Pruning The Tree

Growing a tree this way may stop too early: splitting a particular leaf may not need to an improvement in the sum of squared errors, but if we split anyway, we may find subsequently profitable splits.

For example, suppose that there are two binary covariates, $X_{i1}, X_{i,2} \in \{0, 1\}$, and that

$$\mathbb{E}[Y_i | X_{i,1} = 0, X_{i,2} = 0] = \mathbb{E}[Y_i | X_{i,1} = 1, X_{i,2} = 1] = -1$$

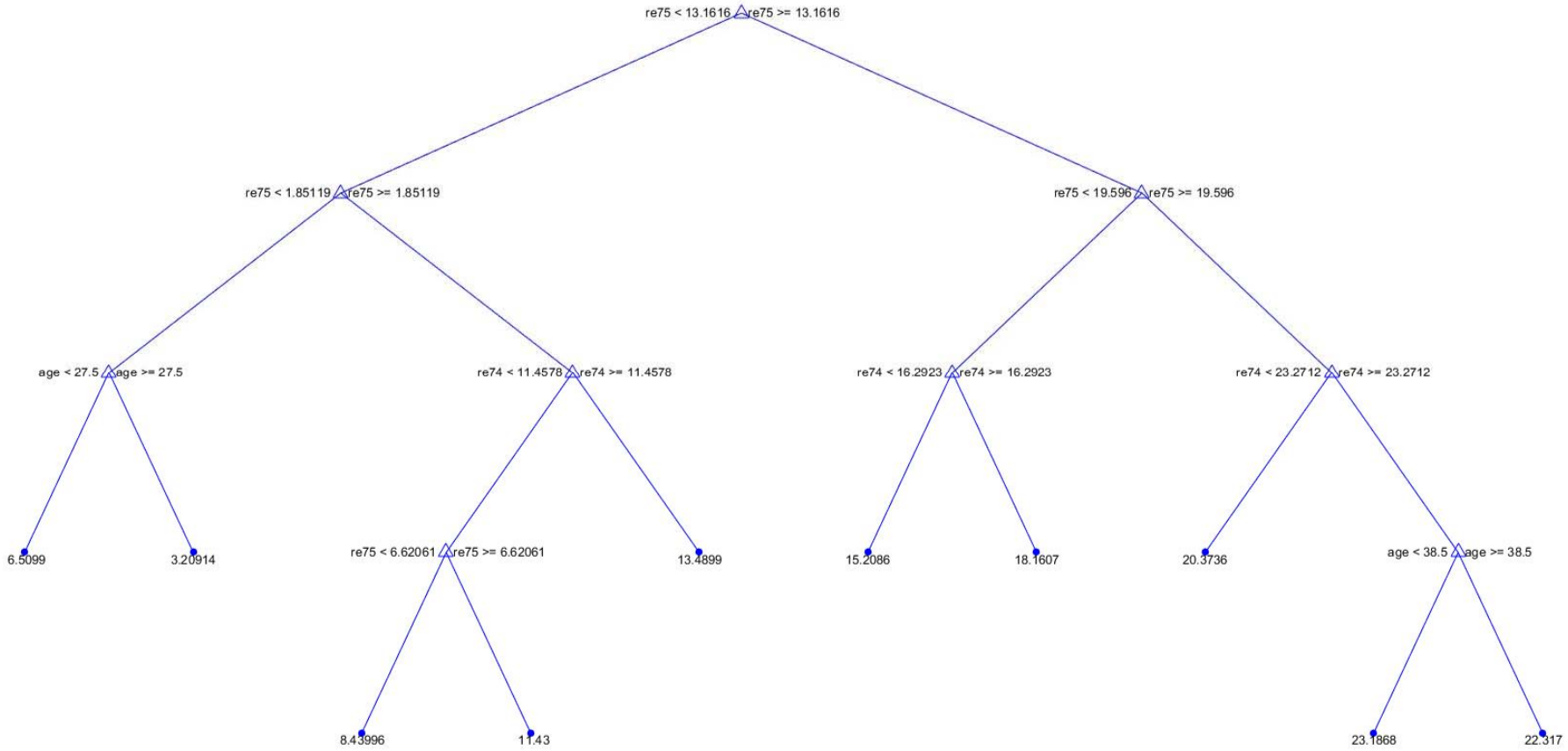
$$\mathbb{E}[Y_i | X_{i,1} = 0, X_{i,2} = 1] = \mathbb{E}[Y_i | X_{i,1} = 0, X_{i,2} = 1] = 1$$

Then splitting on X_{i1} or X_{i2} does not improve the objective function, but once one splits on either of them, the subsequent splits lead to an improvement.

