CS4487 - Machine Learning

Lecture 3a - Linear Classifiers

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

1. Discriminative linear classifiers
2. Logistic regression
3. Support vector machines (SVM)

Classification with Generative Model

- Steps to build a classifier
  1. Collect training data (features $\mathbf{x}$ and class labels y)
  2. Learn class-conditional distribution (CCD), $p(\mathbf{x}|y)$.
  3. Use Bayes' rule to calculate class probability, $p(y|\mathbf{x})$.
- **Note:** the data is used to learn the CCD -- the classifier is secondary.
  - Density estimation is an "ill-posed" problem -- which density to use? how much data is needed?

- Advice from Vladimir Vapnik (inventor of SVM):

> When solving a problem, try to avoid solving a more general problem as an intermediate step.

- **Discriminative solution**
  - Solve for the classifier $p(y|\mathbf{x})$ directly!

- Terminology
  - **"Discriminative"** - learn to directly discriminate the classes apart using the features.
  - **"Generative"** - learn model of how the features are generated from different classes.

Linear Classifier

- **Setup**
  - Observation (feature vectors) $\mathbf{x} \in \mathbb{R}^d$
  - Class $y \in \{-1, +1\}$
- **Goal**: given a feature vector $\mathbf{x}$, predict its class y.
  - Calculate a *linear function* of the feature vector $\mathbf{x}$.
    - $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{j=1}^{d} w_j x_j + b$
      - $\mathbf{w} \in \mathbb{R}^d$ are the weights of the linear function.
      - multiply each feature value with a weight, and then add together.
  - Predict from the value:
    - if $f(\mathbf{x}) > 0$ then predict Class $y = 1$
    - if $f(\mathbf{x}) < 0$ then predict Class $y = -1$
    - Equivalently, $y = \mathrm{sign}(f(\mathbf{x}))$

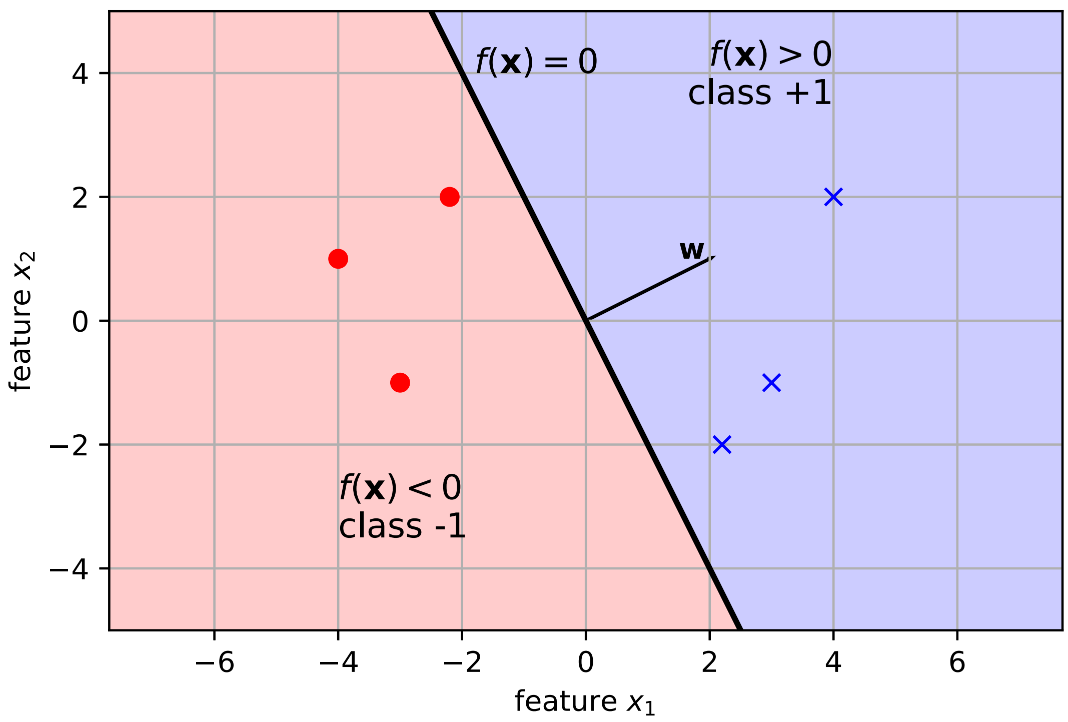
Geometric Interpretation

- The linear classifier separates the features space into 2 *half-spaces*
  - corresponding to feature values belonging to Class +1 and Class -1
  - the class boundary is normal to $\mathbf{w}$.
    - also called the *separating hyperplane*.

- Example: $\mathbf{w} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $b = 0$

In [4]:
```
linclass
```

Out[4]:



Separating Hyperplane

- In a d-dimensional feature space, the parameters are $\mathbf{w} \in \mathbb{R}^d$.
- The equation $\mathbf{w}^T\mathbf{x} + b = 0$ defines a $(d-1)$-dim. linear surface:
  - for $d = 2$, $\mathbf{w}$ defines a 1-D line.
  - for $d = 3$, $\mathbf{w}$ defines a 2-D plane.
  - ...
  - in general, we call it a hyperplane.

