el espacio 3d perderíamos la información de las variables independientes porque en este espacio 3d no están nuestras variables independientes originales sino las proyecciones de las mismas. Entonces, deseamos volver al espacio 2d original proyectando los puntos hacia atrás.

Cuando aplicamos la transformación $f = x - 5$, los puntos simplemente se mueven hacia la izquierda en el mismo eje 1D. Pero cuando aplicamos la transformación $f = (x - 5)^2$, ¿por qué los puntos se mueven a una curva en forma de U en 2D? ¿No deberían los puntos permanecer en el mismo eje en 1D?

Aquí están involucrados dos tipos diferentes de transformación, la primera es una transformación en una dimensión para la coordenada x, y debe verse de esta manera:

$$x' = x - 5$$

Con esta transformación (llamada traslación), los puntos se mueven 5 unidades hacia la izquierda. Luego, ocurre la transformación de mapeo, que es la transformación de mapeo involucrada en Kernel SVM. Pero este debería verse de esta manera:

$$y = (x' - 5)^2$$

donde y es la nueva coordenada que creamos con este mapeo en una dimensión superior.

¿Qué kernel debemos elegir?

Una buena forma de decidir qué kernel es el más apropiado es hacer varios modelos con diferentes kernels, luego evaluar cada uno de sus rendimientos y finalmente comparar los resultados. Luego, elegimos el kernel con mejores resultados. Hay que tener cuidado de evaluar el rendimiento del modelo en nuevas observaciones (preferiblemente con K-Fold Cross Validation que veremos en la Parte 10) y de considerar diferentes métricas (Accuracy, F1-Score, etc.).

3.4.2. Kernel SVM en Python

¿Qué hace aquí el método `fit`?

Simplemente entrena el modelo SVM en `X_train` e `y_train` con un Kernel no lineal. Más precisamente, el método de ajuste recopilará los datos en `X_train` e `y_train`, y a partir de ahí hará todas las operaciones sucesivas que vimos en la lección de intuición: primero, un mapeo en un espacio dimensional superior donde los datos están linealmente separable, luego calculará los vectores de soporte en este espacio dimensional superior y, finalmente, una proyección de nuevo en 2D.

¿Cómo predecir el resultado de una única observación nueva?

Deben ingresarse los datos de la observación única en una matriz, escalarla con nuestro objeto escalador `sc` (necesitamos escalar aquí porque nuestro modelo fue entrenado con datos escalados) y usar el método de predicción como este:

```
single_prediction = classifier.predict(sc.transform(np.array([[30, 80000]])))
```

3.4.3. Kernel SVM en R

¿Cómo saber fácilmente con R si el conjunto de datos no es separable linealmente?

Si es un problema de regresión, hacemos un modelo de regresión lineal múltiple con todas las variables independientes incluidas, y si es un problema de clasificación, hacemos un modelo de regresión logística con todas las variables independientes incluidas. Luego usamos la función de resumen de estadísticas `summary()` en R para ver los niveles de significancia estadística de sus variables independientes. Si algunos de ellos tienen significación estadística (p-valores pequeños), es probable que nuestro conjunto de datos sea linealmente separable. De lo contrario, es posible que el conjunto de datos no se pueda separar linealmente.

Predijimos los resultados de un conjunto de observaciones (el conjunto de prueba). ¿Cómo hacemos lo mismo para una sola observación?

Primero debe crear una nueva variable `single_observación` con los valores de entrada (por ejemplo, Edad = 30 y Salario estimado = 50000) y establecer esta variable como un data frame:

```
single_observation = data.frame(Age = 30, EstimatedSalary = 50000)
```

Luego, simplemente puede predecirse el resultado de esta única observación exactamente como lo hicimos con el conjunto de prueba, simplemente reemplazando `test_set` por `single_observation`:

```
y_pred = predict(classifier, newdata = single_observation)
```

¿Cómo hacer el cambio inverso de escala en R?

Puede hacerse con el siguiente código:

```
training_set[-3] * attr(training_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
test_set[-3] * attr(test_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
```

3.5. Naive Bayes

3.5.1. Idea de Naive Bayes Intuition

¿Naive Bayes es un modelo lineal o no lineal?

Naive Bayes es un modelo no lineal. Lo veremos muy claramente en Python o R al representar el límite de predicción, que será una curva muy suave (derivable) que separará bien las observaciones distribuidas no linealmente.

¿Puede ser $P(x)$ cero?

Sí, $P(X)$ puede ser cero. Eso puede suceder cuando el nuevo punto de datos es un valor atípico y, por lo tanto, no hay otros puntos de datos dentro del radio del círculo. Sin embargo, la fórmula del teorema de Naive Bayes solo es cierta cuando $P(X)$ es diferente de cero. Cuando $P(X)$ es igual a cero, entonces $P(Walks)|X)$ simplemente se define como cero.

¿Cómo decide el algoritmo el círculo?

En la lección de Intuición, vemos que se dibuja un círculo para crear una colección de puntos de datos similar al nuevo punto de datos. El nuevo punto de datos estaba aproximadamente en el centro del círculo y, por lo tanto, vimos que el número de puntos verdes era menor que el número de puntos rojos y, por lo tanto, el nuevo punto pasó a la categoría roja. Pero si hubiéramos dibujado el círculo de manera un poco diferente alrededor del nuevo punto de datos, entonces el número de puntos verdes podría haber sido más que rojo. Entonces, ¿cómo se elige ese círculo? Hay un parámetro en el modelo que decide el radio del círculo, al igual que hay un parámetro que elige el número de vecinos en K-NN.

Naive Bayes vs K-NN

La mejor manera de responder a esa pregunta es probar ambos. Gracias a las plantillas que ofrecemos en nuestro curso, solo lleva unos minutos comparar los resultados. Basta entender que la principal diferencia entre los dos es que Naive Bayes se basa en un enfoque probabilístico, a diferencia del clásico de K-NN.

3.5.2. Naive Bayes en Python

¿Qué hace aquí el método `fit`?

Simplemente entrena el modelo Naive Bayes en `X_train` e `y_train`. Más precisamente, el método de ajuste recopilará los datos en `X_train` e `y_train`, y a partir de ahí hará todas las operaciones sucesivas que vimos en la clase de intuición: primero obtendrá la probabilidad previa, la probabilidad marginal, la probabilidad y la probabilidad posterior, y luego, usando los círculos alrededor de las observaciones, aprenderá cómo clasificar estas observaciones y predecir resultados futuros.

¿Cómo obtener las probabilidades como salida?

Esta es la forma de hacerlo:

```
y_proba = classifier.predict_proba(X_test)
```

Esto nos dará una matriz con 2 columnas, una para 0 y otra para 1.

3.5.3. Naive Bayes en R

¿Por qué necesitamos convertir la variable dependiente como factor?

Hacemos esto para especificar que la variable dependiente es una variable categórica y que 0 y 1 debe tratarse como categorías (factores).

¿En la función `naiveBayes()` se puede utilizar el argumento `formula = Purchased $\sim$ .` similar a especificar la x y la y como vimos en el curso?

Sí, sería lo mismo, pero necesitaríamos especificar el argumento `data`. Básicamente, se debe especificar cuál es la variable dependiente, la variable independiente y el conjunto de entrenamiento. Esa es información suficiente para que la función sepa cómo entrenar el modelo.

3.6. Clasificación con Árboles de Decisión

3.6.1. Idea de la Clasificación con Árboles de Decisión

¿Cómo divide el algoritmo los puntos de datos?

Utiliza la reducción de la desviación estándar de las predicciones. En otras palabras, la desviación estándar se reduce inmediatamente después de una división. Por lo tanto, la construcción de un árbol de decisiones se trata de encontrar el atributo que devuelve la reducción de desviación estándar más alta (es decir, las ramas más homogéneas).

¿Qué es la ganancia de información y cómo funciona en los árboles de decisión?

La ganancia de información en la regresión del árbol de decisión es exactamente la reducción de la desviación estándar que buscamos alcanzar. Calculamos cuánto disminuye la desviación estándar después de cada división. Debido a que se reduce la desviación estándar después de una división, más homogéneos resultan ser los nodos secundarios.

¿Qué es la entropía y cómo funciona en los árboles de decisión?

La Entropía mide el desorden en un conjunto, aquí en una parte resultante de una división. Entonces, cuanto más homogéneos sean los datos en una parte, menor será la entropía. Cuantas más divisiones tengamos, más posibilidades tendremos de encontrar partes en las que los datos sean homogéneos y, por lo tanto, menor será la entropía (cercana a 0) en estas partes. Sin embargo, es posible que aún encontremos algunos nodos donde los datos no sean homogéneos y, por lo tanto, la entropía no será tan pequeña.

3.6.2. Clasificación con Árboles de Decisión en Python

¿Qué hace aquí el método `fit`?

Simplemente entrenará el modelo de árbol de decisión en `X_train` e `y_train`. Más precisamente, el método `fit` recogerá los datos en `X_train` e `y_train`, y a partir de ahí hará todas las operaciones sucesivas que vimos en la lección de intuición: primero seleccionará el criterio (que elegimos que sea la entropía en nuestro código), y luego dividirá los datos en diferentes nodos sucesivos en la dirección de reducción de entropía, hacia nodos más homogéneos. Finalmente, cuando se realicen todas las divisiones,

estará completamente entrenado y listo para predecir los resultados de nuevas observaciones gracias al método `predict ()`.

¿Cómo representar la gráfica de un árbol en Python?

