CS4487 - Machine Learning

Lecture 4b - Non-linear Classifiers

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

Ensemble Classifiers

- *Why trust only one expert?*
  - In real life, we may consult several experts, or go with the "wisdom of the crowd"
  - In machine learning, *why trust only one classifier?*

- Ensemble methods aim to combine multiple classifiers together to form a better classifier.
- Examples:
  - **boosting** - training multiple classifiers, each focusing on errors made by previous classifiers.
  - **bagging** - training multiple classifiers from random selection of training data

---

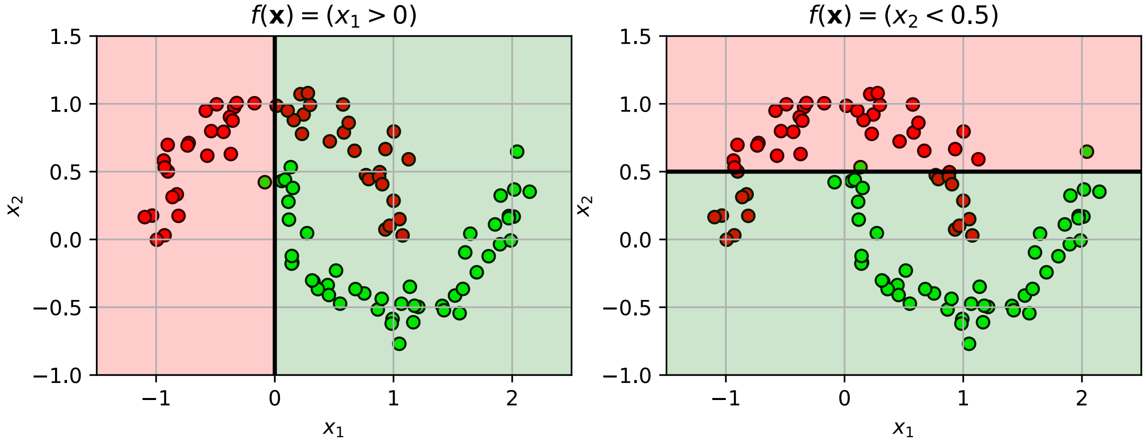
AdaBoost - Adaptive Boosting

- Base classifier is a "weak learner"
  - A simple classifier that can be slightly better than random chance (>50%)
  - Example: *decision stump classifier*
    - check if feature value is above (or below) a threshold.
    - $y = f(x) = \begin{cases} +1, & x_j \geq T \\ -1, & x_j < T \end{cases}$

In [4]:

```
wlfig
```

Out[4]:



- **Idea:** train weak classifiers sequentially
- In each iteration,
  - Pick a weak learner $f_t(\mathbf{x})$ that best carves out the input space.
  - The weak learner should focus on data that is misclassified.
    - Apply weights to each sample in the training data.
    - Higher weights give more priority to difficult samples.

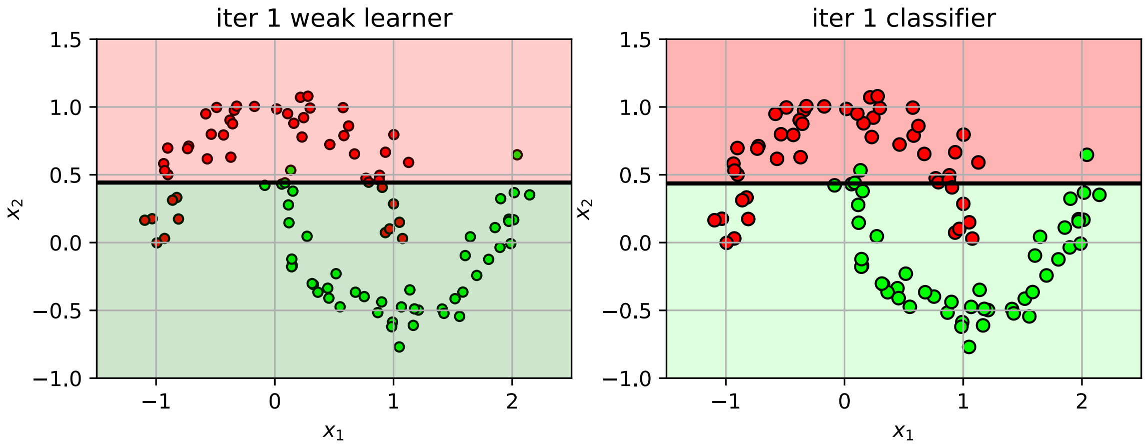
Iteration 1

- Initially, weights for all training samples are equal: $w_i = 1/N$
  - Pick best weak learner.

In [7]:

```
plts[1]
```

Out[7]:

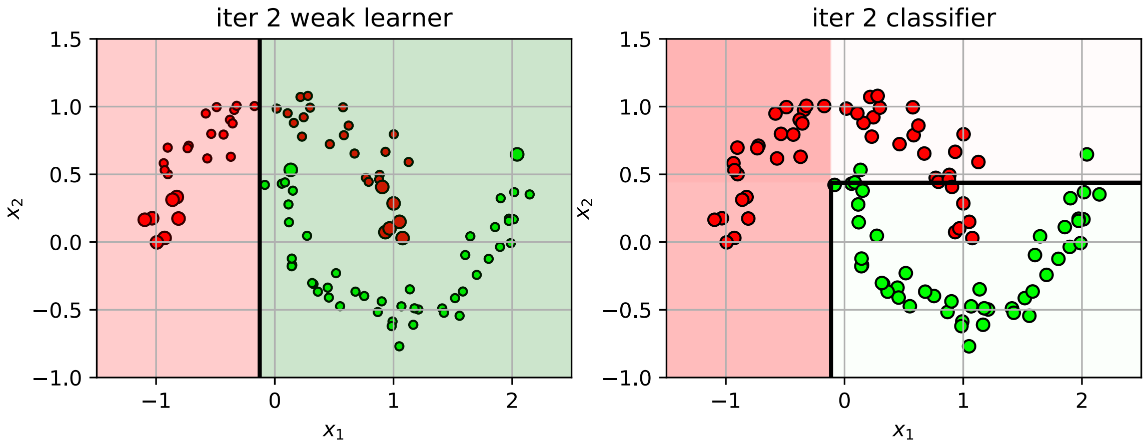


Iteration 2

- points are re-weighted based on the current classification result:
  - increase weights of samples that are misclassified: $w_i = w_i e^{\alpha}$
  - decrease weights of correctly classified samples: $w_i = w_i e^{-\alpha}$
  - $\alpha = 0.5 \log \frac{1-err}{err}$ is based on the current classifier error.
  - (larger circles indicates higher weight)
- using the weighted data, train another weak learner $f_2(\mathbf{x})$.
- the classifier function is the weighted sum of weak learners
  - $f(\mathbf{x}) = \sum_{t=1}^{D} \alpha_t f_t(\mathbf{x})$

In [8]:
```
plts[2]
```

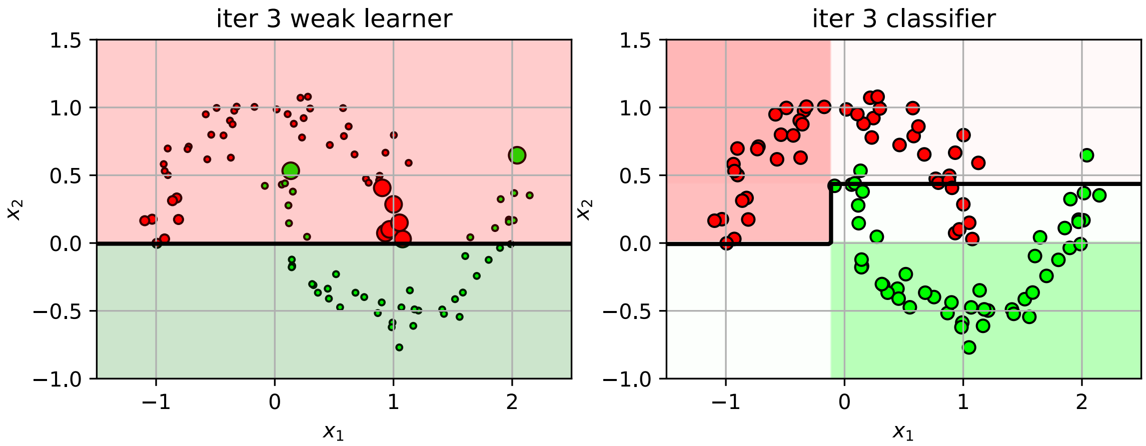
Out[8]:



Keep iterating...

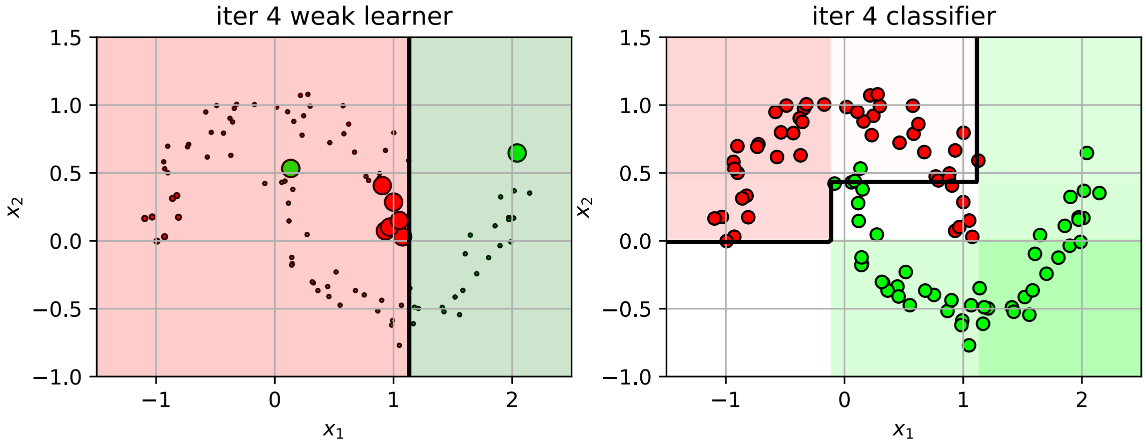
In [9]:
```
plts[3]
```

Out[9]:



