PETR 5313: CRN 38950, Fall 2017
Numerical Application in Petroleum Engineering, Lesson 07: Linear Regression

Ekarit Panacharoensawad, PhD
Terry Fuller Petroleum Engineering Research Building Room 236
ekarit.panacharoensawad@ttu.edu

Cite as: Panacharoensawad, E. (2017) "Numerical application in petroleum engineering, Lesson 07: Linear Regression", presentation slide for Texas Tech PETR 5313 Fall 2017

Linear Regression: Outline

- Closed-form for least square linear regression
- Regularization, L1, L2, elastic net
- Data Preparation
  - Scaling
- Coding the model
  - Making pipeline
  - Making user define estimator (user class)
- Model performance
  - Test / Train / Validation sets
  - Cross-validation
  - Learning curve

Linear least squares closed-form solution

Regression model

$$f(x, \beta) = \sum_{j=1}^{m} \beta_j \phi_j(x)$$

Linear equation $$f(x, \beta) = \beta_0 \phi_0(x) + \beta_1 \phi_1(x)$$

$$f(x, \beta) = \beta_0 + \beta_1 x$$

$$\phi_0 = 1$$

$$\phi_1 = x$$

Least squares: Getting beta values

1 Data point case

x_1	x_2	$\cdots$	x_n	y
1	3.25	$\cdots$	-2.5	200

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \end{bmatrix}$$

Residual: actual - prediction

$$r = y - \sum_{j=1}^{n} x_j \beta_j$$

Least squares: Getting beta values...

Many Data points case

x_1	x_2	$\cdots$	x_n	y
1	3.25	$\cdots$	-2.5	200
1	-3.25	$\cdots$	-50	21.5
1	3.25	$\cdots$	-2.5	16.5
1	-3.25	$\cdots$	0	19.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	3.25	$\cdots$	-50	2.05

$$X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}$$

To get a unique solution, we need n < m.

Least squares: Getting beta values...

Many Data points case (notice m and n in beta and y)

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \qquad X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}$$

$$r_i = y_i - \sum_{j=1}^{n} X_{ij}\beta_j$$

Least squares: Getting beta values...

The best fit line does not pass through every points

- We just need to find betas that minimize the sum square error (SSE)

$$SSE = S = \sum_{j=1}^{m} r_i^2$$

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} |y_i - \sum_{j=1}^{n} X_{ij}\beta_j|^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$$

n columns in X must be linearly independent to get unique solution

https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)

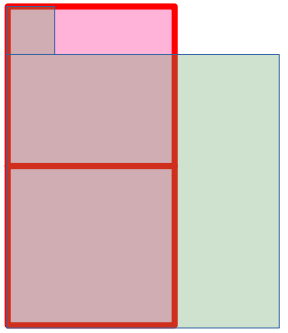
One reason for sum square error

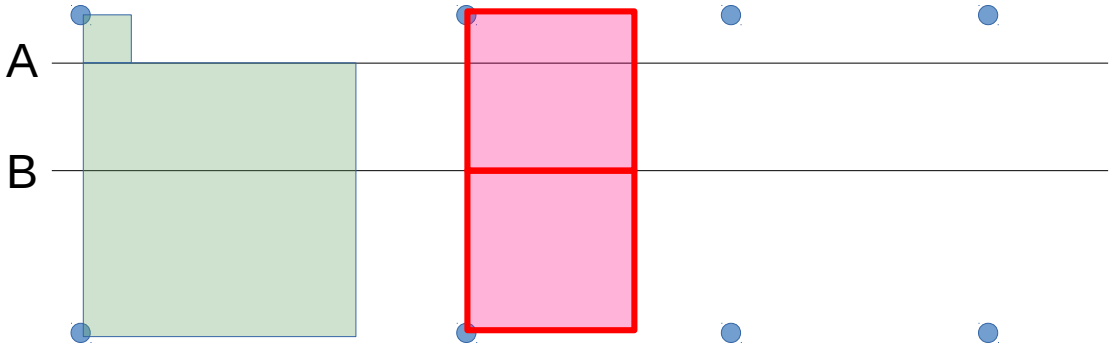
If we use just sum of absolute distance to the lines,

- A and B C are equally good

If we use sum square error, (equivalent to use area)

- We give more weight to the larger deviation cases





Area from case B is less than the area from case A

Least squares: Getting beta values...

Betas that allow SSE to be minimized is defined as $\hat{\beta}$

β_j value that minimize SSE can be found from $\frac{\partial}{\partial \beta_j}$ and set the result to be zero

$$\frac{\partial S}{\partial \beta_j} = \frac{\partial}{\partial \beta_j}\left(\sum_{i=1}^{m} \left| \, y_i - \sum_{j=1}^{n} X_{ij}\beta_j \, \right|^2 \right)$$

$$= \frac{\partial}{\partial \beta_j}\left(\left| \, y_1 - \sum_{j=1}^{n} X_{1j}\beta_j \, \right|^2 + \cdots + \left| \, y_n - \sum_{j=1}^{n} X_{nj}\beta_j \, \right|^2 \right)$$

$$\because \frac{\partial}{\partial \beta_j} \left| \, y_i - \sum_{j=1}^{n} X_{ij}\beta_j \, \right|^2 = 2r_i \frac{\partial}{\partial \beta_j}\left(- \sum_{j=1}^{n} X_{ij}\beta_j \right) = -2r_i X_{ij}$$

