DM825

Introduction to Machine Learning

Lecture 10

Application Guidelines

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

A few useful things to know in Application settings

- It is generalization that counts
  $\rightsquigarrow$ train on training set and evaluate on test set

- data alone is not enough: learners must embed some form of knowledge or assumptions (ie. no free lunch theorem)

  eg: learning a Boolean function 2^{100} from 10^6 examples

  nearest neighbor (instance-based) if good knowledge about what makes similar examples
  graphical models if good knowledge about dependencies of variables
  rule systems if knowledge of the type IF... THEN...

- Overfitting has many faces:
  - Bias: tendency to consistently learn the same wrong thing
    (if true releationship is not linear then we will see error even with infinite training set)
  - Variance: tendency to learn random things irrespective of real signal.
    (we are fitting on a too small training set that does not reflect the true relationship)

In general: if high training error and high test error $\rightsquigarrow$ bias
if low training error but high test error $\rightsquigarrow$ high variance

linear learners have high bias, decision trees high variance

be skeptical of claims that a particular technique "solves" the overfitting problem. It is easy to avoid overfitting (variance) by falling into the opposite error of underfitting (bias).

$\rightsquigarrow$ Diagnostics

- intuition Fails in high Dimensions
  Generalizing correctly becomes exponentially harder as the dimensionality (number of features) of the examples grows, because a fixed-size training set covers a dwindling fraction of the input space.

  Similarity reasoning breaks down in high dimensions:
  eg: nearest neighbor classifier with Hamming distance. Suppose the class is just $x_1 \wedge x_2$. If no other features, easy. But if there are 98 irrelevant features the noise from them leads to random predictions.

- Theoretical Guarantees are not What they seem

  Example of results:

  1. bound on the number of examples needed to ensure good generalization
  2. given infinite data, the learner is guaranteed to output the correct classifier

Example for 1.:

- $\mathcal{H}$ hypothesis space
- Let's say a classifier is bad if its true error rate is greater than ϵ. Probability that a bad classifier is consistent with n random, independent training examples is less than $(1-\epsilon)^n$.
- union bound: Let $A_1, A_2, \ldots, A_k$, be k different independent events. Then

$$P(A_1 \cup A_2 \cup \ldots \cup A_k) \leq P(A_1) + P(A_2) + \ldots + P(A_k)$$

- If b bad classifiers in $\mathcal{H}$ the probability that at least one is consistent is less than $b(1-\epsilon)^n$ (union bound)
- If the learner always returns a consistent classifier, the probability that this classifier is bad is then less than $b(1-\epsilon)n \leq |\mathcal{H}|(1-\epsilon)n$
- So if we want this probability to be less than δ, it suffices to make $n > \ln(\delta/|\mathcal{H}|)/\ln(1-\epsilon) \geq 1/\epsilon(\ln|\mathcal{H}| + \ln 1/\delta)$
- but most interesting hypothesis spaces are doubly exponential in the number of features d: eg Boolean functions of d Boolean variables.

- 2)

  main role of theoretical guarantees is not as a criterion for practical decisions, but as a source of understanding and for algorithm design.

- Feature engineering is the Key

  most important factor is the features used. Learning is easy with many independent features that each correlate well with the class.

  not a one-shot process of building a data-set and running a learner, but rather an iterative process of running the learner, analyzing the results, modifying the data and/or the learner, and repeating.

  automated feature engineering:
  forward search, backward search, filter feature selection

Automated Feature Selection

With number features $n \gg m$ size of training set, potential risk for overfitting.

With n features there are 2^n possible sub-sets. NP-hard problem.
Wrapper model feature selection: Heuristic to search among the 2^n subsets:

Forward search

start with $\mathcal{F} = \emptyset$;
repeat
  for $i = 1 \dots n$ **do**
    try adding feature i from $\mathcal{F}$;
    estimate generalization error using CV
  set $\mathcal{F} = \mathcal{F} \cup$ {best feature found in For loop}
until $|\mathcal{F}| \leq n' \leq n$;
output the best subset seen $\mathcal{F}^*$

Backward search

start with $\mathcal{F} = \{1 \dots n\}$;
repeat
  for $i = 1 \dots n$ **do**
    try removing feature i to $\mathcal{F}$;
    estimate generalization error using CV
  set $\mathcal{F} = \mathcal{F} \setminus$ {best feature found in For loop}
until $|\mathcal{F}| \geq n' \geq 0$;
output the best subset seen $\mathcal{F}^*$

Note that in statistical packages improvements are often assessed by the F-ratio of residual sum of squares.

Wrapper model feature selection require $O(n^2)$ calls to the learning algorithm.
Computationally cheaper methods:

Filter feature selection
for each feature i **do**
  compute some score $S(i)$ of how informative x_i is about y
select features with k highest scores.