Learning the classifier

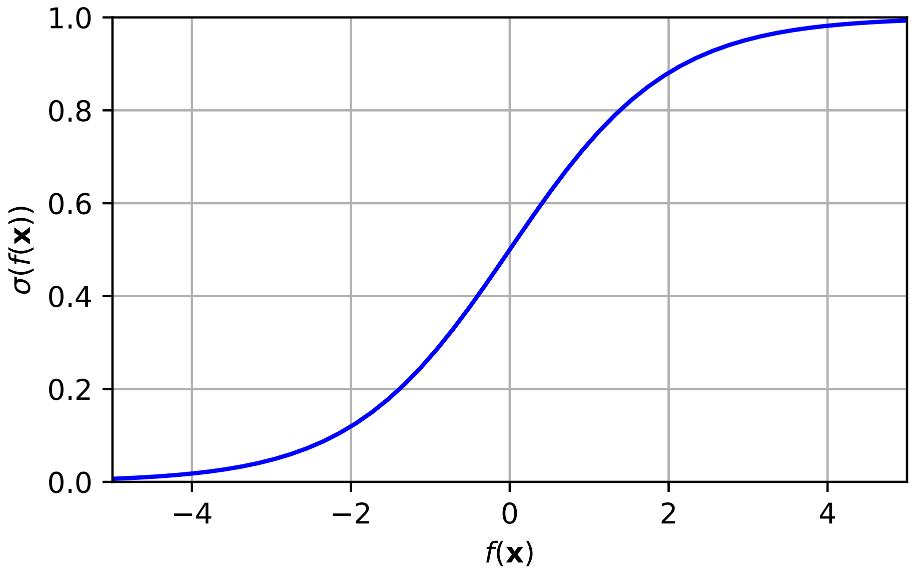
- How to set the classifier parameters $(\mathbf{w}, b)$?
  - Learn them from training data!
- Classifiers differ in the objectives used to learn the parameters $(\mathbf{w}, b)$.
  - We will look at two examples:
    - *logistic regression*
    - *support vector machine (SVM)*

Logistic regression

- Use a probabilistic approach
- Need to map the function values $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ to probability values between 0 and 1.
  - *sigmoid* function maps from real number to interval [0,1]
  - $\sigma(z) = \frac{1}{1+e^{-z}}$

In [6]:
```
sigmoidplot
```

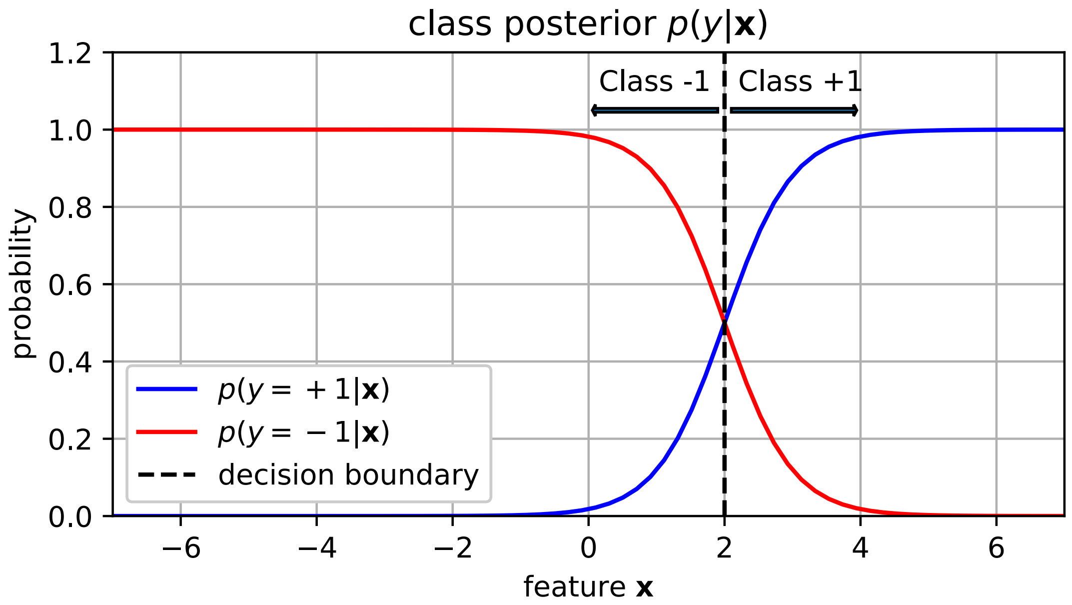
Out[6]:



- Given a feature vector x, the probability of a class is:
  - $p(y = +1|\mathbf{x}) = \sigma(f(\mathbf{x}))$
  - $p(y = -1|\mathbf{x}) = 1 - \sigma(f(\mathbf{x}))$
- Note: here we are directly modeling the class posterior probability!
  - not the class-conditional $p(\mathbf{x}|y)$

In [8]:
```
lrexample
```

Out[8]:



Learning the parameters

- Given training data $\{\mathbf{x}_i, y_i\}_{i=1}^N$, learn the function parameters $(\mathbf{w}, b)$ using maximum likelihood estimation.
- maximize the likelihood of the data $\{\mathbf{x}_i, y_i\}$:

$$(\mathbf{w}^*, b^*) = \operatorname*{argmax}_{\mathbf{w},b} \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i)$$

- to prevent *overfitting*, add a prior distribution on $\mathbf{w}$.
  - assume Gaussian distribution on $\mathbf{w}$ with variance $1/C$

$$(\mathbf{w}^*, b^*) = \operatorname*{argmax}_{\mathbf{w},b} \log p(\mathbf{w}) + \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i)$$

- Equivalently,

$$(\mathbf{w}^*, b^*) = \underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{C}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} \log(1 + \exp(-y_i(\mathbf{w}^T\mathbf{x}_i + b)))$$