Least squares: Getting beta values

$$\therefore \frac{\partial S}{\partial \beta_j} = -2\sum_{i=1}^{m} r_i X_{ij} = -2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)X_{ij}$$

At min(SSE), derivative = 0

$$0 = \sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\hat{\beta}_k\right)X_{ij}$$

$$\sum_{i=1}^{m} X_{ij}y_i = \sum_{i=1}^{m}\sum_{k=1}^{n} X_{ij}X_{ik}\hat{\beta}_k$$

Least squares: Getting beta values

$$\sum_{i=1}^{m} X_{ij} y_i = \sum_{i=1}^{m} \sum_{k=1}^{n} X_{ij} X_{ik} \hat{\beta}_k$$ is only for one j value

We actually have n of the above equation (j = 1, 2, …, n)

$\boldsymbol{X}$ is mxn matrix. $\boldsymbol{y}$ is mx1 matrix. $\boldsymbol{\beta}$ is nx1 matrix.

In matrix form, we have

$$\sum_{i=1}^{m} X_{ij} y_i \rightarrow \boldsymbol{X^T y}$$ $\boldsymbol{X^T y}$ is nx1 matrix

$$\sum_{i=1}^{m} \sum_{k=1}^{n} X_{ij} X_{ik} \hat{\beta}_k \rightarrow \left(\boldsymbol{X^T X}\right) \hat{\boldsymbol{\beta}}$$

Least squares: Getting beta values...

$$\boldsymbol{X^T y} = \begin{bmatrix} X_{11} & X_{21} & \cdots & X_{m1} \\ X_{12} & X_{22} & \cdots & X_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ X_{1n} & X_{2n} & \cdots & X_{mn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\boldsymbol{X^T y} = \begin{bmatrix} \sum_{i=1}^{m} X_{i1} y_i \\ \sum_{i=1}^{m} X_{i2} y_i \\ \vdots \\ \sum_{i=1}^{m} X_{in} y_i \end{bmatrix}$$

Least squares: Getting beta values...

$$\boldsymbol{X^T X} = \begin{bmatrix} X_{11} & X_{21} & \cdots & X_{m1} \\ X_{12} & X_{22} & \cdots & X_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ X_{1n} & X_{2n} & \cdots & X_{mn} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix}$$

$$\boldsymbol{X^T X} = \begin{bmatrix} \sum_{i=1}^{m} X_{i1}X_{i1} & \sum_{i=1}^{m} X_{i1}X_{i2} & \cdots & \sum_{i=1}^{m} X_{i1}X_{in} \\ \sum_{i=1}^{m} X_{i2}X_{i1} & \sum_{i=1}^{m} X_{i2}X_{i2} & \cdots & \sum_{i=1}^{m} X_{i2}X_{in} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{m} X_{in}X_{i1} & \sum_{i=1}^{m} X_{in}X_{i2} & \cdots & \sum_{i=1}^{m} X_{in}X_{in} \end{bmatrix}$$

Least squares: Getting beta values...

$$\left(\boldsymbol{X^T X}\right)\hat{\boldsymbol{\beta}} = \begin{bmatrix} \sum_{i=1}^{m} X_{i1}X_{i1} & \sum_{i=1}^{m} X_{i1}X_{i2} & \cdots & \sum_{i=1}^{m} X_{i1}X_{in} \\ \sum_{i=1}^{m} X_{i2}X_{i1} & \sum_{i=1}^{m} X_{i2}X_{i2} & \cdots & \sum_{i=1}^{m} X_{i2}X_{in} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{m} X_{in}X_{i1} & \sum_{i=1}^{m} X_{in}X_{i2} & \cdots & \sum_{i=1}^{m} X_{in}X_{in} \end{bmatrix} \begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_n \end{bmatrix}$$

$$\left(\boldsymbol{X^T X}\right)\hat{\boldsymbol{\beta}} = \begin{bmatrix} \sum_{k=1}^{n} \sum_{i=1}^{m} X_{i1}X_{ik}\hat{\beta}_k \\ \sum_{k=1}^{n} \sum_{i=1}^{m} X_{i2}X_{ik}\hat{\beta}_k \\ \vdots \\ \sum_{k=1}^{n} \sum_{i=1}^{m} X_{in}X_{ik}\hat{\beta}_k \end{bmatrix}$$

Least squares: Getting beta values...

$\hat{\boldsymbol{\beta}}$ can be calculated by solving system of linear equation

Ax = b

$$(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})\hat{\boldsymbol{\beta}} = \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

In this case $A = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})$

$$b = \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

Polynomial Regression (still linear)

x^0	x^1	x^2	$\cdots$	x^n	y
1	1	1	$\cdots$	1^n	200
1	2	4	$\cdots$	2^n	21.5
1	3	9	$\cdots$	3^n	16.5
1	4	16	$\cdots$	4^n	19.8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	100	10000	$\cdots$	100^n	2.05

After setting x from degree 0 to degree n, the rest is the same

$$(\boldsymbol{X^T X})\hat{\boldsymbol{\beta}} = \boldsymbol{X^T y}$$

Polynomial Regression

First import the class PolynomialFeatures

- `from sklearn.preprocessing import PolynomialFeatures`

Second: Create a variable to be that class

- `poly = PolynomialFeatures(degree = 3)`

Third: Use class method to create polynomial matrix

- x_poly = poly.fit_transform(x.reshape(-1,1))
- x.reshape(-1,1) is for make a column vector
- X = [x1, x2, x3, x4]
- x.reshape(-1,1) = [[x1], [x2], [x3], [x4]]

-1 in reshape means, adjust the number of row automatically

Polynomial Regression

`x = np.linspace(-100,100,200)` create array of 1 row, 200 members.

