ECE 469/ECE 568 Machine Learning

Textbook:
Machine Learning: a Probabilistic Perspective by Kevin Patrick Murphy

Southern Illinois University

October 4, 2024

Cost function: Logistic regression for classification

- Let Y be the output label of a vector of input features of a binary classification problem.
- There are only two classes $\mathcal{C}_0$ and $\mathcal{C}_1$.
- Assume that the output label is zero ($Y = 0$) if the corresponding object belongs to class $\mathbf{C}_0$.
- The output label is one ($Y = 1$) if the corresponding object belongs to class $\mathbf{C}_1$.
- Thus, the output label can take only two values $Y \in \{0, 1\}$, and hence, Y can be modeled as a Bernoulli random variable given the class.

$$Y|\mathcal{C}_i \in \{0, 1\} \sim \text{Bernoulli}(P) \quad \text{for} \quad i = 1, 2.$$

- More specifically, if $Y \sim \text{Bernoulli}(P)$, then the probability mass function (PMF) of Y can be written as $P(Y = 1) = P$ and $P(Y = 0) = 1 - P$.
- PMF of a Bernoulli random variable $Y \in \{0, 1\}$ can be rewritten as

$$P(Y = y) = P^y(1 - P)^{(1-y)}.$$

Cost function: Logistic regression for classification

- Let the data set be $\{\mathbf{x}_m, y_m\}$, and the output variables are $y_m \in \{1, 0\}$ for $m = 1, \cdots, M$.
- The posterior probability of $\mathcal{C}_1$ given $\mathbf{x}_m$ is given by

$$P(\mathcal{C}_1|\mathbf{x}_m) = \sigma(\mathbf{w}^T\mathbf{x}_m)$$

- The likelihood function is given by

$$p(\mathbf{y}|\mathbf{w}) = \prod_{m=1}^{M} \left[P(\mathcal{C}_1|\mathbf{x}_m)\right]^{y_m} \left[P(\mathcal{C}_2|\mathbf{x}_m)\right]^{1-y_m}$$

- Since $P(\mathcal{C}_2|\mathbf{x}_m) = 1 - P(\mathcal{C}_1|\mathbf{x}_m)$, we have

$$p(\mathbf{y}|\mathbf{w}) = \prod_{m=1}^{M} \left[P(\mathcal{C}_1|\mathbf{x}_m)\right]^{y_m} \left[1 - P(\mathcal{C}_1|\mathbf{x}_m)\right]^{1-y_m}$$

Cost function: Logistic regression for classification

- The error/cost function is defined to be the negative of the log-likelihood function as

$$J(\mathbf{w}) = -\ln(p(\mathbf{y}|\mathbf{w})) = -\ln\left[\prod_{m=1}^{M}\left[P(\mathcal{C}_1|\mathbf{x}_m)\right]^{y_m}\left[1-P(\mathcal{C}_1|\mathbf{x}_m)\right]^{1-y_m}\right]$$

- By expanding the logarithm, we have

$$J(\mathbf{w}) = -\sum_{m=1}^{M}\left[y_m \ln(P(\mathcal{C}_1|\mathbf{x}_m)) + (1-y_m)\ln(1-P(\mathcal{C}_1|\mathbf{x}_m))\right]$$

- In ML, this error function is known as the cross-entropy error function.
- Recall that $P(\mathcal{C}_1|\mathbf{x}_m) = \sigma(\mathbf{w}^T\mathbf{x}_m)$, and $p(\mathcal{C}_2|\mathbf{x}_m) = 1-\sigma(\mathbf{w}^T\mathbf{x}_m)$.

