If model working good on Training dataset but working worst on Test data it is condition of overfitting.

Prevention:

1. Cross Validation: Powerful preventive measure.
2. Train with more data: won't work every time.
3. Remove features
4. Early stopping.
5. Regularization
   - Lasso (feature + error take care)
   - Ridge (error reduce)
6. Ensembling → Random Forest
   - reduce (n-estimators) (no of trees)

Types of Cross Validation

Split → 70% | 30% with random_state = 0
↳ 85% accuracy
random_state = 50
↳ 87%

① k-Fold CV

K = 5

1000/5 = 200

Exp1: test 200 | 800 train
Exp2: 200 | test 200 | 600 train
Exp3: 200 | 200 | 200
Exp4: 200 | 200 | 200 | 200
Exp5: 200 | 200 | 200 | 200 | 200

Disadvantages
Don't work / give good accuracy on imbalanced dataset.

② Stratified Cross Validation

(No. of instances of each class in train & test data is taken in equal proportion as present in data.)

K = 5 1000/5 = 200.

200 | 800
200 | equal proportion of each class in train/test.
↳ Train.

1000 records
600 | 400.
A | B

A (Test)	B (Test)
200 × 6/10 = 120	80

A (Train)	B (Train)
800 × 6/10 = 480	800 × 4/10 = 320.

Spearman's Correlation

It is Pearson Correlation Coefficient b/w rank variables

$$\rho_{rg_x, rg_y} = \frac{cov(rg_x, rg_y)}{\sigma_{rg_x}\,\sigma_{rg_y}}$$

When to use:
When data is non-linear distributed. + linear distribution

x_i	y_i	Rank x_i	Rank y_i	$d_i^2 = (x_i - y_i)^2$
86	0	1	1	0
97	20	3	3	0
85	15	2	2	0

n = 3 d_i =

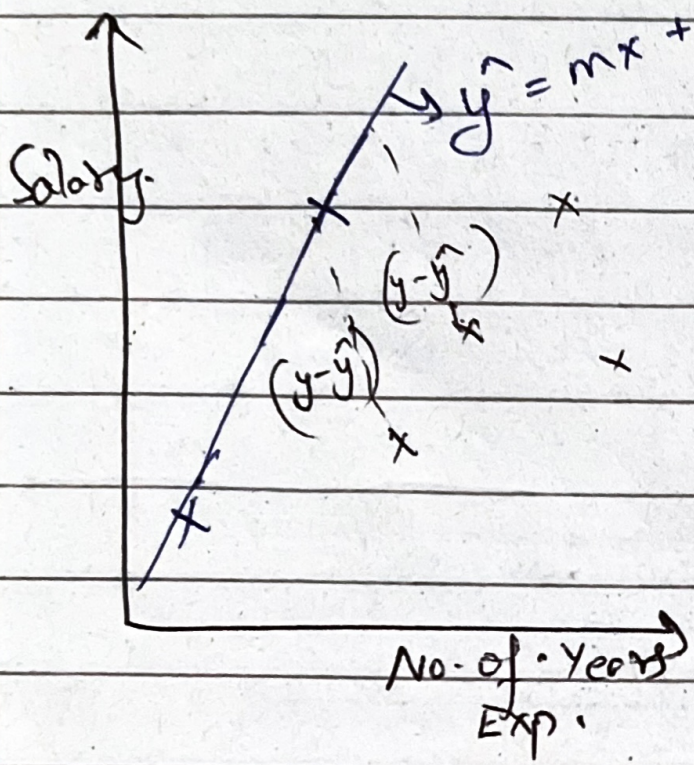
Spearman coeff.

$$S = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

S = 1 → highly correlated

Acc. mean()

Ridge & Lasso Regression



Cost fn.