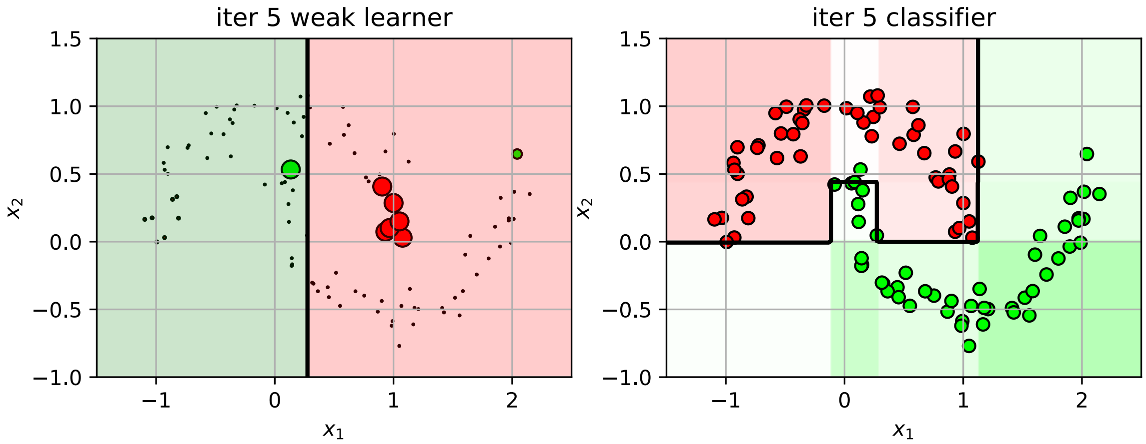
In [10]:
```
plts[4]
```

Out[10]:



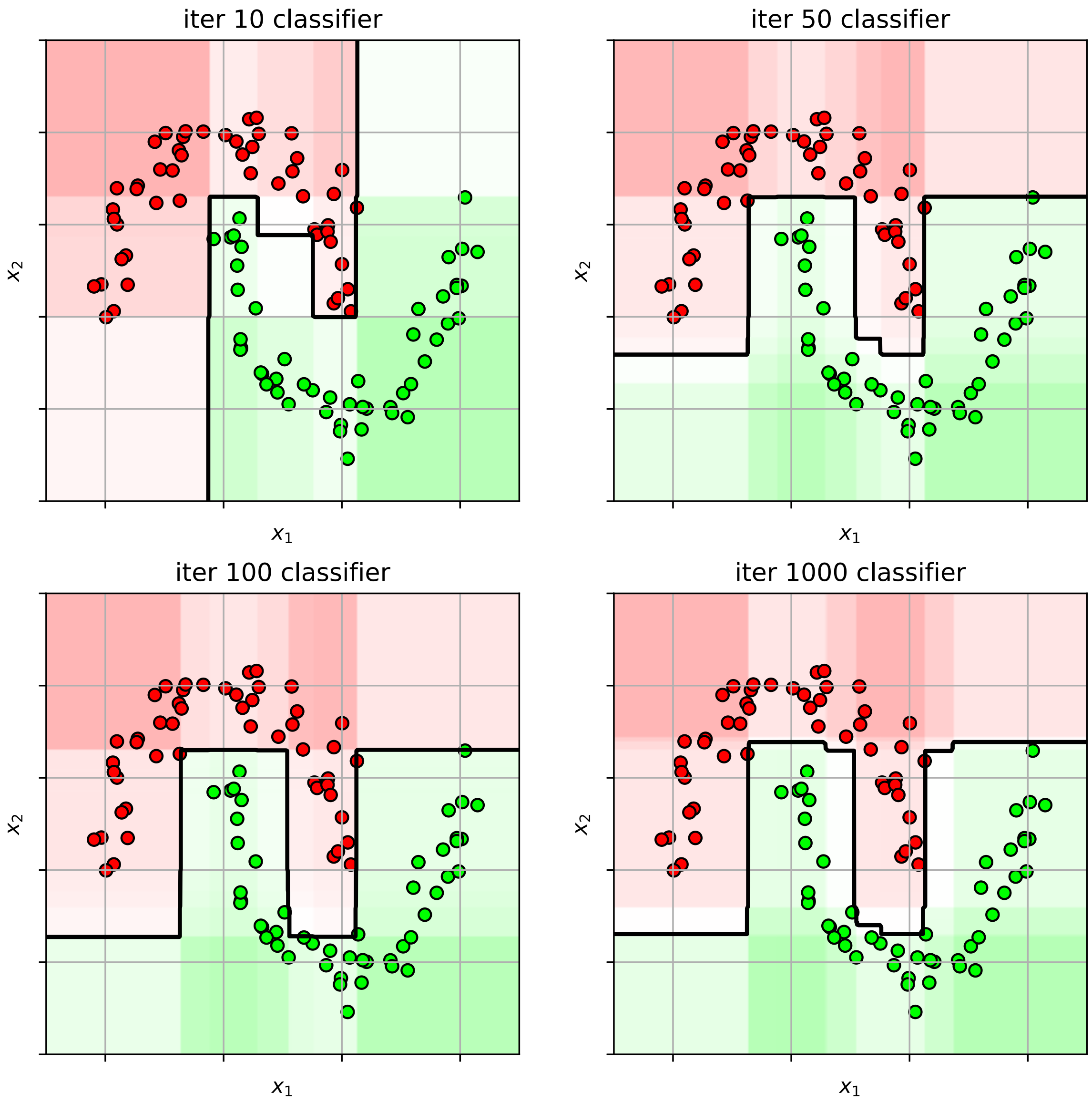
In [11]:
```
plts[5]
```

Out[11]:



- After many iterations...

In [13]:
```
adafig
```
Out[13]:



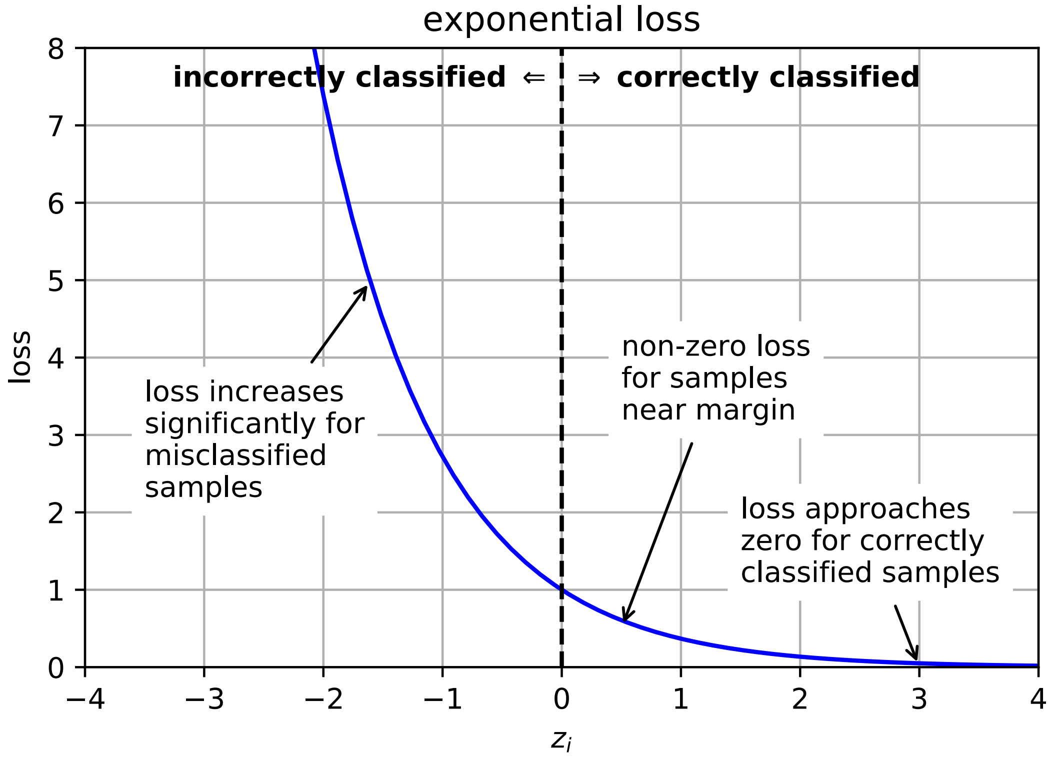
Adaboost loss function

- exponential loss
  - $L(z_i) = e^{-z_i}$
    - $z_i = y_i f(\mathbf{x}_i)$
  - very sensitive to misclassified outliers.

