MECS 6616

Supervised Learning II

Spring 2020

Matei Ciocarlie

Fundamental Supervised ML Problems

$\mathbf{x} \subset \Re^{d1}, \mathbf{y} \subset \Re^{d2}$

$\mathbf{x}_1 \rightarrow \mathbf{y}_1$

$\mathbf{x}_2 \rightarrow \mathbf{y}_2$

$\ldots$

$\mathbf{x}_n \rightarrow \mathbf{y}_n$

Fundamental Supervised ML Problems

$\mathbf{x} \subset \Re^{d1}, \mathbf{y} \subset \Re^{d2}$

$\mathbf{x}_1 \rightarrow \mathbf{y}_1$

$\mathbf{x}_2 \rightarrow \mathbf{y}_2$

...

$\mathbf{x}_n \rightarrow \mathbf{y}_n$

$\mathbf{x}_{new} \rightarrow ?$

Fundamental Supervised ML Problems

$\mathbf{x} \subset \Re^{d1}, \mathbf{y} \subset \Re^{d2}$

$\mathbf{x}_1 \rightarrow \mathbf{y}_1$

$\mathbf{x}_2 \rightarrow \mathbf{y}_2$

...

$\mathbf{x}_n \rightarrow \mathbf{y}_n$

$\mathbf{x}_{new} \rightarrow ?$

Classification:

$d2 = 1$

$y \subset \{c_1, c_2, \ldots, c_k\}$

y is **discrete**

y_i tells you what *class* $\mathbf{x}_i$ belongs to

Fundamental Supervised ML Problems

$\mathbf{x} \subset \Re^{d1}, \mathbf{y} \subset \Re^{d2}$

$\mathbf{x}_1 \rightarrow \mathbf{y}_1$

$\mathbf{x}_2 \rightarrow \mathbf{y}_2$

...

$\mathbf{x}_n \rightarrow \mathbf{y}_n$

$\mathbf{x}_{new} \rightarrow ?$

Classification:

$d2 = 1$

$y \subset \{c_1, c_2, \ldots, c_k\}$

y is **discrete**

y_i tells you what *class* $\mathbf{x}_i$ belongs to

Regression:

$\mathbf{y} \subset \Re^{d2}$

$\mathbf{y}$ is **continuous** (potentially multi-dimensional)

$\mathbf{x}_i$ is being **mapped** to point $\mathbf{y}_i$ in a different space (usually of lower dimensionality)

Nomenclature

$\mathbf{x} \subset \Re^{d1}, \mathbf{y} \subset \Re^{d2}$

$\mathbf{x}_1 \rightarrow \mathbf{y}_1$

$\mathbf{x}_2 \rightarrow \mathbf{y}_2$

...

$\mathbf{x}_n \rightarrow \mathbf{y}_n$

$\mathbf{x}_{new} \rightarrow ?$

$\mathbf{x}$: “features” / “independent variables”
$\mathbf{y}$: “labels” / “dependent variables”

“examples” / “training” / “fitting”

“testing” / “predicting”

Linear Regression

Assume that the relationship between $\mathbf{x}_i \subset \Re^{d1}$ and $y_i \subset \Re$ is **linear**:

$\mathbf{x}_i^T \mathbf{w} + a = y_i$ equivalent to $[1, \mathbf{x}_i^T]\, \mathbf{w} = y_i$

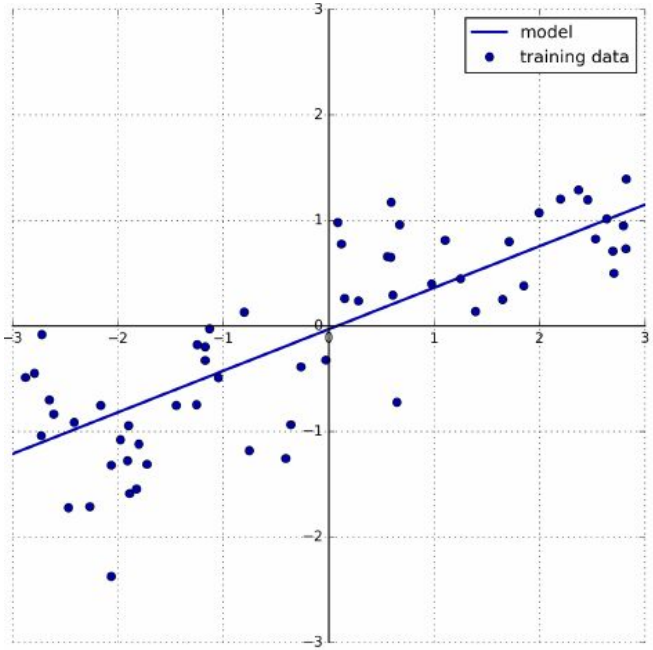
From now on, assume $\mathbf{x}_i^T \leftarrow [1, \mathbf{x}_i^T]$

Linear Regression

Assume that the relationship between $\mathbf{x}_i \subset \Re^{d1}$ and $y_i \subset \Re$ is **linear**:

$$\mathbf{x}_i^T \mathbf{w} + a = y_i \quad \text{equivalent to} \quad [1, \mathbf{x}_i^T]\, \mathbf{w} = y_i$$

From now on, assume $\mathbf{x}_i^T \leftarrow [1, \mathbf{x}_i^T]$



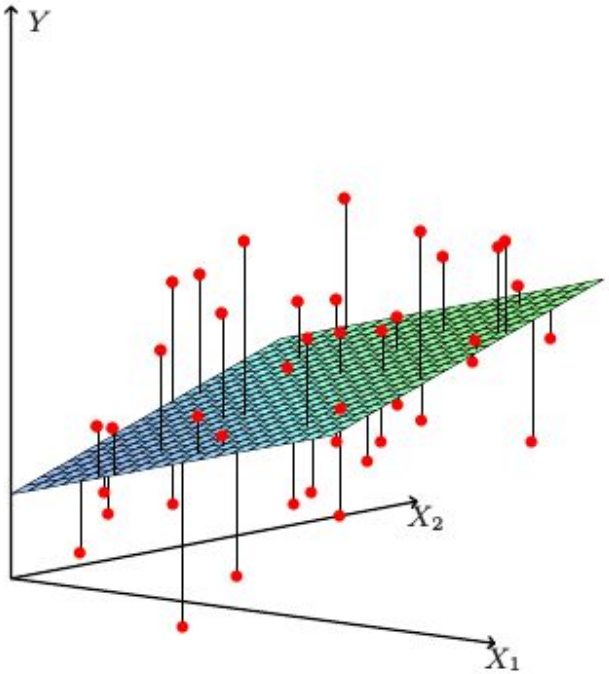
[Muller and Guido, Introduction to Machine Learning with Python]

Linear Regression

Assume that the relationship between $\mathbf{x}_i \subset \Re^{d1}$ and $y_i \subset \Re$ is **linear**:

$\mathbf{x}_i^T \mathbf{w} + a = y_i$ equivalent to $[1, \mathbf{x}_i^T]\, \mathbf{w} = y_i$

From now on, assume $\mathbf{x}_i^T \leftarrow [1, \mathbf{x}_i^T]$



[Hastie et al., The Elements of Statistical Learning]

Linear Regression

Assume that the relationship between $\mathbf{x} \subset \Re^{d1}$ and $y \subset \Re$ is **linear**:

$$\mathbf{x}_i^T \mathbf{w} = y_i$$

We are assuming a linear model, and $\mathbf{w}$ comprises the parameters of the model

Training: given $\mathbf{x}_i, y_i, i \subset \{1,n\}$, compute best possible $\mathbf{w}$

Linear Regression

Assume that the relationship between $\mathbf{x} \subset \Re^{d1}$ and $y \subset \Re$ is **linear**:

$\mathbf{x}_i^T \mathbf{w} = y_i$

Training: given $\mathbf{x}_i, y_i, i \subset \{1,n\}$, compute best possible $\mathbf{w}$

