Machine Learning I

Lecture 13: Decision Trees and Support Vector Machines

Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

Decision Trees

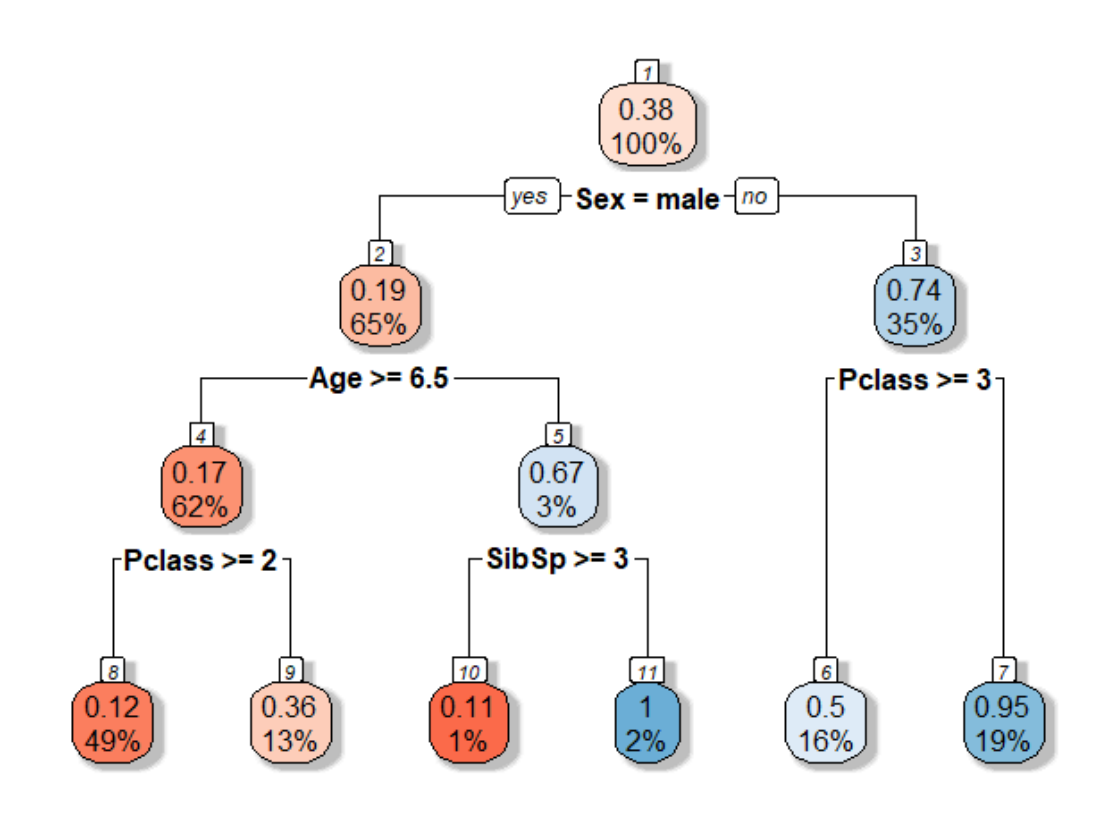
Decision Tree Example: Titanic Dataset

Pclass	Name	Sex	Age	SibSp	Parch	Emb	Sur
3	Kelly, James	m	34.5	0	0	Q	0
3	Connolly, Kate	f	30.0	0	1	S	0
2	McFerson, Jack	m	53.0	0	1	S	1

In the titanic dataset, we try to predict survivors (**Sur**) based on (**Age**), ticket class (**Pclass**), **sex**, number of siblings/spouse present (**SibSp**), parents or children (**Parch**) and embarkment port (**Emb**).

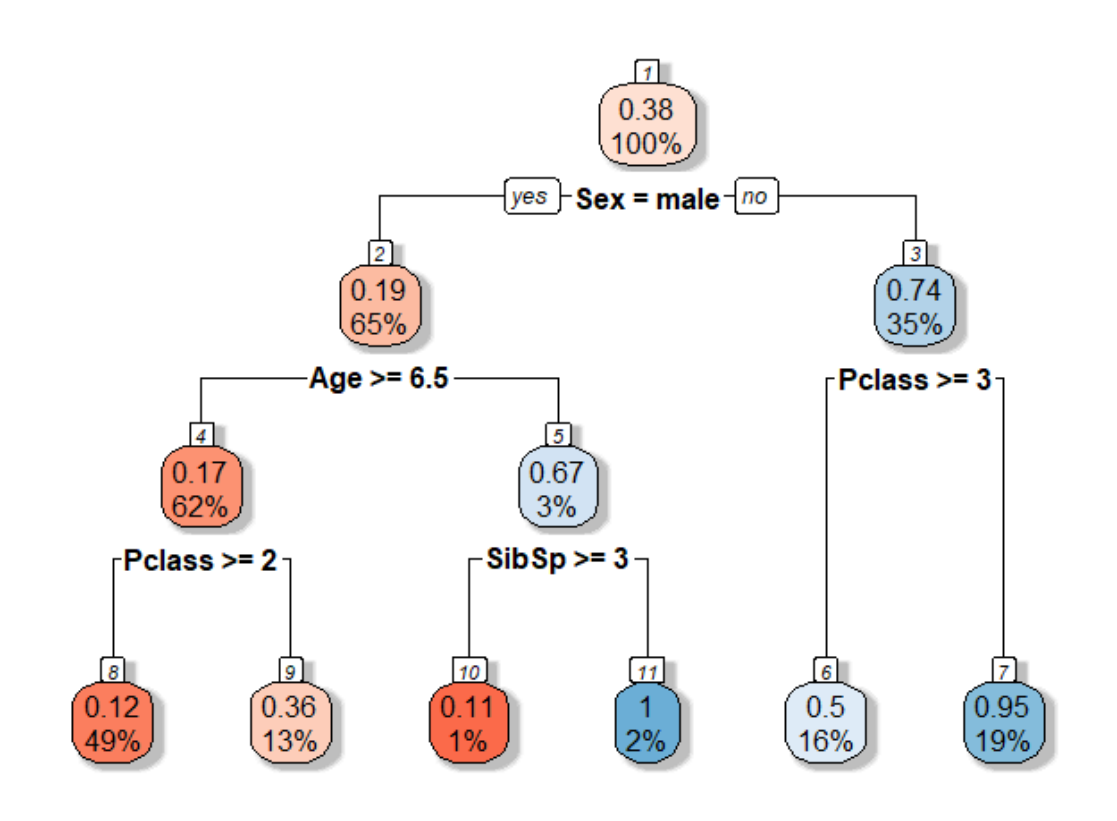
Tree based methods provide clear explanations of their fitting methodology. A binary decision tree classifies the data based on partitioning each feature into two sets at each step.

