Logistical Regression and Scikit-Learn

Machine Learning for Engineering Applications

Fall 2023

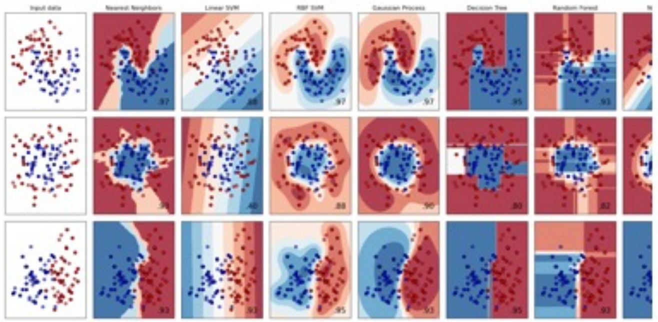
Scikit-learn

- Website:
  - https://scikit-learn.org/stable/
  - Simple and efficient tools for data mining and data analysis
  - Accessible to everybody, and reusable in various contexts
  - Built on **NumPy**, **SciPy**, and **matplotlib**
  - Open source, commercially usable

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

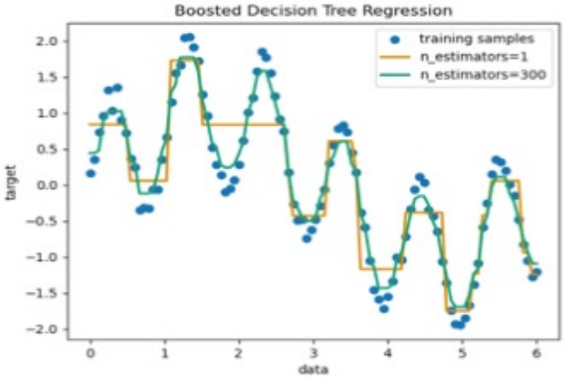


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

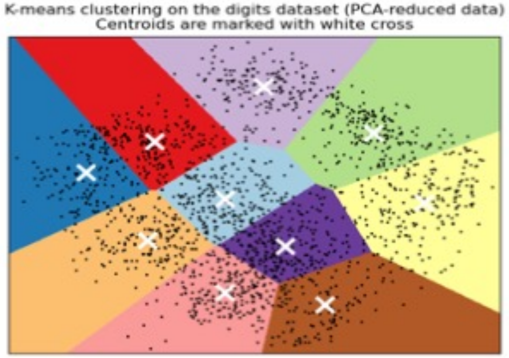


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

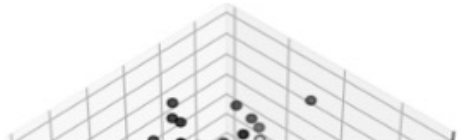


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...



Model selection

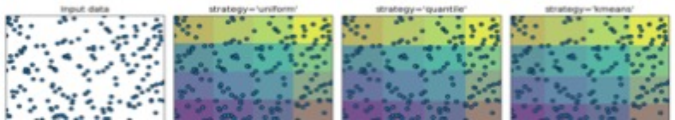
Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Logistic Regression (classification)

- Its logistic because of **probabilities**…not binary logic
- This method is used when linear classification does not perform well in creating categories
- Mode of thinking:

$$logit(p) = log \frac{p}{(1-p)}$$

$$logit(p(y=1|\boldsymbol{x})) = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

