Classification: trees, Neural Nets and Support Vector Machines

American Economic Association,

Continuing Education Program 2018

Machine Learning and Econometrics, Lecture 8

Tuesday January 10th, 9.30-10.45am

Guido Imbens - Stanford University

Outline

1. Classification

2. Relation to Discrete Choice Models

3. Classification Trees

4. Neural Networks for Classification

5. Support Vector Machines

1. Classification

A second canonical problem in supervised learning is **classification**.

The setting is again one where we observe for a large number of units, say N, a p-vector of covariates x_i, and an outcome y_i. The outcome is now a **class**, an unordered discrete variable, $y_i \in \{1, \ldots, M\}$. M may be 2, or fairly large.

The goal is a function $f : \mathbb{X} \mapsto \{1, \ldots, M\}$ that will allow us to assign new units to one of the classes.

• This is like discrete choice problems, but often we are not interested in estimating the probability of $y_i = m$ given $x_i = x$, just the classification itself.

The objective is typically to minimize for new cases (out-of-sample) the miss-classification rate.

$$\frac{1}{N_{\text{test}}} \sum_{i=N_{\text{train}}+1}^{N_{\text{test}}} \mathbf{1}_{y_i \neq f(x_i)}$$

2. Relation to Discrete Choice Models

Consider the two-class (binary y_i) case.

Suppose that the sample is a random sample from an infinite population, where:

$$\text{pr}(y_i = 1 | x_i = x) = \frac{\exp(x'\beta)}{1 + \exp(x'\beta)}$$

Then the optimal classification is

$$\hat{f}(x) = \mathbf{1}\left\{\frac{\exp(x'\beta)}{1 + \exp(x'\beta)} \geq 0.5\right\}$$

or equivalently

$$\hat{f}(x) = \mathbf{1}\left\{x'\beta \geq 0\right\}$$

The standard thing to do in discrete choice analysis would be to estimate β as

$$\widehat{\beta} = \arg\max L(\beta)$$

where

$$L(\beta) = \sum_{i=1}^{n} y_i \exp(x_i'\beta) - \ln\left(1 + \exp(x'\beta)\right)$$

and then classify as

$$\widehat{f}(x) = \mathbf{1}\left\{x'\widehat{\beta} \geq 0\right\}$$

Outliers can be very influential here: for example, if we have a scalar covariate, and $\beta > 0$, a single observation ($x =$ very large, $y = 0$) can change $\widehat{\beta}$ to negative.

The machine learning methods take a different approach. These differences include

1. the objective function,
2. complexity of the models,
3. regularization

We will look at each of these.

β is estimated by focusing on (in- and out-of-sample) classification accuracy. For in-sample classification accuracy, we could consider a maximum score type objective function:

$$\widehat{\beta} = \arg\max \sum_{i=1}^{N} \left| y_i - \mathbf{1}\left\{ x'\beta \geq 0 \right\} \right|$$

This of course is not necessarily an improvement over doing maximum likelihood. Given the true value for β, using $x'\beta > 0$ as a classifier is optimal, and the maximum likelihood estimator for $\beta$is better than the estimator based on classification error.

• The classification rate does **not** give the optimal weight to different classification errors if we know the model is logistic. But, it is **robust** because it does not rely on a model.

With many covariates we can also regularize this estimator (or the maximum likelihood estimator) using an L_1 penalty term:

$$\widehat{\beta} = \arg\max \sum_{i=1}^{N} \left| y_i - \mathbf{1}\left\{ x'\beta \geq 0 \right\} \right| + \lambda \sum_{j=1}^{p} |\beta_p|$$

For logistic regression this goes back to Tibshirani's original LASSO paper:

$$\widehat{\beta} = \arg\max_{\beta} \sum_{i=1}^{n} \left\{ y_i \exp(x_i'\beta) - \ln\left(1 + \exp(x'\beta)\right) \right\} + \lambda \sum_{j=1}^{p} |\beta_j|$$

We could also use other penalties, e.g., ridge, or elastic net.

Many differences in complexity and type of models. Methods and models are very close to regression versions. Slight differences in implementation and tuning parameters.

Here, we consider

1. trees

2. neural networks

3. support vector machines

3. Classification Trees

The basic algorithm has the same structure as in the regression case.

We start with a single leaf. We then decide on a covariate/feature $k = 1, \ldots, p$, to split on, and a threshold c, so that we split the single leaf into two leaves, depending on whether

$$x_{ik} \leq c$$

or not.

The key question is how to compare the different potential splits. In regression problems we use the sum of squared residuals within each potential leave for this. Here we consider alternatives.