Start and End indices of `.linspace` function are inclusive

`x.shape` gives `x(200, )`

x_poly = poly.fit_transform(x.reshape(-1,1)) gives

```
array([[1., -100. ,   10000.,   -1000000.],
       [1.,-98.995, 9800.005, -970151.254],
        ..,    ...,      ...,          ... ,
 ......[1.,  100. ,   10000.,    1000000.]
```

Polynomial Regression (manual)

```
x = np.linspace(-100,100,200)
y = 3*x**3 - 7*x**2 + 16*x - 19
poly = PolynomialFeatures(degree = 3)
x_poly = poly.fit_transform(x.reshape(-1,1))
XTX = x_poly.T.dot(x_poly)
XTy = x_poly.T.dot(y.reshape(-1,1))
Beta = sp.linalg.solve(XTX,XTy)
In : Beta
Out: array([[-19.], [-16.], [ -7.], [ 3.]])
```

Over-fitting

Higher degree of polynomial can fit data better, but cannot extrapolate

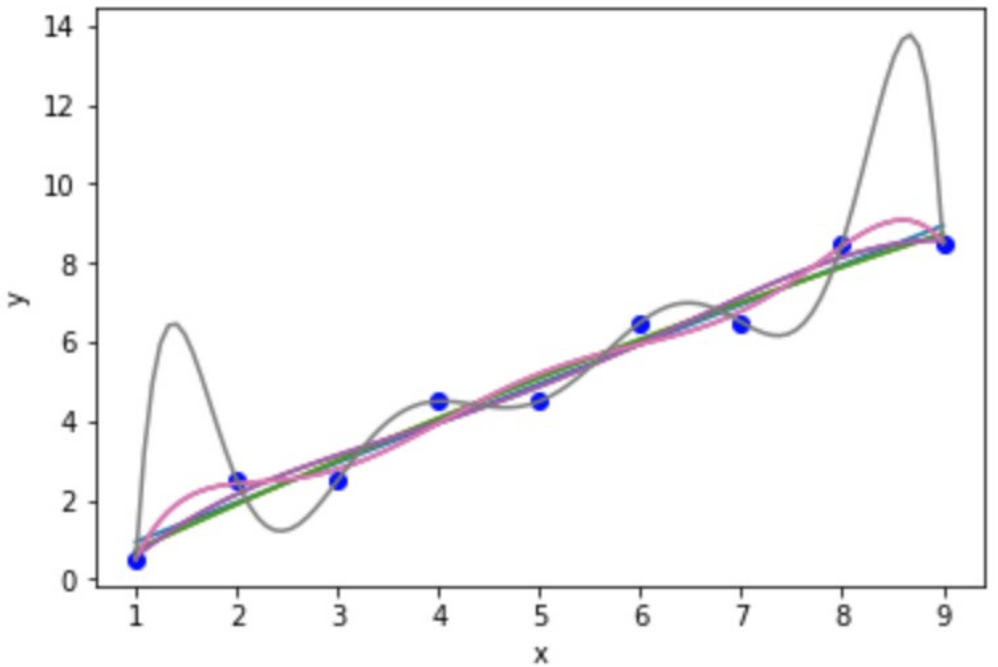
Which line have highest r2 score?

- Deg 8 line

Which line is promising

for extrapolation?

- Deg 1 line



Checking the model performance

Train-Test-Split

- Separate 20% of the data out and use them at the end (test data set)
- Use 80% of the data (train dataset) to train the model

Over-fitting

- Model can match (memorize) the train dataset very well but cannot be used to accurately predicting the test data set

