Alternating decision trees in predicting short-term weather

Matti Palomäki

August 30, 2016

1 Introduction and research problem

This is an account of a practical machine learning project that investigated the use of alternating decision trees (ADTs) in predicting rain from open meteorological data.

To put it simply, the question to be investigated was:
How well does alternating decision tree do as a classifier in predicting short-term rain conditions from meteorological observations?

Prior to trying out this learning method in this context the fit seemed good, at least in principle. Similar to medical diagnostics (see for example [3]), where the input data can be seen as having some fairly simple "symptoms" that a decision tree -type of model models well, the meteorological data has some "symptoms" of rain: eg. high blood pressure is a symptom for heart disease, low air pressure is an indicator for impending rain. In this sort of expert systems for medical diagnostics decision trees have done well, so it seemed fair to hope that a decision tree could model rain reasonably well at the very least.

2 The method:

This project divided roughly into two parts. That is, there's firstly the necessary implementation and experimentation required for setting up the apparatus for the actual investigation of the research problem, and then secondly there's the actual obtaining and analysing the data that goes towards answering the research problem.

2.1 Setting up the implementation

For the implementation part of the project, three main resources were needed:

1. an implementation of the alternating decision tree model,
2. meteorological data sets to be used and
3. the resources for running the the complete system and providing the interfaces between the different parts of the project.

2.1.1 Alternating decision tree (ADT)

Alternating decision tree is a kind of machine learning classification algorithm that in a sense combines *boosting* (a sophisticated form of ensemble learning, [4]) to simpler *classification and regression trees (CART)* (also commonly known as *decision trees*) ([2] p. 546-554). As such, ADT resembles the method of using decision stumps as base learners in ensemble learning, but in an ADT the whole classifier consists of a single tree, and is thus far easier to visualize and easier for a human to grasp.

The tree consists of two types of nodes that alternate: prediction nodes that hold a prediction value, and splitter nodes that hold a branching condition for some subtrees. Note that unlike in an ordinary decision trees, in an ADT all the branches for which the splitter condition is satisfied are traversed. The classification doesn't rest on which leaf of the tree gets traverses to, and indeed multiple leaves of the tree might be reached - or even none of them. Given a data point, the classification output of the tree is the sign of the sum of the prediction values of all the prediction nodes that are traversed to.

Training an ADT uses boosting and is naturally sequential: Initially all the training data has uniform weights. New splitter node and the two prediction nodes attached to it are added at each step so that the information gain they present is maximized (in some implementation specific sense). After creating the new nodes, the weights of the training data are updated so that the data points that are classified wrong by the new nodes increase in weight, whereas the weights of the ones correctly classified diminish.

Following Schapire & Freund [4] I used the AdaBoost update for the weights of the training data: when $D_t(i)$ is the weight of a data point i at step t,

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot e^{-\alpha_t h_t(i)}$$

for all the training data points, indexed with i. Here

- $h_t(i)$ is the prediction of the branch predictor (the predictor that consists of the path through the tree to this splitter node, the splitter node and the following two prediction nodes ([4] p.292)) that terminates at the latest added splitter and prediction nodes
- $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ where
- ϵ_t is the weighted classification error of this branch predictor to the total sum of weights in the branch.

See [4], pages 290-297, or [1] for a more thorough presentation on alternating decision trees.

2.1.2 Open data and application hosting

As for the data sets needed for the project, the Finnish Meteorological Institute has opened to the public a large variety of the data they routinely collect. This data can be queried over the internet and is received in the form of machine-readable XML documents.

This is a very good fit for the kind of project that is being discussed here, as there is an abundance of data pertinent to the subject are readily available

over the internet, and already formatted so that its use in an automated system requires very little extra processing.

A server application was written to act as an interface between the ADT-model, the open data source of the Finnish Meteorological Institute and the potential users of the predictor (users can access the system over the internet with a browser). This server application was also written in C++, utilizing languages internet-oriented Poco libraries (freely obtainable at http://pocoproject.org/).

This server application was hosted on the OpenShift hosting service, this service being one of the relatively few that offer free hosting of small scale projects written in C++.
I acquired an API-key for the FMI open data sets and stored it in the server file system. From there it could be accessed by the server application and hence could be used as an authentication in the data queries made by the application to the FIM servers.