Example of scores:

- absolute correlation between x_i and y on training data
- mutual information
  $MI(x_i, y) = \sum_{x_i \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_i, y) \log \frac{p(x_i,y)}{p(x_i)p(y)}$
  (for binary features with probabilities estimated from training set)
  (Kullback-Leibler divergence measures how different probability distributions $p(x_i, y)$ and $p(x_i)p(y)$ are)

- more Data Beats a cleverer algorithm
  try the simplest learners first (eg, näive Bayes before logistic regression, k-nearest neighbor before support vector machines). More sophisticated learners are usually harder to use, because they have more knobs to turn

- Learn many models, not Just one
  best learner varies from application to application

  model ensembles:
  bagging: generate random variations of the training set by resampling, learn a classifier on each, and combine the results by voting.
  boosting: training examples have weights, and these are varied so that each new classifier focuses on the examples the previous ones tended to get wrong
  stacking: outputs of individual classifiers become the inputs of a "higher-level" learner that figures out how best to combine them

Combining Models

1) **Bagging or bootstrap aggregation**:
$\{(x^1, y^1), (x^2, y^2), \ldots, (x^m, y^m)\}$, goal is predicting y
We bootstrap B times and each time fit a model $f_b(x)$ then:

$$\hat{y} = f_{\text{BAG}}(\vec{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(\vec{x})$$

Motivation:
Suppose $h(\vec{x})$ is the true function: $f_b(\vec{x}) = h(\vec{x}) + \epsilon_b(\vec{x})$

$$E_{\vec{x}}[\{f_b(\vec{x}) - h(\vec{x})\}^2] = E_{\vec{x}}[\{\epsilon_b(\vec{x})\}^2] \qquad \text{sum of squares}$$

Hence expected error on the sample B:

$$E_{AV} = \frac{1}{B}\sum_{b=1}^{B} E_{\vec{x}}[\epsilon_b(\vec{x})^2] \qquad \text{for each model acting individually}$$

$$E_{\text{COM}} = E_{\vec{x}}\left[\left\{\left(\frac{1}{B}\sum_{b=1}^{B} f_b(\vec{x})\right) - h(\vec{x})\right\}^2\right] = E_{\vec{x}}\left[\left\{\frac{1}{B}\sum_{b=1}^{B} \epsilon_b(\vec{x})\right\}^2\right]$$

If we assume:

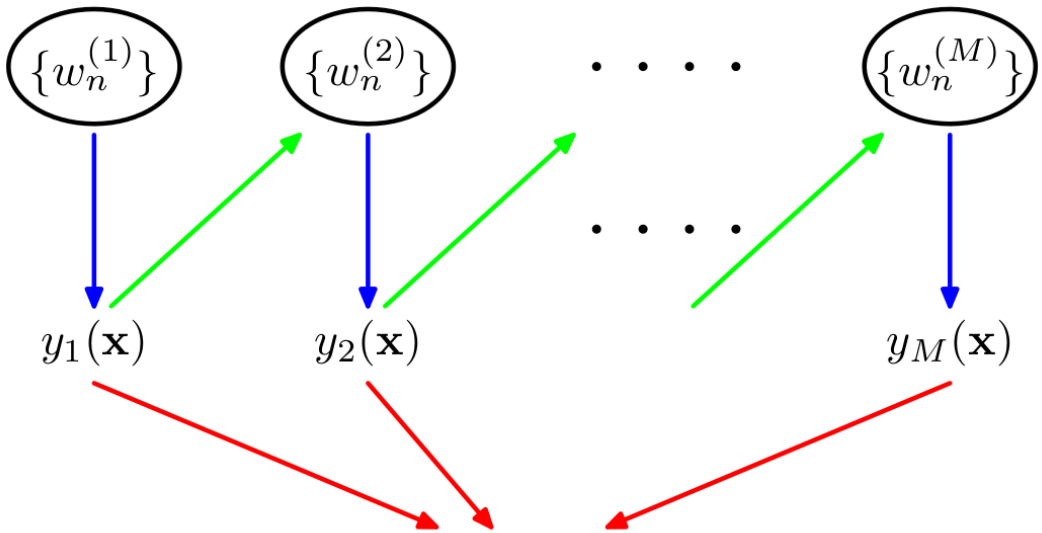
$$E_{\vec{x}}[\epsilon_b(\vec{x})] = 0$$
$$E_{\vec{x}}[\epsilon_b(\vec{x})\epsilon_l(\vec{x})] = 0 \qquad b \neq l$$

Then

$$E_{BAG} = \frac{1}{B} E_{\mathrm{AV}}$$

The independence assumption is clearly not true but it works nevertheless.