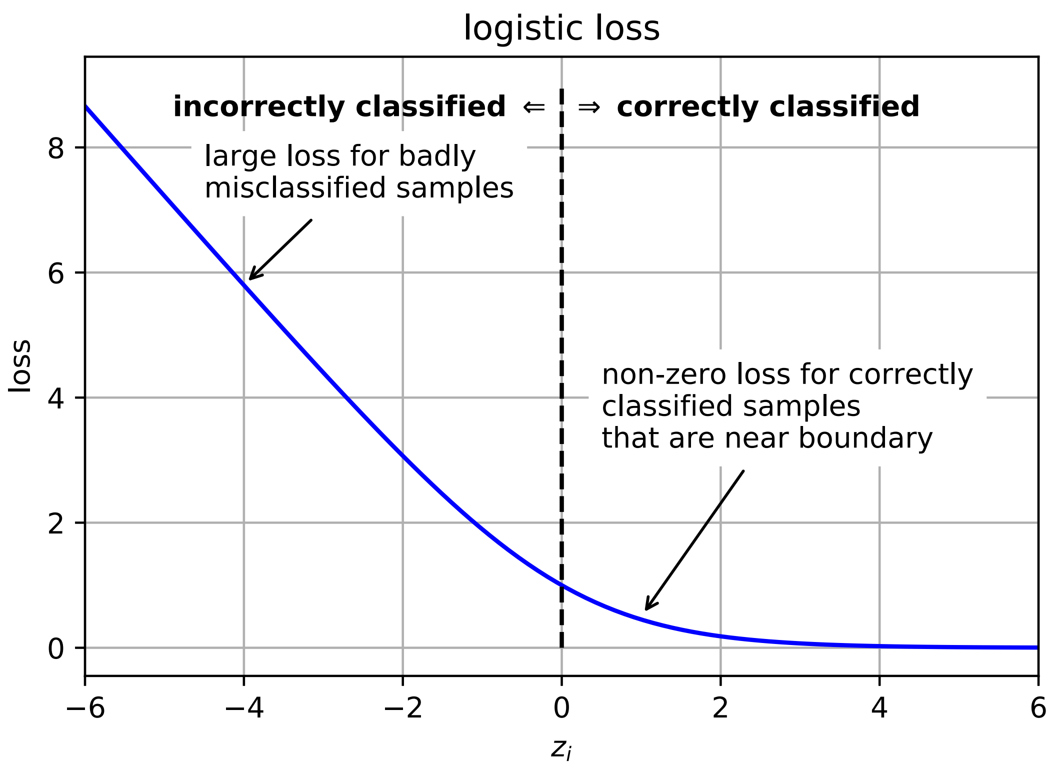
- the first term is the *regularization term*
  - Note: $\mathbf{w}^T\mathbf{w} = \sum_{j=1}^{d} w_j^2$
  - penalty term that keeps entries in $\mathbf{w}$ from getting too large.
  - C is the regularization *hyperparameter*
    - larger C value allow large values in $\mathbf{w}$.
    - smaller C value discourage large values in $\mathbf{w}$.

$$(\mathbf{w}^*, b^*) = \underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{C}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} \log(1 + \exp(-y_i(\mathbf{w}^T\mathbf{x}_i + b)))$$

- the second term is the *data-fit term*
  - wants to make the parameters $(\mathbf{w}, b)$ to well fit the data.
  - Define $z_i = y_i f(\mathbf{x}_i)$
    - Interesing observation:
      - $z_i > 0$ when sample $\mathbf{x}_i$ is classified correctly
      - $z_i < 0$ when sample $\mathbf{x}_i$ is classified incorrectly
      - $z_i = 0$ when sample is on classifier boundary
  - logistic loss function: $L(z_i) = \log(1 + \exp(-z_i))$

In [10]:
```
lossfig
```

Out[10]:



- **no closed-form solution**
  - use an iterative optimization algorithm to find the optimal solution
  - e.g. *gradient descent* - step downhill in each iteration.
    - $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{dE}{d\mathbf{w}}$
    - where E is the objective function
    - η is the *learning rate* (how far to step in each iteration).

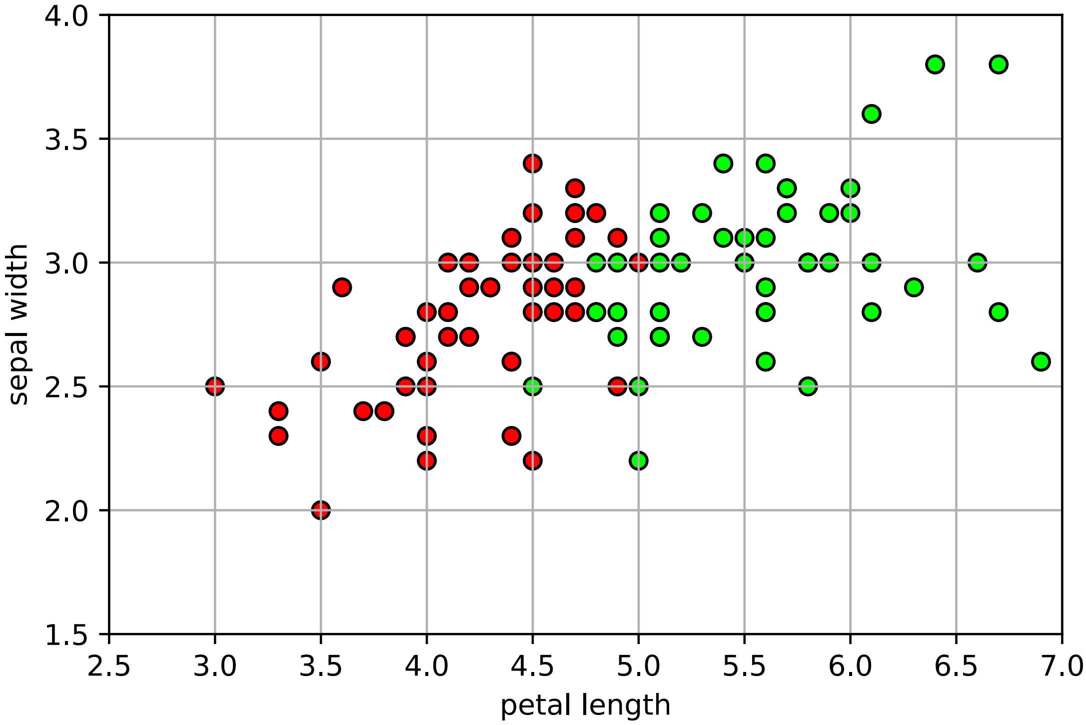
Example: Iris Data

In [11]:
```
# load iris data each row is (petal length, sepal width, class)
irisdata = loadtxt('iris2.csv', delimiter=',', skiprows=1)

X = irisdata[:,0:2]  # the first two columns are features (petal length, sepal w
idth)
Y = irisdata[:,2]    # the third column is the class label (versicolor=1, virgin
ica=2)
                     #  --> automaticaly mapped to (-1, +1) when training classi
fier

print(X.shape)
```

```
(100, 2)
```

