Supervised Learning

Zhirong Yang
Department of Computer Science NTNU

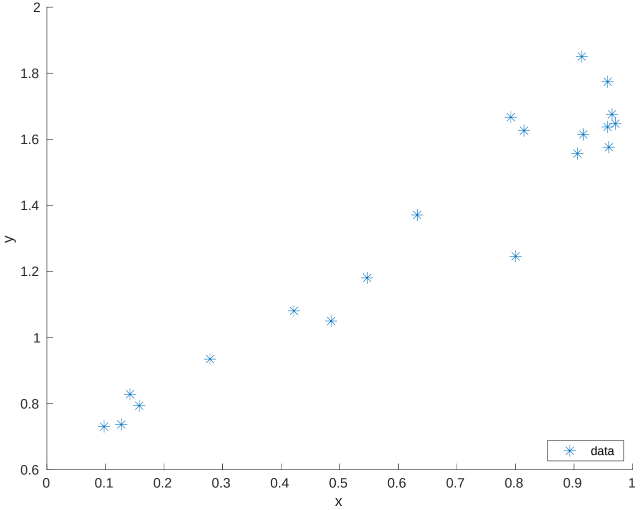
About this lecture

- To introduce the basics of supervised learning
  - Fundamental concepts
  - Simple regression methods
  - Simple classification methods
  - Underfitting, overfitting, validation, and complexity control
- After this lecture, you know
  - Typical supervised learning problem types
  - how to implement simple ML models
  - how to fit an ML model
  - how to tune hyperparameters of ML models

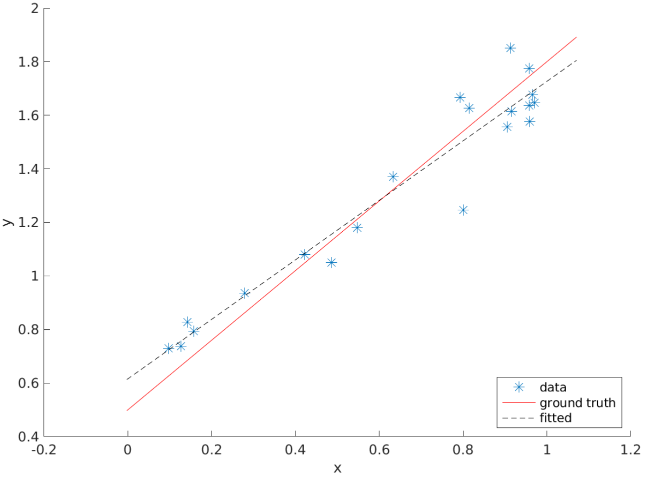
Supervised Learning

- Use a supervised training set $\{x_i, y_i\}_{i=1}^N$
- to fit a model (function) f to infer $y = f(x)$
- f should be accurate beyond the training set
- Typical supervised learning problems:
  - *classification*, where y is categorical
  - *regression*, where y is ordinal

Linear regression: example



Linear regression: example



Linear regression: model fitting (1d)

Training set $\{x_i, y_i\}_{i=1}^N$. Model $y = f(x) = wx + b$.

- Fitting objective:

$$\underset{w,b}{\text{minimize}} \quad \mathcal{J}(w,b) = \sum_{i=1}^{N} [y_i - (wx_i + b)]^2$$

- The minimal appears when $\frac{\partial \mathcal{J}}{\partial w} = 0$ and $\frac{\partial \mathcal{J}}{\partial b} = 0$
  - $\frac{\partial \mathcal{J}}{\partial b} = -2\sum_i [y_i - wx_i - b)] = 0$. So

$$b^* = \frac{1}{N}\sum_i y_i - w\frac{1}{N}\sum_i x_i = \bar{y} - w\bar{x}$$

  - $\mathcal{J}(w, b^*) = \sum_{i=1}^N [(y_i - \bar{y}) - w(x_i - \bar{x})]^2$
  - $\frac{\partial \mathcal{J}}{\partial w} = -2\sum_i [(y_i - \bar{y}) - w(x_i - \bar{x})]\,(x_i - \bar{x}) = 0$. So

$$w^* = \frac{\sum_i (y_i - \bar{y})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

Linear regression: model fitting (multidimensional)

Training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$. Model $y = f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.

