Pipelines
en Scikit-Learn

Alfonso Tobar Arancibia

Data Scientist

03-06-2021

github.com/datacubeR/clases-ml/Slides

Tratamiento de Nulos

Normalmente existen dos tipos de Nulos, los informativos y los MAR (missing at Random). Los informativos pueden ser útiles se pueden utilizar como una variable nueva. En cambio los valores MAR son por errores de proceso, no disponibilidad de información, etc.

La librería `feature-engine` ofrece una gran cantidad de alternativas para poder lidiar con imputaciones siguiendo la API de Scikit-Learn:

- MeanMedianImputer: Permite imputar mediante Mediana o Media.
- ArbitraryNumberImputer: Permite imputar de manera numérica cualquier valor arbitrario.
- DropMissingData: En el caso de no querer imputar es posible eliminar filas con valores nulos.
- AddMissingIndicator: Permite dejar constancia de que hay nulos. Esta estrategia puede ser particularmente útiles para algunos modelos.
- CategoricalImputer: Permite imputar una categoría mediante la Moda o con un indicador de *Missing*.

```
from feature_engine.imputation import MeanMedianImputer
from feature_engine.imputation import ArbitraryNumberImputer
from feature_engine.imputation import DropMissingData
from feature_engine.imputation import AddMissingIndicator
from feature_engine.imputation import CategoricalImputer
```

> Nota: Scikit-Learn Posee estrategias propias de Imputación que se pueden encontrar en `sklearn.impute`. Para este curso usaremos esta librería que ofrece mayor flexibilidad.

Imputación en Categorías

- CategoricalImputer es capaz de lidiar con variables nulos e categorías.
- Esta clase posee dos metodos de imputación:
  - 'frequent' permite imputar por el valor más frecuente, es decir, la moda.
  - 'missing' permite agregar una categoría extra que representa un valor perdido.

```
from feature_engine.imputation import CategoricalImputer
ci = CategoricalImputer(imputation_method = 'frequent')
ci.fit_transform(X_train)
```

- Incluye un parámetro llamado `fill_value` que corresponde al valor con el que se rellenará el indicador de missing.
- Finalmente el parámetro `variable`, permite escoger las variables a las cuales aplicar esta imputación. Por defecto se aplicará a todas las variables tipo Object.

Inputación Numérica

En el caso de querer imputar números hay bastante más opciones:

- Primero imputar valores arbitrarios. Esto es útil cuando se quiere imputar 0 por no contener información o para modelos de árboles valores extremos como -1, 999, etc.

```
an = ArbitraryNumberImputer(imputer_dict = {'Age' : 999})
an.fit_transform(X_train)
```

- `imputer_dict` corresponderá a un diccionario en el que se incluirá el nombre de la variable con el valor a imputar.

Otra estrategia bien común es imputar con Mediana o Media:

```
an = MeanMedianImputer(imputation_method = 'median')
an.fit_transform(X_train)
```

- `imputation_method` corresponderá al tipo de imputación a realizar que puede ser **'mean'** o **'median'**.

Ambos métodos también incluyen el parámetro `variable` para definir en qué variables se quiere realizar este procedimiento.

> Utilizar Media o Mediana permite mantener la distribución de los datos pero añadiendo una mayor concentración al centro de masa de la distribución.

Última Opción: Eliminar Datos

Hay veces que imputar datos no es una opción, la principal opción acá es cuando no vale la pena imputar. Algunos casos:

- Filas completas sin datos.
- Columnas Completas sin Datos

Cuando existe una variable que tiene una gran cantidad de nulos, es mejor eliminarla del modelo, ya que no aporta información relevante para las observaciones.

Cuando existan algunas filas con nulos, eliminarlo es una opción en el entrenamiento, pero no en el proceso de inferencia. Por lo tanto hay que ser cuidadoso acá:

```
dm = DropMissingData(missing_only = False)
dm.fit_transform(X_train).shape
```

