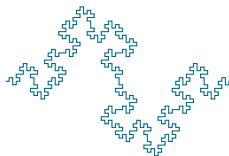


# HEP, HEP, HOORAY

Gábor Melis

@GaborMelis | <http://quotenil.com> | [mega@retes.hu](mailto:mega@retes.hu)

*Franz Inc., Fixnum Services*



2015 May 19

Higgs  
challenge



# the HiggsML challenge

May to September 2014

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

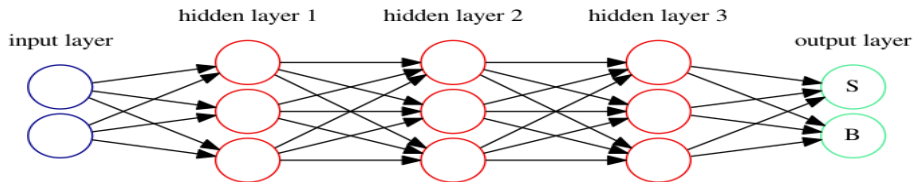
- ▶ Largest Kaggle competition to date.
- ▶ 4 months. Ouch.
- ▶ Classification task: separate tau-tau signal from background on simulated data (4-momenta of particles, some derived features).
- ▶ Unusual evaluation metric.
- ▶ Training set: 250000 labeled, weighted collision events
- ▶ Test set: 550000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



- ▶ inputs: normalized features ( $\sim 30$ ), 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,  $L_1 + L_2$  penalty and a mild sparsity constraint input weights

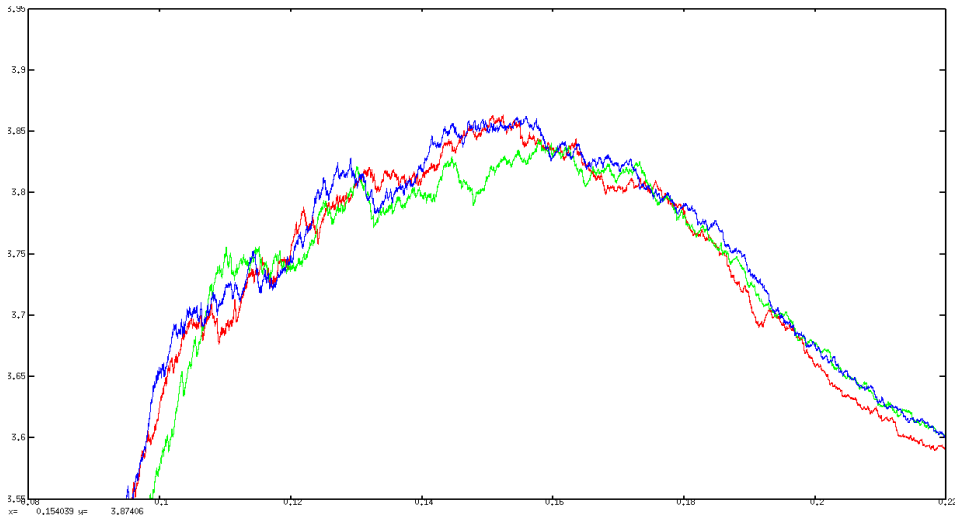
# THE GRAND NEURAL NET HYPERPARAMETER SEARCH

- ▶ 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 ( $L_1$ ,  $L_2$ , sparsity penalties)
- ▶ weight constraints (non-negative, max norm)
- ▶ activation function (sigmoid, tanh, relu, maxout, max-channel, etc)

# NEURAL NETWORK PROS AND CONS

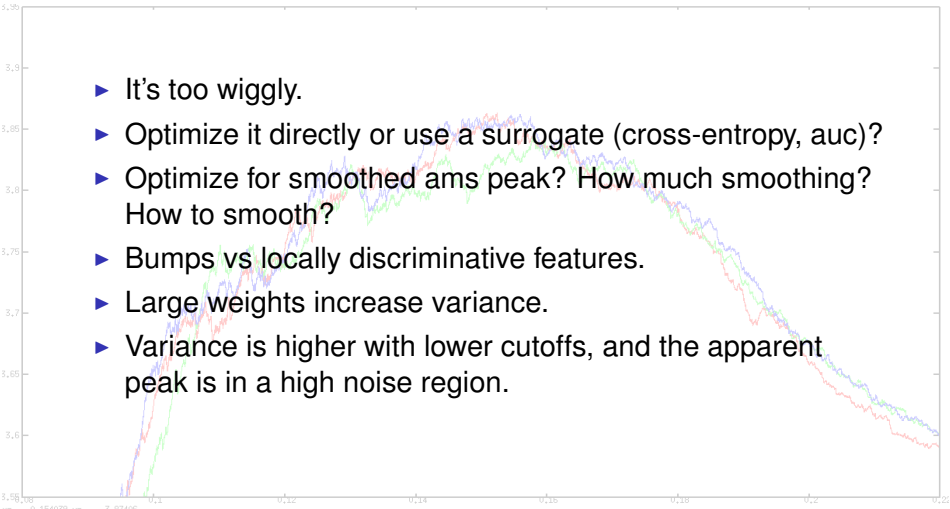
- + Powerful: better by  $\sim 0.05$  than Boosted Decision Trees (BDT, xgboost).
- + Your best bet to find interactions you don't know about.
  - Somewhat sensitive to input encoding, normalization and also to correlated inputs (mitigated by bagging).
  - Can take a long time to train.
  - Lots of choices,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