Cross-entropy cost function for logistic regression

$$J(\mathbf{w}) = -\sum_{m=1}^{M}\left[y_m \ln(\sigma(\mathbf{w}^T\mathbf{x}_m)) + (1-y_n)\ln(1-\sigma(\mathbf{w}^T\mathbf{x}_m))\right]$$

Cost function: Logistic regression for classification

Cross-entropy cost function for logistic regression

$$J(\mathbf{w}) = -\sum_{m=1}^{M} \left[y_m \ln(\sigma(\mathbf{w}^T \mathbf{x}_m)) + (1 - y_n) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x}_m)) \right]$$

- Bad news: There is no known closed-form equation to compute the value of $\mathbf{w}$ that minimizes this cost function. That is there is no equivalent of the Normal equation that we found for the optimal weight vector for linear regression.
- Good news: This cost function is convex, and hence, the gradient descent (or any other optimization algorithm) is guaranteed to find the global minimum (provided that the learning rate is not too large and you have patience to wait long enough).

Cost function: Logistic regression for classification

- Next, to find the optimal $\mathbf{w}$ which minimizes the cost function $J(\mathbf{w})$, we can take the gradient of $J(\mathbf{w})$ and equate it to zero.
- The partial derivatives of this cost function with respect to weights w_j are given by

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i - \sigma(\mathbf{w}^T\mathbf{x}^{(i)})\right]x_j^{(i)}$$

The derivation of the above has been given in slide-11 of this lecture.

Cost function: Logistic regression for classification

- By stacking all partial derivatives in a vector, the gradient of cost function with respect to $\mathbf{w}$ can be written as

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \sum_{m=1}^{M} \left[\sigma(\mathbf{w}^T \mathbf{x}_m) - y_m\right] \mathbf{x}_m$$

- Note that $\sigma(\mathbf{w}^T \mathbf{x}_m) - y_m$ is the error between the output value and the prediction of the model.
- Recall that this gradient is the same form as the gradient of the sum-of-squares error function for the linear regression model.
- Next, we can use gradient descent or any sequential learning algorithms, including stochastic gradient descent for finding optimal $\mathbf{w}$ by using the knowledge of the gradient of the error function.
- For instance, for each of the weight vectors is updated using gradient descent as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla_{\mathbf{w}} J(\mathbf{w})$$

Summary: Logistic regression

In classification, we would like to minimize the cost function and thereby, find the optimal weights for our model.

$$g(\underline{w}, \underline{x}) = \sigma(\underline{w}^T \underline{x})$$

We can use gradient descent for this purpose.

Recall our cost function for classification is

$$J(\underline{w}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \ln\left[g(\underline{w}, \underline{x}^{(i)}) \right] + (1 - y^{(i)}) \ln\left[1 - g(\underline{w}, \underline{x}^{(i)}) \right]$$

$$\min_{\underline{w}} J(\underline{w}) \longrightarrow \hat{w} \leftarrow \text{optimal weights}$$

The partial derivatives of $J(\underline{w})$ with respect to $\underline{w}$ is given by,

$$\frac{\partial}{\partial w_j} \left(J(\underline{w}) \right) = \frac{1}{m} \sum_{i=1}^{m} \left[g(\underline{w}, \underline{x}^{(i)}) - y^{(i)} \right] x_j^{(i)}$$

where $g(\underline{w}, \underline{x}) = \sigma(\underline{w}^T \underline{x})$

Summary: Logistic regression

Note that, this derivative looks exactly the same as what we got for linear regression.

But for linear regression, $g(\underline{w}, \underline{x}) = \underline{w}^T \underline{x}$

For logistic regression, $g(\underline{w}, \underline{x}) = \sigma(\underline{w}^T \underline{x})$

So they are different in the formation.

