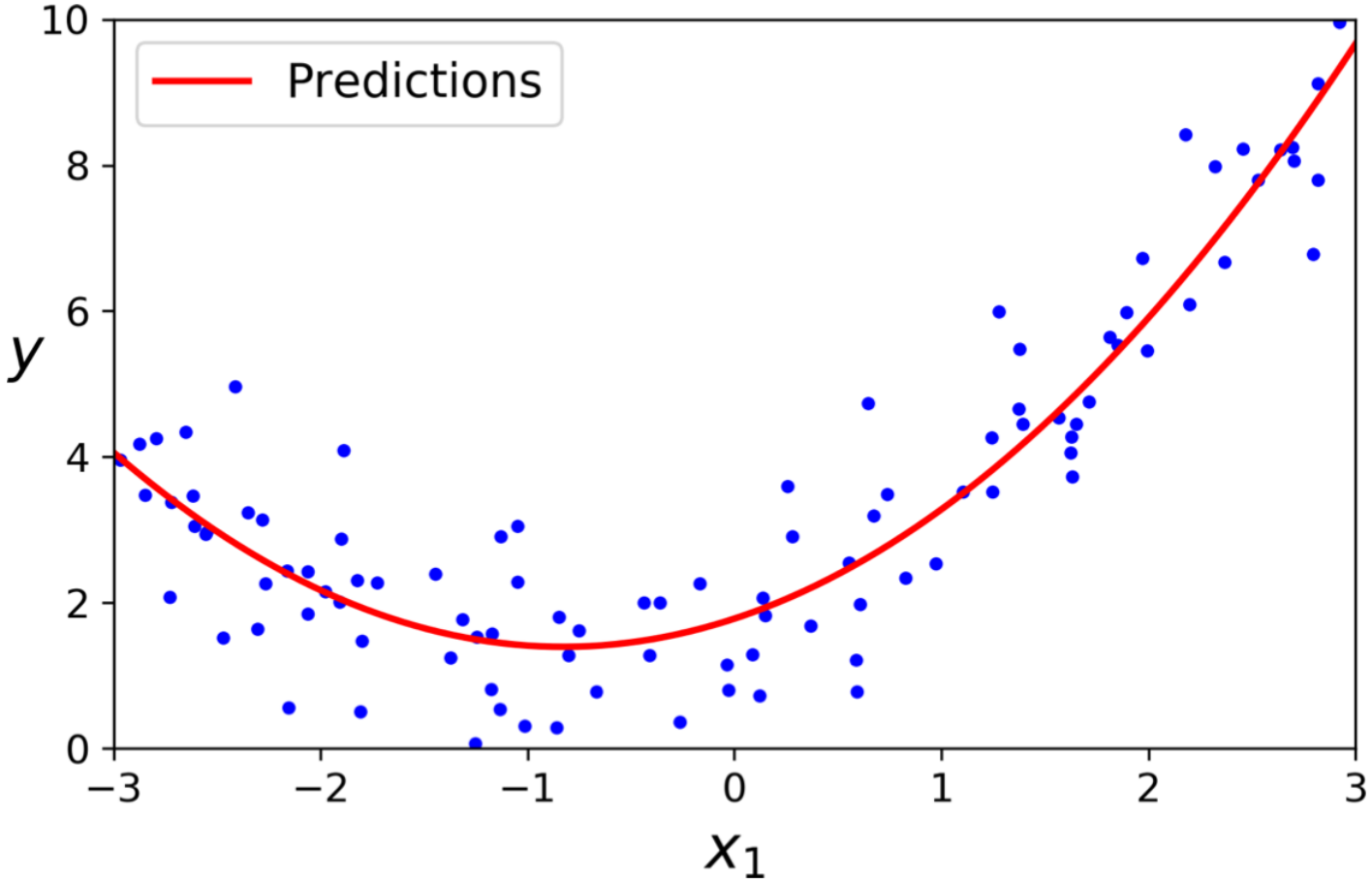
CSY2082 Introduction to Artificial Intelligence

Logistic Regression

Polynomial Regression

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



$$\hat{y} = 0.56x_1^2 + 0.93x_1 + 1.78$$

Logistic Regression

Logistic Regression (also called Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class (e.g., what is the probability that this email is spam?).

