Module 13: Logistic Regression

Quick Reference Guide

Learning Outcomes:

1. Analyze the characteristics and advantages of logistic regression.
2. Use the scikit-learn framework to solve a basic logistic regression problem.
3. Utilize logistic regression for business decision-making.
4. Practice illustrating logistic regression as an optimization problem to determine the coefficients to minimize cross-entropy.
5. Implement strategies to visualize decision boundaries for two variables.
6. Evaluate the cross-entropy for a range of parameters.
7. Apply logistic regression to generate multiclass outputs.
8. Use L1 regularization to select features.
9. Assess multiple models using performance metrics.

Motivation

Overall, k-nearest neighbors is a very intuitive and effective approach to data classification. But, since it requires the use of an entire dataset, classifying a new item with this method can be quite expensive in terms of computation and memory. Other classification models such as linear regression involve only a small number of coefficients to decide the value of a new point. Logistic regression is considered the simplest of these models.

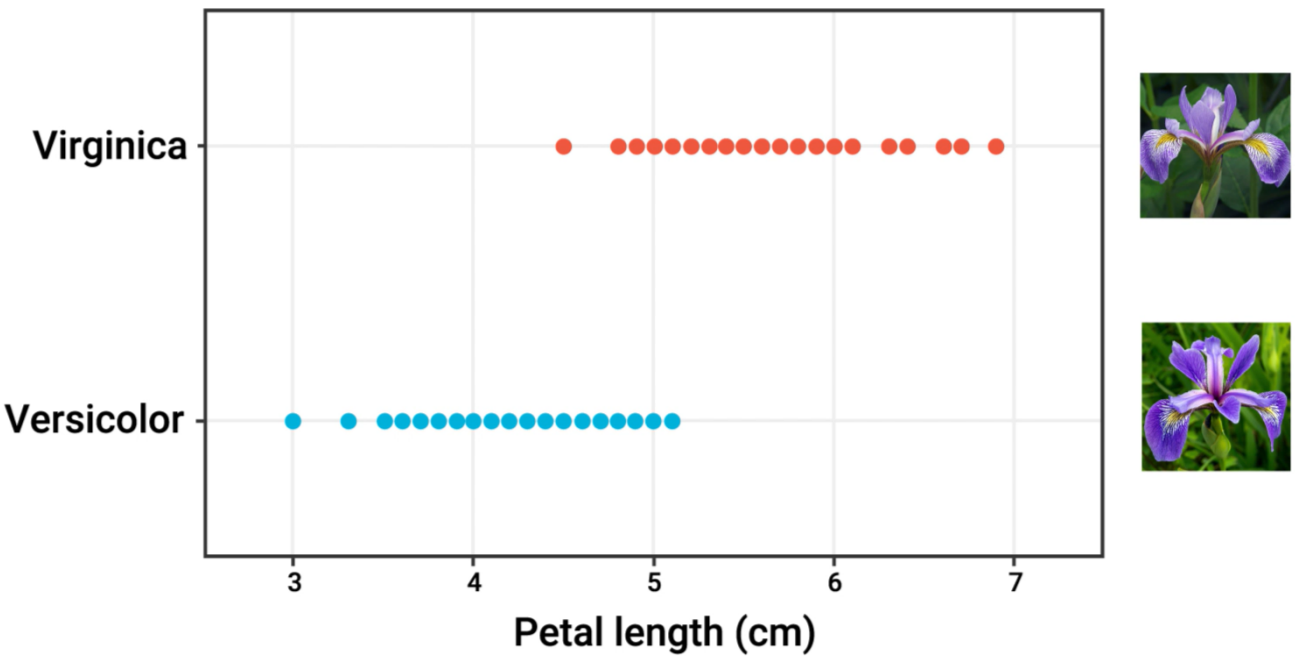
Logistic regression

In logistic regression, the model's output will be a continuous-valued probability that the input belongs to a certain class. Consider the Iris dataset, displayed in a pandas DataFrame, to demonstrate this technique.

	Sepal length (cm)	Sepal width (cm)	Petal length (cm)	Petal width (cm)	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2..3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

There are five columns: The sepal length, sepal width, petal length, petal width. And finally, the species of iris as an integer: 0, 1, or 2. Here, 0 stands for the species setosa, 1 is versicolor, and 2 is a virginica-type iris.

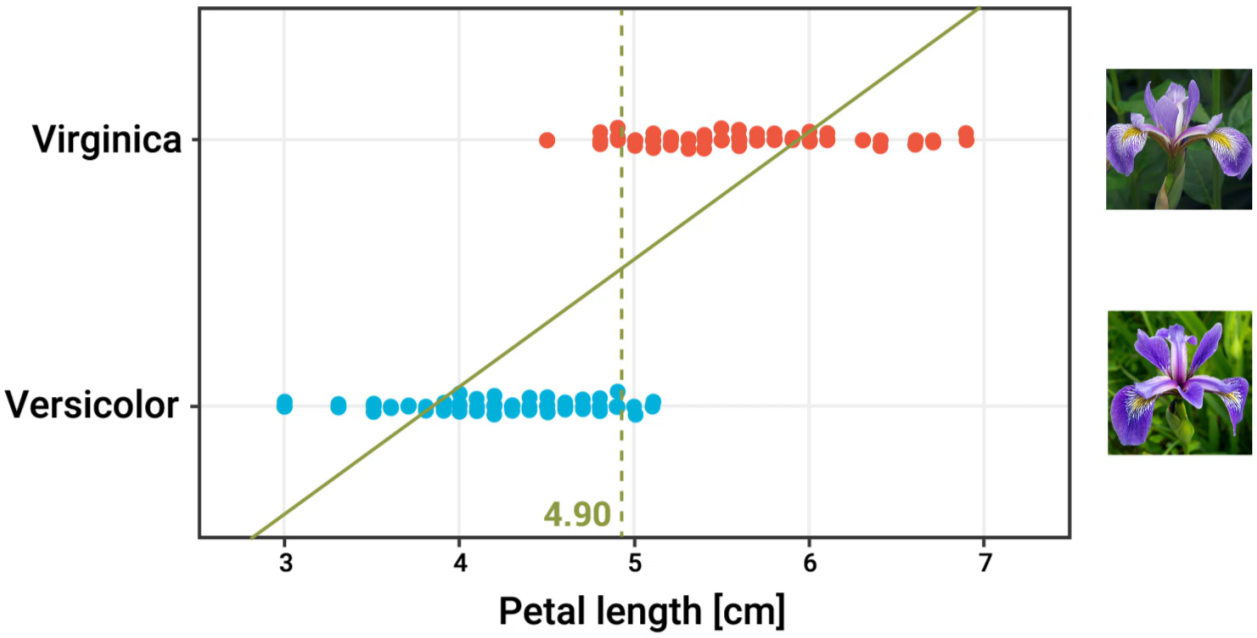
The goal is to build a model that can classify iris flowers into setosa, versicolor, and virginica—given the size of their sepals and petals.