Under-fitting

- Model cannot predict the train dataset

Train-Test-Split Code

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = (
            train_test_split(x , y, test_size = 0.2,
            random_state = 2))
```

random_state is to allow a repeatable result

Same random state means that the sample will be shuffled the same way.

Data Leaking and Leaking type

Data leaking is dangerous because we may think that the model is good, but when we really use it in the real world, it does not work

Leaking Type

Not using test set at the end

- Looking at the test set and do data preprocessing and model selection
- Adjust the hyper parameters (parameters in model) to make a good match with the test set

Use the answer to predict other answers

- Use one or more of the target values (dependent variable) as the feature (independent variable)

Data Leaking in time series data

Leaking Type

Looking from the hindsight

- Use the data to be predicted as the input
- Preprocessing / Model assumptions are based on the information that is not known at the time of the prediction
- Accidentally shuffle the train dataset, causing the future information to be available at the model prediction time

Cross-Validation (concept)

Cross-validation is the way to tell model performance of various models without using the test dataset

Situation: We have 100 models and want to find the best

These 100 model can be the same algorithm with just different hyper-parameters in the model

1) Exclude the test set from the cross-validation process

2) Divide the rest of the data into k-folds (groups)

One group is used as the validation group, the rest is used as the training group. Validate the model k-times to get the cross-validation score (can be average). Then do this for every model.

Cross Validation Score Code

`cross_val_score` Function is for calculating the score (e.g. r2_score) by equally splitting the x domain into 3 folds (default value)

- Data in 1 fold is used as the test set
- Data in another 2 folds are for training the model
- Once each fold is used as the test set, test scores from each training are reported

By default, `cross_val_score` use `KFold` function with default value (no shuffle) to create train/test sets

Cross Validation Score Code

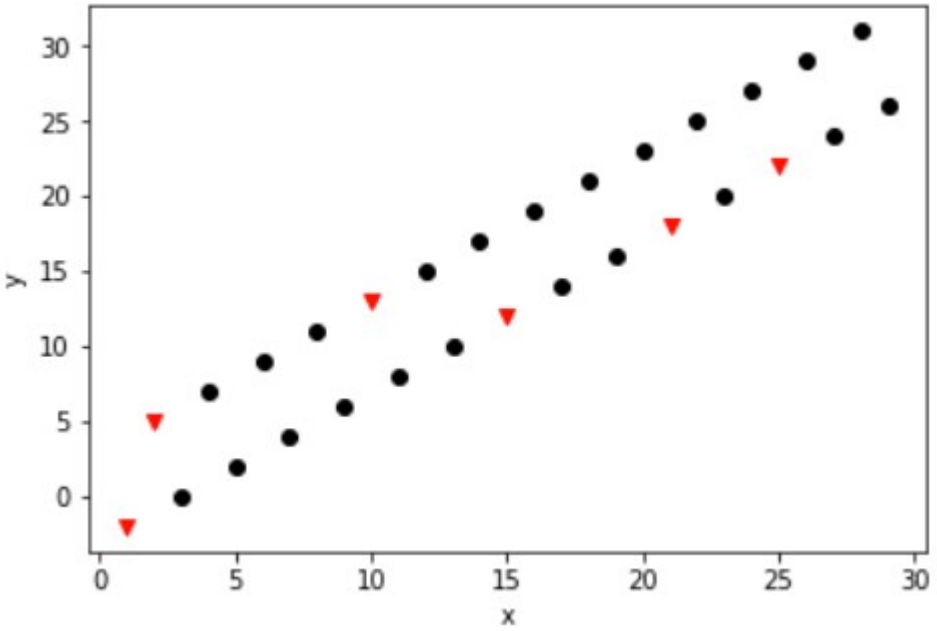
```
from sklearn.model_selection import cross_val_score
for deg in range(1,9):
    poly = PolynomialFeatures(degree = deg)
    x_poly = poly.fit_transform(x_train.reshape(-
1,1))
    model = LinearRegression()
    CV = KFold(n_splits=5, shuffle=False)
    score = cross_val_score(model, x_poly, y_train,
cv = CV)
    print(score, score.mean())
```

cross_val_score result

Result after training with polynomial degree 1 to 8

- Linear model is the best