The system was set up so that a user can

- initiate loading fresh data from FMI. The data loaded is the meteorological data for the previous week, and the data from the corresponding dates from the last six years, twelve observations per day
- initiate learning an ADT from the latest set of meteorological data loaded. The variables loaded are rain/10 minutes, temperature, wind speed, wind direction, gust speed, air pressure, humidity, dew point, range of vision, cloud cover and the dominant weather type
- ask for a prediction of rain/no rain for the next two hours; the latest meteorological observations needed for ADT to make the prediction are loaded from FMI.

The reasoning behind this choice of data to be loaded was, that meteorological variables change according to the time of the year, and this way the ADT is always taught with data that is current in that respect.
An alternative approach would be making the time of the year one of the parameters for the model. I preferred this way that produces current and simpler model each time one is needed to learning a more complex ADT that would cover all the months of a year.

All the code for these implementations can be found from

http://www.github.com/vuolankoinen/ml_cpp

and the hosted project, for the time being, can be viewed and used at

http://ml-vuolankoinen.rhcloud.com/

2.2 Obtaining results with the implementation

In order to avoid overfitting, early stopping was used. This seemed significantly more practical than performing, say, time and computing cycles consuming leave-one-out cross validation ([5] p. 31) each time a user accessing the application initiates the training process of an ADT using that moments meteorological data.

The assumption here was, that the weather conditions that precede rain do not greatly vary over the summer months, and hence the model needed to predict rain does not significantly change in complexity. While the basic complexity of the tree remains uniform, for each time the exact splitting conditions etc. can be learned specifically to match that time of the year. Hence the number of splitter nodes that gives a justifiable stopping condition for one case should also serve well as the stopping condition for other data sets of the same kind.

This assumption might not hold up under closer study, or at least it might prove not to be the best way to set thinga up. Nonetheless it served as a somewhat justifiable starting point and reasonable within the scope of this project.

A reasonable criterion for early stopping was therefore obtained by performing leave-one-out cross validation on a set that was pseudorandomly-chosen from among the numerous data sets available. As the minimal validation error was reached when the ADT was branched until 4 splitter nodes were added to the tree, this number of splitter nodes was used by the system as the stopping criterion for learning in any given case.

This approach to model selection and avoiding overfitting, using one case to perform the model selection, is similar to one presented in [2] p. 559.

The data that was used for model selection was the data set that comprised the meteorological data from FIM ranging between 11 am on 5.7. and 11 am on 12.7. in the years from 2o11 to 2016. It is interesting to note, that in this period the cases of "no rain" dominate the cases of "rain" with proportion of about 15:1. Learning an ADT from this data resulted in the minimum validation error (with 1-0 loss [2] p. 179 - exponential loss could and maybe should be also be considered) when the number of splitter nodes learned for the alternating decision tree was 4. (see figure 1)

This low number of branches seems suspect: even with just a few variables with predictive importance one would expect somewhat more complex tree to capture the phenomenon better.
There are a few plausible explanations. The most likely one is that I was hoping for a too neat a fit between the model I had chosen and the data that it was fed with. There might be some unsuitability of the chosen approach to the raw parameters used in training the algorithm - at the level of single parameters, there are only a few with predicting power, and the model is too crude to pick up on more subtle patterns of the interplay of the parameters in the data.
However, there was no time for thorough re-examination of this result. See the next chapter to find out why.

2.3 Practical problems encountered

Troubleshooting a system that is hosted on a third-party server involves nuances I was not familiar with before undertaking this project. Especially troublesome was figuring out where to look for source of errors once the system was running without interruptions and responding without any errors logged, but when there nonetheless was a suspicion of an erroneous behaviour, this suspicion raised by unlikely-looking predictions in certain scenarios. Narrowing down the possibilities for the source of an error that might or might not exist was not trivial when the potential origins of the error were so many: the interface between the data source and the system, errors in the data received, the implementation of the ADT model, training class of the ADT model, interface between the data and

