ActividadSesion7

May 7, 2020

1 ACTIVIDAD SESIÓN 7

Crea una tubería (pipeline) que realice las siguientes tareas:

- Imputar valores perdidos
- Escalar los valores
- Transformar los valores categóricos en enteros
- Aplicar un algoritmo para entrenar y predecir el resultado.

Se utilizará el dataset del Titanic

1.1 Obtención de datos

Hemos usado los dos dataset que ya venían separados en train y test.

[1]:
```
## Data Preparation and Modeling
import pandas as pd
train = pd.read_csv('titanic_train.csv')
test = pd.read_csv('titanic_test.csv', names=train.columns.values)
train.tail(10)
```

[1]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
688	689	0	3	Fischer, Mr. Eberhard Thelander	male	18.0	0	0	350036	7.7958	NaN	S
689	690	1	1	Madill, Miss. Georgette Alexandra	female	15.0	0	1	24160	211.3375	B5	S
690	691	1	1	Dick, Mr. Albert Adrian	male	31.0	1	0	17474	57.0000	B20	S
691	692	1	3	Karun, Miss. Manca	female	4.0	0	1	349256	13.4167	NaN	C
692	693	1	3	Lam, Mr. Ali	male							
693	694	0	3	Saad, Mr. Khalil	male							
694	695	0	1	Weir, Col. John	male							
695	696	0	2	Chapman, Mr. Charles Henry	male							
696	697	0	3	Kelly, Mr. James	male							
697	698	1	3	Mullens, Miss. Katherine "Katie"	female							

```
692   NaN      0      0    1601   56.4958   NaN        S
693  25.0      0      0    2672    7.2250   NaN        C
694  60.0      0      0  113800   26.5500   NaN        S
695  52.0      0      0  248731   13.5000   NaN        S
696  44.0      0      0  363592    8.0500   NaN        S
697   NaN      0      0   35852    7.7333   NaN        Q
```

[2]:
```
test.head(10)
```

[2]:
```
   PassengerId  Survived  Pclass  \
0          699         0       1
1          700         0       3
2          701         1       1
3          702         1       1
4          703         0       3
5          704         0       3
6          705         0       3
7          706         0       2
8          707         1       2
9          708         1       1

                                                Name     Sex   Age  SibSp  \
0                            Thayer, Mr. John Borland    male  49.0      1
1              Humblen, Mr. Adolf Mathias Nicolai Olsen    male  42.0      0
2  Astor, Mrs. John Jacob (Madeleine Talmadge Force)  female  18.0      1
3                    Silverthorne, Mr. Spencer Victor    male  35.0      0
4                               Barbara, Miss. Saiide  female  18.0      0
5                               Gallagher, Mr. Martin    male  25.0      0
6                             Hansen, Mr. Henrik Juul    male  26.0      1
7     Morley, Mr. Henry Samuel ("Mr Henry Marshall")    male  39.0      0
8                       Kelly, Mrs. Florence "Fannie"  female  45.0      0
9                   Calderhead, Mr. Edward Pennington    male  42.0      0

   Parch    Ticket      Fare    Cabin Embarked
0      1     17421  110.8833      C68        C
1      0    348121    7.6500    F G63        S
2      0  PC 17757  227.5250  C62 C64        C
3      0  PC 17475   26.2875      E24        S
4      1      2691   14.4542      NaN        C
5      0     36864    7.7417      NaN        Q
6      0    350025    7.8542      NaN        S
7      0    250655   26.0000      NaN        S
8      0    223596   13.5000      NaN        S
9      0  PC 17476   26.2875      E24        S
```

Separamos los datos de train y test en características y etiquetas.

[1]:
```
## Separamos en conjunto de test y train
X_train = train.drop('Survived', axis=1)
y_train = train['Survived']

X_test = test.drop('Survived', axis=1)
y_test = test['Survived']
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-1-b45a0dcbd1c5> in <module>
      1 ## Separamos en conjunto de test y train
----> 2 X_train = train.drop('Survived', axis=1)
      3 y_train = train['Survived']
      4
      5 X_test = test.drop('Survived', axis=1)


NameError: name 'train' is not defined
```

1.2 PROCESAMIENTO DE DATOS

Empezamos con el procesamiento de los datos numéricos.