Now define an **impurity** function as a function of the shares of different classes in a leaf:

$$\phi(\mathcal{T}) = \phi(\pi_1, \ldots, \pi_M)$$

satisfying

$$\phi(\pi_1, \ldots, \pi_M) \leq \phi(1/M, 1/M, \ldots, 1/M)$$

$$\phi(\pi_1, \ldots, \pi_M) \geq \phi(1, 0, \ldots, 0) = \phi(0, 1, \ldots, 0) = \phi(0, 0, \ldots, 1)$$

Now consider a tree $\mathcal{T}$ with leaves $t = 1, \ldots, T$. Let

$$\pi_{m|t} \quad \text{and } \pi_t$$

be the share of class m in leaf t and the share of leaf t respectively.

Then the impurity of tree $\mathcal{T}$ is the average of the impurities of the leaves, weighted by their shares:

$$\phi(\mathcal{T}) = \sum_{t=1}^{T} p_t \phi\left(\pi_{1|t}, \ldots, \pi_{M|t}\right)$$

This replaces the average squared residual as the objective function to grow/compare/prune trees.

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the marginal distribution of labels in the subset.

The share of units with label m is p_m. If we randomly label those, we mislabel a fraction $1 - p_m$ of them, leading to

$$\phi(p_1, \ldots, p_M) = \sum_{m=1}^{M} p_m(1 - p_m) = 1 - \sum_{m=1}^{M} p_m^2$$

Information criterion

$$\phi(p_1, \ldots, p_M) = - \sum_{m=1}^{M} p_m \ln(p_m)$$

Note that we do **not** use the within-sample classification error here as the criterion. That would amount to using

$$\phi(p_1, \ldots, p_M) = 1 - \max_m p_m$$

After the first split, we consider each of the two leaves separately. For each leaf we again determine the optimal covariate to split on, and the optimal threshold.

Then we compare the impurity of the optimal split based on splitting the first leaf, and the impurity of the optimal split based on splitting the second leaf, and choose the one that most improves the impurity.

At each step the impurity of the tree improves, until we reach the point that all the leaves are completely pure, or each leaf contains only units with the same value of x.

We keep doing this till we satisfy some stopping criterion.

Typically: stop if the impurity plus some penality (e.g., linear in the number of leaves) is minimized:

$$\phi(\mathcal{T}) + \lambda|\mathcal{T}|$$

The value of the penalty term, λ, is determined by out-of-sample cross-validation.

To evaluate trees, and to compare them out of sample, we typically do use the fraction of miss-classified units in a test sample:

$$\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}\left\{y_i \neq f(x_i)\right\}$$

without any weighting.

4. Neural Networks for Classification

Recall the output layer in the regression version, where we had a single scalar outcome y:

$$z^{(K+2)} = \omega_0^{(K+1)} + \sum_{j=1}^{p_{K+1}} \omega_j^{(K+1)} \alpha_j^{K+1},$$

$$\begin{aligned} f(x) &= g^{(K+2)}\left(z^{(K+2)}\right) = z^{(K+2)} \\ &= \omega_0^{(K+1)} + \sum_{j=1}^{p_{K+1}} \omega_j^{(K+1)} \alpha_j^{K+1}, \end{aligned}$$

Now modify this by defining the outcome to be the vector $\tilde{y}$

$$\tilde{y}_m = \mathbf{1}\{y = m\}$$

Then introduce an M-component vector $z^{(K+2)}$

$$z_m^{(K+2)} = \omega_{m0}^{(K+1)} + \sum_{j=1}^{p_{K+1}} \omega_{mj}^{(K+1)} \alpha_j^{K+1},$$

and transformation (note that this involves all elements of $z^{(K+2)}$, unlike the transformations we considered before)

$$g_m^{(K+2)}\left(z^{(K+2)}\right) = \frac{\exp\left(z_m^{(K+2)}\right)}{\sum_{l=1}^{M} \exp\left(z_l^{(K+2)}\right)}$$

Now we can use all the machinery from the deep neural networks developed for the regression case.

5. Support Vector Machines

Consider the case with two classes, $y_i \in \{-1, 1\}$.