- Fitting objective:

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \mathcal{J}(\mathbf{w}, b) = \sum_{i=1}^{N} \left[y_i - (\mathbf{w}^T\mathbf{x}_i + b) \right]^2$$

- We write

$$\mathbf{u} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \dots & x_{Nd} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

$$\mathcal{J}(\mathbf{w}, b) = \mathcal{J}(\mathbf{u}) = \|\mathbf{y} - \mathbf{X}\mathbf{u}\|^2 = \mathbf{u}^T\mathbf{X}^T\mathbf{X}\mathbf{u} - 2\mathbf{y}^T\mathbf{X}\mathbf{u} + \mathbf{y}^T\mathbf{y}$$

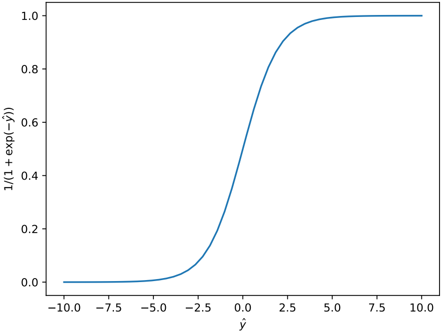
$\frac{\partial \mathcal{J}}{\partial \mathbf{u}} = 2\mathbf{X}^T\mathbf{X}\mathbf{u} - 2\mathbf{X}^T\mathbf{y} = 0$. So $\mathbf{u}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$

See matrix cookbook if needed.

Logistic regression

Logistic regression (LR) is for classification.

- For two-class classification, $y \in \{0, 1\}$
- Directly fit $y \approx \hat{y} = \mathbf{w}^T\mathbf{x} + b$ is not good because $\hat{y}$ can be arbitrary values (hard to interpret)
- LR fits $y \approx \sigma(\hat{y}) = \frac{1}{1+\exp(-\hat{y})}$



Logistic regression: fitting

- The fitting is an optimization over $\mathbf{u} = (\mathbf{w}, b)$:

$$\underset{\mathbf{u}}{\text{minimize}} \quad \mathcal{J}(\mathbf{u}) = \sum_{i=1}^{N} -y_i \log \sigma_i - (1 - y_i) \log(1 - \sigma_i),$$

  where $\sigma_i = \sigma(\hat{y}_i)$.
- Steepest descent method (η: step size)
  1. Randomly initialize $\mathbf{u}$
  2. Repeat
     2.1 calculate the gradient $\nabla = \frac{\partial \mathcal{J}}{\partial \mathbf{u}}$
     2.2 update $\mathbf{u} \leftarrow \mathbf{u} - \eta \nabla$
  3. until $\dfrac{\|\mathbf{u} - \mathbf{u}^{\text{old}}\|}{\|\mathbf{u}^{\text{old}}\|} < \epsilon$

