Feature Importance, Lasso, Ridge, Elastic Net

Prereq (To understand this lecture, you must have already mastered):

1. Multivariate Regression

Topics:

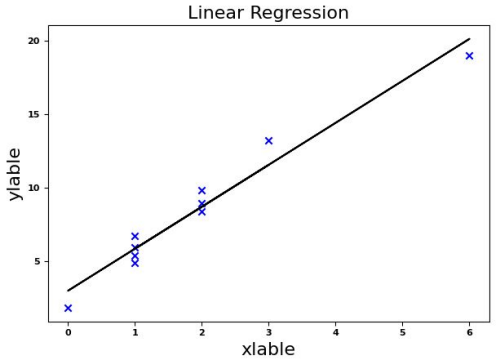
1. Feature Importance
2. Lasso
3. Ridge regression
4. Elastic Net
5. The Horvath Clock.

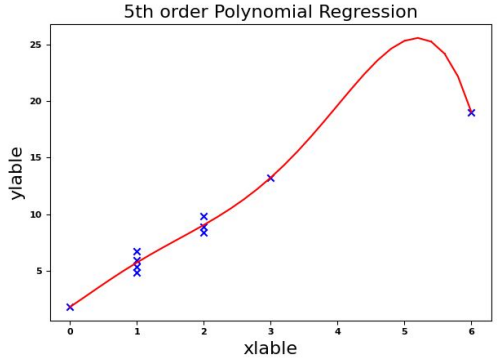
Lecture Written by : Prof. Wu
@ chiehwu.com

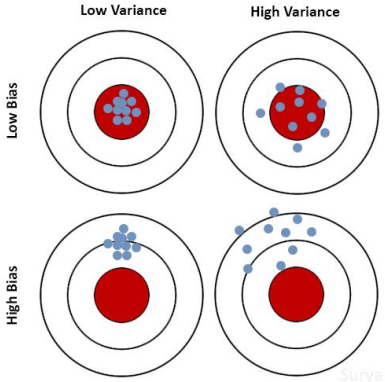
Let's Have A Quick Recap of the Previous Lecture

We learned that when we perform regression

- We want to minimize both the **bias and the variance.**
- However, this often leads to the **bias variance trade off**
  - When we use a more complicated model to lower the bias, it leads to models with high variance.
  - When the variance is too high, it is called overfitting, we used an overly complex model.
  - This can be seen in the two plots below.
- Today, we learn how to use a complicated model while minimizing the variance, this is called **regularization.**





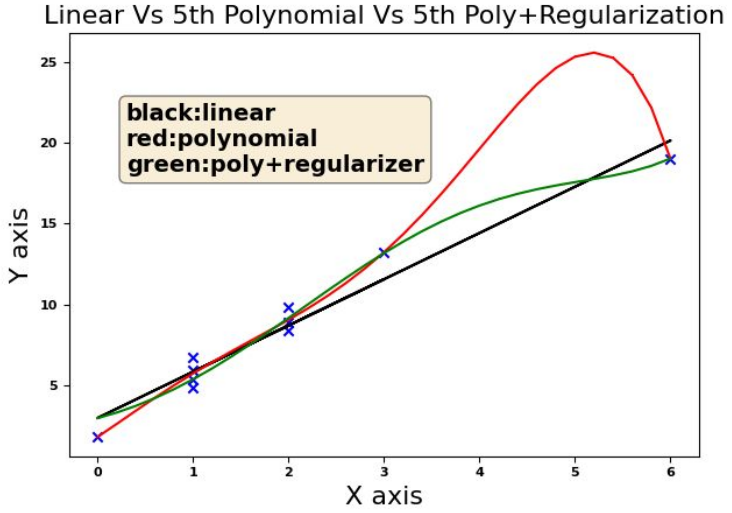


So how does Regularization work?

