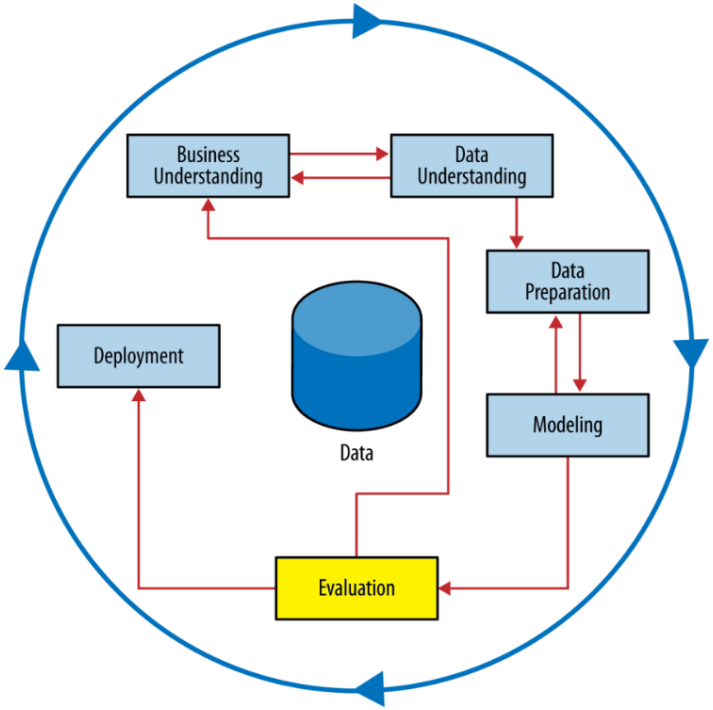
Overfitting and Its Avoidance

PF5

Learning Goals

- Generalization and overfitting
  - Overfitting and model complexity
  - Detect overfitting: fitting graphs
- Model evaluation
  - Hold-out testing
  - Cross-validation
- Overfitting avoidance
  - Nested hold-out testing
  - Grid Search with cross-validation
  - Regularization



Generalization and Overfitting

- When a model gets more **complex**, it often **fits better** to **training data**.
  - A model gets more complicated when more features are used.
    - For trees, model complexity also means the tree size (or number of nodes).
  - There is a tendency to tailor models to training data.
    - "If you torture the data long enough, it will confess".

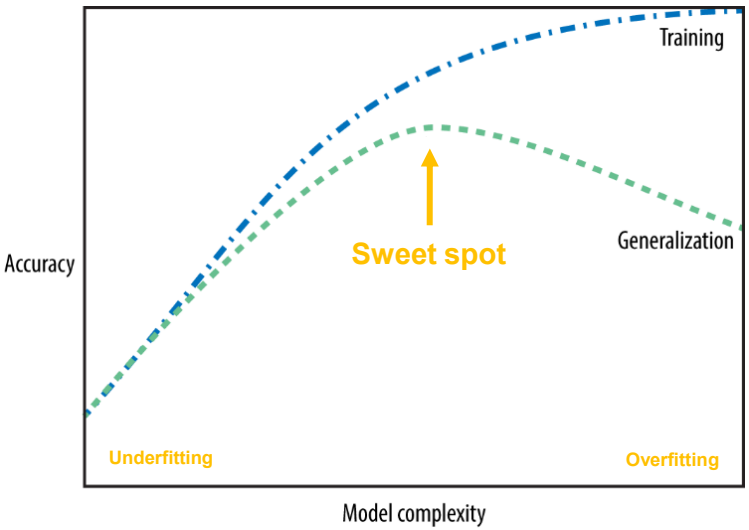
- **Generalization** is the goal of data mining: can this model perform well on the future **unseen data**?
  - If not, you are facing an **overfitting** problem: i.e., finding patterns that well captures training but not the unseen data.
    - Historical performance is no guarantee of future success.

- **Tradeoff** between model fit and simplicity (generalization).
  - Increasing model complexity often means a better fit to training data, but its generalization performance not always increase.
  - How to find the balance?

More Complexity, Better Generalization?

- **Holdout testing**: hold out some data, for which the target values are known, for model evaluation only.

