AiDM 9: From Ensembles to XGboost

- Bagging
- Boosting
- Stacking
- Random Forests + Boosting + Bagging

Chapters 15,16 (ESLI book)

http://statweb.stanford.edu/~tibs/ElemStatLearn

https://www.youtube.com/watch?v=wPqtzj5VZus

The XGBoost algorithm

https://xgboost.readthedocs.io/en/stable/tutorials/model.html

https://www.youtube.com/watch?v=Vly8xGnNiWs

Many ML algorithms for Classification/Regression

- Decision/regression Trees
- Distance based: Nearest-Neighbor, kernel methods
- Statistical (Linear/Logistic Regression, NaïveBayes, BaysianNetworks, MixtureModels, …)
- SVMs, SVMs with kernels => http://mmds.org/mmds/v2.1/ch12-ml2.pdf
- …
- The Weka Book/Software/Site:
  www.cs.waikato.ac.nz/ml/weka/mooc/dataminingwithweka/

Ensembles of models

Main idea:

instead of building a single model build

several models and combine their outputs

Fact:

It ***usually works better*** than a single model!

How to do it?

Breiman, 96: Bagging

Given: a training set **T** and a learning method
(e.g. C4.5, Backprop, Naive Bayes, ...).

1) Create, say, 100 versions of **T** by
resampling it with replacement: $\mathbf{T_1}$, ..., $\mathbf{T_{100}}$

2) For each T_i build a classifier

3) Return an "ensemble" of classifiers:

Classification: majority voting
Regression: average

Bagging: Advantages

+ No thinking, no tuning
+ Better accuracy (almost for free) / “regularization”

- More computations => easily to run on parallel
- Loss of interpretability

Bagging=bootstrap aggregating

error = bias + variance

bias - limitation of the learning method
variance- limitation of the training data

bagging reduces variance

makes sense when:

- the “basic learner” is sensitive to small changes in data
- the “basic learner” overfits the data (e.g. a full tree)
- the data is scarce

Boosting=learning on errors of others

Main idea:

build a sequence of classifiers C1, C2, …, Ck, as follows:

C1 is trained on the original data
C2 “pays more attention” to cases misclassified by C1,
C3 “pays more attention” to cases misclassified by C1 and C2,
etc.

“pay more attention”=> “try to correct errors” =>
=>“attach bigger weights to misclassified cases”

In other words, boosting develops a number of ‘experts’ that specialize in different regions of the data (the more difficult case the more attention it gets).

Freund, Schapire, Breiman: Boosting

Construct an ensemble of classifiers C_1, ..., C_k, as follows:

Let C_1- a "normal" classifier trained on dataset T
(C_1 becomes initial ensemble E1)

For i=2:k

- assign to every case in T a ***weight*** that is "***proportional to the error"*** made on this case by the current ensemble;

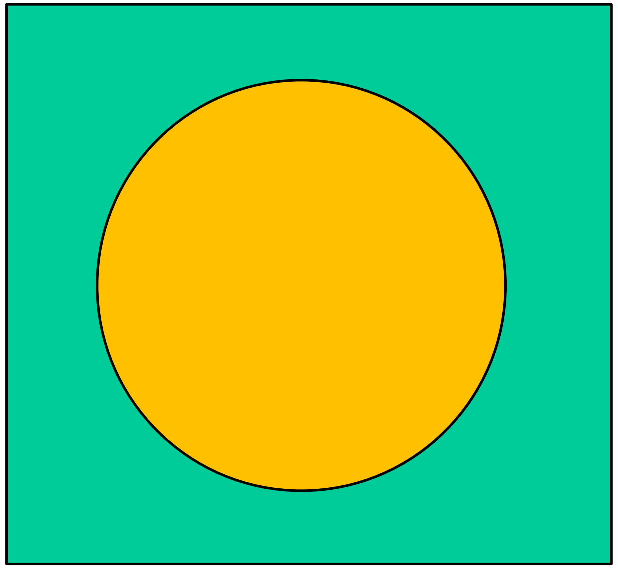
- train C_i tries to correct errors made by E_{i-1} ; $E_i=E_{i-1}+C_i$

"consecutive classifies are trained to correct errors made by the previous ensemble"

Many strategies of weighting errors and weighting $C_{i's}$!

