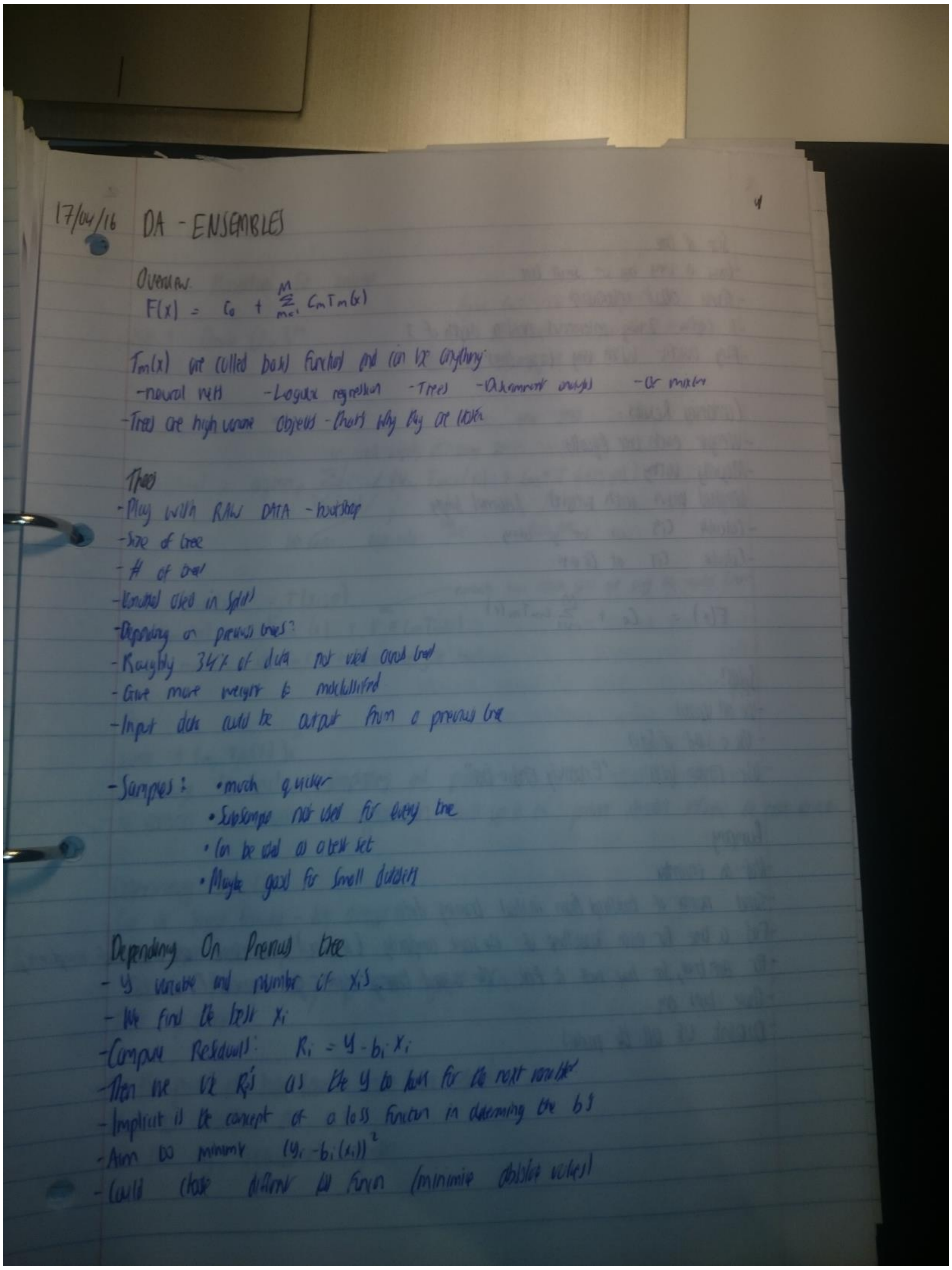


17/04/16 DA - ENSEMBLES

4

Overview:

$$F(x) = c_0 + \sum_{m=1}^{M} c_m T_m(x)$$

$T_m(x)$ are called basis functions and can be anything:

- neural nets
- Logistic regression
- Trees
- Discriminant analysis
- Or mixture

- Trees are high variance objects - that's why they are chosen

Trees

- Play with RAW DATA - bootstrap
- Size of tree
- # of trees
- Variables used in splits
- Depending on previous trees?
- Roughly 34% of data not used in each tree
- Give more weight to misclassified
- Input data could be output from a previous tree

- Samples:
  - much quicker
  - Subsample not used for every tree
  - Can be used as a test set
  - Maybe good for small datasets

Depending On Previous tree

- y variable and number of x_i's
- We find the best x_i
- Compute Residuals: $R_i = y - b_i x_i$
- Then use the R_i's as the y to look for the next variable
- Implicit is the concept of a loss function in determining the b's
- Aim to minimise $(y_i - b_i(x_i))^2$
- Could choose different loss function (minimise absolute values)

Size of tree
- Grow a very big or small tree
- Think about interactions
- To capture 2-way interactions, need a depth of 3
- May overfit with very large trees.

