day-29-titanic-using-pipeline

May 26, 2025

[ ]:
```
import numpy as np
import pandas as pd
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.tree import DecisionTreeClassifier
```

[ ]:
```
df = pd.read_csv('/content/Titanic-Dataset.csv')
```

[ ]:
```
df.head()
```

[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th…	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

1 Let's Plan

[ ]:
```
df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True)
```

[ ]:
```
# Step 1 -> train/test/split
X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['Survived']),
                                                 df['Survived'],
                                                 test_size=0.2,
                                                random_state=42)
```

[ ]:
```
X_train.head()
```

[ ]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
331	1	male	45.5	0	0	28.5000	S
733	2	male	23.0	0	0	13.0000	S
382	3	male	32.0	0	0	7.9250	S
704	3	male	26.0	1	0	7.8542	S
813	3	female	6.0	4	2	31.2750	S

[ ]:
```
y_train.sample(5)
```

[ ]:
```
176    0
246    0
328    1
138    0
828    1
Name: Survived, dtype: int64
```

[ ]:
```
# imputation transformer
trf1 = ColumnTransformer([
    ('impute_age',SimpleImputer(),[2]),
    ('impute_embarked',SimpleImputer(strategy='most_frequent'),[6])
],remainder='passthrough')
```

[ ]:
```
# one hot encoding
trf2 = ColumnTransformer([
    ␣
  ↪('ohe_sex_embarked',OneHotEncoder(sparse_output=False,handle_unknown='ignore'),[1,6])
],remainder='passthrough')
```

[ ]:
```
# Scaling
trf3 = ColumnTransformer([
    ('scale',MinMaxScaler(),slice(0,10))
])
```

[ ]:
```
# Feature selection
trf4 = SelectKBest(score_func=chi2,k=8)
```

[ ]:
```
# train the model
trf5 = DecisionTreeClassifier()
```

2 Create Pipeline

[ ]:
```
pipe = Pipeline([
    ('trf1',trf1),
    ('trf2',trf2),
    ('trf3',trf3),
    ('trf4',trf4),
    ('trf5',trf5)
])
```

3 Pipeline Vs make_pipeline

Pipeline requires naming of steps, make_pipeline does not.

(Same applies to ColumnTransformer vs make_column_transformer)

[ ]:
```
# Alternate Syntax
pipe = make_pipeline(trf1,trf2,trf3,trf4,trf5)
```

[ ]:
```
# train
pipe.fit(X_train,y_train)
```

[ ]:
```
Pipeline(steps=[('columntransformer-1',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('impute_age', SimpleImputer(),
                                                  [2]),
                                                 ('impute_embarked',
SimpleImputer(strategy='most_frequent'),
                                                  [6])])),
                ('columntransformer-2',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('ohe_sex_embarked',
OneHotEncoder(handle_unknown='ignore',
sparse_output=False),
                                                  [1, 6])])),
                ('columntransformer-3',
                 ColumnTransformer(transformers=[('scale', MinMaxScaler(),
                                                  slice(0, 10, None))])),
                ('selectkbest',
                 SelectKBest(k=8,
                             score_func=<function chi2 at 0x79c7544191c0>)),
                ('decisiontreeclassifier', DecisionTreeClassifier())])
```

4 Explore the Pipeline

[ ]:
```
# Code here
pipe.named_steps
```

[ ]:
```
{'columntransformer-1': ColumnTransformer(remainder='passthrough',
                   transformers=[('impute_age', SimpleImputer(), [2]),
                                 ('impute_embarked',
                                  SimpleImputer(strategy='most_frequent'),
                                  [6])]),
 'columntransformer-2': ColumnTransformer(remainder='passthrough',
                   transformers=[('ohe_sex_embarked',
                                  OneHotEncoder(handle_unknown='ignore',
                                                sparse_output=False),
                                  [1, 6])]),
 'columntransformer-3': ColumnTransformer(transformers=[('scale',
MinMaxScaler(), slice(0, 10, None))]),
 'selectkbest': SelectKBest(k=8, score_func=<function chi2 at 0x79c7544191c0>),
 'decisiontreeclassifier': DecisionTreeClassifier()}
```

[ ]:
```
# Display Pipeline

from sklearn import set_config
set_config(display='diagram')
```

[ ]:
```
# Predict
y_pred = pipe.predict(X_test)
```

[ ]:
```
y_pred
```

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

[ ]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

[ ]:
```
0.6256983240223464
```

5 Cross Validation using Pipeline

[ ]:
```
# cross validation using cross_val_score
from sklearn.model_selection import cross_val_score
cross_val_score(pipe, X_train, y_train, cv=5, scoring='accuracy').mean()
```