- When we perform regression, we aim to minimize the average square of the prediction error.
- Regularization works by adding an addition **constraint term** into the minimization problem

$$\min_f \frac{1}{n}\sum_i (f(x_i) - y_i)^2 \quad \xrightarrow[\textbf{extra term}]{\textbf{add an}} \quad \min_f \underbrace{\frac{1}{n}\sum_i (f(x_i) - y_i)^2 + \textbf{Constraint}}_{\textbf{We now minimize this whole thing}}$$

- As a result of adding an extra **Constraint term**.
  - We can continue to use very complicated models
  - While not increasing much variation (shown in figure)
  - Black is a simple linear model.
  - Red is a complex 5th-order polynomial model.
  - Green is also a complex 5th order with constraint.
  - The constraint created a compromise!
- Regularization: Use complicated models without overfitting.



So what function should we add as the Constraint?

Given the Regression function with the **constraint function**

$$\min_f \quad \underbrace{\frac{1}{n}\sum_{i=1}^{n}(f(x_i)-y_i)^2}_{\textbf{Original Regression term}} \quad + \quad \underbrace{\textbf{Constraint function}}_{\textbf{Function is outside of the summation}}$$

There are infinite possible constraint functions you can use.

- However, **3 constraint functions** are the most basic and commonly used.
- Depending on the constraint, the new objective results in a different algorithm.

1. If we use L1 norm of w as constraint then it is called Lasso :

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1$$

2. If we use L2 norm squared of w as constraint then it is called Ridge Regression :

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_2^2$$

3. If we use L1 and L2 norm of w as constraint then it is called Elastic Net :

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1 + ||w||_2^2$$

Practice Taking More Derivative

- Once we add a constraint, Lasso and Elastic Net no long have a closed-form solution.
- We have to use Gradient Descent to solve them.
- Therefore, we just have to know how to take derivatives for each equation.
- Luckily, by now you should be able to do this easily.
- For your 1st exercise, take the derivative of these 3 equations.

$$\textbf{Lasso :} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1$$

$$\textbf{Ridge Regression:} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_2^2$$

$$\textbf{Elastic Net :} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1 + ||w||_2^2$$

Solution to Derivatives of Regularization Objectives

$$\textbf{Lasso :} \quad \frac{df}{dw} \frac{1}{n} \sum_{i=1}^{n} (\phi(x_i)^\top w - y_i)^2 + ||w||_1 = \frac{2}{n} \sum_i (\phi(x_i)^\top w - y_i)\phi(x_i) + sign(w)$$

$$\textbf{Ridge Regression:} \quad \frac{df}{dw} \frac{1}{n} \sum_{i=1}^{n} (\phi(x_i)^\top w - y_i)^2 + ||w||_2^2 = \frac{2}{n} \sum_i (\phi(x_i)^\top w - y_i)\phi(x_i) + 2w$$

$$\textbf{Elastic Net :} \quad \frac{df}{dw} \frac{1}{n} \sum_{i=1}^{n} (\phi(x_i)^\top w - y_i)^2 + ||w||_1 + ||w||_2^2 = \frac{2}{n} \sum_i (\phi(x_i)^\top w - y_i)\phi(x_i) + sign(w) + 2w$$

Why do we call them constraints ?

$$\textbf{Lasso :} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1$$

$$\textbf{Ridge Regression:} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_2^2$$

$$\textbf{Elastic Net :} \quad \min_f \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + ||w||_1 + ||w||_2^2$$

Notices that as the size of the weight w increase, the total value of the objective also increase.

Therefore, using a weight vector w that are large goes against the minimization objective.

(The constraints constrains the solution to smaller values)

This is why the constraint is also called the **penalty term**.

(*It penalizes solutions with large weights values*)

Adding the constraint is like adding an “and” statement saying that
I want to achieve my regression objective and I want the weights to be as small as possible.