In [15]:

```
lossfig
```

Out[15]:

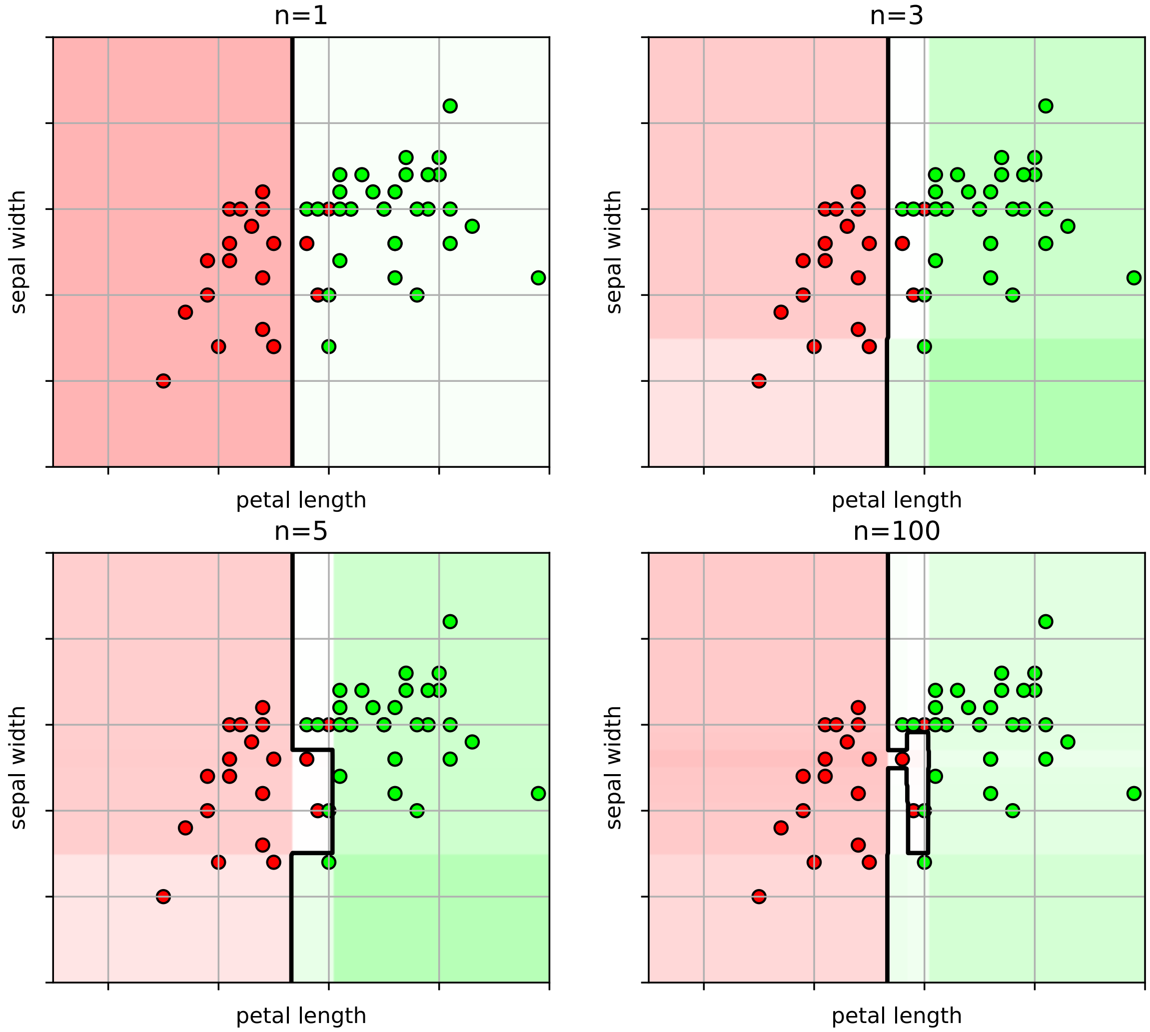


Example on Iris data

- Too many weak-learners and AdaBoost carves out space for the outliers.

In [19]:
```
irisfig
```

Out[19]:

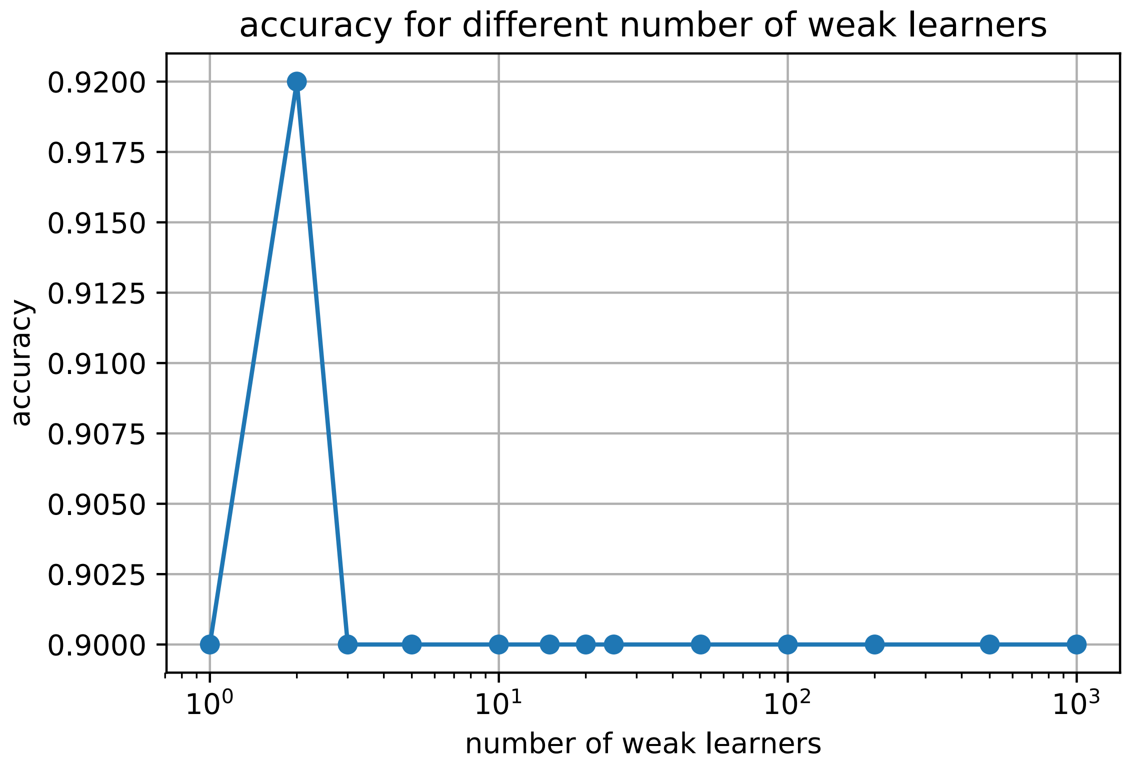


- use cross-validation to select number of weak learners.

In [20]:
```
# setup the list of parameters to try
paramgrid = {'n_estimators': array([1, 2, 3, 5, 10, 15, 20, 25, 50, 100, 200, 50
0, 1000]) }
print(paramgrid)

# setup the cross-validation object
# (NOTE: using parallelization in GridSearchCV, not in AdaBoost)
adacv = model_selection.GridSearchCV(ensemble.AdaBoostClassifier(random_state=44
87),
                                     paramgrid, cv=5, n_jobs=-1)

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

print("best params:", adacv.best_params_)
```

```
{'n_estimators': array([    1,     2,     3,     5,    10,    15,    20,    25,    50
,   100,   200,
         500,  1000])}
best params: {'n_estimators': 2}
```

In [22]:
```
(avgscores, pnames, bestind) = extract_grid_scores(adacv, paramgrid)
paramfig = plt.figure()
plt.semilogx(paramgrid['n_estimators'], avgscores, 'o-')
plt.grid(True)
plt.ylabel('accuracy'); plt.xlabel('number of weak learners')
plt.title('accuracy for different number of weak learners')
plt.show()
```



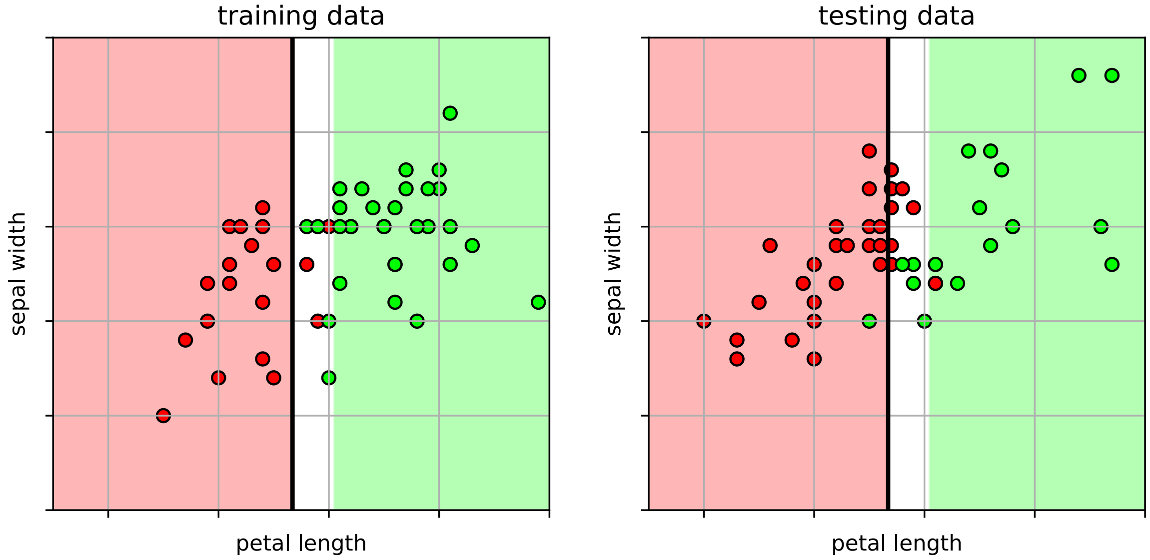
In [23]:
```
# predict from the model
predY = adacv.predict(testX)

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

```
test accuracy = 0.82
```

In [25]:

```
ifig2
```

Out[25]:



- Boosting can do feature selection
  - each decision stump classifier looks at one feature
- One of the original face detection methods (Viola-Jones) used Boosting.
  - extract a lot of image features from the face
  - during training, Boosting learns which ones are the most useful.

AdaBoost Summary

- **Ensemble Classifier:**
  - Combine the outputs of many "weak" classifiers to make a "strong" classifier
- **Training:**
  - In each iteration,
    - training data is re-weighted based on whether it is correctly classified or not.
    - weak classifier focuses on misclassified data from previous iterations.
  - Use cross-validation to pick number of weak learners.

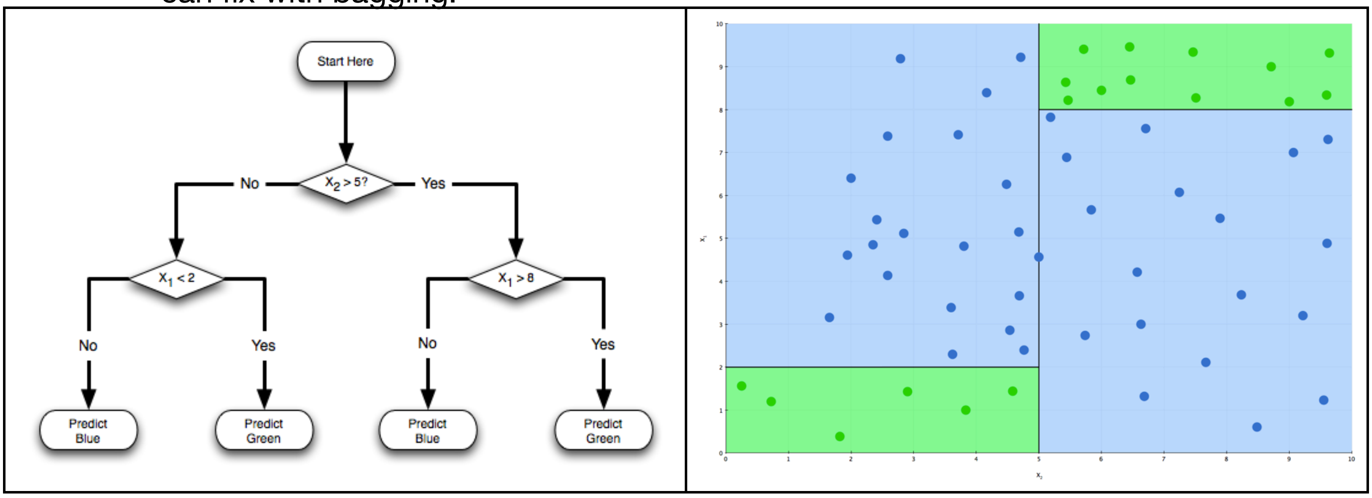
- **Advantages:**
  - Good generalization performance
  - Built-in features selection - decision stump selects one feature at a time.
- **Disadvantages:**
  - Sensitive to outliers.

Outline

1. Nonlinear classifiers
2. Kernel trick and kernel SVM
3. **Ensemble Methods - Boosting, Random Forests**
4. Classification Summary

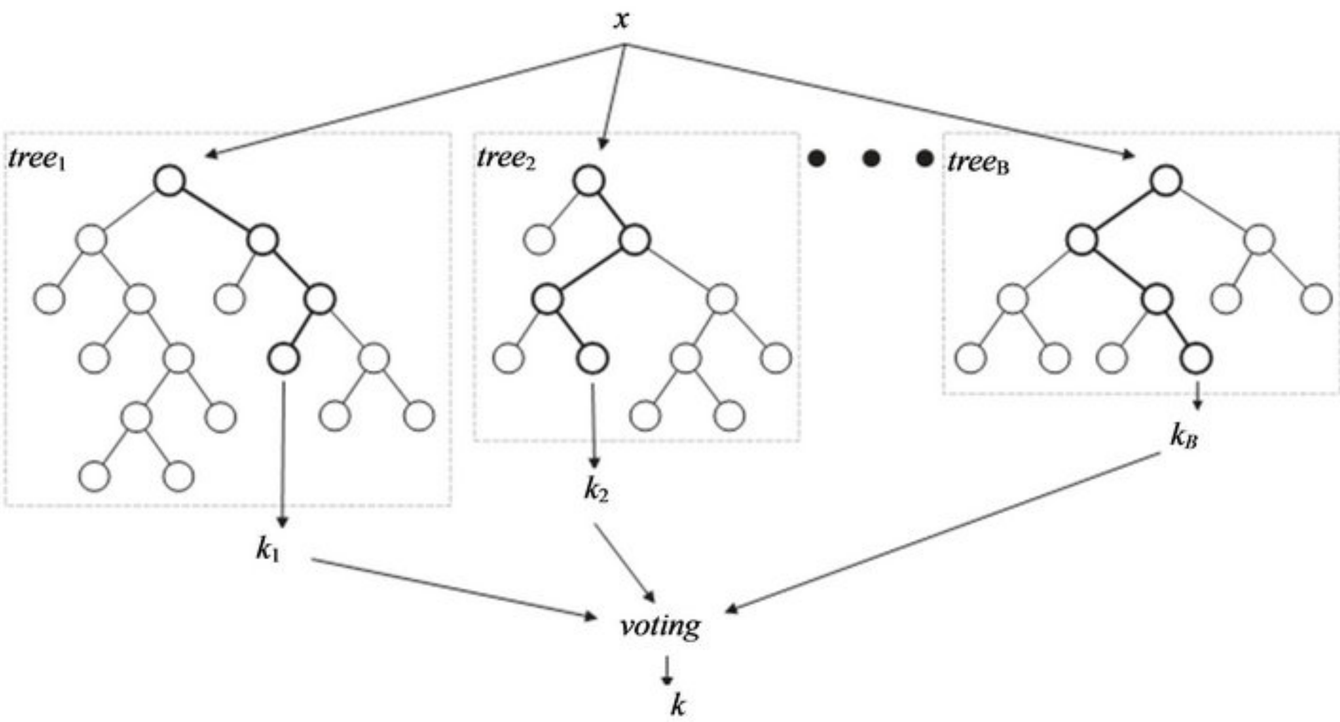
Decision Tree

- Simple "Rule-based" classifier
  - At each node, move down the tree based on that node's criteria.
  - leaf node contains the prediction
- **Advantage:** can create complex conjunction of rules
- **Disadvantage:** easy to overfit by itself
  - can fix with bagging!



Random Forest Classifier

- Use **bagging** to make an ensemble of Decision Tree Classifiers
  - for each *Decision Tree Classifier*
    - create a new training set by randomly sampling from the training set
    - for each split in a tree, select a random subset of features to use
- for a test sample, the prediction is aggregated over all trees.



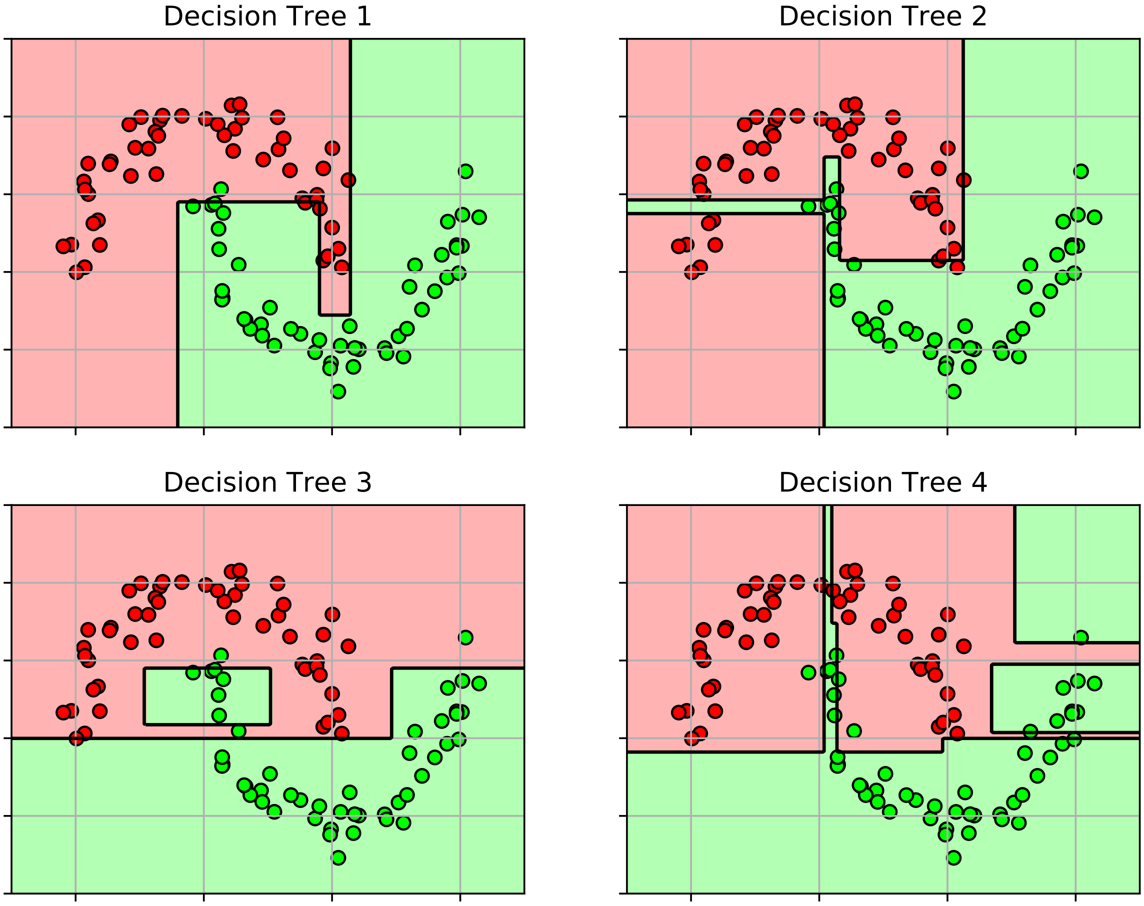
In [26]:
```
# learn a RF classifier
# use 4 trees
clf = ensemble.RandomForestClassifier(n_estimators=4, random_state=4487, n_jobs=
-1)
clf.fit(X3, Y3)
```

Out[26]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=4, n_jobs=-1,
            oob_score=False, random_state=4487, verbose=0,
            warm_start=False)
```

- Here are the 4 decision trees
  - each uses a different random sampling of original training set