Decision Tree Example: Titanic Dataset



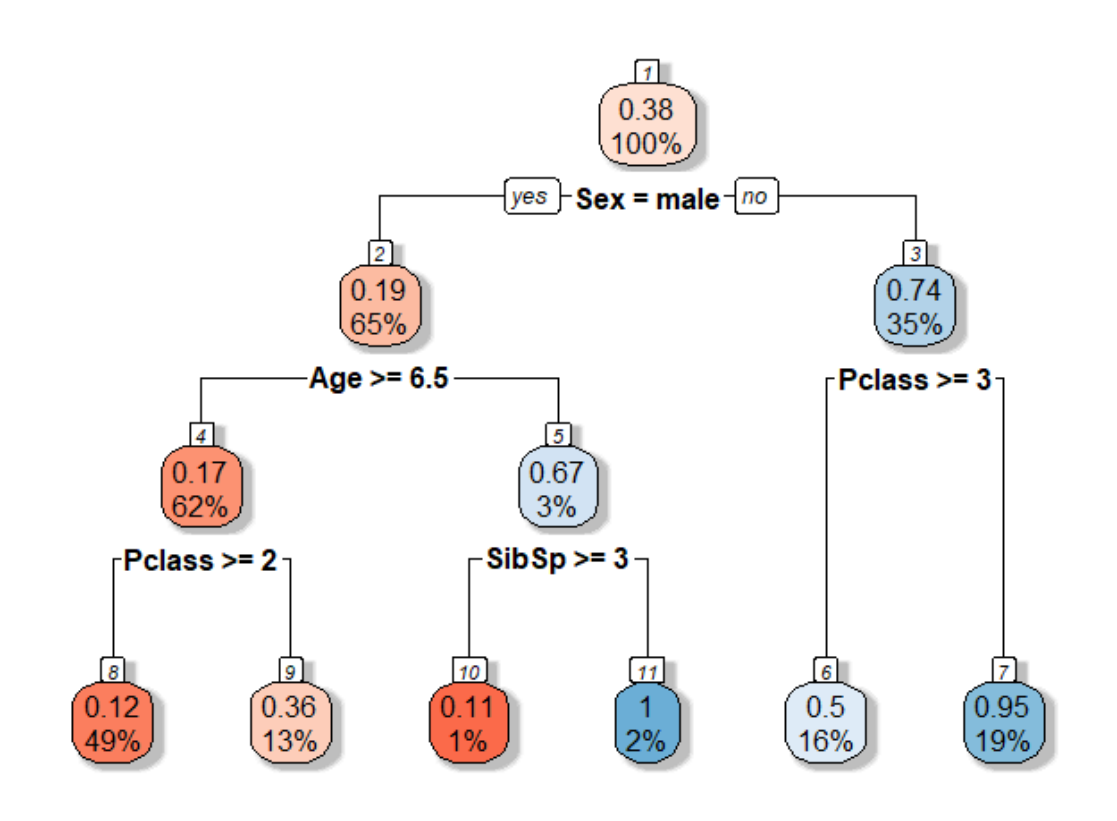
Pclass	Name	Sex	Age	SibSp	Parch	Emb	Sur
3	Kelly, James	m	34.5	0	0	Q	0
3	Connolly, Kate	f	30.0	0	1	S	0
2	McFerson, Jack	m	53.0	0	1	S	1

Decision Tree Example: Titanic Dataset



Each node gives the proportion of people at that node that survived as a decimal, and the percent of to total population that node accounts for. The first node says overall there is a .38 chance of surviving.

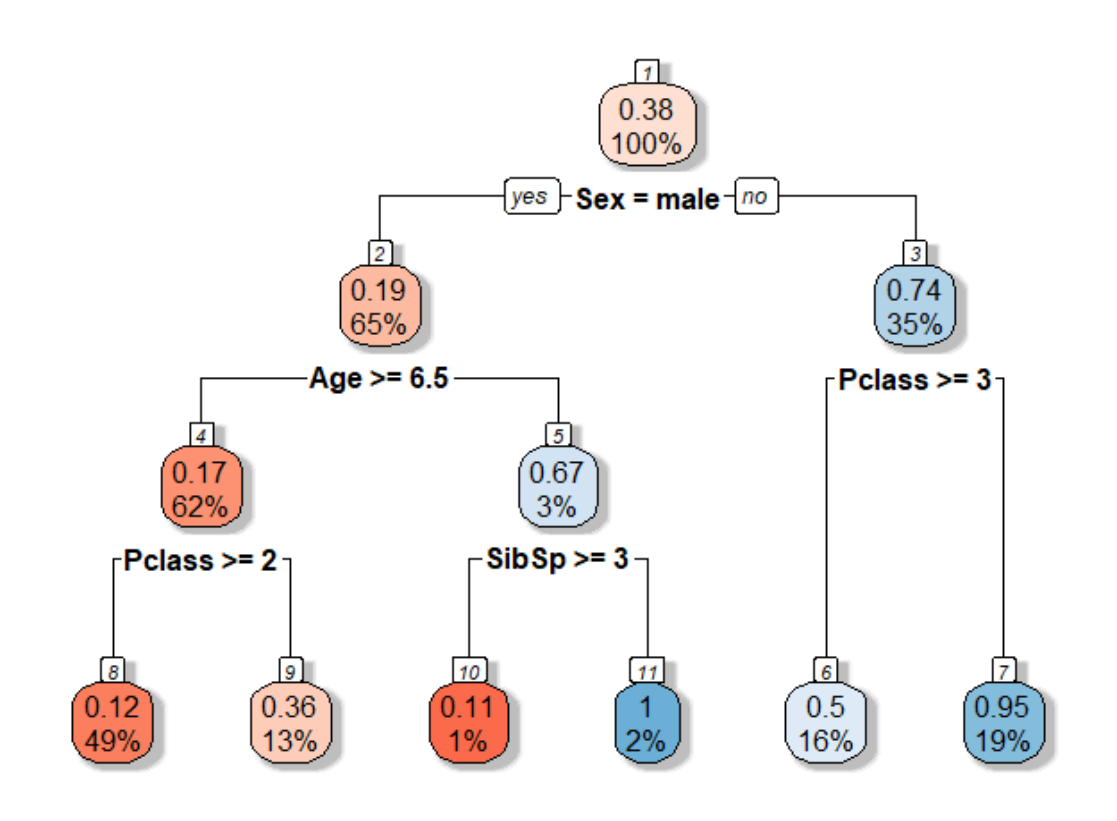
Decision Tree Example: Titanic Dataset



The first splitting tells us that there is a 65% of being male, and males had .19 chance of survival, while there is a 35% chance of being female and females have a .74 chance of surviving.

Then, on the right branch, we see that for females not of third class, the survival rate goes up to .95, but for third class woman it was merely .5.

Decision Tree Example: Titanic Dataset



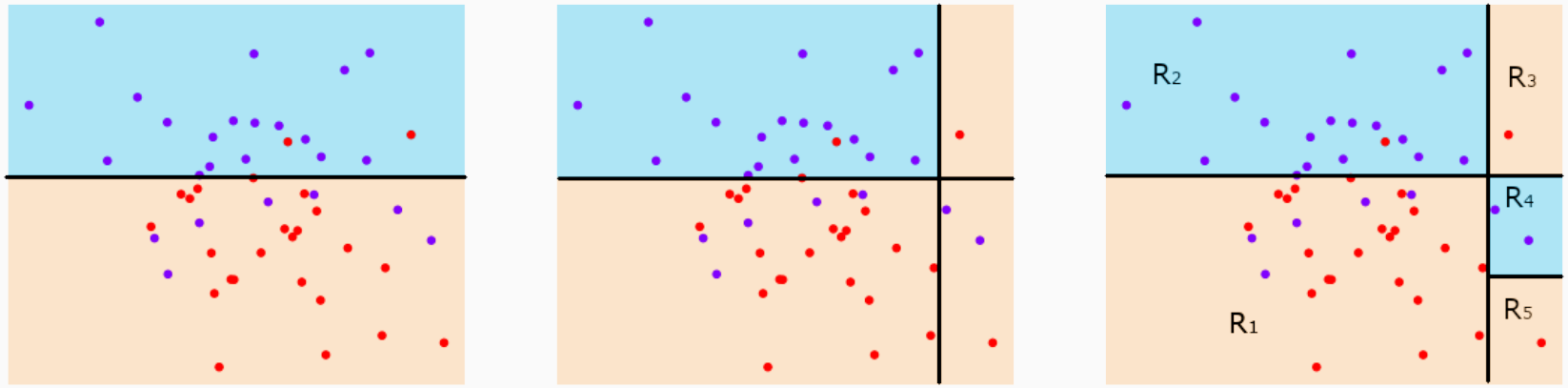
A decision tree gives us a clear organization of the linear choices being made to classify the data, but it comes at a cost: decision trees have a high potential to over fit, especially to sparse noisy data. Therefore we need to understand what the algorithm is doing and what kind of control we have over it.

Splitting on Features

We will describe the CART (Classification And Regression Tree) method of decision tree fitting. The CART model fits a decision tree by recursive splitting along axis aligned dimensions. That is, it splits the data into two region along one feature and models the response by taking the mean label (or probability of each label for categorical data) in each region. We choose the variable and split point to achieve the best fit.

Then, one or both of the regions is split again, and this continues until a stopping rule is applied. CART employees a greedy algorithm, simply selecting the beat feature to split on at each step.

