Statistical Learning, Fall 2024-5

Class notes 8

Support vector classification (SVC)

The final "linear" method we discuss is based on a different, non-probabilistic approach. Recall, a hyper-plane L in $\mathbb{R}^p$ is defined by a linear constraint:

$$L = \left\{x \in \mathbb{R}^p : x^T\beta + \beta_0 = 0\right\},$$

with corresponding properties:

1. If $x_1, x_2 \in L$ then:

$$\beta \perp (x_1 - x_2) \quad : \quad \beta^t(x_1 - x_2) = x_1^T\beta + \beta_0 - (x_2^T\beta + \beta_0) = 0.$$

2. Signed distance of a point x from the hyper-plane L: If $\beta_0 = 0$ and $\|\beta\|_2 = 1$ then it's easy to see that the (signed) Euclidean distance is $d(x, L) = x^T\beta$, generalization for general β, β_0:

$$d(x, L) = \frac{1}{\|\beta\|_2}(x^T\beta + \beta_0).$$

Now, assume we are given a training set $T = (\mathbb{X}, \mathbb{Y})$ for a 2-class classification problem, and assume for simplicity we encode $Y_i \in \{\pm 1\}$. We define a hyperplane L as *separating* for this data if:

$$Y_i = 1 \Leftrightarrow d(X_i, L) > 0 \ , \ Y_i = -1 \Leftrightarrow d(X_i, L) < 0.$$

When the dimension p is low, such a hyperplane may not exist at all. When p is high, it is likely to exist, and then there will be many (a continuum) of such solutions.

The idea in SVC is to select the hyperplane which "best" separates the data, as defined by the minimal distance of any point to the hyperplane:

$$\max_{\beta,\beta_0} \min_i Y_i \cdot \frac{1}{\|\beta\|_2}(X_i^T\beta + \beta_0).$$

This problem can be formulated as equivalent optimization problems:

$$\begin{aligned} &\max M \ \text{ s.t. } \|\beta\|_2 = 1 \ , \ Y_i \cdot (X_i^T\beta + \beta_0) \geq M \ \forall i \\ &\min \|\beta\|_2^2 \ \text{ s.t. } Y_i \cdot (X_i^T\beta + \beta_0) \geq 1 \ \forall i, \end{aligned}$$

that last formulation is the standard SVC problem as described in the literature. It leads to a linear model, where new observations are classified based on the sign of $d(x_0, L(\beta, \beta_0))$.

The term "support vector" relates to the nature of the solution to this problem:

- The optimal coefficient vector $\hat{\beta}$ has the form $\hat{\beta} = \sum_{i=1}^{n} \hat{\alpha}_i Y_i X_i$.
- $\hat{\alpha}_i = 0$ for all except the *support vectors* which attain equality in the constraint:

$$\hat{\alpha}_i > 0 \Rightarrow (X_i^T \hat{\beta} + \hat{\beta}_0) = 1 \ , \ \hat{\alpha}_i = 0 \Leftarrow (X_i^T \hat{\beta} + \hat{\beta}_0) > 1.$$

Since the number of support vectors is typically small, very efficient algorithms for SVC with big data (large n, large p) have been designed based on searching these support vectors, and this is some of the claim to fame of support vector methods.

What we have described so far is the "hard-margin" SVC, an extension is the soft-margin SVC, which allows violations of the constraint:

$$\min \|\beta\|_2^2 \qquad \text{s.t. } Y_i \cdot (X^T \beta + \beta_0) \geq 1 - \xi_i \ \forall i,$$
$$\xi_i \geq 0 \ \forall i \ , \ \sum_{i=1}^{n} \xi_i \leq C.$$

The interesting property of this problem is that with some simple manipulations and given the (Lagrange) equivalence between constrained and penalized optimization, an equivalent formulation gives a ridge-penalized problem with a nice (Hinge) loss function:

$$(\hat{\beta}, \hat{\beta}_0) = \arg\min \sum_{i=1}^{n} \left(1 - Y_i(X_i^T \beta + \beta_0)\right)_+ + \lambda \|\beta\|_2^2.$$

We can see the close similarity to logistic regression with ridge penalty:

$$(\hat{\beta}, \hat{\beta}_0) = \arg\min \sum_{i=1}^{n} \frac{1}{1 + \exp\left[-Y_i(X_i^T \beta + \beta_0)\right]} + \lambda \|\beta\|_2^2.$$

In fact, if we draw the loss functions for both problems we can see they are quite similar (in particular become parallel as the "loss" $-Y_i(X_i^T \beta + \beta_0)$ becomes big).