Combining Results
- Weight each tree equally
- Majority voting
- Weighted mean with weights determined simply
- Calculate C_i's as we go along
- Calculate C_i's at the end

$$F(x) = C_0 + \sum_{m=1}^{M} C_m T_m(x)$$

Splits?
- Use all variables
- Use a subset of splits
- Use random splits "Extremely random tree"

Bumping
- Not an ensemble
- Takes number of bootstraps from initial training data.
- Fits a tree for each bootstrap of the same complexity (# nodes) or some other measure of complexity.
- For each tree, see how well it fits the original training data - gets measure of fit.
- Chose best one
- Doesn't use all the models

17/04/16

DA - ENSEMBLES

Generic Algorithm for ensemble

Step 1: Choose $\{P_m\}_1^M$ — Ignore C_m's for time being; M = number of models to fit

$F_0(x) = 0$ — Start by assuming it is 0

For $m=1$ to M { — iterate over m's to M

$$(P_m, C_m) = \text{argmin}_P \underbrace{\sum_{i \in S_n(n)}}_{\text{controls sampling eg. using } S_n} \underbrace{L}_{\text{loss function}}(\underbrace{y_i}_{\text{target value}}, \underbrace{F_{m-1}(x_i)}_{\text{like SSE}} + C_m * \underbrace{T(x_i; p)}_{\text{Best tree}})$$

$T_m(x) = T(x_i; p)$

$F_m(x) = F_{m-1}(x) + \underbrace{v}_{\text{controls how much you are going to update from}} * C_m T_m(x)$

new = old $F_m x$ + tree we just built

}

write $\{C_m, T_m(x)\}_1^M$

- Can change loss function, sampling and parameter v
- v controls how much the approximation built up to the present iteration influence the next iteration

Determining the C_i's
- Can use simple formula - like average
- Determine at each step
- Calculate afterwards using penalised regression function
- Post processing

- At this point all base learners have been selected
- Have to determine the C_i's
- Do this by regularised regression.

$$\{C_m\} = \arg\min_{C_m} \sum_{i=1}^{N} L\left(y_i, C_0 + \sum_{m=1}^{M} C_m T_m(x) \right) + \lambda \cdot P(c)$$

$P(c)$ is the complexity penalty and λ controlling the amount of regularisation

Bagging
- Bootstrap aggregation - Brieman
- $\lambda(y, \hat{y}) = (y - \hat{y})^2$
- $v = 0$ - builds independent trees - don't determine next ones
- $n = N/2$ bootstrap size, can take other sizes
- $T_m(x)$ - you decide
- $C_0 = 0$ $C_m = 1/m$

- Sample size of n
- Prob being ommitted in a single $(1 - \frac{1}{n})$
- Prob being ommitted in all draws $(1 - \frac{1}{n})^n$
- Approximately 37% of sample excluded
- Works best when predictors are unstable
- Reduces variance
- loose simple structure

- Performs and exhaustive search for best predictor to split on
- Grows trees in series, with later trees dependent on the results of previous trees
- May be more difficult to model and requires more attention to parameter tuning
- On large training sets, can be slow with many predictors
- Can be sensitive to noisy data and outliers in data
- Extremely accurate

17/04/16

DA ENSEMBLE

RANDOM FORESTS

- Ensemble method based on trees - Brieman
- Two kinds of randomness built in
- Cases are selected at random with replacement - training set.
- At each split a random sample of m from M variables are selected
- m can be any number, typically $\sqrt{M}$
- No pruning takes place theoretically
- Typically 100 trees are grown
- About 36% of data not used - data are called out of bag samples (oob)
- Each tree votes for each case in the oob sample.
- Aggregated over all trees
- Every tree carries equal weight.
- Each case is assigned to the class with most votes

