Statistical Learning, Spring 2017

Homework exercise 4

Due date: 26 June 2017

(* Note: another problem will be added to the final version of this homework, probably on boosting *)

1. **Categorical splitting algorithm for CART**
   Prove the favorable property mentioned in ESL 9.2.4: if we are splitting on a categorical variable X_j with q values and looking for the optimal split in terms of either squared error reduction (for regression) or Gini index (for 2-class classification), then an optimal split out of the possible 2^{q-1} splits is always one of the $q-1$ splits defined by:

   - Sort the q groups by their average response: $\bar{y}_{(1)} \leq \bar{y}_{(2)} \leq ... \leq \bar{y}_{(q)}$.
   - Consider only splits along this sequence, i.e., ones which divide according to whether $X_j \in \{g_{(1)}, g_{(2)}, ..., g_{(k)}\}$ for some $k < q$.

   Hint: Prove this by negation, showing that you can improve a split that does not comply with this condition by "switching" values of X_j between the splits.

2. **Playing around with trees**
   Run a variety of tree-based algorithms on our competition data and show their performance. Compare:

   - Small tree without pruning
   - Large tree without pruning
   - Large tree after pruning with 1-SE rule
   - Bagging small trees (100 iterations)
   - Bagging large trees (100 iterations)

   Do this under five-fold cross validation on our competition training set, and use the results of the five different folds to calculate confidence intervals for performance. Plot all the results in a reasonable way (e.g. using `boxplot()`) and comment on them. Explain your choices of "small" and "large".
   Hints: a. Start early since bagging may take a while to run. b. Use as a basis the code from class which implements much of this.

3. **ESL 7.5 (7.6 in 2nd edition): Degrees of freedom of Nearest Neighbors**
   Prove that in the standard i.i.d error model (which the book calls "additive error"), the effective degrees of freedom of k-NN with N observations is N/k.

4. **ESL 7.8 (7.9 in 2nd edition): Trying out model selection methods**
   The use of BIC and bootstrap .362 is optional.
   Tip: for all-subset modeling in R, you can use the function `leaps()` in the package of the same name, which you may need to download from the CRAN repository[1] and install.

[1] http://cran.r-project.org/web/packages/

5. **"Kernel" representation of regular ridge regression**

Consider the ridge regression model in the $p > n$ scenario (assume full rank, so $Rank(X) = n$).

(a) (* +5) Show that the ridge regression solution $\hat{\beta} = (X^T X + \lambda I_p)^{-1} X^T Y$ can be written as $\hat{\beta} = X^T \hat{\alpha}$ where $\hat{\alpha} \in \mathbb{R}^n$ is given by:

$$(XX^T + \lambda I_n)^{-1} Y.$$

Suggested approach (there are other approaches based on algebra identities or optimization theory):

i. Consider the "wasteful" SVD $X = U_{n*p} D_{p*p} V^T_{p*p}$, where the last $p - n$ columns of U and the last $p - n$ rows and columns of D are all zeros. The non-zero columns of U are of course orthonormal (as are all columns of V).

ii. Write the ridge solution $\hat{\beta}$ in terms of this SVD: $\hat{\beta} = VD(D^2 + \lambda I_p)^{-1} U^T Y$ (show the derivation).

iii. Notice that $U^T U$ is a matrix with I_n at the top left and zeros elsewhere. Use this to argue that $VD = VDU^T U$.

iv. Now argue that $U(D^2 + \lambda I_p)^{-1} U^T = (XX^T + \lambda I_n)^{-1}$ (explain why).

(b) Since we now know the optimal $\hat{\beta} \in \mathbb{R}^p$ has a representation as $X^T \hat{\alpha}$ with $\hat{\alpha} \in \mathbb{R}^n$, we can now plug the representation $\beta = X\alpha$ into the ridge criterion $\|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$.
Do this, and rewrite this criterion as a function of only Y, $K = XX^T$ and α.

(c) Explain why this means that if we are given K we can solve the problem in a complexity that does not depend on p. If we have to calculate K as well from X, does this representation still help to solve the problem more efficiently than calculating $\hat{\beta}$ directly? Explain briefly.

(d) If we want to predict at a new point x_0, can we do it based on its n inner products $K_{0i} =< x_0, X_i >$ without needing to represent X or x_0 explicitly?

6. **AdaBoost and ϵ-Adaboost**

This problem refers to the AdaBoost algorithm (Freund and Schapire, 1997), which is used for binary classification with labels $y_i \in \{\pm 1\}$. Adaboost initializes $\hat{f}_0 = 0, w_i \equiv 1$, then for $t = 1...T$ updates:[2]

(a) Fit a classification tree with response y and weights w on the observations, getting tree h_t

(b) Denote by Err_t the (weighted) misclassification error of h_t

(c) Set $\alpha_t = 0.5 \log((1 - Err_t)/Err_t)$

(d) Update weights: $w_i \leftarrow w_i \exp(-\alpha_t (y_i h_t(x_i)))$

The model after step t is $f_t(x) = \sum_{u=1}^{t} \alpha_u h_u(x)$, the final model is f_T, and classification if according to the sign of $f_T(x)$.
As we said in class, from the coordinate descent perspective of boosting, AdaBoost can be viewed as gradient boosting with loss function $L(y, \hat{y}) = \exp(-y\hat{y})$ and line-search steps, explicitly:

- Fitting a classification tree is minimizing $\sum_i w_i y_i h_k(x_i) = \langle wy, h_k(X) \rangle$ (so $w_i y_i$ is the actual gradient)
- The calculated α_t is the solution to the line search problem: $\alpha_t = \arg\min_\alpha \sum_i L(y_i, (f_{T-1} + \alpha h_t)(x_i))$
- The updated w_i is indeed (proportional to) the absolute value of the gradient:

$$w_i \propto \left| \frac{dL(y_i, l)}{dl} |_{l=\hat{f}_t(x_i)} \right|$$

[2]Note that this algorithmic description differs in a couple of points from what I wrote on the board in class – the description here is the accurate one

(a) Choose one of the three properties above and prove explicitly that it holds (for example, if you choose the first one, show how fitting h_k classification tree minimizing weighted miclassification error is equivalent to choosing a coordinate descent direction in the exponential loss function).

(b) The code in www.tau.ac.il/∼saharon/StatsLearn2017/AdaBoost.r implements AdaBoost on the competition data for the problem of whether $y > 3$ or $y \leq 3$. Read the code carefully to make sure you understand the details. Note especially the parameters "method" and "weights" that rpart takes.

i. With 8000-2000 training-test division as in the code, run the algorithm for 1000 iterations and draw a plot of training and test misclassification as a function of iterations. Explain its form.

ii. Change the algorithm from line-search boosting to ϵ-boosting with $\epsilon = 0.01$, not changing the other parameters. Run this version for 1000 iterations and draw the same plot. Discuss the results and compare to the line-search version.

iii. Now change the loss function from the exponential loss to squared error loss, and the approach to the regular gradient boosting with regression tree. Explain briefly in writing what you did, and run with the same parameters ($\epsilon = 0.01, T = 1000$). Classify according to sign of $\hat{y}$ and repeat the same analysis again. Discuss the relative results.

iv. (* +5) Play with the parameters in one (or all) of the algorithms (tree depth or other stopping criteria, ϵ, T) to change the results. Show how you can guarantee much better training results. Can you find settings that give much better testing results?