5.2 The Scikit-Learn API

May 9, 2019

1 5.2 The Scikit-Learn API

In the previous section, we implemented k-nearest neighbors from scratch. Now we will see how to implement it using Scikit-Learn, a Python library that makes it easy to train and use machine learning models. All models are trained and used in the exact same way:

1. Declare the model.
2. Fit the model to training data, consisting of both features X and labels y.
3. Use the model to predict the labels for new values of the features.

Let's take a look at how we would use this API to train a model on the Ames housing data set to predict the 2011 price of the Old Town house from the previous section. Scikit-Learn assumes that the data has already been completely converted to quantitative variables and that the variables have already been standardized (if desired). The code below reads in the data and does the necessary preprocessing.

(All of this code is copied from the previous section. Read the code, and if you are not sure what a particular line does, refer back to the previous section.)

```
In [1]: %matplotlib inline
        import numpy as np
        import pandas as pd
        pd.options.display.max_rows = 5

        housing = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/maste
                              sep="\t")

        housing["Date Sold"] = housing["Yr Sold"] + housing["Mo Sold"] / 12
        features = ["Lot Area", "Gr Liv Area",
                    "Full Bath", "Half Bath",
                    "Bedroom AbvGr",
                    "Year Built", "Date Sold",
                    "Neighborhood"]
        X_train = pd.get_dummies(housing[features])
        y_train = housing["SalePrice"]

        x_new = pd.Series(index=X_train.columns)
        x_new["Lot Area"] = 9000
        x_new["Gr Liv Area"] = 1400
```

```
x_new["Full Bath"] = 2
x_new["Half Bath"] = 1
x_new["Bedroom AbvGr"] = 3
x_new["Year Built"] = 1980
x_new["Date Sold"] = 2011
x_new["Neighborhood_OldTown"] = 1
x_new.fillna(0, inplace=True)

X_train_mean = X_train.mean()
X_train_std = X_train.std()
X_train_sc = (X_train - X_train_mean) / X_train_std
x_new_sc = (x_new - X_train_mean) / X_train_std

X_train_sc
```

Out[1]:

	Lot Area	Gr Liv Area	Full Bath	Half Bath	Bedroom AbvGr	Year Built	\
0	2.743912	0.309212	-1.024618	-0.755074	0.176064	-0.375473	
1	0.187065	-1.194223	-1.024618	-0.755074	-1.032058	-0.342410	
...	...	...	...	...	...	...	
2928	-0.017503	-0.218968	-1.024618	-0.755074	-1.032058	0.087408	
2929	-0.066107	0.989715	0.783894	1.234464	0.176064	0.715604	

	Date Sold	Neighborhood_Blmngtn	Neighborhood_Blueste	\
0	1.620757	-0.09821	-0.058511	
1	1.684822	-0.09821	-0.058511	
...	...	...	...	
2928	-1.518428	-0.09821	-0.058511	
2929	-1.069973	-0.09821	-0.058511	

	Neighborhood_BrDale	...	Neighborhood_NoRidge	\
0	-0.101692	...	-0.157561	
1	-0.101692	...	-0.157561	
...	...	...	...	
2928	-0.101692	...	-0.157561	
2929	-0.101692	...	-0.157561	

	Neighborhood_NridgHt	Neighborhood_OldTown	Neighborhood_SWISU	\
0	-0.245025	-0.297967	-0.129033	
1	-0.245025	-0.297967	-0.129033	
...	...	...	...	
2928	-0.245025	-0.297967	-0.129033	
2929	-0.245025	-0.297967	-0.129033	

	Neighborhood_Sawyer	Neighborhood_SawyerW	Neighborhood_Somerst	\
0	-0.233061	-0.211064	-0.257308	
1	-0.233061	-0.211064	-0.257308	
...	...	...	...	
2928	-0.233061	-0.211064	-0.257308	

```
2929            -0.233061             -0.211064             -0.257308

      Neighborhood_StoneBr  Neighborhood_Timber  Neighborhood_Veenker
0                -0.133073            -0.158694             -0.090862
1                -0.133073            -0.158694             -0.090862
...                    ...                  ...                   ...
2928             -0.133073            -0.158694             -0.090862
2929             -0.133073            -0.158694             -0.090862

[2930 rows x 35 columns]
```

`X_train_sc` is a matrix of all numbers, which is the form that Scikit-Learn expects. Now let's see how to use Scikit-Learn to fit a k-nearest neighbors model to this data.

```
In [2]: from sklearn.neighbors import KNeighborsRegressor

        # Step 1: Declare the model.
        model = KNeighborsRegressor(n_neighbors=30)

        # Step 2: Fit the model to training data.
        model.fit(X_train_sc, y_train)

        # Step 3: Use the model to predict for new observations.
        # Scikit-Learn expects 2-dimensional arrays, so we need to
        # turn the Series into a DataFrame with 1 row.
        X_new_sc = x_new_sc.to_frame().T
        model.predict(X_new_sc)

Out[2]: array([ 132343.33333333])
```

This is the exact same prediction that we got by implementing k-nearest neighbors manually.