Sum of Residuals.

$$\sum_{i=1}^{n} (y-\hat{y})^2$$

$$\hat{y} = mx + c$$

For Above (For training dataset)

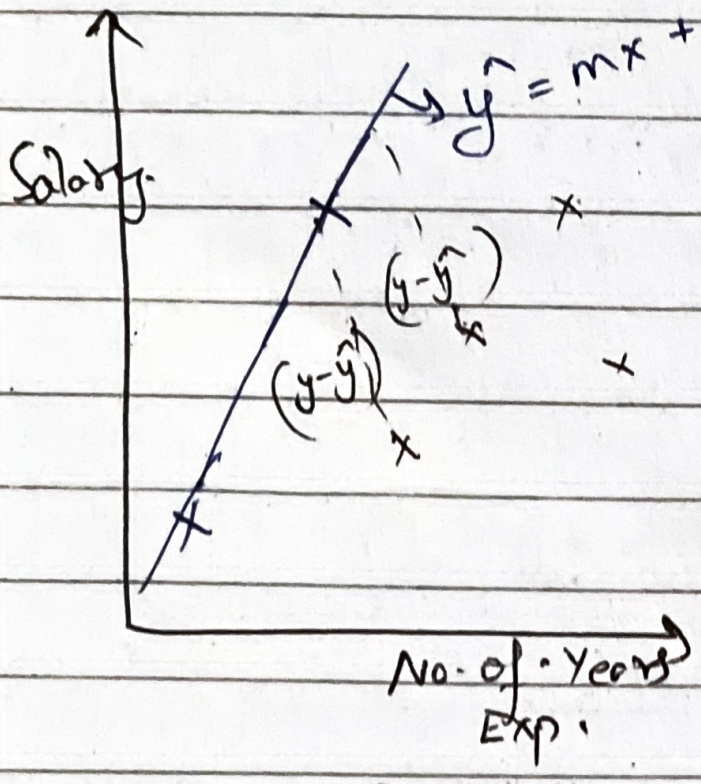
$$\sum_{i=1}^{n} (y-\hat{y})^2 = 0$$

(For training data we are getting low bias or less error result)

(For test dataset) ×

($y-\hat{y}$ is very big) leads to the Condition of Overfitting.

Ridge & Lasso Regression.



Cost fn.

Sum of Residuals.

$$\sum_{i=1}^{n} (y-\hat{y})^2$$

$$\hat{y} = mx + c$$

For Above (For training dataset)

$$\sum_{i=1}^{n} (y-\hat{y})^2 = 0$$

(For training data we are getting low bias or less error result)

(For test dataset) ×

$(y-\hat{y}$ is very big) leads to the Condition of Overfitting.

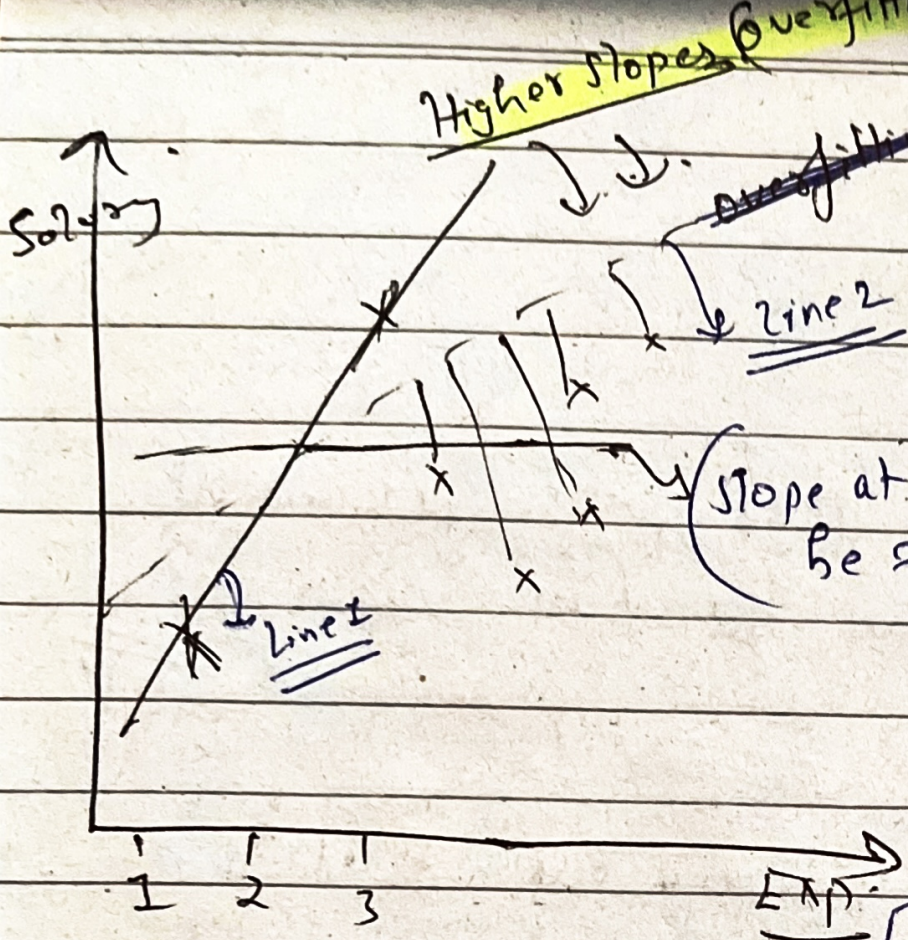
	For train data	For test data
Overfitting	~~Low~~ Error (low)	high Error
Underfitting	high Error	high Error