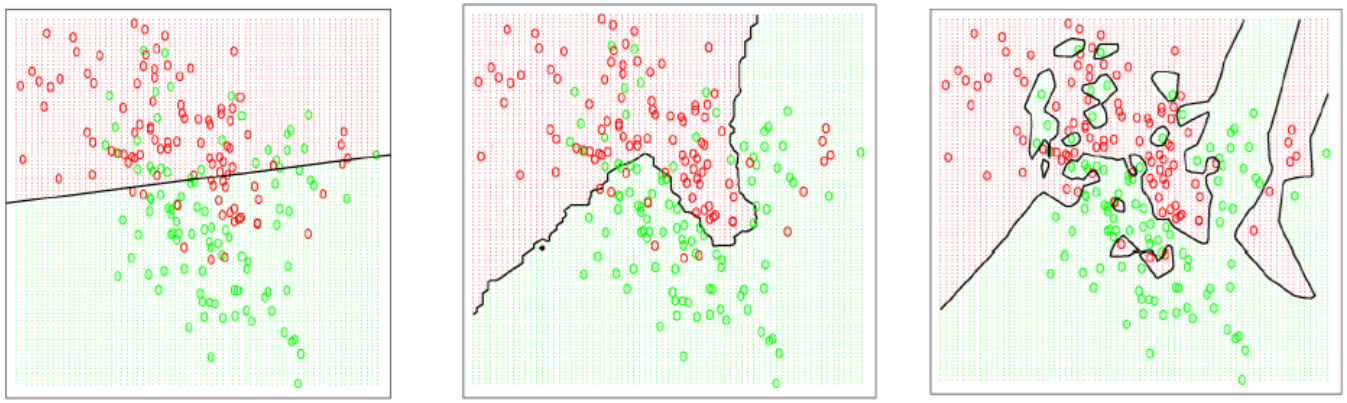
- **Fitting graph**: plot **model performance**, on training and test data, against the **model complexity**.
  - Classification model performance is usually measured by prediction accuracy.



- When model complexity increases:
  - Training performance usually increases.
  - Generalization performance increases first, then drop(**overfitting starts**) at certain point.
  - **Sweet spot** is the highest point on the generalization performance.

Why Overfitting Occurs

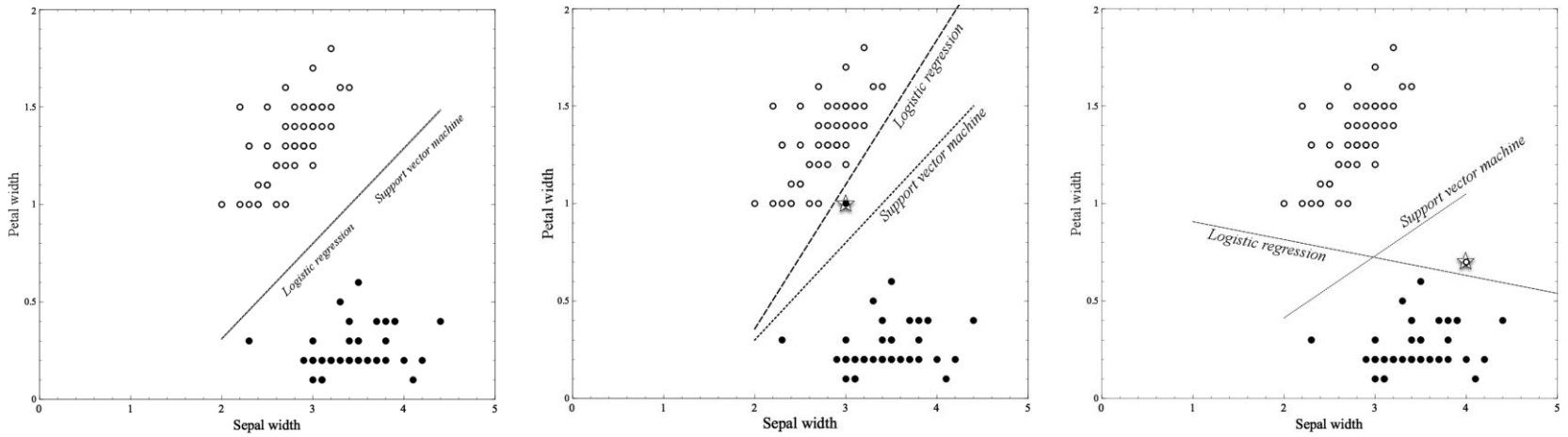
- As a model gets more complex, it may pick up harmful **spurious correlations** between features and target in training data.
  - Each training set is a finite sample of the population, it has variations even without sampling bias.
    - A training set is different from the population (or the other training set).
  - The correlation between features and target in the training set may not exist in the population or future data.