In the case of training a machine learning model to predict for a single observation, Scikit-Learn may seem like overkill. In fact, the above Scikit-Learn code was 5 lines, whereas our implementation of k-nearest neighbors in the previous section was only 4 lines. However, learning Scikit-Learn will pay off as the problems become more complex.

1.1 Preprocessing in Scikit-Learn

We constructed `X_train_sc` and `x_new_sc` above using just basic `pandas` operations. But it is also possible to have Scikit-Learn do this preprocessing for us. The preprocessing objects in Scikit-Learn all follow the same basic pattern:

1. First, the preprocessing object has to be "fit" to a data set.
2. The `.transform()` method actually processes the data. This method can be called repeatedly on multiple data sets and is guaranteed to process each data set in exactly the same way.

It might not be obvious why it is necessary to first "fit" the preprocessing object to a data set before using it to process data. Hopefully, the following examples will make this clear.

1.1.1 Example 1: Dummy Encoding

Instead of using `pd.get_dummies()`, we can do dummy encoding in Scikit-Learn using the `DictVectorizer` tool. There is one catch: `DictVectorizer` expects the data as a list of dictionaries, not as a `DataFrame`. But each row of a `DataFrame` can be represented as a dictionary, where the keys are the column names and the values are the data. `Pandas` provides a convenience function, `.to_dict()`, that converts a `DataFrame` into a list of dictionaries.

```
In [3]: X_train_dict = housing[features].to_dict(orient="records")
        X_train_dict[:2]
```

```
Out[3]: [{'Lot Area': 31770,
          'Gr Liv Area': 1656,
          'Full Bath': 1,
          'Half Bath': 0,
          'Bedroom AbvGr': 3,
          'Year Built': 1960,
          'Date Sold': 2010.4166666666667,
          'Neighborhood': 'NAmes'},
         {'Lot Area': 11622,
          'Gr Liv Area': 896,
          'Full Bath': 1,
          'Half Bath': 0,
          'Bedroom AbvGr': 2,
          'Year Built': 1961,
          'Date Sold': 2010.5,
          'Neighborhood': 'NAmes'}]
```

Now we pass this list to `DictVectorizer`, which will expand each categorical variable (e.g., "Neighborhood") into dummy variables. When the vectorizer is fit to the training data, it will learn all of the possible categories for each categorical variable so that when `.transform()` is called on different data sets, the same dummy variables will be returned (and in the same order). This is important for us because we need to apply the encoding to two data sets, the training data and the new observation, and we want to be sure that the same dummy variables appear in both.

```
In [4]: from sklearn.feature_extraction import DictVectorizer

        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)

        X_train = vec.transform(X_train_dict)
        x_new_dict = {
            "Lot Area": 9000,
            "Gr Liv Area": 1400,
            "Full Bath": 2,
            "Half Bath": 1,
            "Bedroom AbvGr": 3,
            "Year Built": 1980,
            "Date Sold": 2011,
```

```
    "Neighborhood": "OldTown"
}
X_new = vec.transform([x_new_dict])

X_train
```

```
Out[4]: array([[  3.00000000e+00,   2.01041667e+03,   1.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.96000000e+03],
               [  2.00000000e+00,   2.01050000e+03,   1.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.96100000e+03],
               [  3.00000000e+00,   2.01050000e+03,   1.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.95800000e+03],
               ...,
               [  3.00000000e+00,   2.00658333e+03,   1.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.99200000e+03],
               [  2.00000000e+00,   2.00633333e+03,   1.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.97400000e+03],
               [  3.00000000e+00,   2.00691667e+03,   2.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   1.99300000e+03]])
```

1.1.2 Example 2: Scaling

We can also use Scikit-Learn to scale our data. The `StandardScaler` function standardizes data, but there are other functions, such as `Normalizer` and `MinMaxScaler`, that normalize and apply min-max scaling to the data, respectively.