We can add finer control over the regularizers by adding λ

$$\textbf{Lasso :}\ \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda||w||_1$$

$$\textbf{Ridge Regression:}\ \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda||w||_2^2$$

$$\textbf{Elastic Net :}\ \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda_1||w||_1 + \lambda_2||w||_2^2$$

- The λ term is called **Hyperparameters**, it is a scalar value.
- It acts as a weight term to control how much penalty you want.
- If it 0, the regularizer disappears.
- But if you make it into a larger value, it means that you really want the norm $|w|$ to be very small.

Let’s understand feature importance !

Next, let's better understand Linear Regression Weights

Given the following data on cancer predictions

Protein 1	Protein 2	Protein 3	Percentage %
2	3	4	32
3	1	7	27
1	0	0	7
6	7	9	84
...	...	...	...

Table 1: Using Protein to Predict Cancer Rate

After using linear regression, we identified the weights as

$protein1$	$protein2$	$protein3$	1
w_1	w_2	w_3	1
7	6	0.00001	0

$$\implies \quad \Phi w = \hat{y} \quad \implies \quad \begin{bmatrix} 2 & 3 & 4 & 1 \\ 3 & 1 & 7 & 1 \\ 1 & 0 & 0 & 1 \\ 6 & 7 & 9 & 1 \\ \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} 7 \\ 6 \\ 0.00001 \\ 0 \end{bmatrix} = \begin{bmatrix} 32 \\ 27 \\ 7 \\ 84 \end{bmatrix}$$

If we look at the weights carefully, how much does each Protein impact the final percentage?

- Notice how Pretein 3's weight is so small, it doesn't impact the final prediction.
- Therefore, it doesn't matter much the value of protein 3 because its impact on the final prediction is very small after multiplying by w_3.
- This observation tells us that the weights of **linear regression** not only give us the function, it also tell how much each feature contributes to the final prediction.

The constraints create a compromise solution, but when do we use L1 vs L2 norm?

The L1 and L2 norm forces the weight to be small, but **L1 norm** forces w to be small in a special way.

Below are the results from
1. Linear Regression
2. Linear Regression with L2
3. Linear Regression with L1

Linear Regression with L1
w = [2.1 1.0 0]
Accuracy Score: 0.86

L1 pushes the weights smaller, but it makes none important weights nearly 0.

Therefore, we can **identify important factors** easier with L1 regularization, at the cost of lower accuracy.

True Solution
w = [3 2 0]

Regular Linear Regression
w = [3.1 2.0 -0.03]
Accuracy Score: 0.999

Regular Regression gives you the most accurate result

Linear Regression with L2
w = [2.7 1.8 0.25]
Accuracy Score: 0.99

L2 makes the weights smaller and gives you slightly less accurate result

Hyperparameters and Identifying Regularization Weights λ with Train/Validation/Test method

$$\textbf{Lasso}: \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda||w||_1$$

$$\textbf{Ridge Regression}: \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda||w||_2^2$$

$$\textbf{Elastic Net}: \frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)^\top w - y_i)^2 + \lambda_1||w||_1 + \lambda_2||w||_2^2$$

These values are called **hyperparameters**.

They are things you need to set before training the algorithm.

We previously discussed model selection: **Hyperparameter** are also model choices

During the learning of the function stage, we split the data into

Train / Validate / Test

- We make a list of hyperparameter settings
  - Different polynomial orders
  - Test out different λ values

- Under each possible permutation of hyperparameter setting, we perform regression
- We use validation data to see which hyperparameter setting the performed the best and pick them
- We finally use the chosen hyperparameter on the test dataset and report the final accuracy result.

Put Everything We Have Learned So Far Together.

Given data X, label Y, starting w_0, and feature map $\phi(x)$ as

$$\text{data} = \begin{bmatrix} X & Y \\ 0 & 0 \\ 1 & 1 \\ 2 & 1 \\ 3 & 2 \end{bmatrix} \quad , \quad w_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad , \quad \phi(x) = \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix} \quad , \quad \lambda = 0.2 \quad , \quad \eta = 0.01$$

1. Given the data above, show that for the derivative for the LASSO objective can be written into the compact matrix/vector form shown below.

$$\underbrace{\frac{df}{dw} \frac{1}{n} \sum_{i}^{n} (\phi(x_i)^\top w - y_i)^2 + \lambda ||w||_1 = \frac{2}{n} \Phi^\top (\Phi w - y) + \lambda sign(w)}_{\textbf{(Similar to what you did in the homework)}}$$

2. Write the Python code to run 1 gradient descent step with the Lasso objective (L1-norm) and feature map of $\phi(x)$.
3. Add a for loop and minimize the LASSO objective.
4. Plot out the Gradient Descent steps as they lower the objective.

