4 Regression-Based Classification

Certain algorithms that were first developed for regression can be adapted for classification, and some classifiers can be modified to predict continuous values. Converting a linear regressor into a classifier by adding a logistic link, for instance, retains the original coefficient vector, which makes it easy to see how each input feature influences the decision. As these dual-purpose models expose many of the hyper-parameters found in their regression versions, they often provide more opportunities for fine-tuning and clearer interpretability than algorithms designed purely for classification.

4.1 Logistic Regression

Logistic Regression, sometimes called logit regression, models the probability that a sample belongs to a specific class, such as the chance that an email is spam. When this probability exceeds 50%, the model predicts the positive class (label “1”); otherwise, it predicts the negative class (label “0”). It therefore acts as a binary classifier.

The predicted probability is

$$\hat{p} = h_\theta(X) = \sigma(\theta^{\mathrm{T}} \cdot X). \tag{4.1}$$

Here $\hat{p}$ is the estimated probability, and $\sigma(\cdot)$ is the sigmoid function, an S-shaped curve that maps any real number to the interval $(0,1)$. The logistic function is defined in Equation 4.2 and illustrated in Figure 4.1. As before, $h_\theta(X)$ denotes the hypothesis that the input matrix X, augmented with a bias term, belongs to the positive class under the parameters θ.

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{4.2}$$

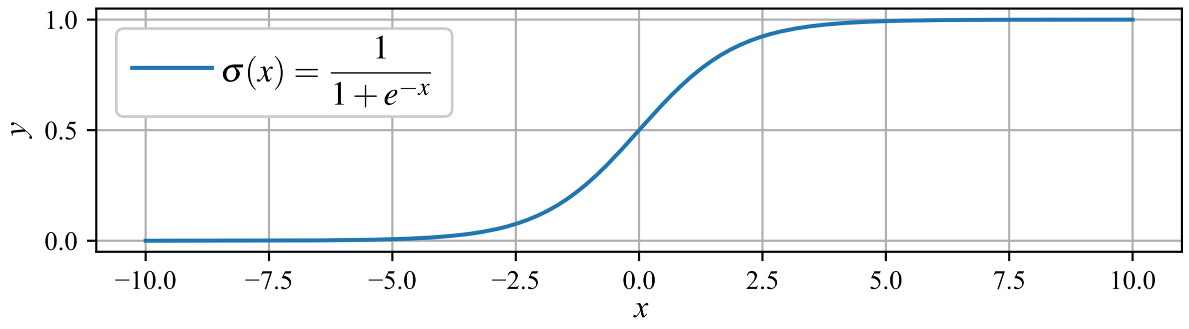

Figure 4.1: Sigmoid function that maps any real-valued input x to a value between 0 and 1.

Once the probability $\hat{p} = h_\theta(X)$ that an instance X belongs to the positive class has been estimated using Logistic Regression, the prediction ($\hat{y}$) can be made. $\hat{y}$ is calculated as

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5, \\ 1 & \text{if } \hat{p} \geq 0.5. \end{cases} \tag{4.3}$$

Note that $\sigma(x) < 0.5$ when $x < 0$, and $\sigma(x) \geq 0.5$ when $x \geq 0$. Therefore, the Logistic Regression model predicts 0 if $\theta^{\mathrm{T}} \cdot X$ is negative, and 0 if it is positive.

Now that we understand how logistic regression assigns probabilities and produces predictions, let's walk through a concise example that shows its training procedure and the associated cost function. Training seeks parameter values θ that give high predicted probabilities to positive examples ($y = 1$) and low probabilities to negative ones ($y = 0$). This aim is captured by the cost defined in equation 4.4, which evaluates a single training sample x. To achieve this, we require a cost function, such as

$$C(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1, \\ -\log(1-\hat{p}) & \text{if } y = 0. \end{cases} \tag{4.4}$$

The considered cost function is plotted in figure 4.2.