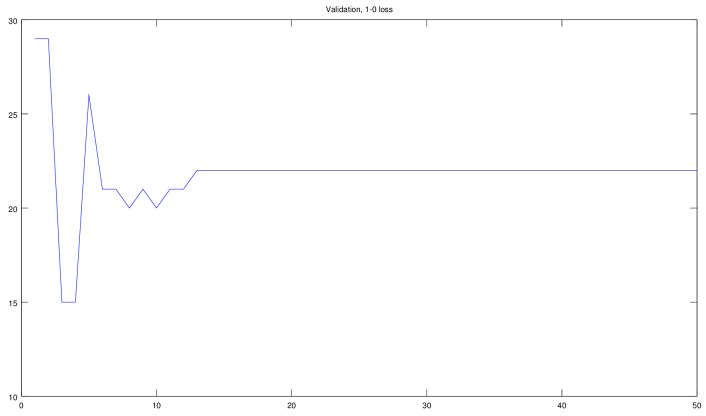

Figure 1: Validation errors for different number of ADT branches

the training model class, interface between the predicting class and the user and specifics of the hosting system, including the environmental variables and file handling.

Other practical problem was posed by the data source that was used, i.e. the open data from the Finnish Institute of Meteorology. FIM open data is supposedly in machine-readable format, but nonetheless the data that was received from HTML-queries to the FIM server occasionally resulted in broken lines in the parsed data files. Identifying this problem and writing extra clean-up routines for ensuring fixed-format input data for the training and prediction algorithms required some extra work.

In using precipitation data from several years, an interesting annoyance was cause by changing standards over the years. Namely, before 2013 the case of "no rain" was presented in the data as "-1.0 mm" of rain, whereas later this has been changed into a more sensible "0.0 mm" of rain. For a learning algorithm data with both of these conventions being used in it has an unnecessary source of confusion.

Also there was an unfortunate case of forgetting to pay attention to the possibility of missing data entries, which also resulted in some hard to catch erroneous behaviour in the system. This type of error was thoughtfully eliminated from earlier versions of the system. However, expanding the program to include more functionality, once it was already functioning, introduced a very situational bug of this origin. Since this type of problem was supposedly already handled since the first lines of code were written, such an error was tricky to catch.

This type of cases where a certain error needed to be singled out and fixed were of course preceded by the stage of more broad problems concerning the use of the resources that were chosen for the project. The documentation and case examples for the FMI open data, OpenShift C++ hosting and Poco libraries were all freely available online, but also somewhat incoherent, spread over multiple sources and in places out of date. For example, for OpenShift, the C++ hosting seems to be a fairly marginal service, as there was no activity from the

company on the services documentation over the last few years and the original project example was in no way practical. Getting the project hosted and running involved a bit of both detective work and tinkering with the script files that were found from the hosting service.

All these difficulties laid out by the realities of the real life in combination meant that the technical side of setting up and troubleshooting the system required significantly larger portion of the time reserved for the project than was originally envisioned. The implementation part of the project dominated the analysis part.

3 Results

Figure 2 details the structure of the alternating decision tree that was trained with the training data of one of the actual test cases, predicting rain in Helsinki at 3 am on 30th of June. The ease of presenting the learned model in such a diagram underlines one of the main strengths of using the alternating decision tree model: The model is easy for a human to read, understand and even use without the aid of a computer, if need be.

Interpreting the contents of the nodes in the ADT depicted in figure 2 lead to some reasonable-sounding observations. From the left branch we see, that if it is not raining right now, quite likely it still won't be raining soon either. From the right branch we find a few points. Firstly, air pressure, which is commonly known to correlate with rain or lack of thereof, makes an appearance in the ADT, as was to be expected - high pressure means most likely no rain. Secondly, even if the pressure's low, it's unlikely to rain if the sky is clear. Thirdly, it's more likely to rain when it's windy.
An interesting point concerning the whole of the tree is that it is impossible for a collection of prediction nodes to sum up to a positive value - i.e. this particular model never predicts "rain", but always "no rain", just with varying level of confidence. Apparently rain is rare enough an event in the training data that was used in training this tree that the safest course of action is to never predict it.

4 Discussion

In undertaking such a project again, experiences of this project would lead me into reserving more time for dealing with practical questions that fall under the category of nuisances. This sort of advice is commonly given in the project management context: little things do take longer than your think - this I would now like to echo.
Also it is to be noted, that the goals of this project would have been better served with more purposeful time management. Better and more conscious prioritizing of tasks would have been helpful. Especially, being willing to skip on from a given stage in the interest of having enough time for the next one, and the one after - even when there were things to fix and polish - would have been useful.

It must be said that the performance of ADT on this application area probably would improve if the data was pre-processed prior learning the model.

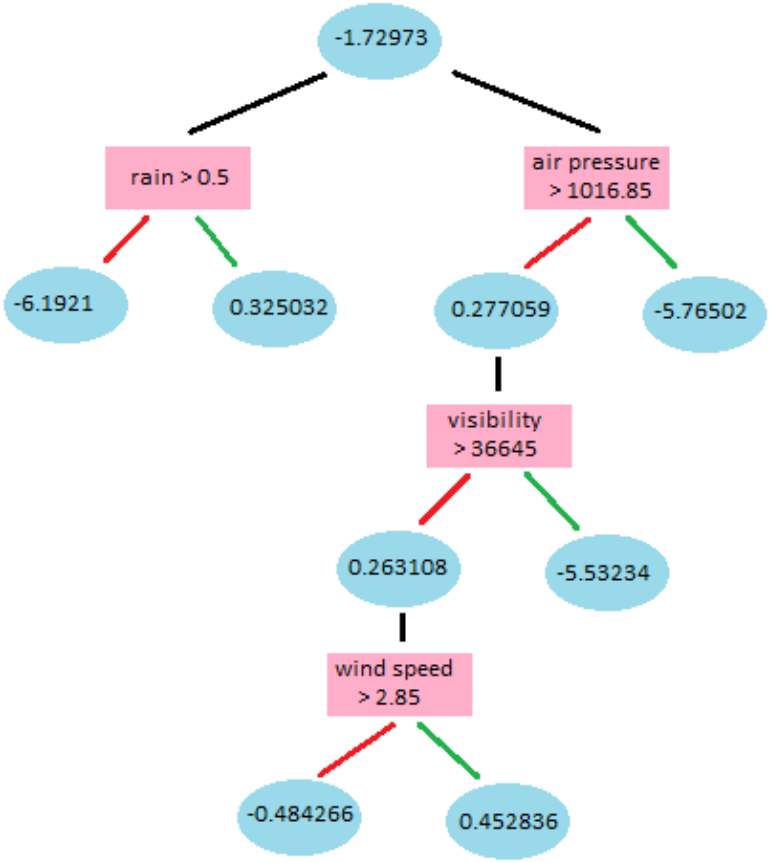

Figure 2: Structure of an ADT learned from the FMI data

Using raw data easily results in many nearly identical branches in the trained tree when there are correlating variables in the input data. This makes the tree that is learned from the data unnecessarily complex and inefficient, as the tree ends up with unrequired splitting nodes followed by near-identical sub-trees on both sides. This could be avoided by the use of latent variables in the learning, these latent variables chosen in the pre-prosessing stage.

The version of ADT implemented here uses the exponential loss, which is known to lead to systems over-emphasizing outliers. One reasonable addition to this project would be to compare the results obtained with this implementation to those that similar system using more outlier-resistant log-loss would give, over a large number of samples.

It is not easy to assess the performance of ADT model recounted here in a sensible way. Weather forecasting is very important area of study with serious practical implications for the accuracy of the methods used, and therefore it's subject to a lot of research. As a result, the models in most of this research are far beyond the simple machine learning algorithm implemented here. There aren't really comparable fairly low-level methods to set the benchmark.

Comparing to the "dummy algorithm" of just predicting what one sees from ones window, however, ADT does fairly well. So, if one has resources to use an ADT but no access to sophisticated weather forecasts, an ADT might be useful. (Say, when you have no internet but happen to be carrying a printed ADT on your person. It could happen.)

References

[1] Yoav Freund. The alternating decision tree learning algorithm. In *In Machine Learning: Proceedings of the Sixteenth International Conference*, pages 124–133. Morgan Kaufmann, 1999.

[2] Kevin P. Murphy. *Machine learning : a probabilistic perspective.* MIT Press,, Cambridge, MA :, cop. 2012.

[3] S. Palaniappan and R. Awang. Intelligent heart disease prediction system using data mining techniques. In *2008 IEEE/ACS International Conference on Computer Systems and Applications*, pages 108–115, March 2008.

[4] Robert E. Schapire. *Boosting : foundations and algorithms.* MIT Press,, Cambridge, MA :, cop. 2012.

[5] Mark Girolami & Simon Rogers. *A first course in machine learning.* CRC Press,, Boca Raton, Fla. :, cop. 2012.