Note

Overfitting → High Variance

↓ Ridge & Lasso

Low Variance



$\lambda \rightarrow$ lambd.

Not lambda. (because this is reserved in Python)

Note

we can use $\lambda(b^2)$ but

$\times$

$w \rightarrow$ slope is a larger parameter as almost all the param. in w which is more significant - omit b

Cost function (for Ridge)

L2

$$\sum_{i=1}^{n} (y - \hat{y})^2 + \lambda (\text{slope})^2$$

λ regularization parameter

Where λ (lambda) range of 0 to + ve. (lambd)

$\downarrow\downarrow$ minimize.

ex) for above graph (line 1)

$\hat{y}$

$\lambda = 1 \times (1.3)^2$

Cost function = 1.69

for line 2

Cost function:

Small value +

1×1

≈ 1.39

$\downarrow$ Reduced

Lasso Regression (helps to reduce overfitting + helps to do feature selection)

↓ L1

Cost function — w will be sparse.

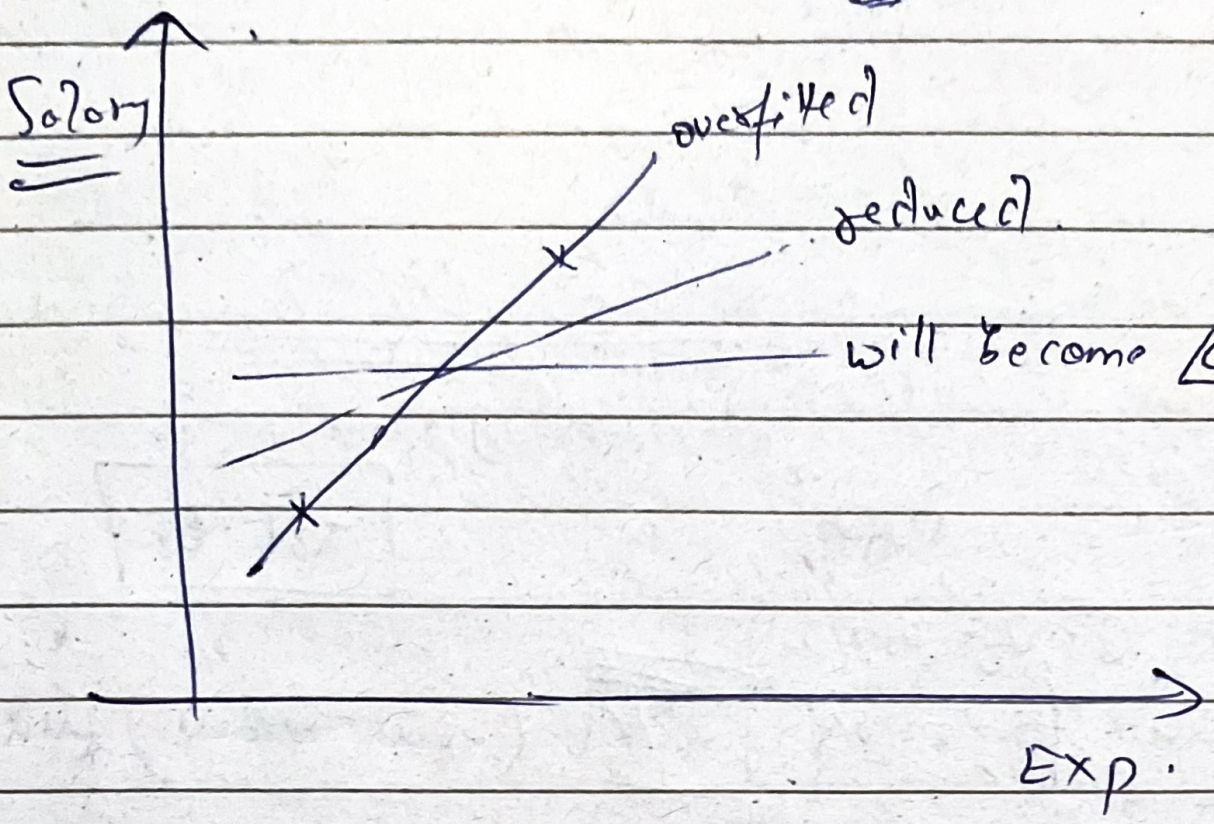
$$\sum_{i=0}^{n} (y - \hat{y})^2 + \lambda |\text{slope}|$$

$$y = m_1x_1 + m_2x_2 + m_3x_3 + C$$

$$\lambda |m_1 + m_2 + m_3|$$

Note:

Whereever any of $m_1, m_2, m_3 \ldots m_n$ parameters are small we remove them in lasso.



So, if m_3 is small (very small)

will remove from calculation.

$$\int x \left| m_2 x_1 + m_3 x_2 \right|$$

will retain.

Ridge & Lasso Regression Implementation

Using (load_boston dataset)

X, Y → dependent feature

↓

Independent features

Linear Regression

```
mse = cross_val_score(lr_regression, X, y,
      scoring = 'neg_mean_squared_error',
