5.
Logistic Regression

Logistic Regression

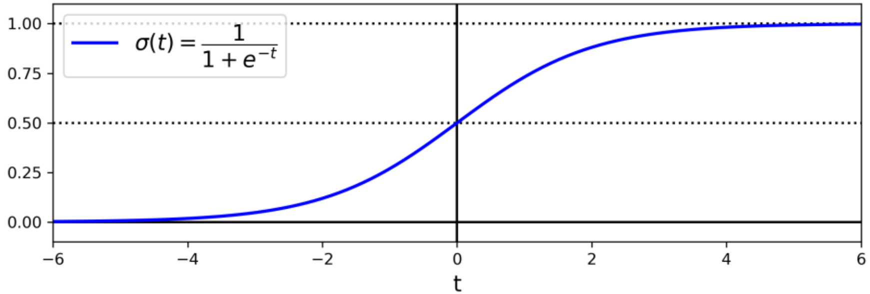
- *Logistic Regression* is used to estimate the probability that an instance belongs to a particular class.
- If the estimated probability is greater than a threshold (e.g., 50%), then the model predicts that the instance belongs to that class (called the *positive class*, labeled "1"), and otherwise it predicts that it does not (i.e., the *negative class*, labeled "0").
- Like a linear regression, a logistic regression model computes a weighted sum of the input features (plus a bias term), but instead of outputting the result directly, it outputs the *logistic* of this result: $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^T \boldsymbol{\theta})$

Estimating Probabilities

- The logistic $\sigma(\cdot)$ is a *sigmoid function* (i.e., *S*-shaped) that outputs a number between 0 and 1:

$$\sigma(t) = \frac{1}{1+\exp(-t)}$$

$$\hat{p} = \frac{1}{1 + e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_n)}}$$



- The logistic regression estimates the probability $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x})$ that instance $\mathbf{x}$ belongs to the positive class, it can predict $\hat{y}$:

$$\hat{y} = \begin{cases} 0, & \text{if} \quad \hat{p} < 0.5 \\ 1, & \text{if} \quad \hat{p} \geq 0.5 \end{cases}$$

Training and Cost Function

- *Objective of training:* set the parameter vector $\boldsymbol{\theta}$ so that the model estimates high probabilities for positive instances ($y = 1$) and low probabilities for negative instances ($y = 0$).
- Cost function of a single training instance:

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

- The cost function over the whole training set (*log loss*) is the average cost over all training instances:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) log\left(1 - \hat{p}^{(i)}\right)\right]$$

Training and Cost Function

- The log loss is convex, so Gradient Descent is guaranteed to find the global minimum.
    - if the learning rate is not too large and you wait long enough.

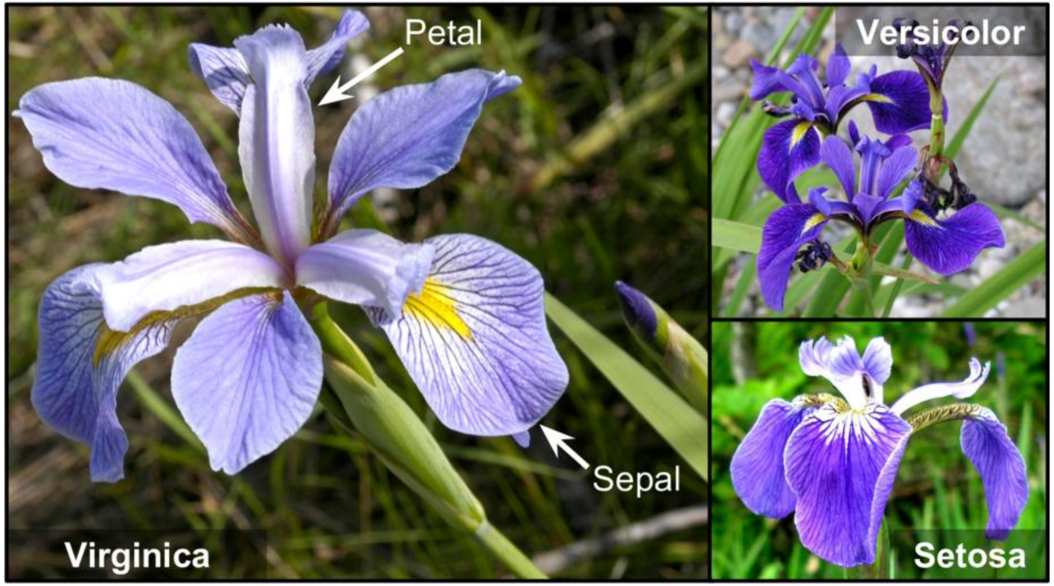
$$\frac{\partial}{\partial \theta_j} \mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