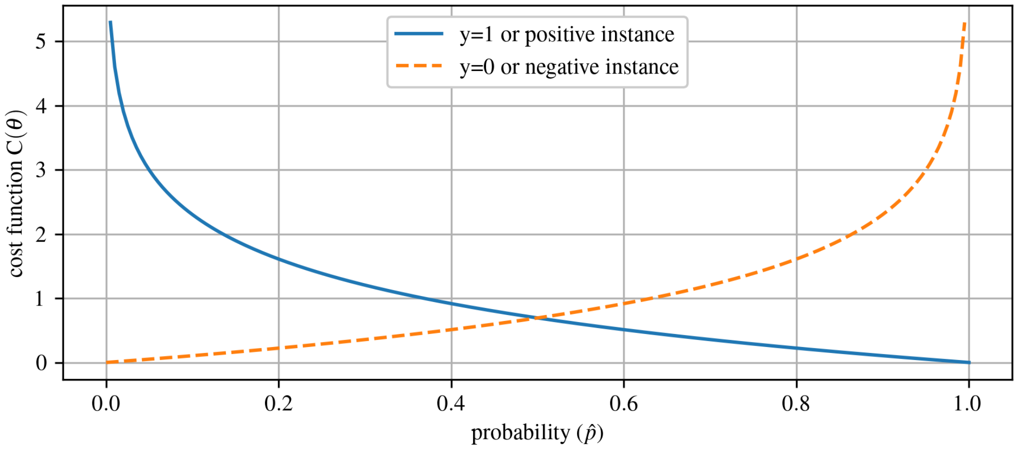

Figure 4.2: Cost function behavior for classification that heavily penalizes incorrect predictions.

The cost function in equation 4.4 behaves intuitively: $-\log(\hat{p})$ increases sharply as $\hat{p} \to 0$, so the loss is large when the model assigns a probability near0 to a positive example. It likewise produces a high loss when the model predicts a probability close to1 for a negative example. Conversely, because $-\log(\hat{p}) \to 0$ as $\hat{p} \to 1$, the loss becomes negligible when the predicted probability is near1 for a positive instance or near0 for a negative one, aligning with our expectations.

The overall cost, denoted $J(\theta)$, is the mean loss across all m training examples. This metric is commonly called the log-loss and written as

$$J(\theta);=;-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right);+;\left(1-y^{(i)}\right)\log\left(1-\hat{p}^{(i)}\right)\right]. \tag{4.5}$$

Because there is no closed-form solution for the parameters θ (embedded within $\hat{p}$), the minimum must be found with an iterative optimizer. The cost surface is convex, so gradient descent or another suitable algorithm will reach the global minimum provided the learning rate is reasonable and enough iterations are allowed. The gradient of the cost with respect to the j^{th} parameter θ_j is given as

$$\frac{\partial J}{\partial \theta_j};=;\frac{1}{m}\sum_{i=1}^{m}\left[\sigma\left(\theta^{\mathrm{T}}X^{(i)}\right)-y^{(i)}\right],x_j^{(i)}. \tag{4.6}$$

This equation resembles the partial derivative used in gradient descent. For every training example, it finds the prediction error, multiplies it by the j^{th} feature value, and then averages these products over all m samples. With the resulting gradient vector of partial derivatives, you can update the parameters using the Batch Gradient Descent algorithm, completing the training of a Logistic Regression model. Stochastic Gradient Descent performs the update after each single example, while Mini-batch Gradient Descent updates the parameters after processing each mini-batch.

Review 4.1 Iris Flower Dataset

The Iris dataset, collected by botanist Edgar Anderson in 1935, lists sepal length, sepal width, petal length, and petal width for 150 flowers drawn equally from three species: Iris-Setosa, Iris-Versicolor, and Iris-Virginic (refer to Figure 4.3)a. Statistician Ronald Fisher analysed these measurements in a 1936 study on linear discriminant analysis, and the data have since become a standard benchmark in statistics and machine learning because they are small, clean, and perfectly balanced, making them ideal for testing classification algorithms and visualisation techniques.