Así es como puede representarse un árbol de decisiones en Python (solo hay que agregar el siguiente código al final del nuestro que vemos en el curso):

```
# Representar el árbol
# En el terminal escribimos: pip install pydot2

from sklearn import tree
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydot
dot_data = StringIO()
tree.export_graphviz(classifier,
                     out_file = dot_data,
                     feature_names = ['Age', 'Estimated Salary'],
                     class_names = ['Yes', 'No'],
                     filled = True,
                     rounded = True,
                     special_characters = True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

3.6.3. Clasificación con Árboles de Decisión en R

¿Cómo se hicieron las divisiones en R?

Se hacen estas divisiones con salarios agrupados en tres niveles, basados en la entropía y la ganancia de información. Las divisiones se realizan de manera que las partes resultantes de las divisiones sean lo más homogéneas posible.

¿Por qué `y_pred` es una matriz en este modelo mientras que en todos los demás modelos es un vector?

Es una especificidad del modelo de árbol de decisión: las predicciones se devuelven en una matriz de dos columnas con las probabilidades $prob(y = 0)$ y $prob(y = 1)$.

3.7. Clasificación con Random Forest

3.7.1. Idea de la Clasificación con Random Forest

¿Cuál es la ventaja y el inconveniente de los bosques aleatorios en comparación con los árboles de decisión?

Ventaja: los bosques aleatorios pueden darnos un mayor poder de predicción que los árboles de decisión. Inconveniente: el árbol de decisiones nos dará más interpretabilidad que los bosques aleatorios, porque

puede dibujarse el gráfico de un árbol de decisiones para ver las diferentes divisiones que conducen a la predicción, como se ve en la clase de intuición. Eso es algo que no puedes hacer con Random Forest.

¿Cuándo usar Random Forest y cuándo usar los otros modelos?

La mejor respuesta a esa pregunta es: ¡debemos probarlos todos!

De hecho, gracias a las plantillas, solo te llevará 10 minutos probar todos los modelos, que es muy poco en comparación con el tiempo dedicado a las otras partes de un proyecto de ciencia de datos (como el preprocesamiento de datos, por ejemplo). Así que no tengas miedo de probar todos los modelos de clasificación y comparar los resultados (a través de la validación cruzada que veremos en la Parte 10). Eso es, te brindamos los mejores modelos en este curso para que los tengas en su kit de herramientas y aumentes tus posibilidades de obtener mejores resultados.

Sin embargo, si deseas algunos atajos, aquí hay algunas reglas generales que te ayudarán a decidir qué modelo usar:

Primero, debes averiguar si tu problema es lineal o no lineal. Aprenderás cómo hacerlo en la Parte 10: Selección de modelos. Luego:

- Si tu problema es lineal, debes optar por un modelo de regresión logística o un modelo SVM, porque ambos son modelos lineales.
- Si tu problema no es lineal, debes optar por K-NN, Kernel SVM, Naive Bayes, Árboles de Decisión o Random Forest. Entonces, ¿cuál deberías elegir entre estos cinco? Eso lo aprenderás en la Parte 10: Selección de modelos. El método consiste en utilizar una técnica muy relevante que evalúa el rendimiento de los modelos, llamada k-Fold Cross Validation, y luego seleccionar el modelo que muestra los mejores resultados. Siéntete libre de pasar directamente a la Parte 10 si ya deseas aprender cómo hacerlo.

3.7.2. Clasificación con Random Forest en Python

¿Cómo sé cuántos árboles debo utilizar?

Primero, te recomendaría elegir el número de árboles experimentando. Por lo general, lleva menos tiempo del que pensamos encontrar el mejor valor ajustando y ajustando el modelo manualmente. En realidad, eso es lo que hacemos en general cuando construimos un modelo de aprendizaje automático: lo hacemos en varias tomas, experimentando varios valores de hiperparámetros como el número de árboles. Sin embargo, también te recuerdo que en la Parte 10 cubriremos la Validación cruzada de k-Fold y la Búsqueda de cuadrícula, que son técnicas poderosas que pueden usarse para encontrar el valor óptimo de un hiperparámetro, como aquí el número de árboles.

3.7.3. Clasificación con Random Forest en R

¿Cómo decidimos cuántos árboles serían suficientes para obtener un resultado relativamente preciso?

Al igual que en Python, puedes encontrar un número relevante de árboles a través de la experimentación y el ajuste manual. Debes usar suficientes árboles para obtener una buena precisión, pero no debes usar demasiados árboles porque eso podría causar un ajuste excesivo. Aprenderás a encontrar el número óptimo de árboles en la primera sección de la Parte 10: Selección del modelos. Se realiza con una técnica llamada Ajuste de parámetros (búsqueda de cuadrícula con validación cruzada de k-Fold).

¿Podemos seleccionar las variables más significativas gracias al p-valor como hicimos antes en R?

No se puede utilizar el p-valor porque los bosques aleatorios no son modelos lineales y los p-valores se aplican solo a los modelos lineales. Por lo tanto, la selección de características están fuera de discusión. Pero podemos realizar la extracción de características, que veremos en la Parte 9: Reducción de dimensionalidad. Eso se puede aplicar a Random Forests y reducirá la cantidad de variables que deben considerarse.

¿Cómo se puede reducir el sobreajuste de los bosques aleatorios?

Puedes hacerlo jugando con los parámetros de penalización y regularización si hay algunos en la clase utilizados para construir el modelo. Sin embargo, la forma mejor y más eficiente de reducir el sobreajuste, especialmente para los bosques aleatorios, es aplicar la validación cruzada de k-fold y optimizar un parámetro de ajuste con remuestreo de los datos. No te preocupes, veremos todo esto en profundidad en la Parte 10: Selección de modelos y Ensemble Learning.

3.8. Evaluación del rendimiento de los modelos de clasificación

¿Cuánto debemos fiarnos de la matriz de confusión?

Podemos usar la matriz de confusión para tener una idea de cuán potencialmente bien puede funcionar nuestro modelo. Sin embargo, no debemos limitar nuestra evaluación del rendimiento a la precisión en una división de prueba de tren, como es el caso en la matriz de confusión. Primero, está el problema de la varianza, por lo que debemos aplicar la validación cruzada de k-Fold para obtener una medida más relevante de la precisión. Aprenderás cómo hacerlo en la primera sección de la Parte 10: Selección de modelo. Entonces, no solo está la accuracy como medida única que debe considerarse. También deben considerarse otras métricas como Precision, Recall y F1 Score. Lo veremos en detalle en la Parte 7.

¿Es la curva de ganancia acumulada lo mismo que la curva CAP?

Sí, es lo mismo, la Curva de Ganancia (Gain Curve) es el nombre de la curva CAP en un contexto de marketing.

Si contactamos a 20000 clientes y 4000 responden, ¿por qué esto no tiene un efecto multiplicador? Entonces, si me dirijo a 100000 clientes, ¿debería obtener una respuesta de 50000?

La idea aquí es que podemos maximizar las ganancias seleccionando a los clientes que pueden comprar más, de modo que si la $P(\textit{cliente compre})$ es mayor que algún umbral, entonces deberíamos enviar un correo electrónico.

La idea es que nunca obtendrás una respuesta de 50.000 ya que, pase lo que pase, solo 10.000 clientes comprarán en cada oferta.

Entonces, lo que nos gustaría hacer es que de nuestra base de clientes de 100,000, deben seleccionarse esos 10,000 clientes exactamente que comprarán para la oferta actual. Si tenemos un modelo tan bueno, que puede identificar con precisión a todos los que van a comprar, entonces podemos simplemente enviar un correo electrónico a esa cantidad de personas.

Aquí no hay efecto multiplicativo. Incluso si enviamos correos electrónicos a todos los clientes (100k), solo se comprarán 10k. (El 90 % de los clientes no suele estar interesado en cada oferta). Nuestro objetivo aquí es minimizar los correos electrónicos que enviamos enviando correos electrónicos a los 10k que comprarán nuestra oferta. Por lo tanto, podemos usar modelos probabilísticos para predecir

si un cliente comprará o no. Si la probabilidad de que el cliente compre es superior a algún umbral, podemos enviarle un correo electrónico. Esto aumentará las posibilidades de que cada cliente compre la oferta, manteniendo al mínimo el número de correos electrónicos que deben enviarse.

¿Deberíamos representar la curva CAP en Python o R?

Se recomienda más la plantilla de Excel proporcionada en el curso. Primero, exporta tus valores en Python:

```
classifier = SVC(kernel = 'rbf', probability = 1, random_state = 0)
Y_pred_proba = classifier.predict_proba(X_test)
Y_cap = np.c_[Y_test, Y_pred_proba[:, 1:]]
dataframe = pd.DataFrame(data = Y_cap)
dataframe.to_csv('CAP.csv', sep=';')
```

Luego, utiliza esta plantilla de curva CAP que te dará directamente la curva CAP una vez que pegues los valores exportados dentro de esta plantilla. En cuanto a R, hay un par de paquetes de R que puedes usar para calcular una curva CAP: `ROCR` y `MASS`.

¿Qué son los sesgos o variaciones bajos / altos en el aprendizaje automático?

- El sesgo bajo es cuando las predicciones de nuestro modelo están muy cerca de los valores reales.
- El sesgo alto es cuando las predicciones de nuestro modelo están lejos de los valores reales.
- La varianza baja: cuando ejecutamos nuestro modelo varias veces, las diferentes predicciones de los puntos de observación no varian demasiado
- La varianza alta: cuando ejecutamos nuestro modelo varias veces, las diferentes predicciones de los puntos de observación varian mucho.

Lo que deseamos obtener cuando construimos un modelo es: un sesgo y una varianza bajas en ambos casos.

Capítulo 4

Clustering

4.1. Clustering con K-Means

4.1.1. Intuición del Clustering con K-Means

¿Dónde podemos aplicar el algoritmo de agrupación por clústeres en la vida real?

Puede aplicarse para diferentes propósitos:

- Segmentación de mercado,
- Medicina con, por ejemplo, detección de tumores,
- Detección de fraudes,
- Simplemente para identificar algunos grupos de clientes en la empresa o negocio.

¿Cómo funciona el truco de la línea perpendicular cuando $k \geq 3$?

Este truco se usa principalmente solo en espacios 2D o 3D. Por lo general, la distancia euclidiana se utiliza en todos los puntos de dimensiones altas para realizar la agrupación. En datos de muy alta dimensión, podríamos hacer un truco, que es una "esfera envolvente", es decir, comenzar en cada centroide (k esferas en total), comenzar a hacer crecer un espacio esférico (círculo en 2D, esfera en 3D, etc.), radialmente hacia afuera . Hasta que las esferas se crucen, todo lo que la esfera engulle pertenece al mismo cluster. Todos los demás puntos pueden asignarse a grupos calculando la distancia perpendicular entre los centroides.

¿Cómo funciona el método del codo cuando están involucradas más de 2 variables?

Es lo mismo pero usamos la distancia euclidiana en n dimensiones para calcular el WCSS (p y q son dos observaciones, y sus coordenadas $p_1, ..., p_n, q_1, ..., q_n$ son las características que se describen en estos puntos de observación):

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

¿Existe una forma alternativa al método del codo para encontrar el número máximo de clusters?

Claro, una forma alternativa sería a través del ajuste de parámetros con búsqueda por cuadrícula, que se verá en la Parte 10 - Selección de modelos.

4.1.2. Clustering de K-Means en Python

¿Para qué sirve exactamente el parámetro `max_iter` en la clase `KMeans ()`?

Cuando aplicamos k-means, volvemos al paso anterior muchas veces (ver las clases de intuición en el curso). Cada vez que hacemos eso, es una iteración. Y el valor `max_iter` es el número máximo de estas iteraciones que se deben hacer.

¿Por qué no consideramos la edad como parámetro?

No consideramos la edad por dos razones:

1. Habíamos verificado previamente que no tenía impacto en la variable dependiente.
2. Quería tener dos variables independientes al final, para que podamos visualizar los resultados en dos dimensiones (ya que una variable independiente corresponde a una dimensión).

¿Cómo obtener estadísticas para cada uno de los clústeres?

La forma más sencilla de obtener información estadística de los clústeres es utilizar el atributo `cluster_centers_` de la clase `KMeans`. Nos dará una matriz con las coordenadas de cada cluster, que es la media de cada columna de características de cada grupo. Luego, también podemos obtener información sobre los puntos de observación utilizando el atributo `labels_`.

Podemos solicitar estos atributos de la siguiente manera:

```
kmeans.cluster_centers_
kmeans.labels_
```

Luego, podríamos obtener algunos valores útiles como el número de elementos en cada grupo, la media de los salarios o puntuaciones de gasto en cada clustr, etc. La mejor manera de hacer esto es poner los cinco grupos en 5 variables de esta manera:

```
Cluster_0 = X[y_kmeans == 0]
Cluster_1 = X[y_kmeans == 1]
Cluster_2 = X[y_kmeans == 2]
Cluster_3 = X[y_kmeans == 3]
Cluster_4 = X[y_kmeans == 4]
```

Entonces se obtiene lo que queramos con estas variables que son exactamente los clústeres.

Por ejemplo, si estamos interesados en tener el recuento de observaciones para cada grupo, bueno, ahora podemos verlo directamente en el Explorador de variables observando el número de filas de cada grupo. O también se puede hacer esto:

```
len(Cluster_0)
len(Cluster_1)
len(Cluster_2)
len(Cluster_3)
len(Cluster_4)
```

Luego, se pueden obtener la media y la desviación estándar de cada característica de los clústers de esta manera:

```
Cluster_0[0].mean()
Cluster_1[0].mean()
Cluster_2[0].mean()
Cluster_3[0].mean()
Cluster_4[0].mean()
Cluster_0[0].std()
Cluster_1[0].std()
Cluster_2[0].std()
Cluster_3[0].std()
Cluster_4[0].std()
```

También puedes obtener el porcentaje de observaciones mediante los siguientes cálculos simples:

```
len(Cluster_0) / 200
len(Cluster_1) / 200
len(Cluster_2) / 200
len(Cluster_3) / 200
len(Cluster_4) / 200
```

Veamos la siguiente línea de código con más detalles:

```
plt.scatter(kmeans.cluster_centers_[:,0],
            kmeans.cluster_centers_[:, 1],
            s = 300,
            c = 'yellow',
            label = 'Centroids')
```

En la función `scatter()` del módulo `plt`:

- el primer argumento es la coordenada x de los centros del clúster (los centroides),
- el segundo argumento es la coordenada y de los centros del grupo (los centroides),
- el tercer argumento s es el tamaño de los puntos del centroide en la gráfica,
- el cuarto argumento es el color de los puntos del centroide en la gráfica,
- el quinto argumento es la etiqueta que aparece en la leyenda correspondiente a los centroides.

4.1.3. Clustering de K-Means en R

¿Cómo no caigo en la trampa de la inicialización aleatoria en R?

R se encarga de evitar caer en una trampa de inicialización aleatoria entre bastidores. Sin embargo, tienes otros paquetes geniales en R donde puedes usar explícitamente k-means++, como el paquete `KMeans_rcpp`.

¿Por qué no hemos utilizado todas las variables como la edad?

No los hemos utilizado porque heemos verificado previamente que no tenían ningún impacto en la variable dependiente. Además, queríamos tener dos variables independientes para que pudiéramos visualizar los resultados en dos dimensiones (porque una variable independiente corresponde a una dimensión).

¿Cuál es el papel de `nstart` aquí?

Debido a que la agrupación de K-medias comienza con k centroides elegidos al azar, se puede obtener una solución diferente cada vez que se invoca la función. El uso de la función `set.seed()` garantiza que los resultados sean reproducibles. Además, este enfoque de agrupamiento puede ser sensible a la selección inicial de centroides. La función `kmeans()` tiene una opción `nstart` que intenta múltiples configuraciones iniciales e informa sobre la mejor. Por ejemplo, agregar `nstart = 25` genera 25 configuraciones iniciales. Este enfoque se recomienda a menudo.

¿Cómo puedo representar los resultados sin la escala y con los nombres de los grupos?

Aquí hay una implementación de dicho diagrama:

```
plot(x = dataset[,1],
     y = dataset[,2],
     col = y_kmeans,
     pch = 19,
     xlim = c(from=min(dataset[,1]), to=max(dataset[,1]+30)),
     xlab = "Ingreso Anual", ylab="Puntuación de Gasto")
clusters = c("Careless", "Standard", "Sensible", "Target", "Careful")
legend('bottomright', legend=clusters, col=1:5, pch=19, horiz=F)
```

4.2. Clustering Jerárquico

4.2.1. Intuición del clustering jerárquico

¿Cuál es el punto del clustering jerárquico si siempre conduce a un grupo por punto de observación?

El punto principal de la agrupación jerárquica es hacer el dendrograma, porque necesitamos comenzar con un solo grupo, luego trabajar hacia abajo para ver las diferentes combinaciones de grupos hasta tener un número de grupos igual al número de observaciones. Y es el dendrograma en sí el que permite encontrar la mejor configuración de agrupamiento.

Cuando se compara la distancia entre dos grupos o un grupo y un punto, ¿cómo se mide exactamente? ¿Se toma el centroide en el grupo y se mide la distancia?

Exactamente, la métrica es la distancia euclidiana entre el centroide del primer grupo y el punto (o el centroide del otro grupo para la distancia entre dos grupos).

¿También necesitamos realizar el escalado de características para la agrupación en clústeres jerárquica?

Sí, porque las ecuaciones de los problemas de agrupamiento involucran la Distancia euclidiana. Siempre que las ecuaciones del modelo involucren la distancia euclidiana, debe aplicarse el escalado de características.

4.2.2. Clustering Jerárquico en Python

¿Deberíamos usar el dendrograma o el método del codo para encontrar ese número óptimo de conglomerados?

Es recomendable usar ambos (es más rápido probar ambos de lo que crees gracias a las plantillas del curso), solo para verificar ese número óptimo. Sin embargo, si realmente solo tienes tiempo para uno, te recomendaría el método del codo. El dendrograma no siempre es la forma más fácil de encontrar el número óptimo de agrupaciones. Pero con el método del codo es muy fácil, ya que la mayor parte del tiempo el codo es muy obvio de detectar.

¿Podrías explicar más sobre cómo se forman los clústeres a través del método de clase de Python `AgglomerativeClustering()`?

La agrupación jerárquica es una familia general de algoritmos de agrupación que construyen agrupaciones anidadas fusionándolos o dividiéndolos sucesivamente. Esta jerarquía de grupos se representa como un árbol (o dendrograma). La raíz del árbol es el grupo único que reúne todas las muestras, siendo las hojas los grupos con una sola muestra. La clase `AgglomerativeClustering()` realiza un agrupamiento jerárquico utilizando un enfoque de abajo hacia arriba: cada observación comienza en su propio cluster y los grupos se fusionan sucesivamente. El método de clase minimiza la suma de las diferencias cuadradas dentro de todos los grupos. Es un enfoque que minimiza la varianza y, en este sentido, es similar a la función objetivo de k-means, pero se aborda con un enfoque jerárquico aglomerativo.

Pensé que los dendrogramas eran como la memoria del algoritmo HC. Si es así, ¿por qué primero hacemos el dendrograma y luego aplicamos el agrupamiento aglomerativo? ¿Eso significa que ejecutamos dos veces el algoritmo de agrupamiento?

El único propósito de hacer primero el dendrograma es tener una idea del número óptimo de conglomerados. Es como la memoria o la historia del algoritmo HC. Pero a través de ese historial / memoria podemos ver el número óptimo de clústeres. Y eso es lo que hacemos en este paso. Luego, cuando aplicamos agrupaciones aglomerativas, podemos seleccionar el número óptimo de agrupaciones que encontramos gracias al dendrograma.

¿Qué es la 'afinidad' en la clase `AgglomerativeClustering()`?

La 'afinidad' se refiere a la distancia utilizada en el algoritmo K-Means, que es la distancia euclidiana (la más clásica, la geométrica). Se refiere a cómo el algoritmo HC define (encuentra) los centroides más cercanos a cada punto. Y luego lo que lo acompaña es el método de Ward utilizado para construir los clústeres, lo que significa que usamos la Suma de cuadrados dentro del clúster como un criterio para reunir y formar nuestros clústeres más homogéneos.

¿Por qué no implementamos simplemente un código para que encuentre automáticamente el número óptimo de clústeres, en lugar de seleccionarlo manualmente o visualmente?

Porque la elección del número óptimo de clústeres también está influenciada por el problema empresarial (objetivos y limitaciones). Por ejemplo, algunos problemas comerciales tienen un número mínimo o máximo de clústeres, o un número mínimo / máximo de elementos por clúster. Tales restricciones requieren que tengamos varias opciones, y la mejor manera de descubrir las mejores opciones es mirando los gráficos (con el dendrograma o el codo).

4.2.3. Clustering Jerárquico en R

¿Por qué no aplicamos el escalado de características en R?

Porque la función que usamos en R se encarga automáticamente de llevar a cabo escalado de características.

¿Cómo puedo visualizar los clústeres con la escala original?

Si desea ver los grupos con escalas originales, puedes usar el siguiente algoritmo:

```
clusters= 1:5
plot(x=dataset[,1], y=dataset[,2],
     col=y_hc, pch=19, xlim=c(from=min(dataset[,1]), to=max(dataset[,1]+20)))
legend('bottomright', inset=0.01, legend=clusters, col=1:5, pch=19, horiz=F)
```

Capítulo 5

Reglas de Asociación

5.1. Apriori

5.1.1. Intuición de Apriori

¿Cuáles son las tres relaciones esenciales entre el soporte, la confianza y el impulso?

Dadas dos películas M_1 y M_2, aquí están las tres relaciones esenciales para recordar:

- Relación entre el soporte y la confianza:

$$\text{confidence}(M_1 \rightarrow M_2) = \frac{\text{support}(M_1, M_2)}{\text{support}(M_1)}$$

* Relación entre el impulso y el soporte:

$$\text{lift}(M_1 \rightarrow M_2) = \frac{\text{support}(M_1, M_2)}{\text{support}(M_1) \times \text{support}(M_2)}$$

* Relación entre el impulso y la confianza (consecuencia de las dos ecuaciones anteriores):

$$\text{lift}(M_1 \rightarrow M_2) = \frac{\text{confidence}(M_1 \rightarrow M_2)}{\text{support}(M_2)}$$

¿Las funciones de confianza y impulso son simétricas?

Dadas las tres ecuaciones de la pregunta anterior, podemos ver fácilmente que:

- La confianza no es simétrica:

$$\text{confidence}(M_1 \rightarrow M_2) \neq \text{confidence}(M_2 \rightarrow M_1)$$

* El impulso es simétrico:

$$\text{lift}(M_1 \rightarrow M_2) = \text{lift}(M_2 \rightarrow M_1)$$

5.1.2. Apriori en Python

Pregunta importante: ¿es R mejor que Python para el aprendizaje de reglas de asociación?

Si, absolutamente. El paquete R está mucho más optimizado y es más fácil de usar. Pero para los que odian a R, nos aseguramos de proporcionar una solución de Python. Sin embargo, si no te importa el lenguaje, te recomendamos encarecidamente que elijas R.

Tengo problemas para encontrar las reglas de asociación en Python. ¿Cómo puedo verlos de forma más explícita?

Lo siento, la versión del paquete cambió después de que grabara la clase Pero puedes verlos simplemente agregando lo siguiente al final de tu código de Python (en una nueva sección justo después de la sección "Visualización de los resultados"):

```
the_rules = []
for result in results:
    the_rules.append({'rule': ','.join(result.items),
                      'support':result.support,
                      'confidence':result.ordered_statistics[0].confidence,
                      'lift':result.ordered_statistics[0].lift})
df = pd.DataFrame(the_rules, columns = ['rule', 'support', 'confidence', 'lift'])
```

Después de la ejecución de este código, deberías ver un DataFrame llamado 'df' que aparece en el 'Explorador de variables' con toda la información necesaria.

5.1.3. Apriori en R

¿De dónde viene el 3 en el cálculo del soporte?

Elegimos 3 porque consideramos los productos que se compran al menos 3 veces al día. Porque menos de 3 veces al día parecía no ser lo suficientemente relevante como para incluirlas en el análisis.

¿Por qué el soporte y la confianza deberían ser pequeños?

Es porque queremos que nuestras reglas sean significativas y reflejen una tendencia real, no algo realmente raro que sucede una vez cada pocos años, y que por accidente entró en nuestros registros.

¿Por qué una regla como el agua mineral (lhs) y la pasta de trigo integral (rhs) no aparecen en la parte superior de la lista de reglas de asociación?

Es porque el agua mineral puede ir con cualquier cosa, por lo que la regla de asociación entre el agua mineral y la pasta de trigo integral no es la más relevante. Piénsalo de esta manera: si compras agua mineral, ¿desea comprar además pasta de trigo integral como primera opción?

¿Cómo medirías qué tan buenas son las reglas aquí? Mencionamos en los vídeos cómo Amazon y Netflix usan algoritmos más avanzados, pero ¿cómo han llegado a la conclusión de que sus algoritmos son superiores?

En el aprendizaje de reglas de asociación, simplemente medimos la relevancia con el lift Cuanto más altos sea el lift, mejores serán las reglas. Amazon y Netflix funcionan con muchos algoritmos, incluidas las reglas de asociación, los sistemas de recomendación y otros modelos avanzados de aprendizaje automático. Evalúan sus modelos a través de métricas de rendimiento y experimentación.

En un escenario real, ¿cuál es el período de tiempo ideal que debemos considerar para hacer un buen modelo de análisis de la cesta de la compra? ¿Y debería hacerse en cada tienda por separado o por región?

Un mes es un buen período de tiempo. Sin embargo, también podría considerarse de 3 a 6 meses para normalizar el efecto de estacionalidad, o podría ejecutarse el mismo modelo todos los meses, lo que preferiríamos hacer para captar las especificidades de cada mes (temporada, tasa de turismo, etc.).

Luego, el análisis de la ceesta de la compra debe realizarse en cada tienda por separado, ya que depende de los comportamientos del cliente dentro de un vecindario específico. Básicamente, los clientes pueden comportarse de manera diferente en diferentes vecindarios.

5.2. Eclat

5.2.1. Intuición de Eclat

¿Cuándo deberíamos usar Eclat en lugar de Apriori?

La única ventaja de Eclat en comparación con Apriori es que es más simple y rápido de usar. Sin embargo, si necesitas realizar un análisis profundo de la cesta de mercado, definitivamente deberías optar por Apriori.

5.2.2. Eclat en Python

¿Podrías proporcionar una implementación de Eclat en Python?

Crédito a Marcello Morchio, alumno del curso en inglés que amablemente compartió su implementación:

```
# Eclat, por Marcello Morchio, 4 de Diciembre 2017

# Importar las librerías
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importar el dataset
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None )
transactions = [[str(dataset.values[i,j]) for j in range(0, dataset.shape[1])
                 if str(dataset.values[i,j]) != 'nan'] for i in range(0, dataset.shape[0])]

# Generar una lista de elementos únicos en las transacciones.
items = list()
for t in transactions:
```

```
    for x in t:
        if(not x in items):
            items.append(x)

# Generar una lista de pares de elementos con valor de soporte relevante
# [[(item_a, item_b) , support_value]]
# support_value se inicializa a 0 para todas las parejas
eclat = list()
for i in range(0, len(items)):
    for j in range(i+1, len(items)):
        eclat.append([(items[i],items[j]),0])

# Calculamos el valor de soporte para cada par buscando transacciones con ambos elementos
for p in eclat:
    for t in transactions:
        if(p[0][0] in t) and (p[0][1] in t):
            p[1] += 1
    p[1] = p[1]/len(transactions)

# Convertimos eclat en un DataFrame ordenado para visualizarlo en el explorador de variables
eclatDataFrame = pd.DataFrame(eclat,
columns = ['rule','support']).sort_values(by = 'support', ascending = False)
```

5.2.3. Eclat en R

Parece que el Agua Mineral está sesgando los resultados de las asociaciones porque es la compra más frecuente, entonces, ¿sería irrazonable simplemente excluirla de los datos y reconstruir el modelo sin ella?

Sí, eso podría ser relevante. Sin embargo, habría que comprobar la participación de los ingresos provenientes del agua mineral. Si la participación es alta, tal vez no sea una mala idea colocarla junto a su mejor producto de asociación después de todo.

Encontramos 5 duplicados en los datos. ¿No deberíamos eliminarlos?

Absolutamente no. Podría significar que 5 personas por accidente compraron lo mismo, por lo que estos registros representan diferentes clientes y luego deben guardarse.

¿Qué es la densidad explicada en la clase de Eclat en R?

La densidad es un término que puede tener diferentes significados en diferentes contextos. En este caso particular, la densidad es la proporción de celdas con elementos (valores distintos de cero en la matriz) sobre el número total de celdas en la matriz.

Capítulo 6

Aprendizaje por Refuerzo

6.1. Upper Confidence Bound (UCB)

6.1.1. Intuición de UCB

¿Podrías explicar con más detalle qué es una distribución? ¿Qué hay en el eje x y en el eje y?

Supongamos que tenemos un experimento con hacer clic en un anuncio 100 veces y calculamos cuál es la frecuencia con la que se hace clic en el anuncio. Luego lo repetimos nuevamente por otras 100 veces. Y nuevamente por otras 100 veces. De ahí obtenemos muchas frecuencias. Si repetimos estos cálculos de frecuencia muchas veces, por ejemplo 500 veces, podemos trazar un histograma de estas frecuencias. Según el Teorema central del límite, tendremos una forma de campana y su media será la media de todas las frecuencias que obtuvimos durante el experimento. Por lo tanto, en conclusión, en el eje x tenemos los diferentes valores posibles de estas frecuencias, y en el eje y tenemos el número de veces que obtuvimos cada frecuencia durante el experimento.

En la primera conferencia sobre intuición, ¿podrías explicar por qué D5 (en naranja) es la mejor distribución? ¿Por qué no es D3 (en rosa)?

En esta situación, 0 es pérdida y 1 es ganancia o victoria. El D5 es el mejor porque está sesgado a la derecha, por lo que tendremos resultados promedio cercanos a 1, lo que significa que tenemos más victorias o ganancias. Y, de hecho, todas las máquinas de casino de hoy en día están cuidadosamente programadas para tener una distribución como D1 o D3. Pero es un buen ejemplo concreto.

6.1.2. UCB en Python

¿Por qué una sola ronda puede tener varios 1 para diferentes anuncios?

Cada ronda corresponde a un usuario que se conecta a una página web, y el usuario solo puede ver un anuncio cuando se conecta a la página web, el anuncio que se le está mostrando. Si hace clic en el anuncio, la recompensa es 1; de lo contrario, 0. Y puede haber varios porque hay varios anuncios en los que al usuario le encantaría hacer clic. Pero solo una bola de cristal nos diría en qué anuncios haría

clic el usuario. Y el conjunto de datos es exactamente esa bola de cristal (aquí estamos haciendo una simulación).

¿Podrías explicar con más detalle qué hacemos con las recompensas en las líneas de código 33y 34?

Vamos a verlo en detalle:

- En la línea 33, solo obtenemos la recompensa en la ronda n específica. Entonces obtenemos un 1 si el usuario hace clic en el anuncio en esta ronda n, y un 0 si el usuario no lo hace.
- En la línea 34 obtenemos la suma de todas las recompensas hasta la ronda n. Entonces, esta suma se incrementa en 1 si el usuario hace clic en el anuncio en esta ronda n, y permanece igual si el usuario no lo hace.

¿Entra realmente en vigor la estrategia UCB durante las primeras rondas?

No al principio, primero debemos tener algunos primeros conocimientos de la respuesta de los usuarios a los anuncios. Ese es solo el comienzo de la estrategia.

En el mundo real, para los primeros diez usuarios que se conectan a la página web, se mostraría el anuncio 1 al primer usuario, el anuncio 2 al segundo usuario, el anuncio 3 al tercer usuario,..., el anuncio 10 al décimo usuario. Entonces comenzaría a actuar el algoritmo.

¿Cuál fue el propósito de este algoritmo? ¿Por qué no podía simplemente contar qué tipo de anuncio convierte más y luego usarlo?

Porque en cada ronda hay un coste (el coste de poner un anuncio en la página). Entonces, básicamente, el propósito de UCB no es solo maximizar los ingresos, sino también minimizar el coste. Y si solo se cuenta qué tipo de anuncio convierte mejor, eso requeriría que se experimentara durante muchas rondas y, por lo tanto, el costo sería más alto y definitivamente no se minimizaría…

6.1.3. UCB en R

¿Cómo obtener este tipo de conjunto de datos?

En la vida real, podríamos hacer esto mediante la transmisión de datos, utilizando Spark o Hive, lo que significa que obtendríamos datos en tiempo real. De lo contrario, si deseas evaluar tus modelos de aprendizaje por refuerzo, puedes simular un conjunto de datos como el que usamos en las clases.

¿Cuál es exactamente el 'number of selections'?

"number of selections" es el número de veces que se muestra el anuncio a un usuario.

No entiendo por qué el algoritmo no selecciona anuncios en las primeras 10 rondas. El instructor dice que no hay una estrategia al principio, por lo que la primera ronda mostrará el anuncio 1, el anuncio 2 de la segunda ronda, el anuncio 3 de la tercera ronda, etc.

Porque en este punto no sabemos nada sobre los anuncios y no podemos aplicar nuestra fórmula porque implica la división por cuántas veces se ha mostrado un anuncio. No podemos dividir por 0…

En Python, el anuncio ganador fue el anuncio no. 4 y en R fue el anuncio no. 5. Sin embargo, se ha utilizó el mismo archivo `Ads_CTR_Optimisation.csv`. ¿Por qué estamos obteniendo dos resultados diferentes usando la misma lógica?

Porque los índices en Python comienzan desde 0 y los índices en R comienzan desde 1. Por lo tanto, Python y R se obtienen el mismo anuncio.

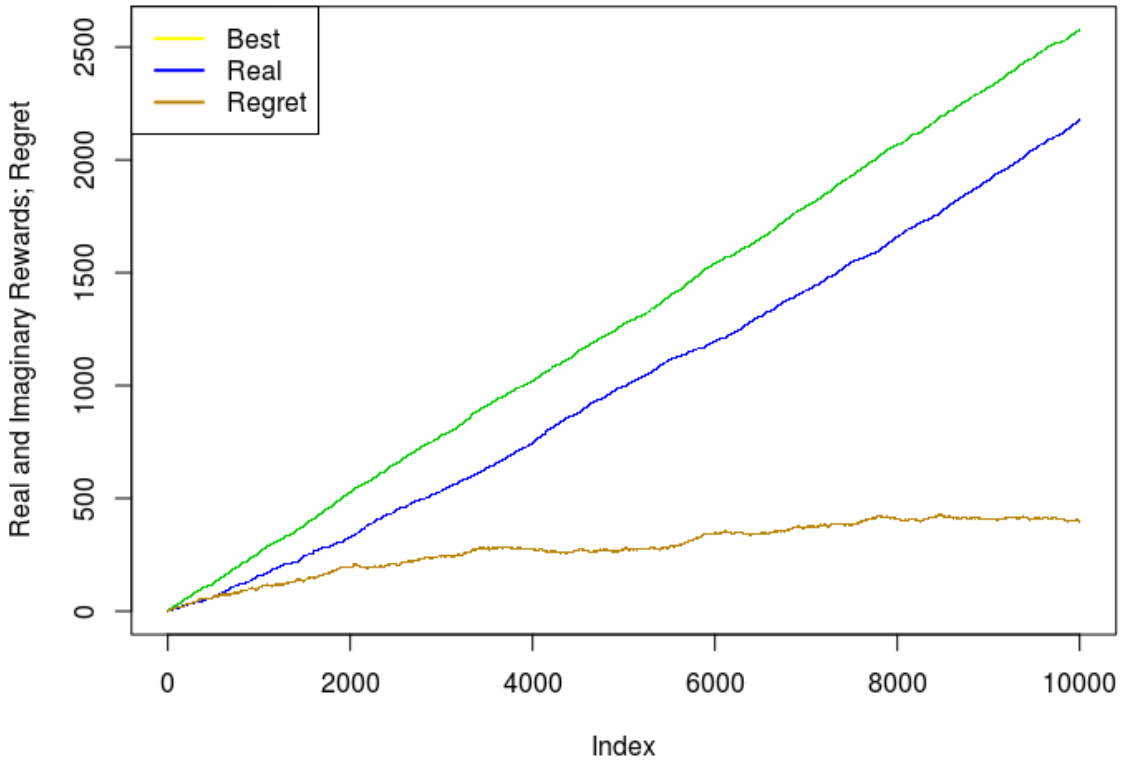
¿Podrías mostrar una implementación en R de la curva de arrepentimiento del UCB?