In [29]:
```
dtfig
```

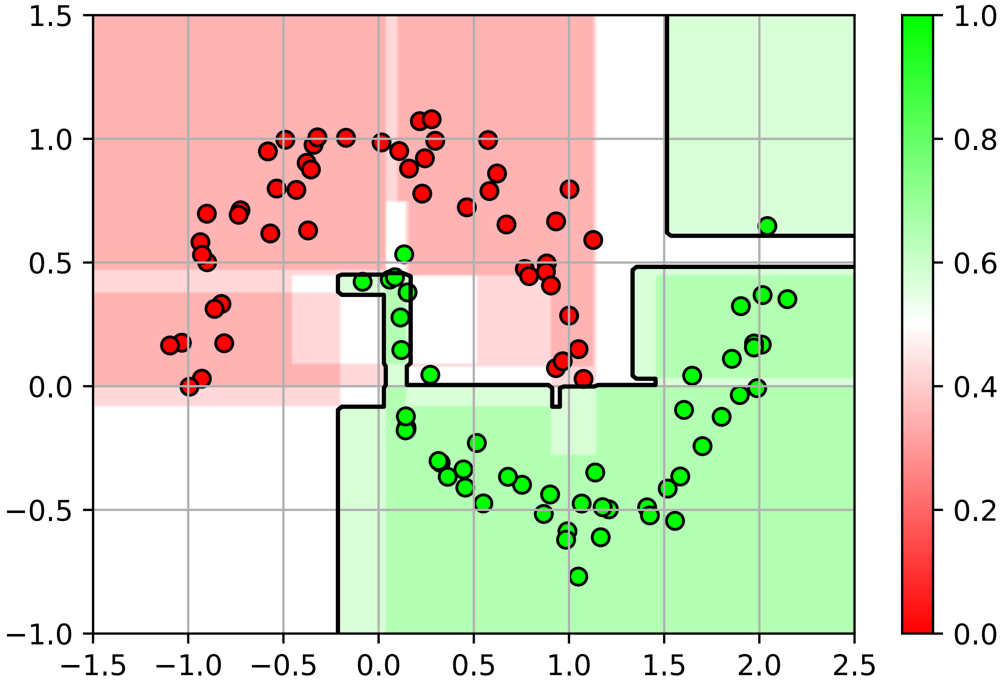
Out[29]:



- and the aggregated classifier

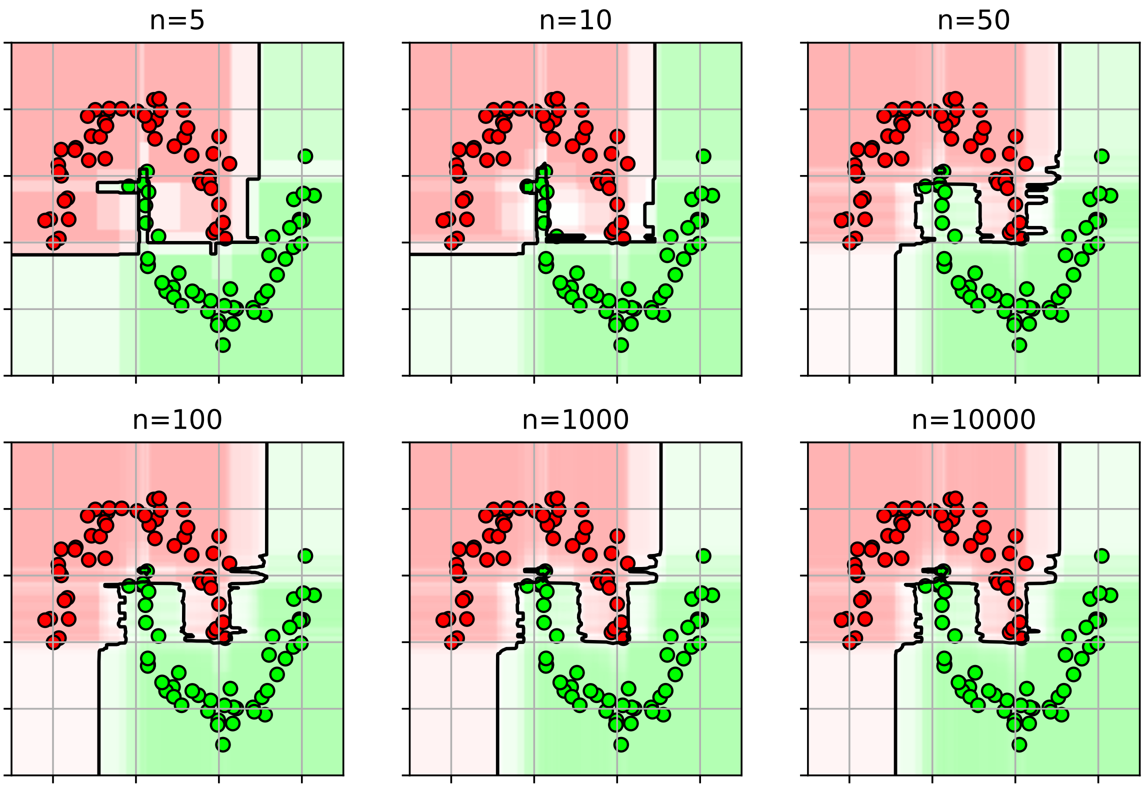
In [30]:
```
rffig
```

Out[30]:



- Using more trees

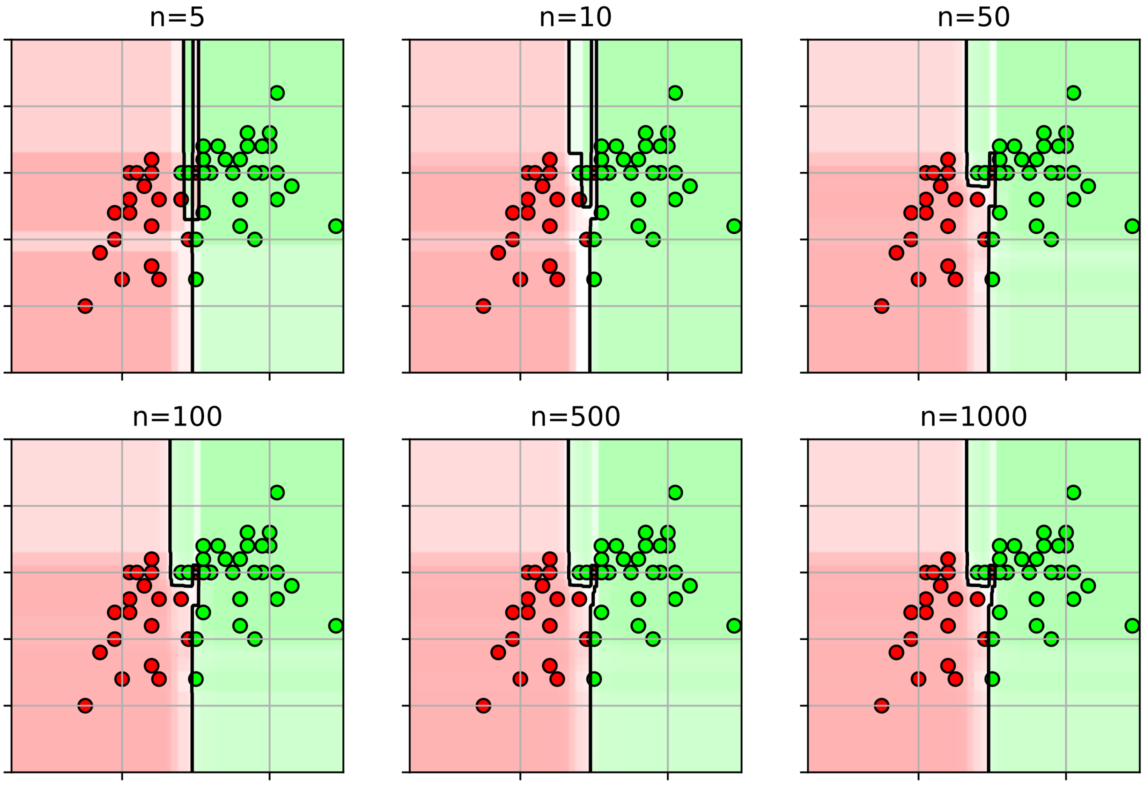
In [31]:
```
# learn RF classifiers for different n_estimators
plt.figure(figsize=(9,6))
clfs = {}
for i,n in enumerate([5, 10, 50, 100, 1000, 10000]):
    clfs[n] = ensemble.RandomForestClassifier(n_estimators=n, random_state=4487,
n_jobs=-1)
    clfs[n].fit(X3, Y3)

    plt.subplot(2,3,i+1)
    plot_rf(clfs[n], axbox, X3)
    plt.scatter(X3[:,0], X3[:,1], c=Y3, cmap=mycmap, edgecolors='k')
    plt.gca().xaxis.set_ticklabels([])
    plt.gca().yaxis.set_ticklabels([])
    plt.title("n=" + str(n))
```



- Try on the iris data

In [32]:

```
# learn RF classifiers for different n_estimators
plt.figure(figsize=(9,6))
clfs = {}
axbox = [2.5, 7, 1.5, 4]

for i,n in enumerate([5, 10, 50, 100, 500, 1000]):
    clfs[n] = ensemble.RandomForestClassifier(n_estimators=n, random_state=4487,
n_jobs=-1)
    clfs[n].fit(trainX, trainY)

    plt.subplot(2,3,i+1)
    plot_rf(clfs[n], axbox, trainX)
    plt.scatter(trainX[:,0], trainX[:,1], c=trainY, cmap=mycmap, edgecolors='k')
    plt.gca().xaxis.set_ticklabels([])
    plt.gca().yaxis.set_ticklabels([])
    plt.title("n=" + str(n))
```



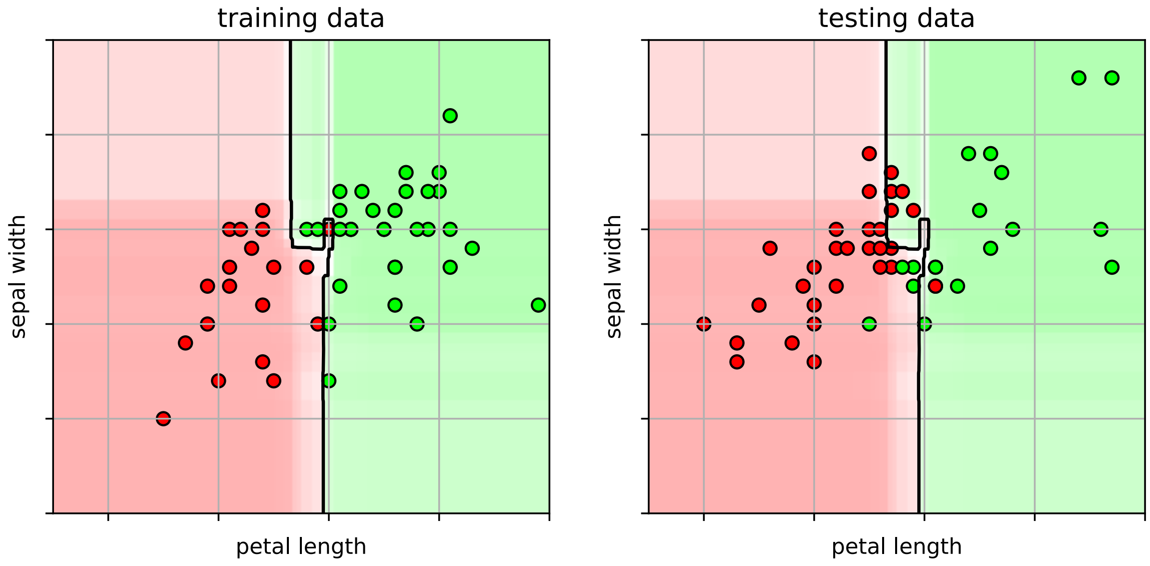
In [33]:

```
# predict from the model
predY = clfs[1000].predict(testX)

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