[4]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant')), ## IMPUTAMOS LOS VALORES␣
 ↪PERDIDOS CON CEROS (fill_value por
                                                      ## defecto es 0)
    ('scaler', StandardScaler())]) ## ESCALAMOS LOS VALORES
numerical_transformer
```

[4]:
```
Pipeline(memory=None,
         steps=[('imputer',
```

```
                 SimpleImputer(add_indicator=False, copy=True, fill_value=None,
                               missing_values=nan, strategy='constant',
                               verbose=0)),
                ('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True))],
         verbose=False)
```

Seguimos con el preprocesamiento de datos categóricos.

[5]:
```
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  ## IMPUTAMOS LOS␣
 ↪VALORES PERDIDOS CON EL VALOR
                                                           ## MAS FRECUENTE
    ('onehot', OneHotEncoder(handle_unknown='ignore')) ## TRANSFORMAMOS LOS␣
 ↪VALORES CATEGÓRICOS EN ENTEROS
])
categorical_transformer
```

[5]:
```
Pipeline(memory=None,
         steps=[('imputer',
                 SimpleImputer(add_indicator=False, copy=True, fill_value=None,
                               missing_values=nan, strategy='most_frequent',
                               verbose=0)),
                ('onehot',
                 OneHotEncoder(categories='auto', drop=None,
                               dtype=<class 'numpy.float64'>,
                               handle_unknown='ignore', sparse=True))],
         verbose=False)
```

[6]:
```
numeric_features = train.select_dtypes(include=['int64', 'float64']).
 ↪drop(['Survived'], axis=1).columns
numeric_features
```

[6]:
```
Index(['PassengerId', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'],
dtype='object')
```

[7]:
```
categorical_features = train.select_dtypes(include=['object']).columns
categorical_features
```

[7]:
```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

[3]:
```
# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
```

```
    ])

preprocessor
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-3-dacfc8035f03> in <module>
         1 # Bundle preprocessing for numerical and categorical data
   ----> 2 preprocessor = ColumnTransformer(
         3     transformers=[
         4         ('num', numerical_transformer, numeric_features),
         5         ('cat', categorical_transformer, categorical_features)


       NameError: name 'ColumnTransformer' is not defined
```

1.3 Aplicamos un modelo de clasificación

Definimos el modelo, entrenamos y obtenemos resultados

[9]:
```
# Define model
from sklearn.ensemble import RandomForestClassifier
rf = Pipeline(steps=[('preprocessor', preprocessor),
                     ('classifier', RandomForestClassifier())])
rf
```

[9]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('imputer',
SimpleImputer(add_indicator=False,
 copy=True,
 fill_value=None,
 missing_values=nan,
 strategy='constant',
 verbose=0)),
```

```
                                                                ('scaler',
StandardScaler(copy=True,
  with_me…
                RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                       class_weight=None, criterion='gini',
                                       max_depth=None, max_features='auto',
                                       max_leaf_nodes=None, max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1, min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=100, n_jobs=None,
                                       oob_score=False, random_state=None,
                                       verbose=0, warm_start=False))],
        verbose=False)
```

[10]:
```
# Preprocessing of training data, fit model
rf.fit(X_train, y_train)
```

[10]:
```
Pipeline(memory=None,
         steps=[('preprocessor',
                 ColumnTransformer(n_jobs=None, remainder='drop',
                                   sparse_threshold=0.3,
                                   transformer_weights=None,
                                   transformers=[('num',
                                                  Pipeline(memory=None,
                                                           steps=[('imputer',
SimpleImputer(add_indicator=False,
 copy=True,
 fill_value=None,
 missing_values=nan,
 strategy='constant',
 verbose=0)),
                                                                  ('scaler',
StandardScaler(copy=True,
  with_me…
                RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                       class_weight=None, criterion='gini',
                                       max_depth=None, max_features='auto',
                                       max_leaf_nodes=None, max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1, min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=100, n_jobs=None,
                                       oob_score=False, random_state=None,
                                       verbose=0, warm_start=False))],
```

```
            verbose=False)
```

[11]:
```
# Preprocessing of validation data, get predictions
preds = rf.predict(X_test)
preds
```

[11]:
```
array([1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0])
```

[12]:
```
print('MAE:', mean_absolute_error(y_test, preds))
```

```
MAE: 0.16580310880829016
```