Uso de Transformers en Scikit-Learn

Alfonso Tobar Arancibia

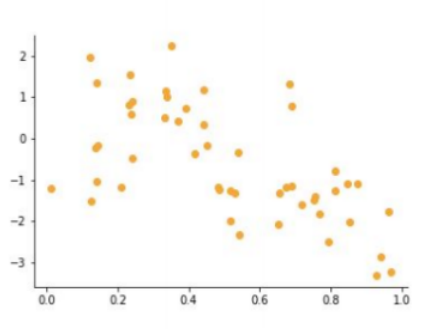
Data Scientist

20-05-2021

github.com/datacubeR/clases-ml/Slides

Un nuevo modelo: KNN

KNN es el abreviado de K nearest neighbors y es uno de los modelos más sencillos, que está basado en distancias. En los problemas de Regresión mide la distancia a los k vecinos más cercanos y su predicción se hace promediando sus valores.




```
from sklearn.neighbors import KneighborsRegressor
# n_neighbors es un hiperparámetro.
# es el valor por defecto
knn = KNeighborsRegressor(n_neighbors = 5)
knn.fit(X,y)
knn.score(X,y)
```

Aplicando a datos

Crear un modelo que prediga la edad de un pasajero utilizando Pclass, SibSp y Fare*1000.

> Nota: A modo de ejercicio aplicaremos el efecto de la Inflación (exagerada) dentro del Modelo.

Calcular métricas de Train y de Test.

Cómo mejorar el modelo?

Muchas veces mejorar el modelo es complejo con la data que se tiene, por lo tanto es necesario, transformar los datos para poder utilizar data de mejor manera.

Este proceso de transformación de datos se conoce como preprocesamiento y **NO HAY UNA FORMULA CORRECTA DE LLEVARLO A CABO**[1].

Para realizar estos procesos de manera ordenada `scikit-learn` posee los transformers.

```
from sklearn.submodule import Transformer
tr = Transformer()
tr.fit(X)
tr.transform(X)
```

[1]: Aunque hay formas que no son correctas de hacerlo para evitar **data leakage**

StandardScaler()

Corresponde a la aplicación del Z-score en las variables de entrenamiento (`features`).

Este proceso puede dividirse en dos:

- Centrar: Restar la media
- Escalar: Dividir por la desviación estándar

> NOTA: Ambas operaciones se aplican por defecto pero se pueden desactivar a través de parámetros.
> Ver docs

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X)
sc.transform(X)
sc.fit_transform(X)
```

> NOTA: El resultado de esta operación es un Numpy Array.

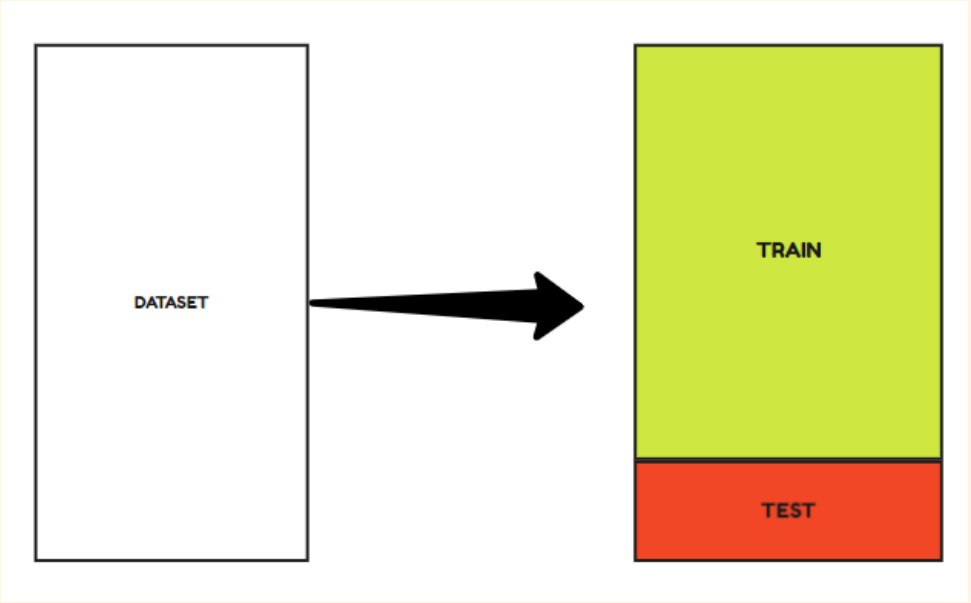
Escalar la data está recomendado cuando hay variables que contienen distintas escalas. En general modelos lineales como LR, KNN, SVM y NN se verán beneficiados.

Data Leakage

Data leakage se traduce como fuga de información y se refiere a una contaminación del proceso de modelamiento en la cual información de su Test set de alguna manera llega al proceso de entrenamiento.

Esto es perjudicial ya que el modelo se entera de cómo es el Test durante su entrenamiento, por lo que genera inestabilidad al momento de generalizar.

La manera más fácil de fugar información es a través del `StandardScaler()`.



Con Fuga

