CS4487 - Machine Learning

Lecture 3b - Support Vector Machines

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

1. Discriminative classifiers
2. Logistic regression
3. **Support vector machines**

Support vector machines

- With logistic regression we used a maximum-likelihood framework to learn the separating hyperplane.
- Let's consider a purely geometric approach...

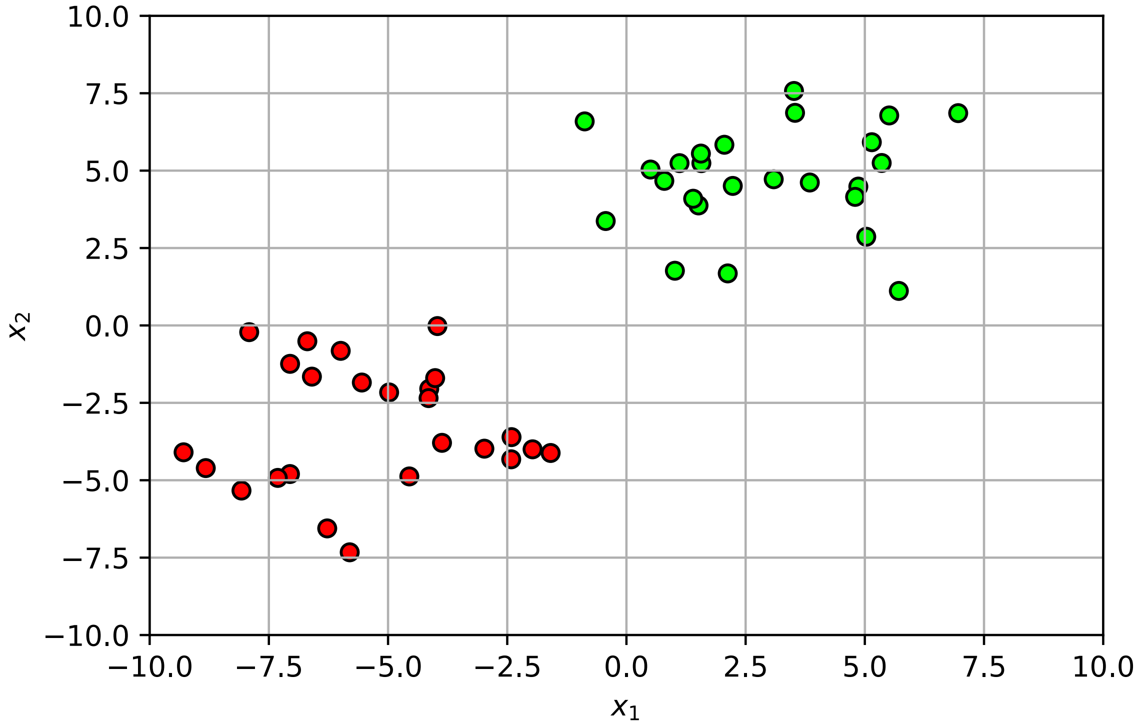
Linearly-Separable Data

- For now, assume the training data is *linearly separable*
  - the two classes in the training data can be separated by a line (hyperplane)

In [3]:

```
lsdatafig
```

Out[3]:



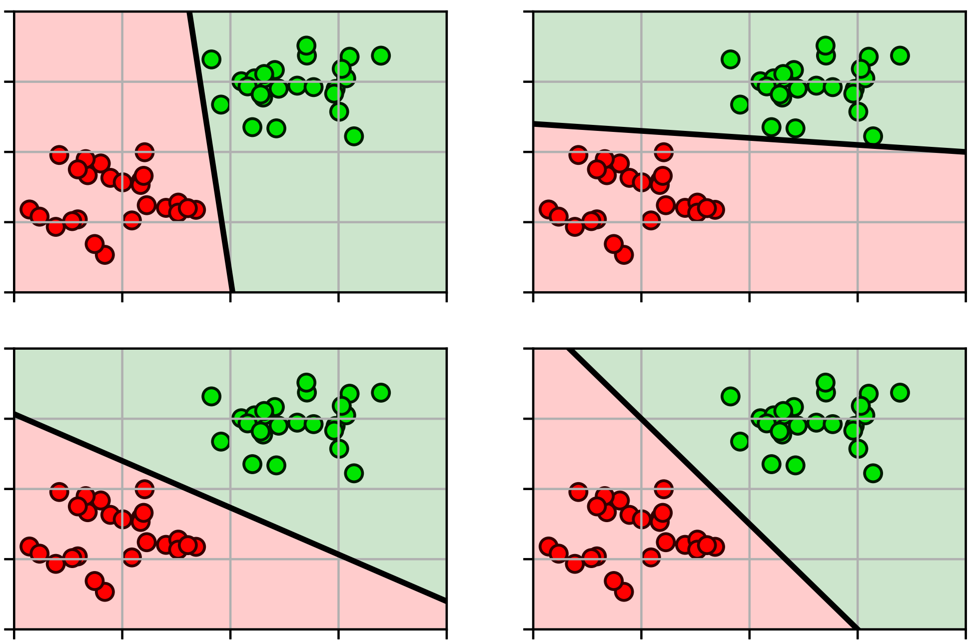
Which is the best separating line?

- there are many possible solutions...

In [6]:

```
seplinefig
```

Out[6]:

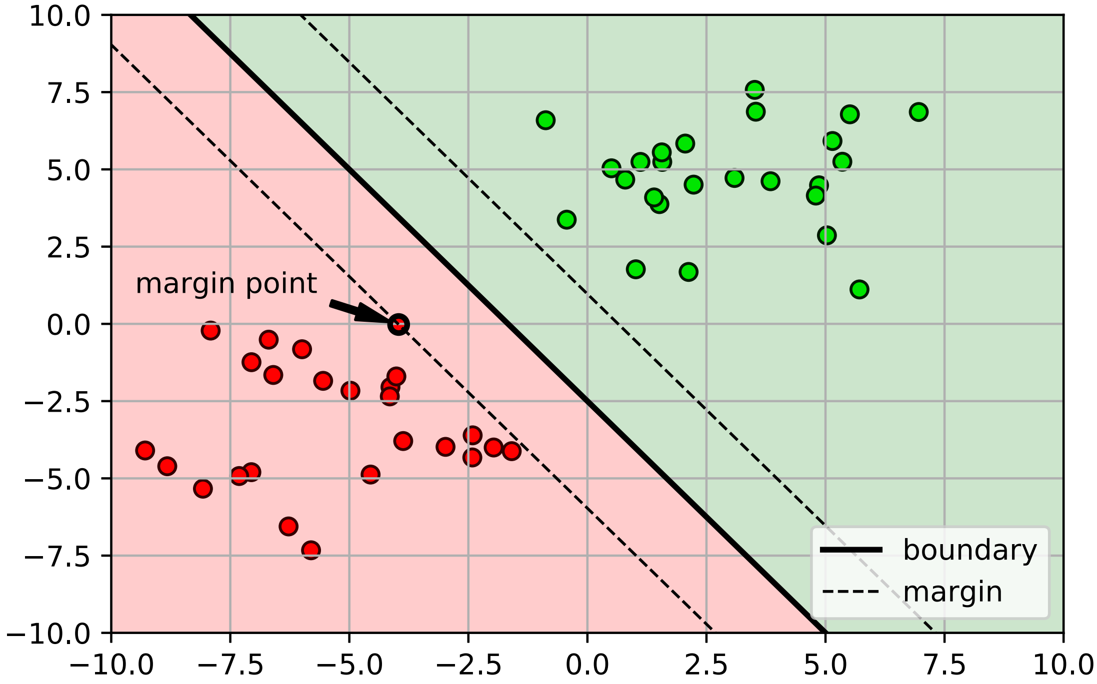


Maximum margin

- Define the space between the separating line and the closest point as the *margin*.
  - think of this space as the "amount of wiggle room" for accomodating errors in estimating $\mathbf{w}$.

In [9]:
```
margfig
```

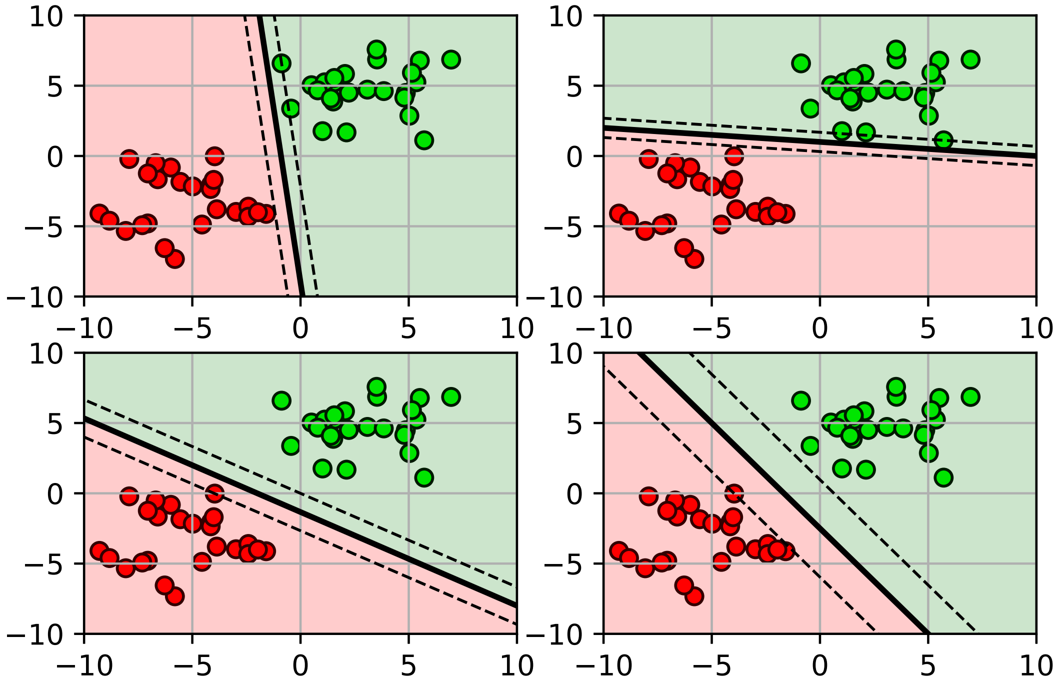
Out[9]:



- **Idea:** the best separating line is the one that *maximizes the margin*.
  - i.e., puts the most distance between the closest points and the decision boundary.

In [11]:
```
margfigs
```

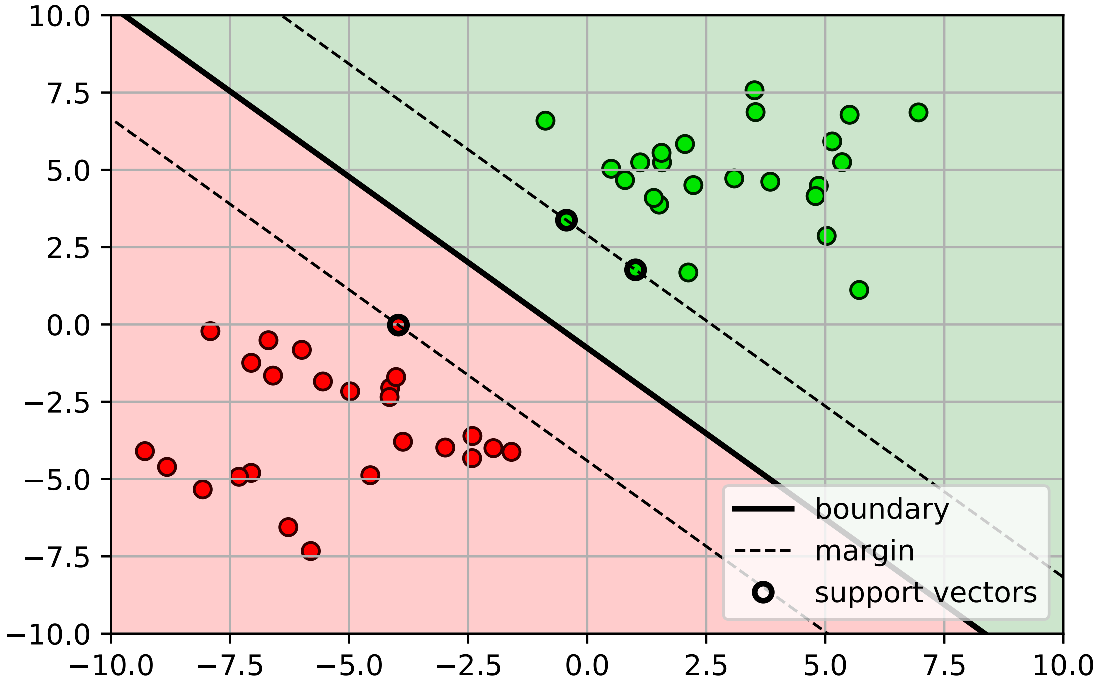
Out[11]:



- *the solution...*
  - by symmetry, there should be at least one margin point on each side of the boundary
  - the points on the margins are called the **support vectors**
    - the points support (define) the margin

In [13]:
```
maxmfig
```

Out[13]:



SVM Training

- given a training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$, optimize:

$$\operatorname*{argmin}_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad 1 \leq i \leq N$$

  - the objective minimizes the inverse of the margin distance, i.e., maximizes the margin.
  - the inequality constraints ensure that all points are either on or outside of the margin.
    - the margin is set to be distance of 1 from the boundary.