In [12]:
```
# a colormap for making the scatter plot: class -1 will be red, class +1 will be
green
mycmap = matplotlib.colors.LinearSegmentedColormap.from_list('mycmap', ["#FF0000
", "#FFFFFF", "#00FF00"])

axbox = [2.5, 7, 1.5, 4] # common axis range

# a function for setting a common plot
def irisaxis(axbox):
    plt.xlabel('petal length'); plt.ylabel('sepal width')
    plt.axis(axbox); plt.grid(True)
```

In [13]:
```
# show the data
plt.figure()
plt.scatter(X[:,0], X[:,1], c=Y, cmap=mycmap, edgecolors='k')
irisaxis(axbox)
```



In [14]:
```
# randomly split data into 50% train and 50% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(X, Y,
  train_size=0.5, test_size=0.5, random_state=4487)

print(trainX.shape)
print(testX.shape)
```

```
(50, 2)
(50, 2)
```

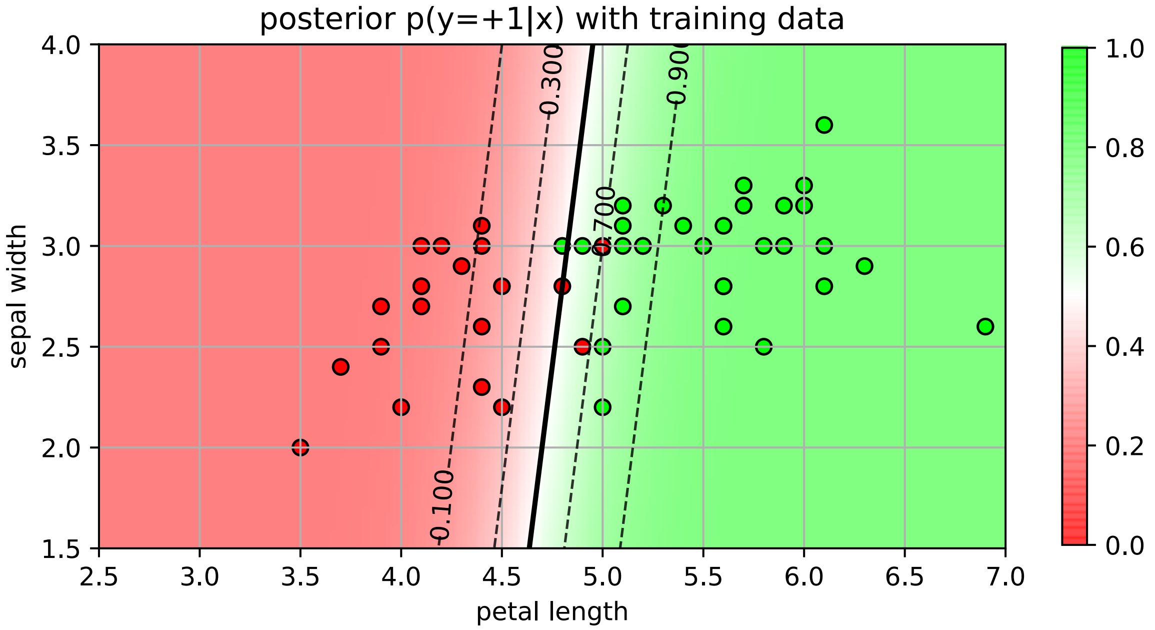
In [15]:
```
# learn logistic regression classifier
# (C is a regularization hyperparameter)
logreg = linear_model.LogisticRegression(C=100)
logreg.fit(trainX, trainY)

print("w =", logreg.coef_)
print("b =", logreg.intercept_)
```

```
w = [[ 4.87521863 -0.61512848]]
b = [-21.67874573]
```

- Equation:
  - $f(x) = (4.87 * petal\_length) - (0.62 * sepal\_width) - 21.68$
- Interpretation:
  - large petal length makes f(x) positive, so large petal length is associated with class +1.

In [19]:

```
# show the posterior and training data
plt.figure(figsize=(8,6))
plot_posterior(logreg, axbox, mycmap)
plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap, edgecolors='k')
plt.title('posterior p(y=+1|x) with training data');
```



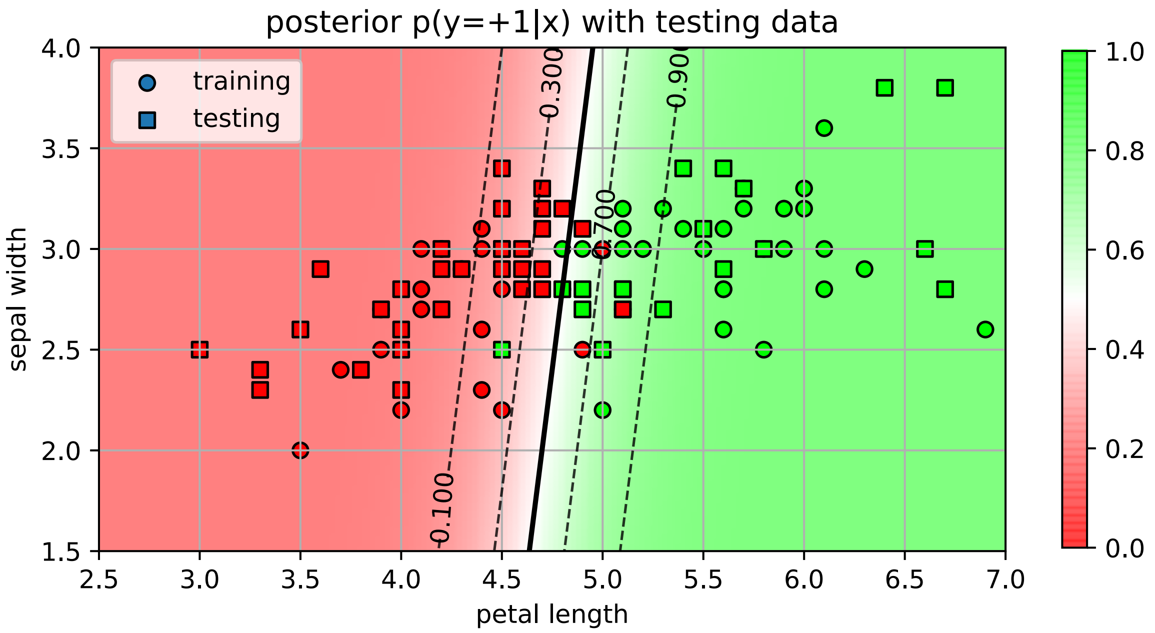
In [20]:

```
# predict from the model
predY = logreg.predict(testX)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("test accuracy =", acc)
```

```
test accuracy = 0.92
```