```
sc = StandardScaler()
X_sc = sc.fit_transform(X)

X_train,X_test,y_train,y_test = train_test_split(X_sc, y, test_size = 0.3, random_state = 123)


knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

evaluation(y_test, y_pred)
```

> RMSE: 12.848766382620424
> MAE: 3.2011839337500954
> R2: 0.1109983720096590 1

Sin Fuga

```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.3, random_state = 123)

sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)

knn = KNeighborsRegressor()
knn.fit(X_train_sc, y_train)
y_pred = knn.predict(X_test_sc)

evaluation(y_test, y_pred)
```

> RMSE: 12.834229356766496
> MAE: 3.201102442614478
> R2: 0.11300885745398226
>
> NOTA: **Nunca aplicar preprocesamiento antes del Data Split.**

Cómo agregar variables categóricas?

Las variables categóricas no son variables que los modelos procesan en primera instancia y normalmente deben pasar primero por un proceso de **encoding**.

`Scikit-Learn` posee algunas funciones de enconding que se pueden encontrar en `sklearn.preprocessing`, pero existe una librería alternativa llamada ~~`category-encoders`~~ feature-engine que será la que utilizaremos.

- La principal ventaja de category-encoders es que devuelve pandas DataFrames lo cuales son más sencillos de trabajar, además de proveer soporte para encodings más avanzados no disponibles en `scikit-learn`.
- Es posible escoger qué variables encodear además de lidiar automáticamente con variables no vistas durante el entrenamiento y missing values. Ver docs

```
from feature_engine.encoding import OrdinalEncoder, OneHotEncoder # Opción recomendada
from sklearn.preprocessing import OrdinalEncoding, OneHotEncoder # no tan recomendada
```

OrdinalEncoder()

Corresponde al encoder más sencillo en el cual cada categoría se le asigna un número entero.

```
from category_encoders import OrdinalEncoder
X = df.Embarked

ord_enc = OrdinalEncoder()
df_new = ord_enc.fit_transform(X)
```

	Embarked
0	S
1	C
2	S
3	S
4	S

	Embarked
0	1
1	2
2	1
3	1
4	1

- Sencilla
- No agranda de tamaño el df de salida.
- Se recomienda para variables ordinales.
- Los modelos de árboles aprovechan de mejor manera este encoding.

```
df_new.Embarked.value_counts()
```

```
1  644
2  168
3  77
4  2
```

Utilizando `.get_params()` es posible obtener la configuración utilizada.

OneHotEncoder()

Quizás corresponde al encoding más popular en el cual se transforma cada categoría en una columna binaria.

```
X = df.Embarked
# variables con nombres de categorías
ord_enc = OneHotEncoder(use_cat_names = True)
ord_enc.fit_transform(X)
```

	Embarked_S	Embarked_C	Embarked_Q	Embarked_nan
0	1	0	0	0
1	0	1	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0

- Computacionalmene más complejo que OrdinalEncoder.
- El df de salida es más grande que el de entrada.
- No se recomienda para variables con un gran número de categorías.
- En general entrega mejores resultados que el OrdinalEncoder.

Aplicando un Modelo

Generar un Modelo para predecir la Edad de los pasajeros del Titanic utilizando 'Pclass','SibSp','Fare' y 'Embarked'.

- Comparar el performance de KNN y de LinearRegression.
- Compare los resultados obtenidos utilizando OrdinalEncoder y OneHotEncoder.
- Mostrar resultados en train y test.

```
KNN + OneHot, Test:
RMSE: 12.716024352894255
MAE: 3.1656873129387146
R2: 0.12927223594120496
```

```
LR + OneHot, Test:
RMSE: 12.478825703086661
MAE: 3.103176867446452
R2: 0.16145354315592952
```

Simplificar Código: Pipelines

Cómo se puede observar a medida que se utilizan más operaciones de preprocesamiento el código comienza a ponerse más complejo y propenso a error. Es por eso que `scikit-learn` introduce el concepto de `Pipeline`.

El `Pipeline` permite el uso de varios estimators y transformers en orden, de tal manera de hacer el código más entendible.

```
from sklearn.pipeline import Pipeline
from feature_engine.imputation import CategoricalImputer

pipe = Pipeline(steps = [
    ('imp', CategoricalImputer(imputation_method = 'frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler()),
    ('knn', KNeighborsRegressor())
])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```