```
test accuracy = 0.8
```

In [35]:
```python
# classifier boundary w/ training and test data
ifig3
```

Out[35]:



- Important parameters for cross-validation
  - `max_features` - maximum number of features used for each split
  - `max_depth` - maximum depth of a decision tree

Outline

1. Nonlinear classifiers
2. Kernel trick and kernel SVM
3. Ensemble Methods - Boosting, Random Forests
4. **Classification Summary**

Feature Pre-processing

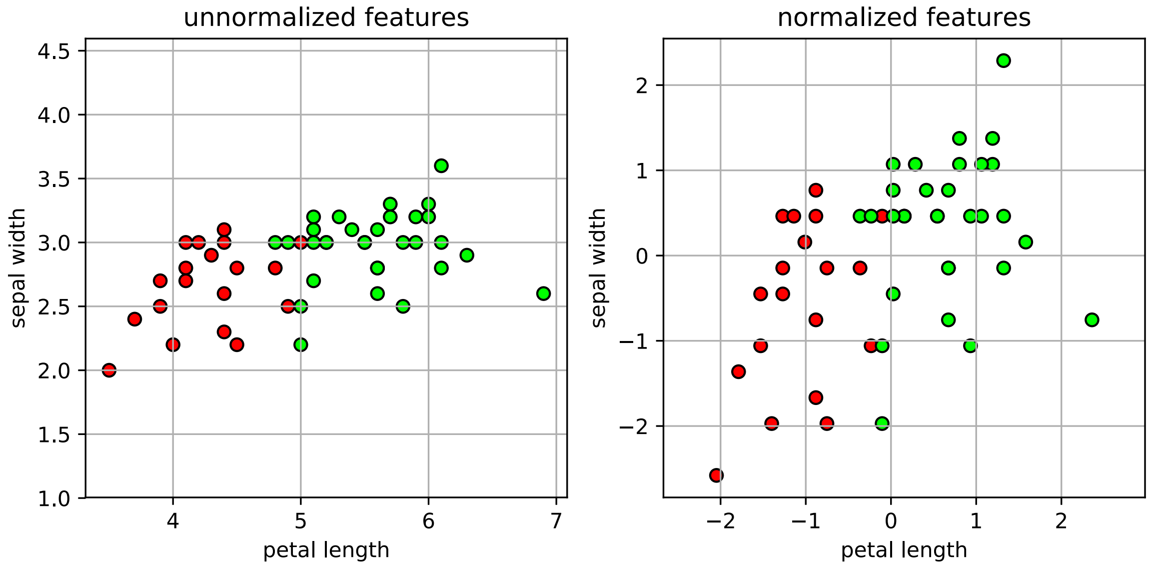
- Some classifiers, such as SVM and LR, are sensitive to the scale of the feature values.
  - feature dimensions with larger values may dominate the objective function.
- Common practice is to *standardize* or *normalize* each feature dimension before learning the classifier.
  - Two Methods...

- **Method 1:** scale each feature dimension so the mean is 0 and variance is 1.
  - $\tilde{x_d} = \frac{1}{s}(x_d - m)$
  - s is the standard deviation of feature values.
  - m is the mean of the feature values.
- **NOTE:** the parameters for scaling the features should be estimated from the training set!
  - same scaling is applied to the test set.

In [36]:
```
# using the iris data
scaler = preprocessing.StandardScaler()  # make scaling object
trainXn = scaler.fit_transform(trainX)   # use training data to fit scaling para
meters
testXn  = scaler.transform(testX)        # apply scaling to test data
```

In [38]:
```
nfig1
```

Out[38]:

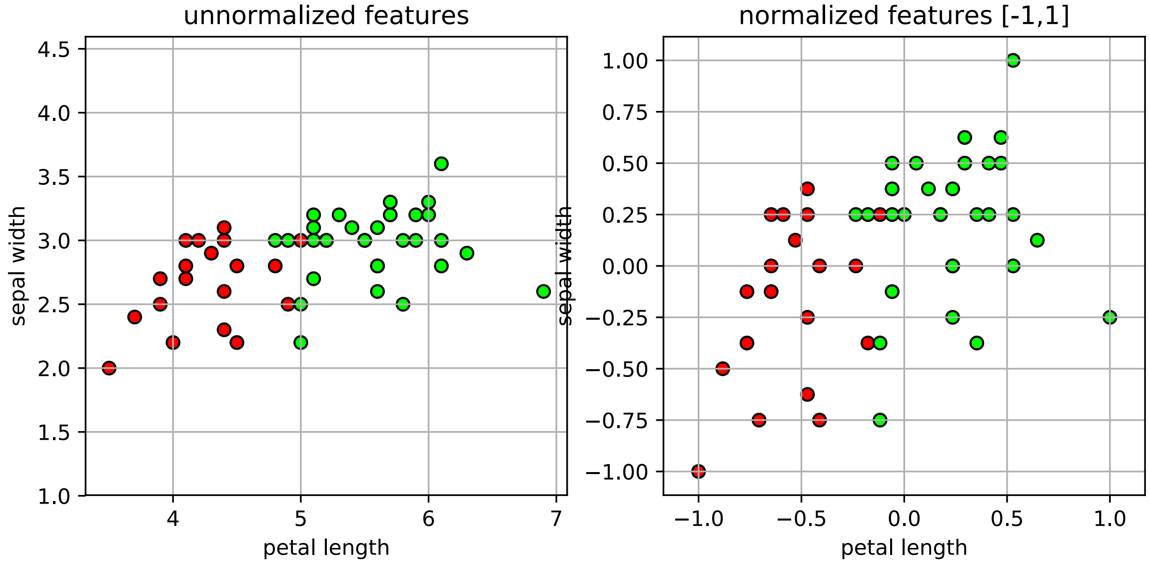


- **Method 2:** scale features to a fixed range, -1 to 1.
  - $\tilde{x_d} = 2 * (x_d - min)/(max - min) - 1$
  - max and min are the maximum and minimum features values.

In [39]:
```
# using the iris data
scaler = preprocessing.MinMaxScaler(feature_range=(-1,1))     # make scaling obje
ct
trainXn = scaler.fit_transform(trainX)   # use training data to fit scaling para
meters
testXn  = scaler.transform(testX)        # apply scaling to test data
```

In [41]:
```
nfig2
```

Out[41]:



Data Representation and Feature Engineering

- How to represent data as a vector of numbers?
  - the encoding of the data into a feature vector should make sense
  - inner-products or distances calculated between feature vectors should be meaningful in terms of the data.

- Categorical variables
  - Example: x has 3 possible category labels: cat, dog, horse
  - We could encode this as: $x = 0$, $x = 1$, and $x = 2$.
    - Suppose we have two data points: $x = cat$, $x' = horse$.
    - What is the meaning of $x * x' = 2$?

One-hot encoding

- encode a categorical variable as a vector of ones and zeros
  - if there are K categories, then the vector is K dimensions.
- Example:
  - x=cat → x=[1 0 0]
  - x=dog → x=[0 1 0]
  - x=horse → x=[0 0 1]

In [42]:
```
# one-hot encoding example
X = [[0], [1], [0], [2], [2]]  # original categorical data {0,1,2}
ohe = preprocessing.OneHotEncoder(sparse=False)
ohe.fit(X)          # finds the number of categories in the training set: 0-max(X
)
ohe.transform(X)   # transform to one-hot-encoding
```

Out[42]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

Binning

- encode a real value as a vector of ones and zeros
  - assign each feature value to a bin, and then use one-hot-encoding

In [43]:
```
# example
X = [[-3], [0.5], [1.5]]  # the data
bins = [-2,-1,0,1,2]      # define the bins

# map from value to bin number
Xbins = digitize(X, bins=bins)

# map from bin number to 0-1 vector
ohe = preprocessing.OneHotEncoder(n_values=len(bins), sparse=False)
ohe.fit(Xbins)
ohe.transform(Xbins)
```

Out[43]:
```
array([[1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Data transformations - polynomials

- Represent interactions between features using polynomials
- Example:
  - 2nd-degree polynomial models pair-wise interactions
    - $[x_1, x_2] \rightarrow [x_1^2, x_1 x_2, x_2^2]$
  - Combine with other degrees:
    - $[x_1, x_2] \rightarrow [1, x_1, x_2, x_1^2, x_1 x_2, x_2^2]$

In [44]:
```
X = [[0,1], [1,2], [3,4]]
pf = preprocessing.PolynomialFeatures(degree=2)
pf.fit(X)
pf.transform(X)
```

Out[44]:
```
array([[ 1.,   0.,   1.,   0.,   0.,   1.],
       [ 1.,   1.,   2.,   1.,   2.,   4.],
       [ 1.,   3.,   4.,   9.,  12.,  16.]])
```

