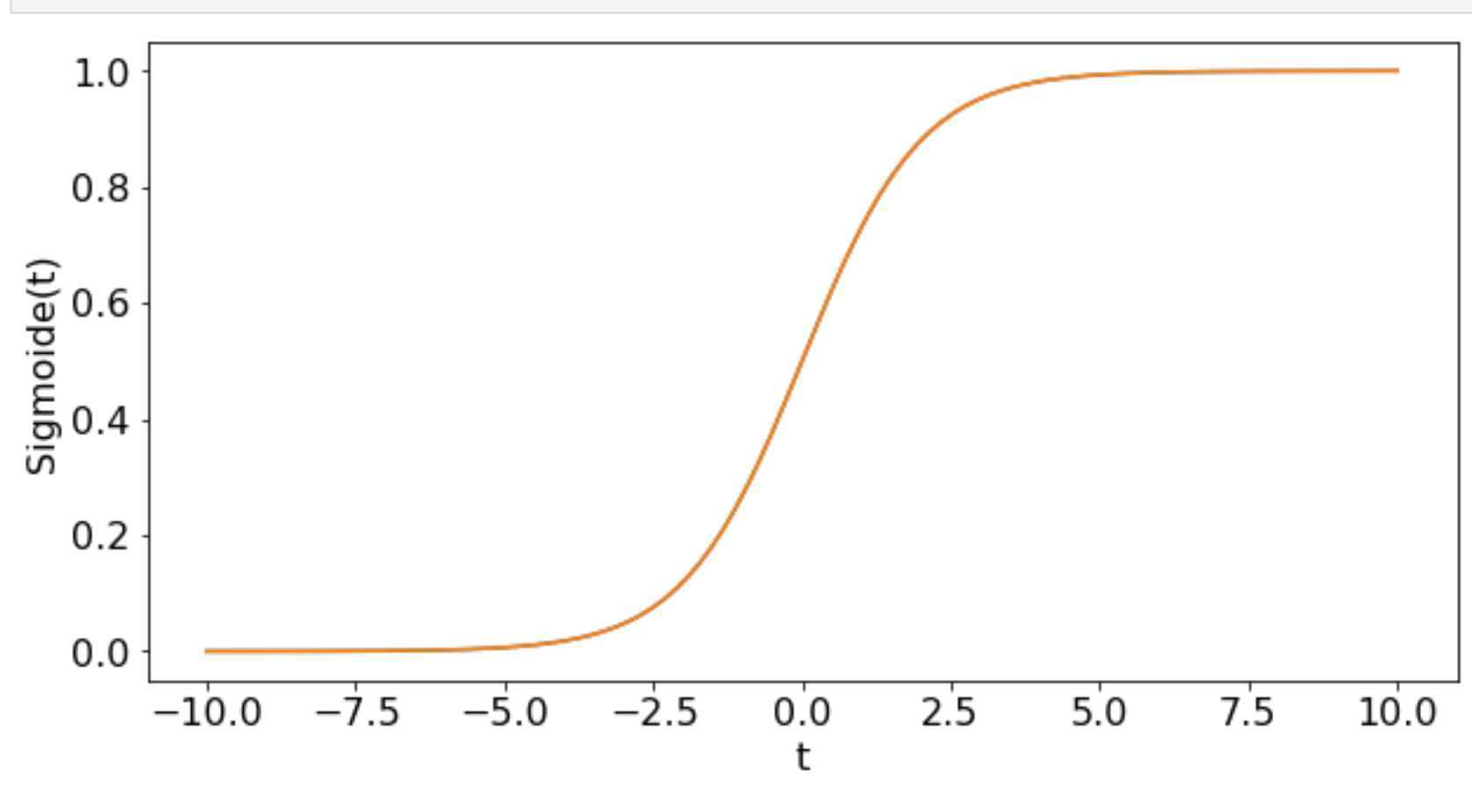
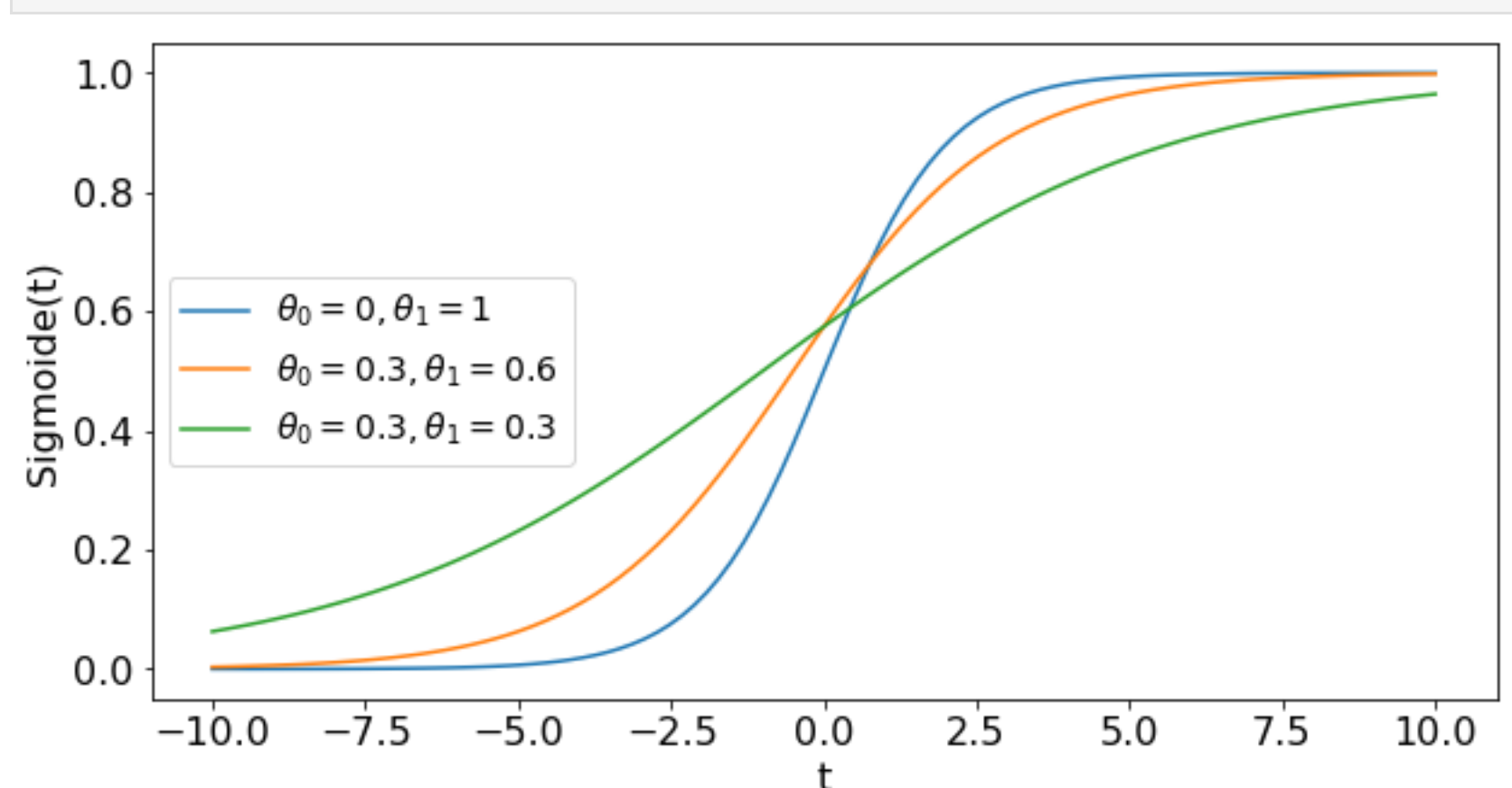
In [1]:
```python
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
import math
```