      cv = 5)
```

```
mean_mse = np.mean(mse)
```

```
print(mean_mse)
```

Note: Value of mean_mse. close to 0 is always better.

↓

-37.131

↓ for Linear Regression

---

cross_val_score ?

Diff. b/w. cv & KFold cv

Scoring in cross_val_score ?

Ridge Regression.

```
from sklearn.linear_model import Ridge.
from sklearn.model_selection import GridSearch(
                                   GridSearchCV
```

~~dig~~:

```
ridge = Ridge()
parameters = {'alpha': [1e-15, 1e-10 --- 1, 5, --- 100
```

→ dict (having +ve values only)

```
ridge_regression = GridSearchCV( ridge, parameters,
                   scoring = 'neg_mean_squared_error',
                   cv=5)

ridge_regression.fit (X, y)

print ( ridge_regression.best_params_ )
print ( ridge_regression.best_score_ )
    {'alpha':100}
```

-29.871 → (for Ridge Regression)

Lasso

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

lasso = Lasso()
```

```
print(lasso_regression.best_params_)
print(lasso_regression.best_score_)
```

{'alpha': 1}

-35.49 (mse)

① which is better than Linear Regression

② As we know Lasso regression drops some of the insignificant features & take care of minimising

Conclusion:

As we have scores from Linear → -37.13

Ridge → -29.87

Lasso → -35.99

① ⇒ Ridge is giving / performing better
better results

② But for large features Lasso is better to avoid complexity as well as redundancy.