Data transformations - univariate

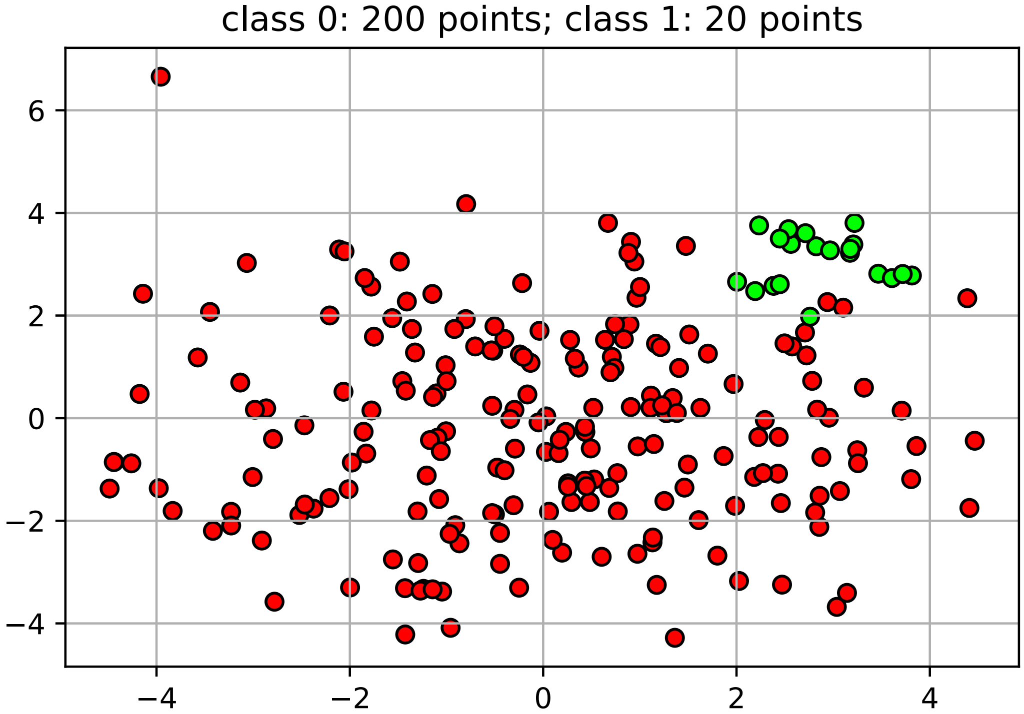
- Apply a non-linear transformation to the feature
  - e.g., x → log(x)
  - useful if the dynamic range of x is very large

Unbalanced Data

- For some classification tasks that data will be unbalanced
  - many more examples in one class than the other.
- **Example:** detecting credit card fraud
  - credit card fraud is rare
    - 50 examples of fraud, 5000 examples of legitimate transactions.

In [46]:
```
udatafig
```

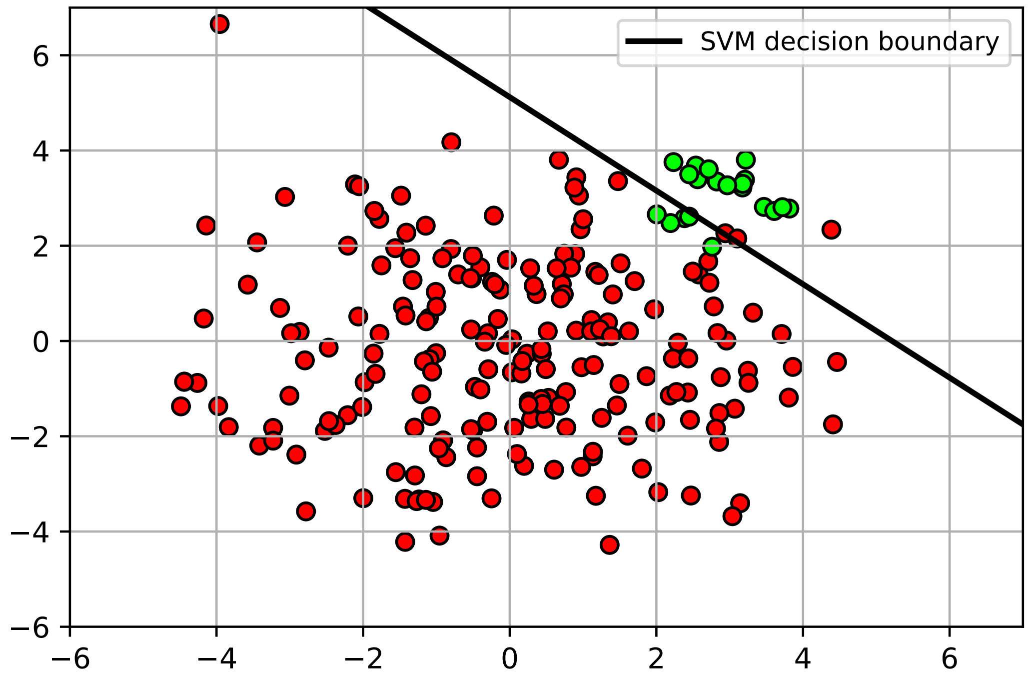
Out[46]:



- Unbalanced data can cause problems when training the classifier
  - classifier will focus more on the class with more points.
  - decision boundary is pushed away from class with more points

In [48]:
```
udatafig1
```

Out[48]:



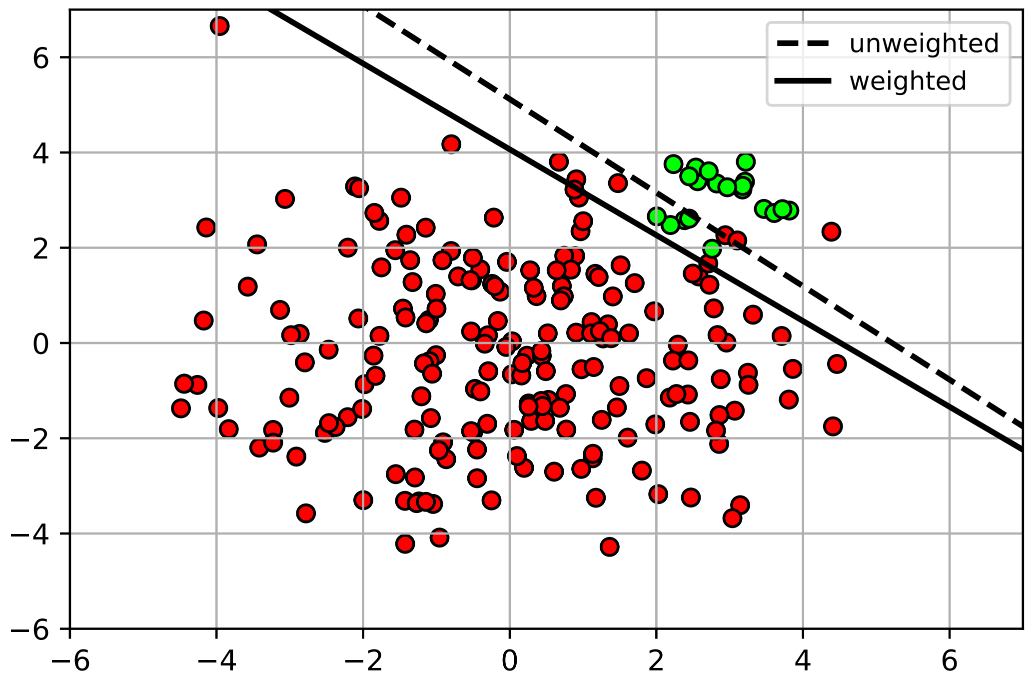
- **Solution:** apply weights on the classes during training.
  - weights are inversely proportional to the class size.

In [49]:
```
clfw = svm.SVC(kernel='linear', C=10,  class_weight='balanced')
clfw.fit(X, Y)

print("class weights =", clfw.class_weight_)
```

```
class weights = [0.55 5.5 ]
```

```
In [51]: udatafig2
```

Out[51]:

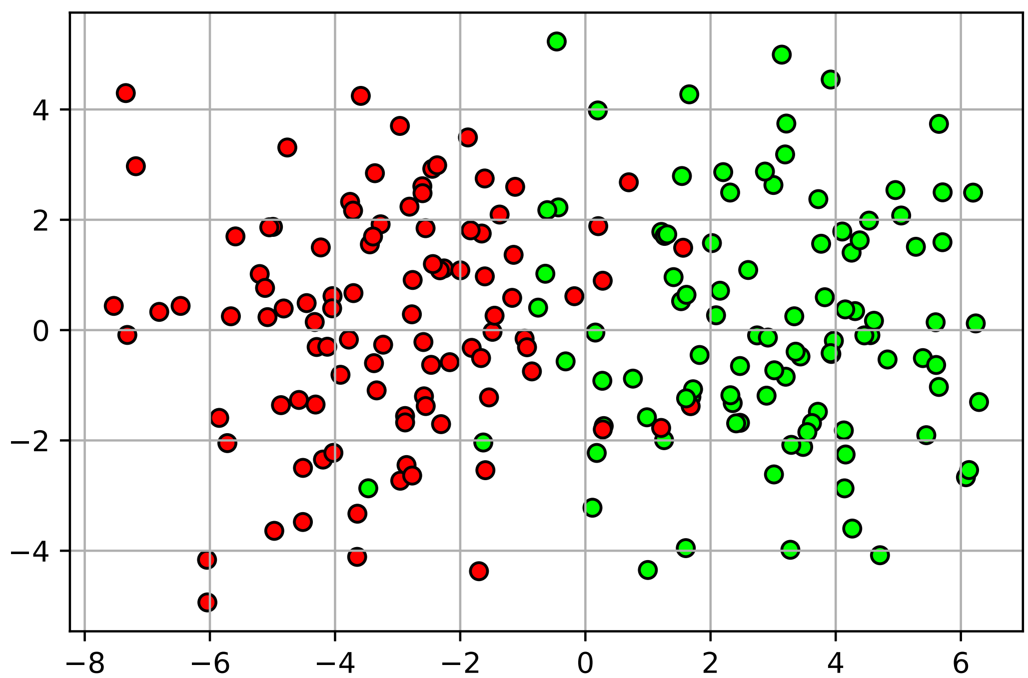


Classifier Imbalance

- In some tasks, errors on certain classes cannot be tolerated.
- **Example:** detecting spam vs non-spam
  - non-spam should *definitely not* be marked as spam
    - okay to mark some spam as non-spam

```
In [53]: udatafig3
```

Out[53]:



- Class weighting can be used to make the classifier focus on certain classes
  - e.g., weight non-spam class higher than spam class
    - classifier will try to correctly classify all non-spam samples, at the expense of making errors on spam samples.