Splitting on Features



For example, on two features X_1 and X_2, we can actually visualize the splitting. For each region above, the probability of each label in each region is given the by the mean label expression in that region. The model is

$$\hat{f}(X) = \sum_{m=1}^{5} c_m \mathbb{1}_{R_m}(X), .$$

Fitting Regression Trees

Let's now turn to fitting a regression tree to a binary labeling:

Given training data (x_i, y_i) with p features $X_1, \ldots, X_p$, let us choose the loss function to be MSE and attempt to fit

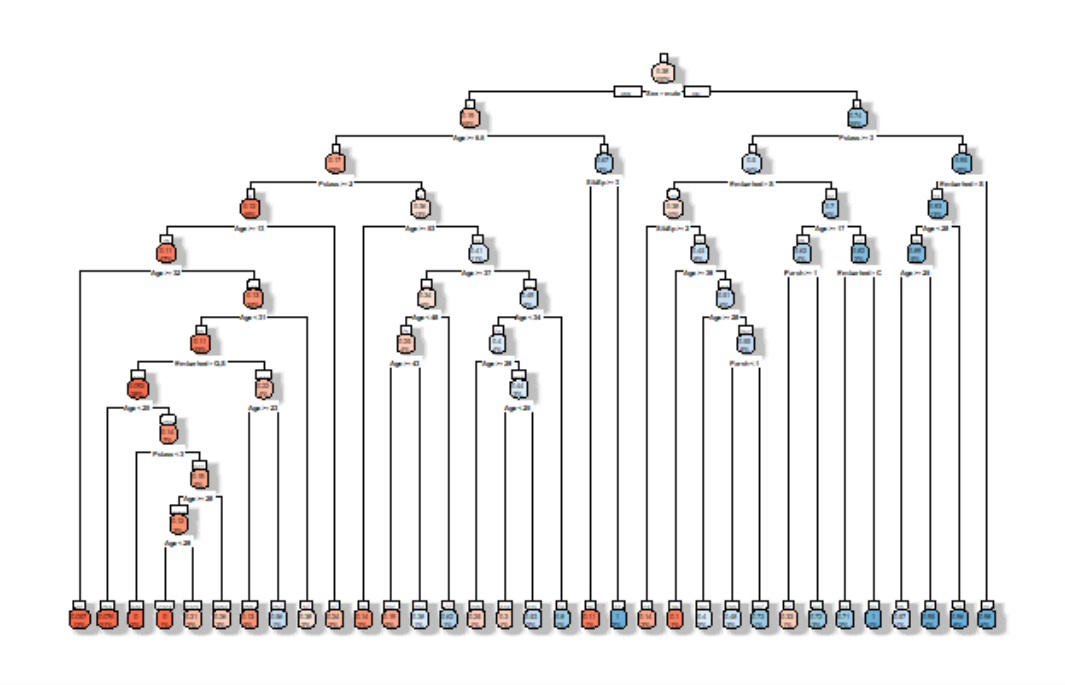
$$\hat{f}(X) = \sum_{m=1}^{M} c_m \mathbb{1}_{R_m}(X)\,.$$

We pick $c_m = \text{mean}(y_i | x_i \in R_m)$ to be the mean y_i in the region. At the first step, splitting $\mathcal{X}$ to minimize loss is the same as finding $R_1(j, s) = \{X | X_j \leq s\}$ and $R_2\{X | X_j > s\}$ such that

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

For any (j, s), the inner part is minimized by $\hat{c}_k = \text{mean}(y_i | x_i \in R_k(j, s))$. Fitting the tree can be done by a series of mj 1-variable regressions.

Bias-Variance Trees



Clearly, if we grow the tree large enough we will over-fit by effectively cutting $\mathcal{X}$ into N regions, each one containing a single data point. In fact, a bit of thought shows that the VC dimension of binary classification using a tree with M terminal nodes is 2^M, so although it may be tempting to fit to a large tree the variance increases dramatically with the number of nodes. Note, that a general tree with k levels will have 2^{k-1} terminal nodes.

Pruning Regression Trees

Sometimes, good splits are hidden beneath less interest ones. **Pruning**, rather than stopping growth, lets us find these without increasing variance too much.

Let T be any subtree of the fit tree $\hat{T}$ obtained by contracting a nonterminal node and define. In **cost-complexity pruning**, m is the index the terminal nodes in T and the regions R_m. Let N_m be the number of training points in each R_m and let $\hat{c}_m$ be the average label in R_m. Let

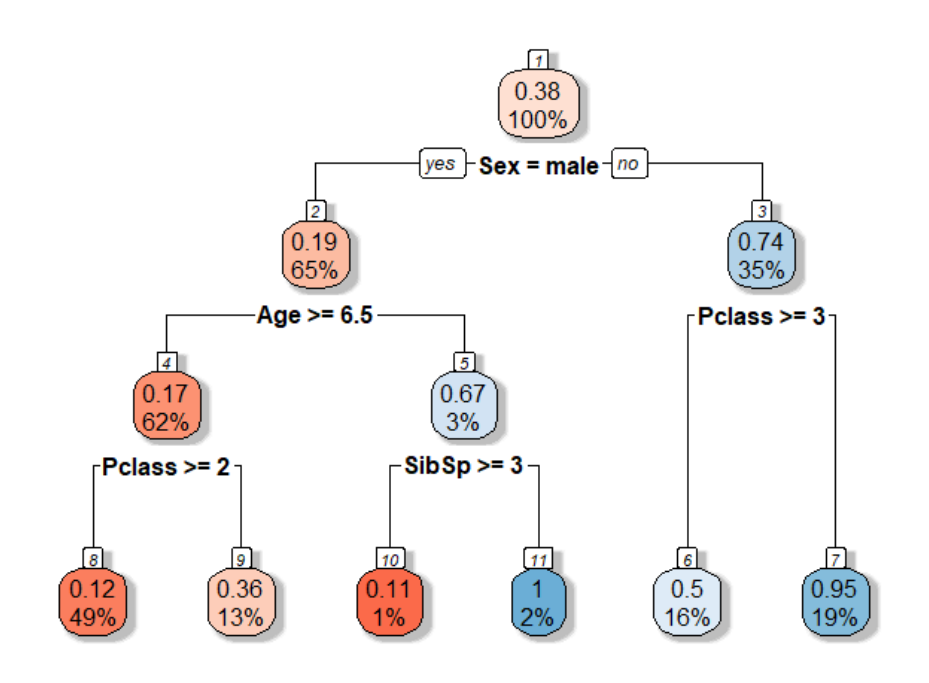
$$Q_m(T) = \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

be the un-normalized error in R. Then the **cost-complexity criteria** is that we minimize

$$C_\alpha(T) = \sum_{m=1}^{|T|} Q_m(T) + \alpha|T|\,,$$

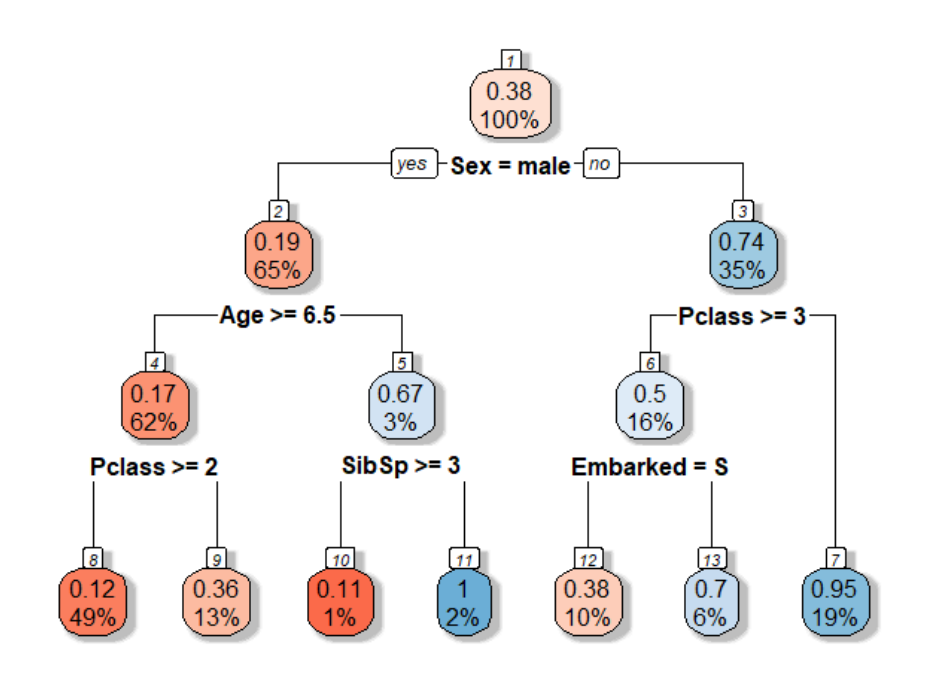
where $|T|$ is the number of terminal nodes remaining in T.