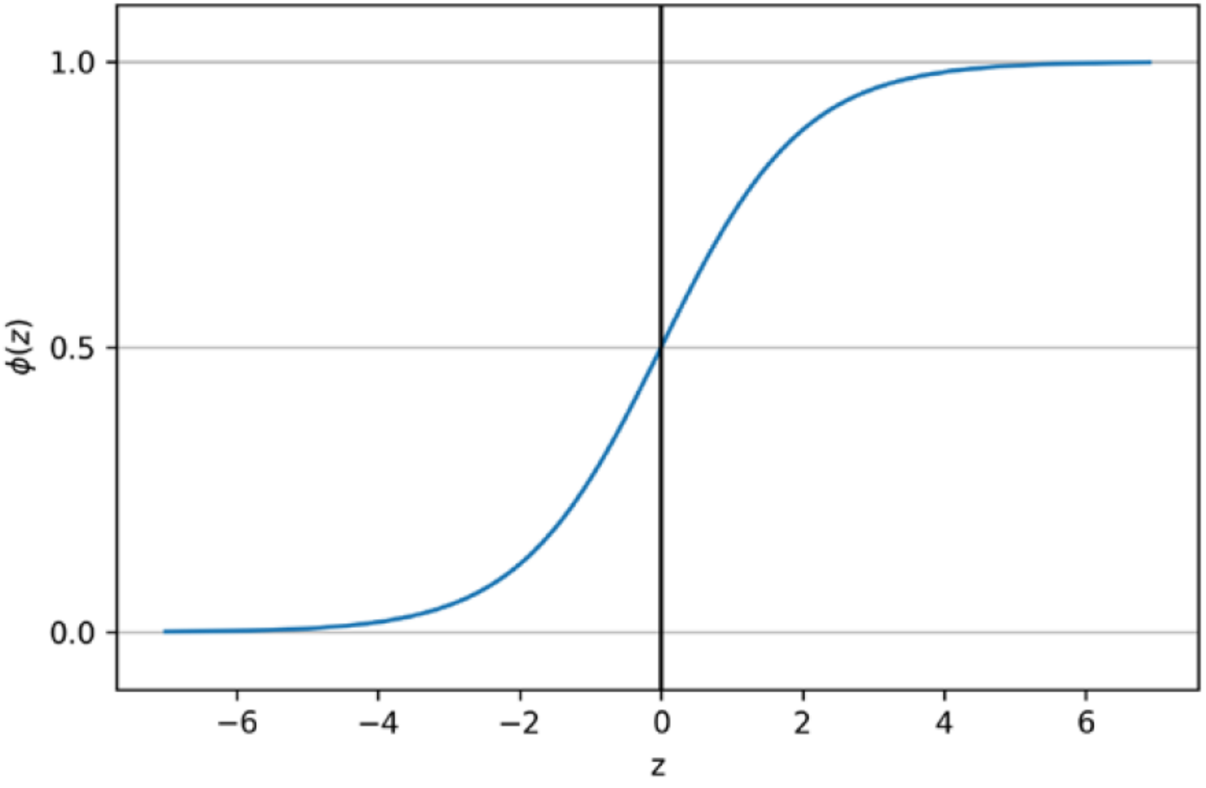
- Goal: Predict the probability of a sample goes into a class

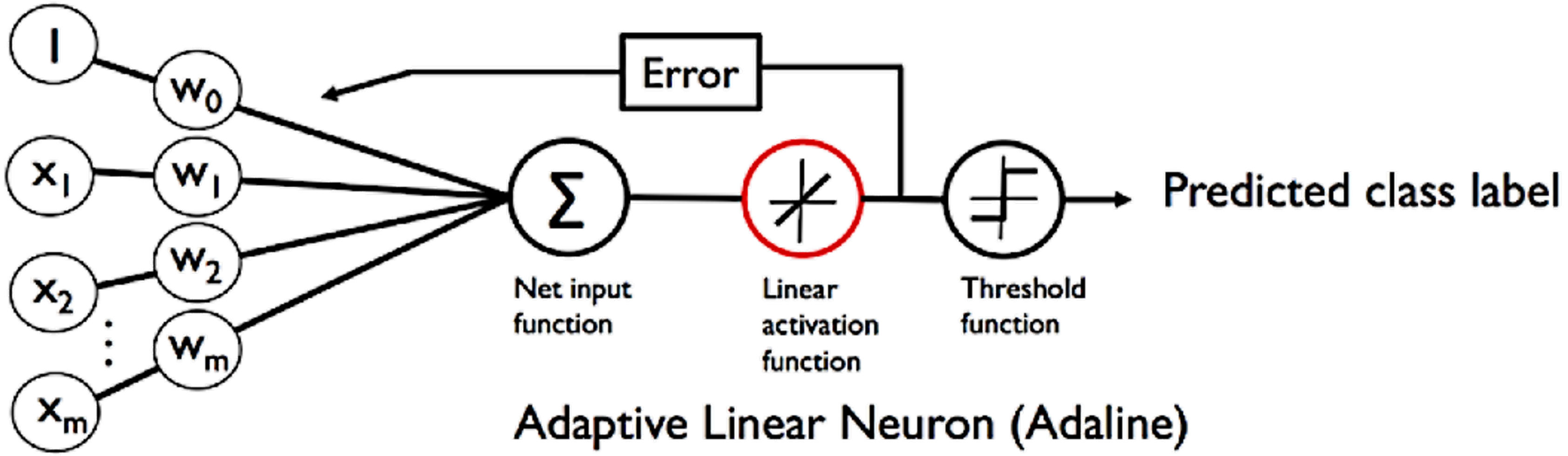
- So, we need the inverse of *logit()*$^{-1}$