Define a hyperplane by

$$\{x | x'\beta + \beta_0 = 0\}$$

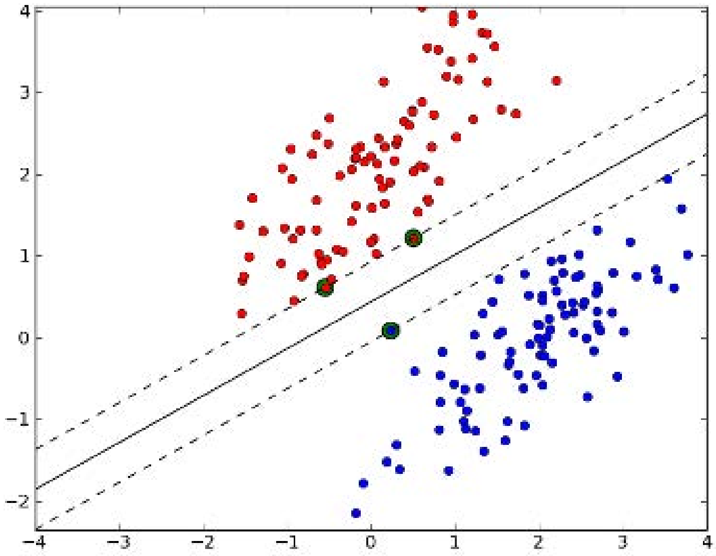
with $\|\beta\| = 1$

The hyperplane generates a classification rule:

$$f(x) = \text{sign}(\beta_0 + x'\beta)$$

Now if we are very lucky, or have lots of covariates, we may find a hyperplane that lead to a classification rule that has no in-sample classification errors, so that for all i.

$$y_i f(x_i) > 0 \quad (y_i = 1, f(x_i) > 0) \text{ or } (y_i = -1, f(x_i) < 0)$$


4
3
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2
3
4

In that case there are typically many such hyperplanes. We can look for the one that has the biggest distance between the two classes:

$$\max_{\beta_0,\beta,|\beta\|=1} M \quad \text{subject to } y_i(\beta_0 + x_i'\beta) \geq M \quad i = 1,\ldots,N$$

This is equivalent to ($\gamma = \beta/(M\|\beta\|), \gamma_0 = \beta_0/(M\|\beta\|)$),

$$\min_{\gamma_0,\gamma} \|\gamma\| \quad \text{subject to } y_i(\gamma_0 + x_i'\gamma) \geq 1 \quad i = 1,\ldots,N$$

This hyperplane depends only on a few units (the support vectors) - outside of those support vectors you can move the points a little bit without changing the value of β.

Note that in this perfect separation case, the logit model is not estimable (parameters go to infinity).

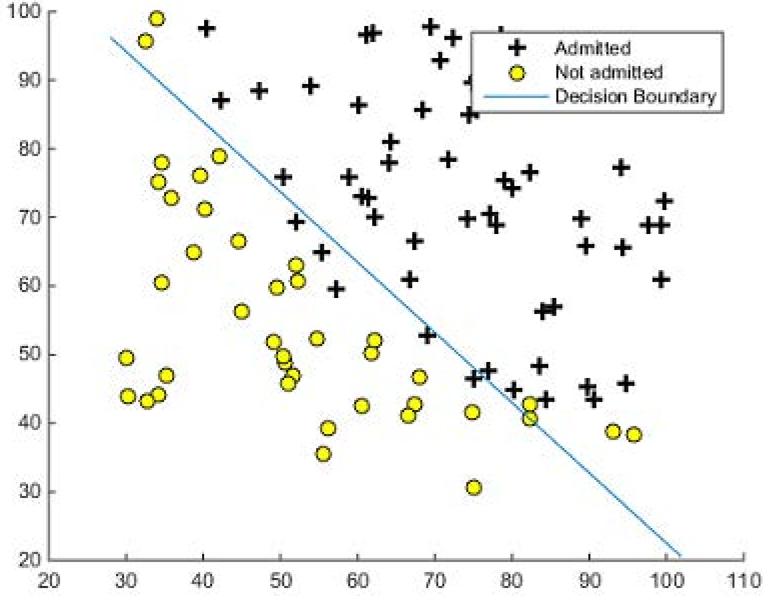
Now it is rare that we can find a hyperplane that completely separates the classes. More generally we solve

$$\min_{\beta_0,\beta,\varepsilon_1,\ldots,\varepsilon_N} \|\beta\| \quad \text{subject to } y_i(\beta_0 + x_i'\beta) \geq 1 - \epsilon_i$$

$$\text{and } \epsilon_i \geq 0,\ i = 1,\ldots,N \quad \sum_{i=1}^{N} \epsilon_i \leq K$$

The ε_i are the Euclidean distance to the margin, for those points that are within the margin from the separating hyperplane (the support vectors). In the separable case we can choose $K = 0$ and there is a solution with all $\varepsilon_i = 0$.

If we choose K is larger, we have a bigger margin, but more points inside the margin, and so a more robust estimate of β (and thus a more robust classifier). We choose K through cross-validation using out of sample classification.


