Regression: Hands-on Session

Ditty Mathew

Machine Learning Camp

22^{nd} June 2018

NumPy

- NumPy stands for Numerical Python
- To import numpy in python
    - import numpy

NumPy

array: N-dimensional array; collection of items of same type

NumPy

array: N-dimensional array; collection of items of same type

```
import numpy as np
```

NumPy

array: N-dimensional array; collection of items of same type

```
import numpy as np
a= np.array([1,2,3])
```

NumPy

array: N-dimensional array; collection of items of same type

```
import numpy as np
a= np.array([1,2,3])
print a
```

NumPy

array: N-dimensional array; collection of items of same type

```
import numpy as np
a= np.array([1,2,3])
print a
print a.shape
```

array: N-dimensional array; collection of items of same type

```
import numpy as np
a= np.array([1,2,3])
print a
print a.shape
```

- array of more than one dimensions

```
a = np.array([[1, 2], [3, 4]])
print a
```

NumPy

NumPy also provides a reshape function to resize an array.

```
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

NumPy

To append a column to a numpy array

```
a = np.array([[1,2,3],[4,5,6]])
np.column_stack((a,[7,8]))
```

NumPy

To return a new array of specified size, filled with zeros

np.zeros((5,2))

To return a new array of specified size, filled with ones

np.ones((5,2))

NumPy

a = np.array([[1,2,3],[4,5,6]])

To fetch value of i^{th} row and j^{th} colummn

a[i,j]

To fetch all values in j^{th} column

- a[:,j]

NumPy

a = np.array([[1,2,3],[4,5,6]])

To fetch value of i^{th} row and j^{th} colummn

a[i,j]

To fetch all values in j^{th} column

- a[:,j]

To fetch all values in i^{th} row

- a[i,:]

NumPy

To multiply two matrices

```
x=np.array([[1,2],[3,4]])
y=np.array([[1,2,3],[3,4,5]])
np.dot(x,y)
```

NumPy

To multiply two matrices

```
x=np.array([[1,2],[3,4]])
y=np.array([[1,2,3],[3,4,5]])
np.dot(x,y)
```

To find a transpose of a matrix

```
y=np.array([[1,2,3],[3,4,5]])
y.transpose()
```

NumPy

To multiply two matrices

```
x=np.array([[1,2],[3,4]])
y=np.array([[1,2,3],[3,4,5]])
np.dot(x,y)
```

To find a transpose of a matrix

```
y=np.array([[1,2,3],[3,4,5]])
y.transpose()
```

To find inverse of a matrix

```
y=np.array([[1,2,3],[3,4,5]])
np.linalg.inv(y)
```

NumPy

- To load data from file

  data =np.loadtxt(open("data.csv","rb"), delimiter = ',')

Linear Regression Model

- Import : from sklearn import linear_model

Linear Regression Model

- Import : from sklearn import linear_model
- Select Model

```
regr = linear_model.LinearRegression(fit_intercept=True)
```

- Import : from sklearn import linear_model
- Select Model

    regr = linear_model.LinearRegression(fit_intercept=True)
- Train the model using training data

    regr.fit(X_train, y_train)
- Make predictions using test data

    y_pred = regr.predict(X_test)

Linear Regression Model

- To retrieve coefficients: $\theta_1, t\theta_2, \ldots$

  regr.coef_

- To retrieve coefficient θ_0

  regr.intercept_

Assignment 1: Single Variable Linear Regression

Load file “data_train_sv.csv” to train_data

Load file “data_test_sv.csv” to train_data

- Train linear regression model using train_data and predict the target values of test_data
- Compute Mean squared error
- Plot the model

Linear Regression : Evaluation Metrics

- Import from sklearn.metrics import mean_squared_error

Linear Regression : Evaluation Metrics

- Import from sklearn.metrics import mean_squared_error
- mean_squared_error(y_test, y_pred)

Training the model

regr.fit(X_train, y_train)

$\theta = (X\_train^T * X\_train)^{-1} * X\_train^T * y\_train$

Training the model

regr.fit(X_train, y_train)

$\theta = (X\_train^T * X\_train)^{-1} * X\_train^T * y\_train$

Prediction of target value of test data

y_pred = regr.predict(X_test)

$y\_pred = X\_test\theta$

Matplotlib

- import matplotlib.pyplot as plt

  plt.scatter(X_test, y_test, color='black')

Matplotlib

- import matplotlib.pyplot as plt

  plt.scatter(X_test, y_test, color='black')

  plt.plot(X_test, y_pred, color='blue', linewidth=3)

Matplotlib

- import matplotlib.pyplot as plt

  plt.scatter(X_test, y_test, color='black')
  plt.plot(X_test, y_pred, color='blue', linewidth=3)
  plt.show()

Assignment 1: Multiple Variable Linear Regression

Load file "housing_data.csv" to data

Split 80% of data to training data and 20% of data to test data

Normalize features (Feature scaling)

- Train linear regression model using train_data and predict the target values of test_data
- Compute Mean squared error

Training and Testing Data Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Let the underlying function be

$$y\_actual = cos^2 2\pi X$$

Plot the function $y_a ctual$