- Inverse: sigmoid-function

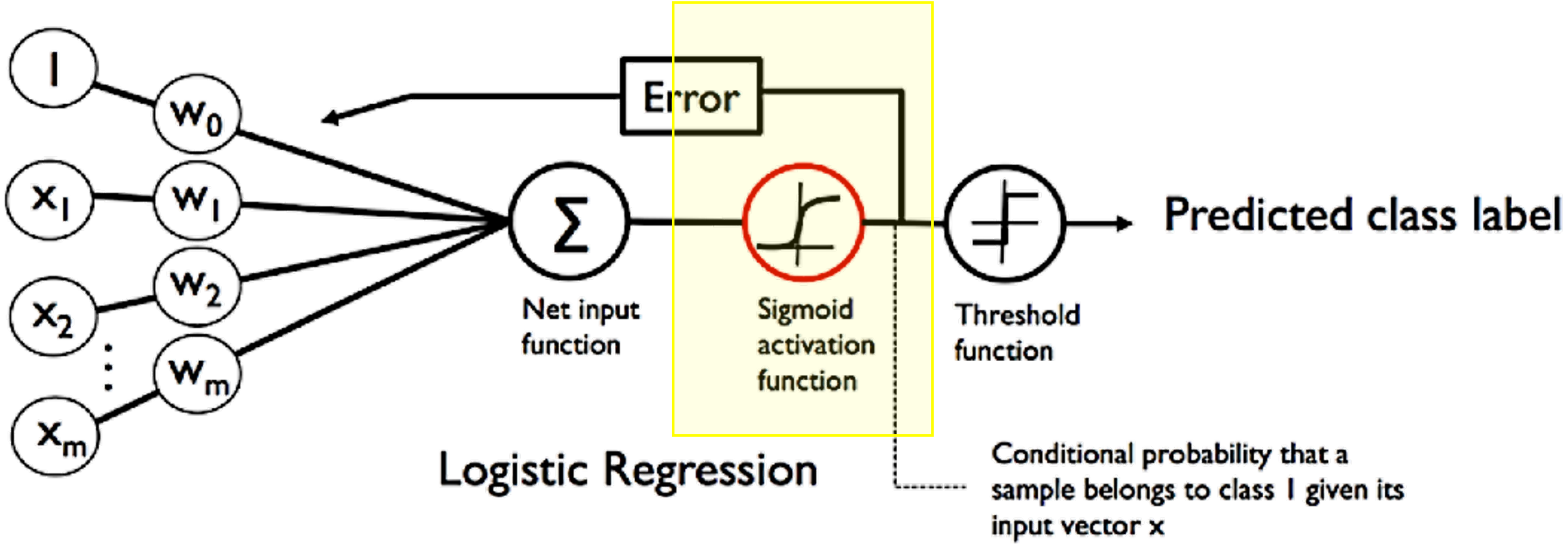
- **Activation function** is the sigmoid-function