```
[ 0.913  0.776  0.903  0.56   0.804] 0.791294014372
[ 0.863  0.773  0.905  0.53   0.769] 0.767837298894
[ 0.865  0.779  0.894  0.261  0.76 ] 0.711904255113
[ 0.859  0.78   0.838  0.138  0.777] 0.678502982241
[ 0.857  0.638  0.72  -0.021  0.68 ] 0.574852667831
[ 0.87   0.596  0.773  0.082  0.468] 0.55774095335
[ 0.872  0.571  0.829  0.082  0.15 ] 0.500972850349
[ 0.469  0.442  0.866  0.099  0.409] 0.456874139338
```



x_train are black dots

Note that test set was not used in cross_val_score calculation

Regularization Method on Linear Regression

Regularization method is to penalize complicated models.

With no regularization

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} |y_i - \sum_{j=1}^{n} X_{ij}\beta_j|^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$$

With L1 regularization

$$\min_{\beta_0,\beta} \left\{ \frac{1}{N} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq \alpha.$$

With L2 regularization

$$\min_{\beta_0,\beta} \left\{ \frac{1}{N} \sum_{i=1}^{N} (y_i - \beta_0 - x_i^T \beta)^2 \right\} \text{ subject to } \|\beta\|_2 \leq \alpha.$$

L_p norm

$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}}$$

$$\|x\|_\infty = \mathbf{max}\{|x_1|, |x_2|, \ldots, |x_n|, \}$$

For

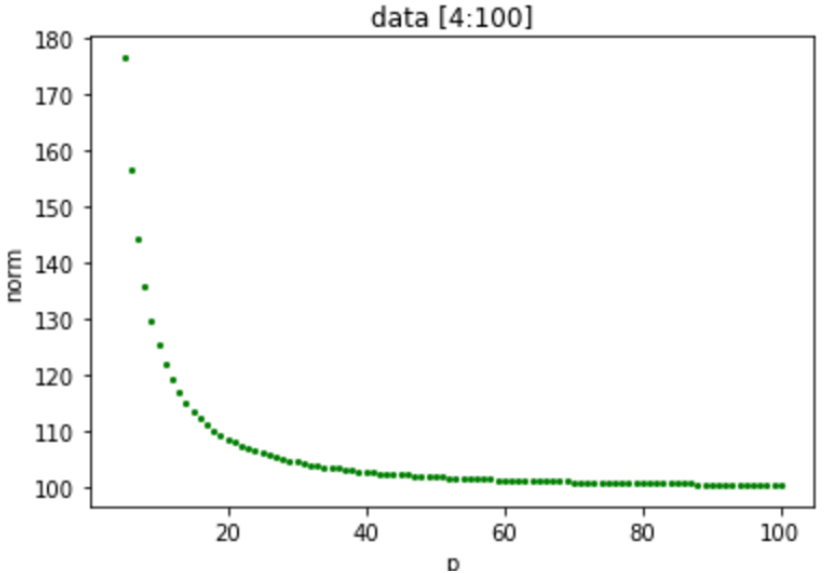
```
a = np.arange(1, 101, dtype = np.longdouble)
```

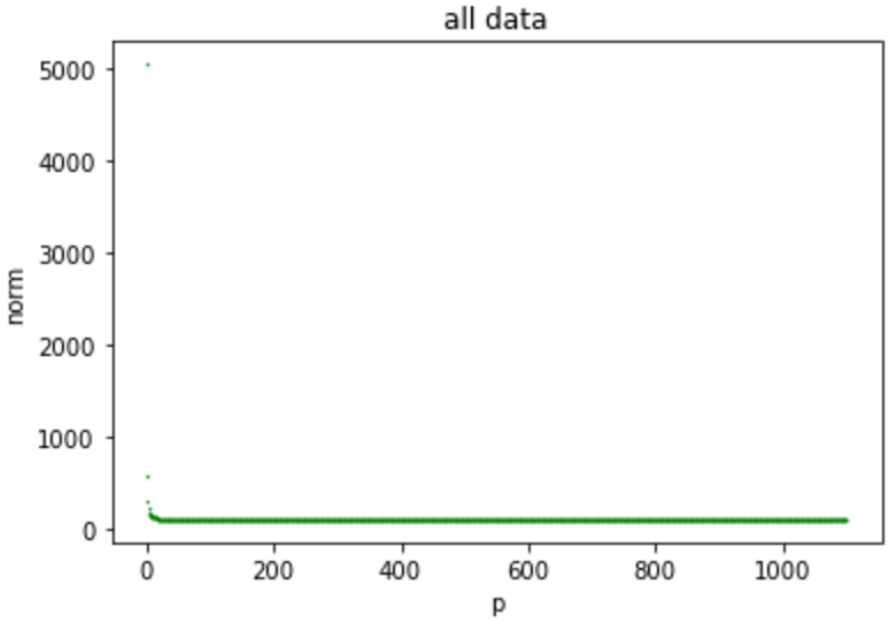
- Number from `1.0 to 100.0`

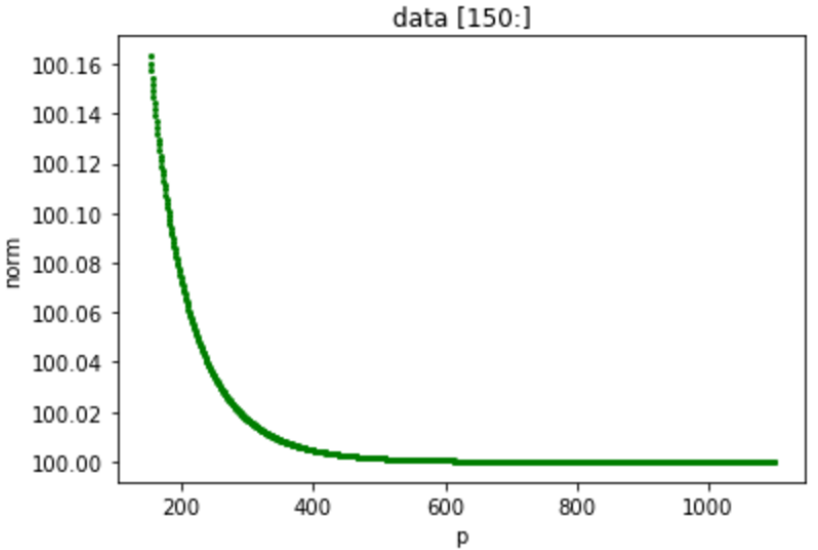
L_p norm

As p increases, the norm value put more weight on the large value

For [1,2,3,4,…,100], Norm value approaches infinity as $p \to \infty$







Elastic Net Regularization

For r = 0, Elastic net become Ridge regression

For r = 1, Elastic net become Lasso regression

For 0 < r < 1, Elastic net is the combination of L1 and L2 regularization methods

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + r\alpha \sum_{i=1}^{n} |\beta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \beta_i^2)$$

Better than Lasso, for correlated independent variables case

- Used in support vector machine classification (not cover)

L1 & L2 Regularization (LASSO & Ridge)

LASSO (Least Absolute Shrinkage and Selection Operator)

- Lasso (L1) tends to make the weight (slope) for irrelevant factor to be zero

Ridge Regression (L2, Tikhonov regularization)

- If we don’t know for sure that only fews features matter, avoid Lasso, use Ridge regression

Elastic net regularization: Better than Lasso for the case of

- Several features are strongly correlated
- There are more features than training instances

Elastic net has more hyper-parameter to be adjusted!

LASSO Implementation