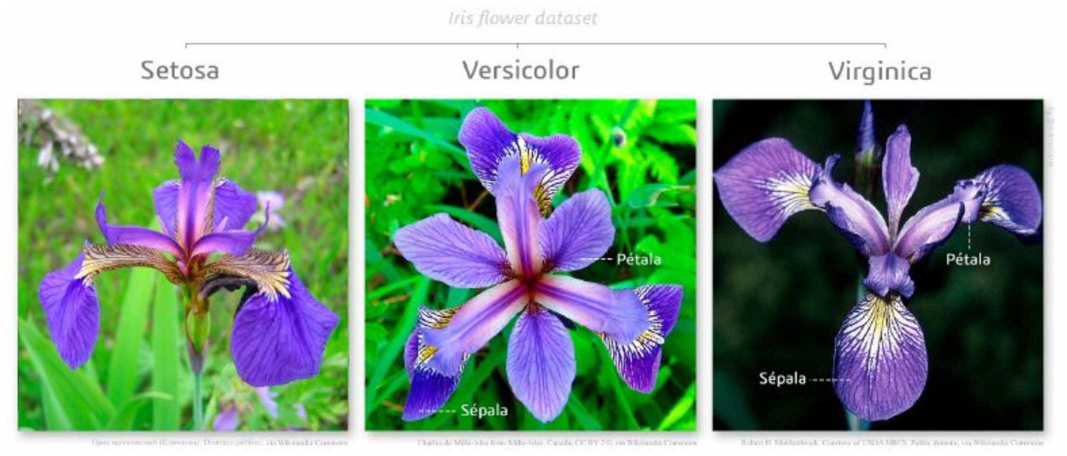

Figure 4.3: Flowers representing three species of iris plants [a]

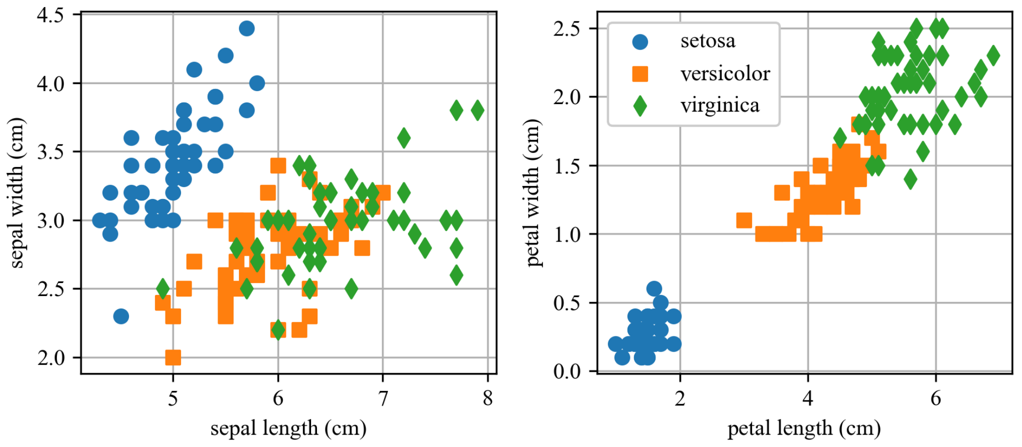

Figure 4.4: Iris dataset scatterplot showing sepal length vs. sepal width (left) and petal length vs. petal width (right).

[a]Diego Mariano, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

Example 4.1 Iris Dataset Exploration
This example introduces the Iris dataset (Figure 4.3), originally compiled by Ronald Fisher. It demonstrates how to load the dataset, access feature and label information, and visualize the relationships between sepal and petal measurements across the three iris species.

4.1.1 1-D Decision Boundaries

The petal width of Iris-virginica flowers usually lies between 1.4 cm and 2.5 cm, whereas the other two species range from 0.1 cm to 1.8 cm. Because these intervals overlap, the classifier's certainty changes across the scale. Above roughly 2 cm, the model is confident that any given sample is Iris-virginica. Moreover, below 1 cm it is equally sure the flower is not Iris-virginica (This means the model redicts high probability for the class "Not Iris-Virginica"). In between the limits of the classes, the model is unsure. When you call `predict()` instead of `predict_proba()`, it outputs the most likely class, creating a decision boundary around 1.6 cm where both class probabilities reach 50 percent. Thus, flowers with petal widths greater than 1.6 cm lead to a prediction of Iris-virginica, while flowers with smaller petal widths yield the opposite prediction even if confidence is low.