$$\phi(z) = \frac{1}{1+e^{-z}}$$





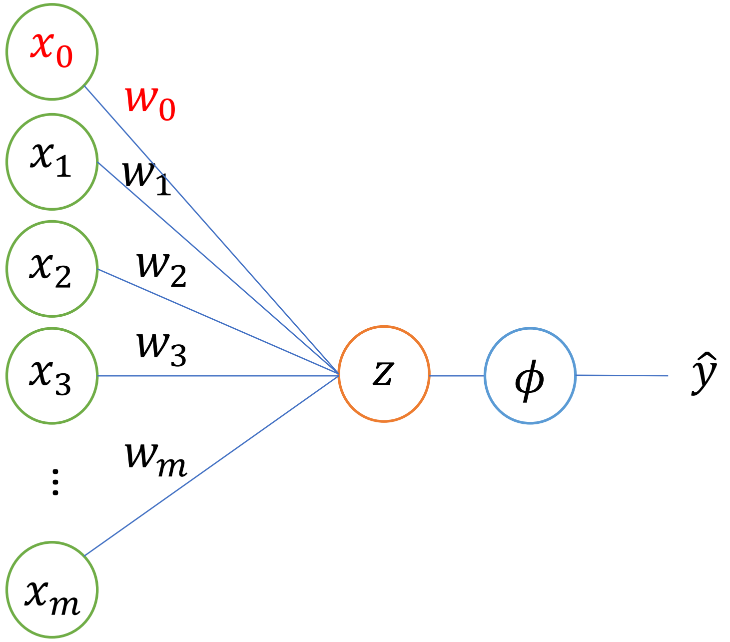
Adaptive Linear Neuron (Adaline)

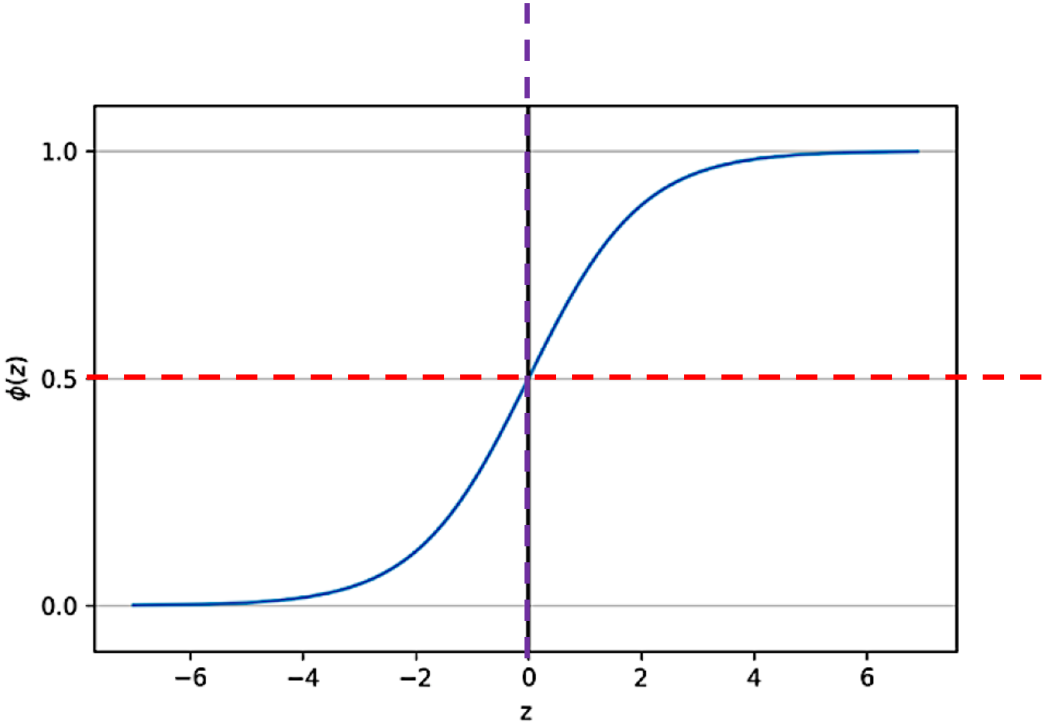


Logistic Regression

- To simplify the decision:

$$\hat{y} = \begin{cases} 1 & if\ z \geq 0.0 \\ 0 & otherwise \end{cases}$$





$$\hat{y} = \begin{cases} 1 & if\ \phi(z) \geq 0.5 \\ 0 & otherwise \end{cases}$$

