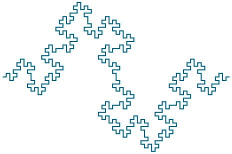
HEP, HEP, Hooray

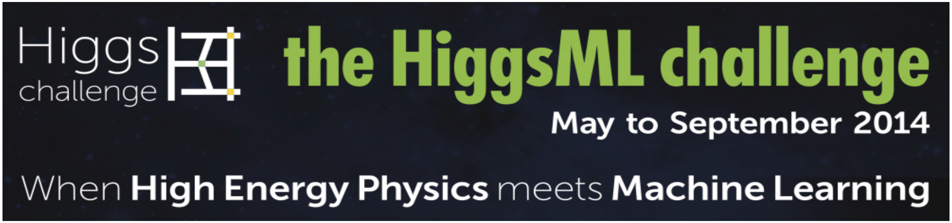
Gábor Melis

@GaborMelis | http://quotenil.com | mega@retes.hu

Franz Inc., Fixnum Services



2015 January 14


Higgs challenge

the HiggsML challenge

May to September 2014

When **High Energy Physics** meets **Machine Learning**

- Largest Kaggle competition to date.
- 4 months. Ouch.
- Classification task: separate tau-tau signal from background on simulated data (4-momenta of particles, some derived features).
- Unusual evaluation metric.
- Training set: 250000 labeled, weighted collision events
- Test set: 550000 events with unknown labels and weights.

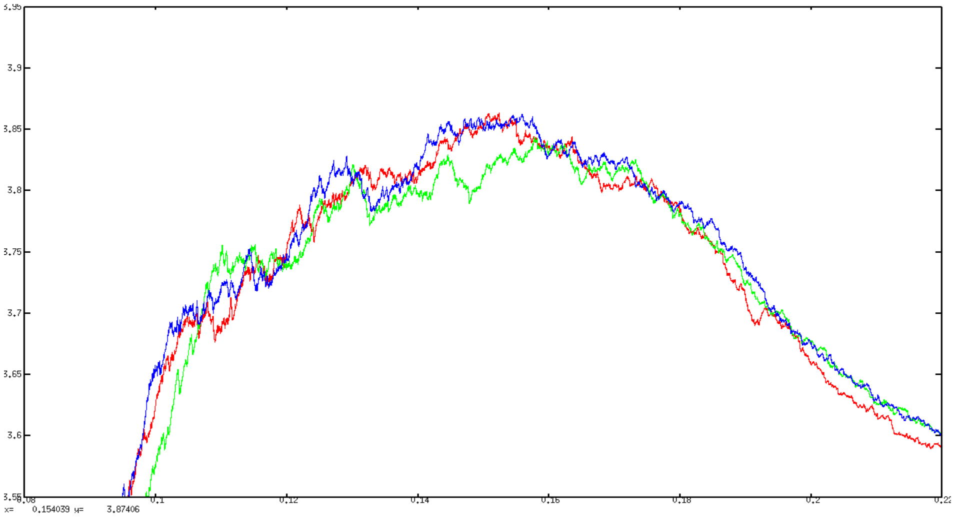
THE TASK

- Binary classification problem with an important twist.
- Classes: Signal vs Background.
- Must select a subset of examples ...
- in order to maximize $AMS = s/\sqrt{b}$ where s is the true positive count and b is the false positive count in that subset.