If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labeled “1”), or else it predicts that it does not (i.e., it belongs to the negative class, labeled “0”). This makes it a binary classifier.

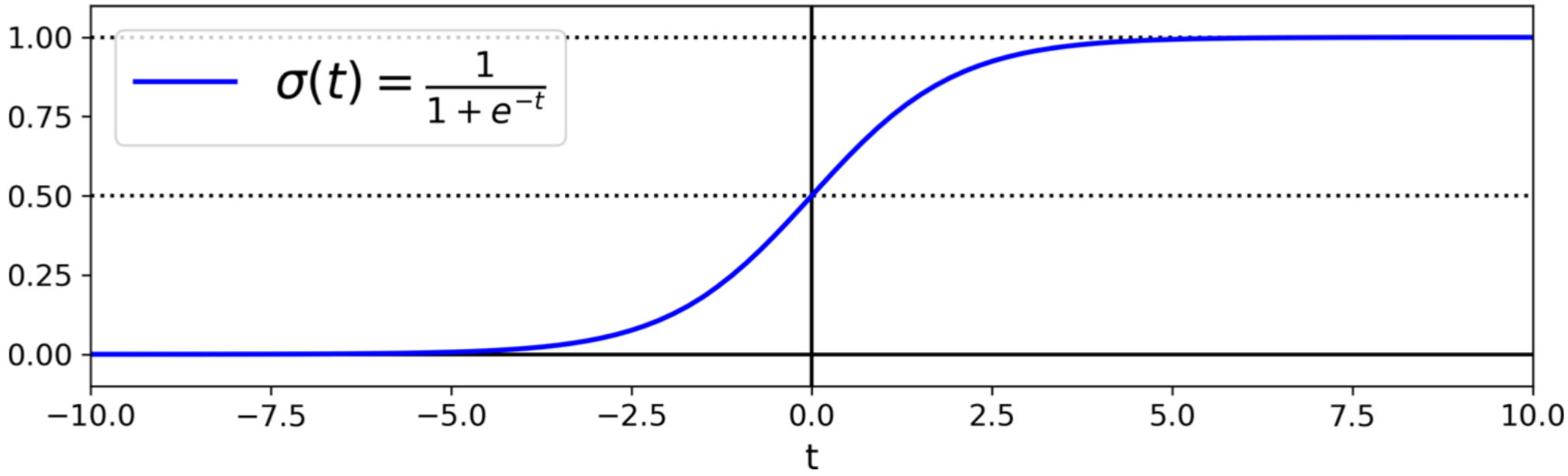
Estimating Probabilities

Instead of outputting the result directly like the Linear Regression model does, it outputs the **logistic** of this result.

The logistic—noted σ(·)—is a **sigmoid** function (i.e., S-shaped) that outputs a number between 0 and 1.