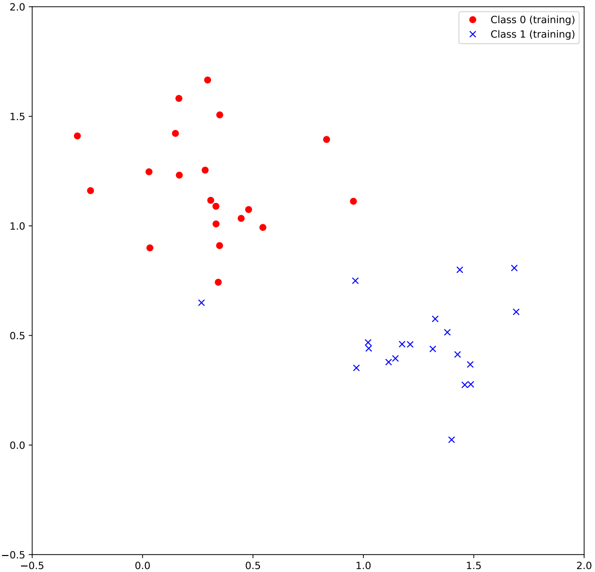
Deriving the gradient

$\mathcal{J}(\mathbf{u}) = \sum_{i=1}^{N} -y_i \log \sigma_i - (1 - y_i) \log(1 - \sigma_i)$, where
$\sigma_i = \frac{1}{1 + \exp(-\hat{y}_i)}$, $\hat{y}_i = \mathbf{w}^T \mathbf{x}_i + b$, and $\mathbf{u} = (w_1, \ldots, w_D, b)$.

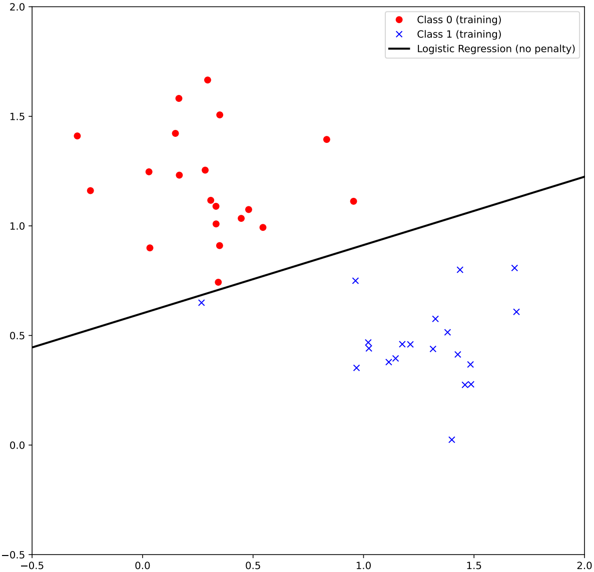
$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_d} &= \sum_i \frac{\partial \mathcal{J}}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial w_d} \\
&= 2 \sum_i \left(-\frac{y_i}{\sigma_i} + \frac{1 - y_i}{1 - \sigma_i} \right) \sigma_i \, (1 - \sigma_i) \, x_{id} \\
&= 2 \sum_i (\sigma_i - y_i) \, x_{id} \\
\frac{\partial \mathcal{J}}{\partial b} &= 2 \sum_i (\sigma_i - y_i)
\end{aligned}$$

Hint: try to avoid loops in Python; use built-in array functions instead.

Logistic regression: example



Logistic regression: example

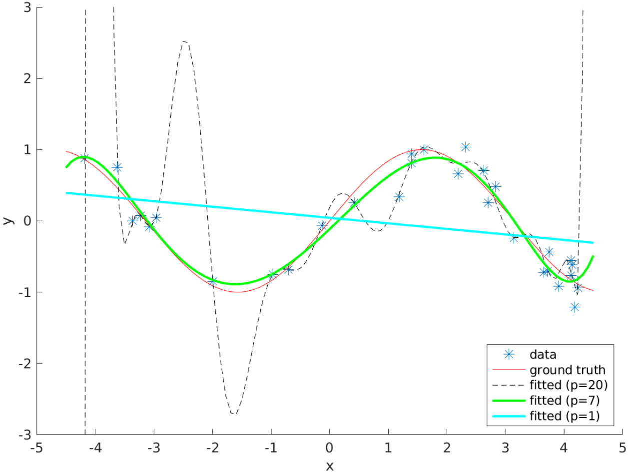


Underfitting and Overfitting

Polynomial regression

- $f(x) = w_p x^p + \cdots + w_1 x + w_0$, where p is a hyperparameter
- reduces to linear regression when $p = 1$
- can be fitted like multidimensional linear regression, with $X_{ik} = x_i^k$ $(k = 0, \ldots, p)$

Polynomial regression: underfitting and overfitting



- Underfitting = too simple; miss too much details
- Overfitting = too complex; sensitive to small perturbations

