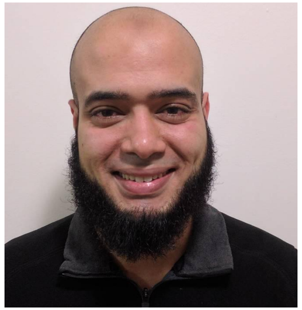
Machine Learning

Regularization - Ridge

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Regularization

- Regularizations are techniques to prevent ML models from overfitting
- Weights Regularization
  - Our modeling **bias** is: smaller weights are better!
  - Change the optimization function to **penalize** the large weights and encourage **weights shrinkage**
  - 2 popular choices: Ridge and Lasso
- Deep Learning Regularization
  - Dropout (cancels the role of some weights randomly during from one iteration to another)
    - The weights fail to collaborate on **memorizing**
  - Batchnorm (has some regularization effect)

Question!

- Below is our **irregularized** version of the cost function
- Add a single term that encourages the weights to be smaller

$$cost(W) = \frac{1}{2N} \sum_{n=1}^{N} (y(X^n, W) - t^n)^2$$

Ridge Regression

- Simply **add a penalty** of $\mathbf{w^2}$ for every weight w
- But, how much should we shrink the weights?
- It depends on the dataset! In some datasets, a small weight is good, however, a very tiny weight might be better for other datasets
- So, we need a hyperparameter to control this
  - Let’s call it lambda: It controls the **strength of the penalty**
- Simply for every weight w, add a penalty of λw^2
  - Logically, λ is a non-negative term [0 - infinity]

Ridge Regression

$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2 \boxed{+\sum_{i=1}^{M}\frac{\lambda}{2}W_i^2}$$

- Recall that we have N examples and M weights. We divide by 2 for easy derivatives
- Given one of the weights Wj, what is the partial derivative of W relative to this variable?!

$$\frac{\partial cost(W)}{\partial W_j} = \frac{1}{N}\sum_{n=1}^{N}(y(X^n, W) - t^n) * X_j^n + \lambda W_j$$

Ridge Regression (Aka L2 regularization)

- We also call it L2 regularization (squared L2 norm)

$$L2 - norm : \|W\|_2 = \sqrt{W_1^2 + ... + W_M^2}$$

$$SquaredL2 - norm : \|W\|_2^2 = W_1^2 + ... + W_M^2$$

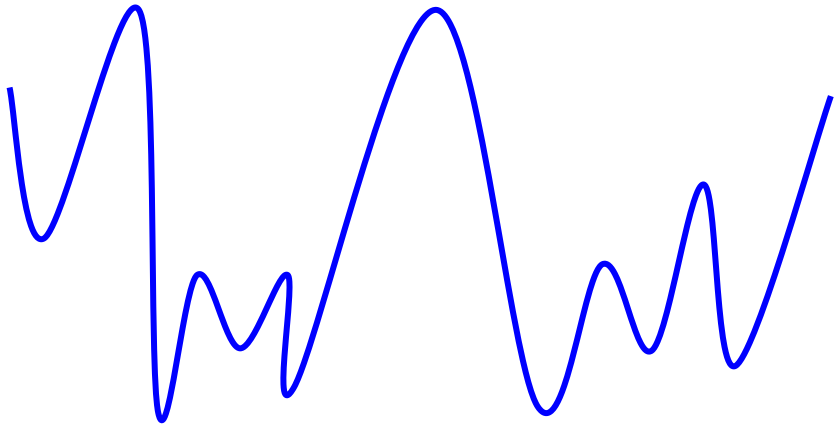
$$\underset{W}{\text{minimize}} \frac{1}{2N}\|XW - t\|_2^2 + \frac{\lambda}{2}\|W\|_2^2$$

$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2 + \sum_{i=1}^{M}\frac{\lambda}{2}W_i^2$$

Intuition

W1	W2	$\|gt - predici\|^2$	$\|w1\| + \|w2\|$	$\|gt - predici\|^2 + \|w1\| + \|w2\|$
1	3	15	4	19
4	2	10	6	16
7	5	300	12	312
150	-25	25	175	200
10	210	20	220	240
100	200	200	300	500
250	300	15	550	565
1000	350	10	1350	1360
2000	3000	0	5000	5000