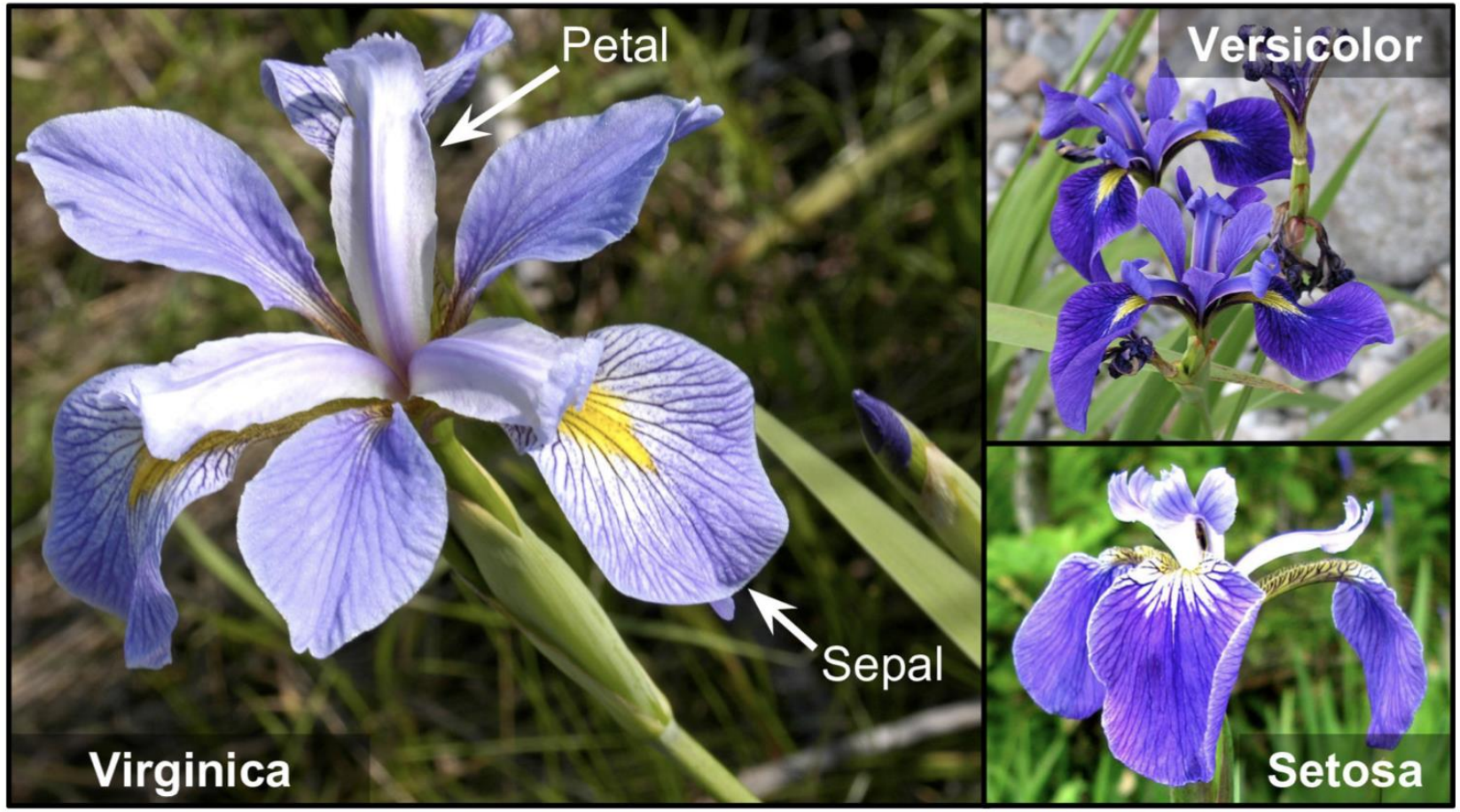
Equation 4-14. Logistic function

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

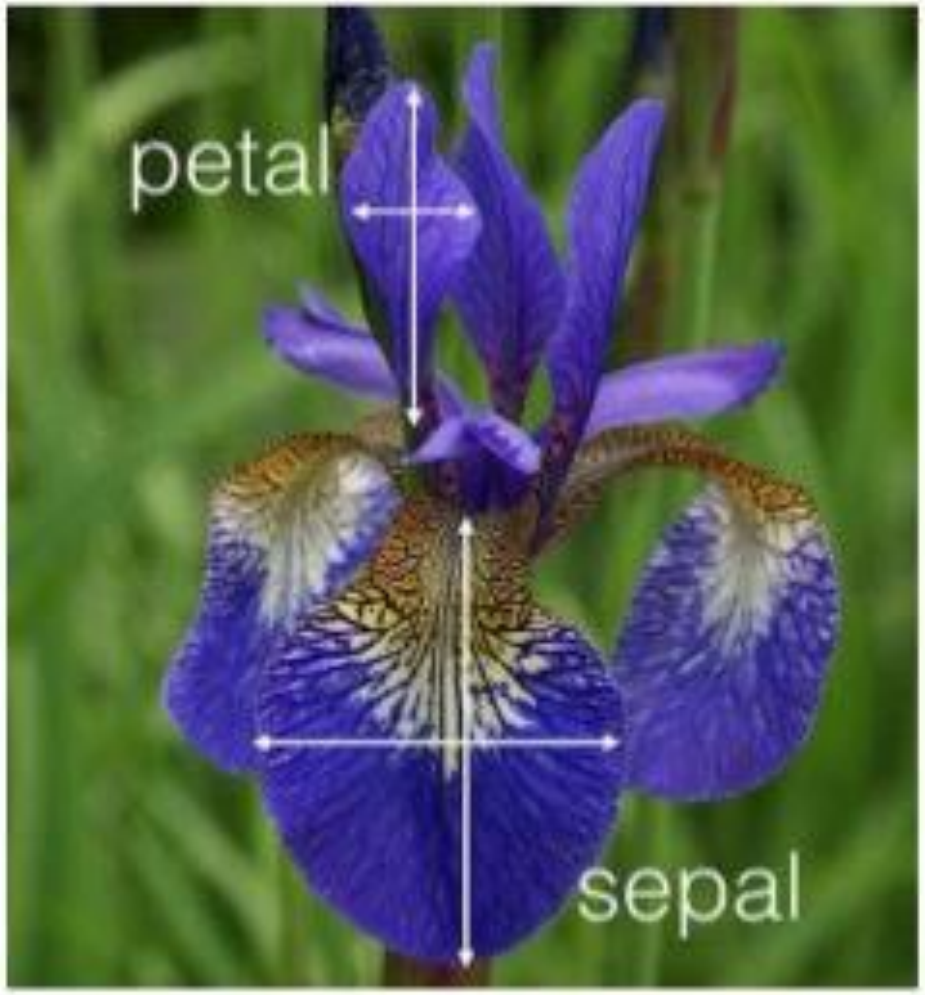


Iris dataset

Let's use the **iris dataset** to illustrate Logistic Regression. This is a famous 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.



Iris dataset



Training / test data

Features				Labels
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

Iris dataset

Import data

```
from sklearn.datasets import load_iris
iris = load_iris(as_frame=True) # import as pandas dataframe
list(iris)
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename’]
```

Iris dataset

```
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

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

```
0    0
1    0
2    0
Name: target, dtype: int64
```

```
iris.target_names
```

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

Iris dataset

Let's try to build a classifier to **detect the Iris-Virginica type based only on the petal width** feature. First let's load the data:

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

X = iris.data[["petal width (cm)"]].values # take values because Pandas dataframe
y = iris.target_names[iris.target] == 'virginica' # True if Iris-Virginica, else False
```

```
x
```

```
array([[0.2],
       [0.2],
       [0.2],
       [0.2],
       [0.2],
       [0.4],