Summary: Logistic regression

Gradient Descent Algorithm for Logistic Regression

repeat {

$$\omega_j := \omega_j - \alpha \frac{\partial}{\partial \omega_j} J(\underline{\omega})$$

(learning rate)

simultaneously update all ω_j

Here $$\frac{\partial}{\partial \omega_j} J(\underline{\omega}) = \frac{1}{m} \sum_{i=1}^{m} \left[g(\underline{\omega}, \underline{x}^{(i)}) - y^{(i)} \right] x_j^{(i)}$$

and $$g(\underline{\omega}, \underline{x}) = \sigma(\underline{\omega}^T \underline{x})$$

Derivatives of cross-entropy cost function

- Recall, the cost function for logistic regression can be written as

$$J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\ln(\sigma(\mathbf{w}^T\mathbf{x}^{(i)})) + (1-y^{(i)})\ln(1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)}))\right]$$

- Derivation of the partial derivative of the cost function for logistic regression can be presented as follows:

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial w_j} &= -\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}}{\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}\frac{\partial\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}{\partial w_j} - \frac{(1-y^{(i)})}{1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}\frac{\partial\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}{\partial w_j}\right] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\left[\frac{y^{(i)}}{\sigma(\mathbf{w}^T\mathbf{x}^{(i)})} - \frac{(1-y^{(i)})}{1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}\right]\frac{\partial\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}{\partial w_j}\end{aligned}$$

- Next, we evaluate $\frac{\partial\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}{\partial w_j}$ as follows:

$$\frac{\partial\sigma(\mathbf{w}^T\mathbf{x}^{(i)})}{\partial w_j} = \sigma(\mathbf{w}^T\mathbf{x}^{(i)})(1-\sigma(\mathbf{w}^T\mathbf{x}^{(i)}))\frac{\partial\mathbf{w}^T\mathbf{x}^{(i)}}{\partial w_j}$$

- Here, we have used the fact that $\frac{d}{dz}\sigma(z) = \sigma(z)(1-\sigma(z))$, where $\sigma(z) = 1/(1+\exp(-z))$ is the logistic sigmoid function.

Logistic regression: Derivatives of cross-entropy cost function

- We can evaluate $\frac{\partial \mathbf{w}^T \mathbf{x}^{(i)}}{\partial w_j} = x_j^{(i)}$ as $\mathbf{w} = [w_0, w_1, w_2, \cdots, w_j, \cdots, w_N]^T$.
- Then, the partial derivatives of the cost function can be derived as

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{m} \sum_{i=1}^{m} \left[\frac{y^{(i)}}{\sigma(\mathbf{w}^T \mathbf{x}^{(i)})} - \frac{(1 - y^{(i)})}{1 - \sigma(\mathbf{w}^T \mathbf{x}^{(i)})} \right] \left[\sigma(\mathbf{w}^T \mathbf{x}^{(i)})(1 - \sigma(\mathbf{w}^T \mathbf{x}^{(i)})) x_j^{(i)} \right]$$

- Upon simplifications, we can obtain the partial derivatives of the cost function as

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{m} \sum_{i=1}^{m} \left[y^i - \sigma(\mathbf{w}^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