SVM Prediction

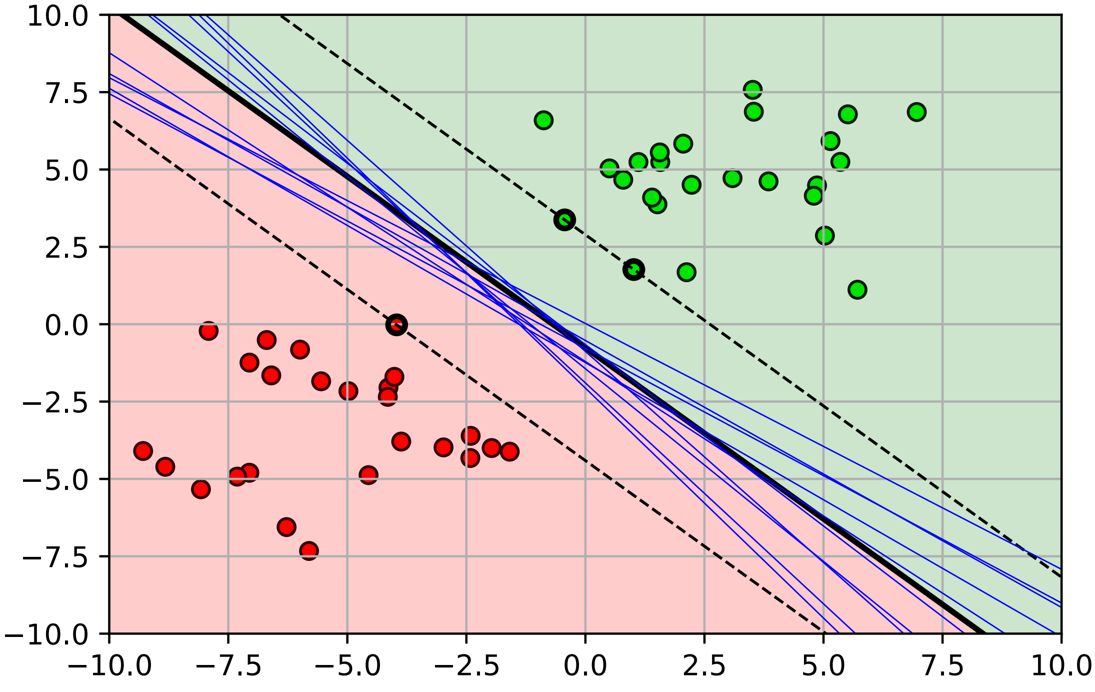
- given a new data point $\mathbf{x}_*$, use sign of linear function to predict class
  - $y_* = \text{sign}(\mathbf{w}^T\mathbf{x}_* + b)$

Why is maximizing the margin good?

- the true $\mathbf{w}$ is uncertain
  - maximizing the margin allows the most uncertainty (wiggle room) for $\mathbf{w}$, while keeping all the points correctly classified.

In [15]:
```
maxmfigw
```

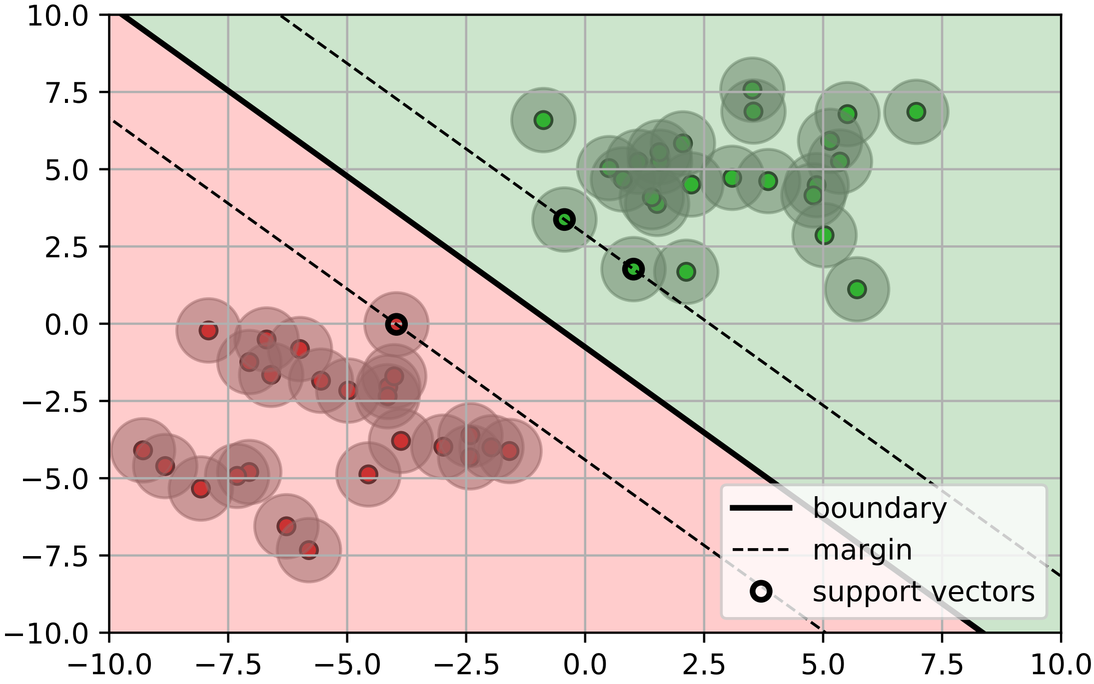
Out[15]:



- the data points are uncertain
  - maximizing the margin allows the most wiggle of the points, while keeping all the points correctly classified.