       [0.3],
       [0.2],
       [0.2],
       [0.1],
       [0.2],
       [0.2],
       [0.1],
       [0.1],
       [0.2],
```

```
y
```

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
```

Iris dataset

Let's try to build a classifier to **detect the Iris-Virginica type based only on the petal width** feature. First let's load the data:

```
# randomly split 25% (default value of test_size is 0.25) of data as the test set.
X_train, X_test, y_train, y_test = train_test_split(X, y)

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

train_test_split

sklearn.model_selection.train_test_split(**arrays*, *test_size*=*None*, *train_size*=*None*, *random_state*=*None*, *shuffle*=*True*, *stratify*=*None*)

Split arrays or matrices into random train and test subsets.

sklearn.linear_model.LogisticRegression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2'*, ***, *dual=False*, *tol=0.0001*, *C=1.0*, *fit_intercept=True*, *intercept_scaling=1*, *class_weight=None*, *random_state=None*, *solver='lbfgs'*, *max_iter=100*, *multi_class='auto'*, *verbose=0*, *warm_start=False*, *n_jobs=None*, *l1_ratio=None*) [source]

Iris dataset

Use log_reg for prediction (classification): predict() vs predict_proba()

`predict(X)`

Predict class labels for samples in X.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

The data matrix for which we want to get the predictions.