La fonction logistique

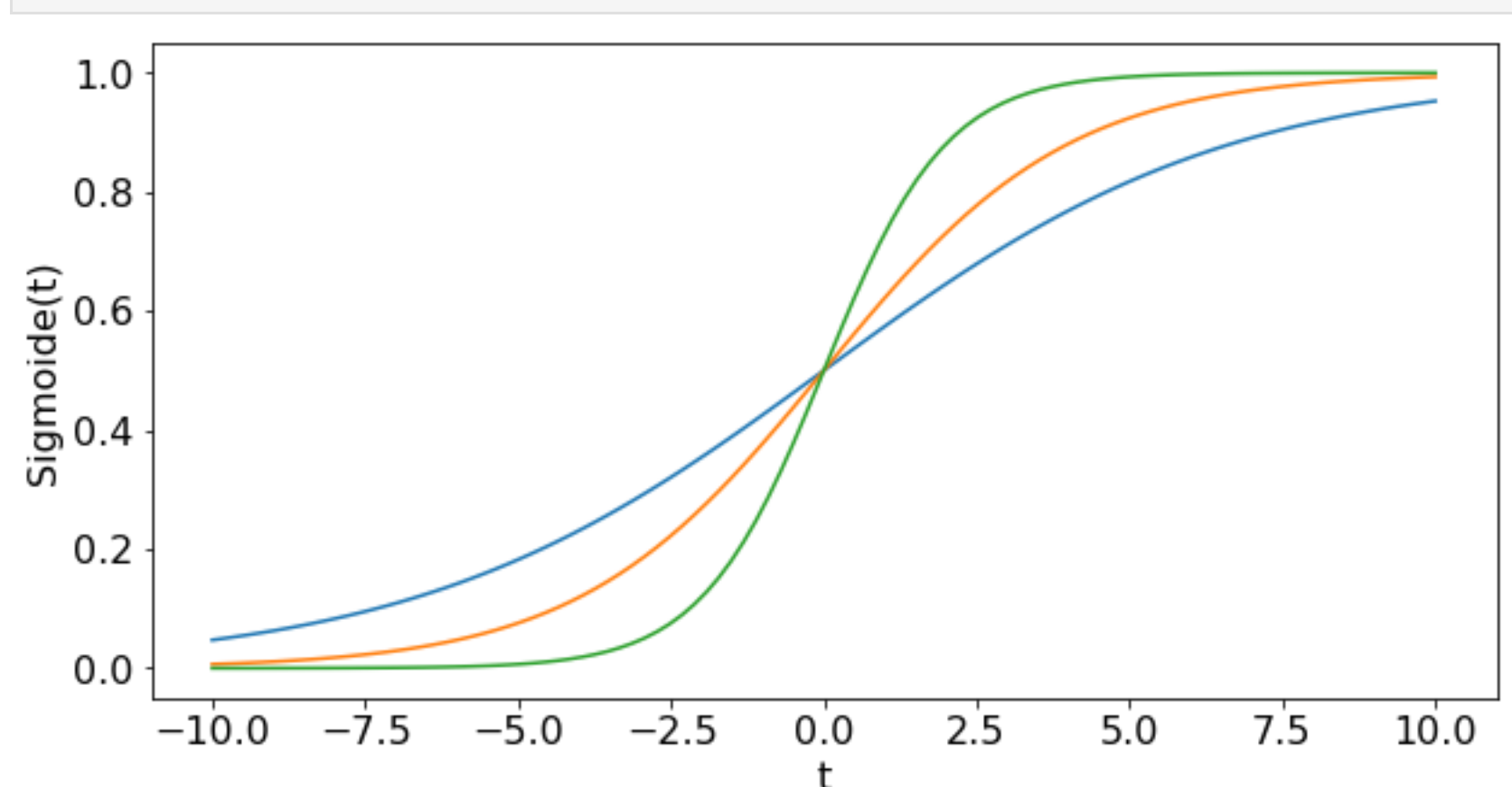
In [2]:
```python
x = np.linspace(-10, 10, 100)
z = 1/(1 + np.exp(-x))
v = np.exp(x)/(1 + np.exp(x))
plt.rcParams.update({'font.size': 16})
fig = plt.figure(figsize=(10, 5))
plt.plot(x, z)
plt.plot(x, v)
plt.xlabel("t")
plt.ylabel("Sigmoide(t)")

plt.show()
```