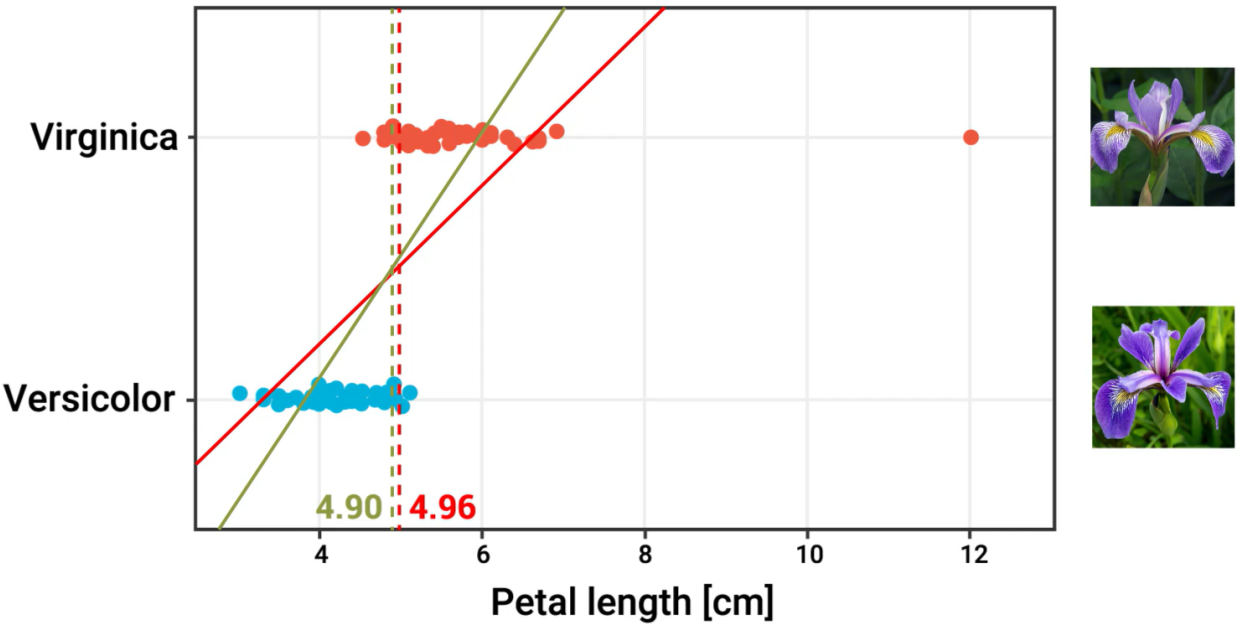
This plot illustrates that virginica petals are typically larger than versicolor. To distinguish between virginica and versicolor based only on their petal lengths requires a decision threshold: All flowers with petals larger than that number are classified as virginica and all flowers with petals smaller than that number are classified as versicolor.

Computing a threshold

To establish a methodology for computing this threshold, you can use **linear regression**. That is, find a straight line that minimizes the sum of the squares of the vertical distances from the line to the data. Where that line crosses the midpoint between versicolor and virginica, represents the placing of the threshold. In this case, the threshold can be defined as 4.9cm as shown.



Introducing an outlier flower to the dataset, and repeating the computation, yields the following plot.

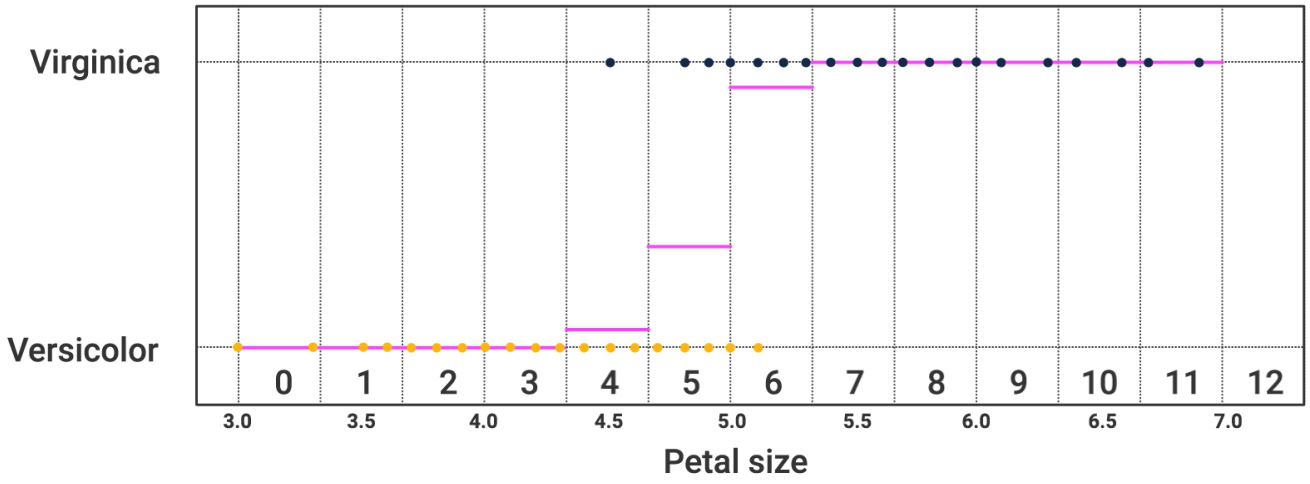


While the new threshold does not differ significantly from the first, it is important to notice the effect it has on the regression line: From the green line without the outlier to the red line with the outlier. In other words, **the linear regression model is sensitive to outliers**.

Logistic regression presents an alternative methodology for computing cutoff thresholds. This method not only avoids the problem of sensitivity but also has another significant advantage, which is that it produces class membership probabilities—instead of bare decision thresholds. These probabilities will be necessary to build classification models for more than two categories of objects.

One Feature, Two Classes

Imagine that you have to guess whether an unseen flower is a versicolor or virginica based only on its petal length. One strategy to achieve this involves going back to the Iris dataset, finding all flowers with similar petal lengths, and then deciding by majority vote. Here is an illustration of such a strategy.



First, you grid the petal length space into a number of bins. Then, you count the number of each type of flower in each bin. Here, the horizontal pink lines represent the probability that a flower, taken from that bin, is virginica. Given that you sample from bin i, the probability that the outcome Y equals virginica, equals the number of virginica flowers in bin i, divided by the total number of flowers in bin i. The probability that the flower is versicolor, is one minus this quantity.

$$P(Y = \text{virginica} \mid \text{bin} = i) = \frac{\text{virginica in bin } i}{\text{flowers in bin } i}$$

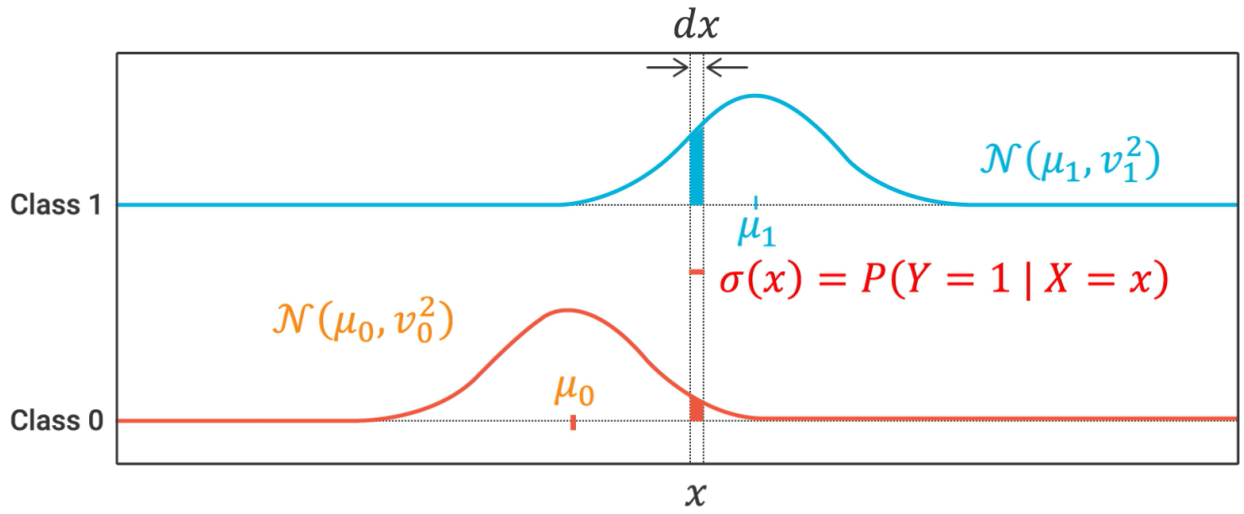
As you move from left to right, from smaller to larger petal lengths, the probability of virginica is expected to rise from zero to one—just as the probability of versicolor is expected to decrease from one to zero. The threshold petal length separating virginica and versicolor should then be placed at the bin corresponding to equal parts of the two flower types. That is, where the probability equals 0.5.

Formalizing the example

To formalize the example, use the appropriate notation: Refer to the output class Y instead of the species of flower, use class names 0 and 1 instead of versicolor and virginica, and instead of petal length, consider some arbitrary feature called X.