In [21]:

```
# show the posterior and training data
plt.figure(figsize=(8,6))
plot_posterior(logreg, axbox, mycmap)
plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap, marker="o", label="
training", edgecolors='k')
plt.scatter(testX[:,0], testX[:,1], c=testY, cmap=mycmap, marker="s", label="tes
ting", edgecolors='k')
plt.title('posterior p(y=+1|x) with testing data');
plt.legend(loc=0);
```

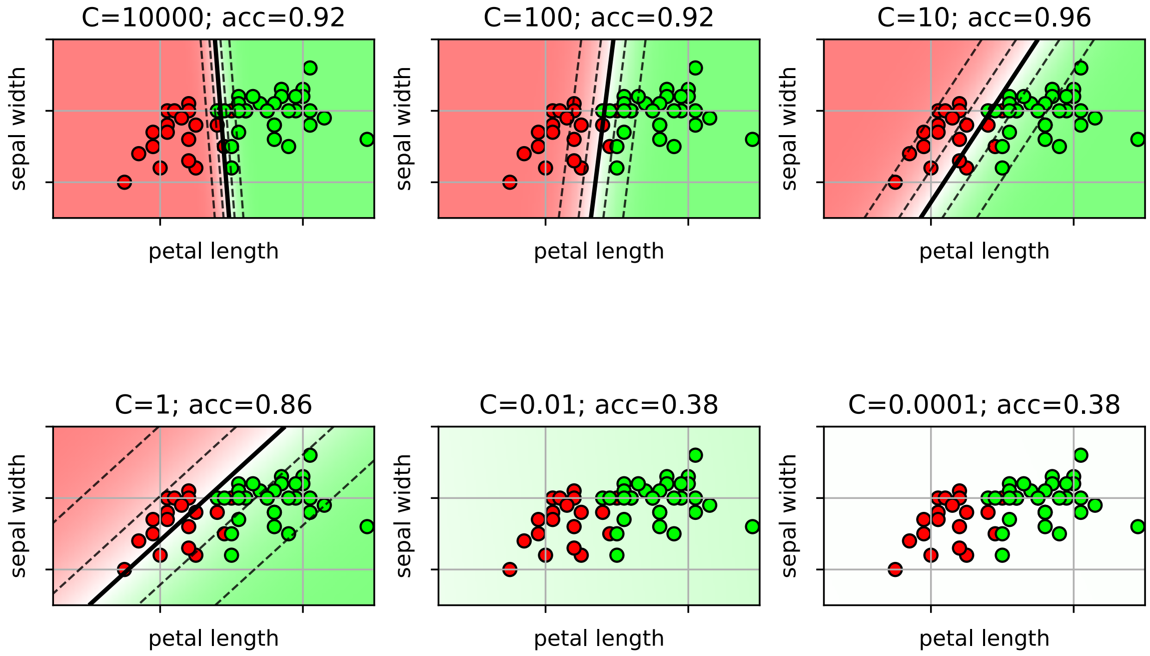


Selecting the regularization hyperparameter

- the regularization hyperparameter C has a big effect on the decision boundary and the accuracy.
- How to set the value of C?

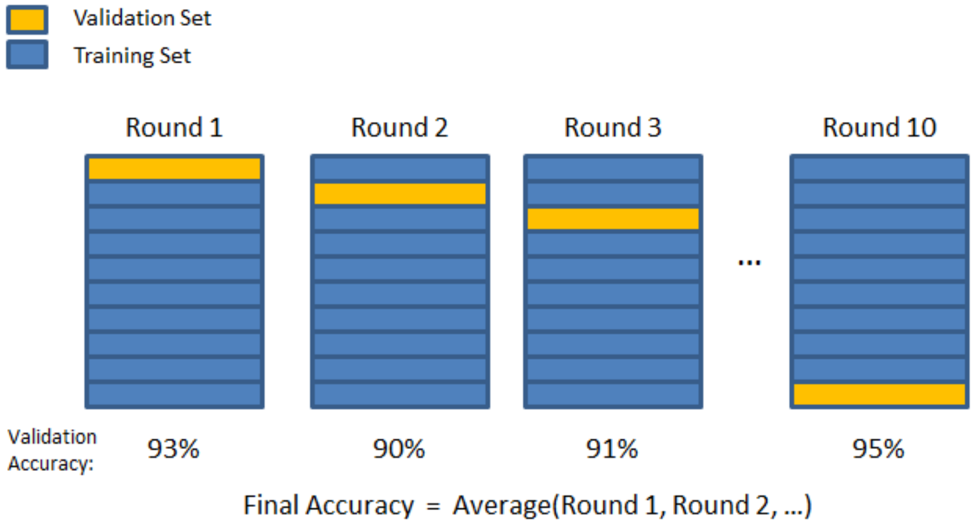
In [23]:
```
lrC
```

Out[23]:



Cross-validation

- Use *cross-validation* on the training set to select the best value of C.
- Run many experiments on the training set to see which parameters work on different versions of the data.
  - Split the data into batches of training and validation data.
  - Try a range of C values on each split.
  - Pick the value that works best over all splits.



- **Procedure**
  1. select a range of C values to try
  2. Repeat K times
     A. Split the training set into training data and validation data
     B. Learn a classifier for each value of C
     C. Record the accuracy on the validation data for each C
  3. Select the value of C that has the highest average accuracy over all K folds.
  4. Retrain the classifier using all data and the selected C.