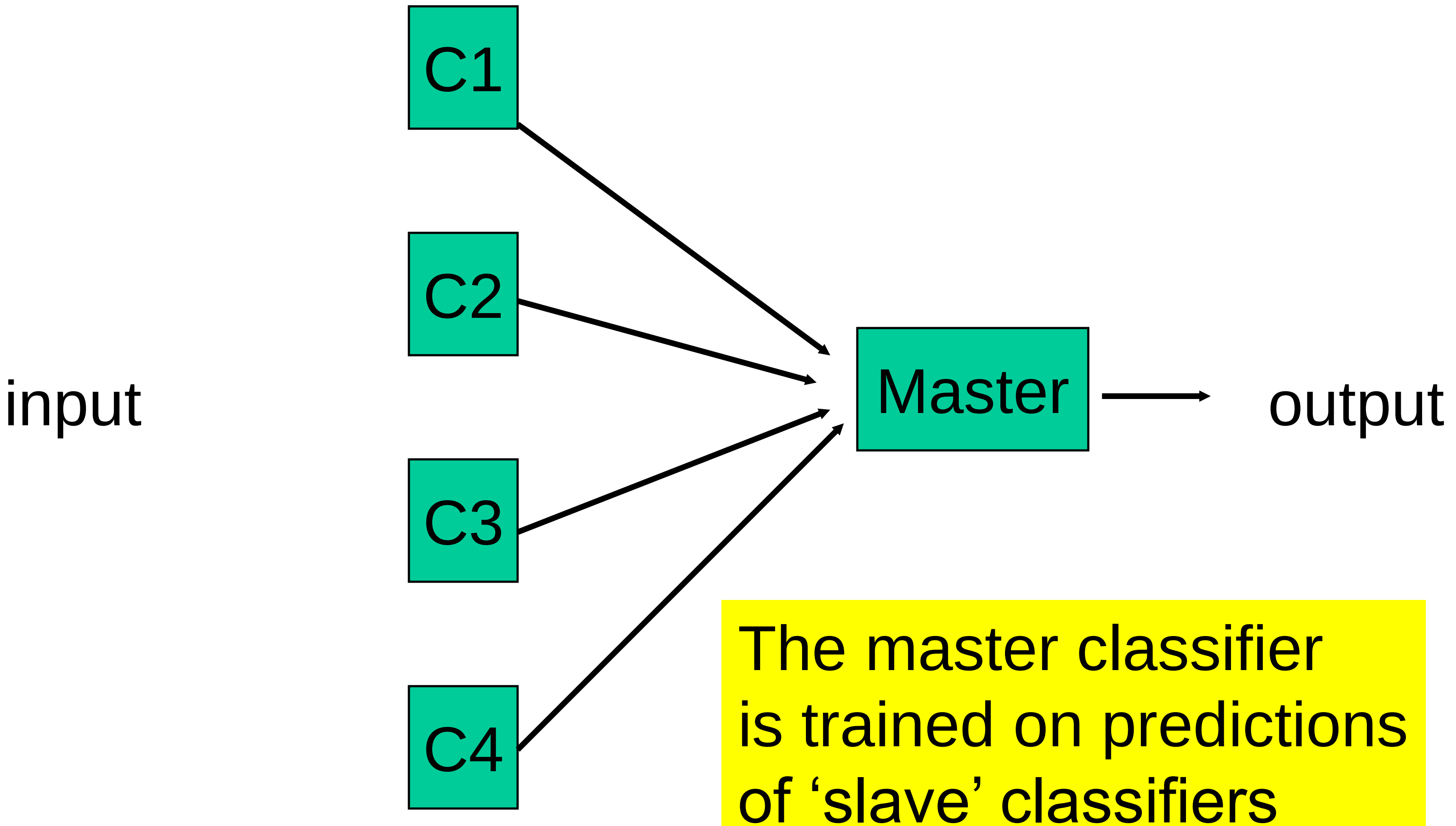
Boosting: properties

- Reduces exponentially fast the error on the training set
- Does not overfit the training set ! (*sometimes…*)
- Most successful with primitive “base classifiers”, e.g., decision stamps, linear regression
- Models expensive to build & difficult to interpret



Example:
separate points within the circle from points outside
with help of “decision stamps”: *x>“a value”* or *y>“a value”*