$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2$$



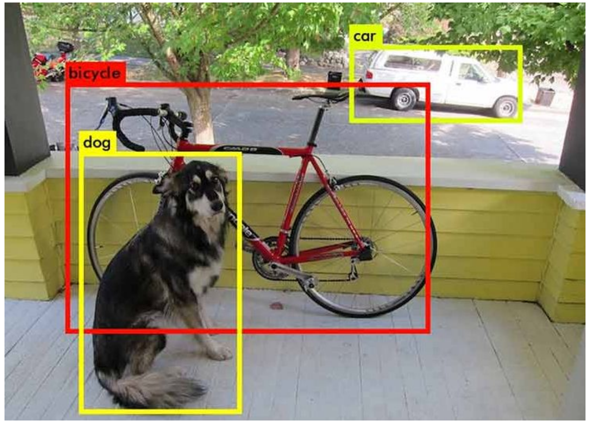
$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2 + \sum_{i=1}^{M}\frac{\lambda}{2}W_i^2$$

Multi-loss components: Detection YOLO example

Regression loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$



Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Should we penalize the Y-intercept?

- Observe in the L2 regularization, we start **from i = 1**
  - This means we do NOT penalize the intercept at W0
- **Intuitively**, we give complete **flexibility** for the line’s intercept
- Recall: from the closed-form solution: c = average(y) - slope * average(x)
- Assume that we **standardize** all the data
  - Then average(x) = 0 and c = constant = average(y)
  - As a result, c doesn’t depend on the slope or xs!
  - Then the intercept does not play a role in the overfitting. It is just relevant to the data
  - Therefore, we don't need to penalize the intercept, and its original optimal formula should be used
- Tip: If you have prior belief that the intercept should be zero, then penalize it.

Fine-Tuning hyperparameter λ

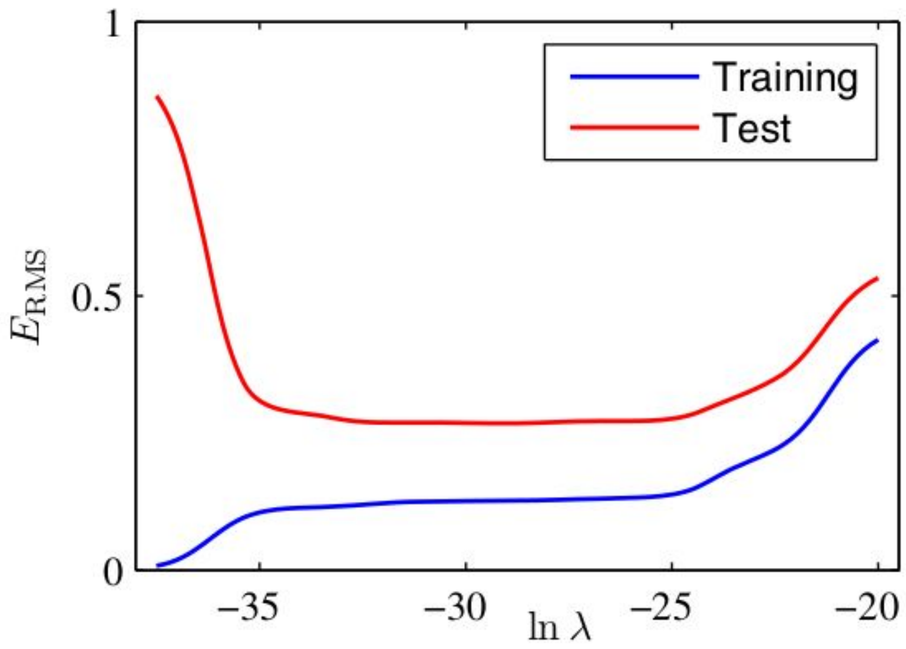
$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2 + \sum_{i=1}^{M}\frac{\lambda}{2}W_i^2$$

- What λ = 0 does imply?
- No regularization!
- What does λ = ∞ (e.g. 10000000000) imply ?
- Even with a very small weight like 0.001, the penalty is 10000000 for a single weight. The optimizer will decide the optimal W = 0 to avoid such high penalty
- The optimal λ is somewhere in the middle. Use **Cross-validation** and experiment with several values to find the optimal one
  - Consider the values: [100, 10, 1, 0.5, 0.1, 0.01, 0.001, 0.0001]

Hyperparameter λ and bias-variance tradeoff

- A higher λ value
  - increases the amount of regularization
  - more shrinkage of the coefficients
  - **lower variance but potentially higher bias**
  - Use when the training data is **limited**, noisy, or when there is a high risk of **overfitting**
- A lower λ value
  - reduces the amount of regularization
    - allowing the model to fit the training data more closely
  - **lower bias but higher variance**
  - when the dataset is larger, cleaner, or when the underlying relationships are complex and require a more flexible model
- Practically; use cross-validation

Figure 1.8 Graph of the root-mean-square error (1.3) versus $\ln \lambda$ for the $M = 9$ polynomial.



