Exercise Set 5a

Hina Arora

(5a)
Pipelines with ColumnTransformers

https://scikit-learn.org/stable/modules/compose.html#pipeline

https://scikit-learn.org/stable/modules/compose.html#column-transformer

Pipelines

“Pipelines can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example feature selection, normalization and classification.

The Pipeline is built using a list of (key, value) pairs, where the key is a string containing the name you want to give this step and value is an estimator object.”

ColumnTransformers

“The ColumnTransformer helps performing different transformations for different columns of the data, within a Pipeline that is safe from data leakage and that can be parametrized. ColumnTransformer works on arrays, sparse matrices, and pandas DataFrames.

Warning: The compose.ColumnTransformer class is experimental and the API is subject to change.”

Copy

```
#Read kaggle train.csv data into a dataframe called df


df = pd.read_csv('data/kaggleTitanic/train.csv')
df.head()
```

Copy

```
#Extract a new column called “Deck” from “Cabin”.

def getDeck(cabin):
    if pd.notna(cabin):
        return cabin[0]
    else:
        return np.nan
df['Deck'] = df['Cabin'].apply(getDeck)
df.head()
```

Question

Set up the data:

- Create a dataframe called X which has all columns in df expect ‘Survived’
- Create a dataframe called y which has only the ‘Survived’ column in df

Answer

```
X = df.drop(['Survived'], axis=1)

y = df['Survived']
```

Question

Split X and y into test and train groups as follows:

- Test (Xtest, ytest) gets 20%
- Train (Xtrain, ytrain) gets 80%
- Use random_state=1

Answer

```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest =
        train_test_split(X, y, test_size=0.2, random_state=1)
```

Question

Set Xtrain, Xtest, ytrain, ytest to copies of these dataframes to deal with SetttingWithCopyWarning

Answer

```
Xtrain = Xtrain.copy()
Xtest = Xtest.copy()
ytrain = ytrain.copy()
ytest = ytest.copy()
```

Copy

```
# Set up preprocessing pipeline for numeric data
# - impute missing values with median

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

numeric_features = ['Age’]

numeric_transformer = Pipeline(steps=[
    ('si', SimpleImputer(missing_values=np.nan, strategy='median'))])
```

Copy

```
# Set up preprocessing pipeline for categorical data
# - impute missing values with constant 'X'
# - one-hot-encode imputed categorical values

from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

categorical_features = ['Pclass', 'Sex', 'Deck’]

categorical_transformer = Pipeline(steps=[
    ('si', SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='X')),
    ('ohe', OneHotEncoder(sparse=False, dtype=int, handle_unknown='ignore'))])
```

Copy

```
# Set up column transformer with preprocessing pipelines for
# numeric and categorical data
# - only keep imputed numeric and ohe catergorical features

from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)],
        remainder='drop') ### remainder='passthrough'
```

Copy

```
# Set up the preprocessing->model pipeline

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

clf = Pipeline(steps=[
        ('pp', preprocessor),
        (‘lr', LogisticRegression(solver='liblinear'))])
```

Copy

```
# fit using the combined preprocessing and model
# pipeline on train data
# - this will automatically run "fit" and "transform"
#   for all pre-processing steps, and "fit" for model
#   step on training data

clf.fit(Xtrain, ytrain)
```

Copy

```
# predict using the combined preprocessing and model
# pipeline on test data
# - this will automatically run "transform" for all
#    pre-processing steps, and "predict" for model step

ypred = clf.predict(Xtest)
```

Copy

```
# evaluate model on test data

from sklearn import metrics
print (metrics.accuracy_score(ytest, ypred))
print (metrics.confusion_matrix(ytest, ypred))
print (metrics.classification_report(ytest, ypred))
```