Admitted
Not admitted
Decision Boundary
100
90
80
70
60
50
40
30
20
20
30
40
50
60
70
80
90
100
110

We can rewrite the problem in a way that makes it easier to link to conventional logistic regression.

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} \left[1 - y_i(\beta_0 + x_i^\top \beta)\right]_+ + \lambda\|\beta\|^2$$

where now the regularization is through the penalty term $\lambda\|\beta\|^2$.

$[a]_+$ is a if $a > 0$ and zero otherwise.

Compare this to regularized logistic regression, which solves

$$\min_{\beta_0,\beta} \sum_{i=1}^{N} \ln\left(1 + \exp\left(-y_i(\beta_0 + x_i^\top \beta)\right)\right) + \lambda \|\beta\|^2$$

SVM does not care about observations with margins greater than one, and uses a linear penalty for others. Logistic regression uses a smooth penalty.

A key aspect of SVM (not proven here) is that there are a finite number of support vectors, so that

$$\hat{\beta} = \sum_{i \in \mathcal{S}} \hat{\alpha}_i x_i,$$

where $\mathcal{S}$ is the set of support vectors.

Substituting this form for β into the optimization problem we can write that as

$$\min_{\alpha_0, \alpha_1, \ldots, \alpha_N} \sum_{i=1}^{N} \left[1 - y_i \left(\alpha_0 + \sum_{j=1}^{N} \alpha_j x_j^\top x_i \right) \right]_+ + \lambda \sum_{i,j} \alpha_i x_i^\top x_j \alpha_j$$

still with λ the tuning paramteter.

So far this is very much tied to the linear representation of the classifier.

We can of course add functions of the original x's to make the model aribtrarily flexible.

It turns out we can do that in a particularly useful way using kernels of the type we use in kernel regression.

This can be generalized by replacing $x^\top x_i$ by a kernel $K(\cdot, \cdot)$ evaluated at x and x_i.

So we change from

$$\min_{\alpha_0,\alpha_1,\ldots,\alpha_N} \sum_{i=1}^{N} \left[1 - y_i \left(\alpha_0 + \sum_{j=1}^{N} \alpha_j x_j^\top x_i\right)\right]_+ + \lambda \sum_{i,j} \alpha_i x_i^\top x_j \alpha_j$$

to

$$\min_{\alpha_0,\alpha_1,\ldots,\alpha_N} \sum_{i=1}^{N} \left[1 - y_i \left(\alpha_0 + \sum_{j=1}^{N} \alpha_j K(x_j, x_i)\right)\right]_+ + \lambda \sum_{i,j} \alpha_i K(x_i, x_j) \alpha_j$$

For example, we can use an radial kernel

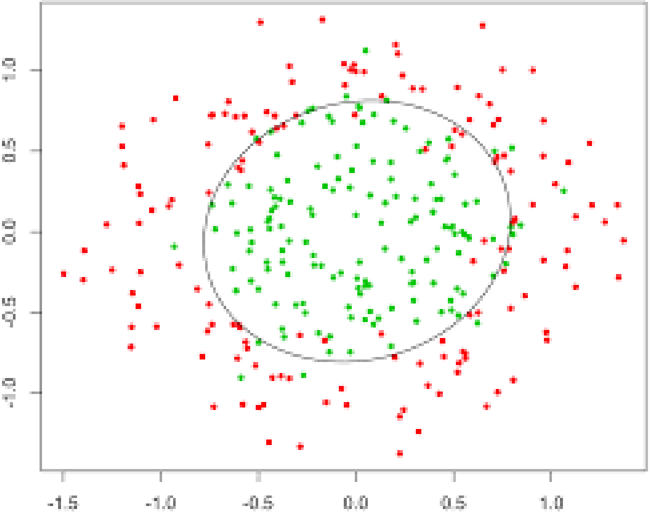
$$K(x, x') = \text{exp}(-\gamma \|x - x'\|)$$

This leads to very flexible surfaces to separate the classes.

• If we choose γ very large, this is essentially the same as before.

• If we choose γ small (close to zero), we get a very non-linear surface separating the classes.

• two tuning parameters, λ and γ.



1.0
0.5
0.0
-0.5
-1.0
-1.5
-1.0
-0.5
0.0
0.5
1.0

References

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Efron, B., and T. Hastie, (2016), *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*, Cambridge University Press.

Herbrich, R. (2016). *Learning kernel classifiers*. Mit Press.

Mohri, M., A. Rostamizadeh, & A. Tawalkar, (2012), *Foundations of Machine Learning*, MIT Press.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press.