

IIT Madras

ONLINE DEGREE

Machine Learning Practice
Indian Institute of Technology, Madras
Polynomial Regression

(Refer Slide Time: 0:12)



Polynomial regression



Namaste, welcome to the next video of Machine Learning Practice course, in this video we will study how to implement polynomial regression with sklearn.

(Refer Slide Time: 0:25)



How is **polynomial regression** model
trained?

Step 1: Apply **polynomial transformation** on the feature matrix.

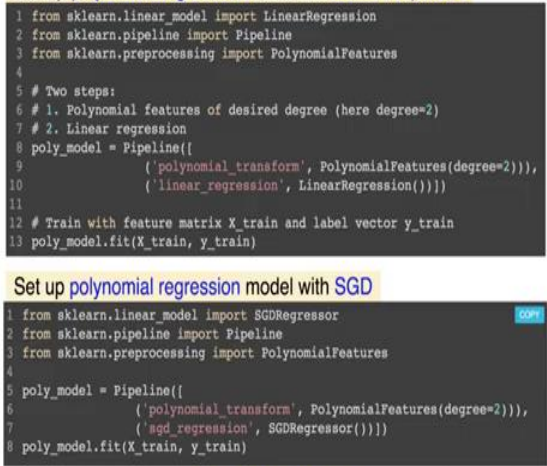
Step 2: Learn **linear regression model** (via **normal equation** or **SGD**) on the **transformed feature matrix**.

Implementation tips: Make use of pipeline construct for polynomial transformation followed by linear regression estimator.

So, let us see how polynomial regression model is trained in sklearn. Polynomial regression is a two-step process where in the first step we apply polynomial transformation on the feature matrix and in the second step we learn linear regression model on the transform feature matrix. The linear regression model can be learned by a normal equation or stochastic gradient descent procedure. Since this is a process where there are, there is a preprocessing

step involved along with estimator we make use of pipeline construct for polynomial transformation followed by linear regression estimator.

(Refer Slide Time: 1:04)



The image displays two code snippets side-by-side, comparing the setup for polynomial regression models. The top snippet, titled 'Set up polynomial regression model with normal equation', shows the import of LinearRegression and the creation of a Pipeline with PolynomialFeatures (degree=2) and LinearRegression. The bottom snippet, titled 'Set up polynomial regression model with SGD', shows the import of SGDRegressor and the creation of a Pipeline with PolynomialFeatures (degree=2) and SGDRegressor. The only difference between the two is the estimator used in the pipeline. A note at the bottom states: 'Notice that there is a single line code change in two code snippets.'

```
Set up polynomial regression model with normal equation
1 from sklearn.linear_model import LinearRegression
2 from sklearn.pipeline import Pipeline
3 from sklearn.preprocessing import PolynomialFeatures
4
5 # Two steps:
6 # 1. Polynomial features of desired degree (here degree=2)
7 # 2. Linear regression
8 poly_model = Pipeline([
9     ('polynomial_transform', PolynomialFeatures(degree=2)),
10    ('linear_regression', LinearRegression())
11])
12 # Train with feature matrix X_train and label vector y_train
13 poly_model.fit(X_train, y_train)

Set up polynomial regression model with SGD
1 from sklearn.linear_model import SGDRegressor
2 from sklearn.pipeline import Pipeline
3 from sklearn.preprocessing import PolynomialFeatures
4
5 poly_model = Pipeline([
6     ('polynomial_transform', PolynomialFeatures(degree=2)),
7     ('sgd_regression', SGDRegressor())
8])
9 poly_model.fit(X_train, y_train)

Notice that there is a single line code change in two code snippets.
```

Let us look at how to implement polynomial regression model with normal equation. So, here we have used a pipeline construct, so we have instantiated an object of a pipeline class with two steps. The first step we have polynomial transformation and the second step we have linear regression estimator.

So, the first step is set by giving a name to the step followed by instantiating the object of polynomial features. Here we have set the degree to be 2, so we will get polynomial features of degree 2 from this particular step. Then we instantiate an object of linear regression estimator. So, this transformed feature set is fed into this estimator for training as well as for evaluation.

We can train this particular model by calling the fit function on the pipeline object and we supply the feature matrix and the label vector as arguments. Here if you are solving a multi output regression problem the y_train will actually be a label matrix. We can also use stochastic gradient descent for solving polynomial regression.

You can notice that there is there is a single line code change in the pipeline setup so we have instead of linear regression we have SGD regressor where we instantiate object of a SGD regressor estimator. There is another code change which is in the import section where we import SGD regressor instead of linear regression in this particular code snippet. So, this is how you can set up polynomial regression model with normal equation and stochastic gradient descent regressor.

(Refer Slide Time: 3:05)

How to use **only** interaction features for polynomial regression?



```
from sklearn.preprocessing import PolynomialFeatures  
poly_transform = PolynomialFeatures(degree=2, interaction_only=True)
```

$[x_1, x_2]$ is transformed to $[1, x_1, x_2, x_1x_2]$.

Note that $[x_1^2, x_2^2]$ are excluded.



So, the polynomial feature transformation will get us polynomial features of all degrees, sometimes we want to exclude higher degree polynomial features corresponding to the individual features and we want only the interaction features. So, the polynomial features class has a parameter `interaction_only` in its constructor which is by default set to false.

If we want any interaction features we should set this parameter to true. So, for an example if we have two features x_1 and x_2 by setting interaction only to true we will get a transformation that contains features 1, x_1 , x_2 , x_1x_2 . So, note that x_1^2 and x_2^2 are excluded because we set interaction only to true.

So, this is how you can perform polynomial regression in sklearn. Now, the question is how do we really set this degree, what is the appropriate value for the degree? And as we discussed in machine learning techniques course this can be found with hyper parameter tuning and that is the topic of our next video.