```
from sklearn.linear_model import Lasso
model = Lasso(alpha=100, max_iter=1000000)
model.fit(x_poly,y_train)
score = cross_val_score(model, x_poly, y_train, cv = 5)
score, score.mean()
```

The above code declare model to be class Lasso with the initialization of alpha and max_iter parameters. After model is fit with the x_poly and y_train, then cross validation score is calculated

Time Complexity

sklearn.linear_model.LinearRegression

sklearn.linear_model.Ridge

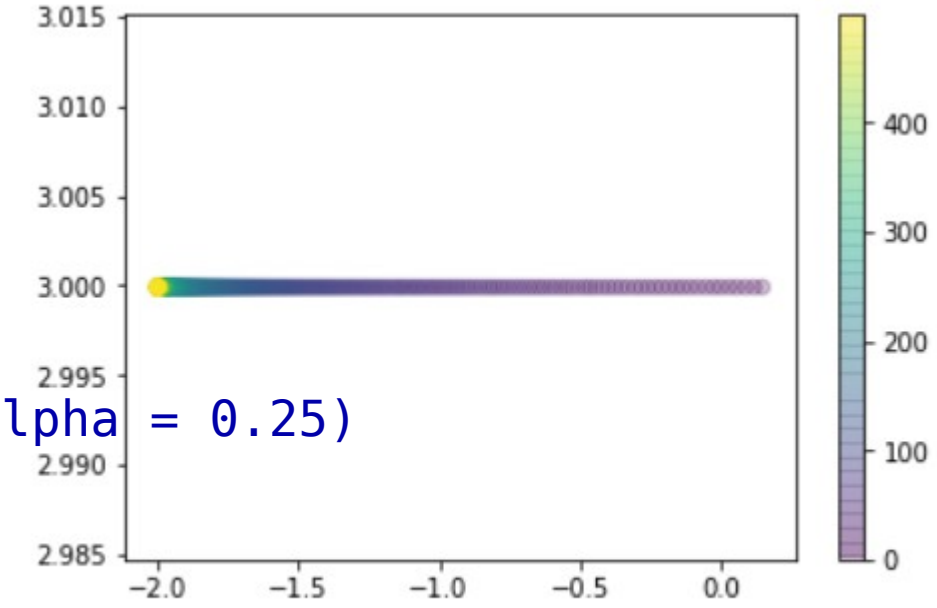
- $O(np^2)$, assume that $n \geq p$, p = number of coef_, n = number of data
- Do not require feature scaling

Sklearn.linear_model.SGDRegressor

- $O(knp_{av})$, k = number of iteration, p_{av} = average number of non-zero attributes per sample
- Require feature scaling
- Can do out-of-core fitting (data don't fit in RAM)

Stochastic Gradient Descent (without scaling)

```
sgd = SGDRegressor(max_iter = 2, penalty = None,
        warm_start = True,  #take the old fitting into consideration
        fit_intercept=True, learning_rate='constant', eta0 = 1e-3)
ans = [[],[],[]]
for i in range(500):
    sgd.fit(X.reshape(-1,1),y)
    ans[0].append(sgd.intercept_)
    ans[1].append(sgd.coef_)
    ans[2].append(i)
plt.scatter(ans[0],ans[1],c=ans[2], alpha = 0.25)
plt.colorbar()
plt.show()
print(ans[0][-1],ans[1][-1])
```




```
[-1.997936143067707] [ 2.999918975485236]
```

Stochastic Gradient Descent (without scaling)

warm_start = True is used so that the result from the current iteration will be used in the next iteration