This motivates **pruning** the tree:

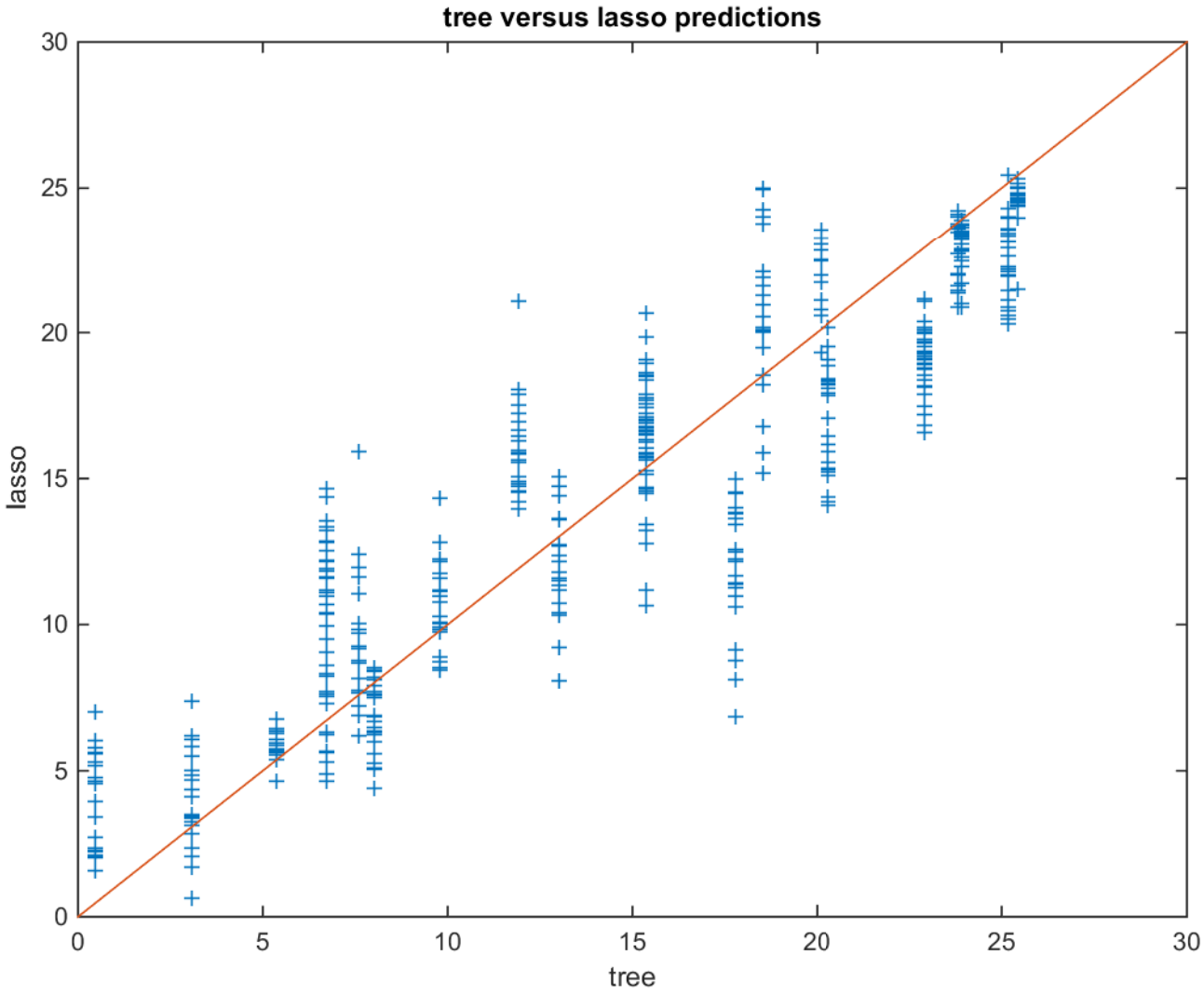
• First grow a big tree by using a deliberately small value of the penalty term, or simply growing the tree till the leaves have a preset small number of observations.