Bias-Variance Trees



For example, lets compare our tree grown for the titanic data

Bias-Variance Trees



For example, lets compare our tree grown for the titanic data, with the pruning of the much larger tree found before. We have found a new nontrivial split in the data.

Classification

To perform classification with a random tree, we need only specify the target error and the splitting criteria. Instead of trying to minimize $\frac{1}{N_m} Q_m$, for each region R_m, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}(y_i = k)$$

be the sample probability of label k in region m. Letting $k(m) = \text{argmax}_k(p_{mk})$ be the most probable label for node m, node impurity can be measured by

Misclass. Error: $$1 - \hat{p}_{mk(m)} = \frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}(y_i = k(m))$$

Gini Index: $$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Cross-entropy: $$\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Splitting Loss Functions

For example, for two classes L and R with 40 observations in each class, say a split creates nodes $N_1 = (30, 10)$ and $N_2 = (10, 30)$ while another node creates $N_3 = (20, 40)$, $N_4 = (20, 0)$. Then

$$\hat{p}_{1L} = .75 \qquad \hat{p}_{2L} = .25 \qquad \hat{p}_{3L} = .333 \qquad \hat{p}_{4L} = 1$$
$$\hat{p}_{1R} = .25 \qquad \hat{p}_{2R} = .75 \qquad \hat{p}_{3R} = .666 \qquad \hat{p}_{4R} = 0$$

Then, the misclassification error as splitting 1 is

$$\boxed{\text{Split 1:}} \qquad \frac{N_1}{N}(1 - \hat{p}_{1k(1)}) + \frac{N_2}{N}(1 - \hat{p}_{2k(2)}) = \frac{40}{80}.25 + \frac{40}{80}.25 = .25\,,$$

and at splitting 2,

$$\boxed{\text{Split 2:}} \qquad \frac{N_3}{N}(1 - \hat{p}_{3k(3)}) + \frac{N_4}{N}(1 - \hat{p}_{4k(4)}) = \frac{60}{80}.333 + \frac{20}{80}1 = .25\,.$$

Splitting Loss Functions

For example, for two classes L and R with 40 observations in each class, say a split creates nodes $N_1 = (30, 10)$ and $N_2 = (10, 30)$ while another node creates $N_3 = (20, 40)$, $N_4 = (20, 0)$. Then

$$\hat{p}_{1L} = .75 \qquad \hat{p}_{2L} = .25 \qquad \hat{p}_{3L} = .333 \qquad \hat{p}_{4L} = 1$$
$$\hat{p}_{1R} = .25 \qquad \hat{p}_{2R} = .75 \qquad \hat{p}_{3R} = .666 \qquad \hat{p}_{4R} = 0$$

On the other hand, the Gini Index $\sum_{j=1,2} \frac{N_j}{N} \sum \hat{p}_{jk}(1 - \hat{p}_{jk})$

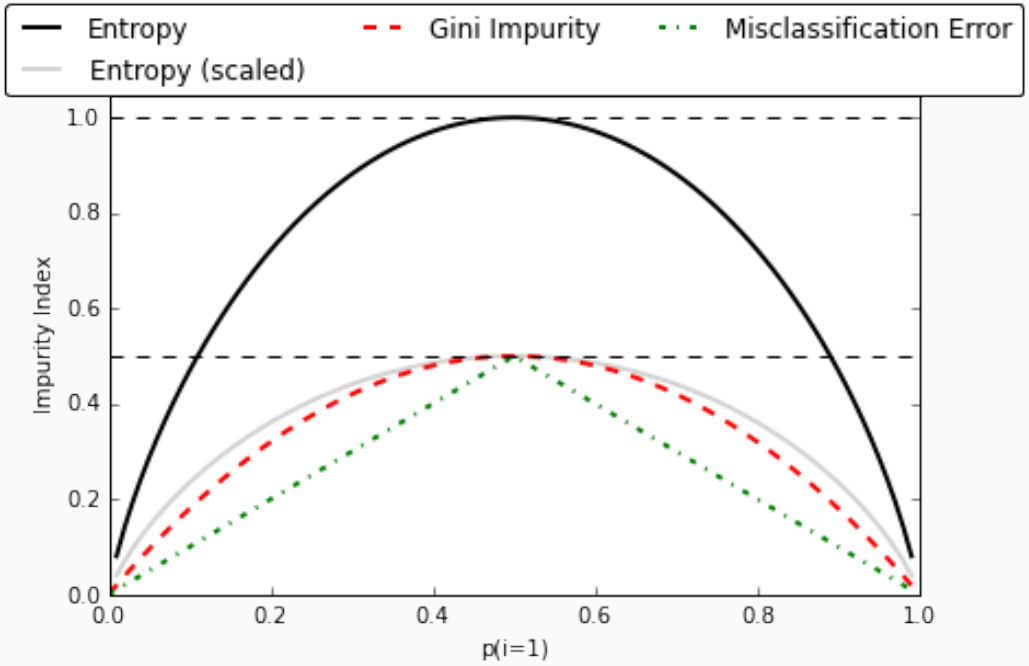
Split 1: $\frac{40}{80}[(.75)(.25) + (.75)(.25)] + \frac{40}{80}[(.75)(.25) + (.75)(.25)] = 0.375\,,$

and at splitting 2,

Split 2: $\frac{60}{80}[(.33)(.66) + (.66)(.33)] + \frac{20}{80}[1 \cdot 0 + 0 \cdot 1] = .33\,.$

The Gini index and cross-entropy are lower for the second splitting.

Bias-Variance Trees



In addition to the Gini index and cross entropy being lower for pure nodes, cross-entropy and the Gini index are more sensitive to the misclassification rate of each node. The are also smooth as we see above. This is why they are most often used in algorithms for decision trees.

Random Forest Models

While random tree models offer explanatory power, we've seen that they also have a very high variance. There are a few ways to diminish this, but a popular on is to use a **random forest classifier.**

A random forest fits by growing ℓ random trees, training each tree on a set of N training data points drawn uniformly from the training data (on average this will contain $\approx 63.2\%$ of the data). It then allows each of the trees to vote on a result for classification, or averages the results for regression.

A random forest as an example of an **ensemble classifier**, which we will say more about later.

Support Vector Machines

A Textual Example

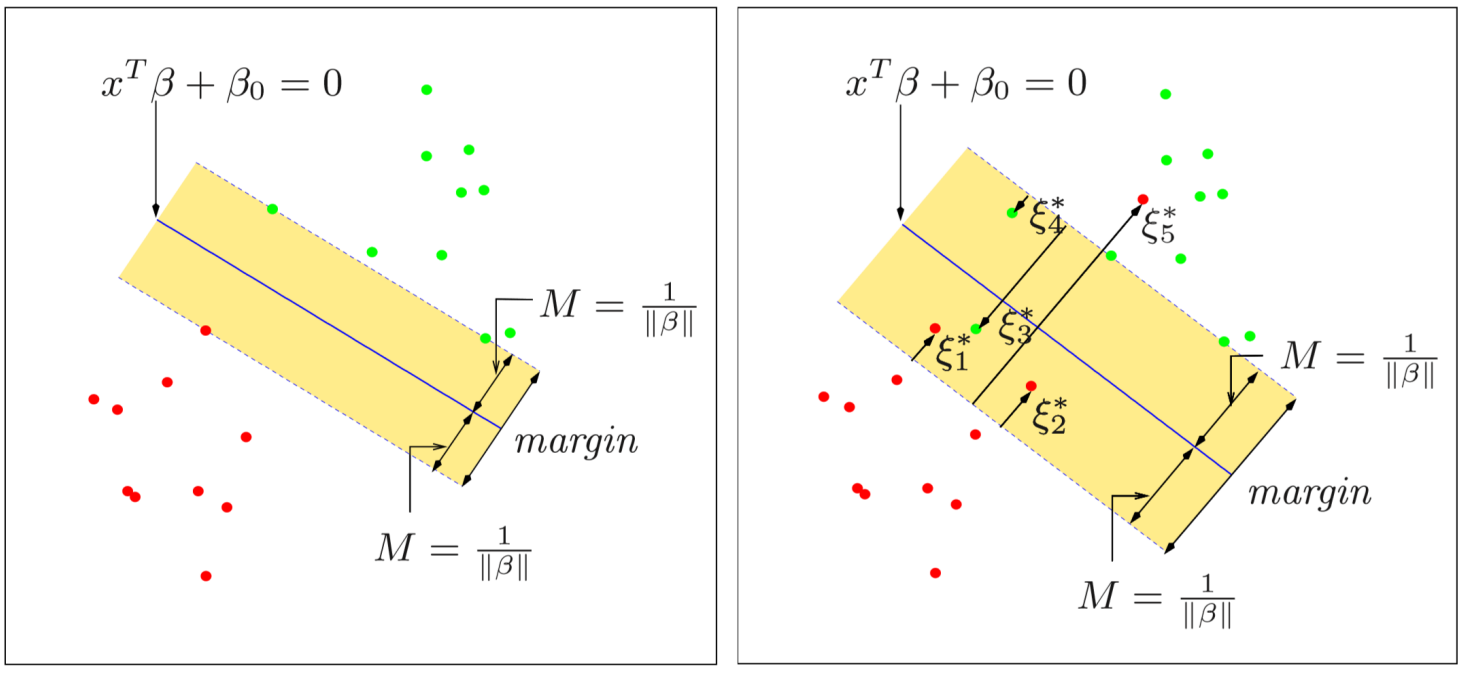
What an adventure!
12 October 2018 | by mariadob – See all my reviews