To derive logistic regression, make the following assumptions:

- The marginal distribution of X for each of the classes 0 and 1 are Gaussian
- The two variances are the same



This means that the probability of X given $Y = 0$ is normally distributed with mean μ_0. And variance v_0^2 and the probability of X given $Y = 1$ is normally distributed with mean μ_1 and variance v_1^2. Furthermore, since $v_0^2 = v_1^2$, both can be denoted by v^2.

$$p(X|Y = 0) \sim \mathcal{N}(\mu_0, v_0^2)$$
$$p(X|Y = 1) \sim \mathcal{N}(\mu_1, v_1^2)$$

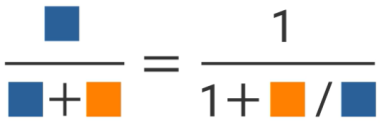
Flower classification

The probability that the items within a given narrow bin are of type 1 will be important in future calculations, so it is a good idea to give this probability its own symbol, σ.

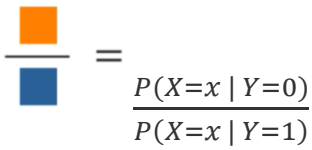
$$p(Y|X = x) = \sigma(x) = \frac{\blacksquare}{\blacksquare + \blacksquare} = \frac{1}{1 + \blacksquare / \blacksquare}$$

In this example, where blue represents virginica and orange represents versicolor, this equals the ratio of the blue shaded area divided by the sum

of blue and orange. Dividing the numerator and denominator by the blue area shows that σ depends only on the ratio of the two areas: Blue divided by orange.



This ratio is an important quantity that is referred to as the **odds ratio**. In this case, it can be denoted as follows:



Based on the assumption that the marginal distributions are Gaussian, you can plug the formula for the Gaussian distribution into the numerator and denominator. Keep in mind that they have different means, μ_0 and μ_1.

$$= \frac{\left(\sqrt{2\pi v}\right)^{-1} \exp\left(-\frac{(x-\mu_0)^2}{2v^2}\right)}{\left(\sqrt{2\pi v}\right)^{-1} \exp\left(-\frac{(x-\mu_1)^2}{2v^2}\right)}$$

Note that, since the variances are assumed to be the same, they cancel out.

You are left with a single exponential function of the difference between two squares, $(x-\mu_0)^2$ and $(x-\mu_1)^2$.

$$= \exp\left[-\frac{1}{2v^2}\left((x-\mu_0)^2 - (x-\mu_1)^2\right)\right]$$

After a few more algebraic calculations, you should arrive at the conclusion that the **odds ratio** is equal to an exponential, e^{-z}, where z is a linear function of x with coefficients β_0 and β_1.

$$z = \beta_0 + \beta_1 x$$

You can compute these betas from the parameters of the marginal distributions, μ_0 and μ_1, and the common variance, v^2.

$$\beta_0 = \frac{\mu_0^2 - \mu_1^2}{2v^2}$$

$$\beta_1 = \frac{\mu_1 - \mu_0}{v^2}$$

This expression for the odds ratio completes the formula for σ.

$$\sigma(x) = \frac{1}{1 + \text{odds ratio}} = \frac{1}{1 + e^{-z}}$$

The $\sigma(x) = \frac{1}{1+e^{-z}}$, where z equals $\beta_0 + \beta_1$ times x.

This function is known as the **sigmoid function** or the **logistic function**.

Computing the threshold

Once you have found the particular logistic function that best fits your data, you can use it to compute the threshold that separates the two classes. You can call this threshold $\bar{x}$. If the threshold is chosen so that the probabilities of each class are the same and equal to 0.5, $\sigma(\bar{x})$ must equal 0.5.

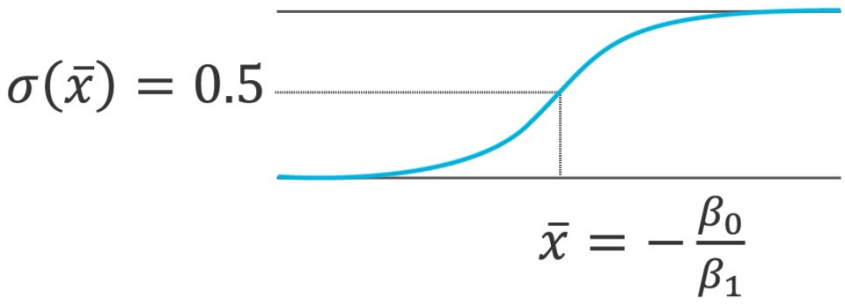
$$\sigma(\bar{x}) = 0.5$$

$$\therefore 0.5 = \frac{1}{1+e^{-\bar{z}}} \Rightarrow \bar{z} = 0$$

$$\Rightarrow 0 = \beta_0 + \beta_1 \bar{x}$$

$$\Rightarrow \bar{x} = -\frac{\beta_0}{\beta_1}$$

Plugging this into the formula, it becomes evident that $\bar{z}$ must equal 0, since $e^0 = 1$. And so, $\beta_0 + \beta_1 \bar{x}$ must equal 0. Therefore the threshold, $\bar{x}$, equals $-\frac{\beta_0}{\beta_1}$.



The approximate solution

The formulas you derived so far are based on some broad assumptions about the marginals, which include that:

- They are Gaussian
- You know their mean and variance
- Their variance are equal

When these quantities are unknown, you can approximate them with the sample means and variances. This is known as the **approximate solution**.

$$\hat{\beta}_0 = \frac{\hat{\mu}_1^2 - \hat{\mu}_2^2}{2\hat{v}^2} \qquad \hat{\beta}_1 = \frac{\hat{\mu}_2 - \hat{\mu}_1}{\hat{v}^2}$$

$\hat{\mu}_1^2$ … Sample mean class 1

$\hat{\mu}_2^2$ … Sample mean class 2

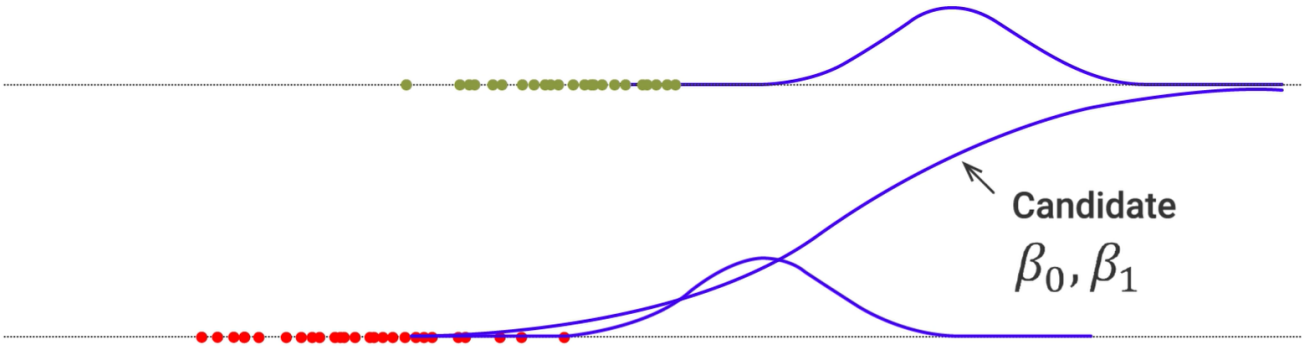
$\hat{v}^2 = \frac{\hat{v}_1^2 + \hat{v}_2^2}{2}$ … Average sample variance

In the real world, the approximate solution is only partially effective. For cases where no closed-form solution exists, a more powerful tool is needed to compute the betas. Here you can adopt the same approach as seen before with other models, such as linear regression. That is, to formulate logistic regression as an optimization problem.

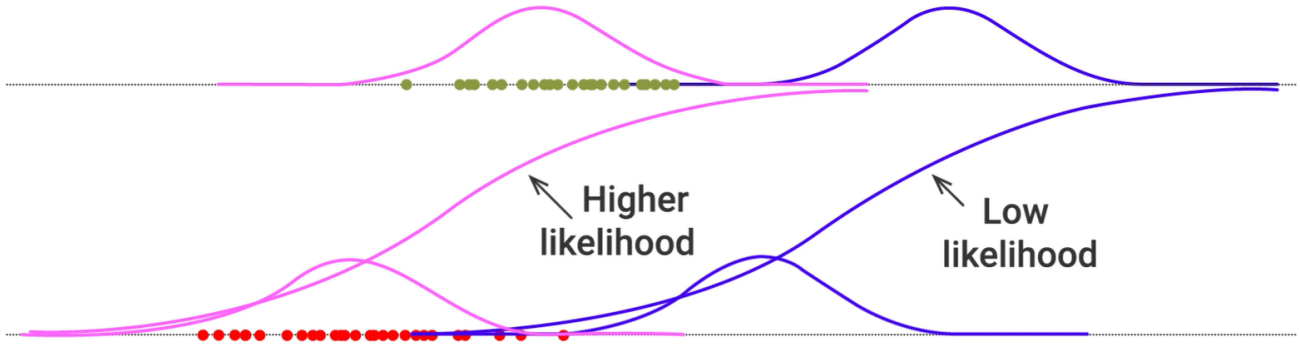
You can choose the parameters β_0 and β_1 using the **likelihood function**. This cost function is defined as: The likelihood of a set of parameters, in this case β_0 and β_1, is the probability that the dataset that you have observed could have been generated by the model implied by those parameters.