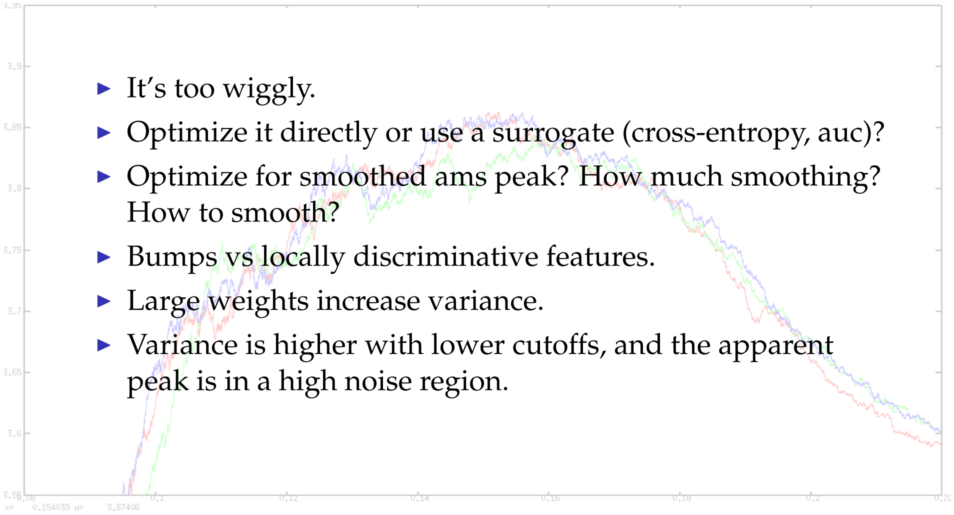
THE BASIC APPROACH

- Train a classifier that assigns scores (probabilities) to examples.
- Sort the examples by their scores.
- Take the examples with scores above a certain threshold.

PROBLEMS WITH AMS

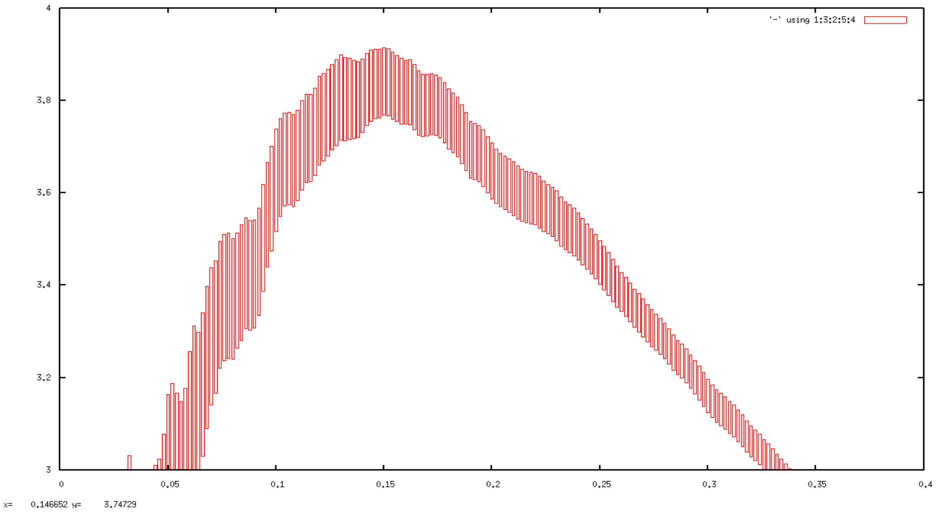


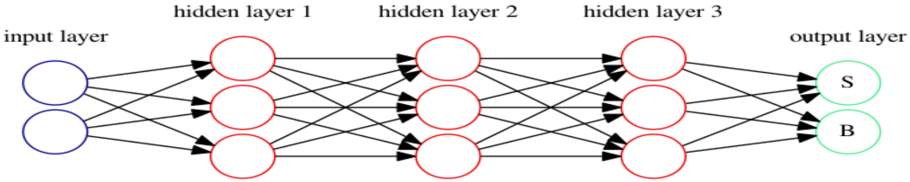
Problems with AMS



- It's too wiggly.
- Optimize it directly or use a surrogate (cross-entropy, auc)?
- Optimize for smoothed ams peak? How much smoothing? How to smooth?
- Bumps vs locally discriminative features.
- Large weights increase variance.
- Variance is higher with lower cutoffs, and the apparent peak is in a high noise region.

PROBLEMS WITH AMS





- inputs: normalized features (~30), some log transformed
- 3 hidden layers of 600 neurons each
- output layer: 2 softmax units (one for signal, one for background)
- activation function: “max channel” in groups of 3
- trained to minimize cross entropy
- regularization: dropout on hidden layers, L1 + L2 penalty and a mild sparsity constraint input weights

Neural Network Pros and Cons

+ Powerful: better by ~0.05 than Boosted Decision Trees (BDT, xgboost).
+ Your best bet to find interactions you don't know about.
- Somewhat sensitive to input encoding, normalization and also to correlated inputs (mitigated by bagging).
- Can take a long time to train.
- Lots of choices, parameters to tune.
- Trained model is difficult to interpret.