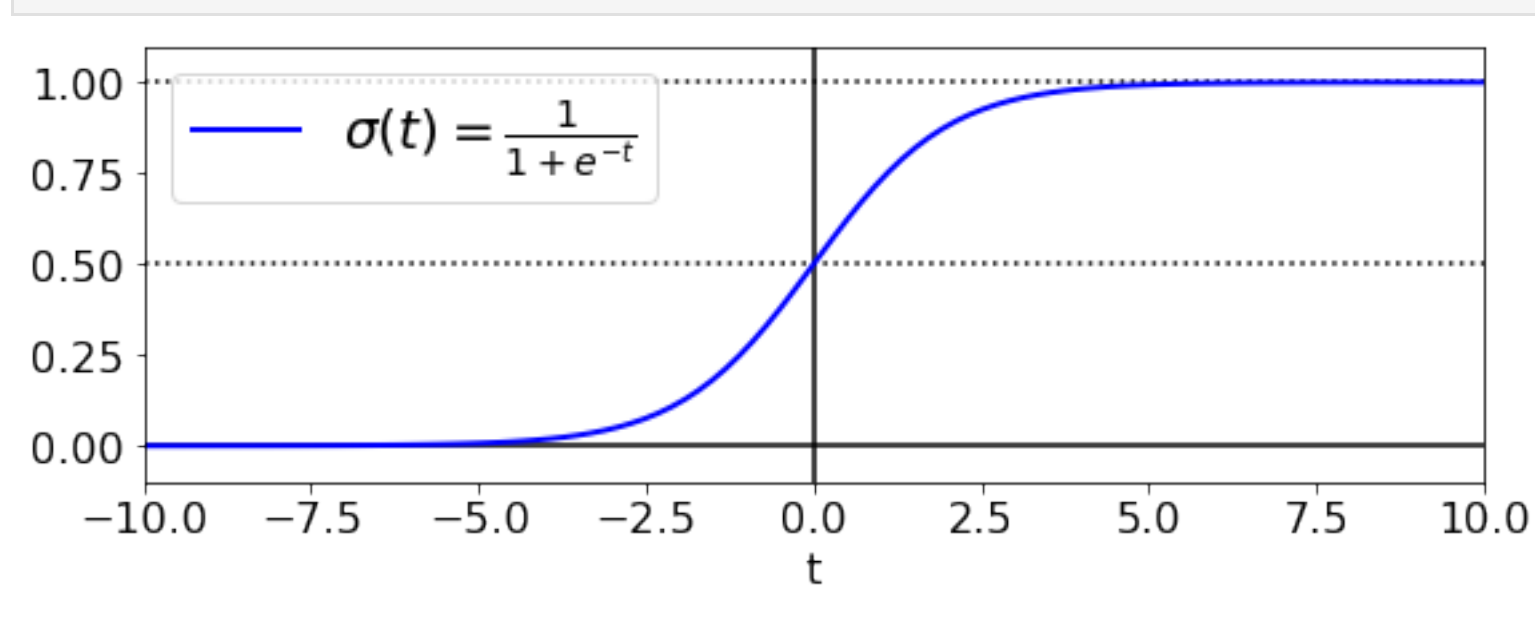
In [3]:
```python
t = np.linspace(-10, 10, 100)
x1 = 1*t
x2 = 0.3+0.6*t
x3=  0.3+0.3*t

z1 = 1/(1 + np.exp(-x1))
z2 = 1/(1 + np.exp(-x2))
z3 = 1/(1 + np.exp(-x3))
plt.rcParams.update({'font.size': 16})
fig = plt.figure(figsize=(10, 5))
plt.plot(x, z1,  label = r'$\theta_0=0, \theta_1=1$')
plt.plot(x, z2,  label = r'$\theta_0=0.3, \theta_1=0.6$')
plt.plot(x, z3,  label = r'$\theta_0=0.3, \theta_1=0.3$')
plt.xlabel("t")
plt.ylabel("Sigmoide(t)")
plt.legend(loc="center left", fontsize=14)
plt.show()
```



In [4]:
```python
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.linspace(-10, 10, 100)
z1 = 1/(1 + np.exp(-0.3*x))
z2 = 1/(1 + np.exp(-0.5*x))
z3 = 1/(1 + np.exp(-x))
plt.rcParams.update({'font.size': 16})
fig = plt.figure(figsize=(10, 5))
plt.plot(x, z1)
plt.plot(x, z2)
plt.plot(x, z3)
plt.xlabel("t")
plt.ylabel("Sigmoide(t)")

plt.show()
```



In [5]:
```python
t = np.linspace(-10, 10, 100)
sig = 1 / (1 + np.exp(-t))
plt.figure(figsize=(9, 3))
plt.plot([-10, 10], [0, 0], "k-")
plt.plot([-10, 10], [0.5, 0.5], "k:")
plt.plot([-10, 10], [1, 1], "k:")
plt.plot([0, 0], [-1.1, 1.1], "k-")
plt.plot(t, sig, "b-", linewidth=2, label=r"$\sigma(t) = \frac{1}{1 + e^{-t}}$")
plt.xlabel("t")
plt.legend(loc="upper left", fontsize=20)
plt.axis([-10, 10, -0.1, 1.1])
#save_fig("logistic_function_plot")
plt.show()
```