- Computing $\mathbf{w}$ amounts to solving a linear system:

$$\mathbf{X}\,\mathbf{w} = \mathbf{y}$$

$$\mathbf{X} = \texttt{block\_row}(\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T)$$

$$\mathbf{y} = [y_1, \ldots, y_n]^T$$

Linear Regression

- Computing best possible $\mathbf{w}$ amounts to solving a linear system:

$$\mathbf{X}\,\mathbf{w} = \mathbf{y},\ \mathbf{X} \subset \Re^{N \times d1}$$

- $\mathbf{X}^T\mathbf{X}$ full rank (what does this imply about N?)
  - pseudoinverse $\mathbf{X}^+$ is also a left inverse, can be computed as $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$
  - $\mathbf{w} = \mathbf{X}^+\mathbf{y}$ is the solution with minimum error norm - least squares fit

$$\text{minimize } \sum_i (y_i - \mathbf{w}\mathbf{x}_i)^2$$

Linear Regression

- Computing best possible $\mathbf{w}$ amounts to solving a linear system:

$$\mathbf{X}\,\mathbf{w} = \mathbf{y},\ \mathbf{X} \subset \Re^{N \times d1}$$

- $\mathbf{X}^T\mathbf{X}$ full rank (well conditioned): $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ least squares fit
- $\mathbf{X}^T\mathbf{X}$ rank-deficient (poorly conditioned): $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$ ridge regression

minimizes $\sum_i (y_i - \mathbf{w}\mathbf{x}_i)^2 + \lambda\sum_j w_j^2$

equivalent to: minimize $\sum_i (y_i - \mathbf{w}\mathbf{x}_i)^2$ with constraint $\sum_j w_j^2 \leq t$

Regularization

- Critical concept in Machine Learning
- An intrinsic belief that our model should:
  - be as “simple” as possible
  - not “go crazy”, especially outside the areas where we’ve seen training examples
- Common implementation: keep tabs on the size of the model parameters (l_2 regularization)

The Kernel Trick

- Perform a non-linear projection of my data into a higher-dimensional space

$$\mathbf{x}_i \rightarrow \Phi(\mathbf{x}_i)$$

- Problem: the new space is too high-dimensional!
- Key idea:
  - We don't actually need the high-dimensional points $\Phi(\mathbf{x}_i)$
  - We only need the distances between them $d(\ \Phi(\mathbf{x}_i)\ ,\ \Phi(\mathbf{x}_j)\ )$

Gram matrix $\mathbf{K} \subset \Re^{N \times N}$, $\mathbf{K}_{i,j} = d(\ \Phi(\mathbf{x}_i)\ ,\ \Phi(\mathbf{x}_j)\ ) = k(\mathbf{x}_i\ ,\ \mathbf{x}_j)$

Kernel Ridge Regression

- Perform a non-linear projection of my data into a higher-dimensional space

$$\mathbf{x}_i \subset \Re^{d1} \quad \rightarrow \quad \Phi(\mathbf{x}_i) \subset \Re^{d2}, \text{ d2 is huge}$$

$$d(\ \Phi(\mathbf{x}_i)\ ,\ \Phi(\mathbf{x}_j)\ ) = k(\mathbf{x}_i\ ,\ \mathbf{x}_j)$$

- RR: invert the d1-dimensional covariance matrix $\mathbf{X}^T\mathbf{X}$
- KRR: invert N-dimensional gram matrix $\mathbf{K} \approx \mathbf{X}_\Phi{}^T\mathbf{X}_\Phi$ computed via the kernel function.
- Pros: working in a much higher dimensional space, might find linear structure
- Cons: running time now cubic in N, as opposed to $d1$

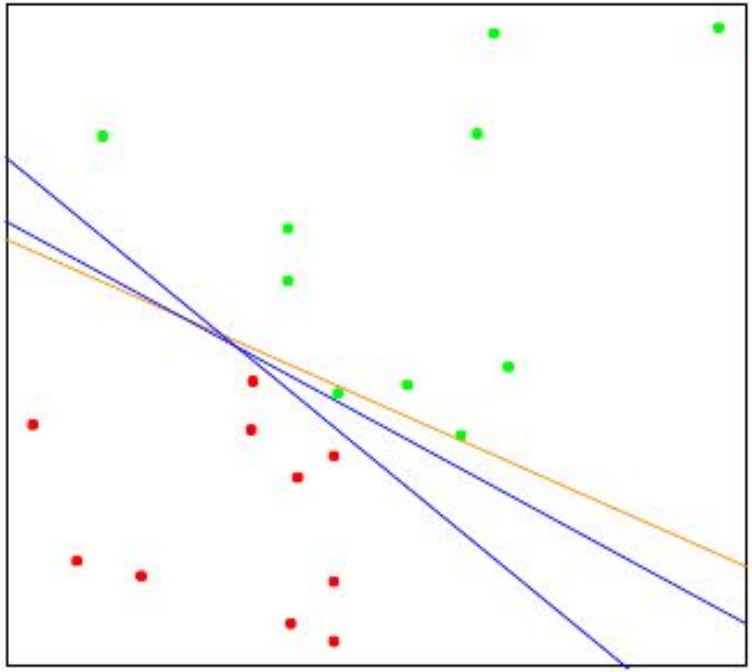
Classification via Linear Regression

- 2-class problems: $y_i \subset \{0,1\}$
- Perform (kernel) (ridge) linear regression exactly as before

$$\mathbf{X}\,\mathbf{w} = \mathbf{y},\ \mathbf{X} \subset \Re^{N \times d1}$$

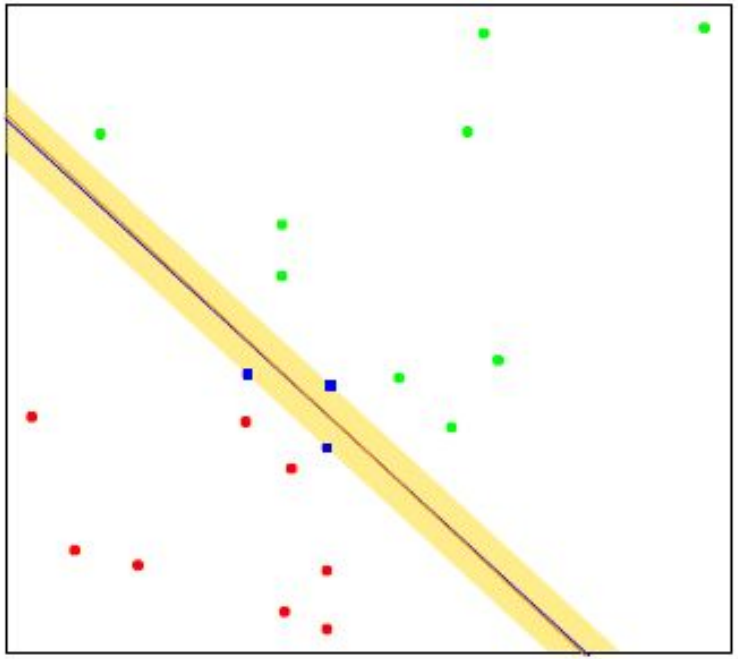
- Testing: $y_{new} = (\mathbf{x}_{new})^T\,\mathbf{w}$
  - if $y_{new} > 0.5$: $\mathbf{x}_{new}$ is in class 1
  - if $y_{new} < 0.5$: $\mathbf{x}_{new}$ is in class 0
- Learns a decision boundary as a hyperplane

Decision Boundaries



[Hastie et al., The Elements of Statistical Learning]

Decision Boundaries



[Hastie et al., The Elements of Statistical Learning]

Optimal Decision Boundaries

- For convenience, assume $y_i \subset \{-1,+1\}$
- Define hyperplane: $f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0$
  - $\mathbf{x}$ on plane: $f(\mathbf{x}) = 0$
  - $\mathbf{x}$ not on plane: $\texttt{sign}(f(\mathbf{x}))$ tells which side of the plane $\mathbf{x}$ is on
- Find a hyperplane that is a good decision boundary

find $\boldsymbol{\beta} + \beta_0$

subject to $y_i(\mathbf{x}_i^T\boldsymbol{\beta} + \beta_0) \geq 1$ for all i

Optimal Decision Boundaries

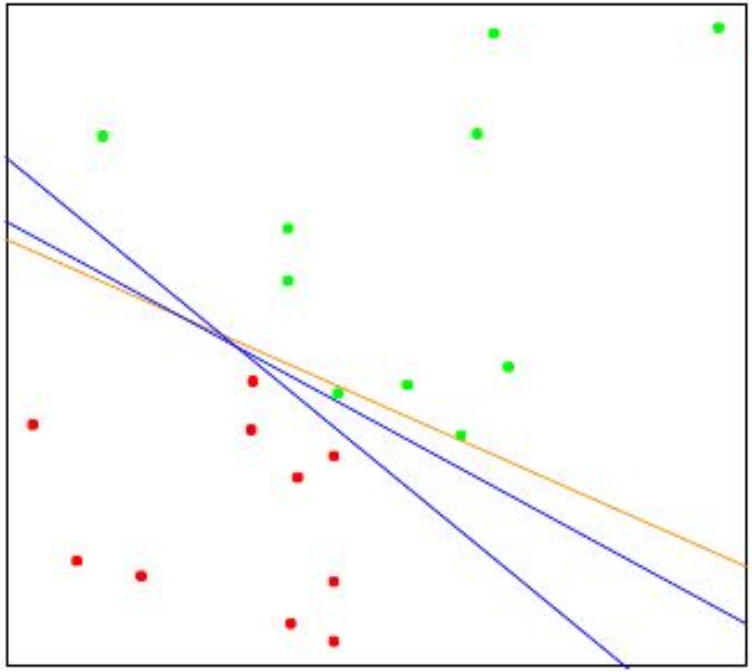
- For convenience, assume $y_i \subset \{-1,+1\}$
- Define hyperplane: $f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0$
  - $\mathbf{x}$ on plane: $f(\mathbf{x}) = 0$
  - $\mathbf{x}$ not on plane: $\texttt{sign}(f(\mathbf{x}))$ tells which side of the plane $\mathbf{x}$ is on
- Find optimal hyperplane as a decision boundary (maximum margin):

$$\text{minimize } \|\boldsymbol{\beta}\|$$

$$\text{subject to } y_i(\mathbf{x}^T\boldsymbol{\beta} + \beta_0) \geq 1 \text{ for all } i$$