Logistic regression via maximum likelihood

Consider a dataset with measurements of two classes: Green and red.



What is the likelihood of this candidate logistic function? This function implies the green and red marginal distributions that are well to the right of the data that you have, are unlikely to have arisen from the blue model. So, this logistic function has very low likelihood.



This pink model, by comparison, has a higher likelihood. It is your objective to find the model with the highest likelihood, given the data.

Cross-entropy

To quantify the likelihood of a candidate β_0 and β_1:

1. Take the product of the probabilities of the data samples according to those parameters

2. Take the logarithm of both sides
3. Use the properties of logarithms to calculate the final objective function for your optimization

Likelihood: $\mathcal{L}(\beta_0, \beta_1) = \prod\big(1 - \sigma(x_i)\big) \prod \sigma(x_i)$

$$= \prod_{i=1}^{N} \big(1 - \sigma(x_i)\big)^{(1-y_i)} \sigma(x_i)^{y_i}$$

You can express it using this formula where N is the number of samples. As next step, you take the logarithm of both sides, which has the beneficial effect of converting the product into a sum that has no effect on the result.

$$\log \mathcal{L}(\beta_0, \beta_1) = \log\left(\prod_{i=1}^{N} \big(1 - \sigma(x_i)\big)^{(1-y_i)} \sigma(x_i)^{y_i}\right)$$

$$= \sum_{i=1}^{N} \log\Big(\big(1 - \sigma(x_i)\big)^{(1-y_i)} \sigma(x_i)^{y_i}\Big)$$

Then, using the properties of logarithms and with a couple of lines of algebra, you get to the final objective function for your optimization.

$$= \sum_{i=1}^{N} \Big((1 - y_i) \log\big(1 - \sigma(x_i)\big) + y_i \log \sigma(x_i)\Big)$$

This formula is known to information theorists as the cross-entropy, or rather the **negative of the cross-entropy**.

$$\log \mathcal{L}(\beta_0, \beta_1) = \sum_{i=1}^{N} \Big((1 - y_i) \log\big(1 - \sigma(x_i)\big) + y_i \log \sigma(x_i)\Big)$$

Instead of maximizing the likelihood, for classification problems you minimize the cross-entropy. The complete optimization problem for logistic regression thus reads: Find β_0 and β_1 that minimize the cross-entropy

$$-\sum_{i=1}^{N}\Big((1-y_i)\log\big(1-\sigma(x_i)\big)+y_i\log\sigma(x_i)\Big)$$

where σ is defined as the logistic function.

$$\sigma(x_i)=\frac{1}{1+e^{-z_i}}$$

$$z_i=\beta_0+\beta_1 x_i$$

One Feature, Two Classes in Code

To run logistic regression in scikit-learn, first load the dataset. For example, you can use the Iris dataset:

data = datasets.load_iris()

Cast the data into a DataFrame:

Iris = pd.DataFrame(data.data, columns=data.feature_names)

For ease of reference, add the target information to the DataFrame under the species column:

Iris['species'] = data.target

Rename the columns to shorten their names:

iris = iris.rename(columns={'sepal length (cm)' : 'sl',
'sepal width (cm)' : 'sw',

```
'petal length (cm)' : 'pl',
'petal width (cm)' : 'pw'})
```

This is the output.

	sl	sw	pl	pw	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

Note that the species are encoded so setosa is 0, versicolor is 1, and virginica is 2.

The simplest case of logistic regression involves two classes and one input. In this example, you will work with the two classes — versicolor and virginica.

Start by selecting the applicable rows in your DataFrame and save them to a new DataFrame, here, called **iris_2species**:

```
iris_2species = iris[ (iris.species==1) | (iris.species==2) ]
```

Train the dataset by selecting the column, pl, as input data and the column, species, as output data:

```
X = iris_2species[ ['pl'] ]
Y = iris_2species[ ['species'] ]
```

You can use the LogisticRegression class to run logistic regression in scikit-learn:

```
from sklearn.linear_model import LogisticRegression
```

Then, you instantiate one of the classes and run the fit method on it, passing in the input data and the target:

```
lr = LogisticRegression().fit(X, y)
```

This will produce an object, called lr, that contains the coefficients of the logistic regression:

```
lr_beta0 = lr.intercept_[0]
lr_beta1 = lr.coef_[0,0]
```

You can apply the formula you derived for the threshold:

```
lr_thresh = -lr_beta0/lr_beta1
```

Note that the output will take on the following format:

```
(lr_beta0, lr_beta1, lr_thresh)
```

Flowers with a petal length exceeding the threshold, which is 4.87 in this case, are considered virginicas whereas the flowers whose petal lengths fall below the threshold will be considered versicolors.

You can compare this output with the approximated threshold values by using the formulas you calculated before. First, consider the input and save the rows corresponding to versicolor in X1 and the rows corresponding to virginicas in X2:

X1 = X[y==1].to_numpy()
X2 = X[y==2].to_numpy()

You use the mean of the two variances in the formula:

var = np.mean((var1,var2))

You can compute the approximate β_0:

ax_beta0 = (mu12 – mu2**2)/2/var**

Next, you compute the approximate β_1:

ax_beta1 = (mu2–mu1)/var

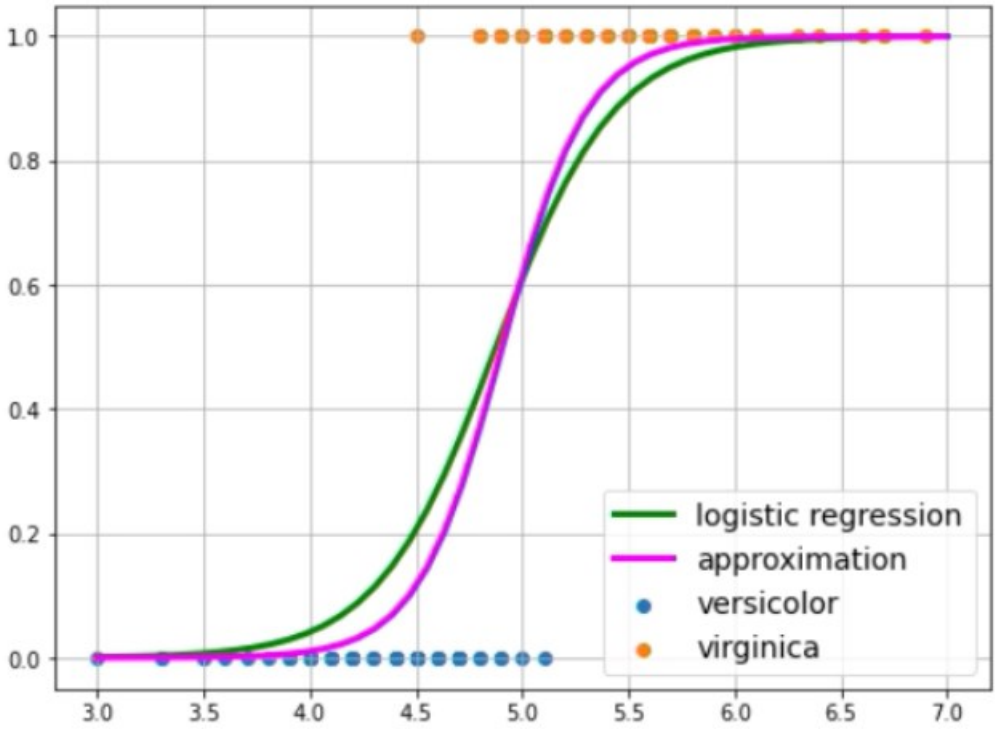
You use the same formula to compute the threshold:

ax_thresh = -ax_beta0/ax_beta1

In this case, the approximated threshold is 4.906 compared to the actual threshold of 4.87 that you calculated above.