Returns:

y_pred : *ndarray of shape (n_samples,)*

Vector containing the class labels for each sample.

Beginner: "Just tell me which class my data most likely belongs to. I don't need details."

`predict_proba(X)` [source]

Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class. Else use a one-vs-rest approach, i.e. calculate the probability of each class assuming it to be positive using the logistic function and normalize these values across all the classes.

Parameters:

X : *array-like of shape (n_samples, n_features)*

Vector to be scored, where `n_samples` is the number of samples and `n_features` is the number of features.

Returns:

T : *array-like of shape (n_samples, n_classes)*

Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Power user: "Tell me the probabilities of my data belonging to each of the given categories. I will decide what to do with them and make my own conclusions."

Decision Boundaries

numpy.linspace

numpy.linspace(*start*, *stop*, *num=50*, *endpoint=True*, *retstep=False*, *dtype=None*, *axis=0*, *, *device=None*)

Return evenly spaced numbers over a specified interval.

Returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*].

Let's look at the model's estimated probabilities for flowers with petal widths varying from 0 to 3 cm

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1) #reshape(-1, 1) turns our one row of numbers into a column, as seen below.

y_proba = log_reg.predict_proba(X_new)
```

```
X_new
```

```
array([[0.        ],
       [0.003003  ],
       [0.00600601],
       [0.00900901],
       [0.01201201],
       [0.01501502],
       [0.01801802],
       [0.02102102],
       [0.02402402],
       [0.02702703],
       [0.03003003],
```

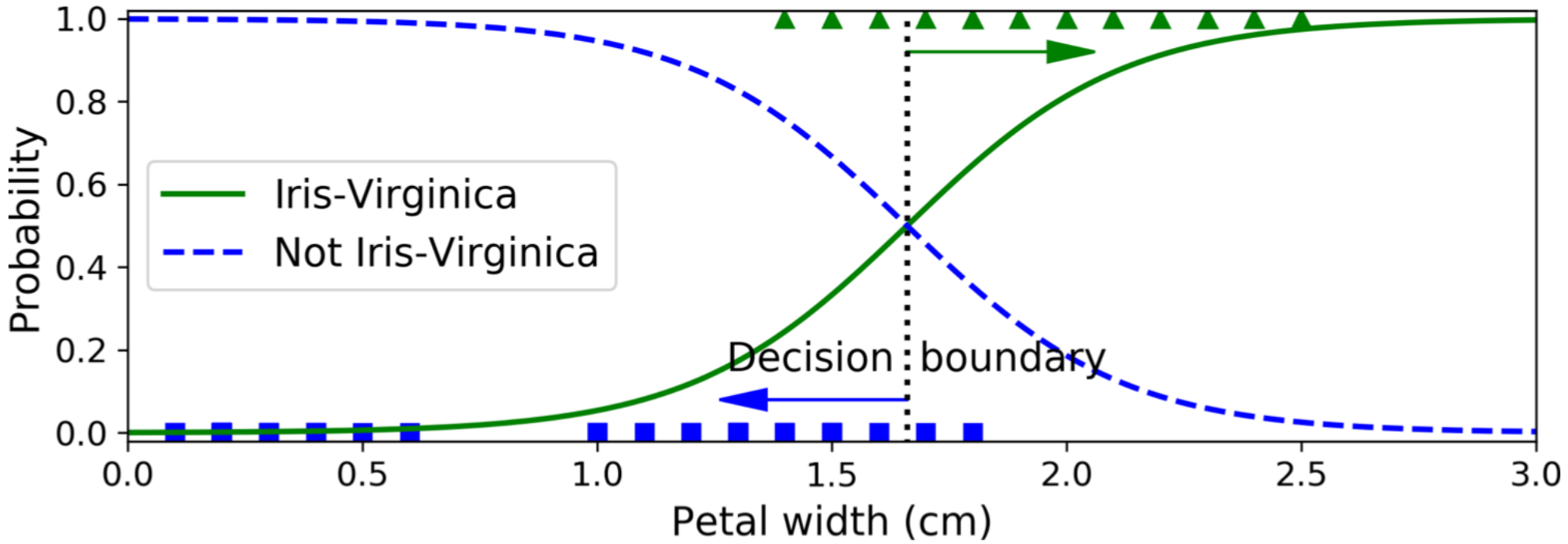
```
y_proba
```