# PROBLEMS WITH AMS

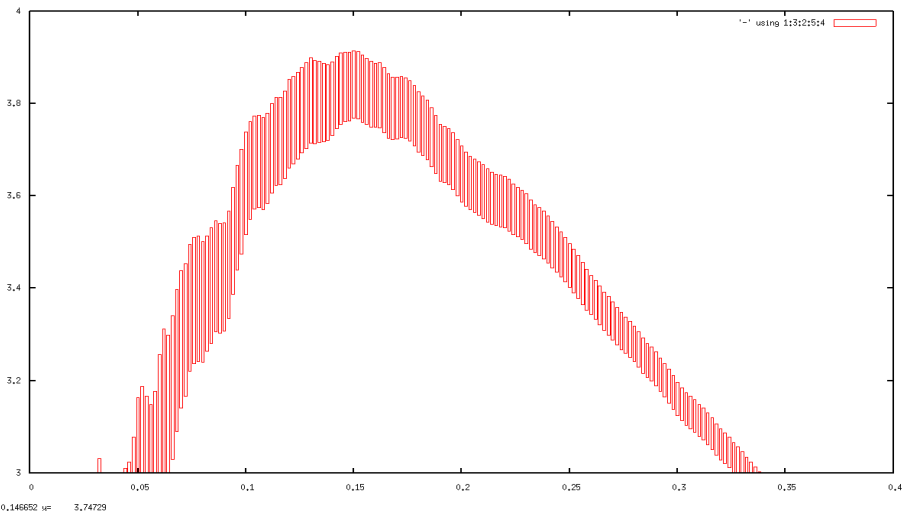




# PROBLEMS WITH AMS

- 
- ▶ 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



# CROSS-VALIDATION

- ▶ Public leaderboard dataset too small, single cutoff, very unreliable.
- ▶ CV showed huge fluctuations across folds ( $\pm 0.15$ ) 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

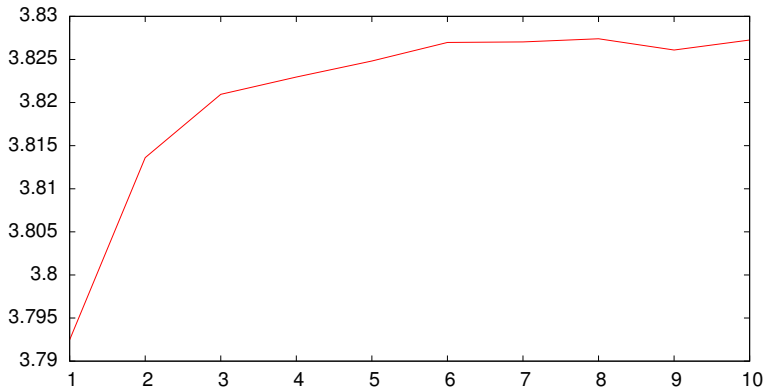
- ▶ Crowded decision boundary: AMS is very sensitive to minor differences in predicted probabilities.
- ▶ Model averaging to the rescue.
- ▶ Bagging: average models of the same kind trained on random subsets of the training set.
- ▶ Improved scores by about 0.1.

## CV BAGGING

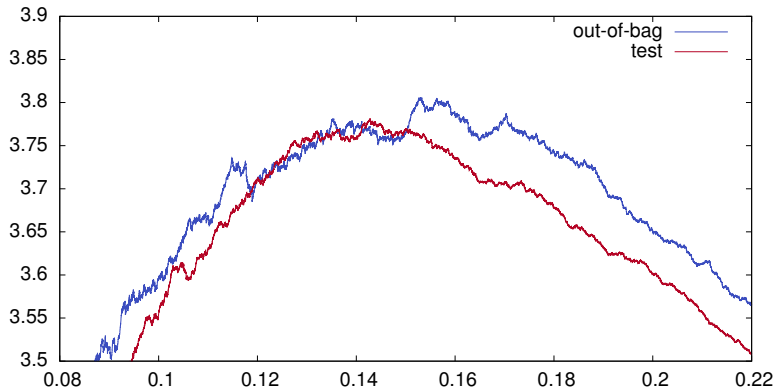
Repeated cross-validated bagging requires very many models to be trained (repetitions x folds x bags). But there is a shortcut: one can get a more reliable CV score, and a more accurate model *at the same time*, faster than with standard bagging by using the CV splits instead of random subsets of the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

# PEAK AMS VS BAG SIZE



# OUT-OF-BAG ESTIMATION VS ACTUAL TEST AMS



# FEATURE ENGINEERING

- ▶ Log transform long tailed features.
- ▶ Normalize all features to zero mean, stddev 1.
- ▶ Normalizing angles by rotational symmetry was ineffective.
- ▶ Drop all azimuthal angle features (PRI\*\_ETA) to reduce variance.
- ▶ DER\_MASS\_MMC and other derived features still useful with NNs (worth about 0.5)
- ▶ No missing value indicators.