You can compare the true logistic regression against the approximate logistic regression on a plot.

```
def sigmoid(beta0,beta1,x):
    return 1/(1+np.exp(-beta0 - beta1*x))
x = np.linspace(3,7)
```



Here, the versicolor flowers are plotted at a level of zero and the virginicas at a level of 1.

```
plt.scatter( X1, np.zeros(50), label='versicolor' )
plt.scatter( X2, np.zeros(50), label='virginica' )
```

You use the betas from scikit-learn and your approximation to evaluate the sigmoid function.

```
plt.plot(x, sigmoid(lr_beta0,lr_beta1,x), linewidth=3, color='green',
label='logistic regression')
```

plt.plot(x, sigmoid(ax_beta0,ax_beta1,x), linewidth=3, color='magenta', label='approximation')

Many Features

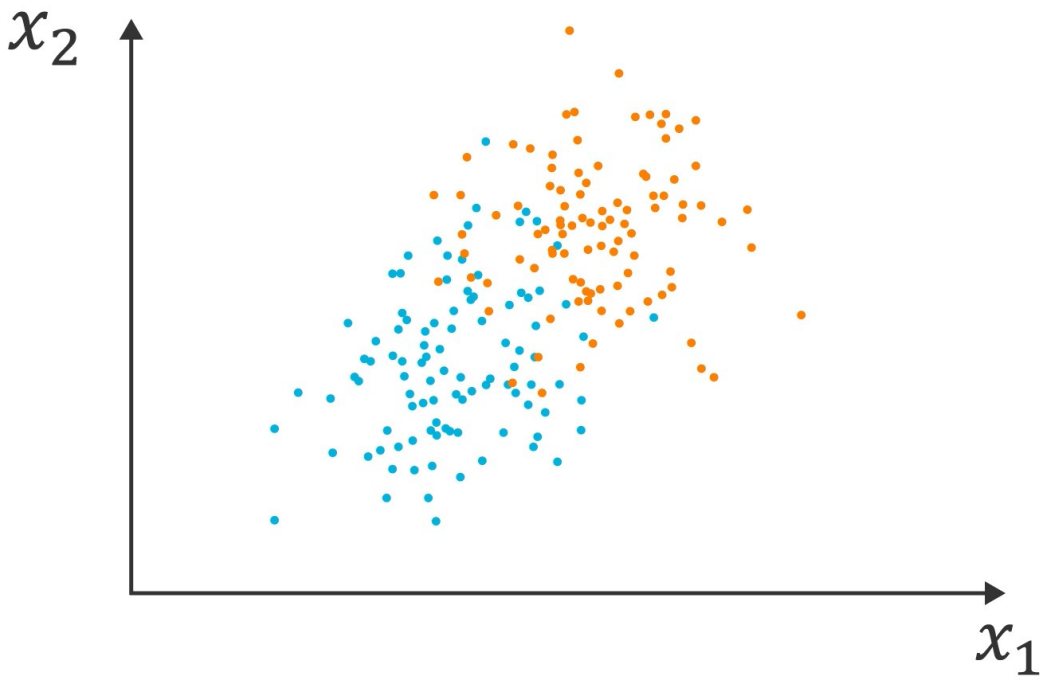
The next step toward classifying the samples according to multiple measurements, involves defining the dataset.

$$\text{Dataset} \qquad \{(x_1^i, x_2^i), y^i\} \; i = 1 \ldots N$$

This dataset can be defined as a collection of N samples, and each sample has values x_1^i and x_2^i and a categorical value y^i. You can plot the data in the x_1, x_2 plane and color each sample according to its category, y^i.

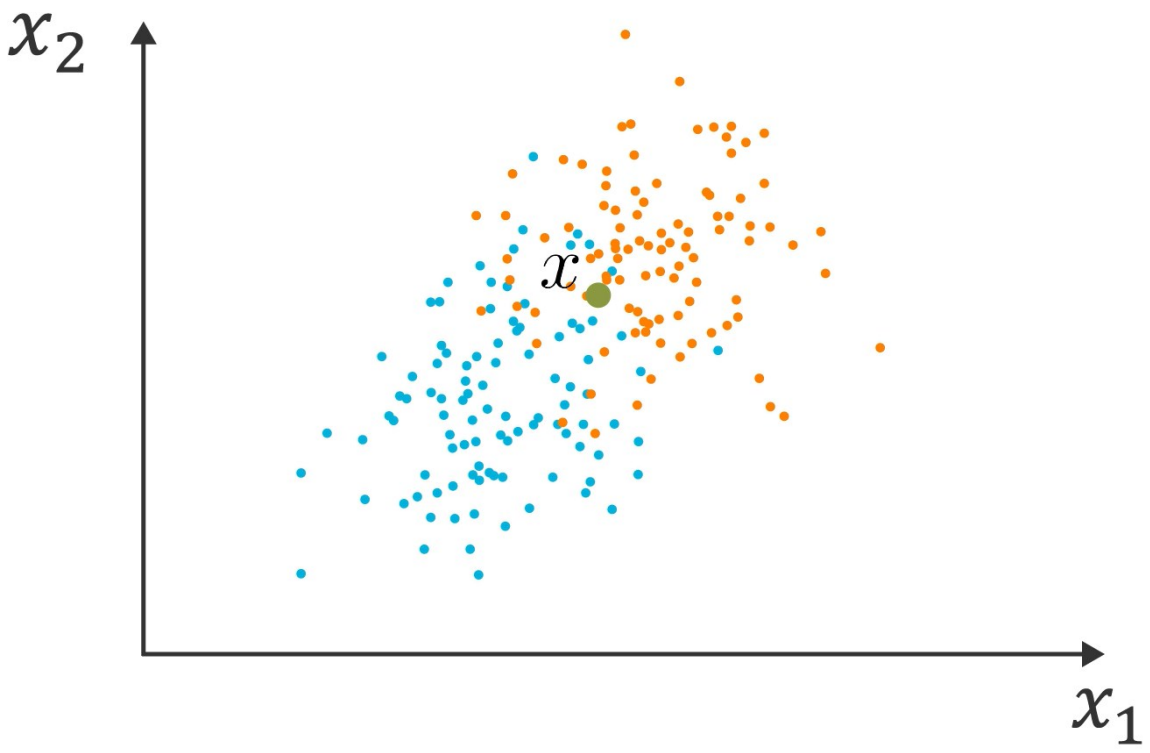
$$y^i = 0$$

$$y^i = 1$$



Classifying a new sample

Now, consider a new, uncategorized sample, x, shown here as a green dot.



How will you decide whether to classify x as orange or blue? You can draw a small box around the new sample, count the number of blue and orange dots inside the box, and decide by majority vote. To formalize this strategy, start by noting the following assumptions:

- The marginals are Gaussian
- The marginals have equal covariance matrices, Σ,
  namely $\mathcal{N}(\mu_0, \Sigma)$ and $\mathcal{N}(\mu_1, \Sigma)$ respectively
- Σ is diagonal

Note that the same formula for the log odds from the one-dimensional case also applies here in the two-dimensional case, except that the terms are now vector valued.

$$\log\frac{\blacksquare}{\blacksquare} = -\frac{1}{2}((x-\mu_0)^{T}\Sigma^{-1}(x-\mu_0) - (x-\mu_1)^{T}\Sigma^{-1}(x-\mu_1))$$

The elements can be notated as follows:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix} \qquad \mu_0 = \begin{bmatrix} \mu_{01} \\ \mu_{02} \end{bmatrix} \qquad \mu_1 = \begin{bmatrix} \mu_{11} \\ \mu_{12} \end{bmatrix}$$

The formula for the log odds is:

$$\log\frac{\blacksquare}{\blacksquare} = -\frac{1}{2}\left(\frac{\mu_{01}^2-\mu_{11}^2}{\sigma_1^2} + \frac{\mu_{02}^2-\mu_{12}^2}{\sigma_2^2}\right) - \frac{\mu_{11}-\mu_{01}}{2\sigma_1^2}x_1 - \frac{\mu_{12}-\mu_{02}}{2\sigma_2^2}x_2$$

$$= -\beta_0 - \beta_1 x_1 - \beta_2 x_2$$

Note that the form of the log odds is identical to what you found in the single-input case. The log odds are again a linear combination of the inputs with an intercept term β_0 and coefficients β_1 and β_2 for the two inputs. To continue on the path toward adding even more input features, you simply need to add more linear terms to the log odds.