- Assumes classes are linearly separable

Decision Boundaries



[Hastie et al., The Elements of Statistical Learning]

Optimal Decision Boundaries

- For convenience, assume $y_i \subset \{-1,+1\}$
- Define hyperplane: $f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0$
  - $\mathbf{x}$ on plane: $f(\mathbf{x}) = 0$
  - $\mathbf{x}$ not on plane: $\texttt{sign}(f(\mathbf{x}))$ tells which side of the plane $\mathbf{x}$ is on
- Find optimal hyperplane as a decision boundary (maximum margin):

$$\text{minimize } \|\boldsymbol{\beta}\| + C\sum_i \xi_i$$

$$\text{subject to } y_i(\mathbf{x}^T\boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i,\ \xi_i \geq 0 \text{ for all } i$$

Optimal Decision Boundaries

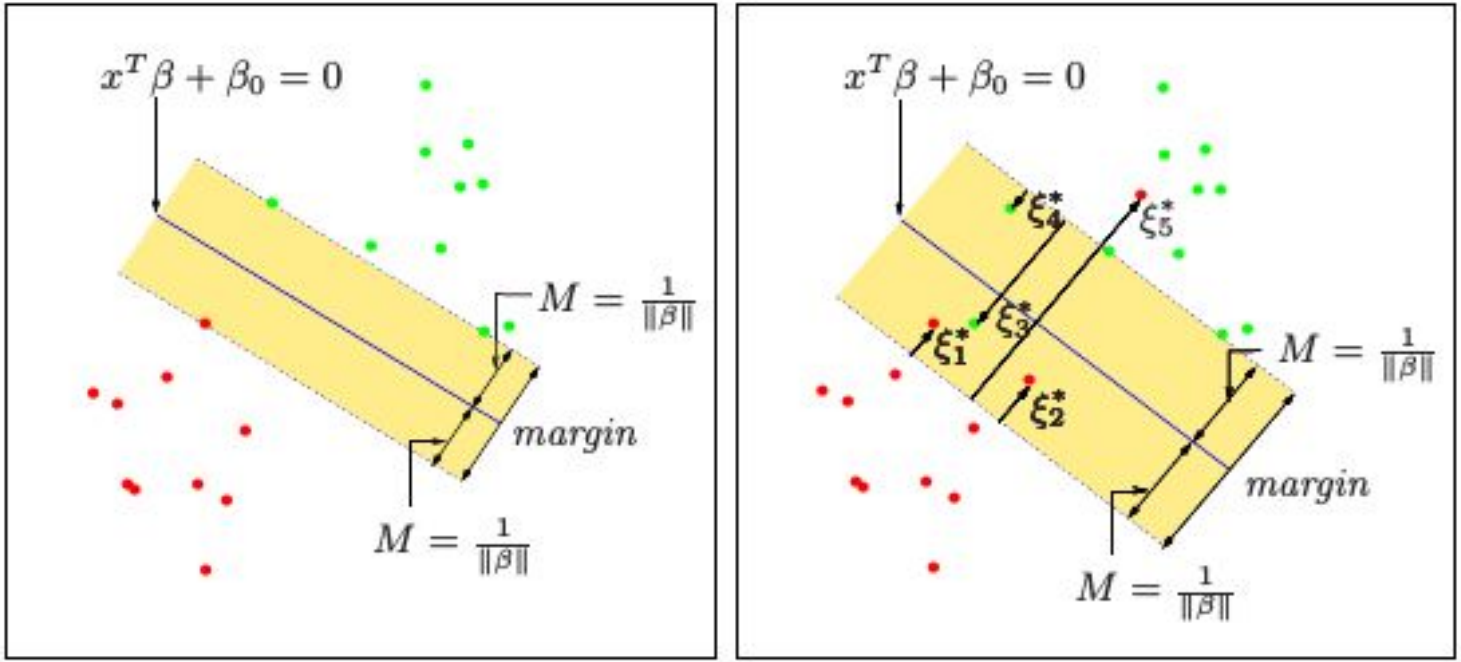
- For convenience, assume $y_i \subset \{-1,+1\}$
- Define hyperplane: $f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} + \beta_0$
  - $\mathbf{x}$ on plane: $f(\mathbf{x}) = 0$
  - $\mathbf{x}$ not on plane: $\texttt{sign}(f(\mathbf{x}))$ tells which side of the plane $\mathbf{x}$ is on
- Find optimal hyperplane as a decision boundary (maximum margin):

$$\text{minimize } \|\boldsymbol{\beta}\| + C\sum_i \xi_i$$

$$\text{subject to } y_i(\mathbf{x}^T\boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i,\ \xi_i \geq 0 \text{ for all } i$$

- Slack variables ξ_i allow some points to be misclassified
- C is a regularization constant: how much do I care about correctly classifying each point?

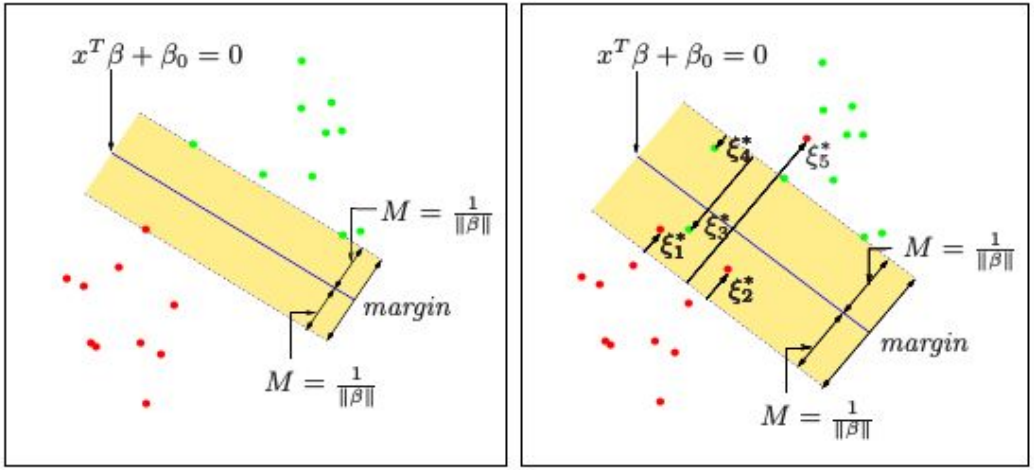
Optimal Decision Boundaries



[Hastie et al., The Elements of Statistical Learning]

Support Vector Machines

- Only points on the margin (for linearly separable data) or on the wrong side of the margine (for inseparable data) determine the classifier
- These points are referred to as “Support Vectors”



[Hastie et al., The Elements of Statistical Learning]