- Assume with degree M=9, our model without regularization has train error = 0
- Tip: Instead of visualizing large values We can use the log function to scale to smaller values
- ERMS = sqrt(MSE)

Source: Pattern recognition - Bishop book

Question!

- Assuming we have N (x, y) points and we trained a ridge regression model (mx + c) using a value of λ = ∞, what are the potential lines that could be found?

- The ridge penalizes the ‘m’ NOT the ‘c’
- As the penalty is so high, the ‘m’ = 0: a **horizontal** line
- What is the best horizontal line for N points?
- c = average(ys)

Question!

$$cost(W) = \frac{1}{2N}\sum_{n=1}^{N}(y(X^n, W) - t^n)^2 + \sum_{i=1}^{M}\frac{\lambda}{2}W_i^2$$

- We can derive the normal equations without regularization
- Do you think we can derive closed-form for the optimal regularized weights?

- Yes, it is still a quadratic equation. Just put derivative = 0 and derive it
  - Eventually, just a new term λI is added, aka the ridge
- In ‘Machine Learning A Probabilistic Perspective’ book, ch 7.5, you can find math to make the computations **numerically stable** in $O(DN^2)$
  - An advantage over the linear regression solution (X^TX might be (nearly) singular)

$$(X^TX + \lambda I)^{-1}X^Ty.$$

Question!

- True or False:
- Recall that: MST ridge has extra error term!
- The train MSE error of linear regression <= train MSE of ridge regression?

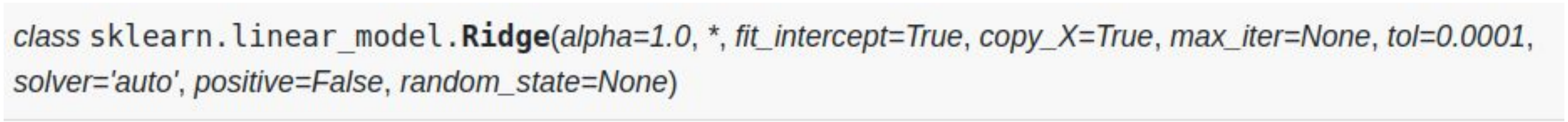
- False: The performance of linear regression and ridge regression **depends on various factors**, including the specific dataset, the amount of noise, and the relationship between variables
  - *General Tip: this is valid answer for many train/test error comparisons*
  - *General Tip: errors can increase and decrease in most of curves (not **strictly** inc/dec)*

Brute Forcing the hyperparameters

- So far, we've encountered three hyperparameters: the learning rate, the regularization penalty (λ), and the polynomial degree
  - Additionally, we must consider whether or not the line has an intercept
- Assume you want to try all combinations of the following:
  - Learning rates: {0.1, 0.01, 0.001}
  - Lambda: {1, 0.5, 0.2, 0.002}
- How many combinations do we have?
- 3 x 4 = 12
- We can simply write 2 nested loops to try all of them
  - For each combination, do k-fold cross validation
- However, some algorithms have several hyperparameters
  - Then writing the code might be annoying
  - Our search space could become extremely large!