- Compare with partial derivatives for linear regression:

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Iris Flower Dataset

- The dataset that contains the sepal and petal length and width of 150 iris flowers of three different species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*.



Load the Dataset

```python
from sklearn.datasets import load_iris

iris = load_iris(as_frame=True)
list(iris)
```

```
['data',
 'target',
 'frame',
 'target_names',
 'DESCR',
 'feature_names',
 'filename',
 'data_module']
```

```python
iris.data.head(3)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

```python
iris.target.head(3)  # note that the instances are not shuffled
```

```
0    0
1    0
2    0
Name: target, dtype: int32
```

```python
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Build a Simple Classifier

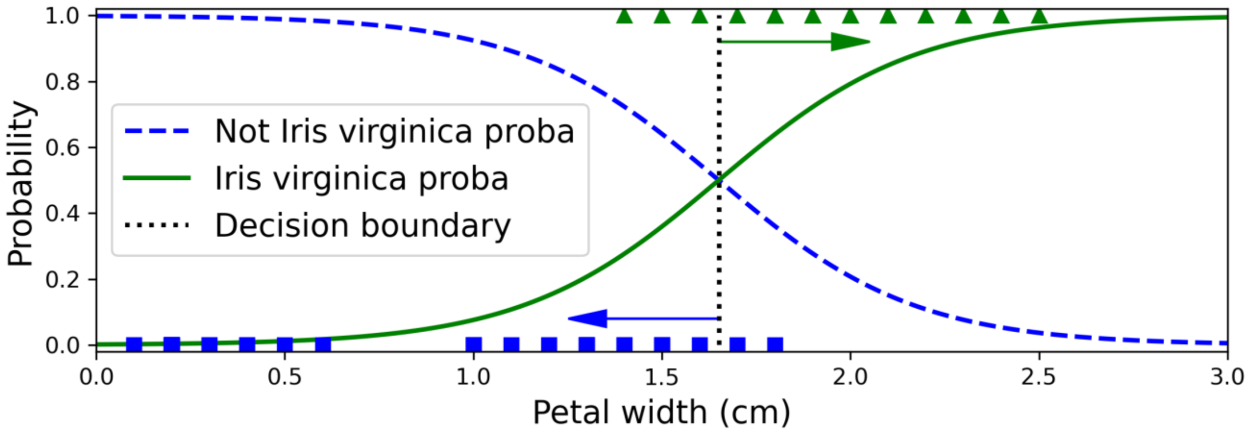
- Build a classifier to detect the *Iris virginica* type based only on the *petal width* feature:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica'
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