```
for i in range(500):
    sgd.fit(X.reshape(-1,1),y)
```

In each iteration, sgd take at most 2 rows (`max_iter = 2`) to search for the fitting parameters (unknown constants) in the model. The training process is done after 500 iteration.

Stochastic Gradient Descent: Learning Rate

At each iteration, the next answer is calculated from

$$w_{n+1} = w_n - \eta \nabla Q_i(w)$$

- W is the vector of unknown constant in the model (can be polynomial coefficient)
- Q is the cost function. Default value is sum-square-error
- Q(w) means that w is the input to calculate Q(w)
- Eta is the learning rate (step-size)

The objective is to move w into the direction that Error (or Q) will decrease quickest.

Stochastic Gradient Descent: Learning Rate

If the learning rate is too much, we may always pass the answer. If the learning rate is too small, we may stop before reaching the answer

Learning rate for SGDRegressor in sklearn

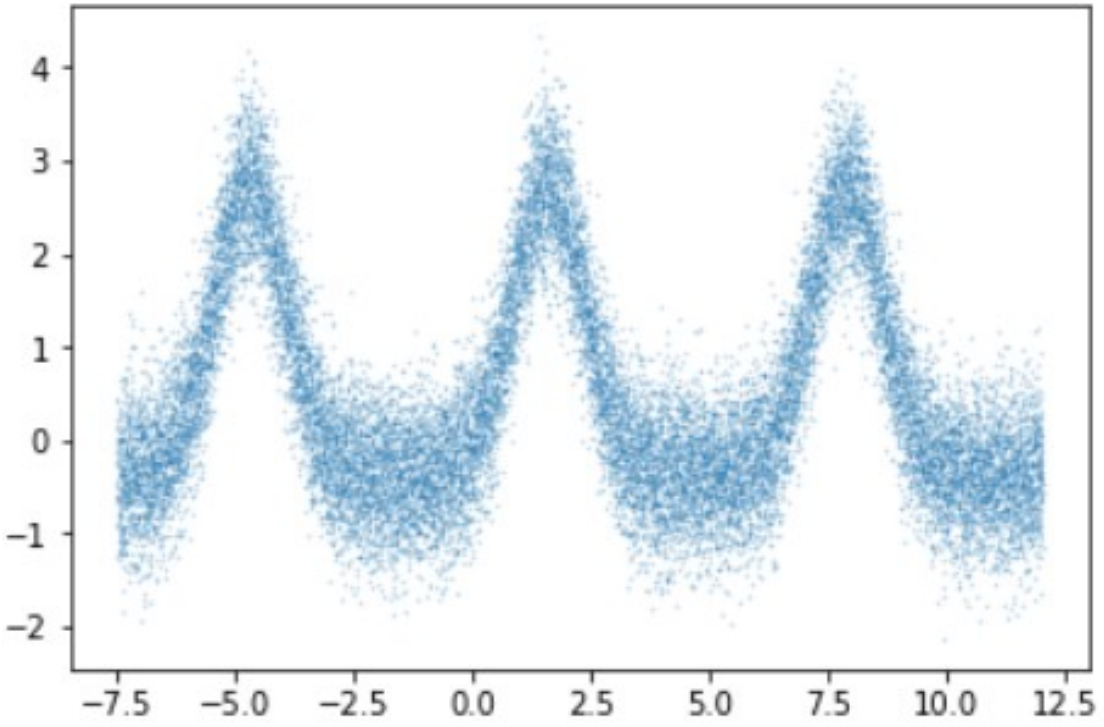
'constant' $\eta = \text{eta0}$

'optimal' $\eta = \frac{1}{\alpha(t_0 + t)}$

'invscaling' $\eta = \frac{\text{eta0}}{t^{\text{power\_t}}}$

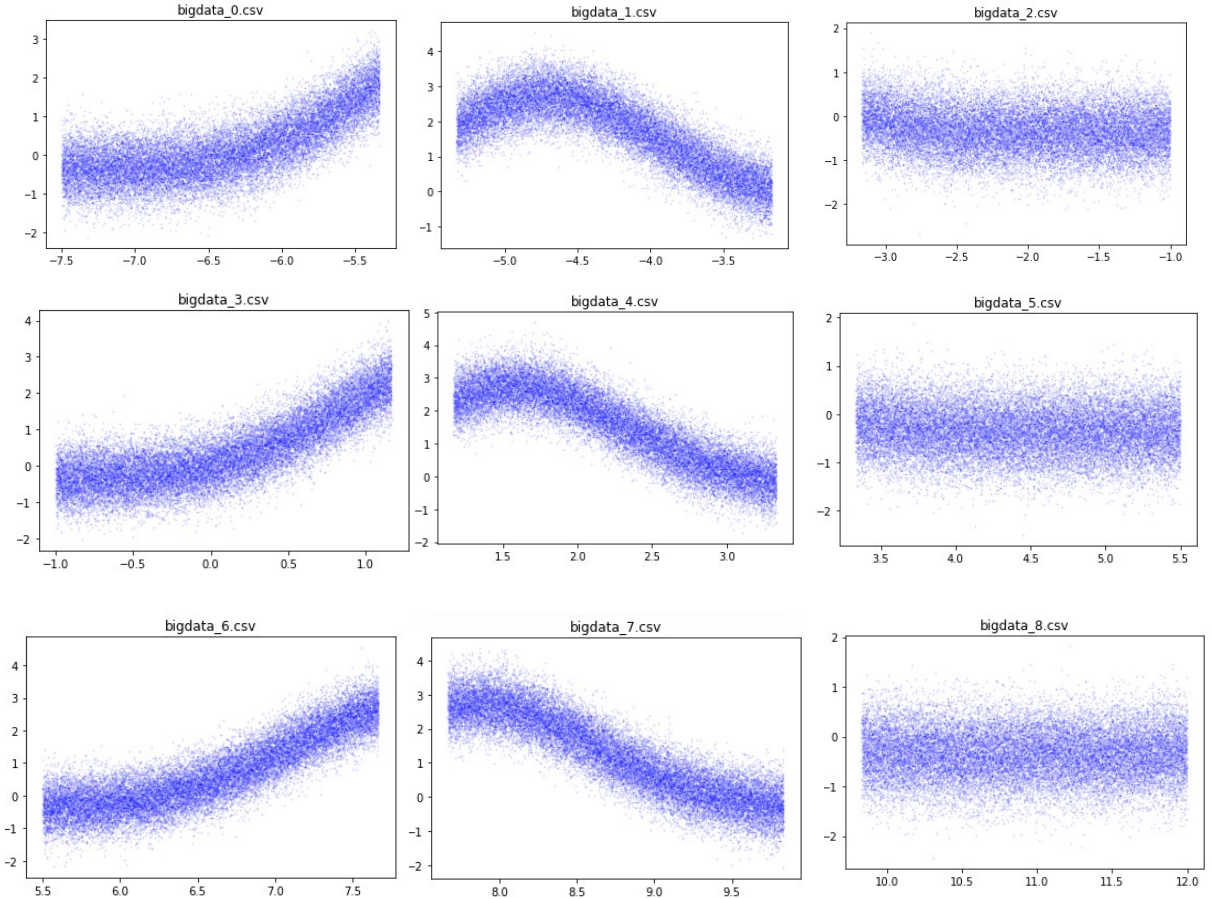
Incremental Learning with SGDRegressor

View of the data (10% of the total data, selected randomly)



Incremental Learning with SGDRegressor

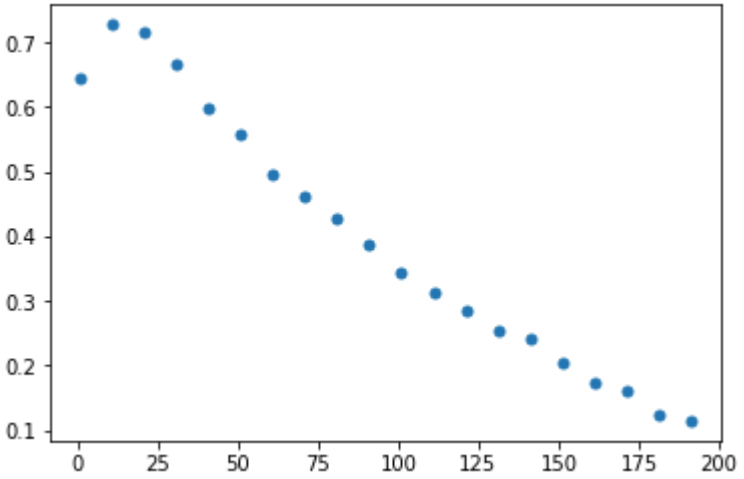