```
X = np.linspace(0, 0.5, 100)
y_actual = np.cos(2 * np.pi * X)**2
```

Let the underlying function be

$$y\_actual = cos^2 2\pi X$$

Plot the function $y_a ctual$

```
X = np.linspace(0, 0.5, 100)
y_actual = np.cos(2 * np.pi * X)**2
```

Generate the dataset by adding noise to the underlying function

Let the underlying function be

$$y\_actual = cos^2 2\pi X$$

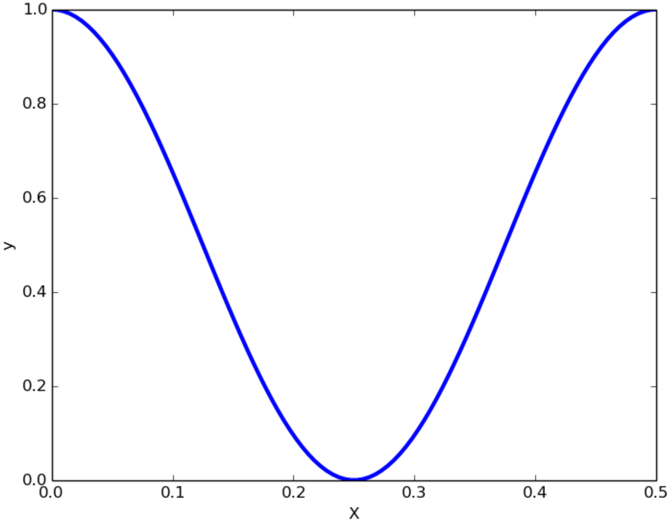
Plot the function y_actual

```
X = np.linspace(0, 0.5, 100)
y_actual = np.cos(2 * np.pi * X)**2
```

Generate the dataset by adding noise to the underlying function