```
array([[9.99250016e-01, 7.49984089e-04],
       [9.99240201e-01, 7.59799387e-04],
       [9.99230257e-01, 7.69743043e-04],
       ...,
       [3.08374822e-03, 9.96916252e-01],
       [3.04400296e-03, 9.96955997e-01],
       [3.00476842e-03, 9.96995232e-01]])
```

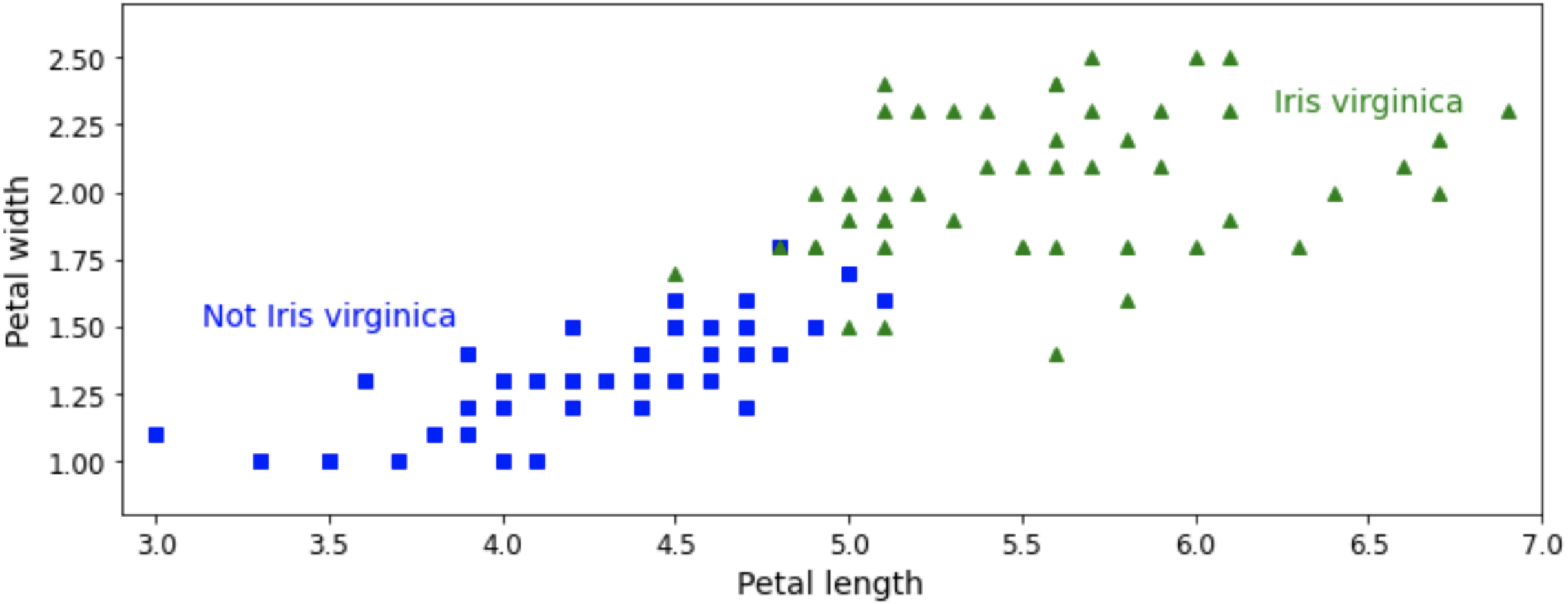
Decision Boundaries

Let’s look at the model’s estimated probabilities for flowers with petal widths varying from 0 to 3 cm

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-Virginica")
```