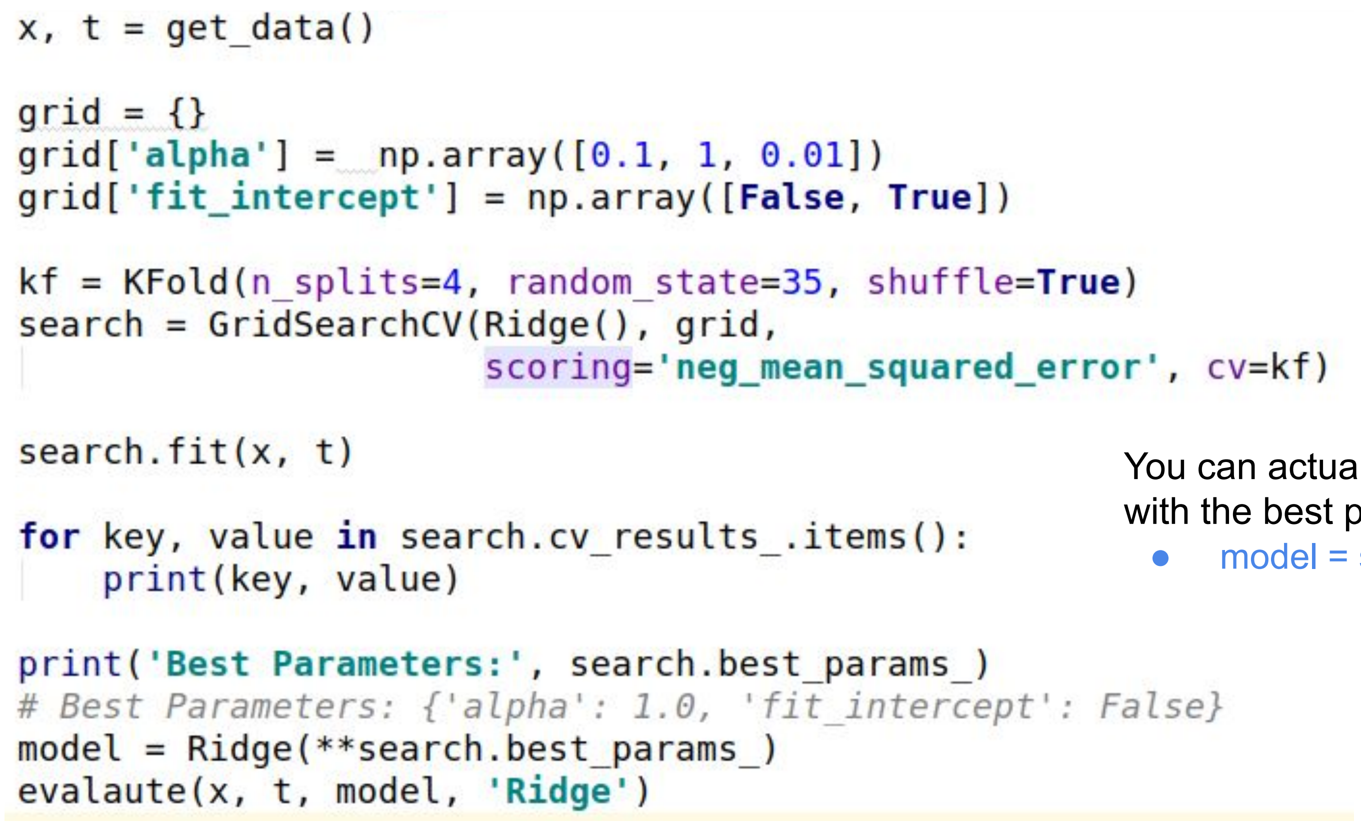
Grid Search

- **Grid search** is a tuning technique that **searches exhaustively** through the specified combinations of **hyperparameters**
- Let's imagine we are considering a Ridge Regressor model and searching through all its provided parameters
  - Practically speaking: alpha and fit_intercept
  - SKlean uses Conjugate **gradient** method to compute Ridge, that is why there are more parameters (e.g. max_iter and tolerance (precision))
    - Understanding these advanced parameters requires more mathematical knowledge



```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001,
solver='auto', positive=False, random_state=None)
```