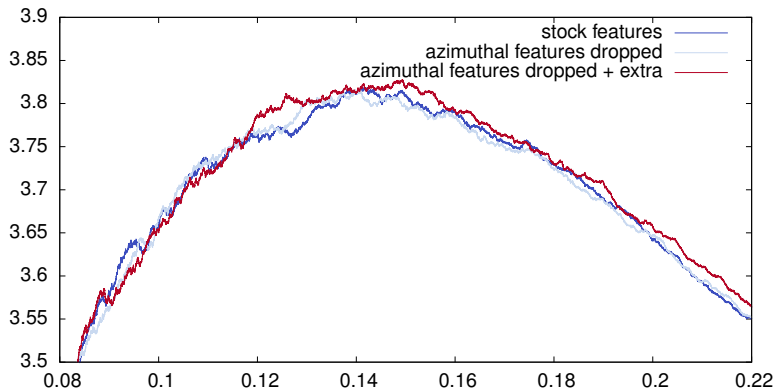
## FEATURE GENERATION

- ▶ Added four derived features based on azimuthal angles like this one:

$$\min(\phi_{\tau} - \phi_{lep}, \phi_{\tau} - \phi_{met}, \phi_{lep} - \phi_{met}), \quad (1)$$

- ▶ Added five derived features based on invariant and transverse masses of the tau, the lepton and the two jets.
- ▶ Added the modulus  $\ln(p_T \times \eta)$  of the pseudo particle whose momentum vector is the sum of the momentum vectors of the tau, the lepton and the jets.
- ▶ Deep NNs benefit less from feature engineering.
- ▶ Additional CAKE features seemed to contribute  $\sim 0.01$  to BDTs, less clear with NNs.

# EFFECT OF FEATURES

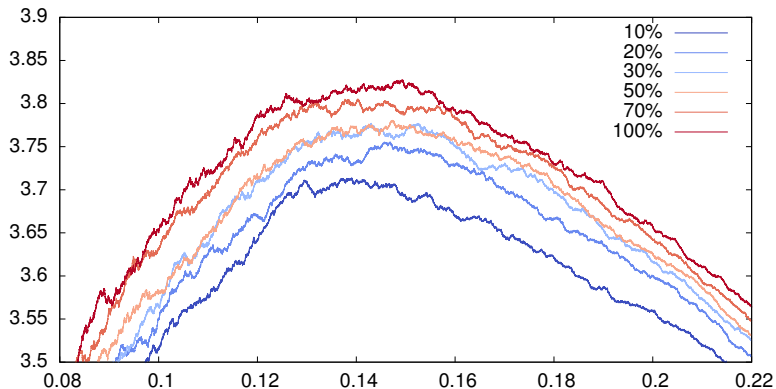


# REGULARIZATION

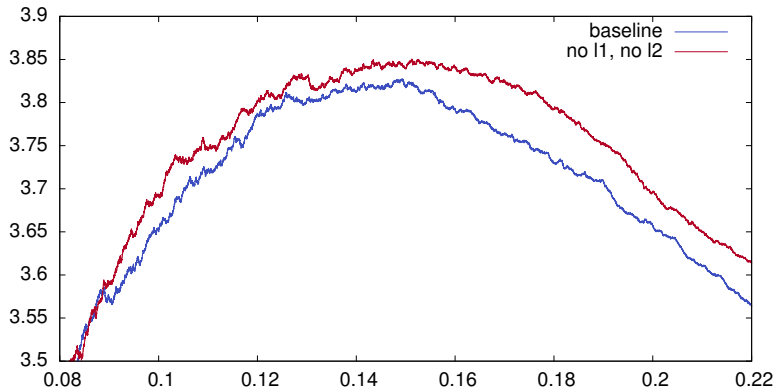
Dropout on the hidden layers and various penalties and constraints on the first layer weights.

- ▶  $L_1$  (encourage sparsity)
- ▶  $L_2$  (discourage large weights)
- ▶ Limit the number of input connections to first layer to 10.

# EFFECT OF DATA SET SIZE



# EFFECT OF REGULARIZATION



# ENSEMBLING

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

# 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. Perhaps this?

$$\frac{\sqrt{100000}}{\sqrt{250000} + \sqrt{100000}} = 0.387$$



## MODEL SELECTION

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

$$\frac{\sqrt{100000}}{\sqrt{250000} + \sqrt{100000}} = 0.387$$

No, there is more information in an entire curve than in a single value.

0.1

## 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 cause overfitting and adding their numeric “relevance” in their place
- ▶ distorting 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.

## 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.