Logistic Cost Function

- Review, **SSE**: $$J(\boldsymbol{w}) = \sum_i \frac{1}{2}\left(\phi\left(z^{(i)}\right) - y^{(i)}\right)^2$$

- **Likelihood** (*L*) is based on a probability value as a function of how well the weights are adjusted in the network:

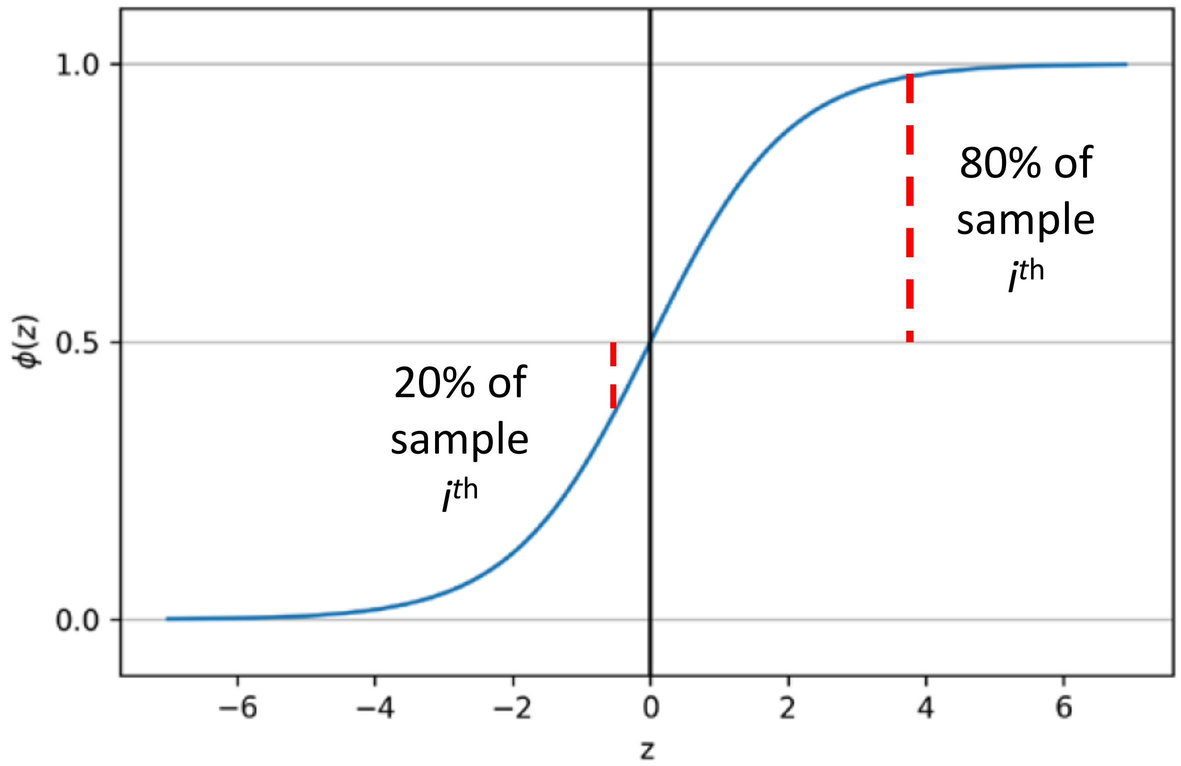
$$L(\boldsymbol{w}) = P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{w})$$