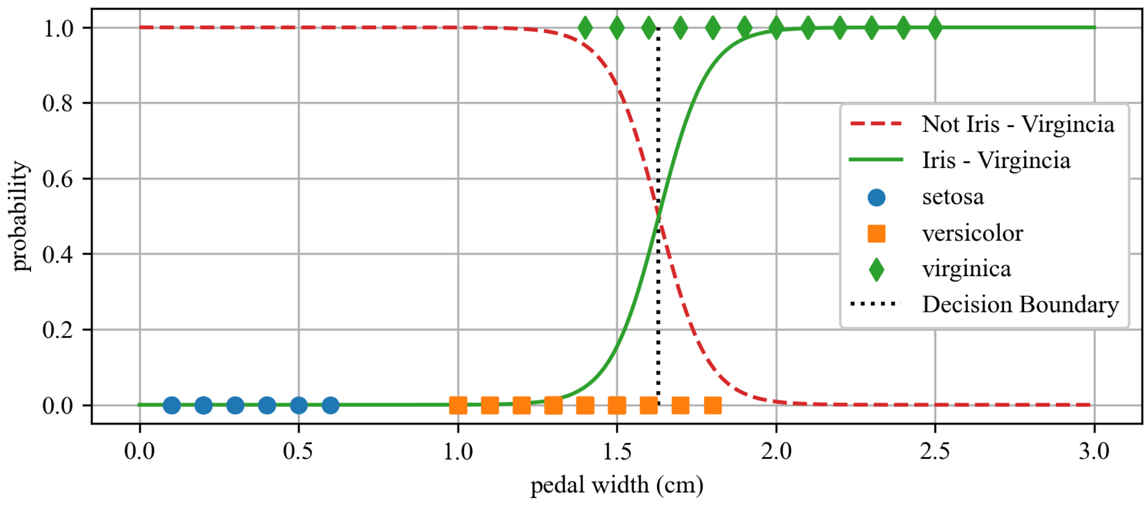

Figure 4.5: Decision boundary for the flowers of three Iris plant species with C set to $C = 10^{10}$.

Example 4.2 1D Logistic Regression
This example builds a logistic regression model to classify Iris-Virginica based on a single feature: petal width. It visualizes class probabilities and shows the decision boundary at the 50% threshold.

4.1.2 2-D Decision Boundaries

Figure 4.6 plots petal width against petal length for the Iris samples. After training, the logistic regression model assigns each point the probability that the flower is *Irisvirginica*. The dashed line marks where this probability equals50 %, forming the decision boundary, which is linear. The parallel contour lines show equal-probability levels from 10% to 90%. Points be beyond the top-right line have a greater than 90% probability of being classified as Iris-Virginica by the model.

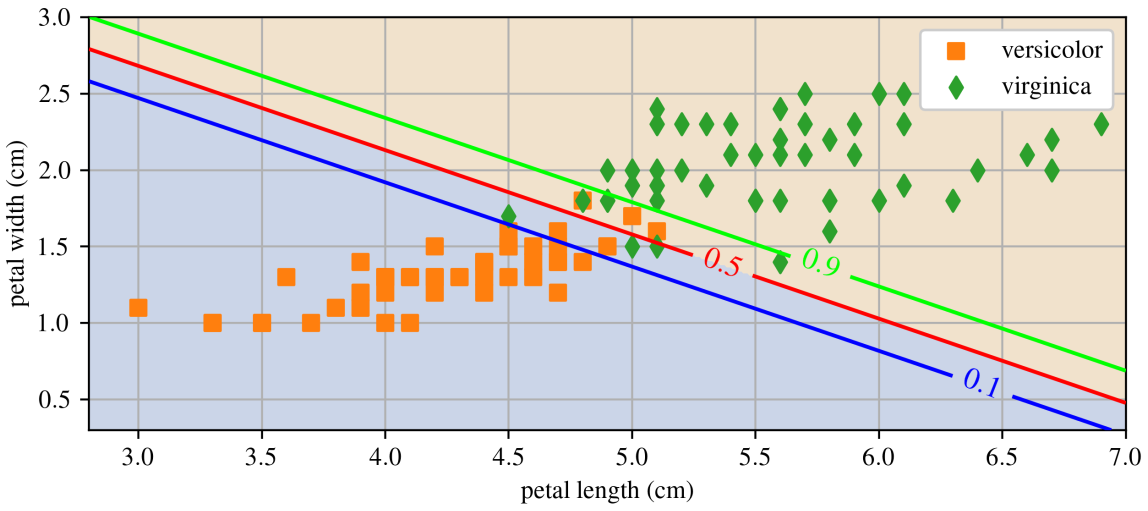

Figure 4.6: A 2D decision boundary for the Iris dataset.

Example 4.3 2D Decision Boundary
This example uses logistic regression to classify the Iris-Virginica species based on petal length and width. A 2D decision boundary is visualized in the petal feature space, with prediction regions and probability contours illustrating classifier confidence.