Regularization Parameter - Linear SVM

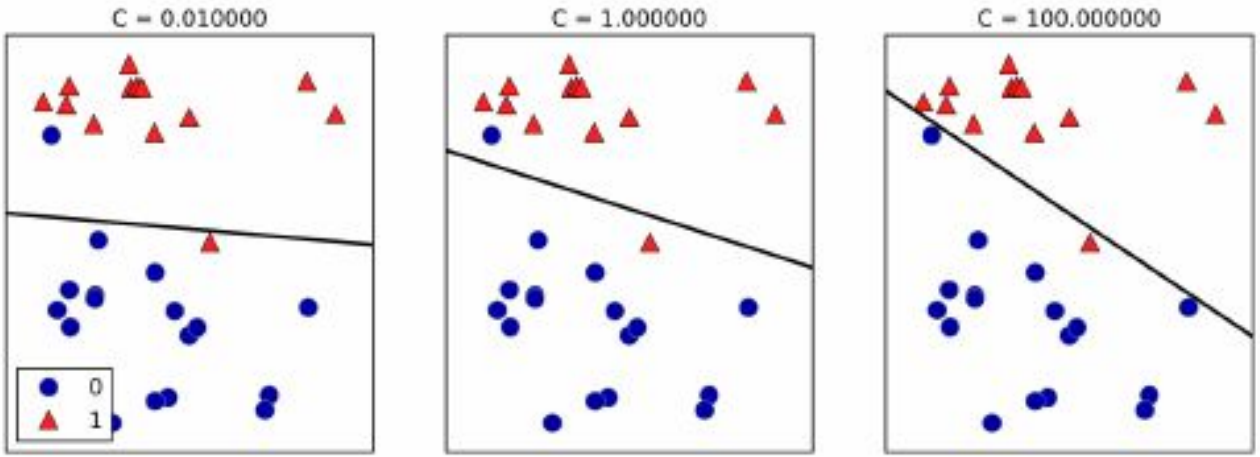


Figure 2-16. Decision boundaries of a linear SVM on the forge dataset for different values of C

[Muller and Guido, Introduction to Machine Learning with Python]

What About Non-linear Boundaries?

What About Non-linear Boundaries?

The Kernel Trick!!!

Kernelized SVMs

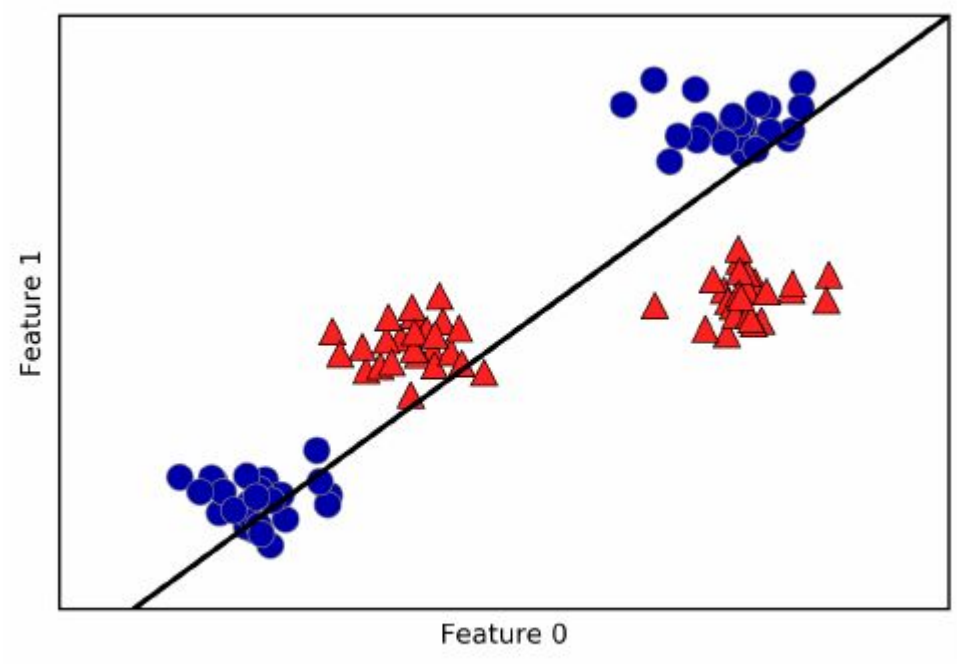


Figure 2-37. Decision boundary found by a linear SVM

[Muller and Guido, Introduction to Machine Learning with Python]

Kernelized SVMs

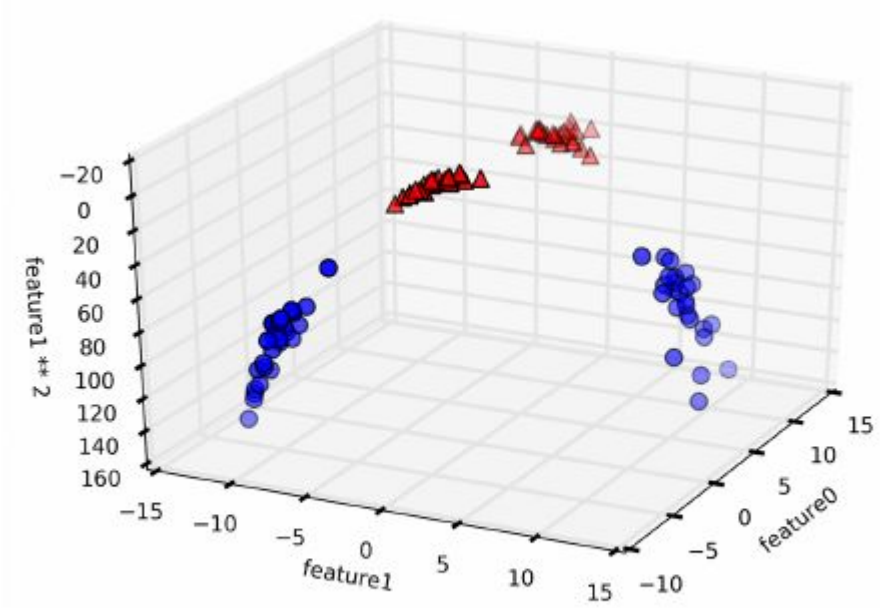


Figure 2-38. Expansion of the dataset shown in Figure 2-37, created by adding a third feature derived from feature1

[Muller and Guido, Introduction to Machine Learning with Python]

Kernel SVMs

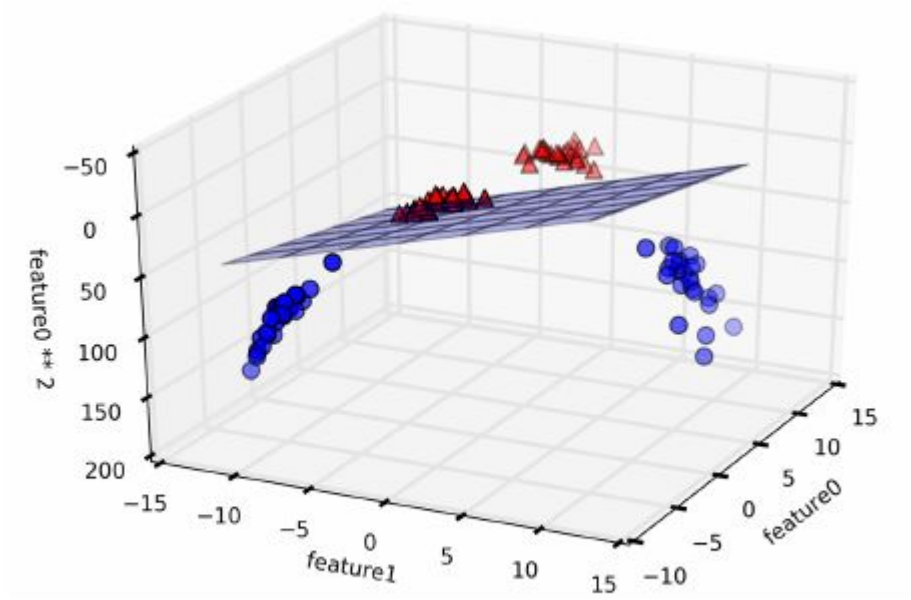


Figure 2-39. Decision boundary found by a linear SVM on the expanded three-dimensional dataset

[Muller and Guido, Introduction to Machine Learning with Python]