FEATURE ENGINEERING

- Log transform long tailed features, zero mean, stddev 1.
- Deep NNs benefit less from feature engineering.
- Mass MMC and other derived features still useful with NNs.
- Additional CAKE features seemed to contribute ~0.01 to BDTs, less clear with NNs.

CROSS-VALIDATION

- Public leaderboard dataset too small, single cutoff, very unreliable.
- CV showed huge fluctuations across folds (± 0.15) which made comparisons almost impossible and slowed progress down.
- Since the AMS is non-linear, it seemed better to merge predictions from different folds and then calculate the AMS instead of averaging the AMSs of different folds.
- Even better: repeat CV with different splits and average the CV scores.

BAGGING

- Crowded decision boundary: AMS is very sensitive to minor differences in predicted probabilities.
- Model averaging to the rescue.
- Bagging: average models of the same kind trained on random subsets of the training set.
- Improved scores by about 0.1.

CV Bagging

Repeated cross-validated bagging requires very many models to be trained (repetions x folds x bags). But there is a shortcut: one can get a more reliable CV score, a more accurate model *at the same time*, faster than with standard bagging by using the CV splits instead of random subsets of the training data.

```
for rep in [0..9]:
  shuffled = data.shuffle()
  for fold in [0..4]:
    [training, test] = shuffled.split_fold(fold, 5)
    train(newModel(), training, test)
    # Stash away the trained model (or only its
    # predictions). Merge and average predictions
    # of all models trained so far, calculate AMS.
```

ENSEMBLING

- CV bagged NNs: 3.83
- CV bagged xgboost: 3.79
- Weighted average of the above two: 3.84

OPTIMIZATION AS SEARCH

- Huge solution space.
- Scores (+ noise) are observable.
- Our goal is to maximize the score.
- A search is performed by wandering around in this space sampling solutions and observing scores.
- We have some prior knowledge about the surface (how smooth it is, where the hills are likely to be)

LEVELS OF SEARCH

Search is performed on two levels:

- Humans do global search by choosing *Algorithms*.
- Algorithms perform local search (“learning”) to produce a *Model*.

Meta Search

But these searches are only in the solution space. We want to search efficiently, so in parallel we must improve our tools and methodology. This will be called the “Process”.

STARTING OUT

Is it worth it to enter this contest?

- Estimate randomness in rankings.
- Understand the rules, understand AMS.
- Guess what kind of knowledge is necessary to win.
- Rewards?

THE PROCESS

When a *Process* is executed by a *Human* it produces an *Algorithm*. When an *Algorithm* is executed on a *Computer* it produces a *Model*.
Humans are programmed in a high-level, mostly declarative language:

```
The Prize isn't that high, you must leave
most of the work to the computer.

To win, don't use the same algorithm
as everyone else.
```

However, if the Human in the middle is capable of introspection, then he can adjust its own algorithm, the Process.

```
Change this Process if you must.
```

The First Try

So how can one perform global search in algorithm space automatically?

The First Try

So how can one perform global search in algorithm space automatically?
With Genetic Programming, for example. For that, we need only a few things:

- Be able to juggle trees of expressions made of *terminals* (here input features) and *operators*.
- and a *fitness function* to guide the search.

Genetic Programming

Tried:

- Raw and preprocessed input features.
- Various simple and domain specific operators.
- Many kinds of fitness functions (AMS, smooted AMS, AUC, cross-entropy, area under ams, etc).
- Penalizing solutions with high variance of fitness over different subsets of the training set.

Result?

Genetic Programming

Tried:

- Raw and preprocessed input features.
- Various simple and domain specific operators.
- Many kinds of fitness functions (AMS, smooted AMS, AUC, cross-entropy, area under ams, etc).
- Penalizing solutions with high variance of fitness over different subsets of the training set.

Result?
Overfitting. With AMS especially, but even with others.

Hedge your Bets

Going was rough with Genetic Programming so the Process got amended:

```
Don't put all your eggs in one basket.
```

In execution, it meant that I started working on neural nets, fully expecting to combine them with GP later in an ensemble or by stacking one on top of the other.