[ ]:
```
np.float64(0.6391214419383433)
```

6 GridSearch using Pipeline

[ ]:
```
# gridsearchcv
params = {
    'trf5__max_depth':[1,2,3,4,5,None]
}
```

[ ]:
```
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(pipe, params, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-24-da6873119a05> in <cell line: 0>()
      1 from sklearn.model_selection import GridSearchCV
      2 grid = GridSearchCV(pipe, params, cv=5, scoring='accuracy')
----> 3 grid.fit(X_train, y_train)

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪fit(self, X, y, **params)
   1022                 return results
   1023
-> 1024             self._run_search(evaluate_candidates)
   1025
   1026             # multimetric is determined here because in the case of a␣
 ↪callable

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1569     def _run_search(self, evaluate_candidates):
```

```
   1570         """Search all candidates in param_grid"""
-> 1571         evaluate_candidates(ParameterGrid(self.param_grid))
   1572
   1573

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪evaluate_candidates(candidate_params, cv, more_results)
    968                     )
    969
--> 970                 out = parallel(
    971                     delayed(_fit_and_score)(
    972                         clone(base_estimator),

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, iterable)
     75             for delayed_func, args, kwargs in iterable
     76         )
---> 77         return super().__call__(iterable_with_config)
     78
     79

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1916             output = self._get_sequential_output(iterable)
   1917             next(output)
-> 1918             return output if self.return_generator else list(output)
   1919
   1920         # Let's create an ID that uniquely identifies the current call.␣
 ↪If the

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in␣
 ↪_get_sequential_output(self, iterable)
   1845                 self.n_dispatched_batches += 1
   1846                 self.n_dispatched_tasks += 1
-> 1847                 res = func(*args, **kwargs)
   1848                 self.n_completed_tasks += 1
   1849                 self.print_progress()

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, *args, **kwargs)
    137             config = {}
    138         with config_context(**config):
--> 139             return self.function(*args, **kwargs)
    140
    141
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py␣
 ↪in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters,␣
 ↪fit_params, score_params, return_train_score, return_parameters,␣
 ↪return_n_test_samples, return_times, return_estimator, split_progress,␣
 ↪candidate_progress, error_score)
    852         # estimators in a pipeline.
    853         # ref: https://github.com/scikit-learn/scikit-learn/pull/26786
--> 854         estimator = estimator.set_params(**clone(parameters, safe=False ))
    855
    856     start_time = time.time()

/usr/local/lib/python3.11/dist-packages/sklearn/pipeline.py in set_params(self,␣
 ↪**kwargs)
    318             Pipeline class instance.
    319         """
--> 320         self._set_params("steps", **kwargs)
    321         return self
    322

/usr/local/lib/python3.11/dist-packages/sklearn/utils/metaestimators.py in␣
 ↪_set_params(self, attr, **params)
     67
     68         # 3. Step parameters and other initialisation arguments
---> 69         super().set_params(**params)
     70         return self
     71

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in set_params(self,␣
 ↪**params)
    281             if key not in valid_params:
    282                 local_valid_params = self._get_param_names()
--> 283                 raise ValueError(
    284                     f"Invalid parameter {key!r} for estimator {self}. "
    285                     f"Valid parameters are: {local_valid_params!r}."

ValueError: Invalid parameter 'trf5' for estimator␣
 ↪Pipeline(steps=[('columntransformer-1',
                   ColumnTransformer(remainder='passthrough',
                                     transformers=[('impute_age', SimpleImputer()
                                                    [2]),
                                                   ('impute_embarked',
                                                    ␣
 ↪SimpleImputer(strategy='most_frequent'),
                                                    [6])])),
                  ('columntransformer-2',
                   ColumnTransformer(remainder='passthrough',
                                     transformers=[('ohe_sex_embarked',
```

```
↪OneHotEncoder(handle_unknown='ignore',
                                                                 ␣
↪sparse_output=False),
                                                  [1, 6])])),
                ('columntransformer-3',
                 ColumnTransformer(transformers=[('scale', MinMaxScaler(),
                                                  slice(0, 10, None))])),
                ('selectkbest',
                 SelectKBest(k=8,
                             score_func=<function chi2 at 0x79c7544191c0>)),
                ('decisiontreeclassifier', DecisionTreeClassifier())]). Valid␣
↪parameters are: ['memory', 'steps', 'transform_input', 'verbose'].
```

[ ]:
```
grid.best_score_
```

[ ]:
```
grid.best_params_
```

7 Exporting the Pipeline

[ ]:
```
# export
import pickle
pickle.dump(pipe,open('pipe.pkl','wb'))
```

[ ]:
```

```