Data Science

Intro to scikit-learn

CS202 11.09.22

Scikit-Learn

Download and Import

Just as we did for all other libraries we need to download scikit-learn.

Use the following command to download:

```
pip install –U scikit–learn
```

Importing will be a little different than other libraries because scikit-learn contains many machine learning models. For linear regression you will import via the following:

```
from sklearn.linear_model import LinearRegression
```

Scikit-Learn

Dependent and Independent Variables

Given data, you have independent variables and a dependent variable. The dependent is the variable which you want to make predictions on. The independent variables are the variables you use to make your prediction.

Hence given a data frame you will want to separate your dependent and independent variables. In our coding demo we will see the following for USA housing data:

```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
                'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price']
```

Let’s decipher what we’re doing:

```
X = USAhousing[['Avg. Area Income', 'Avg. Area
House Age', 'Avg. Area Number of Rooms',
                'Avg. Area Number of Bedrooms',
'Area Population']]


y = USAhousing[‘Price’]
```

We have a data frame called USAhousing. We declare a variable X to be the columns of the data frame which we will use to predict our y variable which is the ‘price’ column of our data.

Scikit-Learn

Training vs. Testing

For a machine learning model we always want to split between training and testing data.

Our training data is the portion of the data which we use to train our machine learning model. In our case it will be the portion of the data which we use to get the best fitted line.

Our testing data is the portion of the data which we use to see how well our best fitted line predicts the data.

We usually want an 80-20 split. We want more data for which we use to train our model and less for which we use to test.

Scikit-Learn

Training vs. Testing

For example: for simplistic reasoning suppose in our USAhousing had a total of 100 houses with all the features and price. 80 of those houses we will choose at random to feed into our machine so our machine can create the best fitted line.

For the remaining 20 houses we will only use the independent variables, i.e. the features of the house and feed it into our machine to see what our machine predicts as prices for those 20 houses. We compare our predicted prices with the true prices of those 20 houses.

Scikit-Learn

Training vs. Testing

How do we do this with sklearn? We need to import a few things:

We begin by importing train test split:

```
from sklearn.model_selection import train_test_split
```

Then we split the data into a training set and a testing set:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=101)
```

X_train, X_test, y_train, y_test are what we are calling the data, i.e. they are just variable names

Lets this decipher this a little further:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=101)
```

X_train, X_test, y_train, y_test are what we are calling the data, i.e. they are just variable names.

X_train will contain the portion of the data which is meant for training. It will also only contain the columns of the data frame which we use as independent variables.

X_test will contain the portion of the data which is meant for training and it also only contains the columns of the data frames which we use for independent variables

y_train will contain the portion of the data which is meant for training which only contains one column of that data that is the dependent variable (namely, “price” in our example)

y_test will contain the portion of the data which is meant for testing which only contains the dependent variable.

Train test split contains the following parameters:

```
train_test_split(*arrays, test_size=None, train_size=None, random_state
=None, shuffle=True, stratify=None)
```

- ***arrays**: sequence of indexables with same length / shape[0].

  Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

- **test_size**: float or int, default=None.

  If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If train_size is also None, it will be set to 0.25.

Train test split contains the following parameters:

```
train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```

- **random_state**: int, RandomState instance or None, default=None

  Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls.

- **shuffle**: bool, default=True

  Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

- **stratify**: array-like, default=None

  If not None, data is split in a stratified fashion, using this as the class labels.

This means in our code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.4, random_state=101)
```

Since we set test_size = 0.4, our test size is going to be 40% of the entire data i.e. we did a 60-40 split. You can play around with splitting once you start your own model. However, most standard would be to do a 80-20 split so you would choose your test_size = 0.2.

We also write random_state = 101. This is very much like random seed.

This is not completely necessary. But if you want to reproduce the same result it can be useful to set a number and it does not have to 101.

Scikit-Learn

Linear Regression

Now we want to build our linear regression:

We begin the import:

```
from sklearn.linear_model import LinearRegression
```

Give our linear regression model a name:

```
lm = LinearRegression()
```

Start fitting the model to our training data:

```
lm.fit(X_train,y_train)
```

And we’re done! It’s as easy as that!

Scikit-Learn

Linear regression

Remember that our best fitted line is of the following form:

$$y = m_1x_1 + m_2x_2 + \ldots + m_nx_n + b$$

Where n is the number of independent variables and each x_i is an independent variable for i =1,..n. Then each m_i is the coefficient of x_i for i = 1,…,n and b is the y-intercept.

We can print each of these using scikit-learn

Scikit-Learn

Linear regression

Remember that we called our linear regression model “`lm`”.

We can ask scikit-learn to show us our y-intercept:

```
print(lm.intercept_)
```

Then we can ask scikit-learn to show us our coefficients nicely in the form of a data frame:

```
pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
```

The code:

```
pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
```

Will produce the following data frame:

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Scikit Learn

Predictions

To see our predictions on our testing set we use the following:

```
lm.predict(X_test)
```

It is usually a good idea to put this into a variable so that we can later test our predictions with the actual y_test, i.e. the true value of the dependent variable.

Scikit-Learn

Scoring

We haven’t quite discussed how we can measure how well our machine learning model’s performance is.

We will go into more depth on:

MAE is the easiest to understand, because it's the average error.

MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

Next time, as this will involve some mathematical concepts.