See how Gradient Descent iteratively improves the function fit
https://www.youtube.com/watch?v=U9etyMgxKII

Finding the Matrix/Vector Form

$$f(w) = \frac{1}{n}\sum_{i}^{N}(\phi(x_i)^\top w - y_i)^2 + \lambda|w|_1 \tag{1}$$

$$\frac{df}{dw} = \frac{2}{n}\sum_{i}^{N}(\phi(x_i)^\top w - y_i)\phi(x_i) + \lambda sign(w) \tag{2}$$

$$= \frac{2}{n}\left((\phi(x_1)^\top w - y_1)\phi(x_1) + (\phi(x_2)^\top w - y_2)\phi(x_2) + (\phi(x_3)^\top w - y_3)\phi(x_3) + (\phi(x_4)^\top w - y_4)\phi(x_4)\right) + \lambda sign(w) \tag{3}$$

You can calculate $\frac{df}{dw}$ using this equation. But this is way too slow. Let's see a faster trick. Let $a_i = (\phi(x_i)^\top w - y_i)$ then

$$\frac{df}{dw} = \frac{2}{n}\left(a_1\phi(x_1) + a_2\phi(x_2) + a_3\phi(x_3) + a_4\phi(x_4)\right) + \lambda sign(w) \tag{4}$$

$$= \frac{2}{n}\begin{bmatrix}\phi(x_1) & \phi(x_2) & \phi(x_3) & \phi(x_4)\end{bmatrix}\begin{bmatrix}a_1\\a_2\\a_3\\a_4\end{bmatrix} + \lambda sign(w) \tag{5}$$

$$= \frac{2}{n}\begin{bmatrix}\phi(x_1) & \phi(x_2) & \phi(x_3) & \phi(x_4)\end{bmatrix}\begin{bmatrix}(\phi(x_1)^\top w - y_1)\\(\phi(x_2)^\top w - y_2)\\(\phi(x_3)^\top w - y_3)\\(\phi(x_4)^\top w - y_4)\end{bmatrix} + \lambda sign(w) \tag{6}$$

$$= \frac{2}{n}\begin{bmatrix}\phi(x_1) & \phi(x_2) & \phi(x_3) & \phi(x_4)\end{bmatrix}\left(\begin{bmatrix}\phi(x_1)^\top\\\phi(x_2)^\top\\\phi(x_3)^\top\\\phi(x_4)^\top\end{bmatrix}w - \begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}\right) + \lambda sign(w) \tag{7}$$

$$= \frac{2}{n}\Phi^\top\left[\Phi w - y\right] + \lambda\, sign(w) \tag{8}$$

Solution 2

With a polynomial feature map, the new dataset becomes

$$\text{data} = \begin{bmatrix} X^2 & X & 1 & Y \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 4 & 2 & 1 & 1 \\ 9 & 3 & 1 & 2 \end{bmatrix} \quad , \quad \Phi = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix} \quad , \quad w_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

The gradient for Lasso is

$$\frac{df}{dw} = \frac{2}{n}\Phi^T(\Phi w - y) + 0.2sign(w) \quad \xrightarrow[\text{then, } w_1 \text{ is}]{} \quad w_1 = w_0 - \eta\frac{df}{dw} \tag{2}$$