4.2 Softmax Regression

Logistic Regression can be generalised to handle many classes in a single model, so there is no need to train and merge several binary classifiers. This multiclass version is called Softmax Regression, or Multinomial Logistic Regression. Softmax Regression provides a straightforward way to perform regression-based classification when more than two classes are present. For an input vector $\mathbf{x}$, the model first computes a score $s_k(\mathbf{x})$ for every class k, then turns these scores into class probabilities with the softmax (normalised exponential) function. The score is obtained exactly as in linear regression:

$$s_k(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta}^{(k)}. \tag{4.7}$$

Here each class has its own parameter vector $\theta^{(k)}$, and the collection of these vectors is usually stored as the rows of a parameter matrix Θ.

After the model has calculated a score for each class given an input $\mathbf{x}$, it converts these scores to probabilities with the softmax function. For class k the predicted probability is

$$\hat{p}_k = \sigma\big(s(\mathbf{x})\big)_k = \frac{\exp\big(s_k(\mathbf{x})\big)}{\sum_{j=1}^{K}\exp\big(s_j(\mathbf{x})\big)}. \tag{4.8}$$

In this expression, $\mathbf{s}(\mathbf{x})$ is the vector of class scores for the input, $\sigma\big(\mathbf{s}(\mathbf{x})\big)_k$ is the probability that $\mathbf{x}$ belongs to class k, and K is the total number of classes. In short, equation 4.8 exponentiates each score and then normalises the results by dividing by the sum of all exponentials so the probabilities sum to one.

Like logistic regression, a Softmax Regression model assigns an input to the class whose predicted probability is largest, which coincides with the class that has the highest score:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\,\sigma\big(\mathbf{s}(\mathbf{x})\big)_k = \underset{k}{\operatorname{argmax}}\,s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\,\big((\theta^{(k)})^{\mathrm{T}}\cdot\mathbf{x}\big). \tag{4.9}$$

NOTE

The argmax operator returns the argument that maximises a function. In this setting, it yields the index k for which the estimated probability $\sigma\big(s(\mathbf{x})\big)_k$ attains its largest value.

With probability estimation and prediction established, we next consider training. The task is to learn parameters that place a large probability on the correct class and correspondingly small probabilities on all others. This objective is met by minimising the cost in equation 4.10, known as the cross entropy, which heavily penalises the model when it assigns a low probability to the true label. Cross entropy is widely used to measure how well predicted class probabilities agree with the actual classes. The cost function is

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)}). \tag{4.10}$$

NOTE

In this expression, $y_k^{(i)} = 1$ when the i^{th} sample's true label is class k, and $y_k^{(i)} = 0$ otherwise.

When considering only two classes ($K = 2$), it's important to highlight that this cost function aligns with the Logistic Regression's cost function, commonly referred to as log loss (refer to Equation 4.5).

The gradient of the cross-entropy cost with respect to $\theta^{(k)}$ is

$$\nabla_{\theta^{(k)}} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{p}_k^{(i)} - y_k^{(i)} X^{(i)}). \tag{4.11}$$

By computing this vector for each class, you obtain the full gradient, which can then be fed to Gradient Descent or another optimizer to find the parameter matrix Θ that minimises the cost.

Applying Softmax Regression to the three-class iris problem in Scikit-Learn is straightforward. The `LogisticRegression` estimator normally uses one-versus-all when more than two classes are present, but setting `multi_class=‘‘multinomial’’` activates true Softmax learning. Choose a solver that supports this option, such as `lbfgs` (see the library documentation). The model applies ℓ_2 regularisation by default, governed by the hyperparameter C. After training, a flower with 5 cm long and 2 cm wide petals is classified as Iris-Virginica with probability 94.2%, while the probability of Iris versicolor is 5.8%.

NOTE

The Softmax Regression classifier is multiclass, not multioutput. As such, Softmax Regression can only predict one class at a time, so it works for problems where each input belongs to exactly one category-like classifying an email as spam, promotions, or updates. It can't be used for cases where multiple labels may apply, such as tagging a news article with topics like politics, economics, and technology all at once.

Figure 4.7 displays the decision regions, shaded with background colours to mark each class. The borders separating any two classes are straight lines. The plot also includes contour curves for the predicted probability of the Iris-Versicolor class. At the point where all three borders meet, every class receives a probability of 33%, so the selected class can have a confidence below 50%.

