# STAT406- Methods of Statistical Learning Lecture 16

Matias Salibian-Barrera

UBC - Sep / Dec 2018

# Classification Trees - Bagging

- We can obtain a "large" number of trees and have them "vote" on the classification of future observations, or average their conditional probabilities estimates $\hat{P}\left(g_j|\mathbf{X}\right)$

# Classification Trees - Bagging
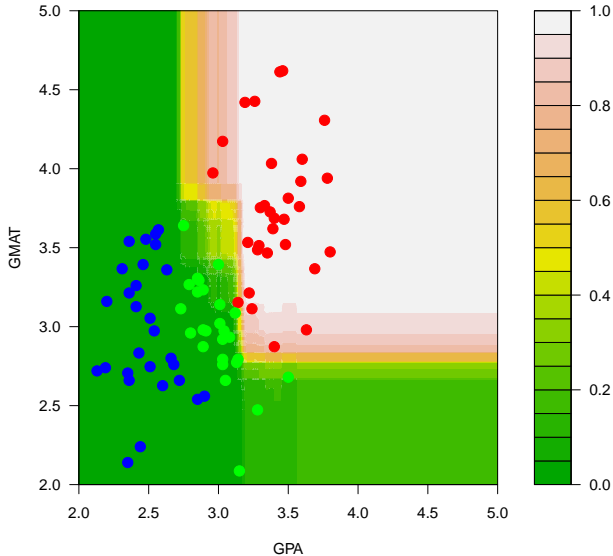
- Our aggregated classifier is

$$\bar{f}(\mathbf{X}) = \left\{ \begin{array}{c} \left(\bar{P}(g_1|\mathbf{X}), \ldots, \bar{P}(g_K|\mathbf{X})\right) \\[2ex] \arg\max(n_1, n_2, \ldots, n_K) \end{array} \right\}$$

where $\left(\bar{P}(g_1|\mathbf{X}), \ldots, \bar{P}(g_K|\mathbf{X})\right)$ are averaged conditional probabilities over the boostrap samples;
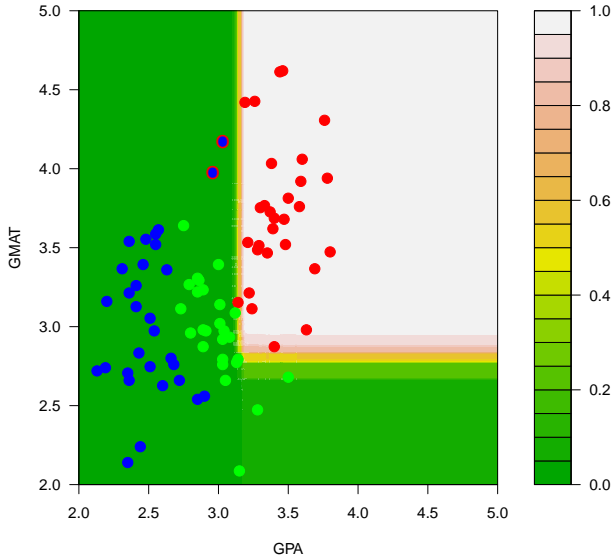
$(n_1, n_2, \ldots, n_K)$ are the number of times each class was selected and
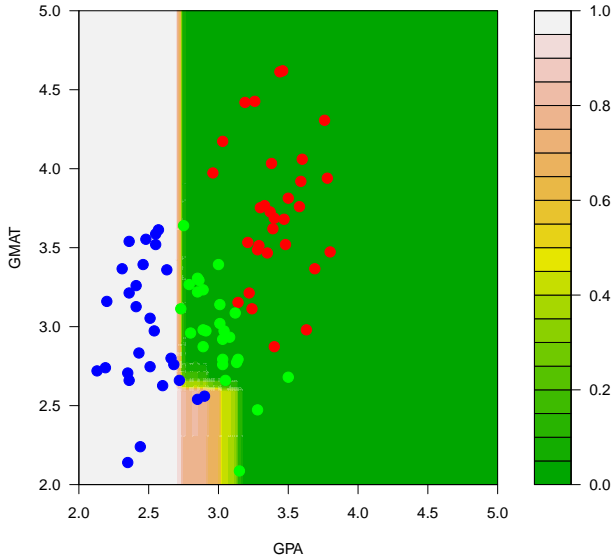$n_1 + n_2 + \cdots + n_K =$
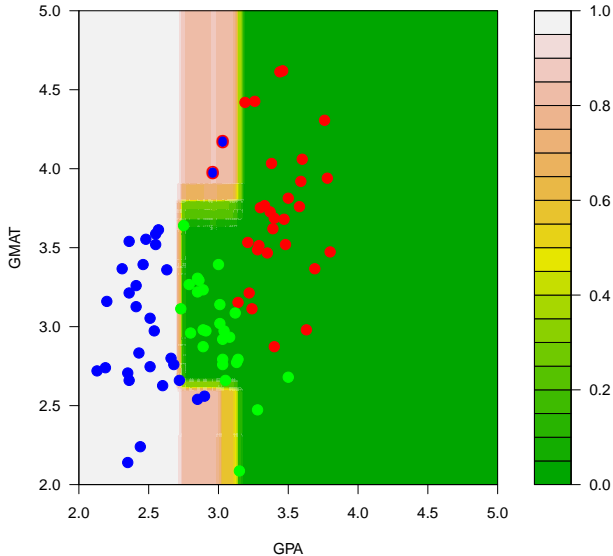number of bootstrap samples