Create 9 files to keep each part of the data (21k lines each)



- Feed the data into SGD file by file
- Preprocessing (scaling) is based on the SD and x-bar of the data in the first file
  - So that one scaling method is used for all data

SGDRegressor Result

The plot is the r-squared value versus training iteration



Train the model to many times can

- Make step size decrease, model is not sensitive to later data
- Over-fit the model with noise in data

SGDRegressor Result...

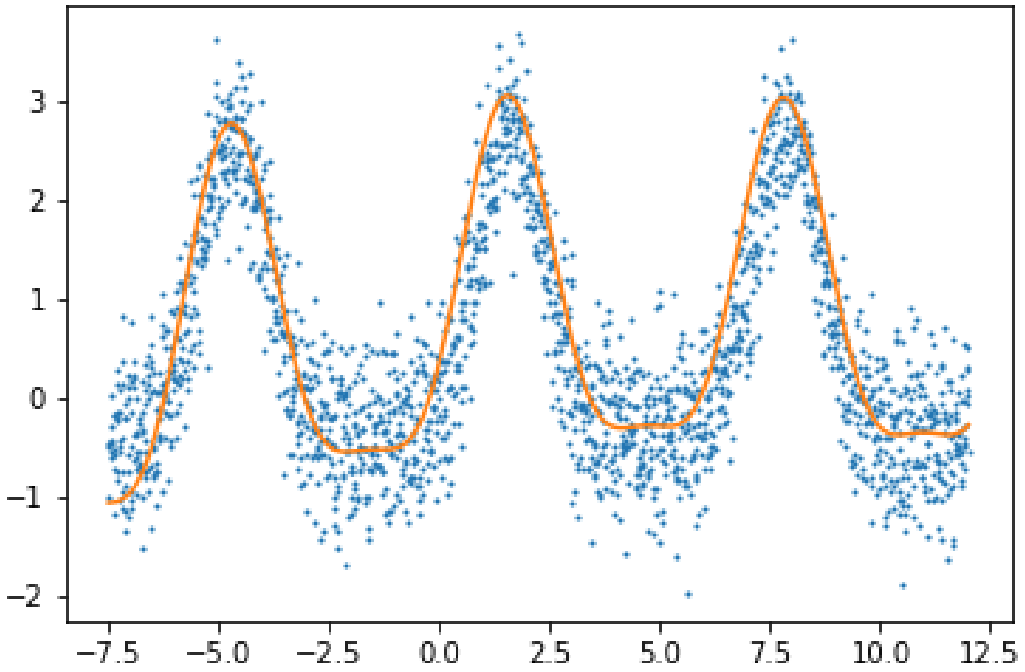
We note that the model can fit the data correctly (if we stop early) even though it see only one part of the data at a time!

Bases for curve fittings are

C0 + C1*x + C2 sin(x) + C3 cos(x) +
(C0 + C1*x + C2 sin(x) + C3 cos(x))**2

True result is from
np.sin(x)*np.exp(np.sin(x))

Noise is from
noise = np.random.normal(size = y_.shape)
y_real = y_ + 0.5 * noise



r2 score = 0.73959262555

Reference:

- Wikipedia
  - https://en.wikipedia.org/wiki/Tikhonov_regularization
  - https://en.wikipedia.org/wiki/Lasso_(statistics)
  - Geron, A. “Hands-On Machine Learning with Scikit-Learn & TensorFlow”, O’Reilly, Boston, USA
- Scikit-learn documentation
  - http://scikit-learn.org/
  - http://scikit-learn.org/stable/modules/sgd.html
- Stackoverflow
  - https://stackoverflow.com/questions/31443840/sgdregressor-nonsensical-result