- Therefore,

$$P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{w}) = \prod_{i=1}^{n} P\left(y^{(i)} \mid x^{(i)}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

- Maximize likelihood with natural log:

$$l(\boldsymbol{w}) = \log L(\boldsymbol{w}) = \sum_{i=1}^{n} \left[y^{(i)} \log\left(\phi\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$



- Therefore, the **cost function**:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right]$$

, and it must be deterministic as the sigmoid decision:

$$J(\phi(z), y; \boldsymbol{w}) = \begin{cases} -\log(\phi(z)) & \text{if } y = 1 \\ -\log(1-\phi(z)) & \text{if } y = 0 \end{cases}$$

- Weights get updated with gradient descent:

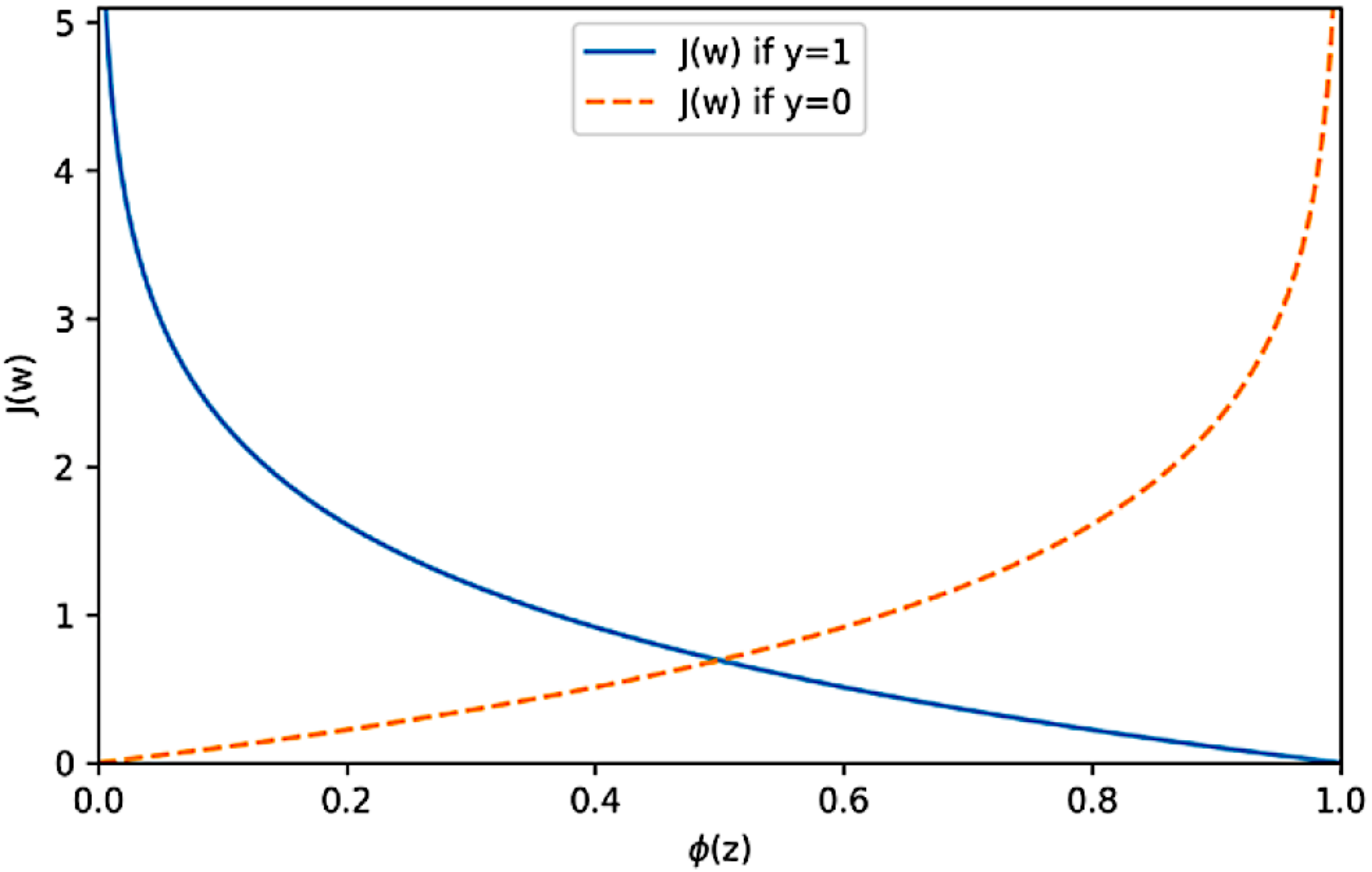
$$\underbrace{w_j := w_j + \eta \sum_{i=1}^{n} \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) x_j^{(i)}}$$