```
# Importando el conjunto de datos
dataset = read.csv('Ads_CTR_Optimisation.csv')

# Implementar UCB
N = 10000
d = 10
ads_selected = integer(0)
numbers_of_selections = integer(d)
sums_of_rewards = integer(d)
total_reward = 0
rewards_at_each_step=integer(0)
best_selection=(rowSums(dataset)==0) # Línea nueva
best_rewards_at_each_step=integer(0) # Línea nueva
for (n in 1:N) {
  ad = 0
  max_upper_bound = 0
  for (i in 1:d) {
    if (numbers_of_selections[i] > 0) {
      average_reward = sums_of_rewards[i] / numbers_of_selections[i]
      delta_i = sqrt(3/2 * log(n) / numbers_of_selections[i])
      upper_bound = average_reward + delta_i
    } else {
        upper_bound = 1e400
    }
    if (upper_bound > max_upper_bound) {
      max_upper_bound = upper_bound
      ad = i
    }
  }
  ads_selected = append(ads_selected, ad)
  numbers_of_selections[ad] = numbers_of_selections[ad] + 1
  reward = dataset[n, ad]
  sums_of_rewards[ad] = sums_of_rewards[ad] + reward
  total_reward = total_reward + reward
  rewards_at_each_step=c(rewards_at_each_step, total_reward) # Línea nueva
  best_rewards_at_each_step[n]=sum(best_selection[1:n]) # Línea nueva
}

# Curva de Arrepentimiento
plot(best_rewards_at_each_step, pch='.', col=3,
    main="Real and Imaginary Rewards; Regret",
    ylab="Reward Numbers")
points(rewards_at_each_step, pch=".", col=4)
points((best_rewards_at_each_step-rewards_at_each_step), pch='.', col="darkgoldenrod")
legend('topleft', legend=c("Best", "Real", "Regret"),
    col=c(7,4, "darkgoldenrod"),  horiz=F, lty=1, lwd=2)
```

Y obtenemos las siguientes curvas:



6.2. Muestreo Thompson

6.2.1. Intuición del Muestreo Thompson

¿Por qué la marca amarilla es la mejor opción y no la marca verde?

La marca amarilla es la mejor opción porque es la más alejada del origen en el eje x, lo que significa que tiene el retorno estimado más alto.

¿Por qué es el muestreo Thompson mejor que UCB?

El muestreo Thompson es mejor que UCB en términos de convergencia de la curva de arrepentimiento. El arrepentimiento es la diferencia entre la recompensa óptima y la recompensa que acumula con nuestro algoritmo. El muestreo Thompson muestra una mejor curva de arrepentimiento que UCB en mi experiencia. Además, el hecho de que UCB sea determinista en lugar de que el muestreo Thompson sea estocástico, ayuda a que el muestreo Thompson supere a UCB. Además en las secciones prácticas

se ve que el muestro Thompson es capaz de encontrar el mejor anuncio más rápido y con más certeza que UCB.

No entiendo cómo Thompson Sampling puede aceptar comentarios retrasados.

Al hacer el muestreo de Thompson, aún podemos realizar actualizaciones en nuestro algoritmo (como hacer nuevas conjeturas para las distribuciones con datos existentes, muestrear de la distribución adivinada, etc.) mientras esperamos los resultados de un experimento en el mundo real. Esto no obstaculizaría el funcionamiento de nuestro algoritmo. Es por eso que puede aceptar retroalimentación retrasada.

¿Cuáles son otros ejemplos de aplicaciones de muestreo de Thompson?

Otra aplicación potencial del problema delbandido multibrazo puede ser la prueba en línea de algoritmos.

Por ejemplo, supongamos que tenemos un sitio web de comercio electrónico y tiene a su disposición varios algoritmos de aprendizaje automático para proporcionar recomendaciones a los usuarios (de lo que sea que venda el sitio web), pero no sabemos qué algoritmo conduce a las mejores recomendaciones…

Podemos considerar el problema como un problema de UCB y definir cada algoritmo de aprendizaje automático como un "brazo": en cada ronda, cuando un usuario solicita una recomendación, se seleccionará un brazo (es decir, uno de los algoritmos) para hacer las recomendaciones y recibiremos la recompensa. En este caso, podemos definir la recompensa de varias formas, un ejemplo simple es "1" si el usuario hace clic / compra un artículo y "0" en caso contrario. Eventualmente, el algoritmo del bandido multibrazo bandidos convergerá y terminará eligiendo siempre el algoritmo que sea más eficiente para brindar recomendaciones. Ésta es una buena forma de encontrar el modelo más adecuado en un problema en línea.

Otro ejemplo que me viene a la mente es encontrar el mejor tratamiento clínico para los pacientes: cada tratamiento posible podría considerarse como un "brazo", y una forma sencilla de definir la recompensa sería un número entre 0 (el tratamiento no tiene ningún efecto) y 1 (el paciente se cura perfectamente).

En este caso, el objetivo es encontrar lo más rápido posible el mejor tratamiento minimizando el arrepentimiento acumulativo (lo que equivale a decir que desea evitar en la medida de lo posible la selección de tratamientos "malos" o incluso subóptimos durante el proceso).

6.2.2. Muestreo Thompson en Python

¿Dónde puedo encontrar un gran recurso sobre la distribución Beta?

El mejor link es el siguiente: Distribución Beta

El histograma muestra 4 como el mejor anuncio para seleccionar. La barra más alta está en 4 en el eje x. ¿Por qué se indica como 5?

Es solo porque los índices en Python comienzan desde 0. Entonces, el anuncio del índice 4 es el anuncio número 5.

Tengo curiosidad por saber cómo se aplicaría el muestreo de Thompson de forma proactiva al ejecutar esta campaña publicitaria teórica. ¿Repetiríamos el programa en cada ronda (es decir, cada vez se presentaran todas los anuncios al usuario)?

Primero, un ingeniero de datos crea una canalización para leer los datos del sitio web y reaccionar a ellos en tiempo real. Luego, una visita a la web genera una respuesta para volver a calcular nuestros parámetros y elegir un anuncio para la próxima vez que alguien se conecte.

6.2.3. Muestreo Thompson en R

¿Podrías explicar la diferencia entre las distribuciones Bernoulli y Beta?

La distribución de Bernoulli se aplica a probabilidades discretas porque proporciona la probabilidad de éxito de un resultado, mientras que la distribución Beta se aplica a funciones de densidad (continuas), utilizando una función de deensidad. Esa es la principal diferencia. Aquí hay algunos recursos excelentes:

- Distribución de Bernoulli
- Distribución Beta

¿Podrías proporcionar alguna explicación sobre la inferencia bayesiana?

Este es bastante bueno: Inferencia Bayesiana

¿Cómo puede la heurística del muestreo Thomson encontrar rápidamente que el quinto anuncio es el mejor en comparación con la heurística de Upper Confidence Bound?

Es difícil explicar la razón teóricamente, eso requeriría hacer una investigación y escribir una demostración matemática extensa. Pero intuitivamente, podría deberse a que UCB se basa en suposiciones optimistas, mientras que el muestreo de Thompson se basa en probabilidades relevantes a través del enfoque bayesiano.

¿Cómo representar la curva de arrepentimiento de muestreo de Thompson en R?

Básicamente, simplemente necesitas agregar lo siguiente al final de nuestro código:

```
rewards_at_each_step=c(rewards_at_each_step, total_reward)
  best_rewards_at_each_step[n]=sum(best_selection[1:n])
}
plot(best_rewards_at_each_step, pch='.', col=3,
     main="Real and Imaginary Rewards; Regret",
     ylab="Reward Numbers")
points(rewards_at_each_step, pch=".", col=4)
points((best_rewards_at_each_step-rewards_at_each_step), pch='.', col="darkgoldenrod")
legend('topleft', legend=c("Best", "Real", "Regret"),
       col=c(7,4, "darkgoldenrod"),  horiz=F, lty=1, lwd=2)
```

Capítulo 7

Procesamiento de los Lenguajes Naturales

7.1. NLP

7.1.1. Intuición del Procesamieto de los Lenguajes Naturales

¿Por qué el modelo de Bag of Words reemplaza todas las letras mayúsculas por minúsculas?

Muchos lenguajes de programación, en particular Python y R, tratan las mayúsculas y minúsculas como símbolos completamente diferentes. Entonces, si no convertimos todo a minúsculas, debemos tener en cuenta que algunas palabras pueden contener letras en mayúscula e incluir código adicional para ellas. Es mucho más sencillo reemplazar todas las mayúsculas por minúsculas.

¿Qué es la sparsity (escasez)?

La escasez ocurre cuando tiene muchos ceros en la matriz (por lo tanto, se llama matriz dispersa). Entonces, cuando reducimos la dispersión, eso significa que reducimos la proporción de ceros en la matriz.

7.1.2. Natural Language Processing en Python

¿Podrías explicar qué significa el `quoting = 3`?

`quoting = 3` ignorará las comillas dobles en los textos. El número de observaciones en el conjunto de datos no coincide con el número de textos debido a una anomalía de división con las comillas dobles. El problema potencial es que si tienes comillas dobles en un string, este string puede separarse accidentalmente y considerarse como otra entrada sin resultado. Entonces, para asegurarnos de evitar este tipo de situaciónes, ignoramos las comillas dobles usando `quoting = 3` .

¿Cómo podemos descargar todo el material NLTK de una sola vez?

Para descargar todo el material en NLTK de una sola vez, debemos abrir una terminal y ejecutar:

```
pip install -U nltk
```

¿Podrías explicar qué es `PorterStemmer()`?

`PorterStemmer()` aplica el stemming para obtener las raíces de las palabras de tus textos. Entonces, por ejemplo, "amado" se convertirá en "amor" después de la derivación.

¿Cómo se puede hacer NLP en español u otros idiomas?

Comprueba primero si hay algunas clases de NLP específicas para tu propio idioma, como es el caso del español, por ejemplo:

```
from nltk.stem.snowball import SpanishStemmer
stemmer = SpanishStemmer()
```

y para las stop words, incluye además esto en su código:

```
set(stopwords.words('spanish'))
```

¿Por qué cambiamos las reseñas de listas de palabras a strings?

Esto es debido a que el método `fit_transform()` de la clase `CountVectorizer()` solamente funciona con strings.

¿Cuál es la diferencia entre "TF-IDF" y "CountVectorizer"?

La diferencia es que TF-IDF aplica la normalización, a diferencia de CountVectorizer. De lo contrario, es lo mismo, si introducimos el parámetro `normalizer = None`, entonces es muy similar. \ Para usar TF-IDF, es lo mismo que para las otras técnicas de NLP, solo importa la clase que quieras usar (digamos TfidfVectorizer de `sklearn.feature_extraction.text`), luego crea un objeto de esta clase e introduce los parámetros deseas para preprocesar tus textos.

¿Por qué eliminamos "NO", dado que "La corteza es buena" no es lo mismo que "La corteza no es buena"?

También funciona al revés: "La corteza es mala" no es lo mismo que "La corteza no es mala". Por lo tanto, esto no se suele tener en cuenta. Por lo tanto, creemos / esperamos que, en promedio, el número de errores cometidos en los conceptos causados por este cambio sea pequeño en comparación.

Una vez que hayamos terminado con la limpieza y obtenido nuestro modelo de bolsa de palabras, ¿por qué elaboramos un modelo de clasificación?

Estamos clasificando las valoraciones para predecir el resultado de las nuevas valoraciones, exactamente como el análisis de sentimientos. Solo que para hacer estas predicciones en las mejores condiciones, primero debemos aplicar el modelo de la bolsa de palabras. Eso preparará los datos en un formato correcto para que se ajusten al clasificador.

¿Qué modelo parece ser la mejor solución para el desafío de la tarea final?

Definitivamente Naive Bayes, que por cierto generalmente se recomienda probar el primero de todos cuando se hace el procesamiento del lenguaje natural.

7.1.3. Natural Language Processing in R

¿Por qué eliminamos los signos de puntuación del texto? Al principio, parece que no contienen información. Pero, ¿por qué no dejamos que el algoritmo de clasificación decida?

Por lo general, es un balance entre dimensionalidad e información. Eliminamos la puntuación para que el conjunto de datos sea más manejable con menos dimensiones. Además, la puntuación no contiene tanta información discriminatoria como las palabras en los procesos de clasificación habituales. A través de la experiencia, la validación cruzada ha apuntado a eliminar la puntuación con un mejor rendimiento. Sin embargo, sí, podríamos dejar que el algoritmo de clasificación decida. Quizás algunos conjuntos de datos funcionen mejor con la puntuación, pero como regla general, es mejor evitar los signos de puntuación.

La frase “la corteza no es buena” se está transformando en “la corteza es buena” después de eliminar palabras irrelevantes. La valoración está cambiando a positiva de negativa. ¿Puedes aclarar cómo afectará esto al aprendizaje y la predicción?

La palabra “no” no es tan útil como parece. Consideremoss las siguientes 2 valoraciones:

- “Yo no pedí pasta. El pan era bueno”.
- “Pedí pasta. Su pan no era bueno”.

Desde el enfoque de “bolsa de palabras” son lo mismo, aunque su significado es muy diferente.

Por lo general, cuando las personas quieren incluir “no” en una oración, también pueden hacerlo reemplazando la siguiente palabra con un antónimo y eliminan la negación.

¿Existe alguna forma o recursos a través de los cuales se pueda echar un vistazo a la lista de palabras no relevantes?

Puedes verlos si introduces en R el siguiente comando:

```
library(tm)
stopwords("english")
```

¿Cuál sería otra técnica además de la matriz de confusión y la curva CAP para evaluar la eficacia del modelo de NLP?

Otra técnica interesante es la validación cruzada de k-Fold, que veremos en la Parte 10 - Selección de modelos.

Capítulo 8

Deep Learning

8.1. Redes Neuronales Artificiales

8.1.1. Intuición de las Redes Neuronales Artificiales

¿En qué categoría caen las RNA? ¿Aprendizaje por refuerzo, no supervisado o supervisado?

Las RNA pueden ser todas 3. Depende de cómo implemente el modelo. Ejemplos:

- Aprendizaje supervisado: CNN clasificando imágenes en imagenet.
- Aprendizaje no supervisado: máquinas Boltzmann, codificadores automáticos, GAN, DC-GANS, VAE, SOM, etc.
- Aprendizaje por Refuerzo: Q-Learning convolucional profundo que juega videojuegos a partir de entrada de píxeles, AlphaGO, etc. Esta rama se denomina "Aprendizaje por refuerzo profundo" y pertenece a la Inteligencia Artificial.

¿Cómo determina la red neuronal cuáles son los pesos óptimos? ¿Cuándo detiene su aprendizaje?

Calcula los pesos óptimos basándose en un algoritmo de optimización empleado para minimizar la función de pérdida con respecto a todos los puntos de datos de entrada. Se detiene cuando la pérdida de entrenamiento se acerca mucho a 0. Además, generalmente elegimos un cierto número de épocas, que es el número de veces que se entrena la red neuronal. Después de la última época, el entrenamiento ha terminado. También existe lo que llamamos "Detención anticipada", que es una técnica que detiene el entrenamiento una vez que obtenemos una reducción de pérdida demasiado pequeña en un conjunto de validación durante las épocas. En ese caso, el entrenamiento se detiene antes de la última época.

¿Por qué debería reducirse la función de costes?

La función de costes es una estimación numérica de cuán equivocadas son nuestras predicciones de redes neuronales. Si reducimos la función de costes, significa que estamos reduciendo los errores cometidos por la red neuronal, lo que a su vez la hace predecir con mayor precisión.

¿Cómo se calcula el peso de cada neurona?

Al comienzo del entrenamiento, los pesos se inicializan aleatoriamente cerca de cero. Luego, en cada una de las muchas iteraciones (o épocas), los pesos se actualizan a través del descenso de gradiente, en

la dirección que disminuye al máximo el error de pérdida (o función de costo) entre las predicciones y los objetivos.

¿Podrías recapitular el proceso de cada iteración de entrenamiento?

Después de que los pesos se inicializan aleatoriamente con valores cercanos a cero, el entrenamiento pasa por muchas iteraciones (el número de iteraciones, también llamado número de épocas, generalmente se decide en la implementación). Aquí está el proceso de cada iteración:

- Propagación hacia adelante: la entrada se propaga hacia adelante dentro de la red neuronal que al final devuelve la predicción.
- Comparamos esa predicción con el objetivo (el valor real que tenemos porque estamos tratando con el conjunto de entrenamiento) y calculamos el error de pérdida entre la predicción y el objetivo.
- Ese error de pérdida se propaga hacia atrás dentro de la red neuronal.
- A través de Gradiente Descendente o el Gradiente Descendente Estocástico, los pesos se actualizan en las direcciones que reducen al máximo el error de pérdida (ya que de hecho el error de pérdida es una función de costo de los pesos).
- Así obtenemos nuevos pesos, listos para hacer una nueva predicción en la próxima iteración. Luego se repite todo el mismo proceso, hasta que el error de pérdida deja de reducirse (parada anticipada) o hasta que el entrenamiento llega a la última época.

8.1.2. Redes Neuronales Artificiales en Python

¿Por qué una RNA sería un mejor modelo / solución que simplemente usar uno de los diferentes modelos que aprendimos en la Parte 3?

No es necesariamente la mejor solución para todos los problemas. Por ejemplo, si el problema es lineal, definitivamente esta no es la mejor solución, ya que sería mucho más eficiente con un modelo de regresión lineal. Sin embargo, si el problema no es lineal y complejo, entonces un RNA puede ser un mejor modelo / solución que los modelos de la Parte 3. En la Parte 10 - Selección del modelo, aprenderemos a evaluar esto.

¿Por qué usamos Sequential y Dense?

Básicamente es muy simple:

- Sequential se utiliza para inicializar el modelo de aprendizaje profundo como una secuencia de capas (a diferencia de un grafo computacional).
- Dense se usa para agregar una capa de neuronas en la red neuronal.

Recibo algunas advertencias cuando ejecuto la sección de código donde agrego las capas con la función `Dense ()`. ¿Cómo puedo arreglarlo?

Simplemente necesitamos reemplazar:

```
classifier.add(Dense(output_dim = 6,
                     init = 'uniform',
                     activation = 'relu',
                     input_dim = 11))
```

por:

```
classifier.add(Dense(units = 6,
                     kernel_initializer = "uniform",
                     activation = "relu",
                     input_dim = 11))
```

Básicamente, algunos nombres de parámetros cambiaron en la API: `output_dim` se convirtió en `units`, `init` se convirtió en `kernel_initializer` y `nb_epoch` se convirtió en `epochs`. Ocurre con frecuencia en las librerías de todos los lenguajes de programación.

¿Cómo podemos decidir el número de capas ocultas?

No existe una regla general y, en general, más capas ocultas pueden brindarnos una mayor precisión, pero también pueden sobreajustar el modelo, por lo que debemos tener cuidado. La clave es la intuición y la experiencia. También puedes experimentar con el ajuste de parámetros que veremos en la Parte 10 - Selección de modelo.

¿Qué hace la función de activación del rectificador lineal unitario?

Dado que el aprendizaje profundo se utiliza para resolver problemas no lineales, los modelos deben ser no lineales. Y el uso de la función rectificadora es realmente hacerla no lineal. Al aplicarlo, se está rompiendo la linealidad entre las neuronas de salida y las neuronas de entrada.

¿Qué necesito cambiar si tengo una salida no binaria, p. Ej. ganar, perder y empatar?

En ese caso, necesitaríamos crear tres variables ficticias para la variable dependiente:

- $(1, 0, 0) \rightarrow$ ganar
- $(0, 1, 0) \rightarrow$ perder
- $(0, 0, 1) \rightarrow$ empatar

Y, por lo tanto, debe realizarse el siguiente cambio:

```
output_dim = 3 # con la nueva API ahora es: units = 3
activation = 'softmax'
loss = 'categorical_crossentropy'
```

¿`nb_epoch` ha cambiado de nombre?

Sí, en el código necesitamos reemplazar `nb_epoch` por `epochs`.

¿Cómo construir la misma red neuronal para la regresión?

Primero necesitamos importar:

`KerasRegressor`

Entonces es casi lo mismo que una ANN para clasificación. La principal diferencia es la función de compilación, donde, por supuesto, necesitamos cambiar la función de pérdida y tomar una función de pérdida para Regresión como el Error cuadrático medio, y eliminar la métrica de precisión:

```
# Inicializando la RNA
regressor = Sequential()

# Agregar la capa de entrada y la primera capa oculta
regressor.add(Dense(units = 6,
```

```
                    kernel_initializer = 'uniform',
                    activation = 'relu',
                    input_dim = 11))

# Añadiendo la segunda capa oculta
regressor.add(Dense(units = 6,
                    kernel_initializer = 'uniform',
                    activation = 'relu'))

# Agregar la capa de salida
regressor.add(Dense(units = 1,
                    kernel_initializer = 'uniform'))

# Compilar la ANN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Ajuste de la RNA al conjunto de entrenamiento
regressor.fit(X_train,
              y_train,
              batch_size = 10,
              epochs = 100)
```

¿De dónde provienen todos los hiperparámetros? ¿Cómo elegimos el número de capas, el número de neuronas en cada capa y todos los demás parámetros?

A través de mucha investigación y experimentación. No existe una regla general para elegir tales números. Por lo general, pueden encontrarse algunas arquitecturas de redes neuronales listas para usar en línea que demuestran obtener buenos resultados. Pero puedes experimentar ajustando el modelo manualmente con otros valores de parámetros. Y, por último, puede utilizar algunas técnicas de ajuste de parámetros como Grid Search con k-Fold Cross Validation para encontrar los valores óptimos. Aprenderás cómo hacerlo en la Parte 10: Selección de modelo.

¿Cómo puedo mejorar la precisión de la ANN?

Esta es la pregunta del millón. Los científicos de datos han estado tratando de averiguar esta respuesta durante un tiempo. Por lo general, los modelos ANN requieren una buena cantidad de ajustes, generalmente en forma de cambio de número de capas ocultas, cambio de fórmulas y normalización de datos. Por lo general, estas cosas te llevarán más lejos.

8.1.3. Redes neuronales Artificiales en R

¿Por qué usamos `as.numeric(factor())`?

Porque para construir la red neuronal usaremos el paquete H2O que espera las entradas como factores numéricos (o variables categóricas numéricas).

Si elijo hidden $= c(a_1, a_2, a_3..a_n)$, ¿Significa esto que hay n capas ocultas en el modelo y $a_1, a_2, ..., a_n$ son el número de nodos en cada capa oculta?

¡Así es, correcto!

¿El número de nodos en cada capa oculta es siempre el mismo? ¿O podemos especificar diferentes nodos en cada capa oculta? ¿Cómo elegimos estos números?

No, puedes jugar con ellos y ver si puedes obtener mejores resultados. Sin embargo, te recomendamos que observes el administrador de tareas para ver si la memoria de tu ordenador es suficiente para manejar tantas capas y sus nodos. Puede elegirlos experimentando manualmente (prueba y error) o mediante el ajuste de parámetros con búsqueda de cuadrícula (consulta la Parte 10 - Selección de modelo).

En Python aplicamos la función rectificadora a las capas ocultas, pero la función sigmoidea a la capa de resultado. ¿Cómo elegir las mismas opciones con R?

Usando H2O en R, se aplicó la función rectificadora a cada capa. Aquí no existe la opción de elegir diferentes funciones de activación para diferentes capas.

¿Python o R para el aprendizaje profundo?

Python Sin duda. Python es utilizado por los mejores científicos de Deep Learning en la actualidad y tiene librerías increíbles (Tensorflow, Keras, PyTorch) desarrolladas para aplicaciones poderosas (clasificación de imágenes, visión por computadora, inteligencia artificial, chatbots, traducción automática, etc.).

8.2. Redes neuronales convolucionales

8.2.1. Intuición de las Redes neuronales convolucionales

¿Cuáles son las diferencias entre CNN y ANN? ¿La CNN se aplica solo a la clasificación de imágenes?

ANN significa redes neuronales en general. Entonces una CNN es una ANN. Las CNN no solo se utilizan para el procesamiento de imágenes, sino que también se pueden aplicar a texto como la comprensión de texto.

¿Hasta qué punto se reducirá el tamaño de la imagen para el detector de características?

En la sección práctica se reducirá a 64 por 64 dimensiones.

¿Cuál es el propósito de los mapas de características?

Estamos creando mapas de características con cada filtro convolucional, lo que significa que estamos creando características que nos ayudan a clasificar el objeto. El ejemplo que se muestra en la lección de intuición es como un filtro de detección de bordes unidimensional. Ese es un mapa de características. Por lo tanto, queremos que nuestro modelo se “active” en un mapa de características solo donde haya un borde. Tendremos varios mapas de características como este que, cuando estén todos juntos, nos ayudarán a identificar el objeto. Esto se ayuda eliminando el negro o los valores negativos.

¿Cuál es el propósito del ReLU?

La principal razón por la que usamos ReLU es porque queremos aumentar la no linealidad en nuestra imagen. Y ReLU actúa como una función que rompe la linealidad. Y la razón por la que queremos romper la linealidad en nuestra red es porque las imágenes en sí mismas son altamente no lineales. De hecho, tienen muchos elementos no lineales como las transiciones entre píxeles.

¿Por qué Max-Pooling considera solo 4 valores para tomar un máximo y no 2 u 8 valores? También en convolución, ¿cómo se forma el detector de características?

Porque se toma un máximo de un punto en la imagen que está representado por un cuadrado de 2x2 en nuestra imagen. Por lo tanto, cubre 4 píxeles.

Después del aplanamiento, ¿obtendremos un vector largo para todas las capas agrupadas o un vector para cada capa agrupada?

Será un vector largo que reúna todas las capas aplanadas.

8.2.2. Redes neuronales convolucionales en Python

¿Cuántas imágenes se requieren para entrenar a un buen modelo? ¿Existe alguna regla general para esta decisión?

10,000 es un buen número. No hay reglas generales, solo que cuanto más tenga, más posibilidades tendrá de obtener una buena precisión.

¿Podría explicar los números 0, 1, 2, 3 y 4 en los mapas de características?

Cuando el detector de características 3x3 cubre una parte de la imagen de entrada, obtiene:

- 0 si no hay 1 en común entre la subtabla 3x3 de la imagen de entrada y el detector de características 3x3,
- 1 si hay un 1 en común entre la subtabla 3x3 de la imagen de entrada y el detector de características 3x3,
- 2 si hay dos 1 en común entre la subtabla 3x3 de la imagen de entrada y el detector de características 3x3,
- 3 si hay tres 1 en común entre la subtabla 3x3 de la imagen de entrada y el detector de características 3x3,
- 4 si hay cuatro 1 en común entre la subtabla 3x3 de la imagen de entrada y el detector de características 3x3.

¿Podrías repasar la propagación hacia adelante de las imágenes de entrada que ocurren dentro de la CNN describiendo en pocas palabras las clases que usamos en Python?

Claro, aquí está el proceso con las descripciones esenciales de las clases utilizadas:

- Sequential se usa primero para especificar que introducimos una secuencia de capas, a diferencia de un gráfico computacional.
- Luego, se usa Convolution2D para agregar la capa convolucional.
- Luego, MaxPooling2D se usa para aplicar Max Pooling a las imágenes de entrada.
- A continuación, se utiliza Flatten para aplanar las imágenes agrupadas.
- Dense se usa para agregar la capa de salida con softmax.

¿Cómo hacer una sola predicción de si una imagen específica contiene un gato o un perro?

Dentro de la carpeta del conjunto de datos, tenemos que crear una carpeta adicional separada (llamémosla “single_prediction”) que contenga la imagen (llamémosla “cat_or_dog.jpg”) que deseamos predecir y ejecutar el siguiente código:

```
import numpy as np
from keras.preprocessing import image as image_utils
test_image = image_utils.load_img('dataset/single_prediction/cat_or_dog.jpg',
                                  target_size = (64, 64))
test_image = image_utils.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict_on_batch(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
```

Capítulo 9

Reducción de la Dimensión

9.1. Análisis de Componentes Principales (ACP)

9.1.1. Intuición del ACP

¿Cuál es el verdadero propósito de la ACP?

El verdadero propósito es principalmente disminuir la complejidad del modelo. Se trata de simplificar el modelo manteniendo la relevancia y el rendimiento. A veces, puedes tener conjuntos de datos con cientos de características, por lo que, en ese caso, solo deseamos extraer muchas menos variables independientes que expliquen al máximo la varianza.

¿Cuál es la diferencia entre ACP y análisis factorial?

El análisis de componentes principales implica la extracción de compuestos que sean combinaciones lineales de variables observadas. El análisis factorial se basa en un modelo formal que predice las variables observadas a partir de factores latentes teóricos. El ACP está destinado a maximizar la varianza total para buscar patrones distinguibles, y el análisis factorial busca maximizar la varianza compartida para constructos o variables latentes.

¿Debería aplicar PCA si mi conjunto de datos tiene variables categóricas?

Podrías probar PCA, pero yo sería muy cuidadoso, porque los valores categóricos pueden tener altas variaciones de forma predeterminada y, por lo general, serán inestables a la inversión de la matriz. Aplica ACP y realiza una validación cruzada para ver si el ACP puede generalizar mejor que los datos reales. Si es así, entonces ACP es bueno para el modelo. (Nuestra matriz de entrenamiento es numéricamente estable). Sin embargo, estoy seguro de que en la mayoría de los casos, ACP no funciona bien en conjuntos de datos que solo contienen datos categóricos. El ACP sin más está diseñado en base a capturar la covarianza en variables continuas. Hay otros métodos de reducción de datos que puedes intentar para comprimir los datos, como análisis de correspondencia múltiple y ACP categórico, etc.

¿Cuál es el mejor recurso adicional aprender sobre ACP?

Mira este video que tiene una explicación asombrosa del ACP y lo estudia en más profundidad.

9.1.2. PCA en Python

¿Qué hace `fit_transform` aquí? ¿Por qué aplicamos fit_transform al conjunto de entrenamiento y solo transformamos el conjunto de prueba?

En el método `fit_transform` hay ajuste y transformación. La parte de ajuste se utiliza para analizar los datos sobre los que aplicamos el objeto (obteniendo los valores propios y los vectores propios de la matriz de covarianza, etc.) con el fin de obtener la información requerida para aplicar la transformación ACP, es decir, extrayendo algunos características principales que explican al máximo la variación. Luego, una vez que el objeto obtiene esta información gracias al método de ajuste, la parte de transformación se usa para aplicar la transformación del ACP. Y dado que el conjunto de prueba y el conjunto de entrenamiento tienen estructuras muy similares, no necesitamos crear un nuevo objeto que ajustemos al conjunto de prueba y luego usarlo para transformar el conjunto de prueba, podemos usar directamente el objeto ya creado y ajustado. al equipo de entrenamiento, para transformar el conjunto de prueba.

¿Cómo averigua cuáles de las variables independientes son los 2 componentes principales?

PCA es una técnica de extracción de características, por lo que los componentes no son una de las variables independientes originales. Estos son nuevos, como una especie de transformación de los originales. \ Es solo con la selección de características que terminas con una de las variables independientes originales, como con Eliminación hacia atrás.

¿Es mejor utilizar la extracción de características o la selección de características, o ambas? En caso de ser ambos, ¿en qué orden?

La extracción de características y la selección de características son dos excelentes técnicas de reducción de dimensionalidad y, por lo tanto, siempre debe considerar ambas.

Lo que recomiendo es hacer primero la selección de características para mantener solo las características relevantes, y luego aplicar la extracción de características en estas características relevantes seleccionadas para reducir aún más la dimensionalidad de su conjunto de datos mientras mantiene suficiente variación.

¿Cuánta razón de varianza total necesitamos usar? ¿Existe algún umbral para una buena relación de varianza total?

Generalmente, un buen umbral es del 50 %. Pero el 60 % es más recomendable.

¿Es más común usar exactamente 2 variables independientes para construir un clasificador, o la gente suele usar más que eso?

En general, la gente simplemente extrae una serie de variables independientes que explican una proporción suficiente de la varianza (normalmente el 60 %). Entonces no siempre son dos. Puede ser más. Y si son dos eso es genial porque entonces puedes visualizar mejor.

¿Existe un método o atributo de Python que pueda ayudar a proporcionar los componentes subyacentes y los signos de los dos componentes principales PC1 y PC2?

Puedes acceder a ellos con components_ attributes:

```
components = pca.components_
```

9.1.3. PCA en R

Si quisiera ver los valores reales de todas las columnas, ¿cómo desescalaría los datos que se presentaron escalados?

Digamos que el vector escalado era `y_train`, luego, para cambiar la escala de su resultado `y_pred`, haga lo siguiente:

```
y_pred_unscaled = y_pred*attr(y_train,'scaled:scale') + attr(y_train, 'scaled:center')
```

¿Cómo puedo ver los componentes principales en R?

Puedes ejecutar el siguiente código:

```
pca$rotation
```

Tenemos una precisión del 100 %, ¿no deberíamos preocuparnos por el sobreajuste?

Si observamos los datos, veremos que es casi perfectamente separable por las 3 líneas dibujadas, lo que significa que la separabilidad es una característica de los datos más que un problema de sobreajuste. Si tuvieramos algo así como 50 líneas y una precisión del 100 % (es decir, cada sección capturaría precisamente una pequeña cantidad de puntos, lo que garantiza que estén correctamente clasificados), entonces probablemente sería un problema de sobreajuste, pero aquí claramente no es el caso. Además, obtuvimos una precisión del 100 % solo en el conjunto de prueba. Hubo algunos puntos mal clasificados en el conjunto de entrenamiento. Y el sobreajuste es más bien lo contrario: una precisión casi perfecta en el conjunto de entrenamiento y una deficiente en el conjunto de prueba.

¿Cómo podemos saber cuáles son las dos variables que se toman para la representación gráfica?

Las dos variables no se encuentran entre sus variables independientes originales, ya que fueron extraídas (Extracción de características), en lugar de ser seleccionadas (Selección de características). Las dos nuevas variables son las direcciones donde hay mayor variación, es decir, las direcciones donde los datos están más dispersos.

9.2. Análisis Discrimiante Lineal (LDA)

9.2.1. Intuición del LDA

¿Podrías explicar de una manera más sencilla la diferencia entre PCA y LDA?

Una forma sencilla de ver la diferencia entre PCA y LDA es que PCA trata todo el conjunto de datos como un todo, mientras que LDA intenta modelar las diferencias entre clases dentro de los datos. Además, PCA extrae algunos componentes que explican más la variación, mientras que LDA extrae algunos componentes que maximizan la separabilidad de clases.

¿Selección de características o extracción de características?

Preferirías elegir la selección de características si deseas mantener toda la interpretación de tu problema, el conjunto de datos y los resultados de su modelo. Pero si no te importa la interpretación y solo deseas obtener predicciones precisas, puedes probar ambos, por separado o juntos, y comparar los resultados de rendimiento. Entonces, sí, la selección de características y la extracción de características se pueden aplicar simultáneamente en un problema dado.

¿Podemos usar LDA para la regresión?

LDA es análisis discriminante lineal. Es una generalización del discriminante lineal de Fisher, un método utilizado en estadística, reconocimiento de patrones y aprendizaje automático para encontrar una combinación lineal de características que caracteriza o separa dos o más clases de objetos o eventos. La combinación resultante puede usarse como clasificador lineal o, más comúnmente, para la reducción de dimensionalidad antes de una clasificación posterior. Sin embargo, para la regresión, tenemos que usar ANOVA, una variación de LDA. LDA también está estrechamente relacionado con el análisis de componentes principales (PCA) y el análisis factorial, ya que ambos buscan combinaciones lineales de variables que expliquen mejor los datos. LDA intenta explícitamente modelar la diferencia entre las clases de datos. La PCA, por otro lado, no tiene en cuenta ninguna diferencia de clase, y el análisis factorial construye las combinaciones de características basadas en diferencias en lugar de similitudes. El análisis discriminante también se diferencia del análisis factorial en que no es una técnica de interdependencia: se debe hacer una distinción entre variables independientes y variables dependientes (también llamadas variables de criterio). LDA funciona cuando las mediciones realizadas en variables independientes para cada observación son cantidades continuas. Cuando se trata de variables independientes categóricas, la técnica equivalente es el análisis de correspondencia discriminante.

9.2.2. LDA en Python

¿Qué variables independientes se encuentran después de aplicar LDA?

Las dos variables independientes que vemos, indexadas por 0 y 1, son nuevas variables independientes que no se encuentran entre sus 12 variables independientes originales. Estas son variables independientes totalmente nuevas que se extrajeron a través de LDA, y es por eso que llamamos Extracción de características de LDA, en lugar de Selección de características, donde matenemos algunas de las variables independientes originales.

¿Cómo decidir el parámetro LDA `n_component` para encontrar el resultado más preciso?

Podemos ejecutar:

```
LDA(n\_components = None)
```

y debería darnod automáticamente los `n_components` ideales.

¿Cómo puedo obtener los dos discriminantes lineales LD1 y LD2 en Python?

Puedes obtenerlos ejecutando la siguiente línea de código:

```
lda.scalings_
```

9.2.3. LDA en R

¿Cómo puedo obtener los dos discriminantes lineales LD1 y LD2 en R?

Puedes obtenerlos ejecutando la siguiente línea de código:

```
lda$scaling
```

No entiendo por qué la versión R de LDA eligió 2 variables independientes automáticamente.

Porque en R, cuando tienes k clases, $k-1$ es igual al número de discriminantes lineales que obtienes.

¿Por qué obtenemos una matriz en LDA pero obtuvimos un marco de datos en PCA, aunque el procedimiento es similar?

Esto se debe simplemente a una diferencia en el formato que proporciona la función como salida. Estos procedimientos son similares independientemente del formato de salida.

9.3. Kernel ACP

9.3.1. Intuición del Kernel ACP

¿Debería utilizarse Kernel ACP para convertir datos separables no linealmente en datos separables linealmente?

Así es, pero no es necesario utilizar Kernel ACP con un clasificador no lineal, ya que los datos serán linealmente separables después de aplicar Kernel ACP y, por lo tanto, un clasificador lineal será suficiente.

¿Cuándo deberíamos usar ACP vs Kernel ACP?

Deberías comenzar con ACP. Luego, si obtiene malos resultados, prueba Kernel ACP.

9.3.2. Kernel PCA en Python

¿Cómo sé si mis datos son linealmente separables o no?

Un buen truco es entrenar primero un modelo de regresión logística. Si obtenemos una precisión realmente buena, debería ser (casi) linealmente separable.

¿Existe una gran diferencia y qué es mejor usar entre Kernel ACP + SVM vs ACP + Kernel SVM?

Sí, hay una diferencia. Utilizamos Kernel PCA + SVM cuando necesitamos transformar los datos en una variedad no lineal de baja dimensión donde los puntos son separables. Utilizamos PCA + Kernel SVM cuando necesitamos transformar los datos a través de una transformación lineal en una variedad de baja dimensión, utilizando estos puntos para transformarlos en un espacio no lineal donde son separables.

¿Cómo decidimos qué kernel es mejor para Kernel ACP?

El RBF Kernel es un gran kernel y es la mejor opción en general. Pero la mejor manera de averiguar qué kernel necesitamos aplicar es hacer algunos ajustes de parámetros con Grid Search y k-Fold Cross Validation. Lo veremos en la Parte 10 - Selección del modelo.

9.3.3. Kernel ACP en R

¿Cómo puedo obtener los componentes principales de Kernel ACP en R?

Simplemente necesitas ejecutar la siguiente línea de código:

```
attr(kpca, "pcv")
```

Capítulo 10

Selección de Modelos y Boosting

10.1. Validación Cruzada k-Fold Cross Validation

10.1.1. Intuición de la validación cruzada k-fold

¿Qué es sesgo / varianza bajo / alto?

Estos conceptos son importantes para comprender la validación cruzada de k-Fold:

- Bajo Sesgo es cuando las predicciones del modelo están muy cerca de los valores reales.
- Alto Sesgo es cuando las predicciones del su modelo están lejos de los valores reales.
- Baja varianza: cuando ejecutamos el modelo varias veces, las diferentes predicciones de sus puntos de observación no variarán mucho.
- Alta varianza: cuando ejecutamos el modelo varias veces, las diferentes predicciones de sus puntos de observación variarán mucho.

¿La validación cruzada de k-Fold mejora el modelo o es solo un método de validación?

La validación cruzada de k-Fold se utiliza para evaluar su modelo. No necesariamente mejora el modelo, pero mejora la comprensión del modelo. Sin embargo, puedes usarlo para mejorar el modelo combinándolo con algunas técnicas de ajuste de parámetros como la búsqueda de cuadrícula (siguiente sección).

¿Cuál es la diferencia entre un parámetro y un hiperparámetro?

Los hiperparámetros y los parámetros son muy similares pero no exactamente lo mismo. Un parámetro es una variable configurable que es interna de un modelo cuyo valor se puede estimar a partir de los datos. Un hiperparámetro es un valor configurable externo a un modelo cuyo valor no puede ser determinado por los datos, y que estamos tratando de optimizar (encontrar el valor óptimo) a través de técnicas de ajuste de parámetros como la búsqueda aleatoria o la búsqueda de cuadrícula.

¿Cuál es un valor bueno / el mejor de k para elegir al realizar la validación cruzada de k-Fold?

Normalmente se recomienda en torno a 10.

10.1.2. k-Fold Cross Validation en Python

Me preguntaba si necesitábamos dividir los datos en X_train, y_train y X_test, y_test para aplicar el método de remuestreo (k-Fold) o ¿podemos aplicarlo directamente en X e y?

Puedes hacer ambas cosas. El buen método es:

- Dividir el conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba.
- Realizar una validación cruzada de k-fold en el conjunto de entrenamiento.
- Realizar la evaluación final de su modelo seleccionado en el conjunto de prueba.

Pero también podemos realizar k-fold Cross-Validation en todo el conjunto de datos (X, y).

¿Qué nos dice exactamente esta desviación estándar?

La desviación estándar de la precisión del modelo simplemente muestra que la varianza de la precisión del modelo es del 6 %. Esto significa que el modelo puede variar alrededor del 6 %, lo que significa que si ejecuto mi modelo con datos nuevos y obtengo una precisión del 86 %, sé que esto tiene una precisión del 80-92 %. El sesgo y la precisión a veces no tienen una relación obvia, pero la mayoría de las veces puede detectar algún sesgo en la validación o prueba de su modelo cuando no funciona correctamente con datos nuevos.

¿Cómo calcular el Recall, la Precision o el F1 Score a partir de la validación cruzada de k-Fold?

Puedes usar la librería de métricas de sklearn para esto. Aquí hay un link.

¿Por qué solo Kernel SVM?

Podemos aplicar k-Fold Cross Validation a cualquier modelo de Machine Learning, y sería muy inteligente hacerlo para cada modelo que creemos Kernel SVM fue solo un ejemplo aquí para explicar cómo aplicar k-Fold Cross Validation.

10.1.3. k-Fold Cross Validation en R

Usamos la matriz de confusión para determinar la precisión de cada pliegue (fold). ¿Qué usarías para los modelos de regresión? ¿R-cuadrado?

No, preferimos usar el MSE (error cuadrático medio), que mide la suma de las diferencias cuadráticas entre los resultados reales y las predicciones.

El ejemplo de validación cruzada de k-Fold en R crea 10 modelos diferentes usando el conjunto de entrenamiento. ¿Cómo puede la media de 10 modelos diferentes evaluar el modelo original? ¿Necesitamos siquiera el modelo original?

A través de la validación cruzada de k-Fold, necesitamos estimar el "error imparcial" en el conjunto de datos de este modelo. La media es una buena estimación de cómo funcionará el modelo en el conjunto de datos. Luego, una vez que estamos convencidos de que el error promedio es aceptable, entrenamos el mismo modelo en todo el conjunto de datos.

En R creamos pliegues a partir del conjunto de entrenamiento. ¿No deberíamos crearlos para todo el conjunto de datos? ¿Por qué solo el set de entrenamiento?

Solo usamos el set de entrenamiento para hacer nuestro modelo. El conjunto de prueba se considera algo que aún no tenemos con nosotros. (Aunque lo hagamos). Entonces usamos el conjunto de entrenamiento

para hacer el mejor modelo y luego evaluamos su desempeño en el conjunto de prueba. Si ajustamos el rendimiento en el conjunto de prueba, entonces tenemos un modelo sesgado para funcionar bien en el conjunto de prueba, que es solo una instancia muy pequeña de todos los datos. Necesitamos que nuestro modelo funcione bien para los datos que aún no hemos visto.

10.2. Grid Search

10.2.1. Grid Search en Python

No puedo importar `GridSearchCV` en mi ordenador ¿Alguna otra opción para este paquete?

Depende del sistema y la versión del paquete, pero intenta reemplazar:

```
from sklearn.model_selection import GridSearchCV
```

por:

```
from sklearn.cross_validation import GridSearchCV
```

¿Qué es este parámetro de penalización C?

El parámetro de penalización C es un parámetro de regularización que puede permitirle hacer dos cosas:

- reducir el sobreajuste,
- anular valores atípicos.

Es igual a $\frac{1}{\lambda}$ en donde λ es el parámetro de regularización clásico usado en las Regresiones Ridge, Lasso, Elastic Net…

¿Se puede extender Grid Search a ANN también?

Sí, la búsqueda en cuadrícula también es posible en ANN. Lo cubrimos en nuestro curso Deep Learning de la A a la Z.

¿Cómo sabemos qué valores debemos probar en Grid Search?

Un buen comienzo es tomar valores predeterminados y experimentar con valores a su alrededor. Por ejemplo, el valor predeterminado del parámetro de penalización C es 10, por lo que algunos valores relevantes para probar serían 1, 10 y 100.

En esta sección de Python, estamos ajustando parámetros en Grid Search para producir la precisión óptima en el conjunto de entrenamiento con k-Fold Cross Validation? ¿Cómo podemos asegurarnos de que estos parámetros serán los más óptimos cuando trabajemos con el equipo de prueba?

Estos parámetros serán los más óptimos cuando se trabaja con el conjunto de prueba porque demostraron ser óptimos en diez conjuntos de prueba diferentes (los diez conjuntos de prueba se pliegan en el proceso de validación cruzada).

10.2.2. Grid Search en R

¿Cómo usar el valor C y sigma calculado en Grid Search?

Una vez que haya encontrado el valor C y sigma óptimo, puede poner estos valores en los argumentos del modelo SVM y obtener una precisión óptima para su modelo.

¿Cuáles son las ventajas y desventajas de usar el paquete caret en R?

Ventajas de caret:

- Al hacer Grid Search, no tenemos que elegir valores de hiperparámetros para probarlo usted mismo, encontrará algunos óptimos para usted.
- Tiene muchas otras funciones además de ser un conveniente contenedor de métodos de aprendizaje automático.
- Tiene su propia división en conjuntos de entrenamiento y prueba y contiene una función que crea división con series de tiempo.
- También tiene herramientas para la limpieza de datos, y todas tienen una implementación muy eficiente.
- Si lo usamos para ajustar un modelo, puede hacer bootstrapping de los datos para mejorar el modelo.

Desventajas:

- Es un paquete grande y ocupa una parte de la memoria.
- La función de bootstrapping es muy intensiva en computación.

En conclusión, caret es un paquete R increíble para la selección de modelos.

¿Qué es este valor de Kappa que vemos en R?

Ese es el kappa de Cohen. Es una métrica que se usa para evaluar el desempeño de un clasificador y también ayuda a evaluar los clasificadores entre ellos. Para obtener más información, consulta esta buena referencia.

Al hacer Grid Search en R, ¿no podemos saber si los datos son linealmente separables o no, como en Python?

Claro, al igual que en Python, puede ver qué Kernel SVM le brinda la mejor precisión usando Grid Search con signo de intercalación. Si el Kernel es lineal, entonces sus datos probablemente sean linealmente separables. Y si el Kernel no es lineal, es probable que los datos no se puedan separar linealmente.

10.3. XGBoost

10.3.1. Intuición del XGBoost

¿Podrías proporcionar una buena fuente que explique cómo funciona XGBoost?

Recomendamos que leer primero sobre los algoritmos de gradient boosting mediante este excelente enlace. Entonces recomendamos ir al siguiente enlace para comprender XGBoost.

10.3.2. XGBoost en Python

Tengo problemas para instalar XGBoost. ¿Cómo puedo instalarlo fácilmente?

Cuando se hizo la clase, no había ningún comando de instalación de conda. Pero ahora existe y es mucho más sencillo instalar el paquete XGBoost. Simplemente debe introduce el siguiente comando dentro de una terminal (o anaconda prompt para usuarios de Windows):

```
conda install -c conda-forge xgboost
```

¿Se puede aplicar XGBoost tanto a la clasificación como a la regresión en Python?

Absolutamente. Para la clasificación, usa la clase XGBClassifier. Y para Regression, usa la clase XGBRegressor.

10.3.3. XGBoost en R

En la sección de código de validación cruzada de k-Fold, línea 35, ¿no debería ser 'training_fold' en lugar de 'training_set'?

¡En efecto! Gracias por darte cuenta. Por supuesto que es el pliegue del entrenamiento. Nuestras disculpas por el error. Por lo tanto, la sección de código correcta es:

```
classifier = xgboost(data = as.matrix(training_fold[-11]),
                     label = training_fold$Exited,
                     nrounds = 10)
```

¿Se puede aplicar XGBoost tanto a la clasificación como a la regresión en R?

Absolutamente. Y usa la misma función `xgboost()` para Clasificación y Regresión.