# Bagged trees -original data

4

# Bagged trees -modified data
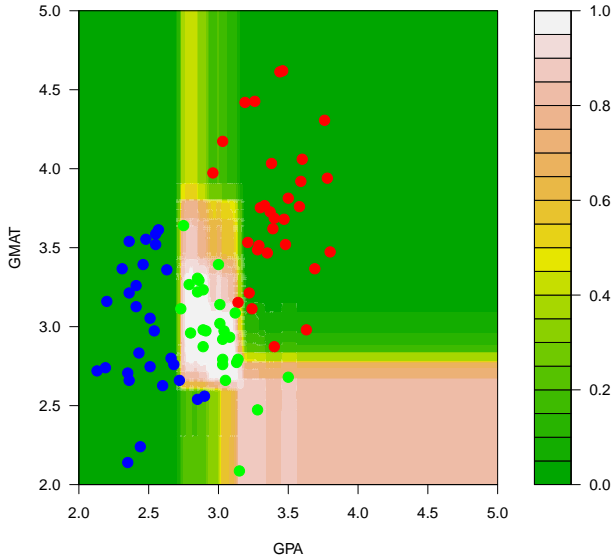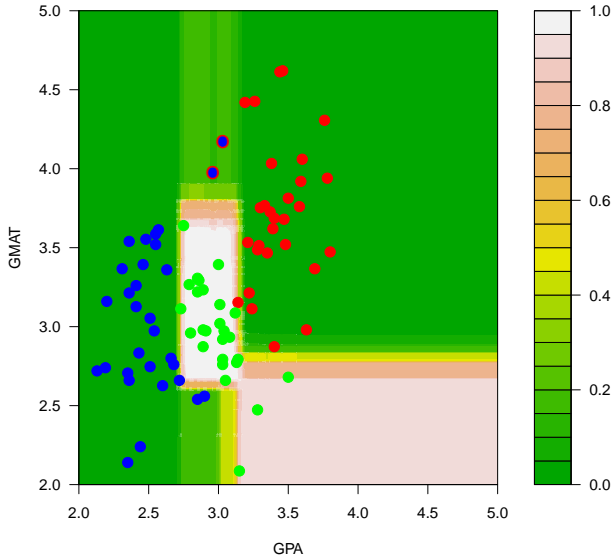
5

# Bagged trees -original data

# Bagged trees - modified data

# Bagged trees -original data

8

# Bagged trees - modified data

# Random forests

- Bagging - averaging identically distributed trees (which may be correlated)

- Random forests - making the "bagged" trees less correlated

- The bootstrapped trees are de-correlated by making them use different features for the splits

10

# Random forests

(1) `for(b in 1:B)`

    (a) Draw a bootstrap sample from the training data

    (b) Grow a "random forest tree" as follows: for each terminal node:

        (i) Randomly select `m` features

        (ii) Pick the best split among these

        (iii) Split the node into two children

    (c) Repeat (b) to grow a (very very) large tree

(2) Return the ensemble of trees $(T_b)_{1 \leq b \leq B}$

# Random forests

- Given a new point **x**, for regression we use

$$\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} T_b(\mathbf{x})$$

- For classification:

$$\hat{f}(\mathbf{x}) = \text{majority vote among} \left\{ T_b(\mathbf{x}), \\ 1 \le b \le B \right\}$$

Q: why not average conditional prob's?