Classification methods: summary

Ways of generating linear decision boundaries:

1. Logistic regression: maximize log likelihood with logit link, generalization to k-class
2. SVC: similar discriminative method derived from a geometric/computational perspective
3. Generative method: LDA (naturally applies to $K > 2$ classes)

Decision trees

The general idea: divide the space $\mathcal{X}$ into "neighborhoods" where the response is homogeneous, and use that for prediction. In trees this is done via *recursive partitioning*. A partition $R = \{R_1, ...R_M\}$ is such that $\bigcup_{j=m}^{M} R_m = \mathcal{X}$ and $R_j \bigcap R_k = \emptyset$, $j \neq k$. A recursive partitioning scheme divides the space recursively into smaller regions, which can be described as a tree. The model has the form:

$$\hat{f}(x) = \sum_{m=1}^{L} \hat{c}_m \mathbb{I}\{x \in R_m\}.$$

For regression $\hat{c}_m$ is a number, for classification it is a class.

A decision tree algorithm (for regression or classification) has three elements:

1. How do we choose a split at each node of the tree?
2. How do we decide when to stop splitting? This can be thought of as a bias-variance tradeoff, where bigger trees are more flexible (less bias) but more overfitted to training (more variance). A tree that never stops and ends up with one observation in each leaf is similar to a 1-NN model.
3. How do we fit a value $\hat{y}$ for each terminal node (leaf)?

Some well known decision tree algorithms:

- ID3, C4.5, C5.0: for classification only, invented in the CS/machine learning community
- Classification and regression trees (CART, Breiman, Friedman, Olshen, Stone 1985): invented in the statistics community
- We are going to mostly describe CART, which is the basis for modern methods we discuss later

CART for regression: splitting process and leaves fits

Criterion: Minimize RSS on training.

Given set of r observations in current node, define for a variable j and possible split point s:

$$L(j,s) = \{i \leq r : x_{ij} \leq s\} \ , \quad R(j,s) = \{i \leq r : x_{ij} > s\}$$

$$\bar{y}_L = \frac{\sum_{i \in L(j,s)} y_i}{|L(j,s)|} \ , \ \bar{y}_R = \frac{\sum_{i \in R(j,s)} y_i}{|R(j,s)|}$$

$$RSS(j,s) = \sum_{i \in L(j,s)} (y_i - \bar{y}_L)^2 + \sum_{i \in R(j,s)} (y_i - \bar{y}_R)^2$$

And the obvious choice is to choose the split that minimizes RSS:

$$(\hat{j}, \hat{s}) = \arg\min_{j,s} RSS(j,s).$$

this is the split we do. We split the node into two according to the chosen split and continue.

Complexity: After sorting each variable (cost of $\mathrm{O}(r \cdot \log(r) \cdot p)$ for a node with r observations), we can use summary statistics to find the optimal split in $O(rp)$, so the overall complexity for a node is $\mathrm{O}(r \cdot \log(r) \cdot p)$. However since the sorting has to be done only once for the entire tree, the complexity can be shown to be $O(n(\log(n) + d)p)$ for a tree of depth d, since typically $d << \log(n)$ we simply get $O(n \log(n)p)$. Hence trees are usually fit with exhaustive search, with some simple tricks.