It's a movie with amazing actors who all made me laugh through the whole movie. Everything is an amazing adventure and its totally made to all the people who loves a bit spooky or scary comic horrormovie

Lets consider a simple text analysis problem, something like taking a large database of movie reviews and predicting sentiment from them (positive or negative). To organize the words, we encode them in a one-hot vector and train a classifier to classify string of them as positive, negative, or neither.

However, our feature set is now huge $d =$(# of words)(length of review). A simple halfspace classifier has VC dimension $d + 1$, making prohibitive amounts of data necessary for classification. In addition, k nearest neighbors may be impossible to compute due to high dimensionality.

Support Vector Machines

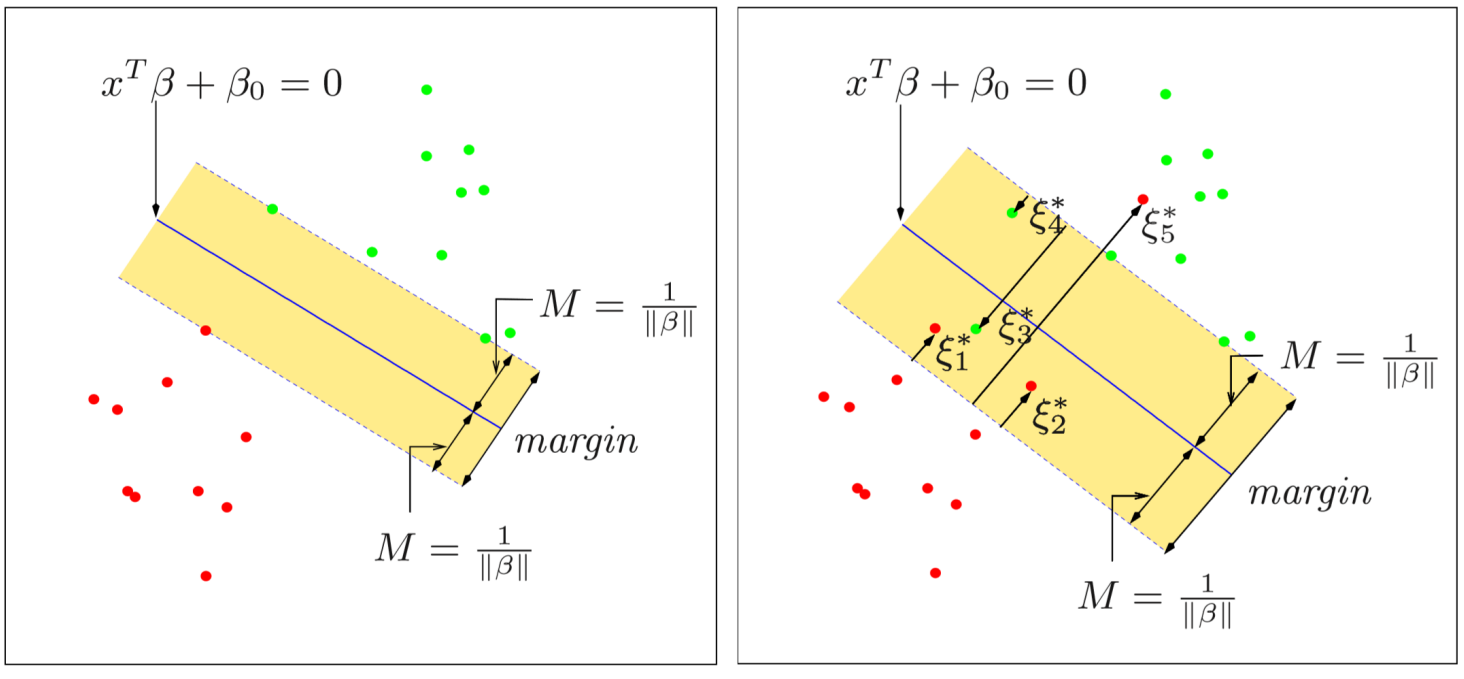


The **support vector machine** algorithmic paradigm (**SVM**) tackles the sample complexity challenge by searching for **large margin separators**. The SVM problem comes in two flavors:

Hard SVM: Training set is linearly separable by a hyperplane (ie, is realizable).

Soft SVM: Training set is not linearly separable.

Support Vector Machines



The **margin** M is the minimum distance from the hyperplane to the points int he training set. It serves two purposes: first, a large margin ensures that even if the hyperplane is perturbed slightly the fit will still hold. Second, the margin speeds training by providing a specific hyperplane target, as opposed to letting any separating hyperplane be a ERM minimizing solution.

Hard SVM

Hard SVM returns the hyperplane that separates the training sets with the largest possible margin M in the realizable case.

Given a hyperplane $x^T\beta + \beta_0 = 0$ with $||\beta|| = 1$, a binary classification rule can be given by

$$G(x) = \text{sign}\left[x^T\beta + \beta_0\right] .$$

The minimum distance between a point x_* and a hyperplane $x^T\beta + \beta_0 = 0$, is $|x_*\beta + \beta_0|$, so the Hard SVM problem is to maximize M, subject to

$$y_i(x_i^T\beta + \beta_0) \geq M\,, \;\; i = 1, \ldots, N\;, \; ||\beta|| = 1\,.$$

Note that by a rescaling $M = 1/||\beta||$ we could equivalently minimize $||\beta||$, subject tom