• Then go back and prune branches or leaves that do not collectively improve the objective function sufficiently.


re75 < 13.1616
re75 >= 13.1616
re75 < 1.85119
re75 >= 1.85119
re75 < 19.596
re75 >= 19.596
age < 27.5
age >= 27.5
re74 < 11.4578
re74 >= 11.4578
re74 < 16.2923
re74 >= 16.2923
re74 < 23.2712
re74 >= 23.2712
6.5099
3.20914
re75 < 6.62061
re75 >= 6.62061
13.4899
15.2086
18.1607
20.3736
age < 38.5
age >= 38.5
8.43996
11.43
23.1868
22.317

Tree leads to 37 splits

Test sample root-mean-squared error
OLS: 1.4093
LASSO: 0.7379
Tree: 0.7865


tree versus lasso predictions
lasso
tree
0
5
10
15
20
25
30

2.c Boosting

Suppose we have a simple, possibly naive, but easy to compute, way of estimating a regression function, a so-called **weak learner**.

Boosting is a general approach to repeatedly use the weak learner to get a good predictor for both classification and regression problems. It can be used with many different weak learners, trees, kernels, support vector machines, neural networks, etc.

Here I illustrate it using regression trees.

Suppose $g(x|\mathbf{X}, \mathbf{Y})$ is based on a very simple regression tree, using only a single split. So, the algorithm selects a covariate $k(\mathbf{X}, \mathbf{Y})$ and a threshold $t(\mathbf{X}, \mathbf{Y})$ and then estimates the regression function as

$$g_1(x|\mathbf{X}, \mathbf{Y}) = \begin{cases} \overline{Y}_{\text{left}} & \text{if } x_{k(\mathbf{X},\mathbf{Y})} \leq t(\mathbf{X}, \mathbf{Y}) \\ \overline{Y}_{\text{right}} & \text{if } x_{k(\mathbf{X},\mathbf{Y})} > t(\mathbf{X}, \mathbf{Y}) \end{cases}$$

where

$$\overline{Y}_{\text{left}} = \frac{\sum_{i:X_{i,k} \leq t} Y_i}{\sum_{i:X_{i,k} \leq t} 1} \quad \overline{Y}_{\text{right}} = \frac{\sum_{i:X_{i,k} > t} Y_i}{\sum_{i:X_{i,k} > t} 1}$$

Not a very good predictor by itself.

Define the residual relative to this weak learner:

$$\varepsilon_{1i} = Y_i - g_1(X_i|\mathbf{X}, \mathbf{Y})$$

Now apply the **same** weak learner to the **new** data set $(\mathbf{X}, \varepsilon_1)$.

Grow a second tree $g_2(\mathbf{X}, \varepsilon_1)$ based on this data set (with single split), and define the new residuals as

$$\varepsilon_{2i} = Y_i - g_1(X_i|\mathbf{X}, \mathbf{Y}) - g_2(X_i|\mathbf{X}, \varepsilon_1)$$

Re-apply the weak learner to the data set $(\mathbf{X}, \varepsilon_2)$.

After doing this many times you get an additive approximation to the regression function:

$$\sum_{m=1}^{M} g_m(x|\mathbf{X}, \varepsilon_{m-1}) = \sum_{k=1}^{K} h_k(x_k) \quad \text{where} \quad \varepsilon_0 = \mathbf{Y}$$

Had we used a weak learner with two splits, we would have allowed for second order effects $h(x_k, x_l)$.

In practice researchers use **shallow** trees, say with six splits (implicitly allowing for 6-th order interactions), and grow many trees, e.g., 400-500.