Under-fitting Good Over-fitting

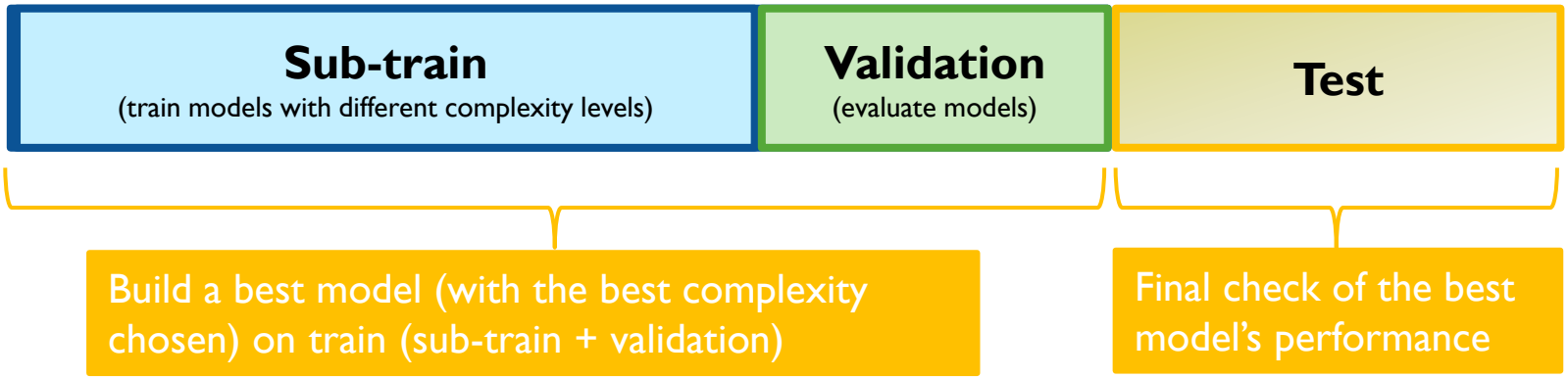
Example: Overfitting in Linear Classifiers

- Some models are by nature more likely to overfit than others.
  - **Logistic regression** is very sensitive to new instances, which can be an outlier that doesn't exist in future data.
    - Therefore, it tends to overfit.
  - **Support vector machine** opts for larger margin around the decision boundary, rather than perfect separation of training instances.
    - Therefore, it is more robust and can avoid overfitting.



Overfitting Avoidance & Complexity Control

- **Nested Holdout Testing**
  - Split the data into train & test fold (usually shuffle data first).
  - Further split the train fold into two parts: sub-train & validation.
  - Train models (with different complexity levels) on sub-train, compare their generalization performance on validation.
    - The best complexity level (model) yields highest generalization performance.
  - With the best complexity chosen, build a model on the entire train fold, then evaluate its generalization performance on the test fold.



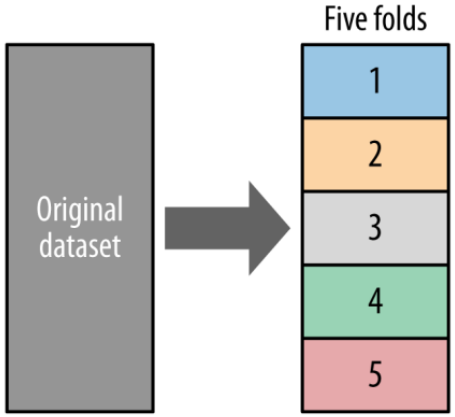
From Holdout Evaluation to Cross Validation