In [17]:
```
maxmfigp
```

Out[17]:

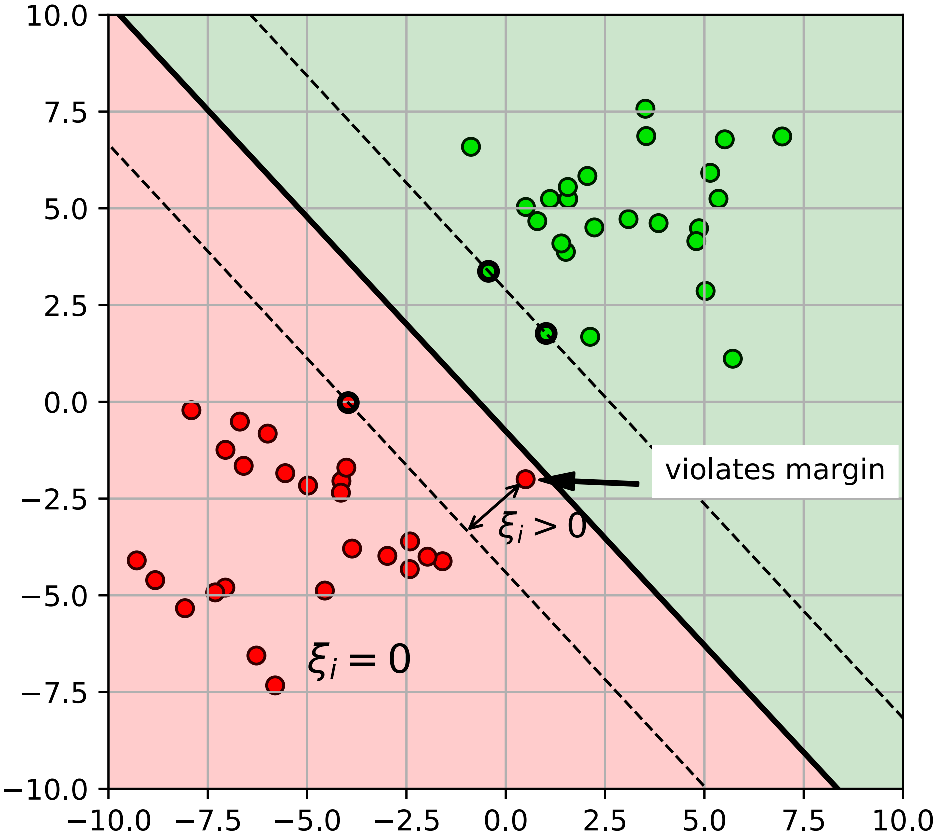


What about non-separable data?

- use the same linear classifier
  - allow some training samples to violate margin
    - i.e., are inside the margin (or even mis-classified)
  - Define "slack" variable $\xi_i \geq 0$
    - $\xi_i = 0$ means sample is outside of margin area (no slack)
    - $\xi_i > 0$ means sample is inside of margin area (slack)

In [19]:
```
slackfig
```

Out[19]:



- introduce a parameter C which is the penalty for each training sample that violates the margin.
  - smaller value means allow more violations (less penalty)
  - larger value means don't allow violations (more penalty)

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i$$

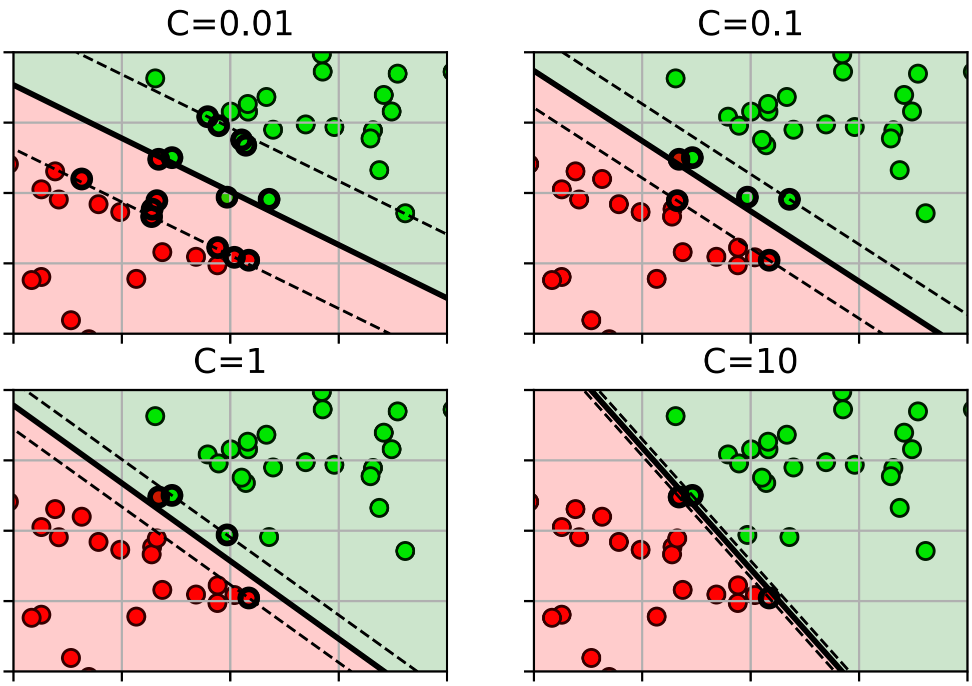
$$\text{s. t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad 1 \leq i \leq N$$

$$\xi_i \geq 0$$

In [21]:

```
Cmargfigs
```

Out[21]:



Loss function

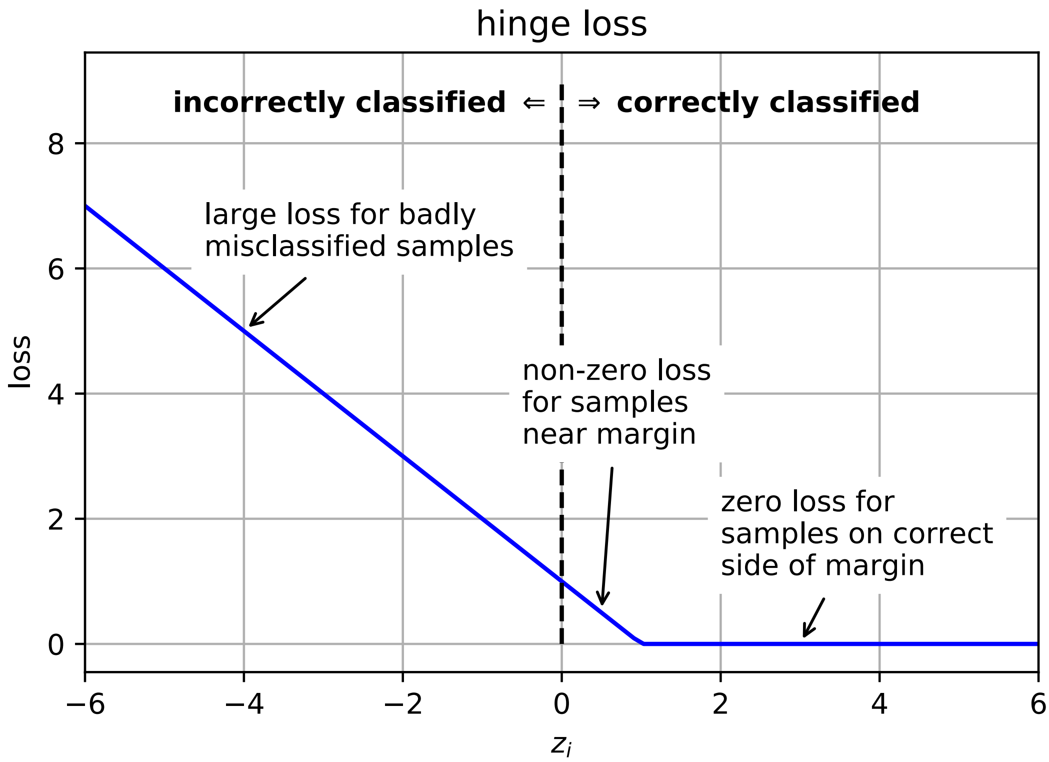
- After some massaging, the objective function is:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{C}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$

  - hinge loss function: $L(z_i) = \max(0, 1 - z_i)$
    - Note: $\max(a, b)$ returns whichever value (a or b) is largest.