Kernel SVMs

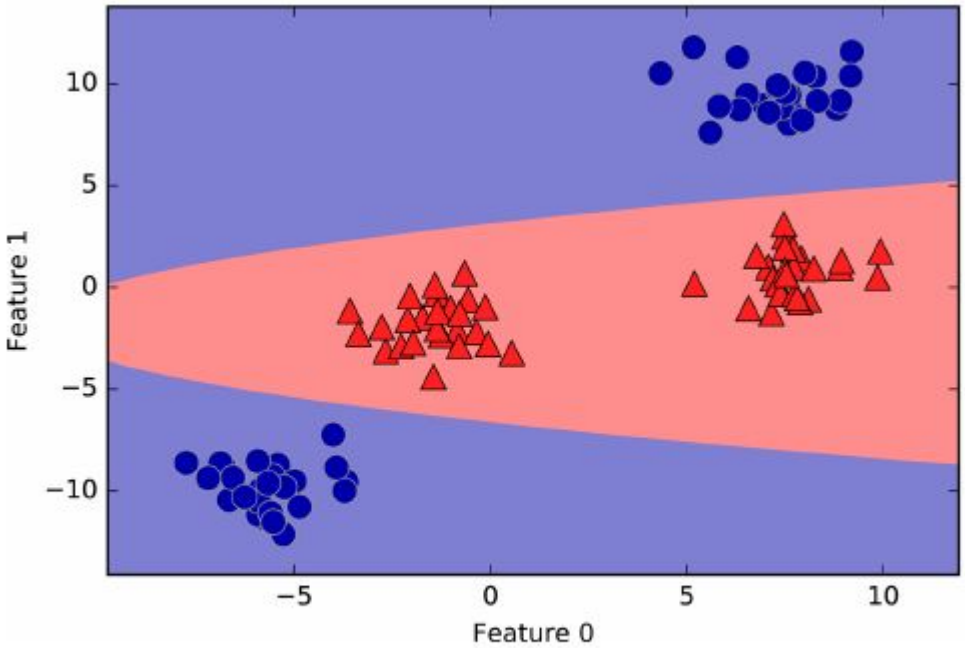


Figure 2-40. The decision boundary from Figure 2-39 as a function of the original two features

[Muller and Guido, Introduction to Machine Learning with Python]

Kernel SVMs

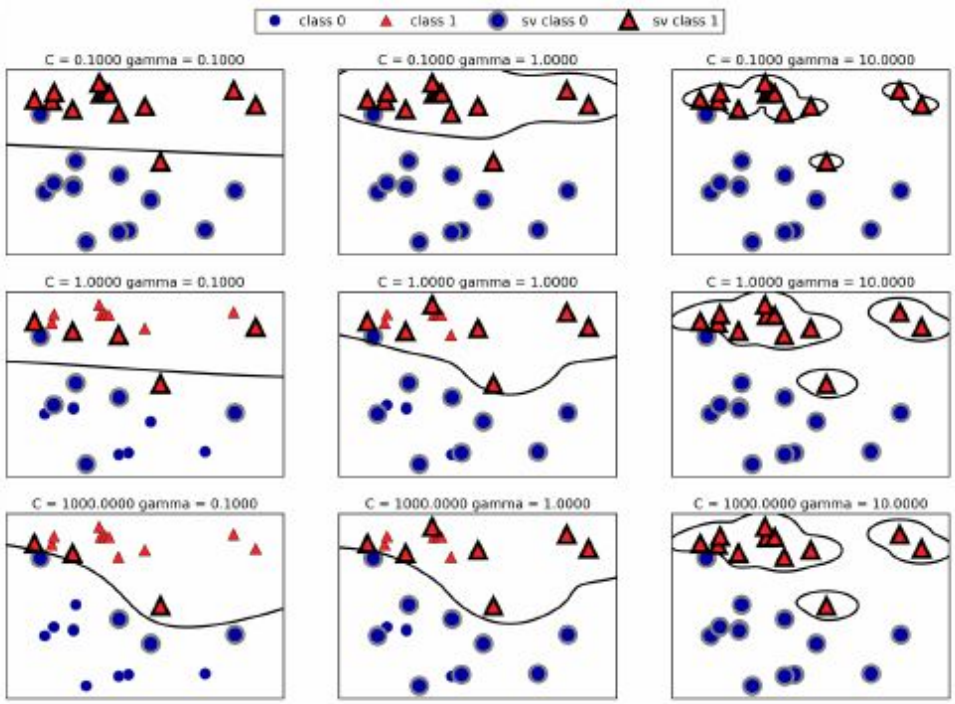


Figure 2-42. Decision boundaries and support vectors for different settings of the parameters C and gamma

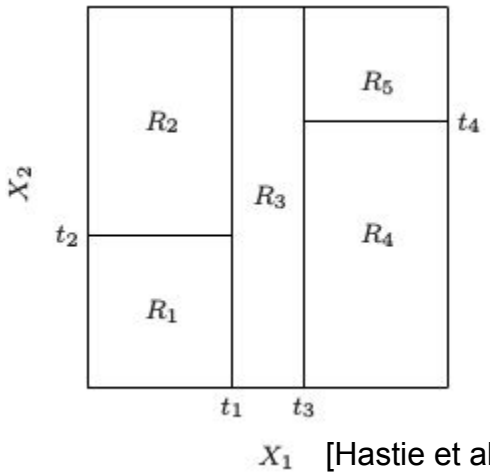
[Muller and Guido, Introduction to Machine Learning with Python]

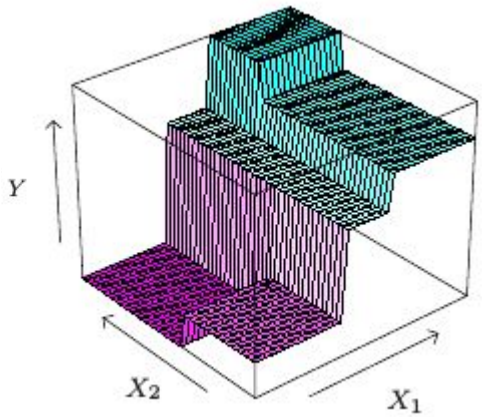
Kernel SVMs

- Regularization parameter has new intuition:
  - Low cost: “simpler” boundary in low-dim, might not fit all points
  - High cost: tries harder to fit all points, “complex” low-dim boundary

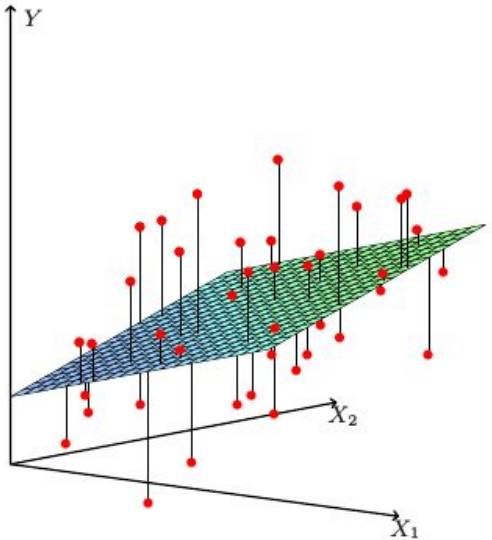
Regression and Classification Trees

Key idea: different models for different regions of the feature space





[Hastie et al., The Elements of Statistical Learning]

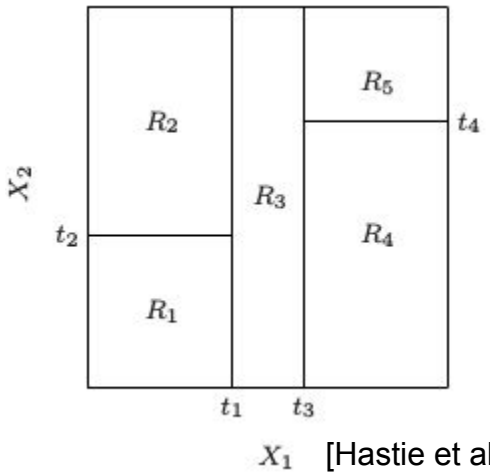


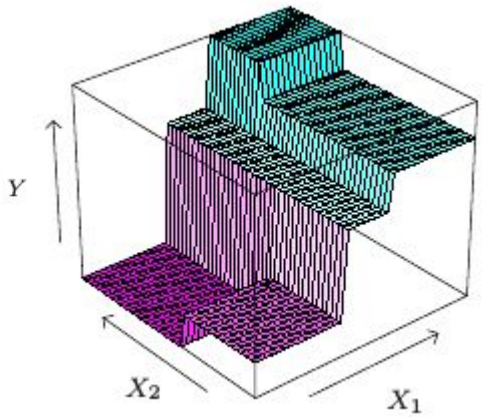
Tree: each region of feature space gets its own model. In this example, each model is just a constant value.