Often the depth of the initial trees is fixed in advance in an ad hoc manner (difficult to choose too many tuning parameters optimally), and the number of trees is based on prediction errors in a test sample (similar in spirit to cross-validation).

2.d Bagging (Bootstrap AGGregatING) Applicable to many ways of estimating regression function, here applied to trees.

1. Draw a bootstrap sample of size N from the data.

2. Construct a tree $g_b(x)$, possibly with pruning, possibly with data-dependent penalty term.

Estimate the regression function by averaging over bootstrap estimates:

$$\frac{1}{B}\sum_{b=1}^{B} g_b(x)$$

If the basic learner were linear, than the bagging is ineffective. For nonlinear learners, however, this smoothes things and can lead to improvements.

2.e Random Forests (Great general purpose method)

Given data $(\mathbf{X}, \mathbf{Y})$, with the dimension of $\mathbf{X}$ equal to $N \times K$, do the same as bagging, but with a different way of constructing the tree given the bootstrap sample: Start with a tree with a single leaf.

1. Randomly select L regressors out of the set of K regressors
2. Select the optimal cov and threshold among L regressors
3. If some leaves have more than N_{min} units, go back to (1)
4. Otherwise, stop

Average trees over bootstrap samples.

• For bagging and random forest there is recent research suggesting asymptotic normality may hold. It would require using small training samples relative to the overall sample (on the order of $N/(\ln(N)^K)$, where K is the number of features.

See Wager, Efron, and Hastie (2014) and Wager (2015).

2.f Neural Networks / Deep Learning

Goes back to 1990's, work by Hal White. Recent resurgence.

Model the relation between X_i and Y_i through hidden layer(s) of Z_i, with M elements $Z_{i,m}$:

$$Z_{i,m} = \sigma(\alpha_{0m} + \alpha'_{1m} X_i), \quad \text{for } m = 1, \ldots, M$$

$$Y_i = \beta_0 + \beta'_1 Z_i + \varepsilon_i$$

So, the Y_i are linear in a number of transformations of the original covariates X_i. Often the transformations are sigmoid functions $\sigma(a) = (1 + \exp(-a))^{-1}$. We fit the parameters α_m and β by minimizing

$$\sum_{i=1}^{N} (Y_i - g(X_i, \alpha, \beta))^2$$

• Estimation can be hard. Start with α random but close to zero, so close to linear model, using gradient descent methods.

• Difficulty is to avoid overfitting. We can add a penalty term

$$\sum_{i=1}^{N} (Y_i - g(X_i, \alpha, \beta))^2 + \lambda \cdot \left(\sum_{k,m} \alpha_{k,m}^2 + \sum_{k} \beta_k^2 \right)$$

Find optimal penalty term λ by monitoring sum of squared prediction errors on test sample.

2.g Ensemble Methods, Model Averaging, and Super Learners

Suppose we have M candidate estimators $g_m(\cdot|\mathbf{X}, \mathbf{Y})$. They can be similar, e.g., all trees, or they can be qualitatively different, some trees, some regression models, some support vector machines, some neural networks. We can try to combine them to get a better estimator, often better than any single algorithm.

Note that we do not attempt to select a single method, rather we look for weights that may be non-zero for multiple methods.

Most competitions for supervised learning methods have been won by algorithms that combine more basic methods, often many different methods.

One question is how exactly to combine methods.

One approach is to construct weights $\alpha_1, \ldots, \alpha_M$ by solving, using a test sample

$$\min_{\alpha_1,\ldots,\alpha_M} \sum_{i=1}^{N} \left(Y_i - \sum_{m=1}^{M} \alpha_m \cdot g_m(X_i) \right)^2$$

If we have many algorithms to choose from we may wish to regularize this problem by adding a LASSO-type penalty term:

$$\lambda \cdot \sum_{m=1}^{M} |\alpha_m|$$

That is, we restrict the ensemble estimator to be a weighted average of the original estimators where we shrink the weights using an L_1 norm. The result will be a weighted average that puts non-zero weights on only a few models.