In [23]:
```
lossfig
```

Out[23]:



Example: Iris Data

In [24]:
```
# 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)

print(X.shape)
```

```
(100, 2)
```

In [25]:
```
# 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 [26]:
```python
# fit the SVM using all the data and the best C
clf = svm.SVC(kernel='linear', C=2)
clf.fit(trainX, trainY)

# get line parameters
w = clf.coef_[0]
b = clf.intercept_[0]
print(w)
print(b)
```

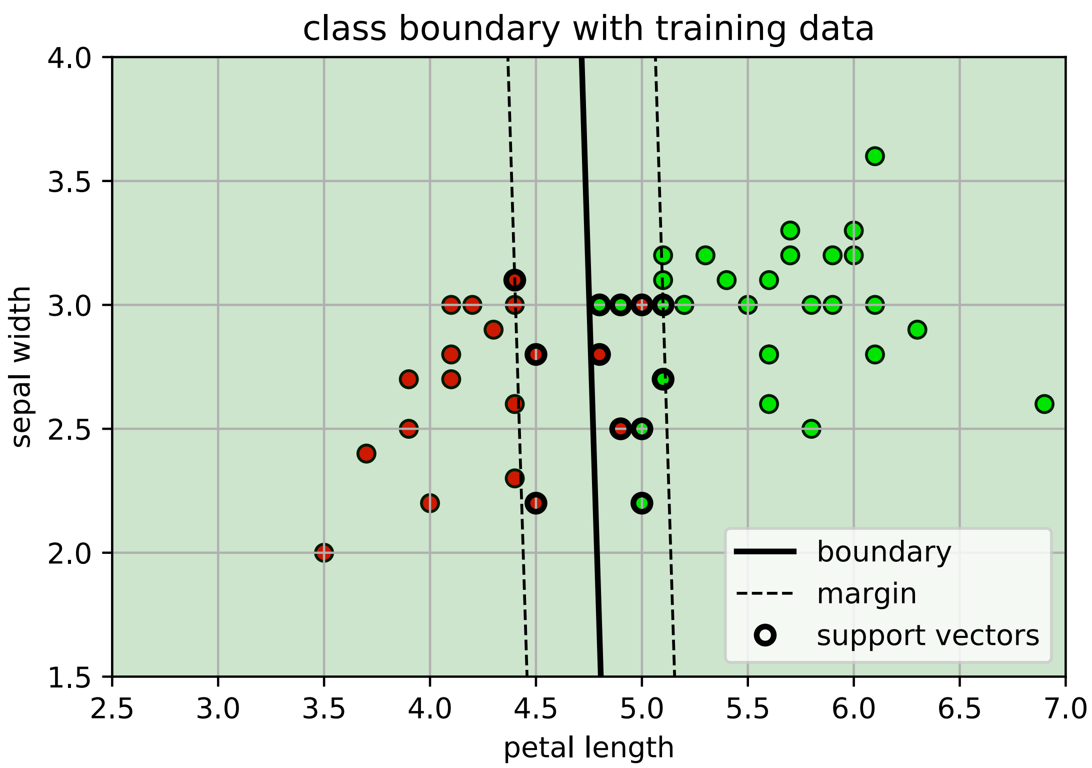
```
[2.87200943 0.10399865]
-13.95923965217775
```

In [27]:
```python
# indices of data points that are support vectors (on or inside the margin)
clf.support_
```

Out[27]:
```
array([ 0,  7, 12, 31, 36, 41, 13, 20, 22, 25, 33, 46], dtype=int32)
```

In [29]:
```python
svmfig
```

Out[29]:



- SVM doesn't have it's own dedicated cross-validation function
- Use the `GridSearchCV` to run cross-validation for a list of parameters
  - calculate average accuracy for each parameter
  - select parameter with highest accuracy, retrain model with all data
  - Speed up: each parameter can be trained/tested separately, specify number of parallel jobs using `n_jobs`

In [30]:
```
# setup the list of parameters to try
paramgrid = {'C': logspace(-3,3,13)}
print(paramgrid)

# setup the cross-validation object
# pass the SVM object, parameter grid, and number of CV folds
# set number of parallel jobs to -1 (use all cores)
svmcv = model_selection.GridSearchCV(svm.SVC(kernel='linear'), paramgrid, cv=5,
                                     n_jobs=-1, verbose=True)

# run cross-validation (train for each split)
svmcv.fit(trainX, trainY);
```

```
{'C': array([1.00000000e-03, 3.16227766e-03, 1.00000000e-02, 3.16227766e-02,
       1.00000000e-01, 3.16227766e-01, 1.00000000e+00, 3.16227766e+00,
       1.00000000e+01, 3.16227766e+01, 1.00000000e+02, 3.16227766e+02,
       1.00000000e+03])}
Fitting 5 folds for each of 13 candidates, totalling 65 fits