$$y_i(x_i^T\beta + \beta_0) \geq 1\,, \;\; i = 1, \ldots, N\,.$$

Sample Complexity Hard SVM

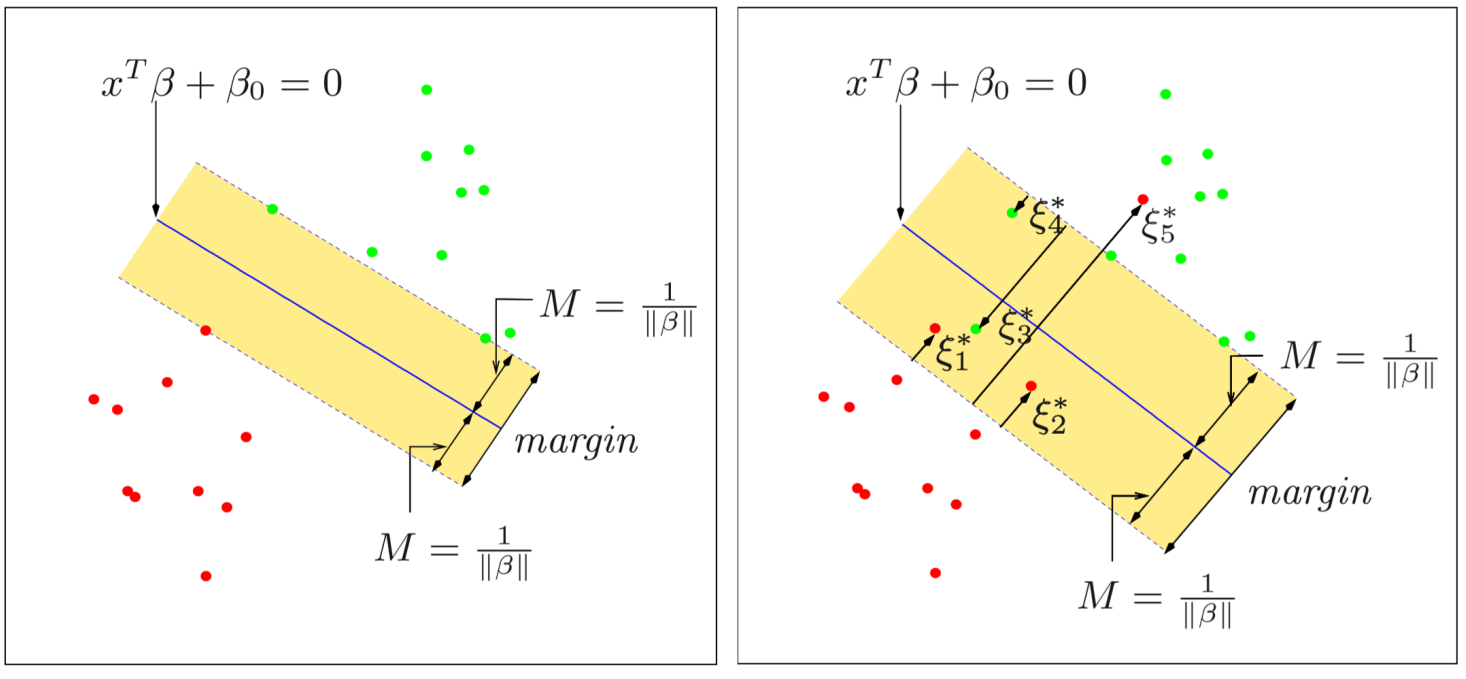
Since the VC dimension of the halfspace classifier was $d + 1$, the fundamental theorem of learning tells us that if the number of training samples is much less than d/ϵ then no algorithm can learn an ϵ accurate halfspace.

For Hard SVM, suppose a randomly chosen training set $\mathcal{T}$ is separable with a margin M and $x \sim \mathcal{D}$ is bounded $||x|| < \rho$ with probability 1. Then for $\delta > 0$, with probability $1 - \delta$ over a choice of $\mathcal{T} \sim \mathcal{D}$, the 0-1 error of the output of Hard SVM is

$$\sqrt{\frac{4\rho^2/M^2}{|\mathcal{T}|}} + \sqrt{\frac{2\log(2\delta)}{|\mathcal{T}|}}.$$

Importantly, this does not depend on the dimensionality of the feature space d. The result above is proved using Rademacher complexity.

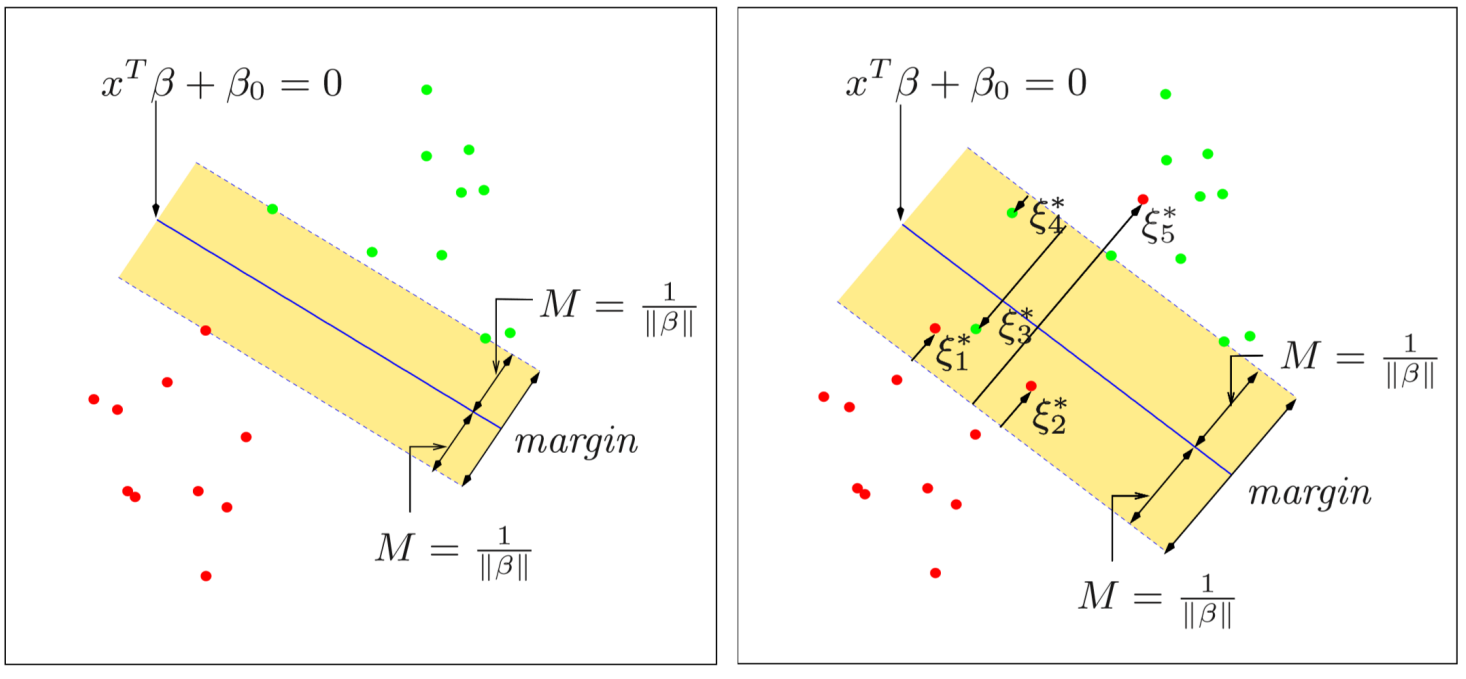
Soft SVM



Suppose now the classes overlap in feature space. We can deal with the overlap by maximizing M, but adjusting M for x_i. Introducing the **slack variables** $\xi_1, \ldots, \xi_N$, we can modify the SVM problem to either

$$y_i(x_i^T\beta + \beta_0) \geq M - \xi_i\,, \quad \text{or} \quad y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i)\,.$$

Soft SVM



$$y_i(x_i^T\beta + \beta_0) \geq M - \xi_i \quad \text{, or} \quad y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i)\,.$$

The two choices lead to different solutions, but the first equation leads to a non-convex optimization problem. The stand SVM classifier uses the second solution for ease of computation.

Sample Complexity for Soft SVM

To formalize the soft SVM problem, we write $M = 1/||\beta||$ and find $\min||\beta||$, subject to

$$y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \,, \forall i \,, \quad \xi_i > 0, \forall i \,, \quad \sum_i \xi_i \leq C \,.$$

By the criteria above, we see that points far inside the class boundary don't effect the shaping of the boundary as much (unlike in LDA). We can rewrite the problem as a loss minimization problem using the hinge loss

$$\ell_\beta^{hinge}(x, y) = \max\{0, 1 - y_i(x_i^T\beta + \beta_0)\}$$

as

$$\min_{\beta,\beta_0}\left(\lambda||\beta||^2 + \text{mean}_{\mathcal{T}}\left(\ell_\beta^{hinge}(x, y)\right)\right).$$

The hinge loss is $||x||$-Lipschitz in β, leading directly to a d-independent form of the sample complexity.

Dual Problems

Another way to solve the SVM problem is via the Lagrangian dual. Minimizing $||\beta||$ with respect to

$$y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \,, \forall i \,, \quad \xi_i > 0, \forall i \,, \quad \sum_i \xi_i \leq C \,.$$

is equivalent to minimizing

$$L_P = \frac{1}{2}||\beta|| + C\sum_{i=1}^{N} \xi - \alpha_i \left[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)\right] - \mu_i\epsilon_i \,,$$

for $P = (C, \alpha_i, \mu_i)$ constants. Setting the derivatives in β, β_0 and ξ_i to 0, gives

$$\beta = \sum_{i=1}^{N} \alpha y_i x_i \,, \quad 0 = \sum_{i=1}^{N} \alpha_i y_i \,, \quad \alpha_i = C - \mu_i \,, \forall i \,,$$

as well as $\alpha_i, \mu_i, \xi_i \geq 0$.