Stacking: learning how to combine experts



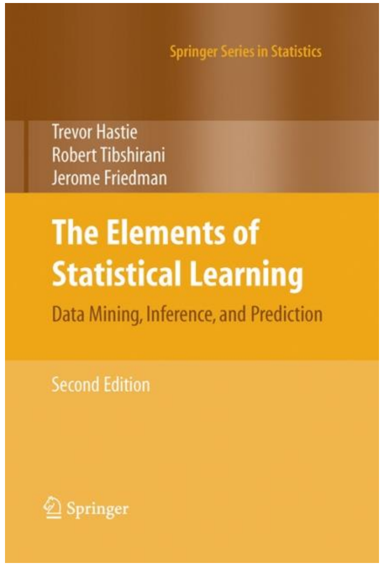
The master classifier
is trained on predictions
of ‘slave’ classifiers

Benchmark Comparisons (many years ago)

- Bagging > boosting > single classifiers
- Boosting > bagging > single classifiers
- SVM>single classifiers
- SVM=Boosting=Bagging

Boosting, Bagging and SVMs are usually better than single classifiers (about 10-20% relative error reduction)

Random Forests (L. Breiman, 2001)



Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data. (looks like bagging!)
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables. (add more randomness: best out of m instead of best of all!)
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\rm rf}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\rm rf}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

Advantages:

- Superior accuracy (comparable with SVM’s, GBDT, NN)
- No cross-validation needed (Out-Of-Bag error estimate)
- Few parameters to tune; highly robust (not very sensitive)
- Trivial to parallelize (?)
- Provides a heuristic measure of importance of variables

Conclusions

- Ensembles usually outperform single models
- Ensembles of “weak” classifiers often outperform single sophisticated classifiers
- Boosting may overfit the data
- Gentle AdaBoost:

  one of many ways of combining/weighting classifiers

Case Study: Benelearn99 competition

A classification problem:

- 5823 cases for training; 4000 for validation
- "Yes"/"No" rate: 6%
- 85 variables

Objective:

select 800 cases from the validation set
with the highest chance of being "Yes"

STEP 1: Framework

Different data sets require different methods ...

How can we decide which one is most appropriate?

Cross-validation:

- data split into train and test set; 6:4
- stratified sampling (the same response rate)
- performance = percentage of correctly predicted “1’s” in top 20%
- each experiment repeated 10 times

average accuracy stability

Conventional Approach

I tried:

Decision trees (C4.5, CHAID, CART, ...)

Naive Bayes Classifier

Logistic Regression

Results:

Logistic Regression > Naive Bayes > Decision Trees

Logistic Regression: best accuracy & smallest variance

Accuracy varying from **11%** to **13%** (expected **88-104** hits)

How could I make it better ???

Boosting Accuracy: Gentle AdaBoost

Gentle AdaBoost (Friedman, Hastie, Tibshirani, 98):

fits an ***additive logistic regression*** model with

a Newton optimization algorithm:

$$score = \sum_i \sum_j a_{ij}(x_i = v_j)$$

Results:

15% on raw data ! (solution 1)

15.5% on pre-processed data (solution 2)

pre-processing: visual inspection of cross-tables 9 variables selected;

one variable manually discretized recoded

Boosting Gentle AdaBoost

- interactions of second order: $(x_i=v)\&(x_j=w)$

  => overfitting

- "clever" weighting strategies

  => no improvement

- "better" pre-processing (recoding)

  => no improvement

- other error measures

  => no improvement

Impressions ...

Gentle AdaBoost is a magic algorithm that:

- outperforms classical methods
- is very fast (7 minutes on pre-processed data)
- no parameters involved (number of iterations?)
- no overfitting (!)
- no preprocessing necessary
- very easy to implement (**27** lines of MATLAB code !!!)

Result: a triple winner!

Benelearn99:

1) Boosting with pre-processed data (129)

2) Boosting with original data (124)

3) Rough Data Models (118)

Others: 62-112 hits

Baselines:

- random selection: about 48 hits
- “straightforward approach”: around 96 hits

Update 2019

My implementation of boosting contained “critical conceptual errors”!

(a wrong weight-update mechanism)

Main conclusion:

Always start with a good “model evaluation scheme”!

State-of-the-art: XGBoost

- Many variants of boosting (10++)
- Friedman (2001): Gradient Boosted Machines

  https://www.jstor.org/stable/pdf/2699986.pdf

- **The best implementation: XGBoost**

  https://xgboost.readthedocs.io/en/stable/tutorials/model.html

Homework:

Study the XGBoost site: you will need it for A3!