CART for regression: fits at leaves

Similar to OLS, we think we want to estimate $\hat{f}(x) \approx E(y|x)$. We interpret the splitting as finding *homogeneous areas* with similar y values in our data, hence hopefully similar $E(y|x)$. Consequently, given a leaf (terminal node) with set of observations $Q \subseteq \{1, ..., n\}$, we estimate:

$$\hat{f} = \bar{y}_Q = \frac{\sum_{i \in Q} y_i}{|Q|},$$

although other variants like using median or fitting simple models in the leaves also exist.

CART for regression: stopping and pruning

CART method: build a big tree, then *prune* it into small tree that predicts best.

Given a *big* tree T_0, a subtree T, denoted $T \subseteq T_0$, is a tree that can be reached from T_0 by deleting terminal subtrees and replacing them with a single node. Formally, T is generated from T_0 by selecting a subset of the nodes in T_0, and for each one of them, replacing the tree hanging from it with a single leaf (terminal node).

We define the collection of *optimally pruned trees* as the set of solutions to the following problem, for all α:

$$T_\alpha = \arg \min_{T \subseteq T_0} RSS(T) + \alpha |T|,$$

where $|T|$ is the number of nodes in T. It is easy to see that $\lim_{\alpha \to 0} T_\alpha = T_0$ the full tree, and $\lim_{\alpha \to \infty} T_\alpha$ is the "empty" tree that has the root only.

The CART book shows that the following greedy algorithm generates all optimal tree $\{T_\alpha : 0 \leq \alpha < \infty\}$:

1. Set $T = T_0$
2. Repeat:
   (a) For each node $u \in T$, denote by T_u the terminal subtree hanging from u. Calculate:

$$s_u = \frac{\text{Increase in RSS from deleting } T_u \text{ and replacing it with a single node}}{|T_u| - 1}.$$

   (b) Delete from T the subtree with the minimal s_u and replace it with a single leaf.

The sequence of trees generated by this algorithm contains all optimally pruned trees T_α as defined above.

The CART book then suggests to select the optimal pruning level $\hat{\alpha}$ by minimizing prediction MSE on holdout data not used in training:

$$\hat{\alpha} = \arg\min_{\alpha} \text{Test MSE of } T_\alpha.$$

The book further suggests that we prefer smaller trees (bigger α) so instead of $\hat{\alpha}$ above it proposes using a bigger value:

$$\hat{\hat{\alpha}} = \text{maximal } \alpha \text{ such that Test MSE of } T_\alpha \text{ is not much worse than Test MSE of } T_{\hat{\alpha}}.$$

The criterion for "not much worse" is one standard error as defined from the holdout data (details omitted), and therefore this is called *the 1-SE rule*.

Trees for classification

Trees in general are quite good for classification, and they use similar approaches for two classes ($K = 2$) or $K > 2$. The main difference in classification vs regression trees is in the splitting criterion instead of RSS. Possible criteria for classification trees:

1. Misclassification error:

$$MC(j,s) = \sum_{i \in L(j,s)} \mathbb{I}\{y_i \neq \hat{y}_L\} + \sum_{i \in R(j,s)} \mathbb{I}\{y_i \neq \hat{y}_R\}$$

2. Gini index (used in CART)

$$Gini(j,s) = \sum_{k=1}^{K} n_L \hat{p}_{kL}(1 - \hat{p}_{kL}) + \sum_{k=1}^{K} n_R \hat{p}_{kR}(1 - \hat{p}_{kR})$$

3. Information gain (used in C4.5)

$$Gain(j,s) = \sum_{k=1}^{K} n_L \hat{p}_{kL} \log(\hat{p}_{kL}) + \sum_{k=1}^{K} n_R \hat{p}_{kR} \log(\hat{p}_{kR})$$

where $\hat{y}_L, \hat{y}_R$ are the most common classes, n_L, n_R are the number of observations in each side of the split, and $\hat{p}_{kL}, \hat{p}_{kR}$ are the empirical proportions of each class in the respective sides of the split.

In practice, MC is not used and the other two usually yield similar splits.

The obvious changes in fitting leaves (majority class) and stopping or pruning rules (using the same criterion as splitting) complete the classification case.