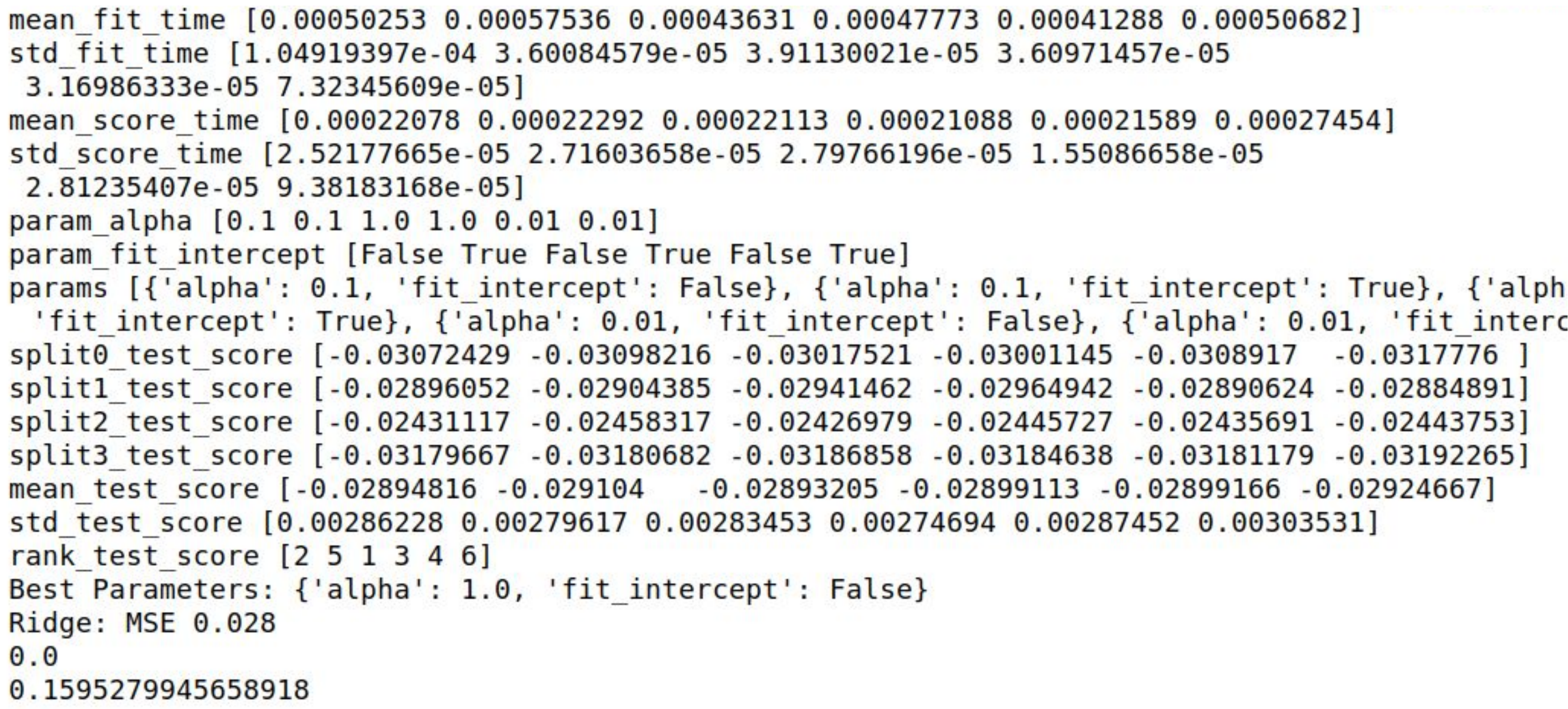
Grid Search: Ridge



```
x, t = get_data()

grid = {}
grid['alpha'] =  np.array([0.1, 1, 0.01])
grid['fit_intercept'] = np.array([False, True])

kf = KFold(n_splits=4, random_state=35, shuffle=True)
search = GridSearchCV(Ridge(), grid,
                      scoring='neg_mean_squared_error', cv=kf)

search.fit(x, t)

for key, value in search.cv_results_.items():
    print(key, value)

print('Best Parameters:', search.best_params_)
# Best Parameters: {'alpha': 1.0, 'fit_intercept': False}
model = Ridge(**search.best_params_)
evalaute(x, t, model, 'Ridge')
```

You can actually use the trained model with the best parameters

- model = search.best_estimator_



```
mean_fit_time [0.00050253 0.00057536 0.00043631 0.00047773 0.00041288 0.00050682]
std_fit_time [1.04919397e-04 3.60084579e-05 3.91130021e-05 3.60971457e-05
 3.16986333e-05 7.32345609e-05]
mean_score_time [0.00022078 0.00022292 0.00022113 0.00021088 0.00021589 0.00027454]
std_score_time [2.52177665e-05 2.71603658e-05 2.79766196e-05 1.55086658e-05
 2.81235407e-05 9.38183168e-05]
param_alpha [0.1 0.1 1.0 1.0 0.01 0.01]
param_fit_intercept [False True False True False True]
params [{'alpha': 0.1, 'fit_intercept': False}, {'alpha': 0.1, 'fit_intercept': True}, {'alph
  'fit_intercept': True}, {'alpha': 0.01, 'fit_intercept': False}, {'alpha': 0.01, 'fit_interc
split0_test_score [-0.03072429 -0.03098216 -0.03017521 -0.03001145 -0.0308917  -0.0317776 ]
split1_test_score [-0.02896052 -0.02904385 -0.02941462 -0.02964942 -0.02890624 -0.02884891]
split2_test_score [-0.02431117 -0.02458317 -0.02426979 -0.02445727 -0.02435691 -0.02443753]
split3_test_score [-0.03179667 -0.03180682 -0.03186858 -0.03184638 -0.03181179 -0.03192265]
mean_test_score [-0.02894816 -0.029104   -0.02893205 -0.02899113 -0.02899166 -0.02924667]
std_test_score [0.00286228 0.00279617 0.00283453 0.00274694 0.00287452 0.00303531]
rank_test_score [2 5 1 3 4 6]
Best Parameters: {'alpha': 1.0, 'fit_intercept': False}
Ridge: MSE 0.028
0.0
0.1595279945658918
```