The general statement for the two-class logistic regression with M inputs can be noted as:

$$P(Y=1|X=x) = \sigma(x) = \frac{1}{1+e^{-z}}$$

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_M x_M$$

$$= \beta_0 + \sum_{i=1}^{M} \beta_i x_i$$

Keep in mind that the random variable X is now multi-dimensional and x is a vector-valued sample. The result is that the probability that a sample x is of class $Y = 1$, equals the logistic function $\sigma(x)$, where z is a linear combination of the M features with coefficients β_0 through β_M.

You can find the betas by minimizing the cross-entropy, just as before.

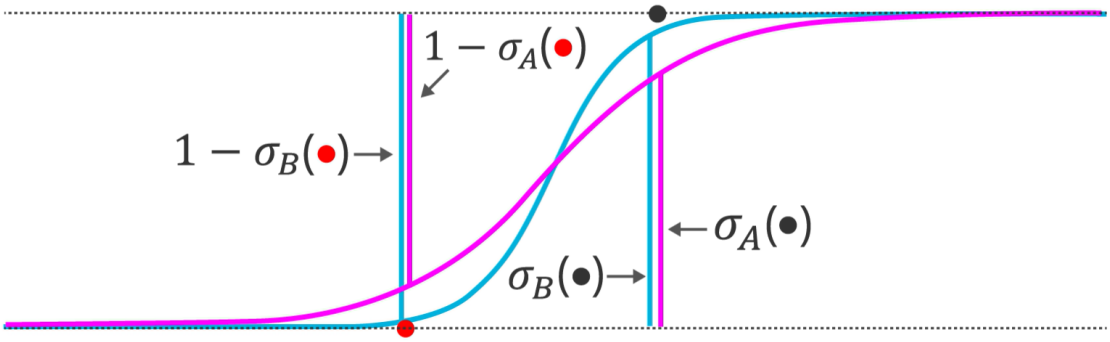
$$\text{minimize} \sum_{i=1}^{N} (1 - y_i) \log\big(1 - \sigma(x_i)\big) + y_i \log \sigma\,(x_i)$$

$$\sigma(x_i) = \frac{1}{1 + \exp\big(-\beta_0 - \sum_{j=1}^{M} \beta_j x_{i,j}\big)}$$

Logistic regression and cleanly separable data

Imagine you have a dataset where there is no overlap between the two classes. That is, there exists a threshold that exactly splits the data with perfect accuracy. What will the optimization problem do in this case?

Consider two candidate solutions: σ_A in pink and σ_B in blue.



To evaluate the likelihood of each of these, take the product of the probabilities of each data point, which are the heights of the vertical blue and pink lines in the figure. Here, σ_B must have a higher likelihood than σ_A, because in every case the vertical lines are longer. You could then take it one step further to a σ_C— that is even steeper than σ_B— and then to a σ_D and so forth ad infinitum. In this case, there is no solution to the optimization problem.

$$\sigma_\infty: \beta_0 \to \infty \text{ and } \beta_1 \to \infty$$

With each iteration of the optimizer, the coefficients β_0 and β_1 will continue to grow without bound and the solution will converge toward an infinitely steep step function.

To avoid this bad situation caused by cleanly separable data, you add a regularization term to the cost function to penalize large coefficient values. You can use either L1 regularization or L2 regularization to limit the size of the coefficients.

L1 or LASSO regularization

$$\text{minimize} \quad \text{CE}(\text{data}, \beta) + \lambda \sum_{j=1}^{M} |\beta_j|$$

L2 or ridge regularization

$$\text{minimize} \quad \text{CE}(\text{data}, \beta) + \lambda \sum_{j=1}^{M} \beta_j^2$$

Both strategies involve appending a term to the cross-entropy cost function. In the L1 or LASSO case, it is a sum of the absolute values of the coefficients. And in the L2 or ridge regression case, it is the sum of the

squares of the coefficients. Note that the **regularization parameter in scikit-learn** is called C, not λ, and it corresponds to the inverse of λ. So, to increase the strength of the regularization, decrease C.

Many Features in Code

The next step in classifying a sample according to multiple features, involves working with two classes and two inputs. So, instead of only using one of the columns from two species, you can use both the petal length and the petal width from the Iris dataset:

X = iris_2species[['pl','pw']]

You define your target as the species:

Y = iris_2species['species']

Next, you call logistic regression, reconstruct it, call the fit method, and pass in the data:

lr = LogisticRegression().fit(X, y)

This will return two coefficients corresponding to the two inputs.

When plotted, this data becomes two dimensional. It no longer lives on a line, but on the petal length, petal width plane.

You can use the **plot_region** function to create the regions corresponding to the virginica and versicolor flowers, respectively.

def plot_region(lr.d1lim,d2lim):

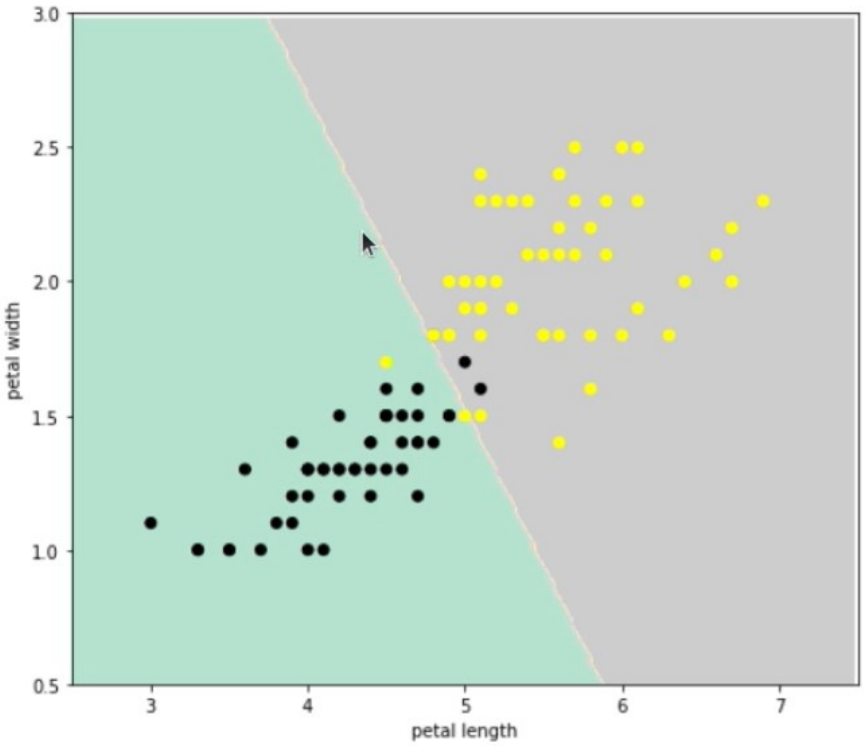
```
d1_grid, d2_grid = np.meshgrid(np.arange(d1lim[0], d1lim[1], 0.02), np.arange(d2lim[1], 0.02))

d12_array = np.c_[d1_grid.ravel(), d2_grid.ravel()]

y_array = lr.predict(d12_array)

y_grid = y_array.reshape(d1_grid.shape)

plt.contour(d1_grid, d2_grid, y_grid, cmap='Pastel2')
```



The regions are separated by a decision boundary: Everything to its right should be considered a virginica flower, whereas everything to the left should be considered a versicolor flower. Note that this model is not perfect since no line can cleanly separate two datasets.

Multiclass

So far, you have been introduced to logistic regression models that can use any number of features to distinguish between two classes of items.

Since real-world problems often involve more than two types of things, a complete picture of logistics requires techniques for tackling problems that involve more than two classes.