Hyperparameter tuning: validation

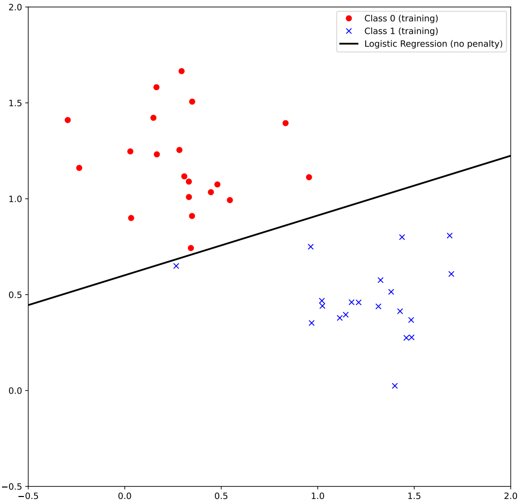
How can we select the best hyperparameters?

- Minimizing training loss $\not\Longrightarrow$ best hyperparameters
- Minimizing testing loss $\Longrightarrow$ best hyperparameters
- But we don't have testing data during the selection
- Solution: split the original training data into
  - *training*
  - *validation*
- Estimate the testing loss with the validation loss
- Select the hyperparameters with the smallest validation loss
- and then re-fit with the whole original training data

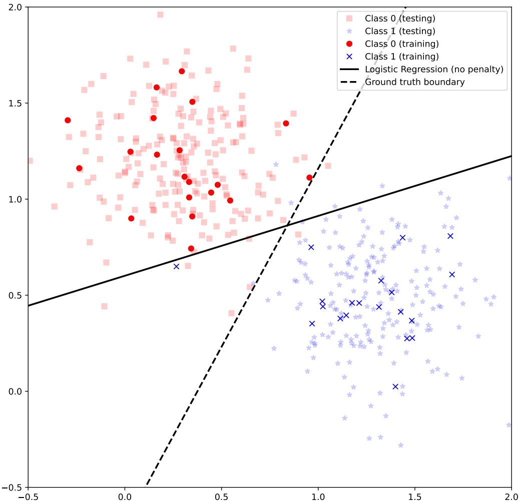
Complexity control

- Training loss itself is not enough as an ML objective
- It should work with some complexity control
  - constraints, e.g.,
    - maximal number of non-zero parameters
    - minimal smoothness (maximal gradient magnitude)
  - additional penalty terms, e.g., training loss $+\lambda\|\mathbf{w}\|_p$
    - $\|\mathbf{w}\|_0$ counts the number of non-zeros in $\mathbf{w}$
    - $\|\mathbf{w}\|_2$ measures the reciprocal of class margin