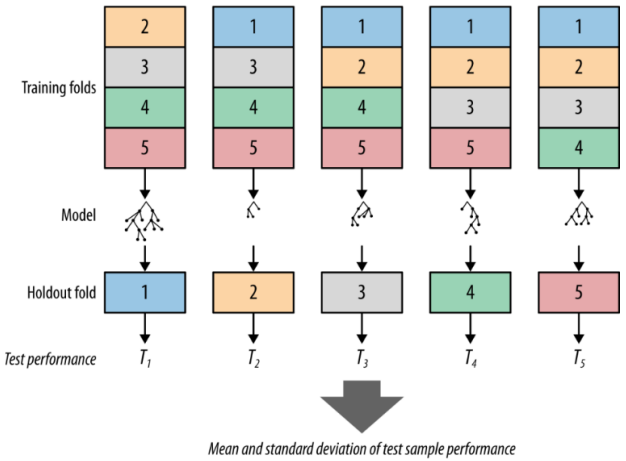
- **Holdout testing** is evaluation in laboratory setting.
  - Holdout data gives an estimate of the generalization performance; however, it is just **one single estimate**.
  - Can it be just a single particularly lucky (or unlucky) choice of training and test data?

- **Cross-validation** is a procedure that systematically swap out data samples for **evaluation**.
  - Use limited data to assess confidence in model performance estimate (e.g., mean, standard deviation).
  - Then we have a more reliable estimate of model performance.

(k-fold) Cross-Validation

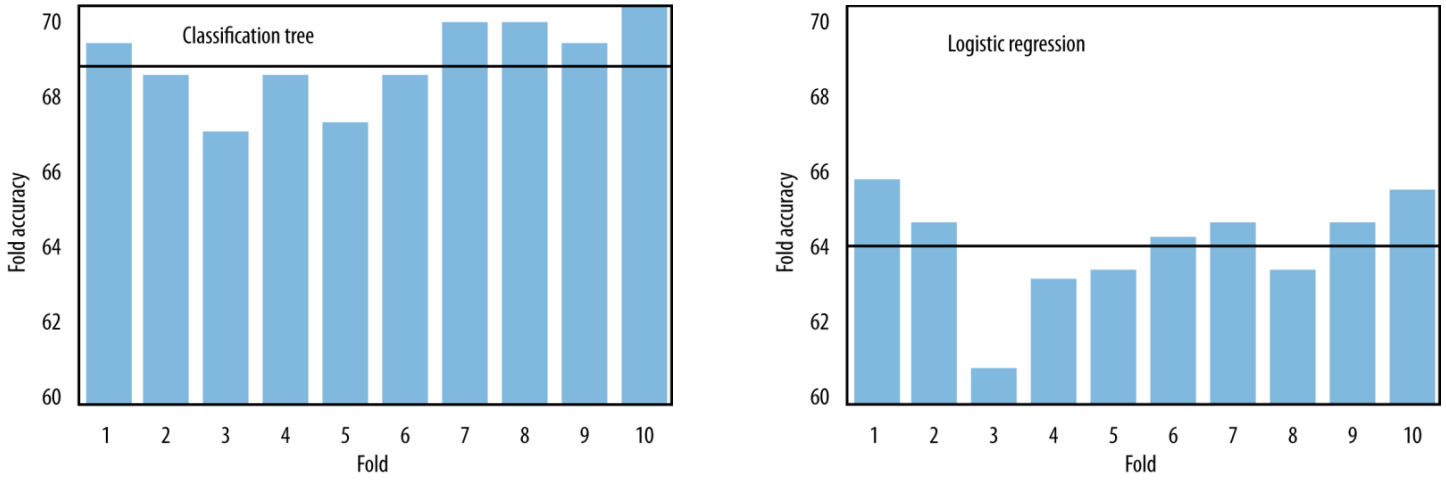
- Split the data into k folds (usually shuffle data first).
- Iterate over data k times, in each iteration (split):
  - A different fold is held out as the test fold;
  - The remaining k -1 folds are combined as the training folds.
- Compute mean, standard deviation of test performances for all k folds.
  - The mean cv score tells the average generalization performance of a model.
  - The standard deviation tells how stable the generalization performance are.