We have

$$\frac{df}{dw} = \frac{2}{n}\begin{bmatrix} 0 & 1 & 4 & 9 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}\left(\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix}\right) + 0.2sign\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right) \tag{3}$$

$$\frac{df}{dw} = \begin{bmatrix} -4.5 \\ -1.5 \\ 0.2 \end{bmatrix} \tag{4}$$

$$w_1 = w_0 - \eta\frac{df}{dw} \tag{5}$$

$$w_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - 0.01\begin{bmatrix} -4.5 \\ -1.5 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.045 \\ 0.015 \\ 0.998 \end{bmatrix} \tag{6}$$

```
#!/usr/bin/env python
import numpy as np
```

```
w = np.array([[0],[0],[1]])
x = np.array([0,1,2,3])
y = np.array([[0],[1],[1],[2]])
n = 4
Φ = np.array([x*x,x, np.ones(len(x))]).T
α = 0.01
λ = 0.2
```

```
print(Φ)
```

```
[[0. 0. 1.]
 [1. 1. 1.]
 [4. 2. 1.]
 [9. 3. 1.]]
```

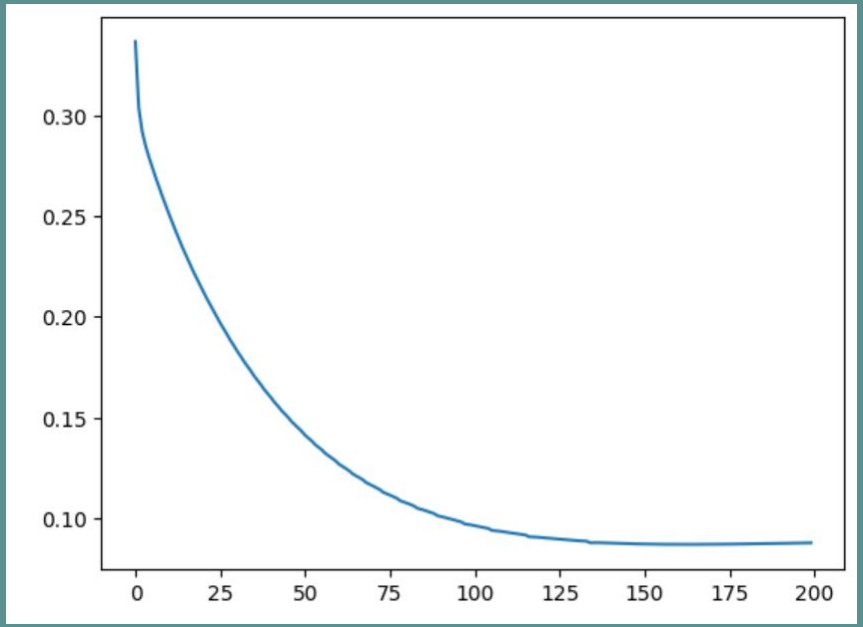
```
∇f = (2/n)*Φ.T.dot(Φ.dot(w) - y) + λ*np.sign(w)
w = w - α*∇f
print(w)
```

```
[[0.045]
 [0.015]
 [0.998]]
```

If we take more than 1 step to 400 steps

I used the exact same equation...

$$\frac{df}{dw} = \frac{2}{n}\Phi^T(\Phi w - y) + 0.2sign(w)$$




```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
```

```python
w = np.array([[0],[0],[1]])
x = np.array([0,1,2,3])
y = np.array([[0],[1],[1],[2]])
n = 4
α = 0.01
λ = 0.2
```

```python
z = np.arange(0,3,0.2)
Φ = np.array([x*x,x, np.ones(len(x))]).T
```

```python
error_list = []
for i in range(200):
    ∇f = (2/n)*Φ.T.dot(Φ.dot(w) - y) + λ*np.sign(w)
    w = w - α*∇f
```

```python
    ŷ2 = Φ.dot(w) # this is the objective error
    ε = (y - ŷ2).T.dot(y - ŷ2)/n    # average error
    error_list.append(ε.item())
```

```python
plt.plot(error_list)
plt.show()
```