Excercice 1: Régression logistique binaire

1 - Lecture des données

In [6]:
```python
from sklearn import datasets
iris = datasets.load_iris()
list(iris.keys())
#print(iris.DESCR)
```

Out[6]:
```
['data',
 'target',
 'frame',
 'target_names',
 'DESCR',
 'feature_names',
 'filename',
 'data_module']
```

In [7]:
```python
# Afficher les différentes caractéristiques
display(iris.feature_names)
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [8]:
```python
# Afficher les différentes classes
display(iris.target_names)
```

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

In [9]:
```python
# Afficher l'encodage utilisé
display(iris.target)
```

```
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, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

2- Codage des données

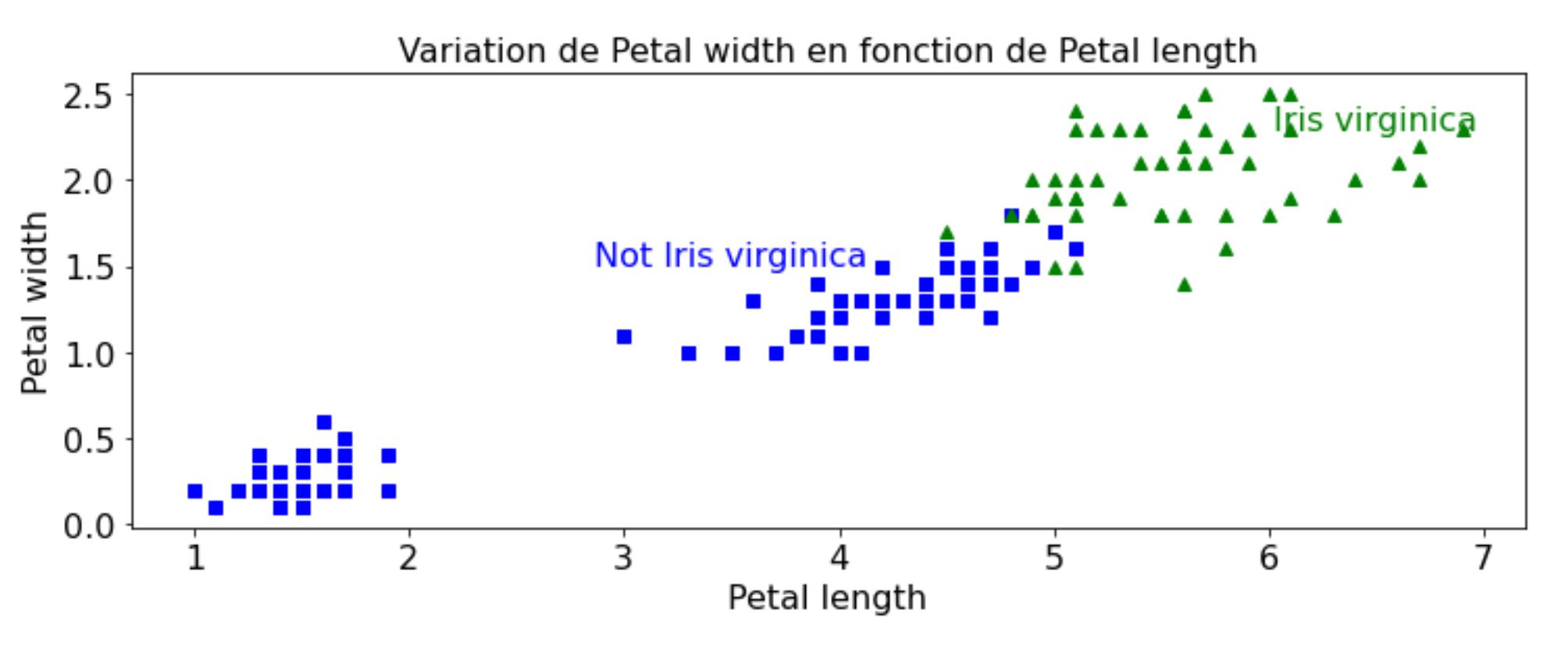
In [10]:
```python
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.int)
```

```
/var/folders/lg/y19wscx13n7g_vwjxll3gd3m0000gn/T/ipykernel_5811/1199862682.py:2: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  y = (iris["target"] == 2).astype(np.int)
```