Trees and categorical features

An important class of explanatory variables is categorical ones (city, car model,...), and many predictive modeling approaches struggle with those. For example, the standard solution in linear regression for a $Q-category$ predictor is "one-hot encoding": adding $Q-1$ binary dummy variables to the regression. Since we know prediction variance is linear in the number of variables, we can see the statistical difficulty, and it can also lead to computational difficulties.

Assume variable j is categorical and its categories are labeled $\{h_1, \dots h_Q\}$. In CART, all splits are binary, so splitting on this categorical variable with Q categories involves partitioning these categories into two subsets: $L \cap R = \emptyset, L \cup R = \{h_1 ... h_Q\}$, such that if $X_{ji} \in L$, then observation i goes left and otherwise right. The optimal split as usual is the one optimizing the splitting criterion (RSS, Gini, or whatever applies).

Thus this raises two types of problems:

- Computational: there are $O(2^Q)$ possible splits (ways of partitioning Q objects into two subsets). Trying them all is not practical when Q is not small. As it turns out, this is actually not a problem for regression and two class classification, see below.
- Statistical: there are $O(2^Q)$ possible splits, so the optimal one may be badly overfitted. To illustrate that, assume we are in a two-class classification problem and $Q = n$ (unique value for each observation, for example an identifier which we mistakenly take as a categorical variable). In this setting, a single split can perfectly fit the training data: send all observations with $y_i = 0$ left and those with $y_i = 1$ right according to their identifiers. This is of course not a useful prediction model for new data. This problem exists also when $Q < n$ to a less extreme extent: it is a variance issue.

The CART book shows that the computational problem can be overcome for regression and two-class classification with the following simple algorithm (for regression):

1. For each category h_q denote $\bar{y}_q$ the average of the observations of class q in the current node.
2. Sort categories in increasing order of $\bar{y}_q$: $\bar{y}_{(1)} \leq \bar{y}_{(2)} \leq \ldots \leq \bar{y}_{(Q)}$, where the notation $_{(q)}$ is for the order statistic (sorted list).
3. The optimal split on the training data is guaranteed to be one of the $Q-1$ splits along this list, separating the categories with smaller $\bar{y}$ from the ones with bigger.

For 2-class classification, we simply replace $\bar{y}_q$ with $\hat{p}_q$, the empirical proportion of class 1 in each category.

It is important to empahsize that this solves the exponential complexity issue ($O(Q \log(Q) + n)$ instead of $O(2^Q + n)$) but not the statistical complexity issue.

C4.5 and its related methods do Q-way splits in this setting, an interesting idea, but not commonly used these days as far as I know.

Trees and missing variables

An important problem, which arises also in our class competition, is dealing with missing values of the features (explanatory variables). Here two many mainstream methods for predictive modeling struggle.

Tree on the other hand offer many interesting solutions for dealing with this, and the one we discuss is the surrogate variable idea in CART: after we choose a split $(\hat{j}, \hat{s})$, we look for other splits on different variables that are "similar" to the chosen split. These can serve to apply the split to observations where the chosen variable $\hat{j}$ is unobserved. (Code demonstration on competition data in class).

Summary: advantages and shortcomings of trees

Advantages:

1. Intuitive interpretation as "neighborhoods" based on splits on single variables
2. Non-parametric nature, flexibility and richness of expression
3. Dealing with important issues that other methods struggle with: categorical features, missing data

Disadvantages:

1. Simply not good predictive modeling approach in terms of accuracy!
   Main problem: greedy approach means the trees are very sensitive to small changes in data $\Rightarrow$ high variance and instability
2. Intuitive appeal is overstated: since they are unstable, we cannot really think of them as expressing "deep truth" in our data, if changing it a bit can change the tree completely!

In the next couple of weeks we will discuss a critical idea: *ensemble methods* like random forest and boosting that attempt to take advantage of the advantages of trees and build accurate predictive models by using trees as a *subroutine* that is applied many times and combined in different ways.