Example: Churn Data

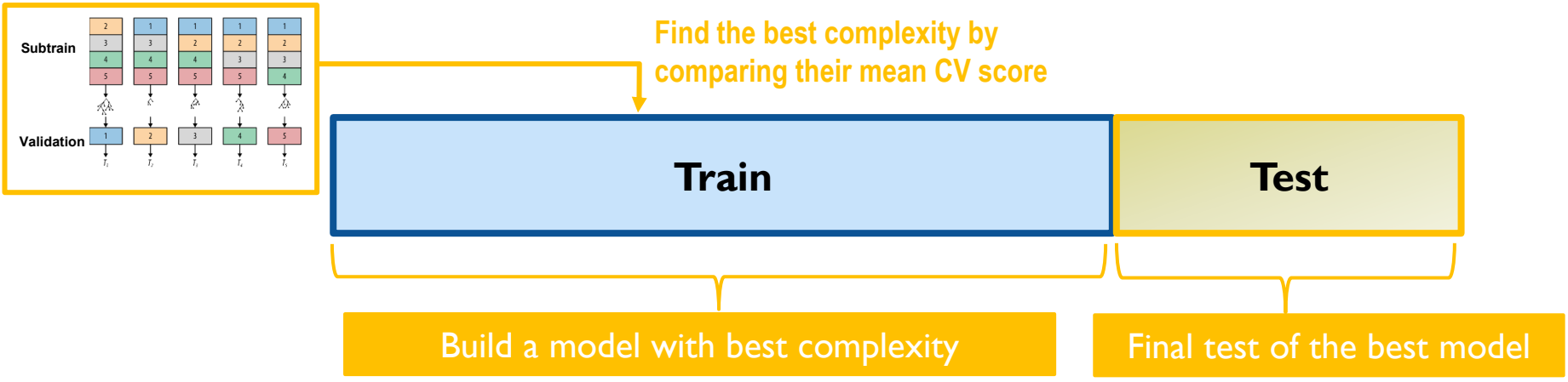
- 10-fold **Cross-validation**:
  - **Decision tree**: mean generalization accuracy is 68.6% (sd = 1.1)
  - **Logistic regression**: mean generalization accuracy is 64.1% (sd =1.3)



- **Conclusion**: decision tree is better than logistic regression due to better average performance and stability.
  - Performance = generalization performance.

Overfitting Avoidance & Complexity Control

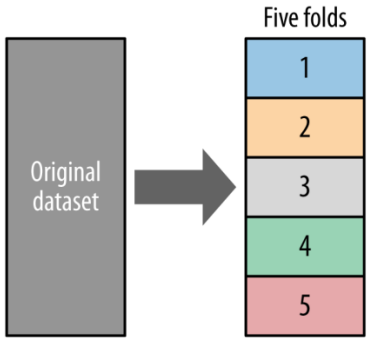
- **GridSearch with Cross-validation**
  - Split the data into train & test fold (usually shuffle data first).
  - Conduct k-fold cross validation on train fold (further split into sub-train and validation) to find the best complexity level.
    - For each complexity level, train model on sub-train and test it on validation for k times, calculate its mean generalization performance (cv score).
    - Compare mean generalization performances to find the best complexity.
  - With the best complexity chosen, build a model on the entire train fold, then evaluate its generalization performance on the test fold.

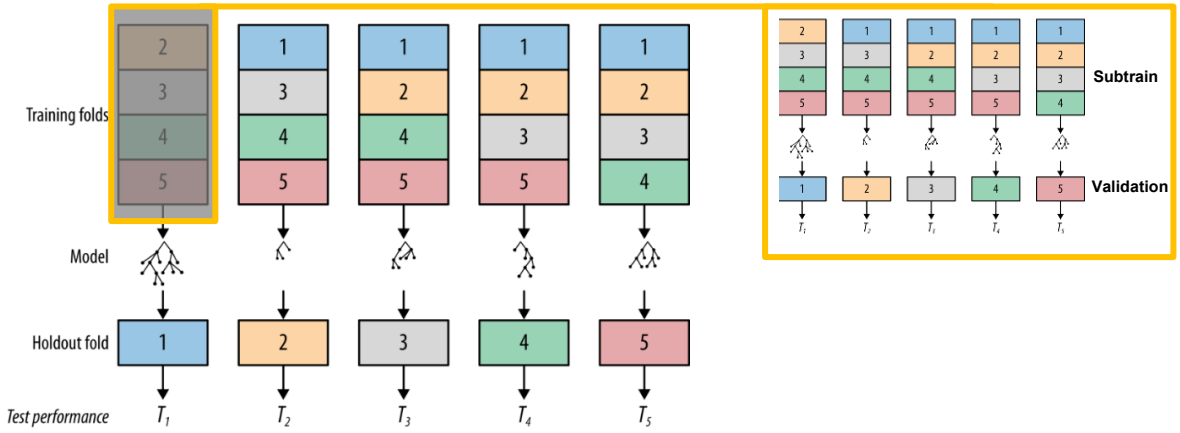


If we compare 2 complexity levels with GridSearch with 5-CV, how many models have been built?