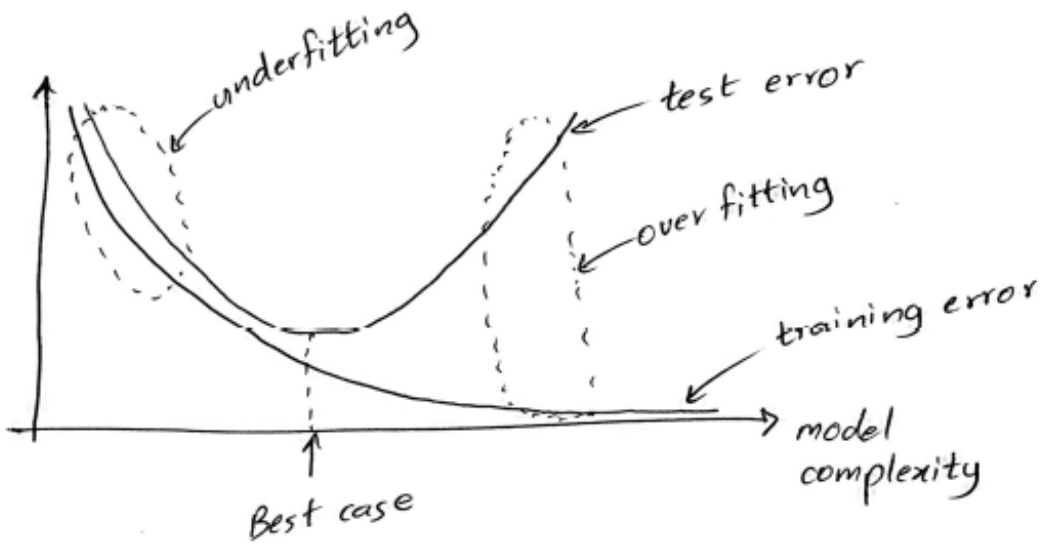
Logistic regression

Regularized Logistic Regression

If your model is too complicated, it may suffer from overfitting.

Logistic regression

Ex: $g(\underline{w}, \underline{x}) = \sigma\left(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 x_2 + w_4 x_1 x_2^2 + w_5 x_1^2 x_2^2 + w_6 x_1^2 + w_7 x_2^2\right)$



To avoid overfitting, we can use regularization.

Modify cost function by adding a regularizer term.

Logistic regression

Cost Function

$$J(\underline{w}) = -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)} \ln\left[g(\underline{w}, \underline{x}^{(i)})\right] + (1-y^{(i)}) \ln\left[1-g(\underline{w}, \underline{x}^{(i)})\right]\right)$$

$$+ \lambda \sum_{j=1}^{n} w_j^2$$

regularization term for $w_1, \ldots, w_n$. w_0 is not included.

λ is a positive constant and can be regarded as a hyper-parameter.

- Recall that in Ridge regression, we modify the original cost function as

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w},$$

where $J(\mathbf{w})$ is the original cost function and $\mathbf{w}^T\mathbf{w} = \sum_{j=1}^{n} w_j^2$.

Logistic regression

Gradient Descent with Regularization

repeat {

$$w_0 =: w_0 - \alpha \sum_{i=1}^{m} \left[g(\underline{w}, \underline{x}^{(i)}) - y^{(i)} \right] x_0^{(i)}$$

$$w_j =: w_j - \alpha \sum_{i=1}^{m} \left[g(\underline{w}, \underline{x}^{(i)}) - y^{(i)} \right] x_j^{(i)} - \lambda w_j$$