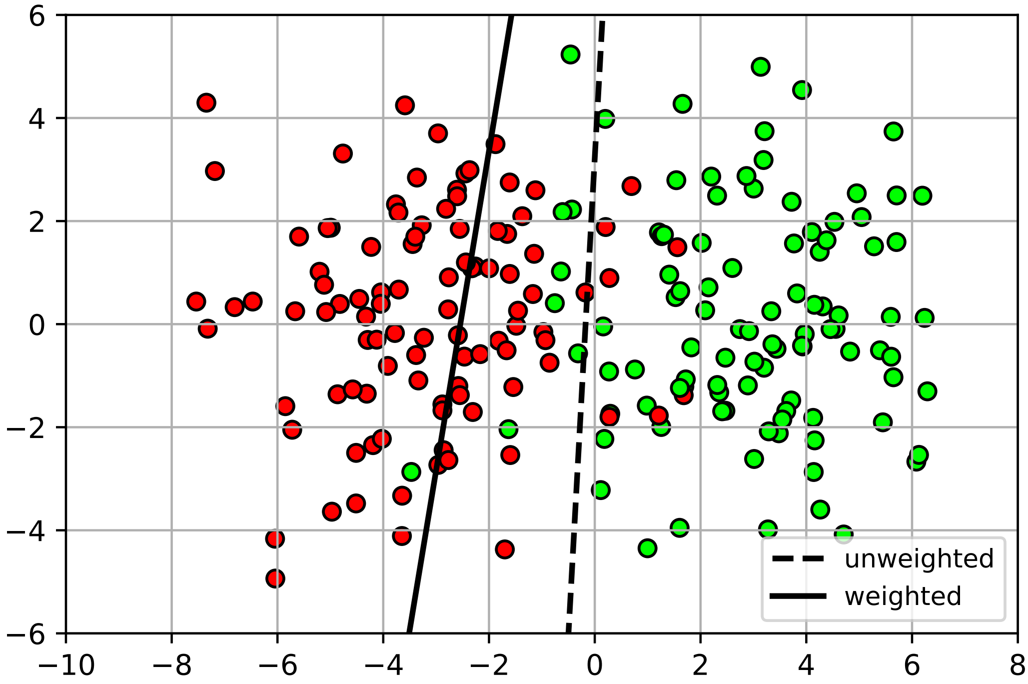
In [54]:

```
# dictionary (key,value) = (class name, class weight)
cw = {0: 0.2,
      1:  5}  # class 1 is 25 times more important!

clfw = svm.SVC(kernel='linear', C=10,  class_weight=cw)
clfw.fit(X, Y);
```

In [56]:

```
udatafig4
```

Out[56]:



Classification Summary

- **Classification task**
  - Observation $\mathbf{x}$: typically a real vector of feature values, $\mathbf{x} \in \mathbb{R}^d$.
  - Class y: from a set of possible classes, e.g., $\mathcal{Y} = \{0, 1\}$
  - **Goal:** given an observation $\mathbf{x}$, predict its class y.

Name	Type	Classes	Decision function	Training	Advantages	Disadvantages
Bayes' classifier	generative	multi-class	non-linear	estimate class-conditional densities $p(x\|y)$ by maximizing likelihood of data.	- works well with small amounts of data. - multi-class. - minimum probability of error if probability models are correct.	- depends on the data correctly fitting the class-conditional.
logistic regression	discriminative	binary	linear	maximize likelihood of data in $p(y\|x)$.	- well-calibrated probabilities. - efficient to learn.	- linear decision boundary. - sensitive to C parameter.
support vector machine (SVM)	discriminative	binary	linear	maximize the margin (distance between decision surface and closest point).	- works well in high-dimension. - good generalization.	- linear decision boundary. - sensitive to C parameter.
kernel SVM	discriminative	binary	non-linear (kernel function)	maximize the margin.	- non-linear decision boundary. - can be applied to non-vector data using appropriate kernel.	- sensitive to kernel function and hyperparameters. - high memory usage for large datasets
AdaBoost	discriminative	binary	non-linear (ensemble of weak learners)	train successive weak learners to focus on misclassified points.	- non-linear decision boundary. can do feature selection. - good generalization.	- sensitive to outliers.
Random Forest	discriminative	multi-class	non-linear (ensemble of decision trees)	aggregate predictions over several decision trees, trained using different subsets of data.	- non-linear decision boundary. can do feature selection. - good generalization. - fast	- sensitive to outliers.

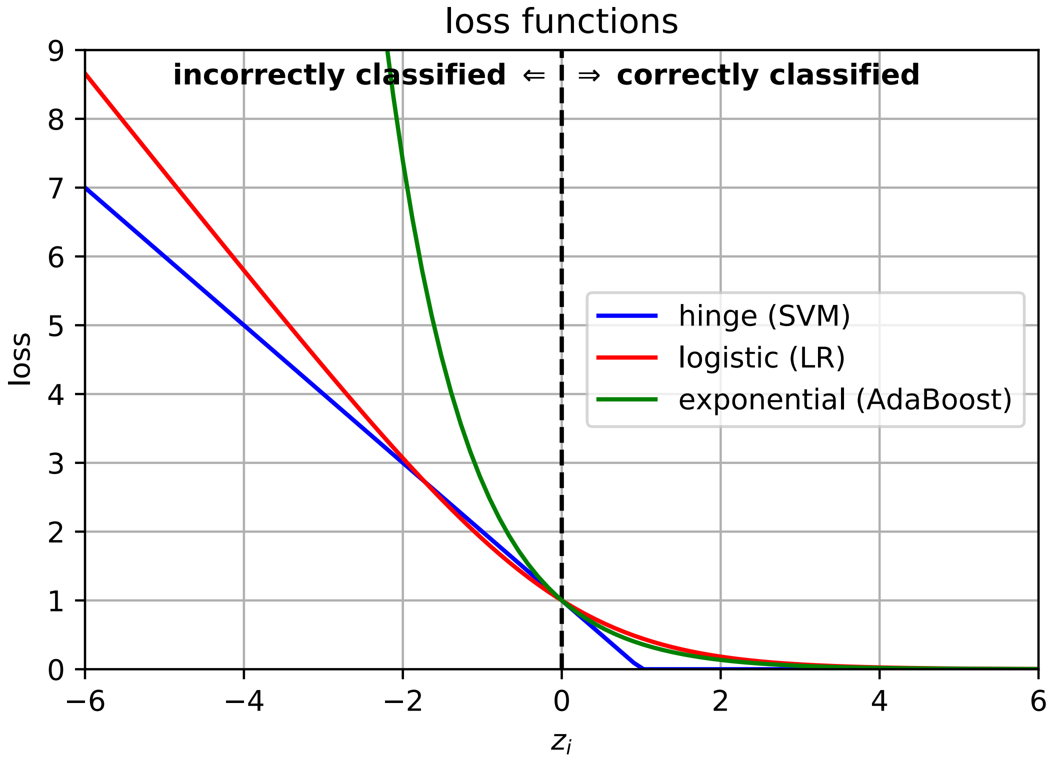
Loss functions

- The classifiers differ in their loss functions, which influence how they work.
  - $z_i = y_i f(\mathbf{x}_i)$

In [58]:

```
lossfig
```

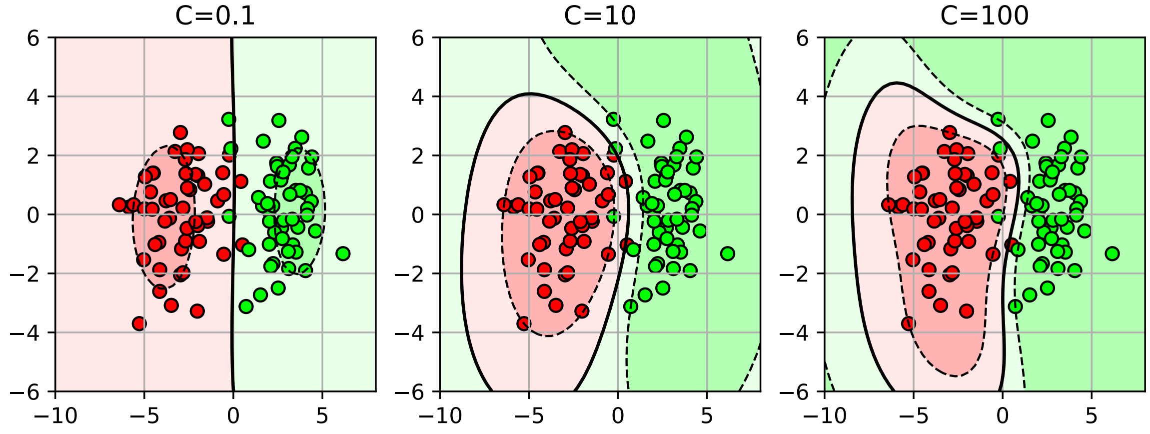
Out[58]:



Regularization and Overfitting

- Some models have terms to prevent overfitting the training data.
  - this can improve *generalization* to new data.
- There is a parameter to control the regularization effect.
  - select this parameter using cross-validation on the training set.

In [60]:
```
ofig
```
Out[60]:



Other things

- *Multiclass classification*
  - can use binary classifiers to do multi-class using *1-vs-rest* formulation.
- *Feature normalization*
  - normalize each feature dimension so that some feature dimensions with larger ranges do not dominate the optimization process.
- *Unbalanced data*
  - if more data in one class, then apply weights to each class to balance objectives.
- *Class imbalance*
  - mistakes on some classes are more critical.
  - reweight class to focus classifier on correctly predicting one class at the expense of others.

Applications

- Web document classification, spam classification
- Face gender recognition, face detection, digit classification

Features

- Choice of features is important!
  - using uninformative features may confuse the classifier.
  - use domain knowledge to pick the best features to extract from the data.

Which classifier is best?

- **"No Free Lunch" Theorem** (Wolpert and Macready)

> "If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems."

- In other words, there is no *best* classifier for all tasks. The best classifier depends on the particular problem.