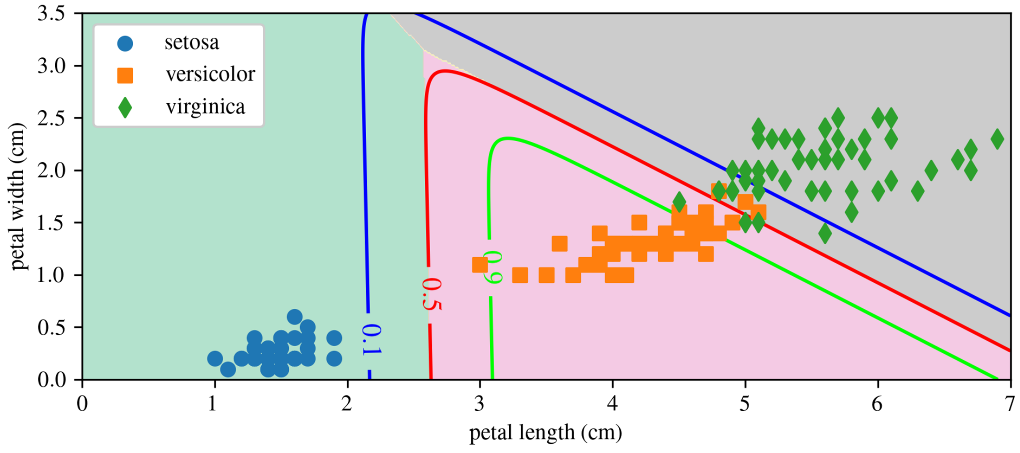

Figure 4.7: Softmax classification for the three iris plant species.

Example 4.4 Softmax Classification
This example applies softmax regression to classify all three iris species using petal length and width. It builds a multinomial logistic regression model and visualizes the decision boundaries along with confidence contours over the 2D petal feature space.

4.3 Examples

Example 4.1

```python
"""
Example 4.1 Introduction to the IRIS data set
@author: austin_downey
"""

import IPython as IP
IP.get_ipython().run_line_magic('reset', '-sf')

import matplotlib.pyplot as plt
import sklearn as sk

cc = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.close('all')


#%% Load your data

# We will use the Iris data set.  This dataset was created by biologist Ronald
# Fisher in his 1936 paper "The use of multiple measurements in taxonomic
# problems" as an example of linear discriminant analysis

iris = sk.datasets.load_iris()

# for simplicity, extract some of the data sets
X = iris['data'] # this contains the length of the pedals and sepals
Y = iris['target'] # contains what type of flower it is
Y_names = iris['target_names'] # contains the name that aligns with the type of the
flower
feature_names = iris['feature_names'] # the names of the features

# plot the Sepal data
plt.figure(figsize=(6.5,3))
plt.subplot(121)
plt.grid(True)
plt.scatter(X[Y==0,0],X[Y==0,1],marker='o')
plt.scatter(X[Y==1,0],X[Y==1,1],marker='s')
plt.scatter(X[Y==2,0],X[Y==2,1],marker='d')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])


plt.subplot(122)
plt.grid(True)
plt.scatter(X[Y==0,2],X[Y==0,3],marker='o',label=Y_names[0])
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',label=Y_names[1])
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',label=Y_names[2])
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend(framealpha=1)
plt.tight_layout()