Test sample root-mean-squared error
OLS: 1.4093
LASSO: 0.7379
Tree: 0.7865
Ensemble: 0.7375

Weights ensemble:
OLS 0.0130
LASSO 0.7249
Tree 0.2621

3. Unsupervised Learning: Clustering

We have N observations on a M-component vector of features, X_i, $i = 1, \ldots, N$. We want to find patterns in these data.

Note: there is no outcome Y_i here, which gave rise to the term "unsupervised learning."

• One approach is to reduce the dimension of the X_i using principal components. We can then fit models using those principal components rather than the full set of features.

• Second approach is to partition the space into a finite set. We can then fit models to the subpopulations in each of those sets.

3.a Principal Components

• Old method, e.g., in Theil (Principles of Econometrics)

We want to find a set of K N-vectors $Y_1, \ldots, Y_K$ so that

$$X_i \approx \sum_{k=1}^{K} \gamma_{ik} Y_k$$

or, collectively, we want to find approximation

$$\mathbf{X} = \Gamma \mathbf{Y}, \qquad \Gamma \text{ is } M \times K, \ M > K$$

This is useful in cases where we have many features and we want to reduce the dimension without giving up a lot of information.

First normalize components of X_i, so average $\sum_{i=1}^{N} X_i/N = 0$, and components have unit variance.

First principal component: Find N-vector $\mathbf{Y}_1$ and the M vector Γ that solve

$$\mathbf{Y}_1, \Gamma = \arg \min_{\mathbf{Y}_1, \Gamma} \text{trace} \left((\mathbf{X} - \mathbf{Y}_1 \Gamma)'(\mathbf{X} - \mathbf{Y}_1 \Gamma) \right)$$

This leads to $\mathbf{Y}_1$ being the eigenvector of the $N \times N$ matrix $\mathbf{X}\mathbf{X}'$ corresponding to the largest eigenvalue. Given $\mathbf{Y}$, Γ is easy to find.

Subsequent $\mathbf{Y}_k$ correspond to the subsequent eigenvectors.

3.b Mixture Models and the EM Algorithm

Model the joint distribution of the L-component vector X_i as a mixture of parametric distributions:

$$f(x) = \sum_{k=1}^{K} \pi_k \cdot f_k(x; \theta_k) \qquad f_k(\cdot;\cdot) \text{ known}$$

We want to estimate the parameters of the mixture components, θ_k, and the mixture probabilities π_k.

The mixture components can be multivariate normal, of any other parametric distribution. Straight maximum likelihood estimation is very difficult because the likelihood function is multi-modal.

The EM algorithm (Dempster, Laird, Rubin, 1977) makes this easy as long as it is easy to estimate the θ_k given data from the k-th mixture component. Start with $\pi_k = 1/K$ for all k.

Create starting values for θ_k, all different.

Update weights, the conditional probability of belonging to cluster k given parameter values (E-step in EM):

$$w_{ik} = \frac{\pi_k \cdot f_k(X_i; \theta_k)}{\sum_{m=1}^{K} \pi_m \cdot f_m(X_i; \theta_m)}$$

Update θ_k (M-step in EM)

$$\theta_k = \arg\max_{\theta} \sum_{i=1}^{N} w_{ik} \cdot \ln f_k(X_i; \theta)$$

Algorithm can be slow but is very reliable. It gives probabilities for each cluster. Then we can use those to assign new units to clusters, using the highest probability.

Used in duration models in Gamma-Weibull mixtures (Lancaster, 1979), non-parametric Weibull mixtures (Heckman & Singer, 1984).

3.c The k-means Algorithm

1. Start with k arbitrary centroids c_m for the k clusters.

2. Assign each observation to the nearest centroid:

$$I_i = m \quad \text{if} \quad \|X_i - c_m\| = \min_{m'=1} \|X_i - c_{m'}\|$$

3. Re-calculate the centroids as

$$c_m = \sum_{i:I_i=m} X_i \Big/ \sum_{i:I_i=m} 1$$

4. Go back to (2) if centroids have changed, otherwise stop.

- k-means is fast

- Results can be sensitive to starting values for centroids.