3- Représentation de la dispersion des données

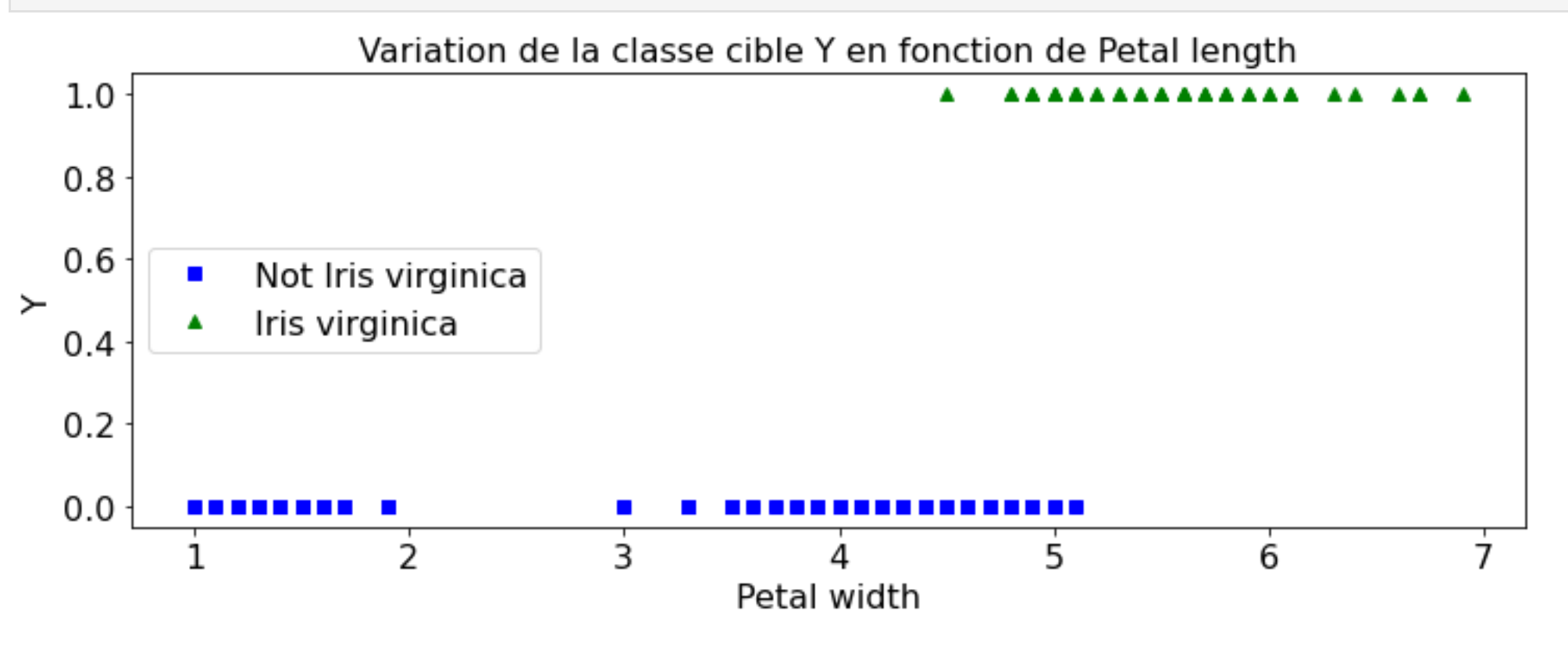
In [11]:
```python
plt.figure(figsize=(12, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs")
plt.plot(X[y==1, 0], X[y==1, 1], "g^")
plt.xlabel("Petal length", fontsize=16)
plt.ylabel("Petal width", fontsize=16)
plt.title("Variation de Petal width en fonction de Petal length", fontsize=16)
plt.text(3.5, 1.5, "Not Iris virginica", fontsize=16, color="b", ha="center")
plt.text(6.5, 2.3, "Iris virginica", fontsize=16, color="g", ha="center")
```

Out[11]:
```
Text(6.5, 2.3, 'Iris virginica')
```



4- Représentation de la dispersion en fonction de la variable cible

In [12]:
```python
plt.figure(figsize=(12,4))
plt.plot(X[y==0,0], y[y==0], "bs", label="Not Iris virginica")
plt.plot(X[y==1,0], y[y==1], "g^", label="Iris virginica")
plt.ylabel("Y", fontsize=16)
plt.xlabel("Petal width", fontsize=16)
plt.title("Variation de la classe cible Y en fonction de Petal length", fontsize=16)
plt.legend(loc="center left", fontsize=16)
plt.show()
```



5- Entrainement d'un modèle de régression logistique en se basant sur petal width

In [13]:
```python
# On considère la variable petal width
# On considère la détection de la classe virginica
# On procède donc par un codage 1 if Iris virginica, else 0

X = iris["data"][:, 3:]
y = (iris["target"] == 2).astype(np.int)
```

```
/var/folders/lg/y19wscx13n7g_vwjxll3gd3m0000gn/T/ipykernel_5811/1349082317.py:6: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  y = (iris["target"] == 2).astype(np.int)
```

In [14]:
```python
from sklearn.linear_model import LogisticRegression
# instantiate the model
log_reg = LogisticRegression()
# fit the model with data
log_reg.fit(X, y)
```

Out[14]:
```
LogisticRegression()
```

6- Paramètres du modèles

On a une seule varaible caractéristique, on a donc deux paramètres θ_0 et θ_1

In [15]:
```python
log_reg.coef_
```

Out[15]:
```
array([[4.3330846]])
```

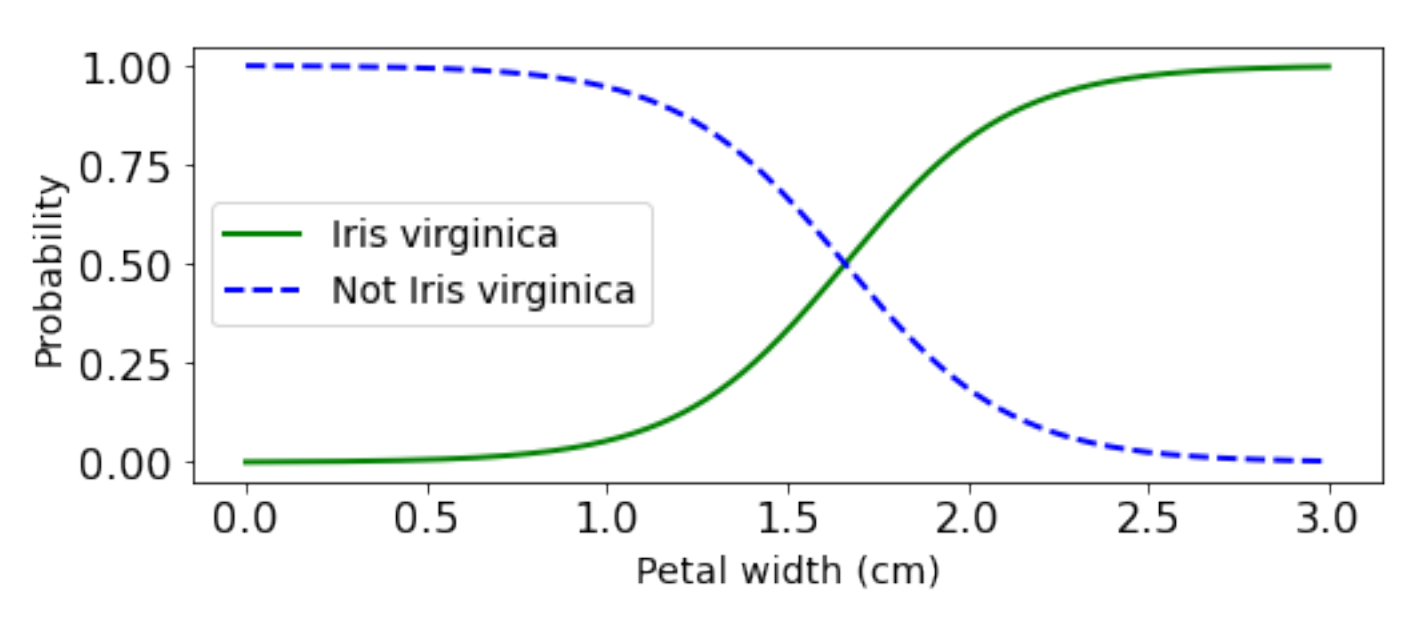
In [16]:
```python
log_reg.intercept_
```