Generic Algorithm

- Determine base learners
- Determine weight c_i's

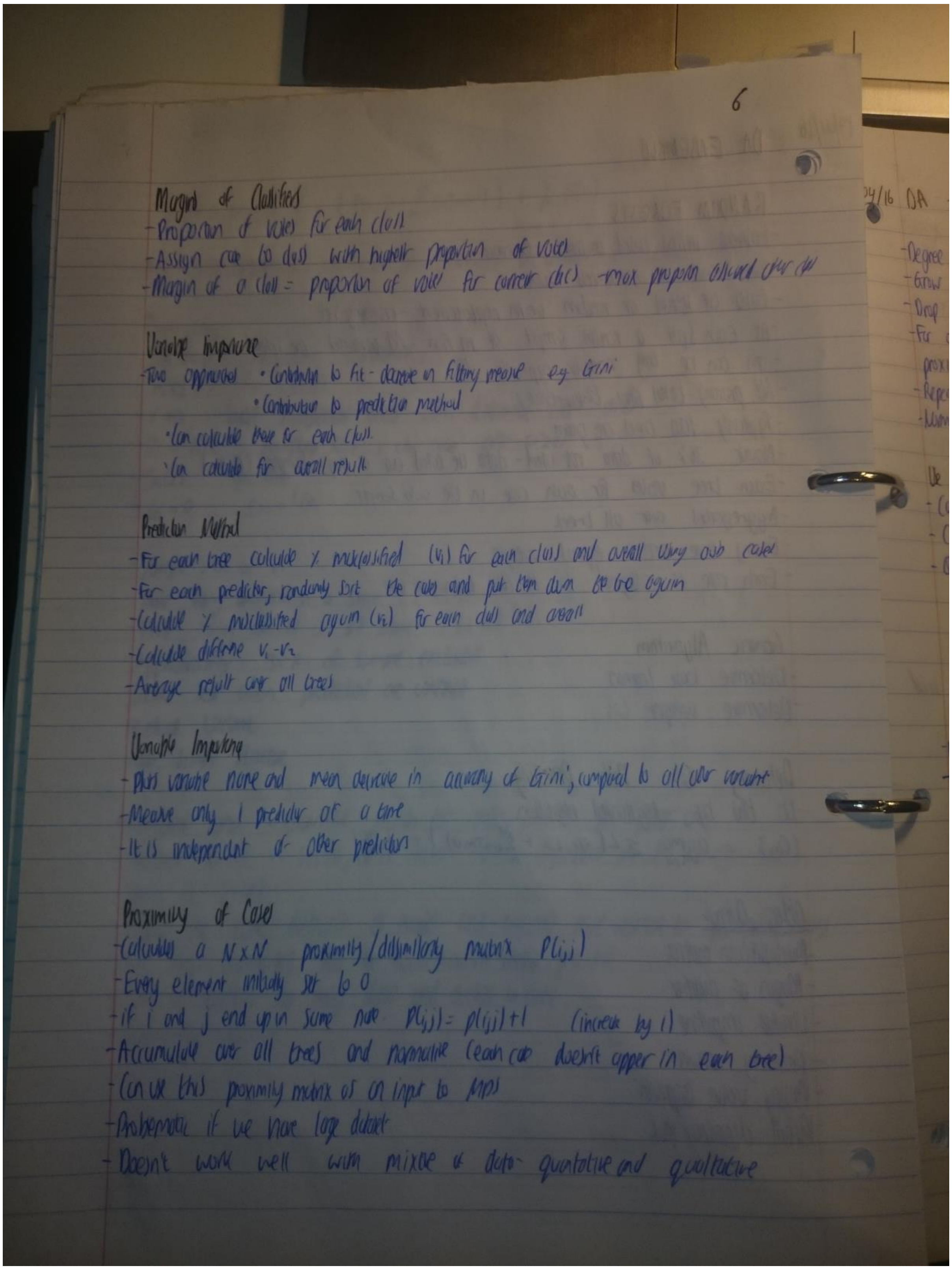
Determining c_i's - Post Processing

- Do this by regularised regression.

$$\{c_m\} = \underset{c_m}{\arg\min} \sum L\left(y_i, c_0 + \sum_{m=1}^{M} c_m T_m(x)\right) + \lambda . P(c)$$

<u>Other Output</u>

- Misclassification matrix.
- Margin of classifier
- Variable importance
- Proximity matrix
- Missing value imputation
- Partial dependency plots



Margins of Classifiers
- Proportion of votes for each class
- Assign case to class with highest proportion of votes
- Margin of a class = proportion of votes for correct class - max proportion allocated other class

Variable Importance
- Two approaches
  - Contribution to fit - decrease in fitting measure eg Gini
  - Contribution to prediction method
- Can calculate these for each class.
- Can calculate for overall result

Prediction Method
- For each tree calculate % misclassified (v_1) for each class and overall using oob cases
- For each predictor, randomly sort the cases and put them down the tree again
- Calculate % misclassified again (v_2) for each class and overall
- Calculate difference $v_1 - v_2$
- Average result over all trees