Decision Boundaries

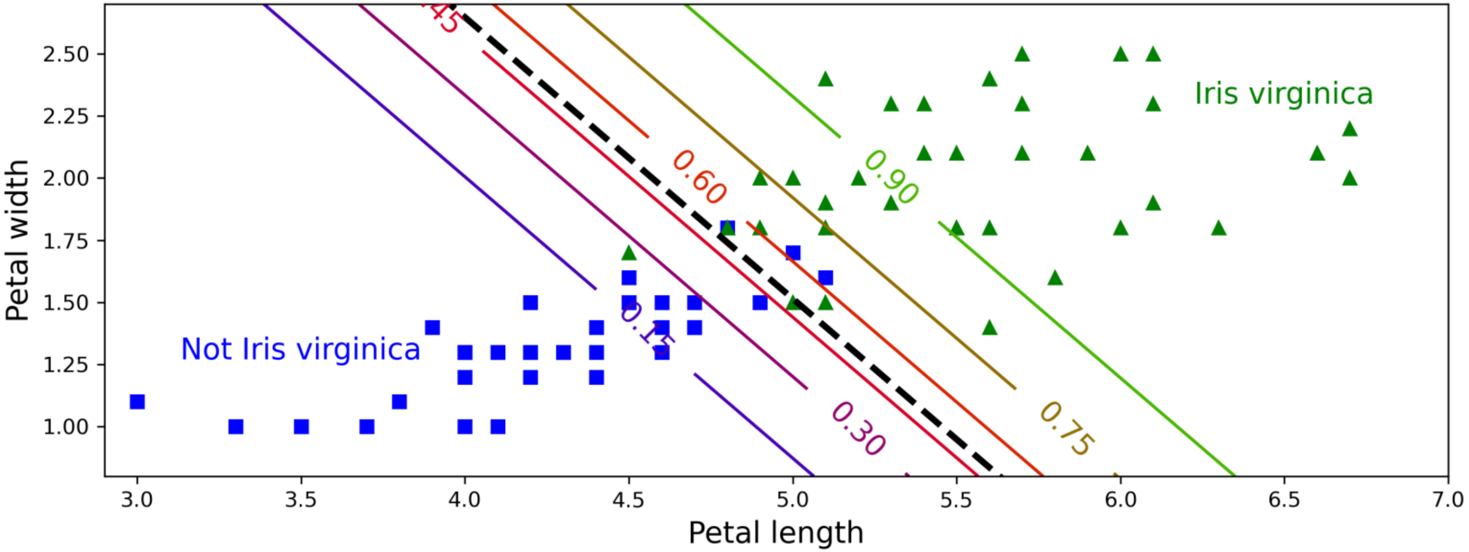
➢ The model's estimated probabilities for flowers with petal widths varying from 0 cm to 3 cm:

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)  # reshape to get a column vector
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0, 0]
```



Decision Boundaries

- A logistic regression model to detect the *Iris virginica* type based on two features: *petal length* and *width*.



Softmax Regression

- *Softmax Regression* (aka. *Multinomial Logistic Regression*): generalized version of the logistic regression model that support multiple classes directly, without having to train and combine multiple binary classifiers.
- Given an instance $\mathbf{x}$, the softmax regression first computes a score $s_k(\mathbf{x})$ for each class k, then estimates the probability of each class by applying the *softmax function* to the scores:

$$s_k(\mathbf{x}) = \left(\boldsymbol{\theta}^{(k)}\right)^T \mathbf{x}$$

$$\hat{p}_k = \sigma\left(\mathbf{s}(\mathbf{x})\right)_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Softmax Regression

- The softmax regression classifier predicts the class with the highest estimated probability.
  - which is simply the class with the highest score:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^{\mathsf{T}}\mathbf{x}\right)$$

- The softmax regression classifier predicts only one class at a time (i.e., it is multiclass, not multioutput).

Training Softmax Regression

- The objective is to have a model that estimates a high probability for the target class.
- This is equivalent to minimizing the *cross entropy* cost function:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- $y_k^{(i)}$ is the target probability that the i-*th* instance belongs to class k.
- When there are just two classes ($K = 2$), this cost function is equivalent to *log loss* (the Logistic Regression's cost function).

Training Softmax Regression

- Scikit-Learn's `LogisticRegression` uses softmax regression automatically when you train it on more than two classes.
- We also need a solver that supports softmax regression.
  - The default solver "`lbfgs`" solver, is such a solver.
- It also applies ℓ_2 regularization by default, which you can control using the hyperparameter `C`:

```
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

softmax_reg = LogisticRegression(C=30, random_state=42)
softmax_reg.fit(X_train, y_train)
```

Decision Boundaries

➢ What is type of an iris with petals that are 5cm long and 2cm wide?

```
softmax_reg.predict([[5, 2]])
```

```
array([2])
```

```
softmax_reg.predict_proba([[5, 2]]).round(2)
```

```
array([[0.  , 0.04, 0.96]])
```