- scikit-learn already has built-in `cross_validation` module (more later).
- for logistic regression, use *LogisticRegressionCV* class

In [24]:

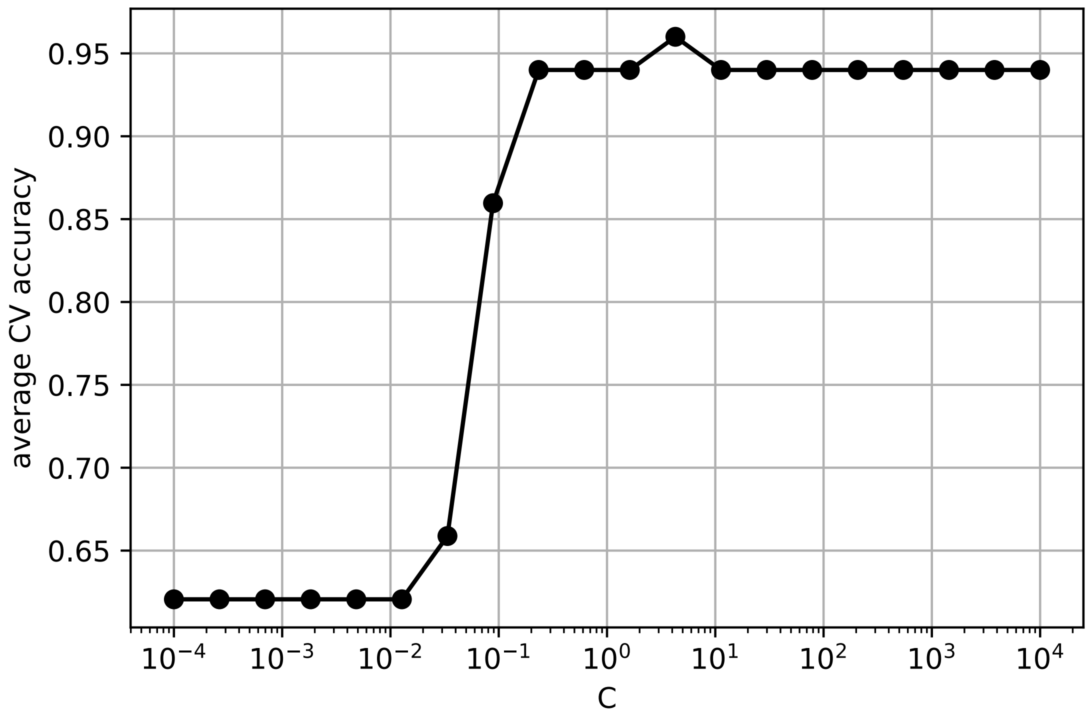
```
# learn logistic regression classifier usinc CV
#  Cs is an array of possible C values
#  cv is the number of folds
#  n_jobs is the number of parallel jobs to run (makes it faster)
#    -1 means use all cores
logreg = linear_model.LogisticRegressionCV(Cs=logspace(-4,4,20), cv=5, n_jobs=-1
)
logreg.fit(trainX, trainY)

print("w=", logreg.coef_)
print("b=", logreg.intercept_)

# predict from the model
predY = logreg.predict(testX)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("test accuracy=", acc)
```

```
w= [[4.61911642 0.72396452]]
b= [-24.24716674]
test accuracy= 0.9
```

Which C was selected?

In [25]:
```
print("C =", logreg.C_)

# calculate the average score for each C
avgscores = mean(logreg.scores_[2],0)  # 2 is the class label
plt.figure()
plt.semilogx(logreg.Cs_, avgscores, 'ko-')
plt.xlabel('C'); plt.ylabel('average CV accuracy')
plt.grid(True);
```

```
C = [4.2813324]
```



Multi-class classification

- So far, we have only learned a classifier for 2 classes (+1, -1)
  - called a **binary classifier**
- For more than 2 classes, split the problem up into several binary classifier problems.
  - **1-vs-rest**
    - *Training:* for each class, train a classifier for that class versus the other classes.
      - For example, if there are 3 classes, then train 3 binary classifiers: 1 vs {2,3}; 2 vs {1,3}; 3 vs {1,2}
    - *Prediction:* calculate probability for each binary classifier. Select the class with highest probability.

Example on 3-class Iris data

In [26]:

```
# load iris data each row is (petal length, sepal width, class)
irisdata = loadtxt('iris3.csv', delimiter=',', skiprows=1)

X = irisdata[:,0:2]  # the first two columns are features (petal length, sepal w
idth)
Y = irisdata[:,2]    # the third column is the class label (setosa=0, versicolor
=1, virginica=2)

print(X.shape)
```

```
(150, 2)
```

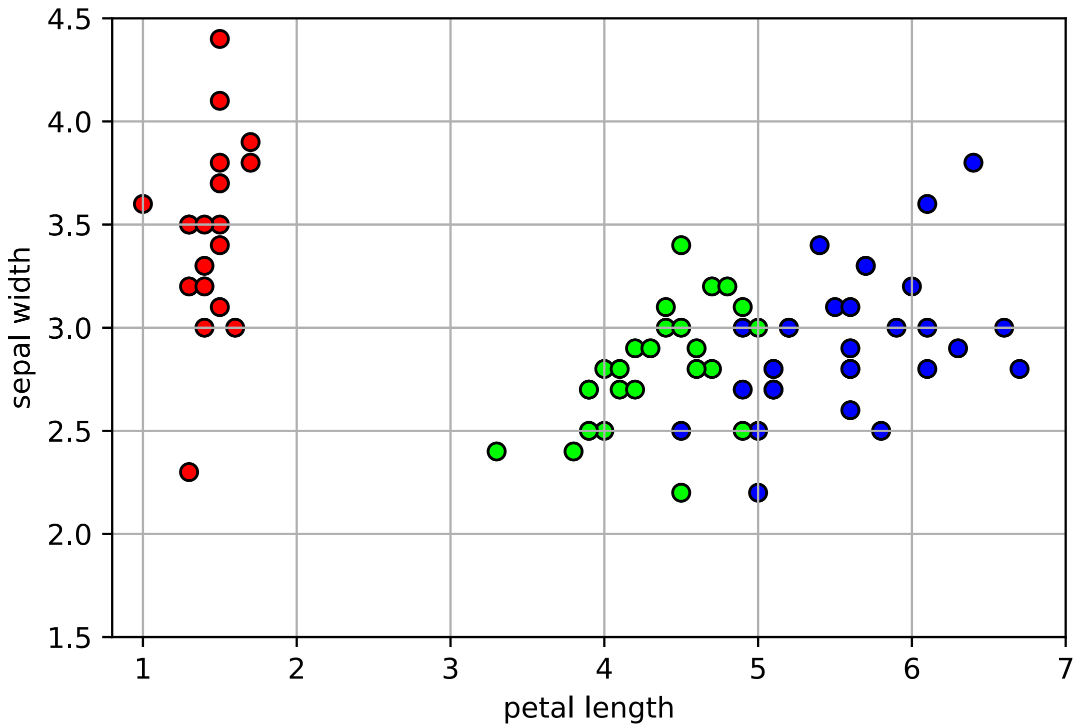
In [27]:

```
# randomly split data into 50% train and 50% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(X, Y,
  train_size=0.5, test_size=0.5, random_state=4487)

print(trainX.shape)
print(testX.shape)
```

```
(75, 2)
(75, 2)
```