In the previous section, we standardized both the training data and the new observation with respect to the *training data*. To specify that the standardization should be with respect to the training data, we fit the scaler to the training data. Then, we use the scaler to transform both the training data and the new observation.

```
In [5]: from sklearn.preprocessing import StandardScaler

        scaler = StandardScaler()
        scaler.fit(X_train)

        X_train_sc = scaler.transform(X_train)
        X_new_sc = scaler.transform(X_new)

        X_train_sc
```

```
Out[5]: array([[ 0.17609421,  1.62103356, -1.02479289, ..., -0.15872127,
                 -0.0908778 , -0.37553701],
               [-1.03223376,  1.68510949, -1.02479289, ..., -0.15872127,
                 -0.0908778 , -0.34246845],
               [ 0.17609421,  1.68510949, -1.02479289, ..., -0.15872127,
                 -0.0908778 , -0.44167415],
               ...,
               [ 0.17609421, -1.32645923, -1.02479289, ..., -0.15872127,
                 -0.0908778 ,  0.68265709],
```

```
       [-1.03223376, -1.51868702, -1.02479289, ..., -0.15872127,
        -0.0908778 ,  0.0874229 ],
       [ 0.17609421, -1.07015551,  0.7840283 , ..., -0.15872127,
        -0.0908778 ,  0.71572565]])
```

1.2 Putting It All Together

The following example shows a complete pipeline: from reading in the raw data and processing it, to fitting a machine learning model and using it for prediction.

```
In [6]: # Read in the data.
        housing = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/master
                              sep="\t")

        # Define the features.
        housing["Date Sold"] = housing["Yr Sold"] + housing["Mo Sold"] / 12
        features = ["Lot Area", "Gr Liv Area",
                    "Full Bath", "Half Bath",
                    "Bedroom AbvGr",
                    "Year Built", "Date Sold",
                    "Neighborhood"]

        # Define the training data.
        # Represent the features as a list of dicts.
        X_train_dict = housing[features].to_dict(orient="records")
        X_new_dict = [{
            "Lot Area": 9000,
            "Gr Liv Area": 1400,
            "Full Bath": 2,
            "Half Bath": 1,
            "Bedroom AbvGr": 3,
            "Year Built": 1980,
            "Date Sold": 2011,
            "Neighborhood": "OldTown"
        }]
        y_train = housing["SalePrice"]

        # Dummy encoding
        vec = DictVectorizer(sparse=False)
        vec.fit(X_train_dict)
        X_train = vec.transform(X_train_dict)
        X_new = vec.transform(X_new_dict)

        # Standardization
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_train_sc = scaler.transform(X_train)
        X_new_sc = scaler.transform(X_new)
```

```
        # K-Nearest Neighbors Model
        model = KNeighborsRegressor(n_neighbors=30)
        model.fit(X_train_sc, y_train)
        model.predict(X_new_sc)
```

```
Out[6]: array([ 132343.33333333])
```

2 Exercises

Exercise 1. (This exercise is identical to Exercise 2 from the previous section, except it asks you to use Scikit-Learn.) You would like to predict how much a male diner will tip on a bill of \$40.00 on a Sunday. Use Scikit-Learn to build a *k*-nearest neighbors model to answer this question, using the Tips dataset (`https://raw.githubusercontent.com/dlsun/data-science-book/master/data/tips.csv`) as your training data.

```
In [10]: tips = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/master/d

         tips_features = ["total_bill", "sex", "day"]

         tips_X_train_dict = tips[tips_features].to_dict(orient="records")
         tips_X_new_dict = [{
             "total_bill":40.00,
             "sex":"Male",
             "day":"Sun"
         }]

         tips_y_train = tips["tip"]

         tips_vec = DictVectorizer(sparse=False)
         tips_vec.fit(tips_X_train_dict)
         tips_X_train = tips_vec.transform(tips_X_train_dict)
         tips_X_new = tips_vec.transform(tips_X_new_dict)

         tips_scaler = StandardScaler()
         tips_scaler.fit(tips_X_train)
         tips_X_train_sc = tips_scaler.transform(tips_X_train)
         tips_X_new_sc = tips_scaler.transform(tips_X_new)

         tips_model = KNeighborsRegressor(n_neighbors=30)
         tips_model.fit(tips_X_train_sc, tips_y_train)
         tips_model.predict(tips_X_new_sc)
```

```
Out[10]: array([ 3.902])
```

```
In [21]: tips_X_train
```

```
Out[21]: array([[  0.  ,   0.  ,   1.  , ...,   1.  ,   0.  ,  16.99],
                [  0.  ,   0.  ,   1.  , ...,   0.  ,   1.  ,  10.34],
                [  0.  ,   0.  ,   1.  , ...,   0.  ,   1.  ,  21.01],
                ...,
                [  0.  ,   1.  ,   0.  , ...,   0.  ,   1.  ,  22.67],
                [  0.  ,   1.  ,   0.  , ...,   0.  ,   1.  ,  17.82],
                [  0.  ,   0.  ,   0.  , ...,   1.  ,   0.  ,  18.78]])
```