Out[16]:
```
array([-7.1947083])
```

7- Représentation des frontières de décision

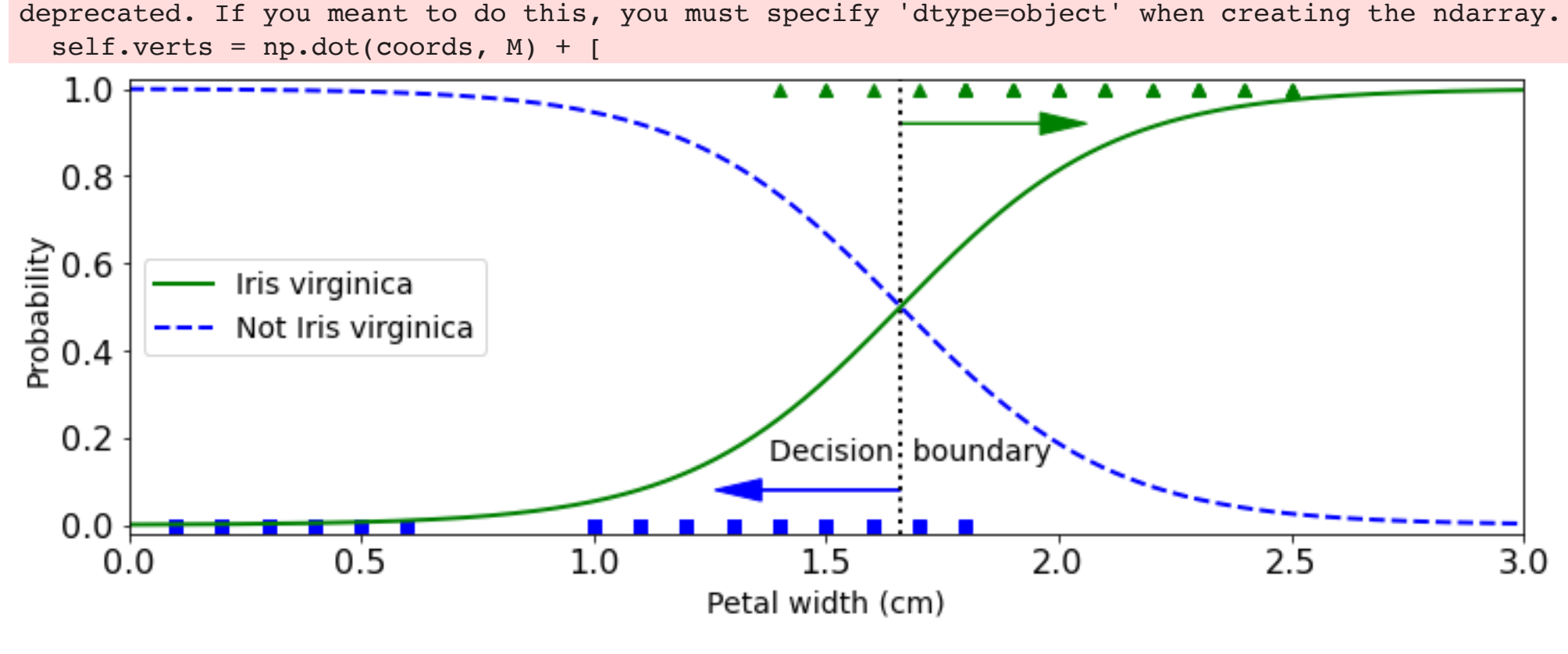
In [17]:
```python
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.figure(figsize=(8, 3))
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2, label="Not Iris virginica")
plt.xlabel("Petal width (cm)", fontsize=14)
plt.ylabel("Probability", fontsize=14)
plt.legend(loc="center left", fontsize=14)
```