for $j = 1, \ldots, n$

where $g(\underline{w}, \underline{x}) = \sigma(\underline{w}^T \underline{x})$

Logistic regression

Multi-Class Classification

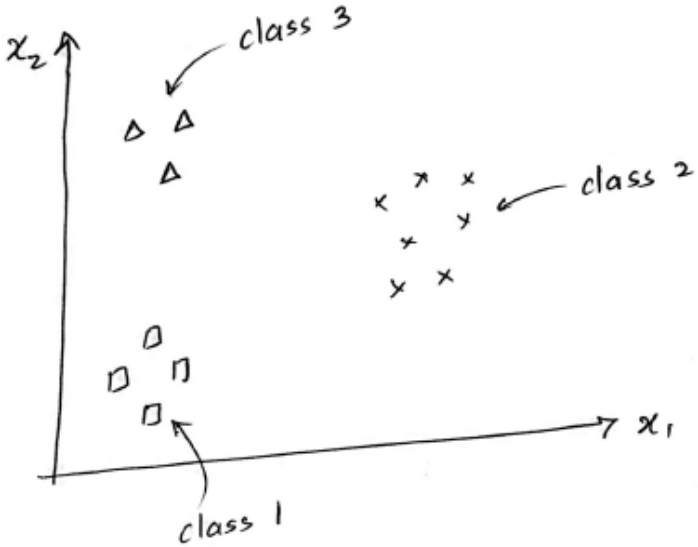
Example: E-mail filtering

work, social, online trade, fraud

$y=1$ $y=2$ $y=3$ $y=4$

$y \in \{1, 2, 3, 4\}$

Here we have more than two classes $\Rightarrow$ multi-class classification

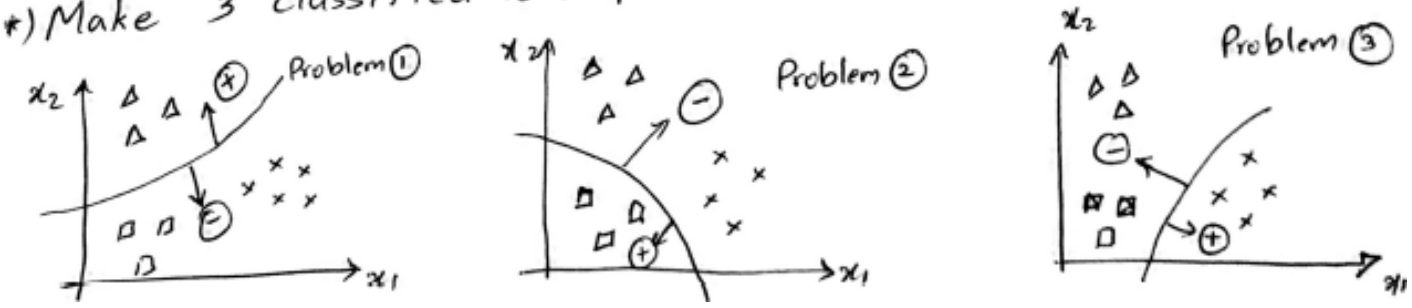


Logistic regression

One-Versus-All Classification.

Assume that we have 3 classes.

*) Make 3 classification problems.



Logistic regression

Here, $g^{(l)}(\underline{w}, \underline{x}) = P(Y = l \mid \underline{x}, \underline{w})$ for $l = 1, 2, 3$

logistic sigmoid function

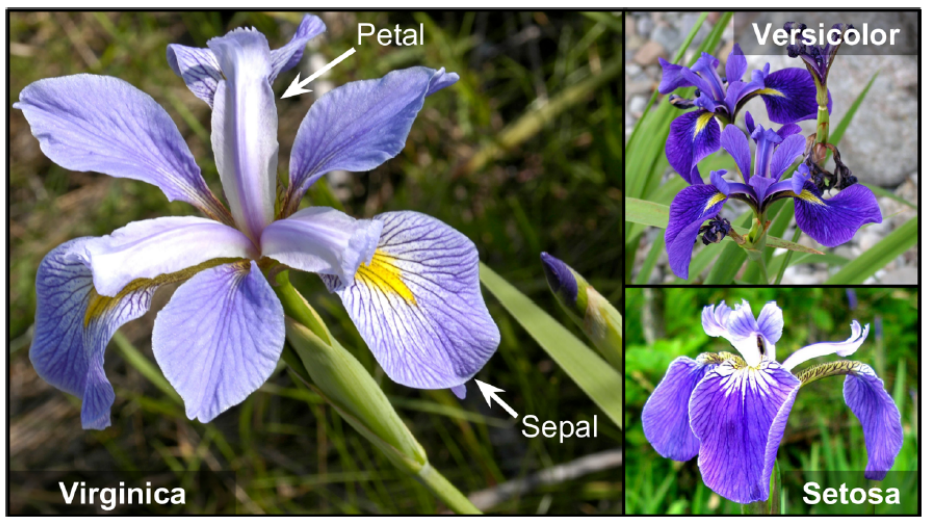
Train a logistic regression classification model $g^{(l)}(w, x)$ for each class l to predict the probability that $Y = l$.

On a new input $\underline{x}$, make a prediction, pick the class l that maximizes