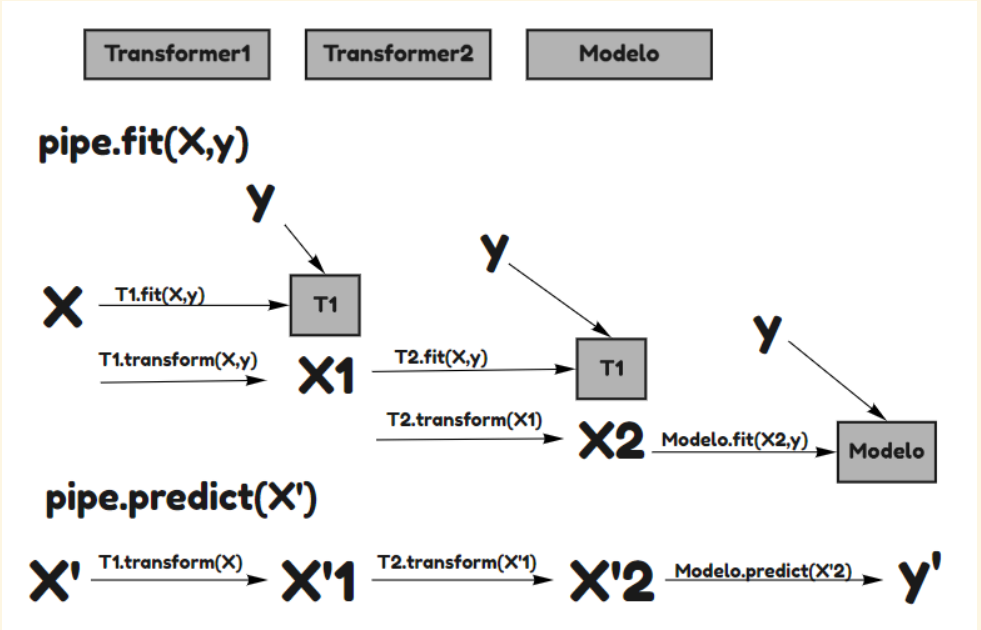
Pipelines

Trabajar con modelos de Machine Learning requiere de la combinación de muchos pasos: Preprocesamiento, Encoding, Estandarización, Selección de Variables, Reducción de Dimensionalidad, etc.

Esto puede complicar nuestro proceso de modelamiento, en especial en el proceso de Inferencia, generando que sea dificil de reproducir en un ambiente productivo.

Además es importante destacar que en caso de utilizar estrategias complejas de validación como lo es el K-Fold, hay una capa adicional de complejidad para evitar problemas de Data Leakage.

Para ello se introducen los Pipelines:



Pipelines

```
from sklearn.pipeline import Pipeline
pipe = Pipeline(steps = [
    ('imp_cat', CategoricalImputer(imputation_method = 'frequent')),
    ('n_imputer', MeanMedianImputer(imputation_method = 'median')),
    ('ohe', OneHotEncoder())
])
```

En este caso, se llevarán a cabo 3 procedimientos.

1. Imputación categórica, utilizando estrategia **'frequent'**,
2. Inputación numérica utilizando **'median'**.
3. OneHotEncoder para transformar variables categóricas en Dummies.

Los únicos requisitos que tiene un Pipeline:

- Se pueden utilizar cuantos steps se requieran,
- Pueden haber cuantos transformers se requieran,
- Puede haber como máximo un Estimator y este tiene que ir al final del Pipeline.

El Pipeline se comportará de acuerdo a su último step:

- Tendrá `fit()`, `transform()` o `fit_transform()` si finaliza en un Transform.
- Tendrá `fit()`, `predict()` y `score()` en el caso de terminar con un Estimator.

Combinando Pipelines con GridSearchCV

Es común utilizar Pipelines realizar estrategias de Cross Validation ya que utilizar CV complica el proceso de entrenamiento, en especial, si se quiere evitar el Data Leakage.

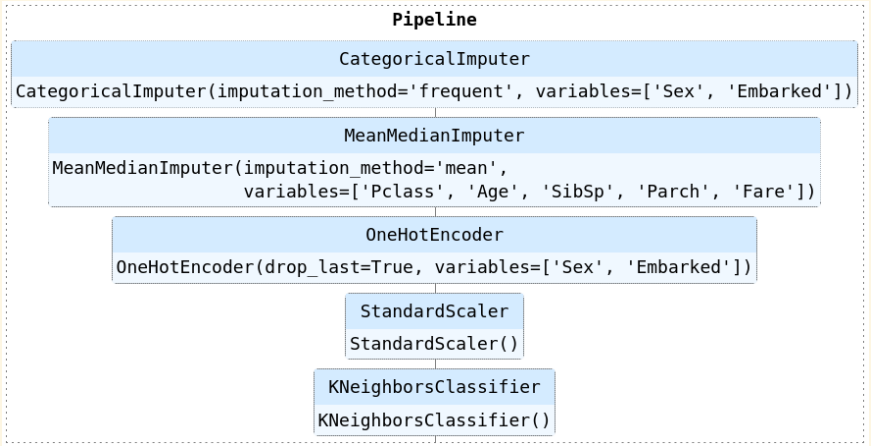
```
pipe_knn_cv = Pipeline(steps = [
    ('imp_cat', CategoricalImputer(imputation_method = 'frequent')),
    ('n_imputer', MeanMedianImputer(imputation_method = 'mean')),
    ('ohe', OneHotEncoder(drop_last = True)),
    ('sc', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

params = {'imp_cat__imputation_method': ['frequent','missing'],
          'n_imputer__imputation_method': ['mean','median'],
          'sc': [StandardScaler(), 'passthrough'],
          'knn__n_neighbors': np.arange(5,25,2)}

search = GridSearchCV(pipe_knn_cv, params, cv = 5, scoring = 'accuracy',
                      n_jobs = -1, verbose = 10, return_train_score = True)

search.fit(X_train, y_train)
```

Combinando Pipelines con GridSearchCV



GridSearchCV + Pipelines permite realizar búsquedas mucho más exhaustivas de la siguiente forma:

- Parámetros: Utilizando la convención `nombre_etapa__parametro`: Donde `nombre_etapa` corresponde al nombre entragado en el pipeline y `parametro` es el nombre del parámetro del Transformer/Estimator. Cabe destacar que se debe separar por dos underscores: __.
- Distintos Transformers/Estimators: En el caso de querer probar muchas combinaciones de Transformers/Estimators se pueden coloca en una lista aludiendo al nombre de la etapa.
- Saltar Pasos: Utilizando la palabra **'passthrough'** es posible desactivar alguno de los pasos del Pipeline.