Why is the prediction $\hat{y} = \Phi w$

Note that $\hat{y} = f(x)$, and remember that we are using a 2nd order polynomial that looks like

$$\hat{y} = f(x) = w_1 x^2 + w_2 x_2 + w_3 = \begin{bmatrix} x^2 & x & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{2}$$

With 4 samples, we can write them together as

$$\hat{y} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \\ x_4^2 & x_4 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \Phi w \tag{3}$$

This is why the average error is

$$\frac{1}{n}(\hat{y} - y)^\top (\hat{y} - y) = \textbf{average error}. \tag{4}$$

```python
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
```

```python
w = np.array([[0],[0],[1]])
x = np.array([0,1,2,3])
y = np.array([[0],[1],[1],[2]])
n = 4
α = 0.01
λ = 0.2
```

```python
z = np.arange(0,3,0.2)
Φ = np.array([x*x,x, np.ones(len(x))]).T
```

```python
error_list = []
for i in range(200):
    ∇f = (2/n)*Φ.T.dot(Φ.dot(w) - y) + λ*np.sign(w)
    w = w - α*∇f
```

```python
    ŷ2 = Φ.dot(w) # this is the objective error
    ε = (y - ŷ2).T.dot(y - ŷ2)/n    # average error
    error_list.append(ε.item())
```

```python
plt.plot(error_list)
plt.show()
```

Using Sklearn for these algorithms

This is a lib written by me, install by **pip install wplotlib**

```
#!/usr/bin/env python
#
import numpy as np
import wplotlib
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
#
X = np.array([[1,1],
              [2,1],
              [1,1],
              [3,1],
              [2,1],
              [6,1],
              [2,1],
              [0,1],
              [1,1],
              [1,1]])
#
w = np.array([[3],[2]])
y = X.dot(w) + np.random.randn(10,1)
```

```
linR = LinearRegression().fit(X, y)
ŷ = linR.predict(X)
```

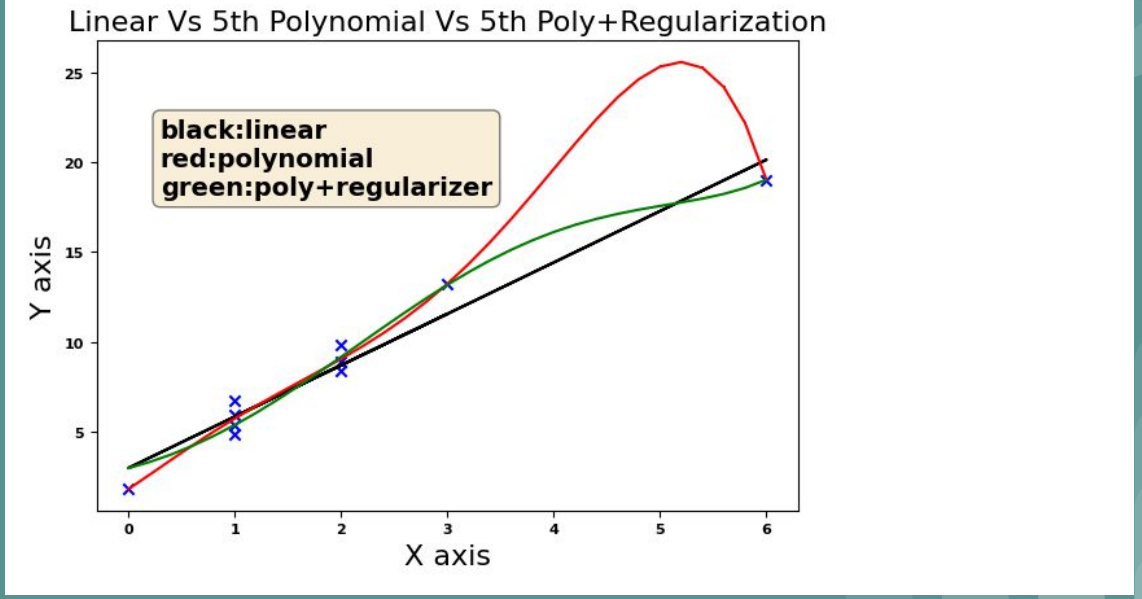
```
p = wplotlib.lines(X[:,0], ŷ, color='black', show=False)

poly = PolynomialFeatures(5)
newX = poly.fit_transform(X)

linR = LinearRegression().fit(newX, y)
X2 = np.arange(0,6.2,0.2).reshape(31,1)
X2 = np.hstack((X2, np.ones((31,1))))
X3 = poly.fit_transform(X2)
ŷ = linR.predict(X3)

q = wplotlib.lines(X2[:,0], ŷ, color='red', show=False)
```

```
linR = Ridge(3).fit(newX, y)
X2 = np.arange(0,6.2,0.2).reshape(31,1)
X2 = np.hstack((X2, np.ones((31,1))))
X3 = poly.fit_transform(X2)
ŷ = linR.predict(X3)
#
wplotlib.lines(X2[:,0], ŷ, color='green', show=False)
wplotlib.scatter(X[:,0], y, 'Linear Vs 5th Polynomial Vs 5th Poly+Regularization','X axis', 'Y axis',
                 imgText='black:linear\nred:polynomial\ngreen:poly+regularizer',yTextShift=1.20,
                 ticker_fontsize=8 , show=False)
p.show()
```