$\boxed{\text{Adaline}}$

$$\Delta w_j = -\eta \underbrace{\frac{\partial J}{\partial w_j}}$$

$\boxed{\text{Logistic Regression}}$

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left[-y^{(i)} \log\left(\phi\left(z^{(i)}\right) \right) - \left(1 - y^{(i)}\right) \log\left(1 - \phi\left(z^{(i)}\right)\right) \right]$$


J(w) if y=1
J(w) if y=0
J(w)
ϕ(z)
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0

Over & Under Fitting

- **Overfitting** a model is a very common issue in mathematics, statistics, and Machine Learning

- Model does well with the training data

- Model "*fails*" with the test data

- ***Reasons***: many
  - Too many parameters, or
  - Model to complex for the parameters, or
  - Quality of data is bad, or
  - Not enough data, or
  - Regulation of weights, or
  - Etc.…

- **Underfitting:** the model is not sophisticated enough to handle the number of parameters from the dataset

- Model will have to scarifies precision with very “loose” decisions

- ***Fix***:
  - Change the type of model, or
  - Reduce the # of features, or
  - Quality of data is bad, or
  - Regulation of weights, or
  - Etc.…

- The **L1 & L2 Regularization** techniques help model to not over (or under) fit

- L2 regularization is defined as:

$$\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

- Lambda is the regularization parameter (*Dr. Valles's terms: the magnitude of the penalty*)

- The L2 helps a cost function to level the playing field of the weights -> *preventing overfitting* -> *model learns in a more flexible manner*.

- The effect in the ***Logical Regression Cost Function***:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi\left(z^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-\phi\left(z^{(i)}\right)\right)\right] \boxed{+\frac{\lambda}{2}\|w\|^2}$$