Out[17]:
```
<matplotlib.legend.Legend at 0x11c763b80>
```

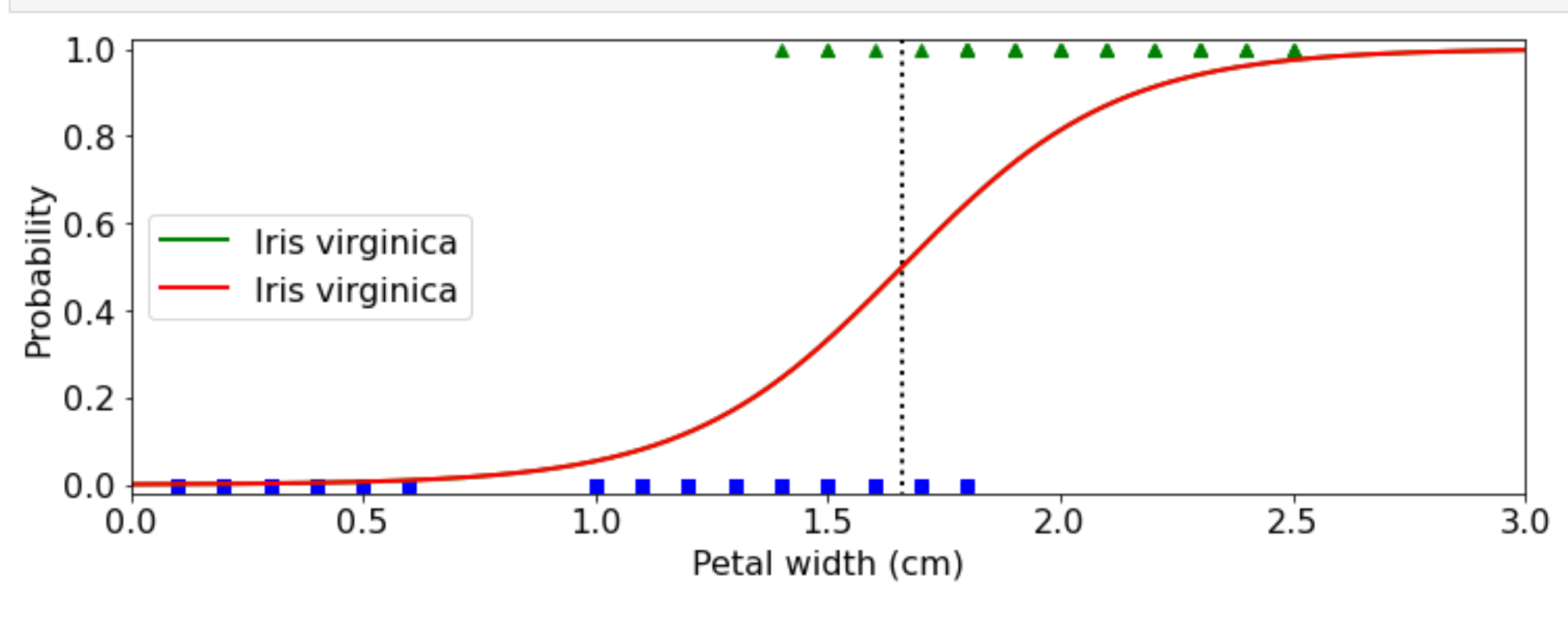


In [18]:
```python
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]

plt.figure(figsize=(12,4))
plt.plot(X[y==0], y[y==0], "bs")
plt.plot(X[y==1], y[y==1], "g^")
plt.plot([decision_boundary, decision_boundary], [-1, 2], "k:", linewidth=2)
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2, label="Not Iris virginica")
plt.text(decision_boundary+0.02, 0.15, "Decision  boundary", fontsize=14, color="k", ha="center")
plt.arrow(decision_boundary, 0.08, -0.3, 0, head_width=0.05, head_length=0.1, fc='b', ec='b')
plt.arrow(decision_boundary, 0.92, 0.3, 0, head_width=0.05, head_length=0.1, fc='g', ec='g')
plt.xlabel("Petal width (cm)", fontsize=14)
plt.ylabel("Probability", fontsize=14)
plt.legend(loc="center left", fontsize=14)
plt.axis([0, 3, -0.02, 1.02])
#save_fig("logistic_regression_plot")
plt.show()
```

```
/Users/nmezghani/opt/anaconda3/lib/python3.8/site-packages/matplotlib/patches.py:1444: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  self.verts = np.dot(coords, M) + [
```



In [19]:
```python
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]

x1=  log_reg.intercept_+log_reg.coef_*X_new
z1 = 1/(1 + np.exp(-x1))


plt.figure(figsize=(12, 4))
plt.plot(X[y==0], y[y==0], "bs")
plt.plot(X[y==1], y[y==1], "g^")
plt.plot([decision_boundary, decision_boundary], [-1, 2], "k:", linewidth=2)
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica")
plt.plot(X_new, z1, "r-", linewidth=2, label="Iris virginica")
#plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2, label="Not Iris virginica")
#plt.text(decision_boundary+0.02, 0.15, "Decision  boundary", fontsize=14, color="k", ha="center")
#plt.arrow(decision_boundary, 0.08, -0.3, 0, head_width=0.05, head_length=0.1, fc='b', ec='b')
#plt.arrow(decision_boundary, 0.92, 0.3, 0, head_width=0.05, head_length=0.1, fc='g', ec='g')
plt.xlabel("Petal width (cm)", fontsize=16)
plt.ylabel("Probability", fontsize=16)
plt.legend(loc="center left", fontsize=16)
plt.axis([0, 3, -0.02, 1.02])
#save_fig("logistic_regression_plot")
plt.show()
```