Dual Problems

Substituting

$$\beta = \sum_{i=1}^{N} \alpha y_i x_i \,, \quad 0 = \sum_{i=1}^{N} \alpha_i y_i \,, \quad \alpha_i = C - \mu_i \,, \forall i \,,$$

into

$$L_P = \frac{1}{2}||\beta|| + C \sum_{i=1}^{N} \xi - \alpha_i \left[y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \right] - \mu_i \epsilon_i \,,$$

yields

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \,.$$

Imposing the Karush-Kuhn-Tucker constraints leads to a unique solution to what is now an explicit quadratic programming problem.

Dual Problems

Imposing the Karush-Kuhn-Tucker constraints

$$\begin{aligned} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] &= 0 \\ \mu_i\xi_i &= 0 \\ y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) &\geq 0 \end{aligned}$$

leads to a unique solution to

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j x_i^T x_j \, .$$

as an explicit quadratic programming problem.

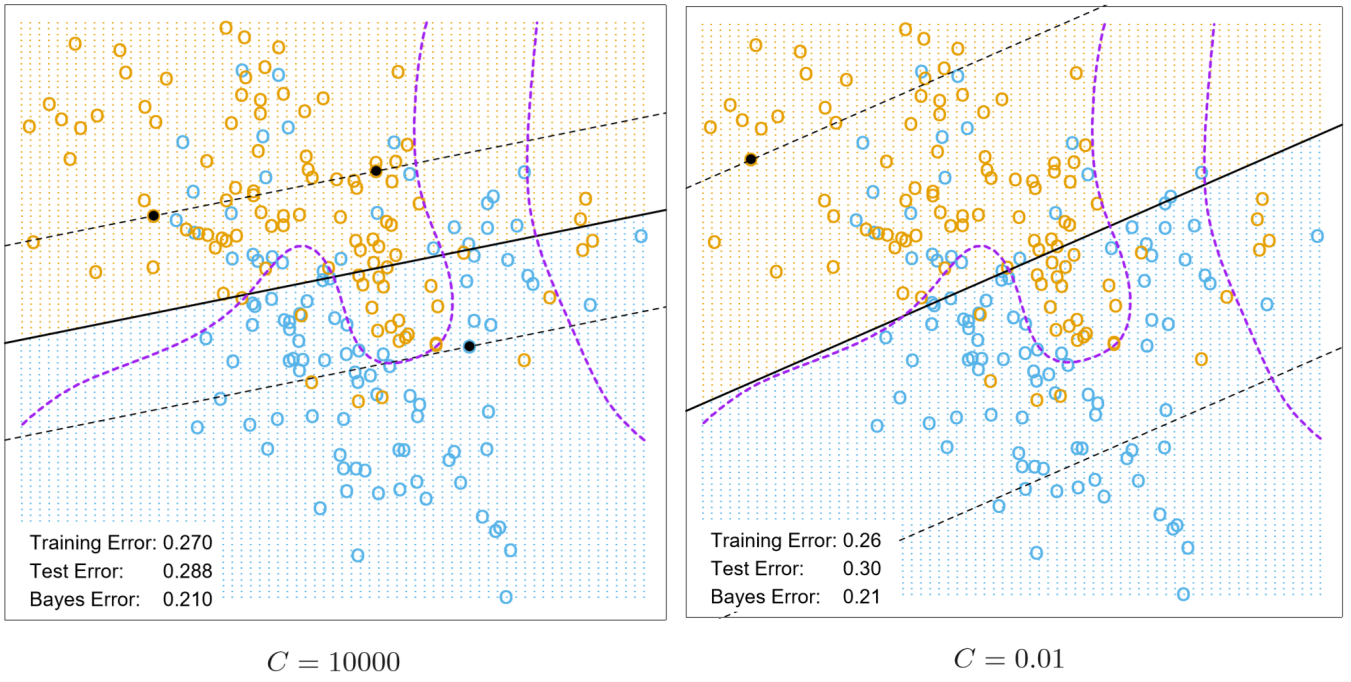
Dual Problems

It is from the condition $\beta = \sum_{i=1}^{N} \alpha y_i x_i$ that we get the name **support vector machine**. The Karush-Kuhn-Tucker constraints include

$$\alpha_i \left[y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \right] = 0 \,,$$

so $\hat{\alpha}_i$ is only nonzero when $y_i(x_i^T \beta + \beta_0) = (1 - \xi_i)$. This means the sum $\beta = \sum_{i=1}^{N} \alpha y_i x_i$ is written only in terms of certain **support vectors.**Some of these vectors will lie on the edge of the margin $\hat{\xi}_i = 0$ and will have $0 < \hat{\alpha}_i < C$, the remainder will have $\hat{\alpha}_i = C$

Example: Two Label Mixture



As an example, we see the fit of two SVM's to mixed data. The support points are all the points on the wrong side of the margin. The black dots are the support points falling exactly on the margin. On the left, 62% of the points are support points, on the right 85% are.

SVM's and the Kernel Trick

Polynomial Dimensionality

We saw before that fitting decision boundaries with higher dimensional polynomials was equivalent to constructing features $X_i X_j$, $X_i X_j X_k$, etc. for all monomials up to the desired degree. We also saw that this almost immediately became both computationally prohibitive, and the sample complexity dwarfs the size of our training set.

However, the SVM may have solved the sample complexity problem for us, since its complexity does not depend on the number of features. The kernel trick will help to make the the SVM on monomial feature space computationally efficient.

Polynomial Dimensionality

Assume that we construct a nonlinear classifier by fitting $\mathcal{X}$ into an expanded feature space $\bar{\mathcal{X}}$ via $h : \mathcal{X} \to \bar{\mathcal{X}}$. For example, in two variables h may be

$$h(x_1, x_2) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)\,.$$

A binary linear classifier then fits to $\hat{f}(x) = h(x)^T \hat{\beta}$ in $\mathbb{R}^6$ with $\hat{G}(x) = \text{sign}(\hat{f}(x))$.

For a Soft SVM, the solution to

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle h(x_i), h(x_j) \rangle$$

can be written

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle h(x_i), h(x_j) \rangle + \beta_0\,.$$

Kernels

The trick is to notice that

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \langle h(x_i), h(x_j) \rangle + \beta_0 \, .$$

only depends on h through its inner product. In fact, we need not even specify h, we only need to know its **kernel**

$$K(x, x') = \langle h(x), h(x') \rangle \, .$$

For example, for degree 2 polynomials

$$K(X, X') = (1 + \langle X, X' \rangle)^2$$
$$K(X, X') = (1 + X_1 X_1' + X_2 X_2')^2$$
$$K(X, X') = 1 + 2X_1 X_1' + 2X_2 X_2' + (X_1 X_1')^2 + (X_2 X_2')^2 + 2(X_1 X_1' X_2 X_2')$$

Setting $h(X) = (1, \sqrt{2}X_1, \sqrt{2}X_2, X_1^2, X_2^2, \sqrt{2}X_1 X_2)$,
$K(X, X') = \langle h(X), h(X') \rangle$.