Linear regression: one single model over entire feature space

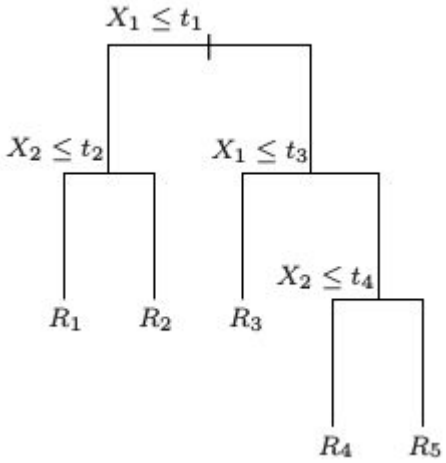
Regression and Classification Trees

Key idea: different models for different regions of the feature space





[Hastie et al., The Elements of Statistical Learning]

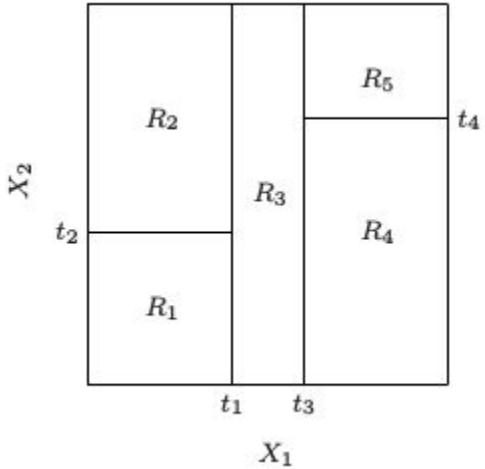


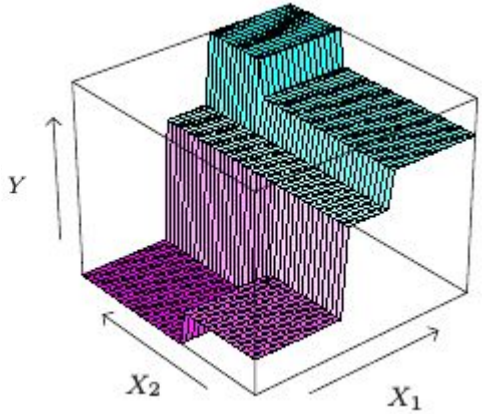
Tree: each region of feature space gets its own model. In this example, each model is just a constant value.

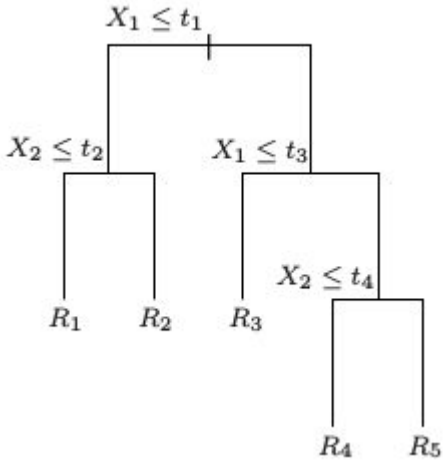
Feature space can be partitioned using binary splitting (very common).

Regression and Classification Trees

Key idea: different models for different regions of the feature space







- How do I choose partitioning thresholds?
- When do I stop partitioning?
- What model do I use in each region?

Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- What model do I use in each region?

Constant model most commonly used: $c_m = \texttt{avg}(y_i \mid \mathbf{x}_i \subset R_m)$

Prediction: $y_{new} = c_m \mid \mathbf{x}_{new} \subset R_m$

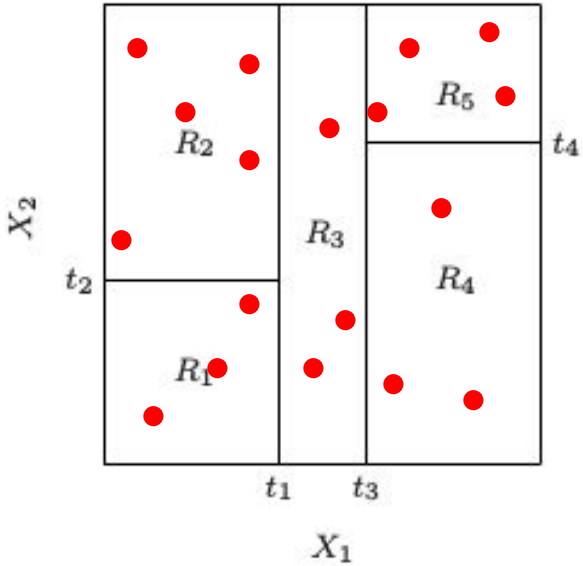
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



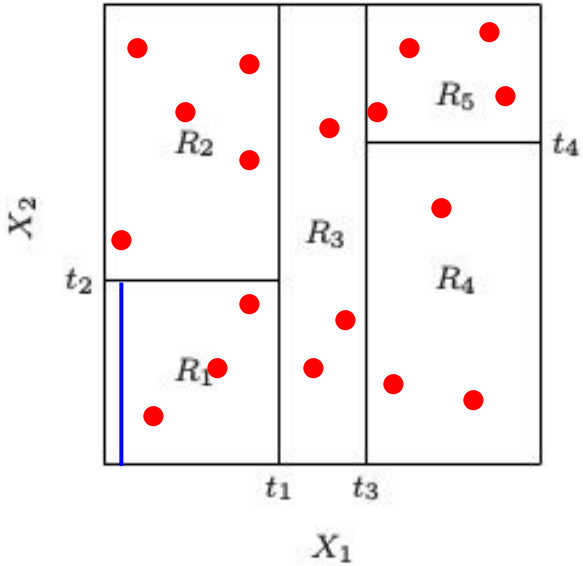
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



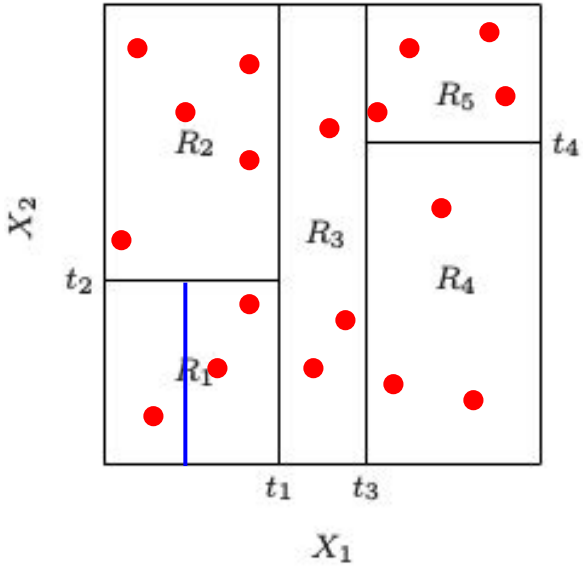
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



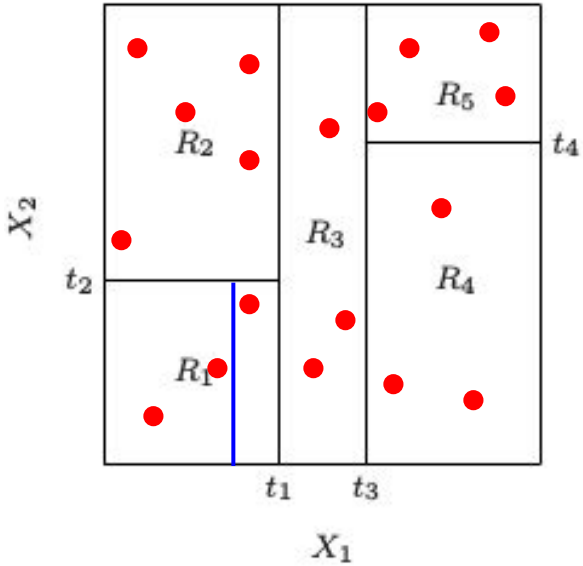
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

- Repeat until all leaves are "small enough"
  - For each leaf:
    - For each possible splitting dimension:
      - For each possible split point:
        - Compute error reduction obtained from split
  - Choose split with greatest error reduction



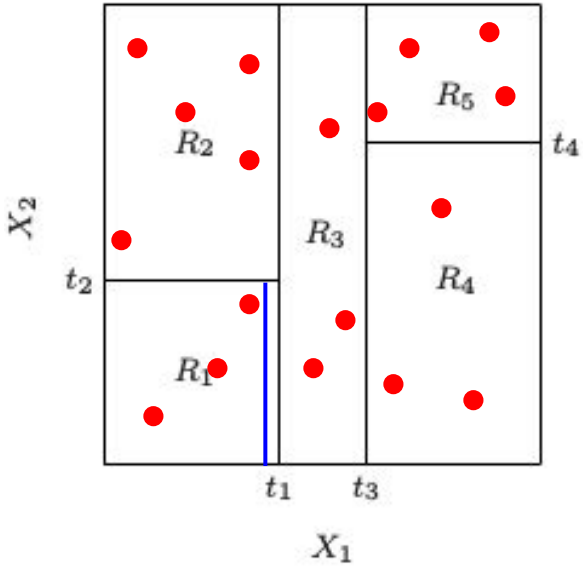
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