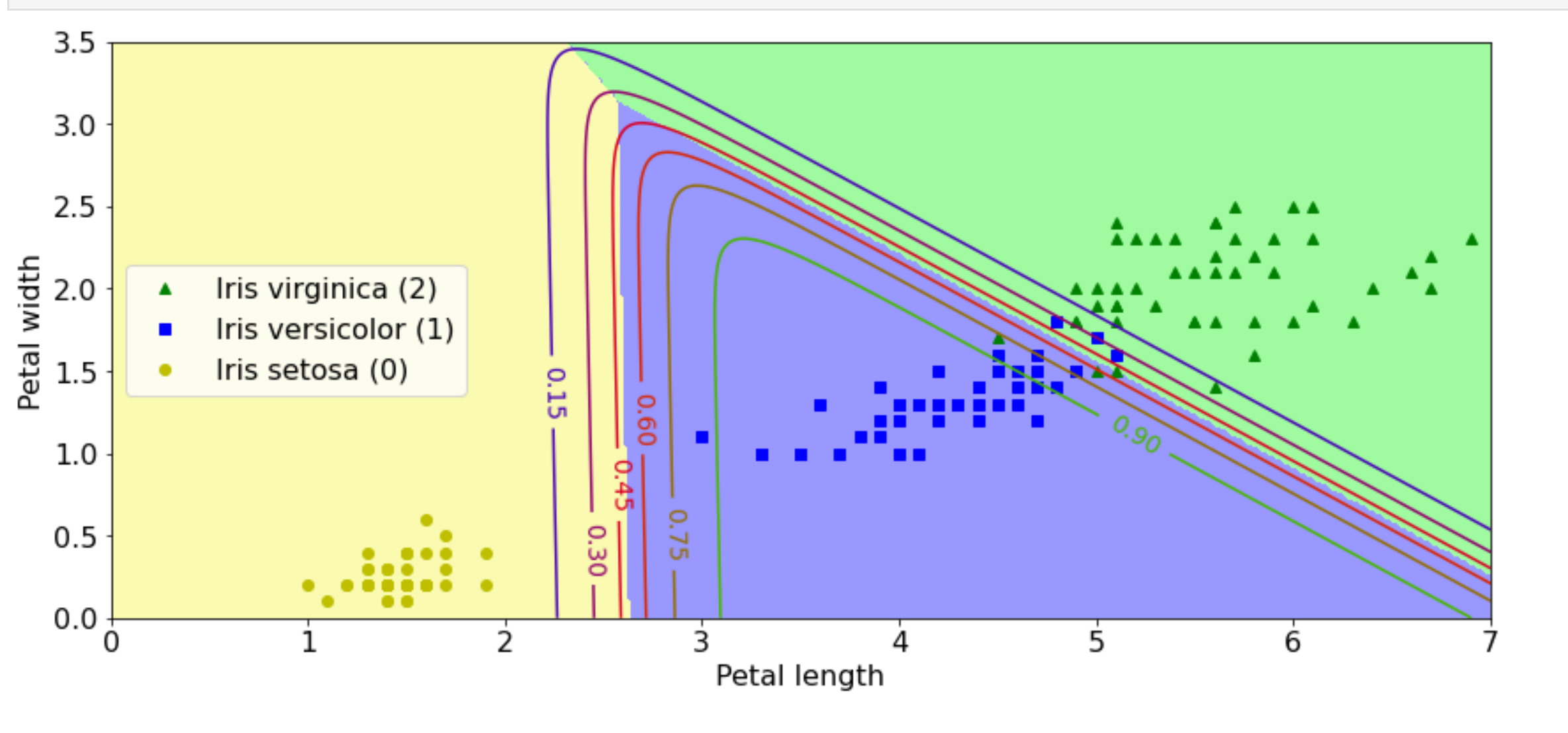


Excecice 2: Regression Softmax

In [20]:
```python
# La regression logistique peut être transformée
# en regression softmax en donnant à l'hyperparamètre
# multi_class la valeur "multinomial"

from sklearn.linear_model import LogisticRegression
X = iris["data"][:, (2, 3)]
y = iris["target"]
softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10, random_state=42)
softmax_reg.fit(X, y)
```

Out[20]:
```
LogisticRegression(C=10, multi_class='multinomial', random_state=42)
```

In [21]:
```python
# les frontières de décision entre les classes prises
# deux à deux sont linéaires.
# La figure présente les probabilités pour la classe
# Iris versicolor (1), représentées par des courbes de probabilités

x0, x1 = np.meshgrid(
        np.linspace(0, 8, 500).reshape(-1, 1),
        np.linspace(0, 3.5, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]

y_proba = softmax_reg.predict_proba(X_new)
y_predict = softmax_reg.predict(X_new)

zz1 = y_proba[:, 1].reshape(x0.shape)
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(14, 6))
plt.plot(X[y==2, 0], X[y==2, 1], "g^", label="Iris virginica (2)")
plt.plot(X[y==1, 0], X[y==1, 1], "bs", label="Iris versicolor (1)")
plt.plot(X[y==0, 0], X[y==0, 1], "yo", label="Iris setosa (0)")

from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])

plt.contourf(x0, x1, zz, cmap=custom_cmap)
contour = plt.contour(x0, x1, zz1, cmap=plt.cm.brg)
plt.clabel(contour, inline=1, fontsize=14)
plt.xlabel("Petal length", fontsize=16)
plt.ylabel("Petal width", fontsize=16)
plt.legend(loc="center left", fontsize=16)
plt.axis([0, 7, 0, 3.5])
plt.show()
```



Prédiction de la classe d'apatenance d'observations

In [22]:
```python
softmax_reg.predict([[5, 2],[4, 1], [1, 2], [7, 3], [7, 0.5]])
```

Out[22]:
```
array([2, 1, 0, 2, 2])
```

Calcul de probabilité d'appartenance de l'observation [5,2]

In [23]:
```python
# On peut calcluer la probabilité d'appartenance de
# l'échantillon [5,2]. On remarque que la valeur la plus élevé
# est pour la classe 2.
softmax_reg.predict_proba([[5, 2]])
```

Out[23]:
```
array([[6.38014896e-07, 5.74929995e-02, 9.42506362e-01]])
```

In [24]:
```python
# On peut déterminer la classe d'appartenance en
# utilisant la fonction argmax également
softmax_reg.predict_proba([[5, 2]]).argmax()
```

Out[24]:
```
2
```

In [ ]: