Advanced course in machine learning
582744
Lecture 9

Arto Klami

Sir David MacKay (22 April 1967 - 14 April 2016)
[Photo: Graham Turner]
`http://www.inference.phy.cam.ac.uk/mackay/itila/book.html`
`http://withouthorair.com`

Outline

Nonlinear supervised models

The kernel methods were based on the idea of mapping the features through a nonlinear function and then applying a linear model

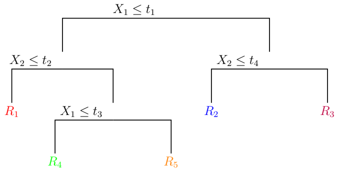
The final model was
$f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_n \alpha_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}) = \sum_n \alpha_n k(\mathbf{x}_n, \mathbf{x})$

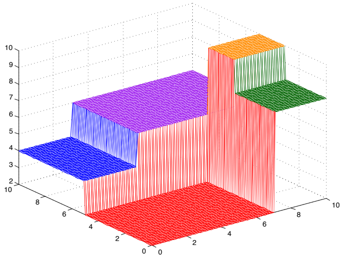
Adaptive basis function models assume

$$f(\mathbf{x}) = b + \sum_m w_m \phi_m(\mathbf{x})$$

for M basis functions $\phi_m(\cdot)$ that are learnt from the data

Classification and regression trees





A binary tree where each node splits the data into two halves according to one feature – these are called *axis parallel splits*

CART for classification and regression trees

Classification and regression trees

The tree can represent arbirarily complex patterns despite very simple decisions. As adaptive basis function model it is

$$f(\mathbf{x}) = \sum_m w_m \mathbb{I}(\mathbf{x} \in R_m)$$

where w_m is the mean prediction (scalar for regression, class probabilities for multi-class classification) and R_m is defined by the splits

Learning this kind of models is rather different from the stuff we have seen thus far – forget the gradients for a while

Growing a tree

Finding the optimal partitioning is NP-complete. In practice the most well known variants (CART, C4.5, ID3) use greedy algorithms

Recursively pick a node and threshold t so that the sum of the losses for the new leaves is smallest

The loss is simply squared error for regression and misclassification for classification (or some other related losses)

Possible thresholds (for scalar regression) can be enumerated; just sort the unique values

Stop when the loss does not increase a lot, some maximum depth is reached, the leaves are sufficiently homogeneous, or whenever you want to...

Trees and overfitting

If we continue until all elements are in their own leaves, the tree will definitely overfit – it is an extremely flexible classifier we just trained to reach zero training loss

The stopping heuristics mentioned on the previous slide are approximations for early stopping

A more common approach is to grow a full tree and prune it; this allows avoiding some local optima

Remove nodes using cross-validation error, pick a smallest tree that is within (for example) one standard deviation of the full tree

Example CART method: ID3

- Start with all data in a root node
- Calculate information gain

$$H(S) - p_1 H(S_1) - p_2 H(S_2)$$

  for each variable, where $H(S)$ is the entropy of the classes for data set S and $S = S_1 \cup S_2$
- Split according to the variable with the highest information gain
- Recursively repeat the above, excluding variables already used higher up in the tree
- Stop if: All samples are in the same class or all variables have been used

Decision trees - pros and cons

- Fast (especially when applying), robust to errors in training data
- (Kind of) easy to interpret
- Direct multi-class support, work for mixed inputs
- Invariant to monotonous data transformations
- The greedy process does not guarantee high accuracy
- Unstable: Small changes in the input data might result in very different data (which means the interpretations are fragile)

Ensemble methods

Ensemble methods refer to methods that combine predictions of several supervised methods: $f(y|\mathbf{x}) = \sum_m w_m f_m(y|\mathbf{x})$

The predictions of ensemble methods can be seen as voting; each model votes for their output – sometimes also called *committee methods*

Ensembles typically work better if the base models are more versatile – either completely different algorithms or high-variance solutions of one algorithm

Bayesian model averaging would use the posterior probabilities of the models as weights, but is conceptually very different

Bagging

Bagging (bootstrap aggregating) is one of the simplest ensemble methods, often used together with decision trees

Learn M trees for bootstrapped versions of the data (pick N samples with replacement) and average the predictions with unit weights

$$f(\mathbf{x}) = \frac{1}{M} \sum_m f(\mathbf{x}|\theta(\mathbf{X}_m))$$

Decreases the variance of the prediction without changing the bias

Helps to a degree, but the trees are often highly correlated (because the input data are so similar)

Random forests

Random forests improve on bagging by learning individual trees on randomly chosen subset of the data as well as randomly chosen subset of the features

The idea is to make the models in the ensemble different from each other, so that hopefully some tree solves problems that are hard for others

Each tree is sparse (wrt to the original features), which should help avoiding overfitting

Since we only used subset of samples to train each tree, we can also estimate the validation error in some way (out-of-bag error)

Random forests are often very accurate, but we cannot interpret them as easily as individual trees

Boosting

The strength of random forests came from combining multiple models that are not so accurate (remember we used randomly sampled features and data points)

Boosting goes even further along these lines, formalizing a procedure of how a collection of *weak learners* can be converted into a strong classifier

A weak learner is any classifier that is at least slightly better than random: The empirical misclassification rate is at most $0.5 - \gamma$

Boosting – idea

Imagine we learn one classifier that works correctly for most samples but incorrectly for a few

What if we now take the samples for which the model was wrong and train another classifier for those alone? It probably classifies some of them correctly

Repeat this several times and create an ensemble of the classifiers

Boosting formulates this intuitive idea, so that we are guaranteed to reach zero empirical loss

Boosting

Each *base learner* is some adative basis function model, often a decision tree or a decision stump (tree with just one node)

We assume the base learners are better than random, but not necessarily by much (that is, they can be weak learners)

We learn M learners sequentially, training each one on a data set with weighted samples (without re-visiting the old models!)

The weights are smaller for samples classified well by other learners; the exact weights depend on the algorithm and the loss

The final ensemble weights the base learners based on their relative accuracies

Adaboost

The final classifier is given by

$$F_M(\mathbf{x}) = \sum_{m=1}^{M} \beta_m f_m(\mathbf{x}),$$

a weighted sum of the base classifiers that output -1 or 1

The decision is made by the sign of $F_M(\mathbf{x})$

Given that we have already learnt F_{m-1}, we want to find the best new classifier $f_m(\mathbf{x})$ and its weight β_m

Note that this is not an iterative algorithm; we do not return to the earlier ones but simply learn each base learner once

AdaBoost

Adaboost uses the exponential loss

$$L = \sum_n e^{-y_n F_m(\mathbf{x}_n)},$$

which as a function of the *m*th classifier is

$$L = \sum_n e^{-y_n F_{m-1}(\mathbf{x}_n)} e^{-y_n \beta_m f_m(\mathbf{x}_n)} = \sum_n w_n e^{-y_n \beta_m f_m(\mathbf{x}_n)}$$

We can directly optimize for the *m*th classifier as if it was the first one if we weight the samples by

$$w_n = e^{-y_n F_{m-1}(\mathbf{x}_n)}$$

...which tends to zero for correctly classified samples and to infinity for incorrectly classified ones

AdaBoost

Further simplification of $\sum_n w_n e^{-y_n \beta_m f_m(\mathbf{x}_n)}$ gives

$$\begin{aligned} L &= e^{-\beta_m} \sum_{y_n = f_m(\mathbf{x}_n)} w_n + e^{\beta_m} \sum_{y_n \neq f_m(\mathbf{x}_n)} w_n \\ &= (e^{\beta_m} - e^{-\beta_m}) \sum_n w_n \mathbb{I}(y_n \neq f_m(\mathbf{x}_n)) + e^{-\beta_m} \sum_n w_n \\ &= C \sum_n w_n \mathbb{I}(y_n \neq f_m(\mathbf{x}_n)) + D \end{aligned}$$

and hence we need to minimize the weighted classification error using any suitable algorithm for the base learner

The weight depends on the weighted error rate ϵ_m as

$$\beta_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m},$$

which is positive for all m and bigger for more accurate classifiers

AdaBoost

Easy to implement, no parameters besides the number of learners M (and the choice of the base learner)

Training loss decays exponentially in M, quickly converging to zero

Intuitively would overfit extremely, but in practice does not do so – we will get back to this

AdaBoost is, however, vulnerable to noisy labels since the weights are exponential and it overfits with too strong base learners

Other boosting techniques

- L2boost: Squared loss, next learner fitted to the residual
- LogitBoost: Log-loss, penalizes misclassification only linearly instead of exponentially but requires Newton updates
- Sparse boosting: Require base learners to use only one feature – close relationship with sparse regularizers for linear models

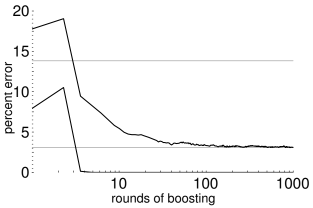
Also note the relationship with mixtures of supervised models:

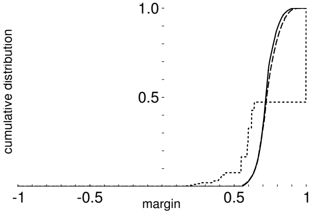
$$p(y|\mathbf{x}_n) = \sum_m \pi_m p(y|\mathbf{x}_n, \theta_m)$$

which would fit M learners using EM in a symmetric fashion, instead of sequentially learning them

Why is boosting good?

Boosting makes the empirical risk zero, but often does not overfit; see `http://rob.schapire.net/papers/explaining-adaboost.pdf`





AdaBoost maximizes a margin similar to SVM

Can be viewed as approximation to l_1 regularization

Re-cap

- Linear and generalized linear models; fast and easy, gradient-based optimization
- Sparsity and regularization; crucial for high dimensionality, often in practice by l_1 regularization
- Kernel-methods: Represent $\mathbf{w} = \sum_n \alpha_n \phi(\mathbf{x}_n)$ and operate with the inner products $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ stored as a kernel
- Decision trees: Fast algorithms that can reach zero empirical loss

Ensemble methods in principle applicable for all, but often used together with decision trees; they need fast algorithms and can tolerate weak accuracy

So which one should we pick?

No free lunch -theorem (Wolpert, 1996):
No algorithm is better than any other for minimizing the expected risk for classification over all possible tasks

“All possible tasks” includes patological cases designed to be bad for given algorithms – when solving practical problems we expect to see somehow regular data sets

No universally best learning method, but for practical tasks some methods are better than others

Large-scale comparisons

By comparing algorithms on multiple data sets we can analyze the average behavior:

- Caruna and Niculescu-Mizil (2006): Boosted decision trees, random forests, bagged decision trees, SMVs and neural networks outperform logistic regression, naive Bayes, kNN and individual decision trees (discussed in more detail in Section 16.7)
- Fernandez-Delgado et al. (2014) evaluated 179 classifiers on 121 data sets: Parallel random forests the best, on average reaching 94% of the best accuracy; SVM with Gaussian kernels reaches 92%; C5.0 decision trees and MLPs also near the top

Crudely: Non-linearity is needed in general cases, some sort of max-margin ideas or ensembles are often good