```

Example 4.2

```python
"""
Example 4.2 1D Decision boundary for the Iris dataset
@author: austin_downey
"""

import IPython as IP
IP.get_ipython().run_line_magic('reset', '-sf')

import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk


cc = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.close('all')


#%% Load your data

# We will use the Iris data set.  This dataset was created by biologist Ronald
# Fisher in his 1936 paper "The use of multiple measurements in taxonomic
# problems" as an example of linear discriminant analysis

iris = sk.datasets.load_iris()

# for simplicity, extract some of the data sets
X = iris['data'] # this contains the length of the pedals and sepals
Y = iris['target'] # contains what type of flower it is
Y_names = iris['target_names'] # contains the name that aligns with the type of the
flower
feature_names = iris['feature_names'] # the names of the features

# plot the Sepal data
plt.figure(figsize=(6.5,3))
plt.subplot(121)
plt.grid(True)
plt.scatter(X[Y==0,0],X[Y==0,1],marker='o')
plt.scatter(X[Y==1,0],X[Y==1,1],marker='s')
plt.scatter(X[Y==2,0],X[Y==2,1],marker='d')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])


plt.subplot(122)
plt.grid(True)
plt.scatter(X[Y==0,2],X[Y==0,3],marker='o',label=Y_names[0])
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',label=Y_names[1])
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',label=Y_names[2])
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend(framealpha=1)
plt.tight_layout()


#%% Train a Logistic Regression model

# define the features (X) and the output (Y)
X_pedal = iris["data"][:, 3:] # consider just the petal width
y_pedal = iris["target"] == 2 # 1 if Iris-Virginica, else 0

# Build the logistic Regression model and train it.
log_reg = sk.linear_model.LogisticRegression( C=1)
log_reg.fit(X_pedal, y_pedal)
# Note: The hyper-parameter controlling the regularization strength of a
# Scikit-Learn LogisticRegression model is not alpha (as in other linear models),
# but its  inverse: C. The higher the value of C, the less the model is regularized.

# Build a range of the feature (X) to predict over. Here we just consider pedal width.
X_new = np.linspace(0, 3, 1000)
X_new = np.expand_dims(X_new, axis=1)

# Use the Logistic Regression Model to predict the pedal type based on pedal width
y_proba = log_reg.predict_proba(X_new)

# plot the probability plots
plt.figure(figsize=(6.5,3))
plt.grid(True)

# plot the data used for training, set at 0 and 1
plt.scatter(X[Y==0,3],np.zeros(50),marker='o',label=Y_names[0],zorder=10)
plt.scatter(X[Y==1,3],np.zeros(50),marker='s',label=Y_names[1],zorder=10)
plt.scatter(X[Y==2,3],np.ones(50),marker='d',label=Y_names[2],zorder=10)

# plot the probability
plt.plot(X_new,y_proba[:,0], '--',color=cc[3],label='Not Iris - Virgincia')
plt.plot(X_new,y_proba[:,1],color=cc[2], label='Iris - Virgincia')

# find and plot the 50% decision boundary
x_at_50 = X_new[np.argmin(np.abs(y_proba[:,0] - 0.5))]
plt.vlines(x_at_50,0,1,color='k',linestyles=':',label='Decision Boundary')

plt.xlabel('pedal width (cm)')
plt.ylabel('probability')
plt.legend(framealpha=1)
plt.tight_layout()

# make predictions on the model trained on the data.
log_reg.predict([[1.7]])
```

Example 4.3

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Example 4.3 2D Decision boundary for the Iris dataset

Developed for Machine Learning for Mechanical Engineers at the University of
South Carolina

@author: austin_downey
"""

import IPython as IP
IP.get_ipython().run_line_magic('reset', '-sf')


import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk


cc = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.close('all')


#%% Load your data

# We will use the Iris data set.  This dataset was created by biologist Ronald
# Fisher in his 1936 paper "The use of multiple measurements in taxonomic
# problems" as an example of linear discriminant analysis

iris = sk.datasets.load_iris()

# for simplicity, extract some of the data sets
X = iris['data'] # this contains the length of the petals and sepals
Y = iris['target'] # contains what type of flower it is
Y_names = iris['target_names'] # contains the name that aligns with the type of the
flower
feature_names = iris['feature_names'] # the names of the features

# plot the Sepal data
plt.figure(figsize=(6.5,3))
plt.subplot(121)
plt.grid(True)
plt.scatter(X[Y==0,0],X[Y==0,1],marker='o',zorder=10)
plt.scatter(X[Y==1,0],X[Y==1,1],marker='s',zorder=10)
plt.scatter(X[Y==2,0],X[Y==2,1],marker='d',zorder=10)
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])


plt.subplot(122)
plt.grid(True)
plt.scatter(X[Y==0,2],X[Y==0,3],marker='o',label=Y_names[0],zorder=10)
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',label=Y_names[1],zorder=10)
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',label=Y_names[2],zorder=10)
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend(framealpha=1)
plt.tight_layout()


#%% plot the Linear decision boundary in 2D "Petal" space

# build the training and target set.
X_train = X[:, (2, 3)]  # petal length, petal width
y_train = Y == 2

```

```
# build the Logistic Regression model
log_reg = sk.linear_model.LogisticRegression(C=10**10)
# Note: The hyper-parameter controlling the regularization strength of a Scikit-Learn
# LogisticRegression model is not alpha (as in other linear models), but its
# inverse: C. The higher the value of C, the less the model is regularized.