12

# Out-of-bag error estimates

- Each bagged tree is trained on a bootstrap sample

- Predict the observations not in the bootstrap sample with that tree

- One will have "about" $B/3$ predictions for each point in the training set

- These can be used to estimate the prediction error (classification error rate) without having to use CV

# Out-of-bag error estimates

- For each training observation $(y_i, \mathbf{x}_i)$, obtain a prediction using only those trees in which $(y_i, \mathbf{x}_i)$ was **NOT** used

- In other words, let $\mathcal{I}_i$ the set of trees (bootstrap samples) where $(y_i, \mathbf{x}_i)$ does not appear, then

$$\hat{y}_i \; = \; \frac{1}{|\mathcal{I}_i|} \sum_{j \in \mathcal{I}} T_j(\mathbf{x}_i)$$

# Random forests

- This error estimate can be computed at the same time as the trees are being built

- When this error estimate is stabilized we can stop adding trees to the ensemble

# Example

*OOB example*

16

# Random forests

- Feature ranking - relative importance of each variable

- Given a single tree $T$, at each node $t$ split we can compute the sum of reductions in sum of squares (or gini or deviance measures) $m_t^2$

- We assign this squared measure $m_t^2$ to the variable (feature) used in the split

# Random forests

- To each feature, we assign the sum of "squared gains" attributed to it

- For the i-th variable $X_i$ we have

$$\mathcal{J}_i^2(T) = \begin{cases} m_t^2 & \text{if split involved } X_i \\ 0 & \text{otherwise} \end{cases}$$
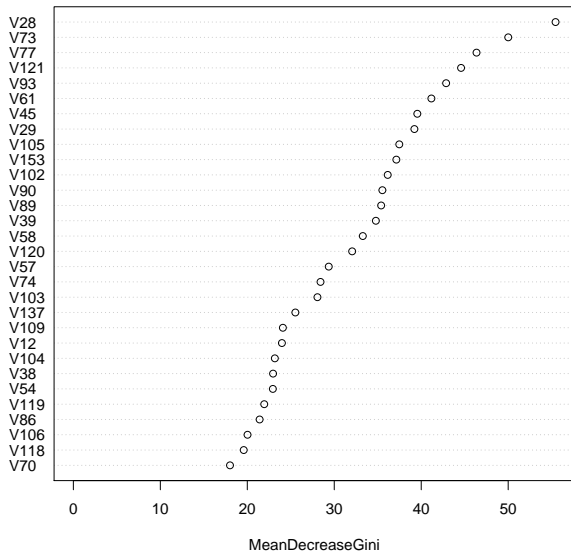
# Random forests

- For a random forest we use

$$\mathcal{J}_i^2 \; = \; \sum_T \mathcal{J}_i^2(T)$$

- In other words, we sum (or average) the importance of the variable across the trees in the forest

# Random Forest - plot



Zip codes

# Ensembles

- Ensembles of classifiers

- Combine classifiers trained on the same (or similar [e.g. bootstrapped]) data

- Consensus is reached by (equally weighted) voting or averaged estimated probabilities.

- Bagging and Random Forests are examples of ensembles.

# Boosting

- Originally proposed for classification

- Main idea: sequentially re-train a simple classifier assigning more importance to points that were previously misclassified

# Boosting

- The end result is a weighted average of all the classifiers

- Interesting ideas:

  - Not all components of the ensemble are treated equally

  - Members of the ensemble use information about other members

  - The underlying loss function has a "margin" (unlike 0-1 losses)

# Boosting - AdaBoost.M1

**Algorithm**. Data $(y_i, \mathbf{x}_i)$, with $y_i \in \{-1, 1\}$

- Set initial weights $w_i = 1/n$, $1 \leq i \leq n$
- For $j = 1, \ldots, K$
- Build a classifier $T_j(\mathbf{x})$ to the data using weights $w_i$, $1 \leq i \leq n$

# Boosting - AdaBoost.M1

- Let

$$e_{\mathbf{j}} = \sum_{\mathbf{i}=1}^{n} w_{\mathbf{i}} I\left(y_{\mathbf{i}} \neq T_{\mathbf{j}}(\mathbf{x_i})\right) \Big/ \sum_{\ell=1}^{n} w_{\ell}$$

- Let $\alpha_{\mathbf{j}} = \log\left((1 - e_{\mathbf{j}})/e_{\mathbf{j}}\right)$ and

$$w_{\mathbf{i}} = w_{\mathbf{i}} \exp\left(\alpha_{\mathbf{j}} I\left(y_{\mathbf{i}} \neq T_{\mathbf{j}}(\mathbf{x_i})\right)\right), \mathbf{i} = 1, \ldots, n$$

- Final classifier:

$$T(\mathbf{x}) = \text{sign}\left(\sum_{\mathbf{j}=1}^{K} \alpha_{\mathbf{j}} T_{\mathbf{j}}(\mathbf{x})\right)$$