There are three methods for adapting binary classification models to multiclass problems:

- The **one-vs-rest** method
- The **one-vs-one** method
- The **multinomial regression** method

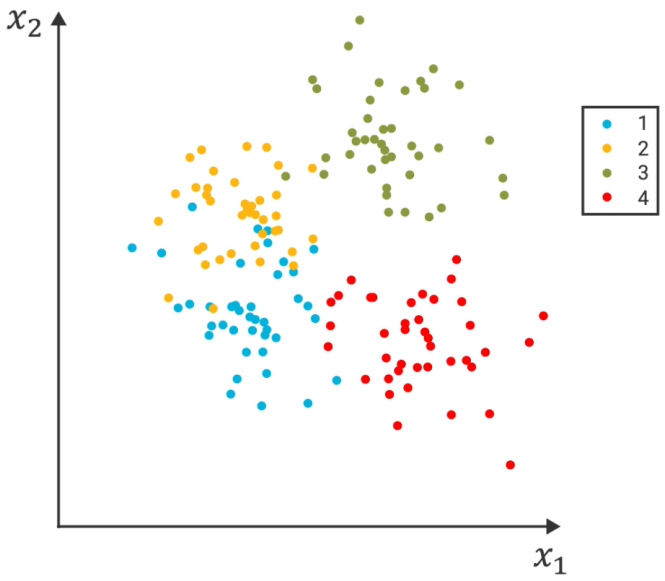
The first two, one-vs-rest and one-vs-one, are generic and can be used with other classification models aside from logistic regression.

One-vs-rest classification requires that the underlying binary classification model must return a continuous probability for the prediction, not simply a binary class assignment.

One-vs-one classification does not have this requirement. It can be used to extend any binary classifier into multiclass problems.

The last approach, multinomial regression, is a generalization of logistic regression to the multiclass case.

To illustrate, consider this two-dimensional problem with four classes:



One-vs-rest

The one-vs-rest, or OVR, strategy uses a binary classifier to build K separate models, each pitting one class against all of the others. The first of these classifiers will be trained to distinguish class 1 from all other classes. But the important part of the model is the continuous-valued probability of class 1 as a function of x.

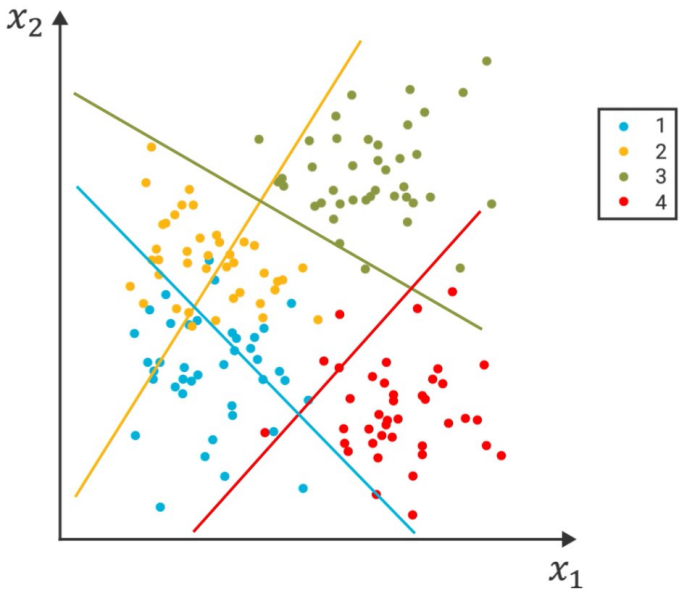
$$P(Y = 1|X = x) = \sigma_1(x)$$

If the binary classifier is logistic regression, then this will be a logistic function, denoted as $\sigma_1(x)$.

The one-vs-rest model consists of K separate binary classifiers. To predict the class of a new data point, you evaluate each of the binary classifiers and select that which produces the largest value. That is, select the class that the point x is most likely to belong to according to the K binary classifiers.

The class regions that result from the one-vs-rest approach can be denoted using this formula:

$$\{\sigma_1(x), \sigma_2(x), \sigma_3(x), \sigma_4(x)\}$$



Note that the boundaries are linear despite the fact that they were obtained by taking the maximum over sigmoid functions. The one-vs-rest extension of logistic regression, therefore, is a linear classifier. The OVR approach made use of the probability estimate generated by the binary classifier. However, not all binary classifiers produce a probability estimate. The perceptron, for example, generates a decision boundary, but no probability estimate. For this type of classifier, you can use the one-vs-one approach.

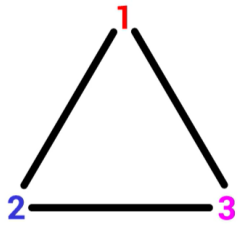
One-vs-one

This approach runs the binary classifier on every pair of classes. So, if there are three classes, it will require running one versus two, one versus three, and two versus three.

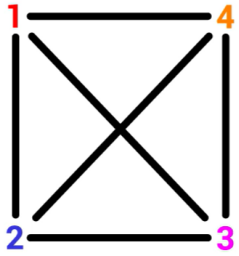
The total number of executions of the binary classifier is described as:

$$Number\ of\ executions = K\frac{(K-1)}{2}$$

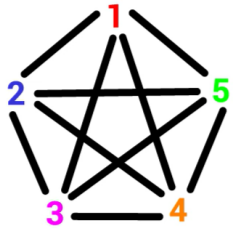
As such, four classes will require the construction of six binary models and five classes is equal to ten models, et cetera.



The three-binary model

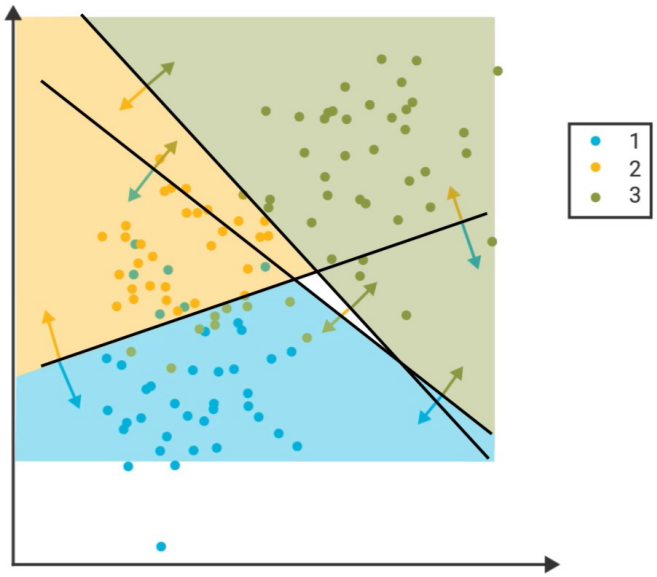


The six-binary model



The ten-binary model

Consider the simpler case of three classes. To classify a new data point, you need to evaluate all three binary classifiers and decide by majority vote. In this case, the green sample is regarded as class three by two of the models and as class one by the other. Hence, it is predicted to be of class 3.



The class regions created with one-vs-one are indicated above. The boundaries of these regions are boundaries of the binary classifiers. If the binary classifiers are linear, then the one-vs-one extension will also be linear. Note, however, that there is a problem with this picture. The region in the middle is not shaded. This is because the voting in this region resulted in a three-way tie amongst all of the classes. So, the data cannot be classified. This is a drawback of the one-vs-one approach.

Comparing one-vs-rest and one-vs-one

The one-vs-rest method	The one-vs-one method
Linearly dependent on K	Quadratically dependent on K
Imbalanced data	Balanced data
Probability required	Probability not required
Able to classify all data points	Unable to classify all data points

Multinomial Regression

Multinomial regression is the most general form of logistic regression. Similar to one-vs-rest, it also requires the calculation of a continuous probability for each class.

Multinomial regression is similar to one-vs-one, in its requirement that each of the models be trained by comparing one class to just one other.

To perform multinomial regression for the model one-vs-K:

- Specify a reference class
- Build $K-1$ logistic regression models by comparing each of the classes 1 through $K-1$ to the reference class K
- Train the coefficients, $\beta_{1,0}$ through $\beta_{1,M}$, combined linearly with the features to produce the log odds

1 vs. K

$$\log\frac{P(Y=1)}{P(Y=K)} = -\beta_{10} - \sum_{j=1}^{M}\beta_{1j}x_j = -\beta_1 \cdot x$$

As shown above, you can simplify the notation by defining a vector, β_1, of coefficients for this model and denoting the linear combination as $-\beta_1 \cdot x$.

You then do the same for classes 2 through $K-1$.