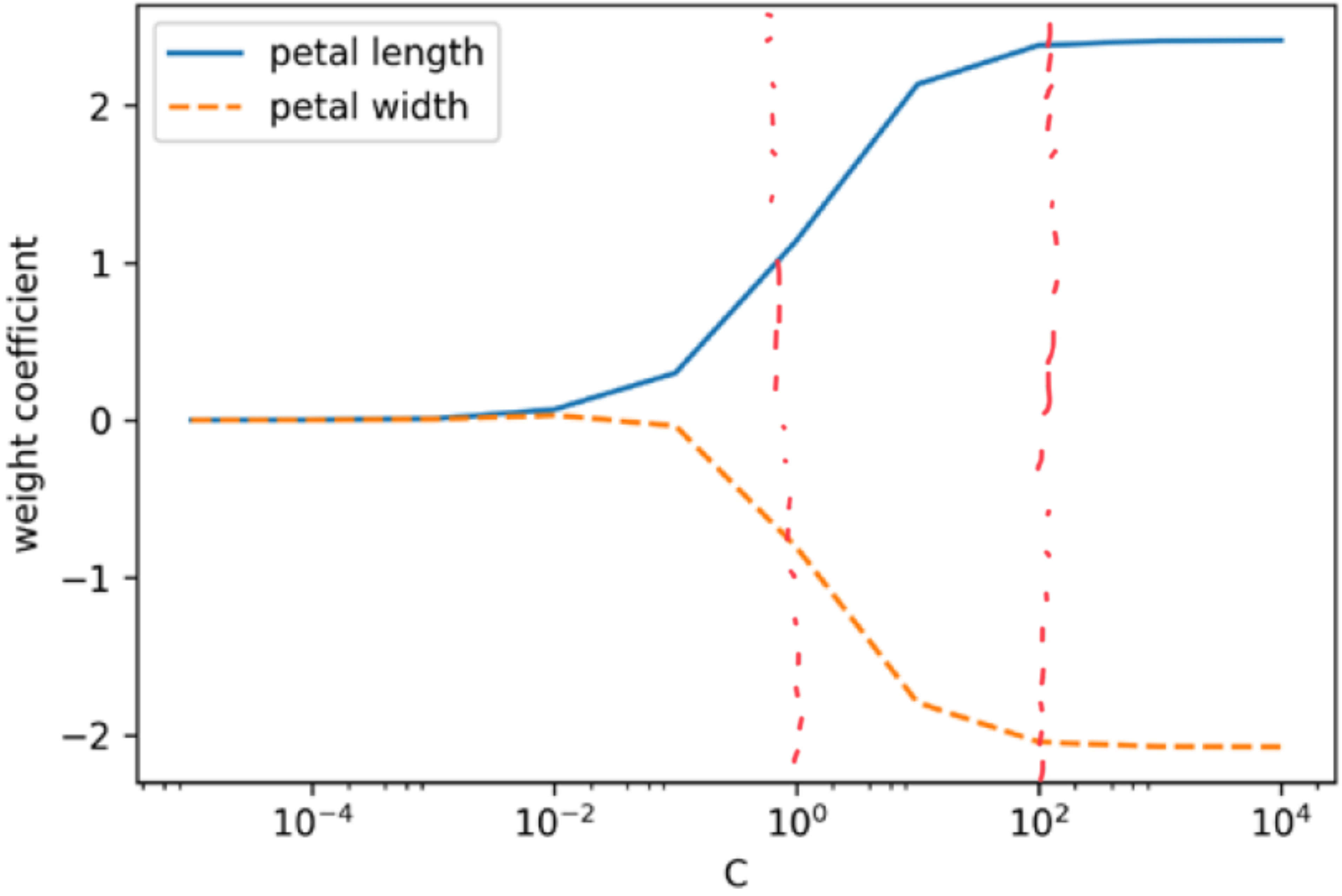
- Lambda -> increase values -> the regularization strength increases

- In Python code: $\lambda = \frac{1}{C}$

- **C-parameter**: Inverse Regularization Parameter

```
weights, params = [], []
for c in np.arange(-5, 5):
    ... lr = LogisticRegression(C=10.**c, random_state=1)
    ... lr.fit(X_train_std, y_train)
    ... weights.append(lr.coef_[1])
    ... params.append(10.**c)
weights = np.array(weights)

plt.plot(params, weights[:, 0], label='petal length')
plt.plot(params, weights[:, 1], linestyle='--',
    ... label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
plt.show()
```



petal length
petal width
weight coefficient
2
1
0
−1
−2
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
C

Python IDEs

- Jupyter Notebooks https://jupyter.org/
- ATOM: https://atom.io/
- PyCharm: https://www.jetbrains.com/pycharm/
- Visual Studio Code (VS Code): https://code.visualstudio.com/
- Spyder: https://www.spyder-ide.org/ or though…
- Anaconda: https://www.anaconda.com/

For next class…

- Continue reading Chapters 2 & 3
- Homework #1 will be posted on Wed.
- LEAP Cluster training on Wed.

Hands On

- Open your IDE to develop the in-class exercise:

- Use the IRIS dataset on the Logistic Regression model
  - **Find the best C-values for the highest accuracy**

- You can work solo, in pair, or table!

- Provide the following analysis:
  - Training accuracy
  - Test accuracy
  - Number of test misclassifications
  - **Use the code in Page 56** to plot the decision regions