2) **Boosting**



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Adaptive Boosting (AdaBoost) (for both classification and regression)
training in sequence + weighted majority voting scheme

$\{x_1, \ldots, \vec{x}_m\}, y_i \in \{-1, 1\}$

Set weights $w_i^1 = 1/B, i = 1, \ldots, m$

for $b = 1 \ldots B$ **do**

Fit model by weighted error function:

$$J_b = \sum_{i=1}^{m} w_i^b I\left\{h_b(\vec{x}^i) \neq y^i\right\}$$

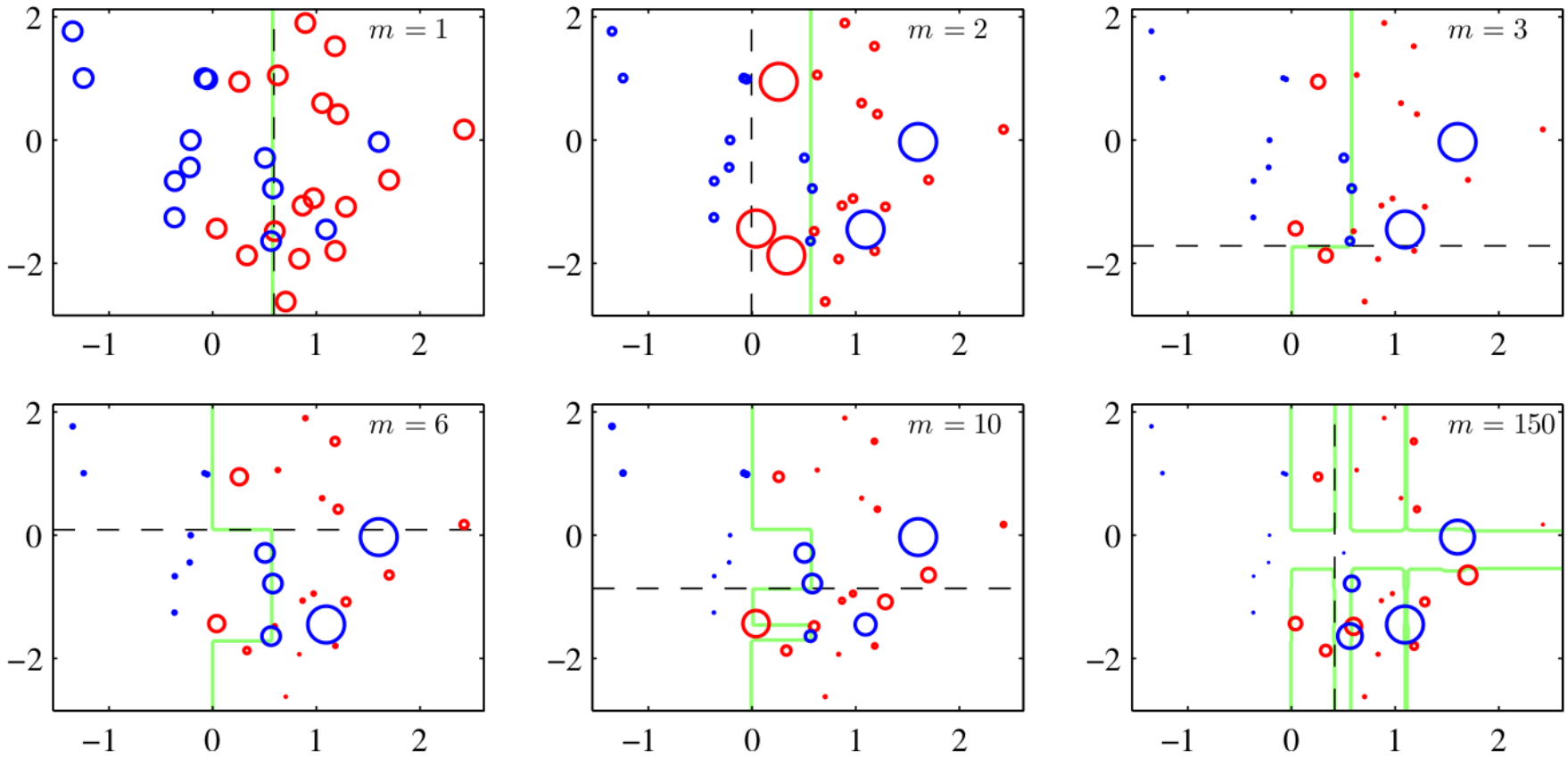
Update weights:

$$\epsilon_b = \frac{\sum_{i=1}^{m} w_i^b I\left\{h_b(\vec{x}^i) \neq y^i\right\}}{\sum_i w_i^b} \qquad \alpha_b = \ln \frac{1 - \epsilon_b}{\epsilon_b}$$

$$w_i^{b+1} = w_i^b \exp\{\alpha_b I\{h_b(\vec{x}) \neq y^i\}\}, \qquad i = 1, \ldots, m$$

Predict with $\operatorname{sign}\left(\sum_{b=1}^{B} \alpha_b h_b(\vec{x})\right)$

Example: Decision Stumps (single node decision tree)



- Simplicity Does not imply Accuracy
  given two classifiers with the same training error, the simpler of the two will likely have the lowest test error. But there are many counter examples to this in the literature.

- Representable Does not imply Learnable
  Eg, functions with local optima

- Correlation Does not imply Causation

  learning is usually applied to observational data, where the predictive variables are not under the control of the learner, as opposed to experimental data, where they are.

  we would like to predict the effects of our actions, not just correlations between observable variables. If one can obtain experimental data (eg, by randomly assigning visitors to different versions of a Web site), then one should do so.