```
noise = np.random.normal(0, 0.1, 100)
y= y_actual+noise
```

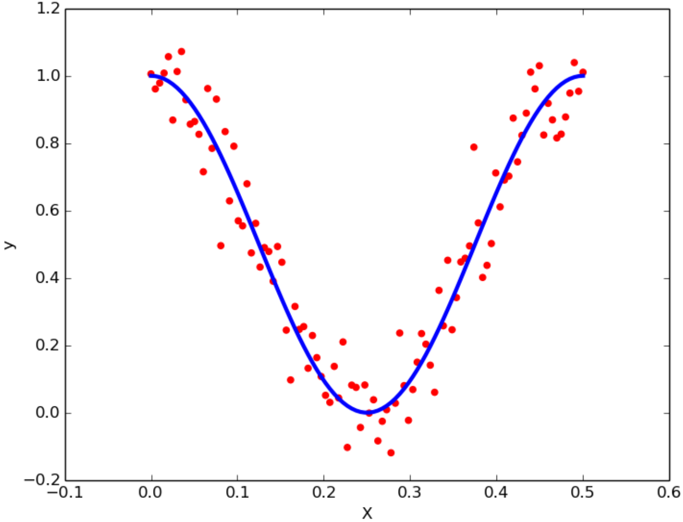
Plot underlying function y_actual and y



1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.1
0.2
0.3
0.4
0.5
X

Assignment 3: Polynomial Curve Fitting

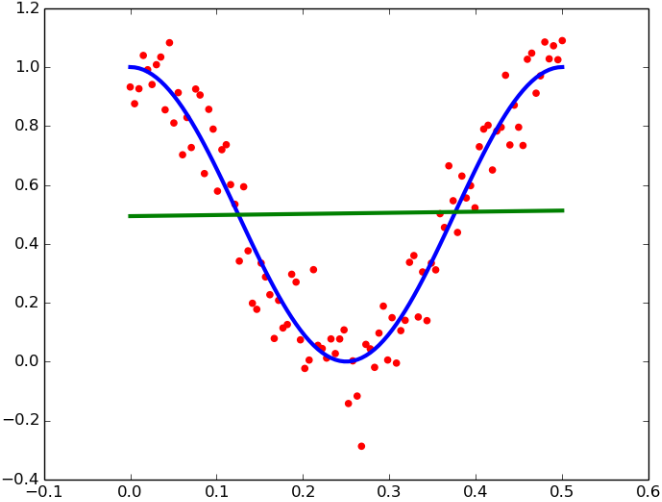
Dataset Generation



Assignment 3: Polynomial Curve Fitting

Fit linear regression model

Assignment 3: Polynomial Curve Fitting



Assignment 3: Polynomial Curve Fitting

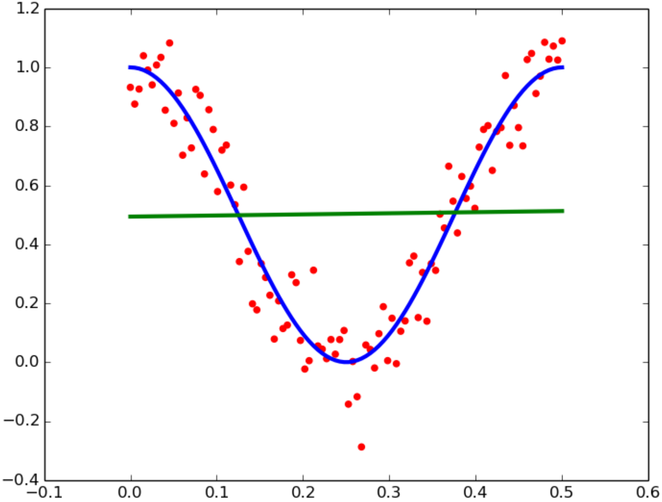
- Import

  from sklearn.preprocessing import PolynomialFeatures
  from sklearn.pipeline import Pipeline

1. deg=2
2. polynomial_features = PolynomialFeatures(degree=deg,include_bias=True)
3. linear_regression = linear_model.LinearRegression()
4. pipeline = Pipeline([("polynomial_features", polynomial_features), ("linear_regression", linear_regression)])
5. pipeline.fit(X_train, y_train)

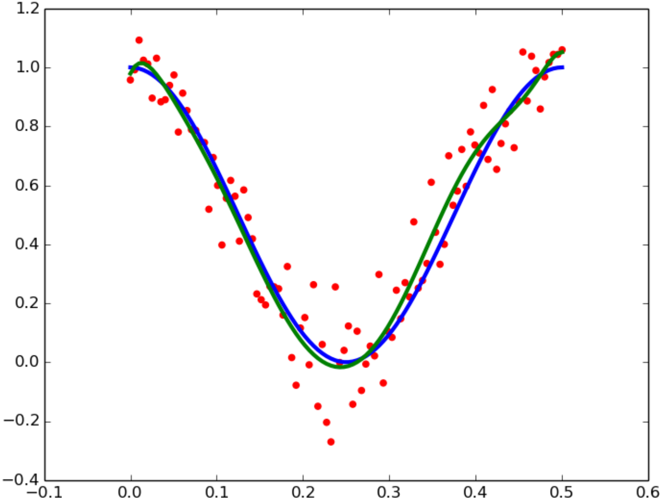
Assignment 3: Polynomial Curve Fitting

deg=1



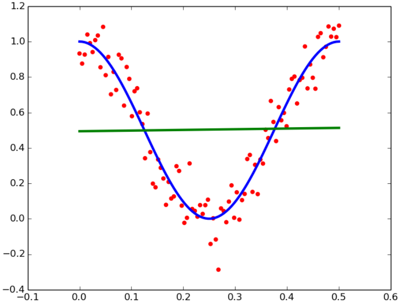
Assignment 3: Polynomial Curve Fitting

deg=10

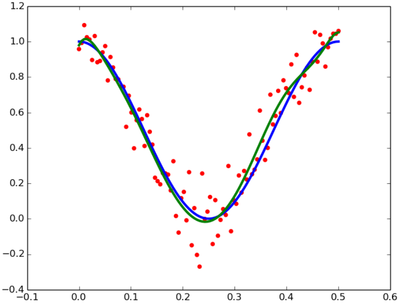


Assignment 3: Polynomial Curve Fitting

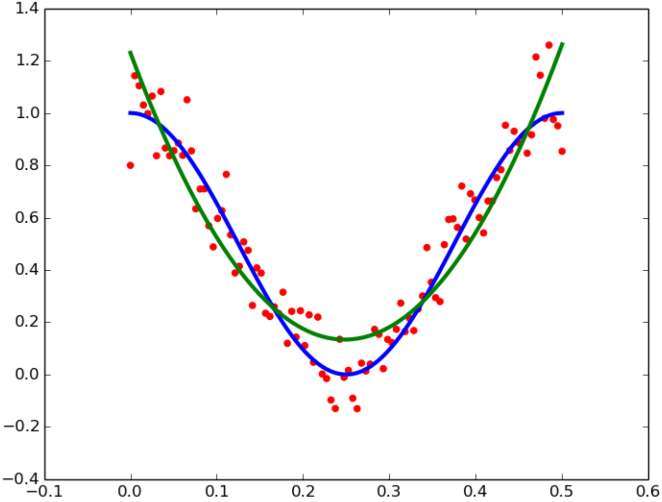
Underfiting



Overfiting



Assignment 3: Polynomial Curve Fitting



Regularization

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y(i))^2 + \lambda\theta^T\theta$$

λ is the regularization parameter

Regularization

```
model = linear_model.Ridge(alpha=0.001, fit_intercept=True)
```

To use sample datasets in Sklearn

```
from sklearn import datasets
data = datasets.load_breast_cancer()
X=data.data
y=data.target
```

Logistic Regression

```
classifier = linear_model.LogisticRegression()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```