Study and copy this code and see if you can understand this code

Python code that generated the plot

```
#!/usr/bin/env python
#
import numpy as np
import wplotlib
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
#
X = np.array([[1,1],
              [2,1],
              [1,1],
              [3,1],
              [2,1],
              [6,1],
              [2,1],
              [0,1],
              [1,1],
              [1,1]])
#
w = np.array([[3],[2]])
y = X.dot(w) + np.random.randn(10,1)
```

```
linR = LinearRegression().fit(X, y)
ŷ = linR.predict(X)
```

```
p = wplotlib.lines(X[:,0], ŷ, color='black', show=False)

poly = PolynomialFeatures(5)
newX = poly.fit_transform(X)

linR = LinearRegression().fit(newX, y)
X2 = np.arange(0,6.2,0.2).reshape(31,1)
X2 = np.hstack((X2, np.ones((31,1))))
X3 = poly.fit_transform(X2)
ŷ = linR.predict(X3)

q = wplotlib.lines(X2[:,0], ŷ, color='red', show=False)
```

```
linR = Ridge(3).fit(newX, y)
X2 = np.arange(0,6.2,0.2).reshape(31,1)
X2 = np.hstack((X2, np.ones((31,1))))
X3 = poly.fit_transform(X2)
ŷ = linR.predict(X3)
#
wplotlib.lines(X2[:,0], ŷ, color='green', show=False)
wplotlib.scatter(X[:,0], y, 'Linear Vs 5th Polynomial Vs 5th Poly+Regularization','X axis', 'Y axis',
                 imgText='black:linear\nred:polynomial\ngreen:poly+regularizer',yTextShift=1.20,
                 ticker_fontsize=8 , show=False)
p.show()
```

This runs the linear model

This draws the black line

This runs the polynomial regression

This runs the ridge regression with polynomial model

Python code that performs Lasso, Ridge, Elastic Net

```
import numpy as np
from sklearn import preprocessing
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
```

```
X = np.array([[1,2,4],
              [2,1,0],
              [1,0,3],
              [3,4,3],
              [2,0,2],
              [6,1,7],
              [2,3,6],
              [0,3,4],
              [1,0,2],
              [1,1,1]])
```

```
X = preprocessing.scale(X)
```

```
w = np.array([[3],[2],[0]])
y = X.dot(w) + 0.2*np.random.randn(10,1)
```

```
lasso = Lasso()
lasso.fit(X,y)
ŷ = lasso.predict(X)
w = lasso.coef_
s = lasso.score(X,y)
print(w)
print(s)
```

```
[2.0115088  1.04370954 0.        ]
0.8514009621059027
```

```
ridge = Ridge().fit(X, y)
w = ridge.coef_
s = ridge.score(X,y)
print(w)
print(s)
```

```
[[2.57907066 1.74287548 0.31438103]]
0.9881962558156895
```

```
regr = ElasticNet(random_state=0).fit(X, y)
w = ridge.coef_
s = ridge.score(X,y)
print(w)
print(s)
```

```
[[2.57907066 1.74287548 0.31438103]]
0.9881962558156895
```