$$\max_l \; g^{(l)}(\underline{w}, \underline{x})$$

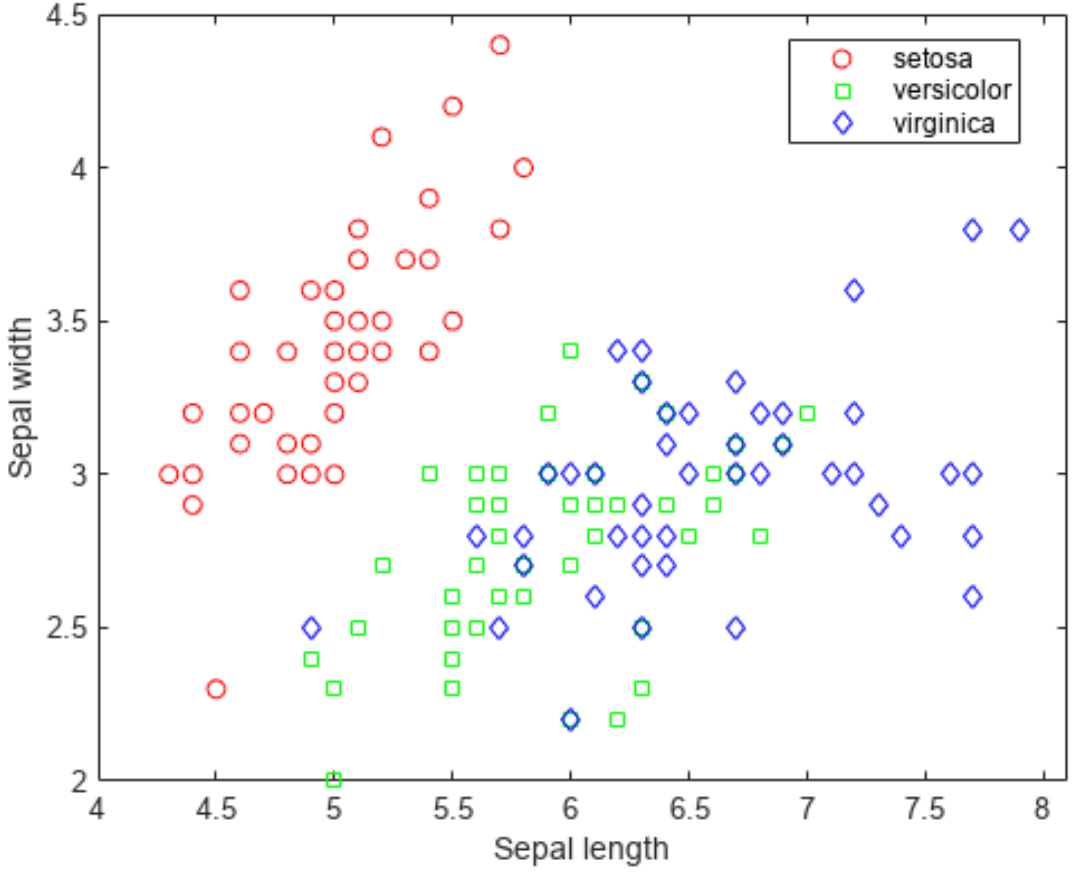
Example: Classification with logistic regression

- The well-known iris dataset can be used to illustrate logistic regression.
- Iris data-set contains the sepal and petal length and width of 150 iris flowers of three different species: Iris setosa, Iris versicolor, and Iris virginica



Example: Classification with logistic regression

- For example, we can visualize data objects by just looking at two features “Sepal length" and “Sepal width".



Example: Classification with logistic regression

- Our task is to build a classifier to detect the Iris virginica type based only on the petal width feature.
- First step is to load the Iris data-set to peek at available data by loading data with “sklearn library" for machine learning with Python:

```
from sklearn.datasets import load_iris
iris = load_iris(as_frame=True)
```

- To peek at the first three input feature entries of Iris data-set:

```
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

- The corresponding targets can be shown as

```
iris.target.head(3)
```

```
0    0
1    0
2    0
```

- Note that features/instances are not shuffled within this Iris data-set.

Example: Classification with logistic regression

- To find the target names:

```
iris.target_names
```

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

- Next step is to split the data and train a logistic regression model on the training set. This is because our task is to 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)
```

Example: Classification with MNIST data-set

- The MNIST dataset consists of a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau.
- Each image is labeled with the digit it represents.
- For instance, the digits from the MNIST data-set can be viewed as