3.d. Mining for Association Rules: The Apriori Algorithm

Suppose we have a set of N customers, each buying arbitrary subsets of a set $\mathcal{F}_0$ containing M items. We want to find subsets of k items that are bought together by at least L customers, for different values of k.

This is very much a data mining exercise. There is no model, simply a search for items that go together. Of course this may suggest causal relationships, and suggest that discounting some items may increase sales of other items.

It is potentially difficult, because there are M choose k subsets of k items that could be elements of $\mathcal{F}_k$. The solution is to do this sequentially.

You start with $k = 1$, by selecting all items that are bought by at least L customers. This gives a set $\mathcal{F}_1 \subset \mathcal{F}_0$ of the M original items.

Now, for $k \geq 2$, given $\mathcal{F}_{k-1}$, find $\mathcal{F}_k$.

First construct the set of possible elements of $\mathcal{F}_k$. For a set of items F to be in $\mathcal{F}_k$, it must be that any set obtained by dropping one of the k items in F, say the m-th item, leading to the set $F_{(m)} = F/\{m\}$, must be an element of $\mathcal{F}_{k-1}$.

The reason is that for F to be in $\mathcal{F}_k$, it must be that there are at least L customers buying that set of items. Hence there must be at least L customers buying the set $F_{(m)}$, and so $F_{(m)}$ is an $k-1$ item set that must be an element of $\mathcal{F}_{k-1}$.

5. Support Vector Machines and Classification

Suppose we have a sample (Y_i, X_i), $i = 1, \ldots, N$, with $Y_i \in \{-1, 1\}$.

We are trying to come up with a classification rule that assigns units to one of the two classes -1 or 1.

One conventional econometric approach is to estimate a logistic regression model and assign units to the groups based on the estimated probability.

Support vector machines look for a boundary $h(x)$, such that units on one side of the boundary are assigned to one group, and units on the other side are assigned to the other group.

Suppose we limit ourselves to linear rules,

$$h(x) = \beta_0 + x\beta_1,$$

where we assign a unit with features x to class 1 if $h(x) \geq 0$ and to -1 otherwise.

We want to choose β_0 and β_1 to optimize the classification. The question is how to quantify the quality of the classification.

Suppose there is a hyperplane that completely separates the $Y_i = -1$ and $Y_i = 1$ groups. In that case we can look for the β, such that $\|\beta\| = 1$, that maximize the margin M:

$$\max_{\beta_0,\beta_1} M \quad \text{s.t.} \quad Y_i \cdot (\beta_0 + X_i\beta_1) \geq M \ \forall i, \quad \|\beta\| = 1$$

The restriction implies that each point is at least a distance M away from the boundary.

We can rewrite this as

$$\min_{\beta_0,\beta_1} \|\beta\| \quad \text{s.t.} \quad Y_i \cdot (\beta_0 + X_i\beta_1) \geq 1 \ \forall i.$$

Often there is no such hyperplane. In that case we define a penalty we pay for observations that are not at least a distance M away from the boundary.

$$\min_{\beta_0,\beta_1} \|\beta\| \quad \text{s.t.} \quad Y_i \cdot (\beta_0 + X_i\beta_1) \geq 1 - \varepsilon_i,\ \varepsilon_i \geq 0\ \forall i,\ \sum_i \varepsilon \leq C.$$

Alternatively

$$\min_{\beta} \frac{1}{2} \cdot \|\beta\| + C \cdot \sum_{i=1} \varepsilon_i$$

subject to

$$\varepsilon_i \geq 0, \quad Y_i \cdot (\beta_0 + X_i\beta_1) \geq 1 - \varepsilon_i$$

With the linear specification $h(x) = \beta_0 + x\beta_1$ the support vector machine leads to

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{N} \lfloor 1 - Y_i \cdot (\beta_0 + X_i\beta_1) \rfloor_+ + \lambda \cdot \|\beta\|^2$$

where $\lfloor a \rfloor_+$ is a if $a > 0$ and zero otherwise, and $\lambda = 1/C$.

Note that we do not get probabilities here as in a logistic regression model, only assignments to one of the two groups.

Thank You!