Kernels

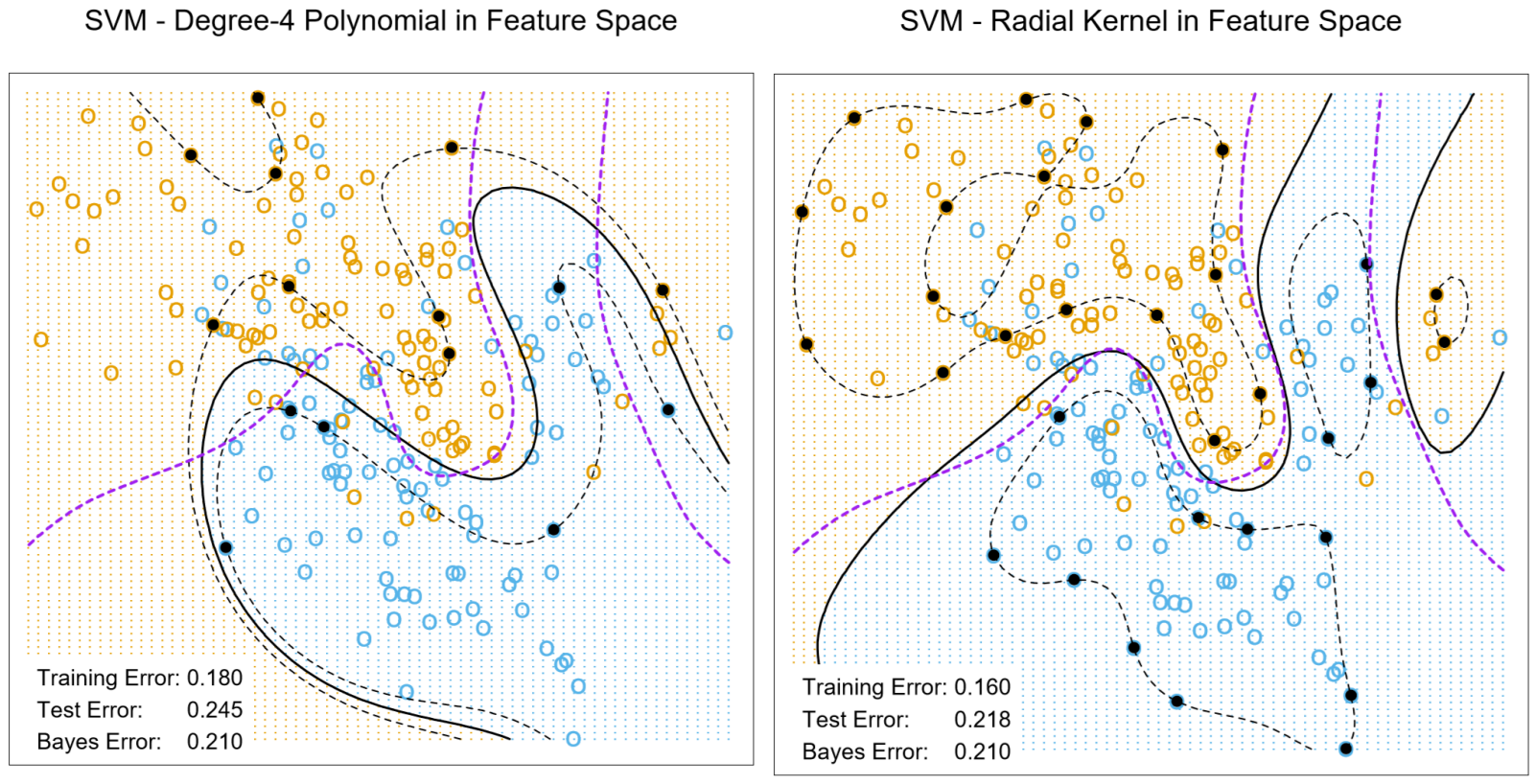
This rule hold generally. Three popular choices for transformations with kernels are

Degree d Polynomials	$K(X, X') = (1 + \langle X, X' \rangle)^d$
Radial Basis	$K(X, X') = \exp\left(-\gamma \|\|X - X'\|\|^2\right)$
Neural Networks	$K(X, X') = \tanh(w\langle X, X' \rangle + b)$

Radial Basis Functions act like activation functions in NN's, except that they are Gaussian in the radial distantance from a point ξ_i:

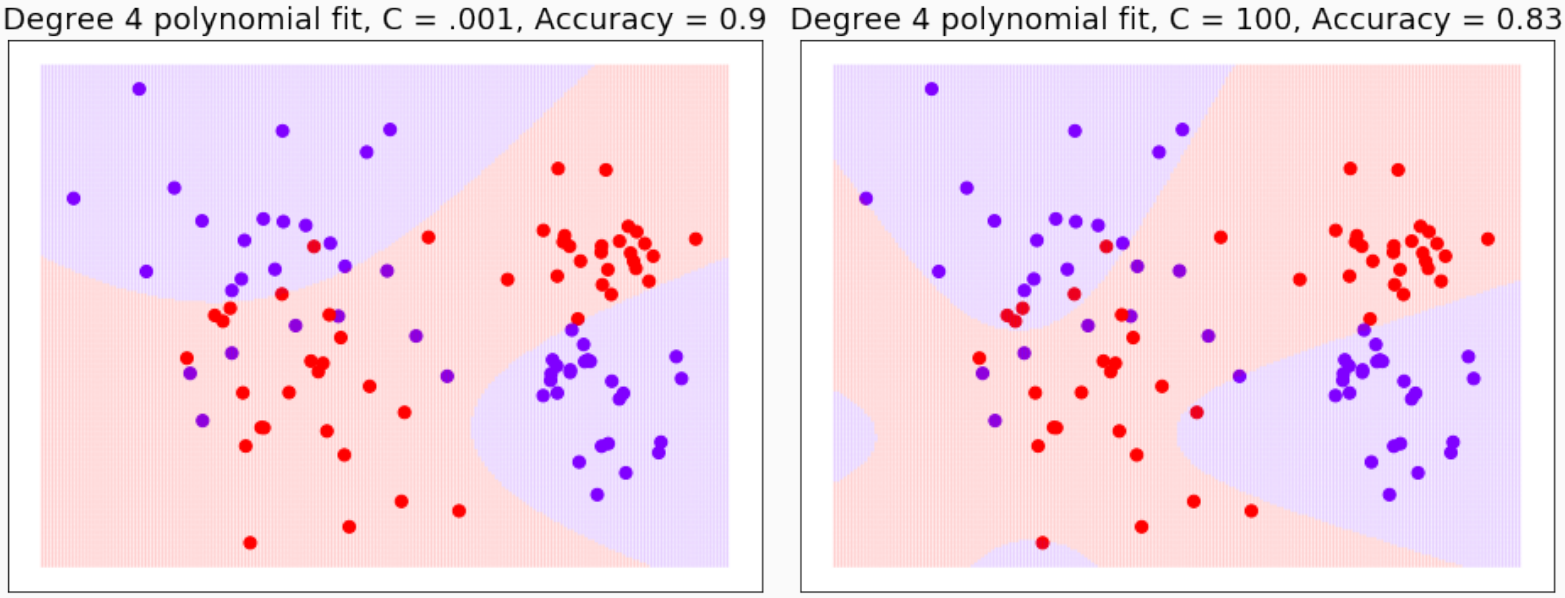
$$h_i(x) = \exp\left(\frac{(x_i - \xi_i)^T (x_i - \xi_i)}{\lambda_i}\right).$$

Example: Two Label Mixture



Here, we compare the degree 4 polynomial in the right to the radial classifier on the left. The radial classifier produces unions of ovoids while the polynomial produces a spline-like boundary.

Example: Two Label Mixture



The role of C is clearer in the enlarged feature space since perfect separation often is achievable. Interestingly, the right plot takes consideration of more data (and as a results takes significantly longer to train) while giving a worse results.

Curse of Dimensionality

Do SVM's provide a cure for the curse of dimensionality? The answer, as always, is going to be in the negative. Although in many cases they greatly outperform other methods, their success relies on other features in the data.

For example, suppose that we expanded a polynomial with $p \gg 1$ features, but that the separation was really between X_1 and X_2. The kernel trick does not allow fully general inner products. In the polynomial expansion, all terms are treated equally so the kernel would not easily find this simpler structure.

You could change the model, but of course that requires prior knowledge.

Curse of Dimensionality

This is just the beginning of the SVM story, and a lot of modern research goes into improving feature generation and detection, and kernel construction, and regularization.

SVM give us computational access to much more complicated models as well. We could now redo all of discriminant analysis in terms of higher degree functions and SVMs, called Flexible Discriminant Analysis, or Guassian clustering Mixture Discriminant Analysis.

References

References for this lecture: BS, Chapters 15, 16, 18 and ESII, Chapters 9, 12.

For more about SVM uses and generalization, see ESII Chapter 12. For more about SVM proofs and complexity computations, see SB Chapter 26.

More hands on with random forests in R: `https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf`

Image reference: Comparison of Tree Loss functions: `https://github.com/rasbt/python-machine-learning-book`