Why are Constraints so important (The world peace objective)

Let's say you tell the computer to create world peace by minimizing war

$$\min_f \quad war(f(x))$$

Gradient Descent does **not care** how you achieve the final objective.

So the computer algorithm might tell you that the best solution to minimize war is

To kill all humans.

Gradient Descent does not have the same values as us, it must have constraints

$$\min_f \quad war(f(x)) + \text{no killing humans}$$

Constraints cannot guarantee our safety, because the best solution might then be

To kill all food supply
Remove the sun
Trigger a nuclear war

Machine Learning can be dangerous if we are not careful.

The Gestational Age Prediction and their constraints

Data:

Various levels of chemical detected in blood and urine from pregnant woman

Label:

The number of weeks from pregnancy to birth

Importance:

- Babies born too early (before 37 weeks) can have major health complication and may die
- Our goal is to see if the detection of certain chemical can predict early birth

5. How do we take advantage of the prior that gestational ages should be between the range of 22 to 43?
   - With the following two constrains

$$l_1 = \sum_i^N relu(y_i - 43)$$

$$l_2 = \sum_i^N relu(22 - y_i)$$

6. How do we minimize false positive error (type 1 or α)?
   - With the following constrain

$$l_3 = \sum_i^N relu(y_i - 37) relu(37 - \hat{y}_i)$$

7. How do we minimize false negative error (type 2 or β)?
   - With the following constrain

$$l_4 = \sum_i^N relu(37 - y_i) relu(\hat{y}_i - 37)$$

The Horvath Clock

As a hobby, I study the biology of why people get older. In 2013, the Horvath clock was discovered.

Data:

The level of various genes turning on or off

Label:

The age of the individual

Read the article here !

Importance:

- We all know some people look younger or older than their age.
- If how genes turn on and off can predict a person's age well, it tells us that some people may be old but their DNA is turning on or off like a young person.

Scientists have already discovered some ways to slow down your aging speed.

a. Smoking and Alcohol damages cellularly and promote DNA mutation.
b. Avoid highly processed food (especial high in sugar)
c. Avoid food contaminated by heavy metal (dark chocolate, protein powder)
d. Use sunscreen every day.
e. Regular exercise, and don't just sit there all day
f. Eat less and early.
g. Make friends and not enemies, mend your broken relationships
h. Learn to Manage stress
i. Sleep at the same time everyday and early.

Which regression method did they use?

Elastic Net

$$\min_{w} \quad \frac{1}{2}\sum_{i}^{N}(w^{\top}\phi(x_i) - y_i)^2 + \lambda_1|w|_1 + \lambda_2|w|_2^2$$

In-Class work

1. Load the stock_prediction_data.csv and stock_price.csv
   a. stock_prediction_data.csv contains the data of stock prices and world events
   b. stock_price.csv contains the difference in stock price the next day.
2. Split the data into Train, validation, test
3. Preprocess the data, remove center and scale it.

4. Perform 2nd order polynomial regression with
   a. Lasso constraint
      i. Solve it with gradient descent
      ii. Solve it with sklearn library (compare your results)
      iii. Use the validation data to identify and ideal lambda value

   b. Ridge constraint
      i. Solve it with gradient descent
      ii. Solve it with sklearn library (compare your results)
      iii. Use the validation data to identify and ideal lambda value

The TA was nice enough to write the code for you. Study this code if you can't do it yourself. Link

End of the day...

Remember to submit your In-class work

Drawn by AI @ https://lexica.art/