2 vs. K $\quad \log\frac{P(Y=2)}{P(Y=K)} = -\beta_2 \cdot x$

$\vdots \qquad\qquad \vdots$

K*−1 vs. *K $\quad \log\frac{P(Y=K-1)}{P(Y=K)} = -\beta_{K-1} \cdot x$

From each of these, you express the probability of any class κ as the probability of class K times the exponential of $-\beta_1 \cdot x$. Here, κ is any class from 1 to $K-1$.

$$P(Y=\kappa) = P(Y=K)\exp(-\beta_\kappa \cdot x)$$

Because the class probabilities must add up to one, the probability of class $K=1$ minus the sum of probabilities from 1 to $K-1$.

$$\sum_{\kappa=1}^{K} P(Y=\kappa) = 1 \;\Rightarrow\; P(Y=K) = 1 - \sum_{\kappa=1}^{K-1} P(Y=\kappa)$$

You can use the expression for the probability of class κ that you just derived and obtain the probability of class K:

$$P(Y=K) = \frac{1}{1+\sum_{\kappa=1}^{K-1} exp(-\beta_\kappa \cdot x)}$$

where

$$\kappa = 1 \ldots K$$

Using this expression, you get the probability of the other classes, 1 through $K-1$, as the odds of the class divided by the same denominator.

$$P(Y=\kappa) = \frac{exp(-\beta_\kappa \cdot x)}{1+\sum_{\kappa=1}^{K-1} exp(-\beta_\kappa \cdot x)}$$

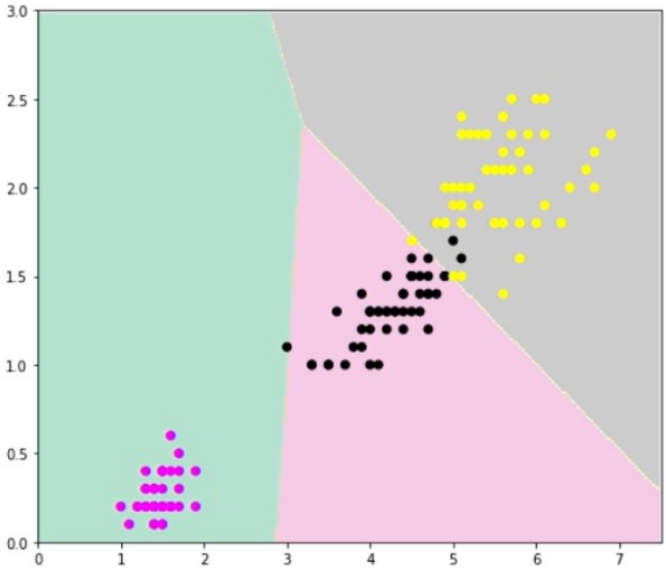
Multiclass in Code

Recall that there are three types of flowers in the original Iris dataset: Virginica, versicolor and setosa. There are two possible tactics for classifying the flowers according to petal length and petal width, namely

one-vs-rest and multinomial. These two methods use very similar code and both require passing the parameter, **multi_class**, into the constructor for logistic regression, calling the fit method, and supplying it with the data.

lr = LogisticRegression(multi_class='ovr').fit(X, y)

If plotted, three regions separate the virginica, versicolor, and setosa flowers.

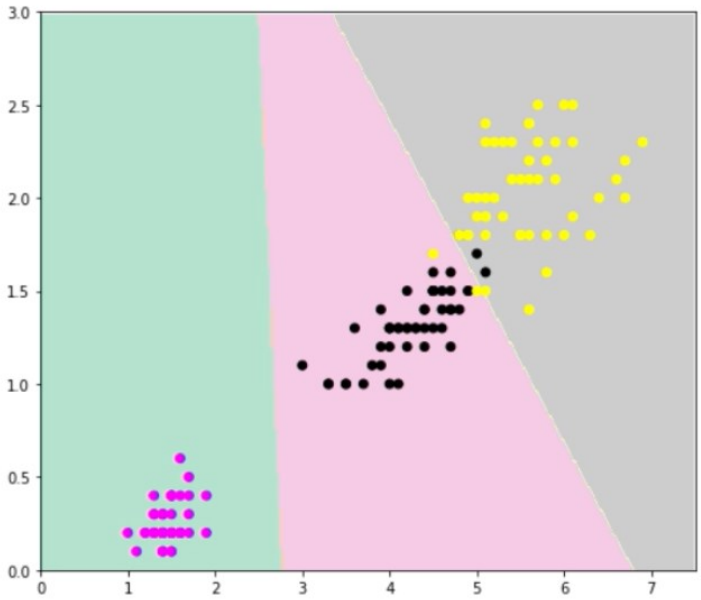


Multinomial regression

The code for the multinomial method is noted as follows:

lr = LogisticRegression(multi_class='multinomial').fit(X, y)

It creates the following chart:



To decide which is better, you would have to evaluate the methods using metrics like precision, accuracy, and so forth. This would require splitting the data into a training dataset and a testing dataset.

Feature selection via L1 regularization

Suppose you have a large dataset with many features and you don't know which ones to use in your model.

Returning to the Iris dataset, you have four features and want to use all of them to separate two species: Virginica and versicolor.

X = iris_2species.iloc[:,:4]
Y = iris_2species.iloc[:,-1]

You can run LogisticRegression but be sure to add the regularization penalty.

lr = LogisticRegression(penalty='l1', C=c, solver='liblinear').fit(X, y)

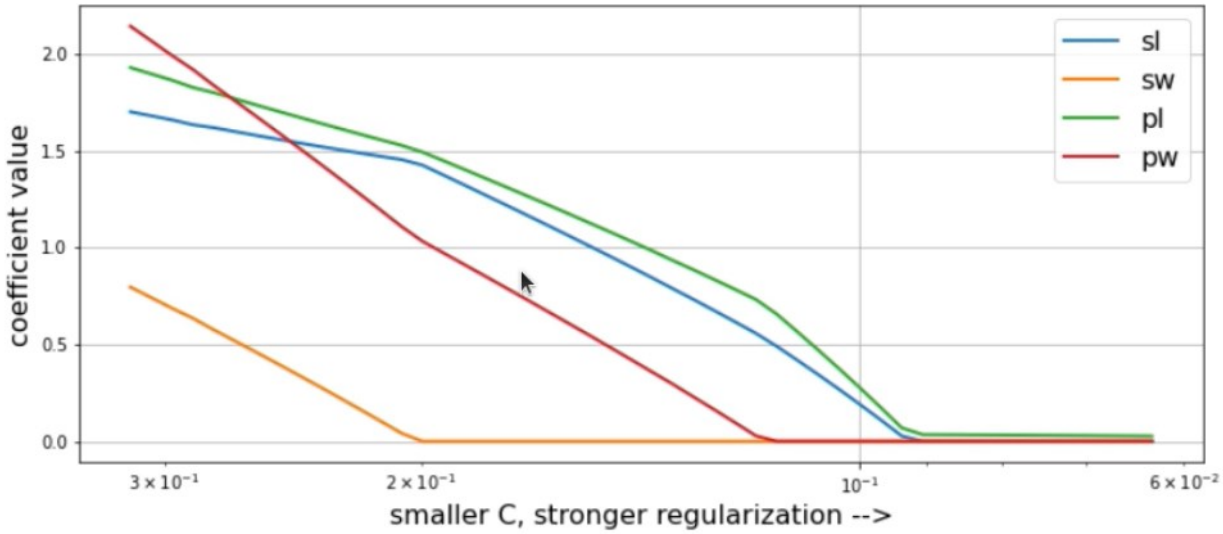
You vary the strength of the penalty, C, from ten to the minus 1.2 to ten to the minus 0.5:

C = np.logspace(-1.2,-0.5)

Next, you loop this over and store the resulting coefficients in an array:

coef[i] = lr.coef_

When plotting this, you can flip the x-axis so that C gets smaller from left to right. This highlights how the regularization gets stronger as C gets smaller.



This means that, as the regularization gets stronger, it is becoming more and more expensive to keep coefficients that are not doing much work in the model. And so the first model is going to sacrifice the sepal width coefficient and then the petal width, and then the sepal length, and then the petal length.

So, if you wanted to build a model, but you only could include one feature, that feature should be the petal length, because that is the most valuable

feature. If you could include two features, then you would add to it the sepal length. If you can include three, you should add the petal width.

This principle can be applied to models with hundreds of features so that the features are ranked according to their value in the task of modeling.