Complexity control example: penalized logistic regression

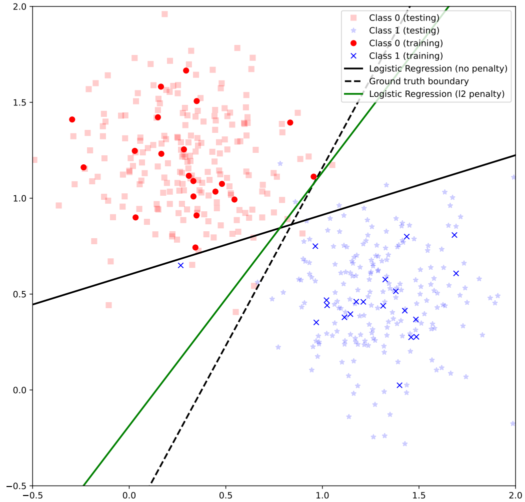


Complexity control example: penalized logistic regression



Complexity control example: penalized logistic regression

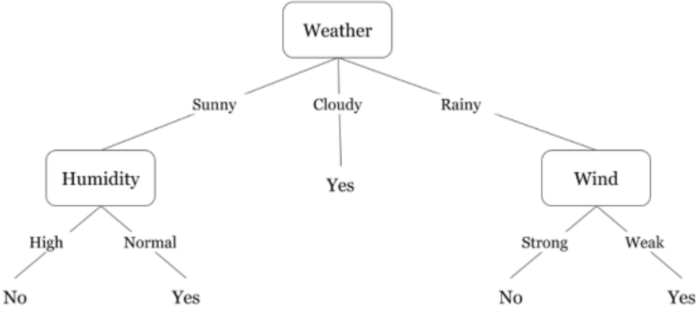
$$\underset{\mathbf{w},b}{\text{minimize}} \sum_{i=1}^{N} \left[y_i - \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}_i - b)} \right]^2 + 100\|\mathbf{w}\|_2^2$$



Other classification methods

Decision Tree: example

Go for tennis?



Decision Tree: fitting/tree building

- Repeatedly select the best feature to split the data
- Stop splitting, e.g., when reaching
  - maximum depth,
  - minimum number of samples for each split,
  - minimum number of samples in the leaf node, or
  - maximum number of features
- Pruning could be applied to control complexity
- Inference
  - navigate to a leaf node using the splitting rules
  - return the label of the leaf node

Which feature to split?

At a certain node i, choose the feature which maximizes

$$\begin{aligned}\text{Gain} =& \text{Impurity before split} - \text{Impurity after split} \\ =& \text{Impurity}(i) - \sum_{j \in \text{ChildrenOf}(i)} \text{Impurity}(j)\end{aligned}$$

Example Impurity functions:

- GINI:

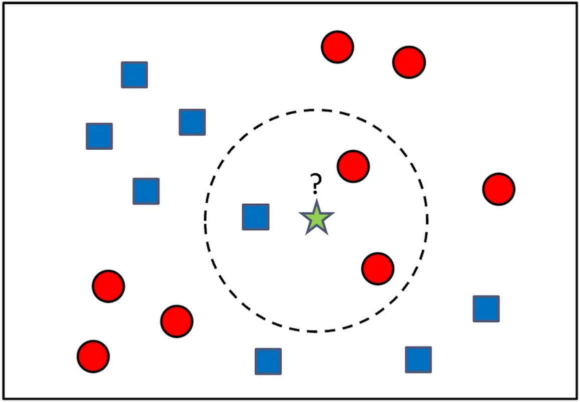
$$\text{Impurity}(i) = \sum_c P(c|i)\,[1 - P(c|i)] = 1 - \sum_c [P(c|i)]^2$$

- Entropy:

$$\text{Impurity(i)} = -\sum_c P(c|i) \log_2 P(c|i)$$

where $P(c|i)$ is the fraction of samples belonging to class c at i.

K-Nearest Neighbor Classifier



Naive Bayes Classifier

Naive Bayes (NB) classifiers = simple "probabilistic classifiers"

- By Bayes rule: $p(C = k|\mathbf{x}) = \dfrac{p(\mathbf{x}|C = k)p(C = k)}{p(\mathbf{x})}$
- NB specifies
  - $p(C = k)$ by uniform or empirical frequency
  - a simple probabilistic model (e.g., Gaussian, multinomial, Bernoulli) of $p(\mathbf{x}|C = k) = \prod_{d=1}^{D} p(x_d|C = k)$
  - "naive" = assume the features are mutually independent
- NB classifies by

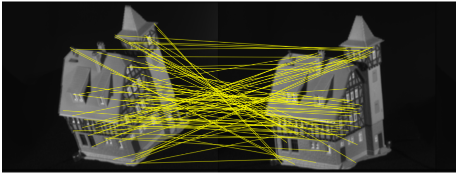
$$\arg\max_k \quad \log p(C = k) + \sum_d \log p(x_d|C = k)$$

- usually leads to a linear classifier

Other types of supervised learning

To give you some flavor beyond standard classification/regression

- Recommendation (ranking)
  - like regression, but we only care the top scores
- Matching
  - discrete solution space; need good approximation



- Active learning
  - actively pick the supervised pairs for each round
- Multitask learning
  - sharing parameters in classifiers/regressors