[Parallel(n_jobs=-1)]: Done  65 out of  65 | elapsed:    0.2s finished
```

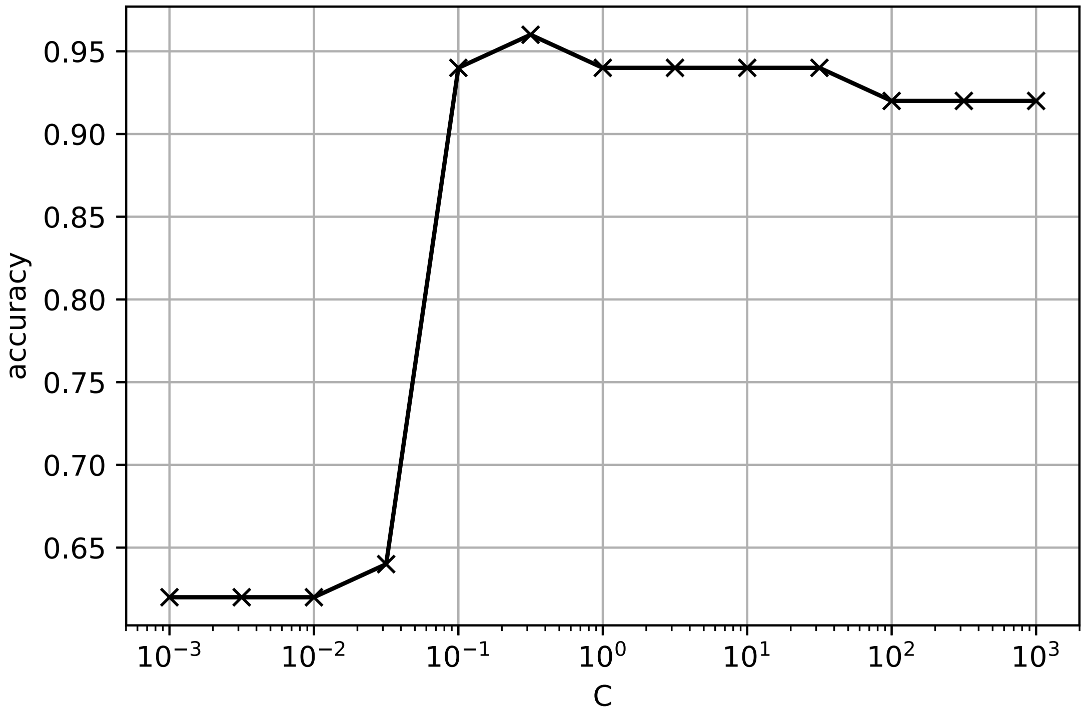
In [31]:
```
# show the test error for each parameter set
for m,p in zip(svmcv.cv_results_['mean_test_score'], svmcv.cv_results_['params']
):
    print("mean={:.4f} {}".format(m,p))
```

```
mean=0.6200 {'C': 0.001}
mean=0.6200 {'C': 0.0031622776601683794}
mean=0.6200 {'C': 0.01}
mean=0.6400 {'C': 0.03162277660168379}
mean=0.9400 {'C': 0.1}
mean=0.9600 {'C': 0.31622776601683794}
mean=0.9400 {'C': 1.0}
mean=0.9400 {'C': 3.1622776601683795}
mean=0.9400 {'C': 10.0}
mean=0.9400 {'C': 31.622776601683793}
mean=0.9200 {'C': 100.0}
mean=0.9200 {'C': 316.22776601683796}
mean=0.9200 {'C': 1000.0}
```

In [32]:

```
# make a plot
allC = []
allscores = []
for m,p in zip(svmcv.cv_results_['mean_test_score'], svmcv.cv_results_['params']
):
    allC.append(p['C'])
    allscores.append(m)

plt.figure()
plt.semilogx(allC, allscores, 'kx-')
plt.xlabel('C'); plt.ylabel('accuracy')
plt.grid(True)
```



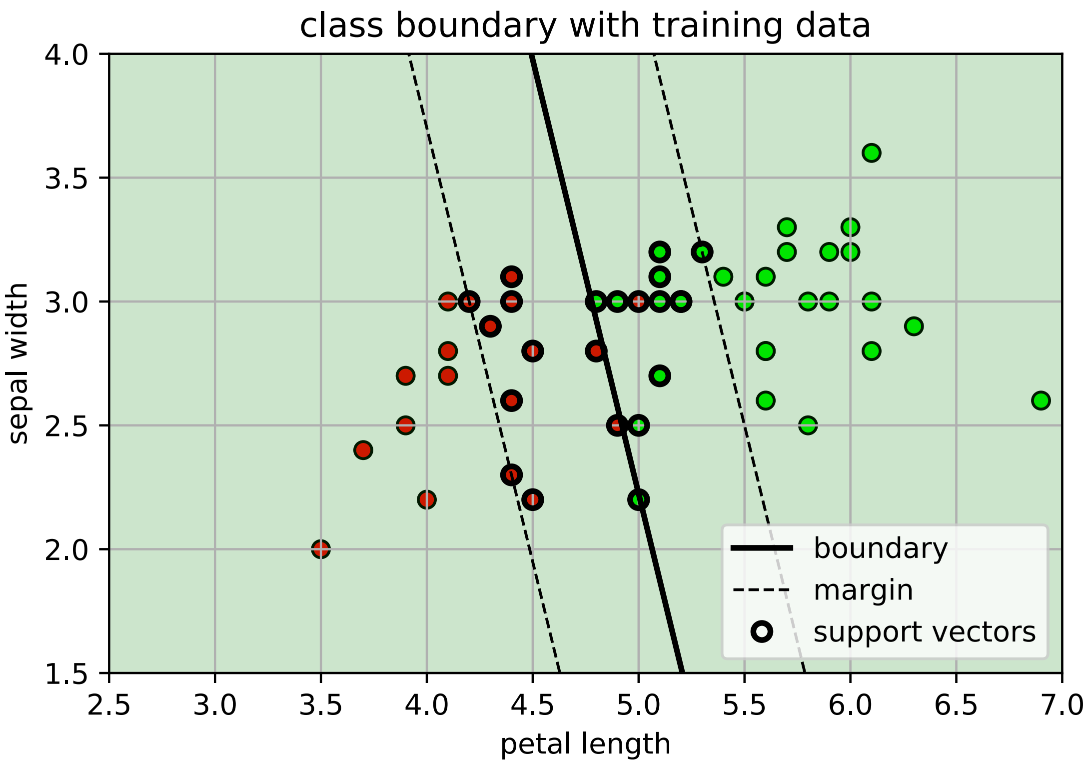
In [33]:

```
# view best results and best retrained estimator
print(svmcv.best_params_)
print(svmcv.best_score_)
print(svmcv.best_estimator_)
```

```
{'C': 0.31622776601683794}
0.96
SVC(C=0.31622776601683794, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [34]:

```
plt.figure()
plot_svm(svmcv.best_estimator_, axbox, mycmap)
plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap, edgecolors='k')
plt.xlabel('petal length'); plt.ylabel('sepal width')
plt.title('class boundary with training data');
```



In [35]:

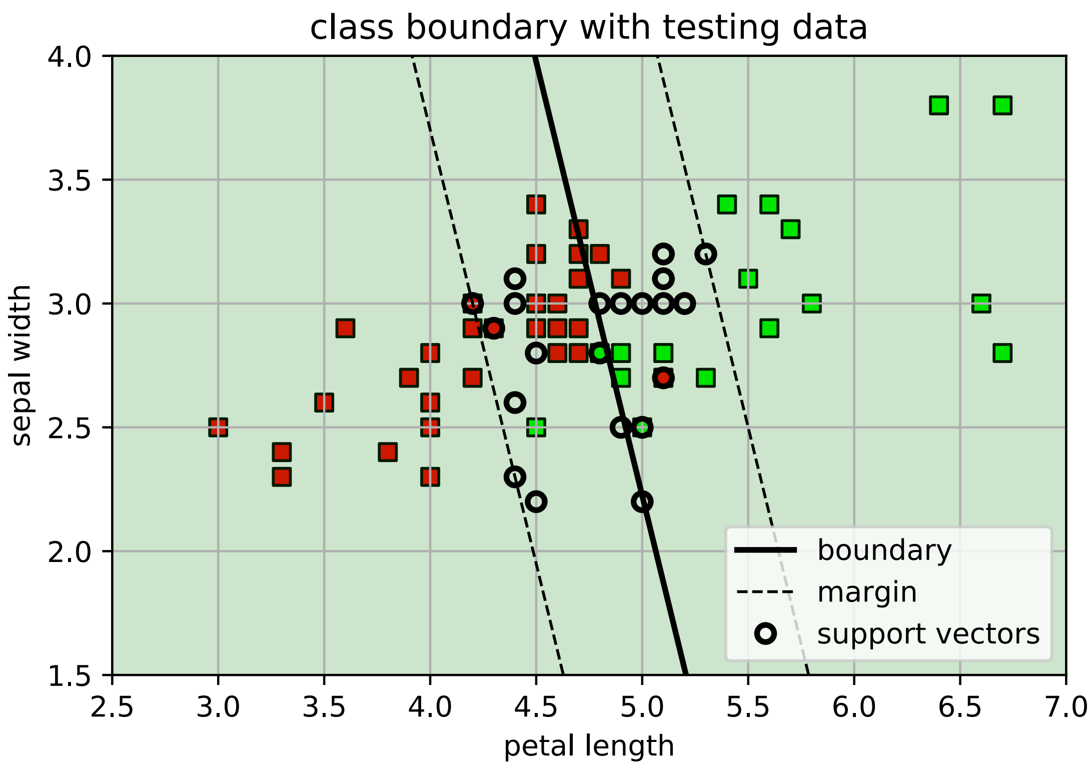
```
# Directly use svmcv to make predictions
predY = svmcv.predict(testX)

acc = metrics.accuracy_score(testY, predY)
print("test accuracy = " + str(acc))
```

```
test accuracy = 0.88
```

In [36]:

```
# Plot test data
plt.figure()
plot_svm(svmcv.best_estimator_, axbox, mycmap)
plt.scatter(testX[:,0], testX[:,1], c=testY, cmap=mycmap, marker='s', edgecolors
='k')
plt.xlabel('petal length'); plt.ylabel('sepal width')
plt.title('class boundary with testing data');
```



Multi-class SVM

- In sklearn, `svm.SVC` implements "1-vs-1" multi-class classification.
  - Train binary classifiers on all pairs of classes.
    - 3-class Example: 1 vs 2, 1 vs 3, 2 vs 3
  - To label a sample, pick the class with the most votes among the binary classifiers.
- Problem:
  - 1v1 classification is very slow when there are a large number of classes.
    - if there are C classes, need to train $C(C-1)/2$ binary classifiers!

1-vs-all SVM

- Use the `multiclass.OneVsRestClassifier` to build a 1-vs-all classifier from any binary classifier.
  - For `GridSearchCV`, use `'estimator__C'` as the parameter label for `C` in the SVM.

In [38]:
```
msvm = multiclass.OneVsRestClassifier(svm.SVC(kernel='linear'))

# setup the parameters and run CV
paramgrid = {'estimator__C': logspace(-3,3,13)}
msvmcv = model_selection.GridSearchCV(msvm, paramgrid, cv=5, n_jobs=-1, verbose=
True)
msvmcv.fit(trainX, trainY)
print(msvmcv.best_params_)
```

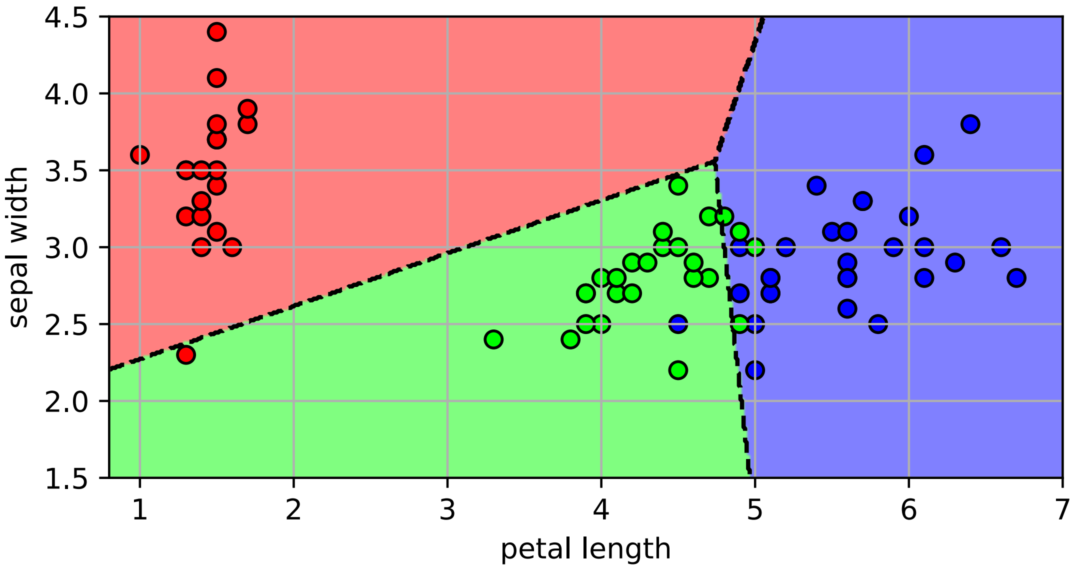
```
Fitting 5 folds for each of 13 candidates, totalling 65 fits
{'estimator__C': 31.622776601683793}

[Parallel(n_jobs=-1)]: Done  65 out of  65 | elapsed:    0.5s finished
```

3-class decision boundaries

In [40]:
```
svm3fig
```

Out[40]:



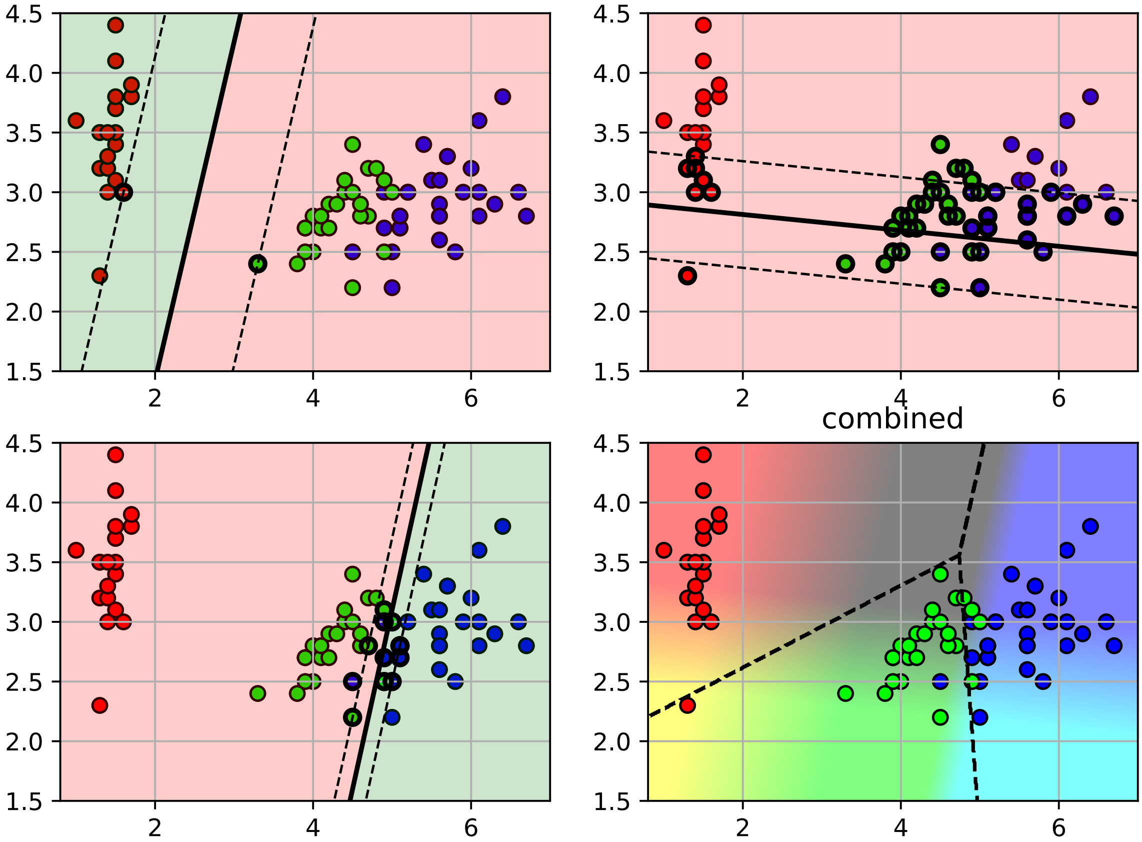
Decision boundaries for each binary classifier

In [41]:
```
for bclf in msvmcv.best_estimator_.estimators_:
    print(bclf.coef_)
```

```
[[-1.04615354  0.36923066]]
[[-0.1491237  -2.23855735]]
[[ 4.99792746 -1.66592287]]
```

In [43]:
```
bfig
```

Out[43]:



SVM Summary

- **Classifier:**
  - linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
  - given new sample $\mathbf{x}_*$, predict $y_* = \text{sign}(\mathbf{w}^T\mathbf{x}_* + b)$.
- **Training:**
  - Maximize the margin of the training data.
    - i.e., maximize the separation between the points and the decision boundary.
  - Allow some training samples to violate the margin.
    - Use cross-validation to pick the hyperparameter C.

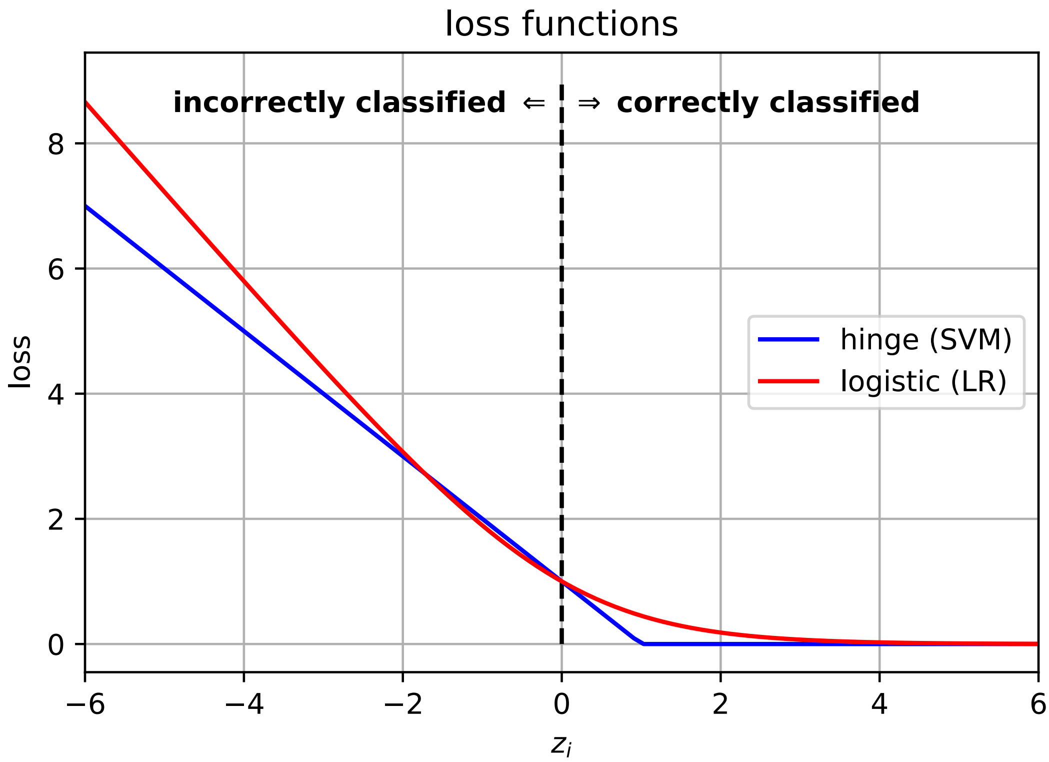
Summary

- **Linear classifiers:**
  - separate the data using a linear surface (hyperplane).
  - $y = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$
- **Two formulations:**
  - logistic regression - maximize the probability of the data
  - support vector machine - maximize the margin of the hyperplane

- **Loss functions**
  - SVM - ensure a margin of 1 between boundary and closest point
  - LR - push the classification boundary as far as possible from all points

In [45]:
```
lossfig
```

Out[45]:



- **Advantages:**
  - SVM works well on high-dimensional features (d large), and has low generalization error.
  - LR has well-calibrated probabilities.
- **Disadvantages:**
  - decision surface can only be linear!
    - Next lecture we will see how to deal with non-linear decision surfaces.