# train the Logistic Regression model
log_reg.fit(X_train, y_train)

# build the x values for the predictions over the entire "petal space"
x_grid, y_grid = np.meshgrid(
        np.linspace(2.8, 7, 500),
        np.linspace(0.3, 3, 200),
    )
X_new = np.vstack((x_grid.reshape(-1), y_grid.reshape(-1))).T # build a vector format of the mesh grid

# predict on the vectorized format
y_predict = log_reg.predict(X_new)
y_proba = log_reg.predict_proba(X_new)

# convert back to meshgrid shape for plotting
zz_predict = y_predict.reshape(x_grid.shape)
zz_proba = y_proba[:, 1].reshape(x_grid.shape)

# plot the 2D "petal space"
plt.figure(figsize=(6.5,3))
plt.grid(True)
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',color=cc[1],label=Y_names[1],zorder=10)
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',color=cc[2],label=Y_names[2],zorder=10)
plt.contourf(x_grid, y_grid, zz_predict, cmap='Pastel2')
contour = plt.contour(x_grid, y_grid, zz_proba, [0.100,0.5,0.900],cmap=plt.cm.brg)
plt.clabel(contour, inline=1) # add the labels to the plot
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend()
plt.tight_layout()



```

Example 4.4

```python
"""
Example 4.4 Softmax decision boundary for the Iris dataset
@author: austin_downey
"""

import IPython as IP
IP.get_ipython().magic('reset -sf')

import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk


cc = plt.rcParams['axes.prop_cycle'].by_key()['color']
plt.close('all')


#%% Load your data

# We will use the Iris data set.  This dataset was created by biologist Ronald
# Fisher in his 1936 paper "The use of multiple measurements in taxonomic
# problems" as an example of linear discriminant analysis
iris = sk.datasets.load_iris()

# for simplicity, extract some of the data sets
X = iris['data'] # this contains the length of the petals and sepals
Y = iris['target'] # contains what type of flower it is
Y_names = iris['target_names'] # contains the name that aligns with the type of the
flower
feature_names = iris['feature_names'] # the names of the features

# plot the Sepal data
plt.figure(figsize=(6.5,3))
plt.subplot(121)
plt.grid(True)
plt.scatter(X[Y==0,0],X[Y==0,1],marker='o',zorder=10)
plt.scatter(X[Y==1,0],X[Y==1,1],marker='s',zorder=10)
plt.scatter(X[Y==2,0],X[Y==2,1],marker='d',zorder=10)
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])

plt.subplot(122)
plt.grid(True)
plt.scatter(X[Y==0,2],X[Y==0,3],marker='o',label=Y_names[0],zorder=10)
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',label=Y_names[1],zorder=10)
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',label=Y_names[2],zorder=10)
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend(framealpha=1)
plt.tight_layout()


#%% Softmax Regression

# build the training and target set.
X_train = X[:, (2, 3)]  # petal length, petal width
y_train = Y

# build and train the softmax model
softmax_reg = sk.linear_model.LogisticRegression(multi_class="multinomial",
        solver="lbfgs", C=10)
softmax_reg.fit(X_train, y_train)

# build the x values for the predictions over the entire "petal space"
x_grid, y_grid = np.meshgrid(
        np.linspace(0, 7, 500),
        np.linspace(0, 4, 200),
    )
X_new = np.vstack((x_grid.reshape(-1), y_grid.reshape(-1))).T # build a vector format of the mesh grid

# predict on the vectorized format
y_predict = softmax_reg.predict(X_new)
y_proba = softmax_reg.predict_proba(X_new)


# convert back to meshgrid shape for plotting
zz_predict = y_predict.reshape(x_grid.shape)
zz_proba = y_proba[:, 1].reshape(x_grid.shape) # the selected column selects the probability that the data falls within this class.

# plot the 2D "petal space"
plt.figure(figsize=(6.5, 4))
plt.scatter(X[Y==0,2],X[Y==0,3],marker='o',label=Y_names[0],zorder=10)
plt.scatter(X[Y==1,2],X[Y==1,3],marker='s',label=Y_names[1],zorder=10)
plt.scatter(X[Y==2,2],X[Y==2,3],marker='d',label=Y_names[2],zorder=10)
plt.contourf(x_grid, y_grid, zz_predict, cmap='Pastel2')
contour = plt.contour(x_grid, y_grid, zz_proba, [0.100,0.5,0.900], cmap=plt.cm.brg)
plt.clabel(contour, inline=1)
plt.xlabel(feature_names[2])
plt.ylabel(feature_names[3])
plt.legend()
plt.tight_layout()
```