Grid Search with pipeline

- Assume we have a pipeline, for example:
  - MinMaxScaler followed by Ridge Regressor
- Assume we want to try different values of alphas
- If we use make_pipeline, the code will fail because alpha is passed to MinMaxScaler which doesn't expect it
- Instead, we use the Pipeline object as follows:
  - pipeline = Pipeline(steps=[("scaler", preprocessor), ('somemodel', Ridge())])
  - In other words: we have list of items, each item has a name to match object with parameter
- In order to specify that the alpha parameter only applies to 'somemodel':
  - Use ourmodel__ as a prefix (and so on)
  - grid = {'somemodel__alpha': alphas}

Handling Large Search Space

- The major challenge here is we might want to try several hyperparameters and for each hyperparameter a wide range
  - This might be **hundreds/thousands** of trials
- This is not an easy concern and requires smart thinking / art / experience
  - Try to start with key variables. For each variable test a few potential values for each
    - E.g. For lambda: {10, 5, 1, 0.1, 0.01, 0.001, 0.0001} or even a fewer
  - Say you found lambda = 0.01 is a good one. Now, you might try values around it
  - You might fix all parameters except one only and explore it. But be careful the chosen value can be bad when we try changing the other variables
  - Consider early filtering using RandomizedSearchCV where random values (maybe according to a give distribution) are tried first. Use the insights to guide on next trials
  - If there are common numbers in the literature for your problems, try them first
    - E.g. learning rates: 0.01 and 0.001

Model Selection and Hyperparameters

- In order to select the best model, we first wanted to try **different models** (e.g. SVM, Linear Regression) and use **cross-validation** to compute the average performance of each model.
    - Then comparing the models can help us select the best one!
- We learned that we can use grid search and cross-validation to select the best hyperparameters for each model
- So overall:
    - **For each model**: select its best hyperparameters (grid search + CV)
    - **Compare** the models and **select** the best one
- There is a potential for selection bias
- Can you find it?

Model Selection and Hyperparameters

- Process:
  - **For each model**: select its best hyperparameters (grid search + CV)
  - **Compare** the models and **select** the best one
- There is a potential for selection bias when we use the **same dataset** for both model selection and hyperparameter selection
  - The issue is that the hyperparameters are optimized to maximize performance on the validation set, which can lead to overfitting to the **specific** characteristics of that data.
  - Consequently, the selected hyperparameters may not generalize well to unseen data, resulting in poor performance when the model is applied in real-world scenarios
- Solutions:
  - In theory: use one dataset to determine the hyperparameters, and another to select the model
  - One way to achieve this is through **nested cross-validation**

About Dataset Size

- When working with small datasets, it can be difficult to achieve satisfactory performance
- In many cases small datasets don’t provide satisfactory performance for the customers, unless you have domain experience or fine-tuning in deep learning
- With medium to large datasets, it is often possible to use a train/test split without the need for extensive k-fold cross-validation.
- We can easily have multiple sub-datasets for different purposes (e.g. model selection, hyperparameters)
- In industry, many of the success stories rely on large datasets
- Collecting large datasets is expensive, hard or impossible for many problems

Handling Overfitting

- Remember, when we examined the weights, we identified that there was overfitting
- How can we prevent this?
  - Reduce the complexity of your model
  - Use regularization strategically to avoid underfitting or overfitting
  - Increase the amount of training data
    - Utilize data-augmentation techniques or add noise to the data
  - **Proper Early stop**
    - Don’t keep training if getting very slight improvements

Question!

- Your trained model has 100% training accuracy and 99% test accuracy
- What are your thoughts?

- If this is an easy dataset, this may happen
- However, in real datasets a high training accuracy could be a sign of overfitting
- A high test accuracy maybe due to:
  - Simple test set or limited diversity in the test data
  - **Data leakage** from test to train
  - A mistake in the machine learning experimentation process
- Tip: It is important to investigate and understand the reasons for high performance before reporting it

Related Materials

- Ridge: StatQuest, StatQuest
- Nested-cross validation:
  - This Paper (published in 2010) discusses the selection bias we mentioned empirically
  - N-CV Video/Article -Article -Article -Article -Article

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."