In [28]:

```
# look at training data

axbox3 = [0.8, 7, 1.5, 4.5]
# make a colormap for viewing 3 classes
mycmap3 = matplotlib.colors.LinearSegmentedColormap.from_list('mycmap', ["#FF000
0", "#00FF00", "#0000FF"])

plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap3, edgecolors='k')
plt.axis(axbox3); plt.grid(True);
plt.xlabel('petal length'); plt.ylabel('sepal width');
```



In [29]:

```
# learn logistic regression classifier (one-vs-all)
mlogreg = linear_model.LogisticRegression(C=10)
mlogreg.fit(trainX, trainY)

# now contains 3 hyperplanes and 3 bias terms (one for each class)
print("w=", mlogreg.coef_)
print("b=", mlogreg.intercept_)

# predict from the model
predY = mlogreg.predict(testX)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("test accuracy=", acc)
```

```
w= [[-3.09131694  2.52132269]
 [ 0.06064355 -1.58022283]
 [ 3.35076433 -3.48981157]]
b= [ 0.73591801  3.79651516 -6.36532274]
test accuracy= 0.9733333333333334
```

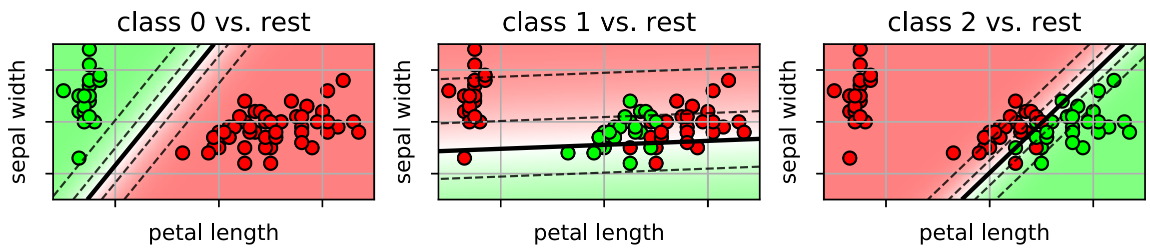
- the individual 1-vs-rest binary classifiers

In [31]:

```
print("w=", mlogreg.coef_)
print("b=", mlogreg.intercept_)

mlrfig
```

```
w= [[-3.09131694  2.52132269]
 [ 0.06064355 -1.58022283]
 [ 3.35076433 -3.48981157]]
b= [ 0.73591801  3.79651516 -6.36532274]
```

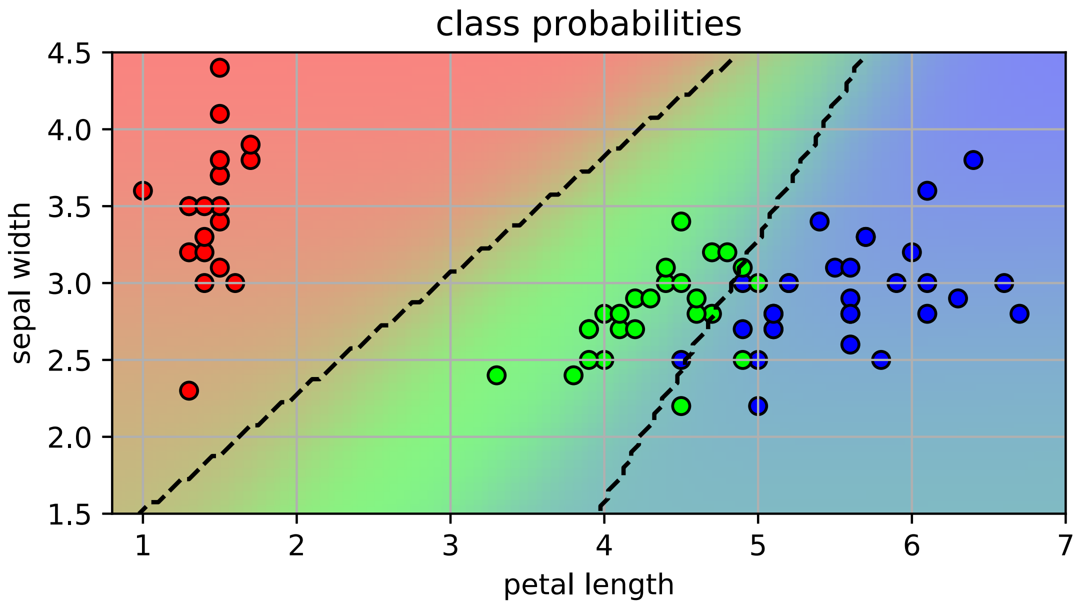
Out[31]:



- the final classifier, combining all 1 vs rest classifiers

In [33]:
```
lr3class
```

Out[33]:



Multiclass logistic regression

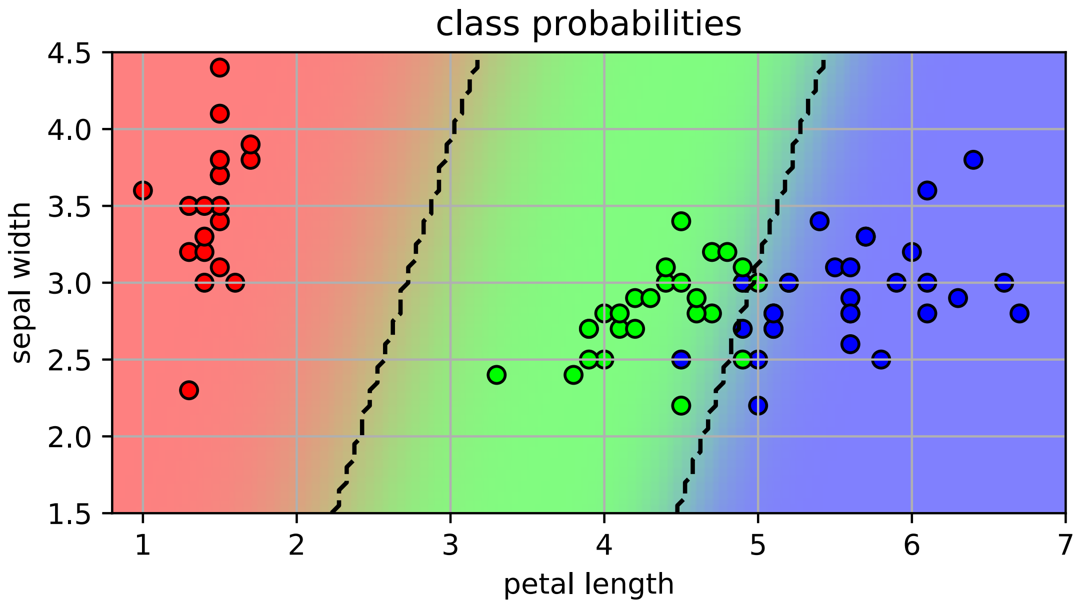
- Another way to get a multi-class classifier is to define a multi-class objective.
  - One weight vector $\mathbf{w}_c$ for each class c.
- Define probabilities with **softmax** function
  - analogous to sigmoid function for binary logistic regression.
  - $p(y=c|\mathbf{x}) = \frac{\exp(\mathbf{w}_c^T\mathbf{x})}{\exp(\mathbf{w}_1^T\mathbf{x})+\cdots+\exp(\mathbf{w}_K^T\mathbf{x})}$
  - The class with largest reponse of $\mathbf{w}_c^T\mathbf{x}$ will have the highest probability.
- Estimate the $\{\mathbf{w}_j\}$ parameters using MLE as before.

In [34]:
```
# learn logistic regression classifier
mlogreg = linear_model.LogisticRegression(C=10,
            multi_class='multinomial', solver='lbfgs')
            # use multi-class and corresponding solver
mlogreg.fit(trainX, trainY)

# now contains 3 hyperplanes and 3 bias terms (one for each class)
print("w=", mlogreg.coef_)
print("b=", mlogreg.intercept_)

# predict from the model
predY = mlogreg.predict(testX)

# calculate accuracy
acc = metrics.accuracy_score(testY, predY)
print("test accuracy=", acc)
```

```
w= [[-4.13092437  1.30718735]
 [-0.71717021  0.23609022]
 [ 4.84809458 -1.54327757]]
b= [ 11.46078594   5.40723484 -16.86802078]
test accuracy= 0.9733333333333334
```

In [36]:
```
lr3classm
```

Out[36]:

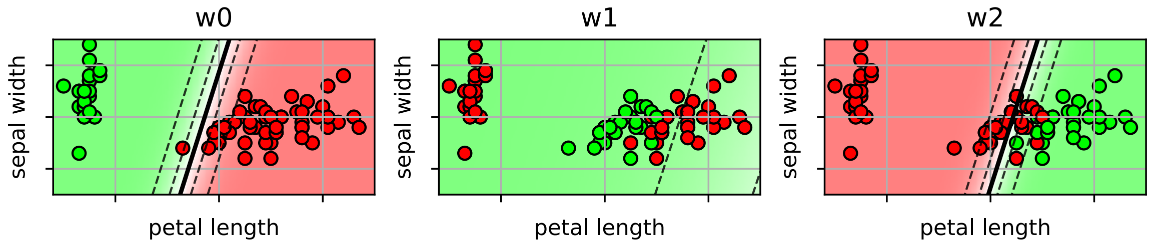


- individual weight vectors work together to partition the space

In [38]:
```
print("w=", mlogreg.coef_)
print("b=", mlogreg.intercept_)

lr3lvr
```

```
w= [[-4.13092437  1.30718735]
 [-0.71717021  0.23609022]
 [ 4.84809458 -1.54327757]]
b= [ 11.46078594   5.40723484 -16.86802078]
```

Out[38]:



Logistic Regression Summary

- **Classifier:**
  - linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
  - Given a feature vector $\mathbf{x}$, the probability of a class is:
    - $p(y = +1|\mathbf{x}) = \sigma(f(\mathbf{x}))$
    - $p(y = -1|\mathbf{x}) = 1 - \sigma(f(\mathbf{x}))$
    - *sigmoid* function: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - logistic loss function: $L(z) = \log(1 + \exp(-z))$
- **Training:**
  - Maximize the likelihood of the training data.
  - Use regularization to prevent overfitting.
    - Use cross-validation to pick the regularization hyperparameter C.

- **Classification:**
  - Given a new sample $\mathbf{x}^*$:
    - pick class with highest probability $p(y|\mathbf{x}^*)$:
      - $y^* = \begin{cases} +1, p(y = +1|\mathbf{x}^*) > p(y = -1|\mathbf{x}^*) \\ -1, \text{otherwise} \end{cases}$
    - alternatively, just use $f(\mathbf{x}^*)$
      - $y^* = \begin{cases} +1, f(\mathbf{x}^*) > 0 \\ -1, \text{otherwise} \end{cases} = \text{sign}(f(\mathbf{x}_*))$