08-06-2021

Algoritmos de clasificación(aprendizaje supervisado) . Variables categóricas

Clasificación binaria

Regresión logística. No se considera regresión sino clasificación. La variable que se predice es categórica las que se utilizan para predecir pueden ser categóricas o no

Es un tipo de análisis de regresión para predecir variables categóricas. Valores entre 0 y 1. Nos da una probabilidad

Max\_iter=700 🡪 número de iteraciones que te permitan encontrar el mínimo de error

Las categorías han de pasarse a número

Validación cruzada: obtener un conjunto de entrenamiento independiente (extraído del conjunto de entrenamiento total) y conocer el error que se comete en cada conjunto de validación de la iteración. Se utiliza para evitar el sobreentrenamiento

En el sobreaprendizaje los errores de entrenamiento continúan bajando y los de validación después de ir bajando vuelven a subir porque se empieza a aprender detalles no relevantes en el aprendizaje que nos interesa

Matriz de confusión

Falsos positivos y negativos. Nos permite identificar en que sentido se está equivocando el modelo creado

Entrenamiento y clasificación de dígitos con regresión logística

Si el score para los datos de le muestra de train es bastante mayor que para la de test quiere decir que está sobreentrenado. Es un modelo válido para los valores de train pero no es un modelo generalista.

10-06-2021

Para ver como de buenas o malas son tus métricas se crea un modelo “baseline”

r2 es la varianza explicativa. Si se acerca a 1 el modelo explica correctamente. Ha podido predecir correctamente todas las predicciones. El score en muchas ocasiones utiliza la r2 (r cuadrática)

RMSE: si está por debajo de 1, cuanto más se aproxima al 1, hay menos errores. Si hay mucha diferencia ente el MAE y el RMSE a favor del RMSE puede haber outliers que no recoge el MAE

RMSLE.- se utiliza cuando la variable target se pasa a logaritmo (por ser valores muy altos). Se calcula con: metrics.mean.squared\_log\_error. No se puede utilizar si en la variable hay valores negativos ( a estos valores no se les puede aplicar el logaritmo neperiano)

Cuanto más se aproxime mi algoritmo de regresión a la distribución normal mejor está haciendo las predicciones.

Cuando el valor de r2 o de la varianza explicativa es negativo cuanto más cerca de 1, menos confusión/aleatoriedad. Implica que el modelo no es explicativo

Pasar columnas categóricas a números

from sklearn import preprocessing

le=preprocessing.LabelEncoder()

le.fit()

le.transform(data) 🡪 pasa una variable categórica a numérica. Tb se puede utilizar le.fit\_transform(data)

le.inverse.transform(data)🡪 pasa la variable que hemos transformado a numérica, de nuevo, a categórica. Le podemos solicitar con el valor numérico que nos identifique a qué categoría corresponde

Pasar dos listas a diccionario

dic(zip(columna\_1, columna\_2))

Integración continua del equipo de proyecto

Cada una de las partes del proyecto es un epic branch

Las tareas más pequeñas del epic branch son los tickets

PR.- pull request. Se envía a la epic branch que corresponda y habrá alguien que supervise lo que se ha transmitido al epic Branch. Cuando se hayan enviado todo los tickets a epic Branch se envía a develop donde alguien comprueba que todo es correcto. Una vez que se haya subido todos los tickets a los diferentes epic Branch y subidos a develop se sube todo a master donde varias personas supervisarán que todo funciona correcto

Algoritmo knn (k nearest neighbors). Es un algoritmo de clasificación. Aprendizaje supervisado. Es no paramétrico

Para pocos datos y que no las variables no sigan una distribución normal

No aprende patrones entrenados sino de los datos de entrenamiento.

Elige a que clase pertenece el nuevo elemento calculando la distancia euclídea al elemento del numero de elementos que le decimos que utilice (ej. k=3. A tres elementos de cada una de las clases). El que tenga más elementos con menor distancia al que se predice es el que se elige. Si están empatados decide en función del centroide de cada clase de elementos más cercano al elemento del que se ha de predecir la clase. Si aún así es igual la distancia del centroide al elemento que se predice toma una decisión al azar.

Podemos obtener la distancia a la que se encuentran los vecinos del elemento que queremos predecir. Nos da la distancia y los elementos que son.

Fiabilidad del modelo: neigh.predict.prob

Distinto algoritmo al KNN: nearestNeighbors

metric.accuracy\_score(y\_test,y\_predict)

knn.score (x\_train, y\_train)

Hay una función para probar múltiples números de K

random\_state = 35 🡪 semilla

otro algoritmo: SVM (support vector machine). Para datos no lineares. Vale tanto para clasificación como para regresión. Se utiliza más para clasificación. Suele dar buenos resultados para clasificación

SVC.- support vector clasification

SVR.- support vector regression

Genera un hiperplano que sea el que menor error tenga, es decir que permita segregar mejor las categorías.

Cuando no se puede utilizar un hiperplano recto se utilizan los kernel (caso de que las categorías se sitúan en círculos concéntricos). El kernel sitúa un conjunto de datos en una dimensión superior. Cambiaría la proyección de los datos

Link de interés:

<https://www.kaggle.com/prashant111/svm-classifier-tutorial>

<https://medium.com/pursuitnotes/support-vector-regression-in-6-steps-with-python-c4569acd062d>

**Git branches**

Hacer todo esto en un nuevo repositorio

Se puede crear una rama secundaria a partir de una rama principal (main) a la que se trasladará lo que tenemos desde la que exportamos. Si añadimos algo nuevo a la rama principal, al crear una rama secundaria 2 nos trasladará lo nuevo creado en la rama principal

De la rama original podemos crear cuantas ramas queramos en las que se incluirán los nuevos contenidos que en cada momento se hayan incluido en la rama principal

Para incluir en la rama principal lo nuevo generado en alguna de las secundarias hacemos merge

Hay que tener mucho cuidado con que rama se sube. Si se sube a la principal una versión anterior a la última en la que se han hecho cambios. Se subirá a la principal es versión anterior sin modificaciones. Puedes seleccionar la rama que trasladas a la principal.

Hacer en el ticket Branch git add y git commit antes de hacer merge. Una vez que se hace merge hay que hacer de nuevo git add y git commit

Giti pull y git push son de una rama a la misma rama. Con merge es de una rama a otra

Conflicto: si generamos un cambio en epic Branch directamente y a la vez se hace ese cambio en ticket Branch. Tenemos que decidir con cuales de estos cambios nos quedamos. Una vez resuelto, de nuevo git add y git commit

Siendo epic Branch (origin) hacemos git push origin y nombre de la rama

Una vez hecho esto se hace una pull request (petición de pull)

Desde github hacer la pull request desde epic branch a develop (alguien ha de supervisar el cambio que se solicita admitir para pasar a develop)

No creamos ninguna rama desde github