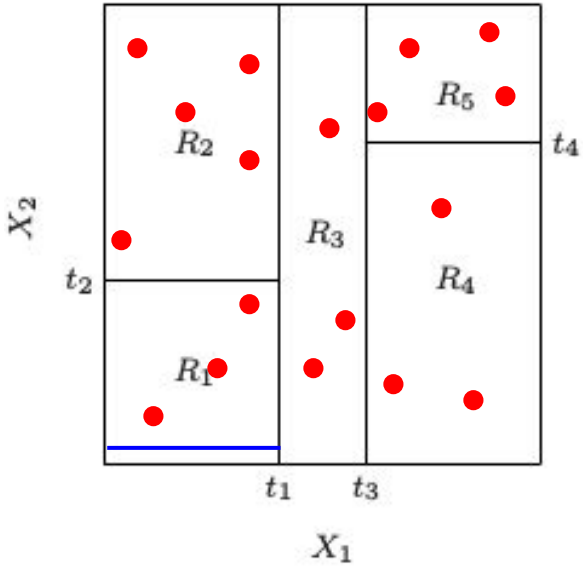
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

- Repeat until all leaves are "small enough"
  - For each leaf:
    - For each possible splitting dimension:
      - For each possible split point:
        - Compute error reduction obtained from split
  - Choose split with greatest error reduction



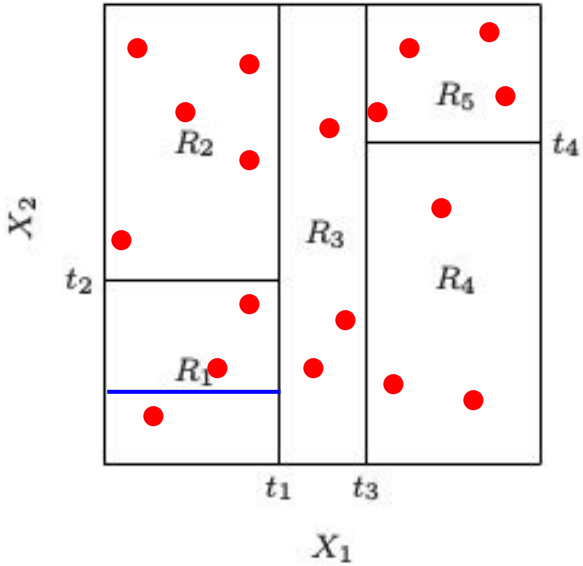
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



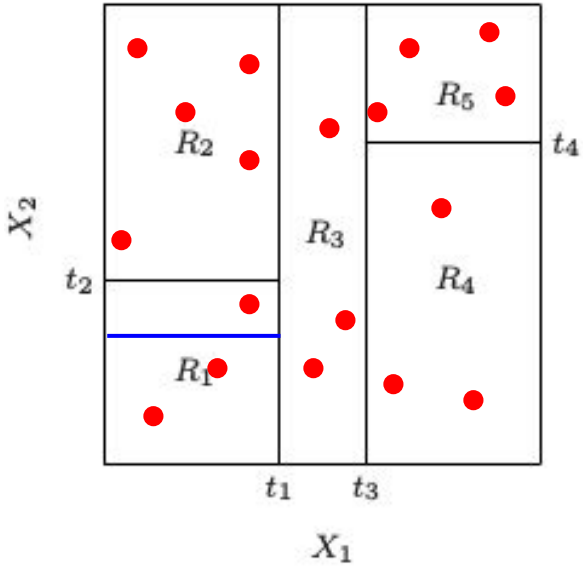
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



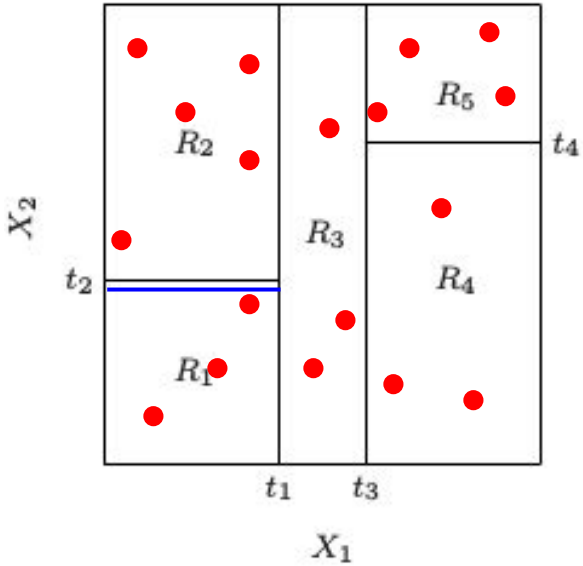
Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.
Use greedy algorithm:

```
- Repeat until all leaves are "small enough"
    - For each leaf:
        - For each possible splitting dimension:
            - For each possible split point:
                - Compute error reduction
                  obtained from split
    - Choose split with greatest error reduction
```



Regression Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

- Constant model in each region: $c_m = \texttt{ave}(y_i \mid \mathbf{x}_i \subset R_m)$
- How do I partition the space?

Globally optimal partition is intractable to compute.

Use greedy algorithm to partition top-down.

Prune bottom-up to desired tree size:

- `Repeat until desired tree size is reached:`
  - `Merge adjacent leaves that lead to smallest increase in fitting error.`

Classification Trees

Assume feature space partitioned into M regions: $R_1, R_2, \ldots, R_M$

Prediction: $y_{new} = \texttt{argmax}_k \, [\text{count}(y_i = k \mid \mathbf{x}_i \subset R_m)]$ where $\mathbf{x}_{new} \subset R_m$

Example: Spam Classification

- Classify a piece of text as email/spam

% of words that are “business”
% of words that are “address”
…. (48 in total)

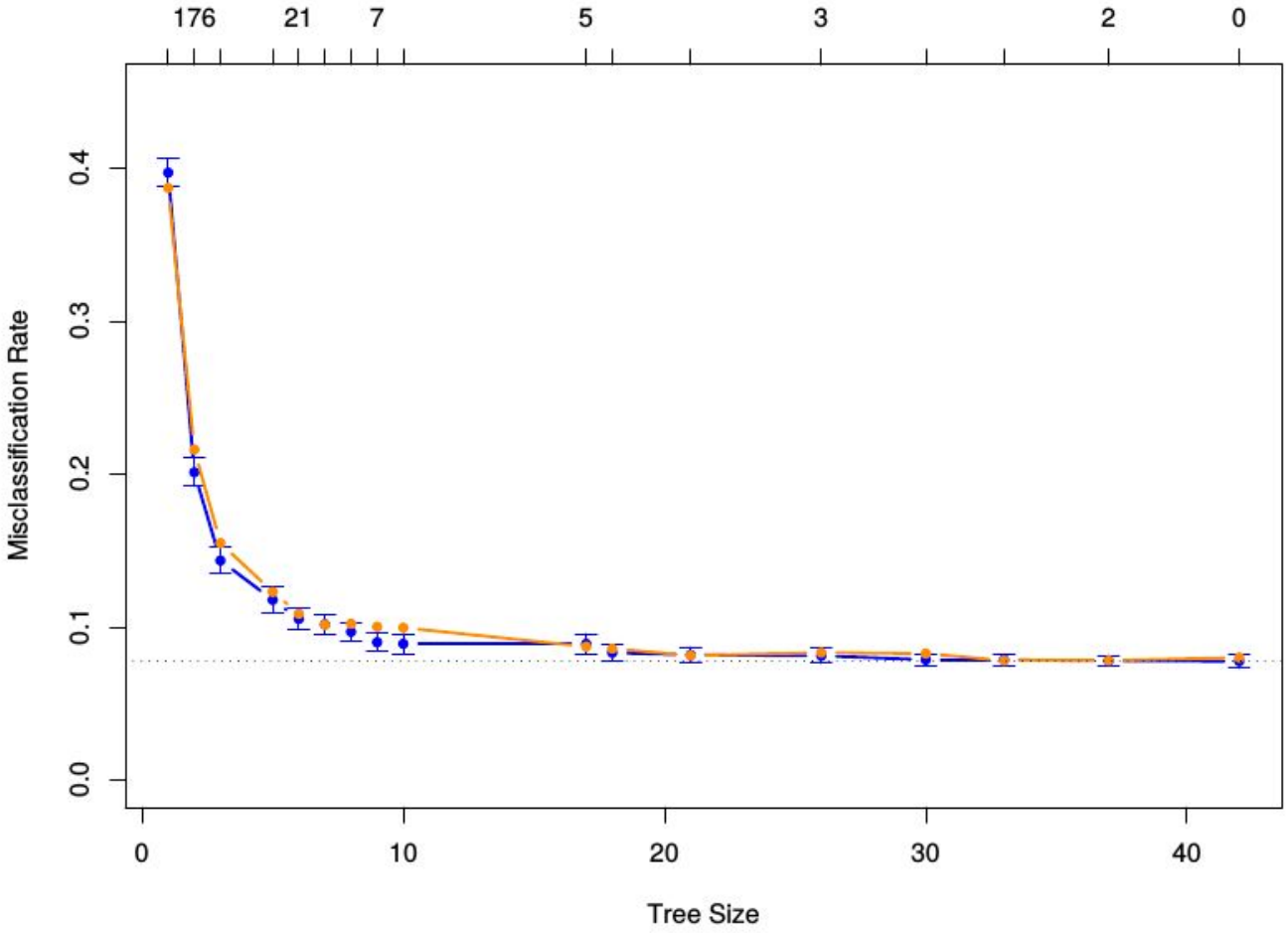
% of chars that are ;
% of chars that are (
… (6 in total)

length of uninterrupted sequence of capital letters: AVG, MAX, TOT
(3 in total)