Optional: Nested CV for GridSearch

- **Nested Cross-validation for GridSearch**
  - **Outer CV**: shuffles and splits the data into k-folds, and iterate over the data k times.
  - **Inner CV**: in each iteration, (1) conduct k-fold cross-validation on train fold to find the best complexity; and (2) train a model (with best complexity) on the entire train fold, evaluate it on the test fold.
    - Each train fold was further split into sub-train vs. validation.
    - Each iteration may return different best complexity, as different train data were used in inner CV. If so, choose the one rated as the best most of the time.





If we compare 2 complexity levels with Nested GridSearch with 5-CV, how many models have been built?

Recap: Objective Functions

- Linear Regression:
  - minimize sum of **squared error**

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

- Logistic Regression:
  - minimize sum of **log loss**

$$\min_w \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) \text{ where } \hat{y}_i = P(+)$$

- SVM:
  - minimize sum of **hinge loss**
    - 0 if on the correct side of its margin (and hyperplane).
    - Positive if on the wrong side of margin (or hyperplane).

$$\min_w \sum_{i=1}^{n} \xi_i \text{ where } \xi_i = \max_i [0, 1 - y_i \widehat{y_i}]$$

- All **objective functions** aim to **maximize the model fit**, i.e., minimize the error (or loss) in different formats.

$$\min_w error(x, w)$$

Avoid Overfitting for Mathematical Functions

- How can we find the balance between model fit and simplicity through **objective functions**?
  - Keeping a model simple helps with generalization performance.

$$\min_{w}[\, error(x, w) + \; penalty\,(w)]$$

- **Regularization**: instead of just maximizing model fit, we optimize a combination of fit and simplicity in the objective function.
  - By minimizing the error/loss on predictions, it maximizes model fit.
  - By minimizing the penalty on coefficients, it controls the magnitude of coefficients to keep a model simple.
    - Absolute coefficient = the **importance of features** in predicting the target.

Regularization with (Hyper)parameters

- L2 penalty (**Ridge**): the sum of squared coefficients
  - A model has larger (absolute) coefficients brings larger penalty in the objective function.

$$\sum_{j=1}^{p} w_j^2$$

- L1- penalty (**Lasso**): sum of the absolute coefficients
  - Zero out many coefficients, therefore, automatically perform **feature selection** and **complexity control**.

$$\sum_{j=1}^{p} |w_j|$$

- **Ridge** or **lasso regression** are regularized regression models: they minimize both residual (error) and the penalty on coefficients.
  - Linear regression is an unregularized model, which only minimize RSS.
- **GridSearchCV** is used to select best regularization (hyper)parameter α.
  - Larger α = stronger regularization: i.e., a simpler model with lots of coefficients either very small or 0.

$$\min_{w}[\, error(x, w) + \ \alpha * \ penalty\ (w)]$$

Regularization: Logistic Regression & SVM

- **SVM** $\min_w \left\{ \sum_{j=1}^{p} w_j^2 + C \sum_{i=1}^{n} \xi_i \right\}$ where $\xi_i = \max_i [0, 1 - y_i \hat{y}_i]$
- **Logistic Regression** $\min_w \left\{ \sum_{j=1}^{p} w_j^2 + C \sum_{i=1}^{n} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) \right\}$
- **C** is also a regularization (hyper)parameter:
  - C (always positive) is inversely proportional to regularization strength: a bigger C means weaker control of coefficient size.
- Bigger C: less tolerant of the loss --> more complicated model (better fit).
  - Larger coefficients: in SVM it also means a narrower margin with less SVs.
- Smaller C: more tolerant of the loss --> simpler model (worse fit).
  - Smaller coefficients: in SVM it also means a wider margin with more SVs.