- A code that fetches the MNIST dataset from OpenML.org can be as follows:

```
>>> softmax_reg.predict([[4, 2.5]])
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', as_frame=False)
```

Example: Classification with MNIST data-set

- Note that MNIST dataset contains images, and we can get the data as NumPy arrays.
- We can seek one of these arrays as

```
>>> X, y = mnist.data, mnist.target
>>> X
array([[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
...,
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.]])
>>> X.shape
(70000, 784)
>>> y
array(['5', '0', '4', ..., '4', '5', '6'], dtype=object)
>>> y.shape
(70000,)
```

Example: Classification with MNIST data-set

- In MNIST data-set, there are 70000 images, and each image has 784 features.
- This is because each image is $28 \times 28 = 784$ pixels, and each feature simply represents one pixel's intensity, from 0 (white) to 255 (black).
- To peek at one digit from the MNIST data-set, we can extract first an instance's feature vector, then reshape it to a 28×28 array, and finally display it using Matplotlib's imshow() function.

```
import matplotlib.pyplot as plt
def plot_digit(image_data):
image = image_data.reshape(28, 28)
plt.imshow(image, cmap="binary")
plt.axis("off")
some_digit = X[0]
plot_digit(some_digit)
plt.show()
>>> y[0]
'5'
```

Appendix: Cost function for Logistic regression with basis function expansions

- Let the data set be $\{\phi_n, y_n\}$, where $\phi_n = \phi(\mathbf{x}_n)$ and the output variables are $y_n \in \{1, 0\}$ for $n = 1, \cdots, N$.
- The posterior probability of $\mathcal{C}_1$ given ϕ_n is given by

$$p(\mathcal{C}_1|\boldsymbol{\phi}_n) = \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$$

- The likelihood function is given by

$$p(\mathbf{y}|\mathbf{w}) = \prod_{n=1}^{N} \left[p(\mathcal{C}_1|\boldsymbol{\phi}_n)\right]^{y_n} \left[p(\mathcal{C}_2|\boldsymbol{\phi}_n)\right]^{1-y_n}$$

- Since $p(\mathcal{C}_2|\phi_n) = 1 - p(\mathcal{C}_1|\phi_n)$, we have

$$p(\mathbf{y}|\mathbf{w}) = \prod_{n=1}^{N} \left[p(\mathcal{C}_1|\boldsymbol{\phi}_n)\right]^{y_n} \left[1 - p(\mathcal{C}_1|\boldsymbol{\phi}_n)\right]^{1-y_n}$$

Appendix: Cost function for Logistic regression with basis function expansions

- The error/cost function is defined to be the negative of the log-likelihood function as

$$J(\mathbf{w}) = -\ln(p(\mathbf{y}|\mathbf{w})) = -\ln\left[\prod_{n=1}^{N}\left[p(\mathcal{C}_1|\phi_n)\right]^{y_n}\left[1-p(\mathcal{C}_1|\phi_n)\right]^{1-y_n}\right]$$

- By expanding the logarithm, we have

$$J(\mathbf{w}) = -\sum_{n=1}^{N}\left[y_n \ln(p(\mathcal{C}_1|\phi_n)) + (1-y_n)\ln(1-p(\mathcal{C}_1|\phi_n))\right]$$

- In ML, this error function is known as the cross-entropy error function.
- Recall that $p(\mathcal{C}_1|\boldsymbol{\phi}_n) = \sigma(\mathbf{w}^T\boldsymbol{\phi}_n)$, and $p(\mathcal{C}_2|\boldsymbol{\phi}_n) = 1-\sigma(\mathbf{w}^T\boldsymbol{\phi}_n)$.
- Hence, the cost function is given by

$$J(\mathbf{w}) = -\sum_{n=1}^{N}\left[y_n \ln(\sigma(\mathbf{w}^T\boldsymbol{\phi}_n)) + (1-y_n)\ln(1-\sigma(\mathbf{w}^T\boldsymbol{\phi}_n))\right]$$