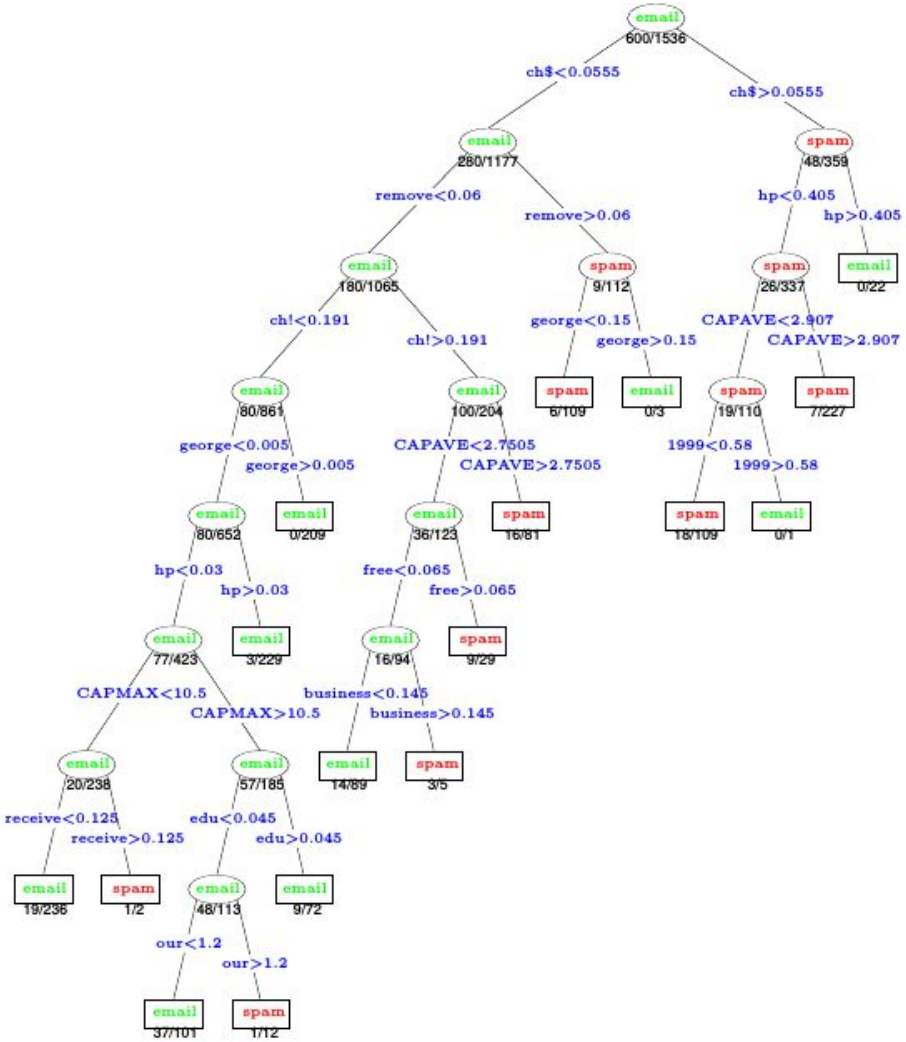
57-dimensional feature space

email / spam

- 4,601 labeled examples, split into training set (3,065) and testing set (1,536)



[Hastie et al., The Elements of Statistical Learning]


email
600/1536
ch$<0.0555
ch$>0.0555
email
280/1177
spam
48/359
remove<0.06
remove>0.06
hp<0.405
hp>0.405
email
180/1065
spam
9/112
spam
26/337
email
0/22
ch!<0.191
ch!>0.191
george<0.15
george>0.15
CAPAVE<2.907
CAPAVE>2.907
email
80/861
email
100/204
spam
6/109
email
0/3
spam
19/110
spam
7/227
george<0.005
george>0.005
CAPAVE<2.7505
CAPAVE>2.7505
1999<0.58
1999>0.58
email
80/652
email
0/209
email
36/123
spam
16/81
spam
18/109
email
0/1
hp<0.03
hp>0.03
free<0.065
free>0.065
email
77/423
email
3/229
email
16/94
spam
9/29
CAPMAX<10.5
CAPMAX>10.5
business<0.145
business>0.145
email
20/238
email
57/185
email
14/89
spam
3/5
receive<0.125
receive>0.125
edu<0.045
edu>0.045
email
19/236
spam
1/2
email
48/113
email
9/72
our<1.2
our>1.2
email
37/101
spam
1/12

Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

[Hastie et al., The Elements of Statistical Learning]

Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

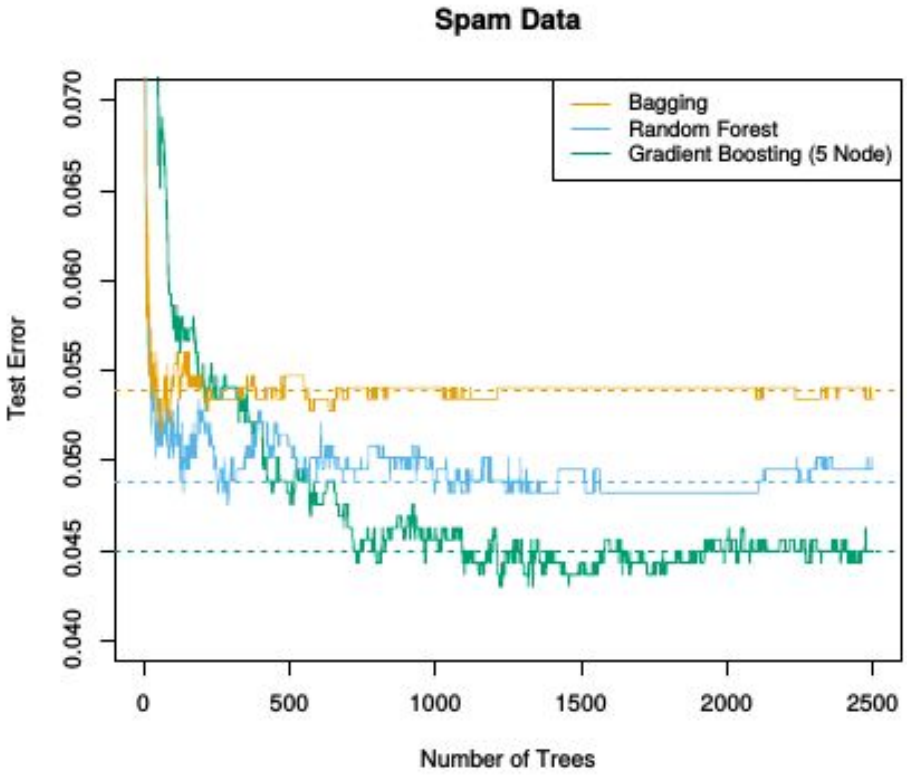
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

Reduce correlation between individual trees

[Hastie et al., The Elements of Statistical Learning]

Random Forests



Boosting and Bagging

Train a “committee” of models, then aggregate their predictions.

- Simple example: each model trained on a subset of training data
- Aggregation can come in many flavors: simple average, weighted vote, etc.
- Each model is considered “weak”, their aggregate is “strong”

Both techniques work best when individual models are uncorrelated.

Hyperparameters

- Ridge regression: λ minimize $\sum_i (y_i - \mathbf{w}\mathbf{x}_i)^2 + \lambda \sum_j w_j^2$
- SVM: C minimize $\|\boldsymbol{\beta}\| + C\sum_i \xi_i$
- Classification / Regression Trees
  - initial tree depth
  - tree size after pruning
- Random Forests: number of trees
- ….

- Tuning parameters / hyperparameters
  - they are not optimized during training/fitting
  - they usually govern the training process, or regulate model capacity