Decision Boundaries



Exercise:

- How do we measure the accuracy of our model?
  - Consider a different probability threshold for the output.
    - Use predict_proba() to get the probabilities, then set your own threshold to decide which class each sample belongs to.
  - Use the test set to evaluate the model
    - Use predict(test_X) to generate prediction and compare with test_y

CSY2082 Introduction to Artificial Intelligence

Softmax Regression

Softmax Regression

The Logistic Regression model can be generalized to support multiple classes: **Softmax Regression**.

The idea is : when given an instance x, the Softmax Regression model first computes a score $s_k(x)$ for each class k, then estimates the probability of each class by applying the Softmax function (aka normalized exponential) to the scores.

it computes the exponential of every score, then normalizes them (dividing by the sum of all the exponentials)

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

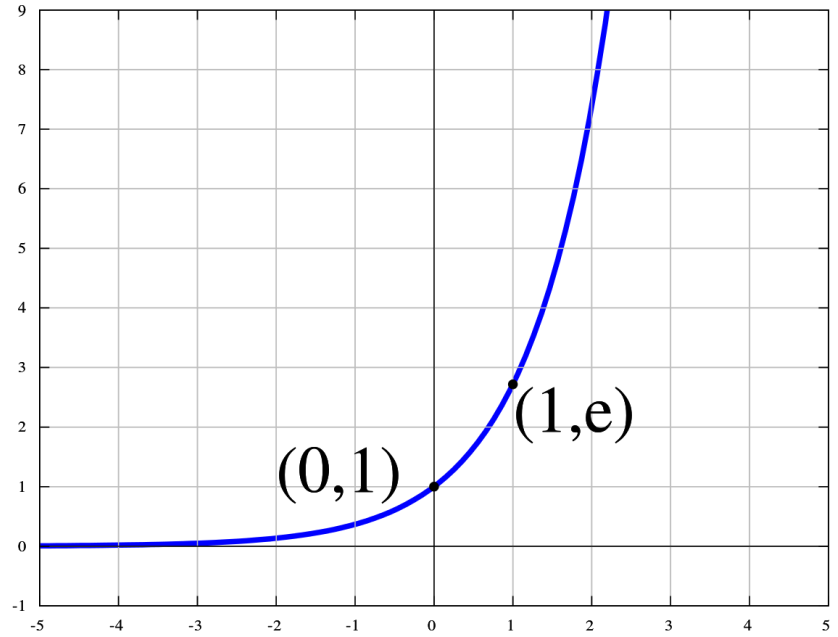
- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.

Softmax Regression

class	s	exp(s)	Estimated probability
1	1	2.71	0.017
2	5	148.41	0.937
3	2	7.38	0.046

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

- K is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class k given the scores of each class for that instance.



The natural exponential function along part of the real axis

Softmax Regression

Futher readling: https://homl.info/xentropy

Cross entropy is frequently used to measure how well a set of estimated class probabilities match the target classes.

The cross entropy between two probability distributions p and q is defined as

$$H(p, q) = -\Sigma_x p(x) \log q(x)$$

Equation 4-22. Cross entropy cost function

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

Softmax Regression

Let’s use softmax regression to classify the iris plants into all three classes.