THE GRAND NEURAL NET HYPERPARAMETER SEARCH

- training algorithm (SGD, conjugate gradients, etc)
- learning rate
- batch size
- momentum
- momentum type (normal, nesterov)
- number of layers
- widths of layers
- weight initialization (uniform, gaussian, orthonormal)
- regularization (l1, l2, sparsity penalties)
- weight constraints (non-negative, max norm)
- activation function (sigmoid, tanh, relu, maxout, max-channel, etc)

ASSESSING PROGRESS

Wanted: tight feedback loops on all levels.

- Process: Monitor progress, change method of working if stuck (e.g. read physics/ML literature to get ideas).
- Algorithm: monitor how scores evolve during training.

Thwarted by noisy evaluation. Addressed by repeated CV, bagging and CV-bagging finally.

THE GRAND NEURAL NET HYPERPARAMETER SEARCH 2

Just redo the whole thing. Only it now takes 20 times longer to get a score. But at least the score is reliable.

THE GRAND NEURAL NET HYPERPARAMETER SEARCH 2

Just redo the whole thing. Only it now takes 20 times longer to get a score. But at least the score is reliable.
Once the possibilities are exhausted, or when the progress stops it's time to move on to something else ...

BEST FAILURES

- pseudo labeling (labeling test data with predicted labels late in the training)
- separate neural pathway for radians to reduce overfitting possibilities
- giving more weight to examples near the decision threshold
- splitting CV folds keeping pri-jet-num stratified
- removing features that casuse overfitting and adding their numeric “relevance” in their place
- distoring input with noise (dropout, additive, multiplicative)

FEATURE GENERATION

Trained xgboost on the same CV-bagging folds as the NN with slightly worse results than the NN. It was faster to train than the NN so it could be used as a fitness function for feature generation to explore into territory the Neural Net and the decision tree were blind to.

FEATURE GENERATION

Trained xgboost on the same CV-bagging folds as the NN with slightly worse results than the NN. It was faster to train than the NN so it could be used as a fitness function for feature generation to explore into territory the Neural Net and the decision tree were blind to.
But GP overfitted.

ENSEMBLING

- Bagging worked well but was slow to converge.
- CV bagging was faster (2 fold, 5 fold, any-fold).
- Taking a simple weighted average models of CV-bagged NN and xgboost models looked good.
- Stacking models, bags of models with a more powerful method overfitted badly (GP, Differential Evolution).
- Directly optimizing for the AMS (with a GP, for instance) overfitted terribly.

Model Selection

Repeated CV was the first key. The second one was to estimate how much weight the public leaderboard score shall be given.

Model Selection

Repeated CV was the first key. The second one was to estimate how much weight the public leaderboard score shall be given. 0.1

RANDOM OBSERVATIONS

Fail fast:

- Devise experiments to demonstrate usefulness/uselessness of ideas.
- Write unit tests.

Be able to retrace your steps:

- Keep a journal.
- Use version control.

RECAP

The meta-conclusion is that we need a tight feedback loop on multiple levels. For that we need:

- Reliably estimate score.
- Quickly try out new models (needs right hardware, software, process).

The conclusion is:

- Tuning the hell out of the same model may sometimes be worth it.
- Ensembling does not always work.
- Evolutionary Algorithms never work.

WHERE IS THE BOTTLENECK?

AMS is noisy, new features seem to help only marginally, feature generation overfits, optimizing AMS directly overfits, even ensembling can overfit. Data is extremely expensive. Simulated data is simply is just very, very expensive.

WHERE IS THE BOTTLENECK?

AMS is noisy, new features seem to help only marginally, feature generation overfits, optimizing AMS directly overfits, even ensembling can overfit. Data is extremely expensive. Simulated data is simply is just very, very expensive.
We need better regularization or much more data.

WHERE IS THE BOTTLENECK?

AMS is noisy, new features seem to help only marginally, feature generation overfits, optimizing AMS directly overfits, even ensembling can overfit. Data is extremely expensive. Simulated data is simply is just very, very expensive.
We need better regularization or much more data.

- Incorporate prior knowledge on what kind of interactions are expected (for example, by restricting connections in the neural network topology).
- Train a generative model that mimicks the simulator (i.e produces data, label pairs) and is quick.

FUTURE DIRECTIONS

A generative model could:

- act as a generative classifier itself
- or generate more training data cheaply for a discriminative classifier. Ideally, the generative model is good enough so that its samples are indistinguishable from real data. See Generative Adversarial Networks for an example of how to train a Generative and a Discriminative neural network jointly.