Variable Importance
- Plots variable name and mean decrease in accuracy of Gini, compared to all other variables
- Measure only 1 predictor at a time
- It is independent of other predictors

Proximity of Cases
- Calculates a $N \times N$ proximity/dissimilarity matrix $P(i,j)$
- Every element initially set to 0
- if i and j end up in same node $P(i,j) = P(i,j) + 1$ (increase by 1)
- Accumulate over all trees and normalise (each case doesn't appear in each tree)
- Can use this proximity matrix as an input to MDS
- Problematic if we have large dataset
- Doesn't work well with mixture of data - quantitative and qualitative

20/04/16 DA - RF

- Degree to which individual observations are classified alike
- Grow a tree as usual
- Drop all the training (in bag and out of bag) down the tree
- For all possible pairs of cases, if a pair lands in the same terminal node, increase their proximity by 1.
- Repeat until all trees have been grown
- Normalise by dividing by the number of trees

Use of proximity matrix
- Can be used as input for MDS
- Can be used for imputing data
- Quantitative data
    - For each variable, calculate median value and use for missing cases
    - Grow tree and calculate proximities
    - Weighted average over non-missing cases using proximities as weights
    - Assign this as new imputed value
    - Re-iterate
- RF. doesn't deal with missing data
- Only top 100 closest cases are recorded

Partial dependency plots
- Show how each predictor is related to response holding other variables constant
- Let x be the initial predictor of interest with v distinct values
- Construct v datasets for each of the v values of x leaving all other variables untouched
- For the v datasets predict the response using random forests
- Calculate a single value averaged over all observations for each dataset
- Average the k predictions over trees
- k categories in output class ($k=2$ normally)
- Plot the following function against the v values of x

$$F_k(x) = \log[p_k(x)] - \frac{1}{k}\sum_{k=1}^{k} \log[p_k(x)]$$

- Compute for each category of predictor. For 2 cats - mirror image of each other

- For two category, being in class 1:

$$F(x) = \log[p_1(x)] - \frac{1}{2}[\log(p_1(x)) + \log(1 - p_1(x))]$$

$$p_1(x) = \frac{e^{2F(x)}}{1 + e^{2F(x)}}$$

- We are plotting half the log odds

Advantages
- One of most accurate learning algorithms available - produces a highly accurate classifier
- Can handle thousands of input variables without variable deletion
- Gives variable importance estimation
- Maintains accuracy when a large proportion of data is missing
- Has methods for balancing error in class population unbalanced data sets
- Simple to implement
- Can use proximity matrix as an input to MDS

- Sometimes overfit some data sets with noisy classification tasks
- For data including categorical variables with a different number of levels, RF is biased in favour of those attributes with more levels
- Therefore, variable importance scores from model is not accurate
- Can be slow to run
- Difficult to see size and direction of main effects - no graph output

Other Points
- P correlation between the trees depends on m
- Increasing p increases the forest error rate
- Increasing the strength of the individual trees decreases forest error rate
- The larger m is, the 'better' the tree
- Reducing m reduces both the correlation and strength
- Increasing m increases both " "
- Find optimum m - results suggest $\sqrt{p}$ where p is the number of variables

/04/16 DA - R.F.

Tuning Parameters
- Node size for growing trees
- Number of trees
- Number of predictors sampled

Costs and R.F.
- Cannot include costs like in a single tree
- But we can alter priors
- Can change the cut-off used to assign class
- In other words do not use majority voting

- For 2 classes we can alter the priors to reflect differences in cost of misclassification

$$\pi^{new}(j) = \frac{C(j|i) \times \pi(j)}{\Sigma_i \, C(i|j) \times \pi_i}$$

$C(j|i) = C(j) =$ cost of misclassifying j

$$\pi^{new}(j) = \frac{C(j) \times \pi(j)}{\Sigma_i \, C(i) \times \pi(i)}$$

VR - Trees
- Base learner is V.R
- At each node toss a coin with α probability of a head
- If a head is obtained, select a split as usual
- Otherwise chose a random feature with a random split point
- The parameter α controls the degree of randomness

Unsupervised learning and RF
- Can just use R.F to create a proximity matrix as input to clustering or multidimensional scaling.
- Another approach:
- Create a new outcome variable with 2 classes
- The first class is all the original data

- Create a synthetic second class of same size - class 2
- For class 2 the independent variables are created by sampling at random from the univariate distributions of the original data

- Can you distinguish between old data and new synthetic data?

- Object is to see if there is structure in the original data
- If the misclassification rate is high (40%) this suggests that there is no structure
- The original data is like random independent data
- low misclassification rates suggest that there is structure in the original data.