Scikit-Learn’s LogisticRegression classifier uses softmax regression automatically when you train it on more than two classes. It also applies ℓ2 regularization by default, which you can control using the hyperparameter C. #the lower the value the stronger the penalty. Check documentation.

```
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)

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

Softmax Regression

So the next time you find an iris with 5 cm long and 2 cm wide petals, you can ask your model to tell you what type of iris it is, and it will answer Iris-Virginica (class 2) with 94.2% probability (or Iris-Versicolor with 5.8% probability):

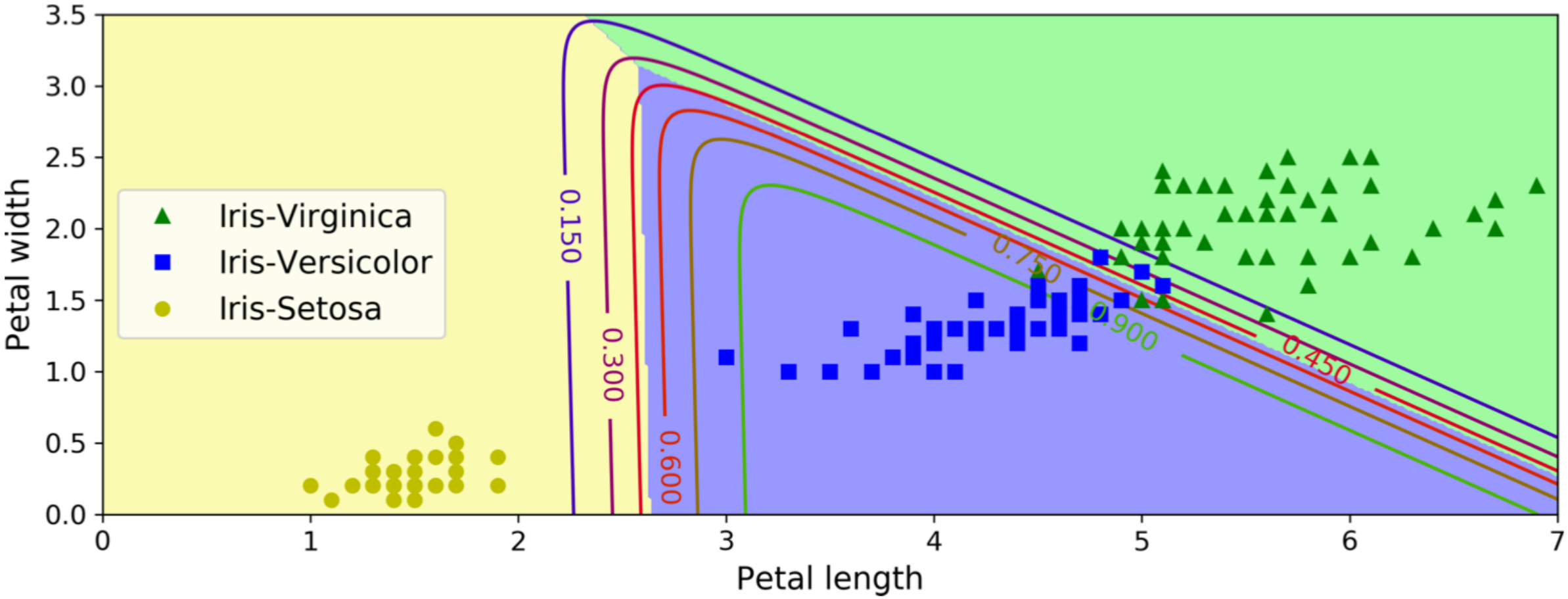
```
>>> softmax_reg.predict([[5, 2]])
array([2])
>>> softmax_reg.predict_proba([[5, 2]])
array([[6.38014896e-07, 5.74929995e-02, 9.42506362e-01]])
```

remember, predict() often expect s an array as the input, so even if you have only one data point, it must be wrapped in an array.

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

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

Softmax Regression



Exercise:

- Use all four features of the iris dataset to develop a logistic regression model
  - Compare the performance with the 2-feature model.
  - try sklearn.metrics.accuracy_score() as the performance metric.

accuracy_score

sklearn.metrics.accuracy_score(*y_true*, *y_pred*, *, *normalize*=*True*, *sample_weight*=*None*) [source]

Accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true.