Direct optimization of discovery significance in machine learning analysis with application in SUSY stop search

E.Reeves

Contents

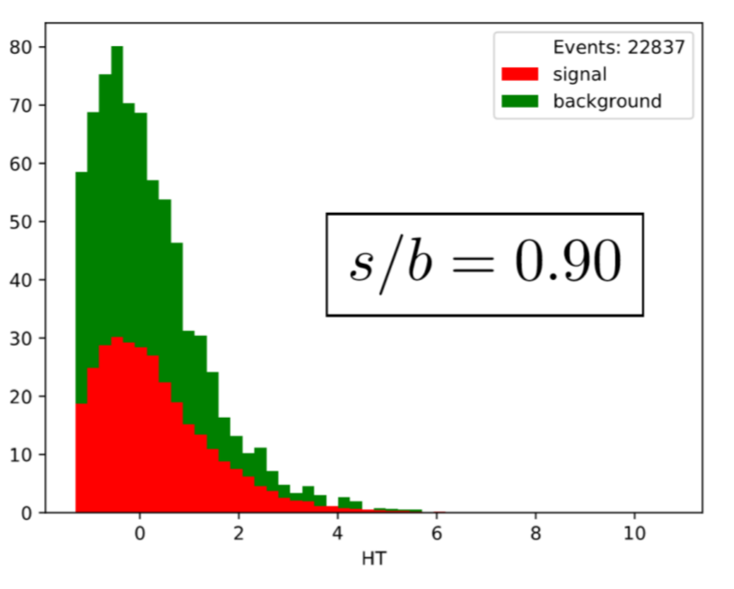
- Project overview
- Monte Carlo Model
- Results
- My contributions
- What have I learnt?

Direct optimization of discovery significance

- Most important aspect is the significance of the signal counts over the background counts
- Purity of background classification is not so important
- Usually accuracy is maximized though minimizing binary cross entropy
- One can define a loss function based around direct optimization of discovery significance, in this case the Asimov discovery significance[1]
- We maximize Z_A through minimizing $1/Z_A$

$$Z_A = \sqrt{2((s+b)\ln[\frac{(s+b)(b+\sigma_b^2)}{b^2+(s+b)\sigma_b^2}] - \frac{b^2}{\sigma_b^2}\ln[1+\frac{\sigma_b^2 s}{b(b+\sigma_b^2)}]} \rightarrow \frac{s}{\sqrt{s+b}}$$

s = correctly classified signal events
b = incorrectly classified background events
σ = systematic uncertainty



[2]HEP approach signal prediction

[1]arXiv:10071727v3 [2]M.Shchedrolosiev presentation, summer 2018

XGBoost implementation

- Uses second order approximation
- Additive training (boosting)
- Minimizes every next tree, f_t
- Loss function, l_{Asimov}, defined as $1/Z_A$
- g_i is the gradient
- Asimov score used as metric for early stopping procedure
- Autograd used for automatic differentiation

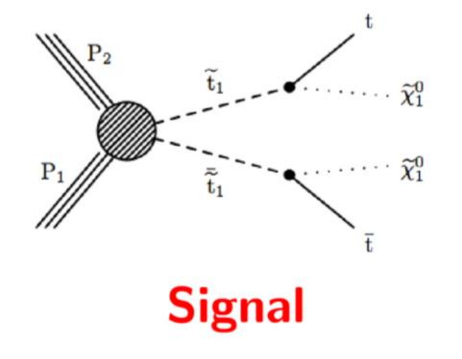
$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(_i)] + \Omega(f_t)$$

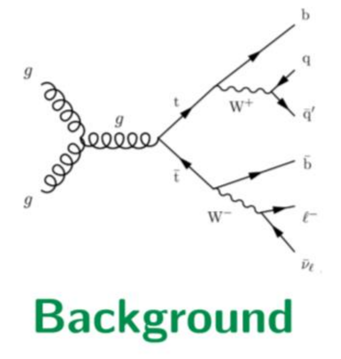
$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$$

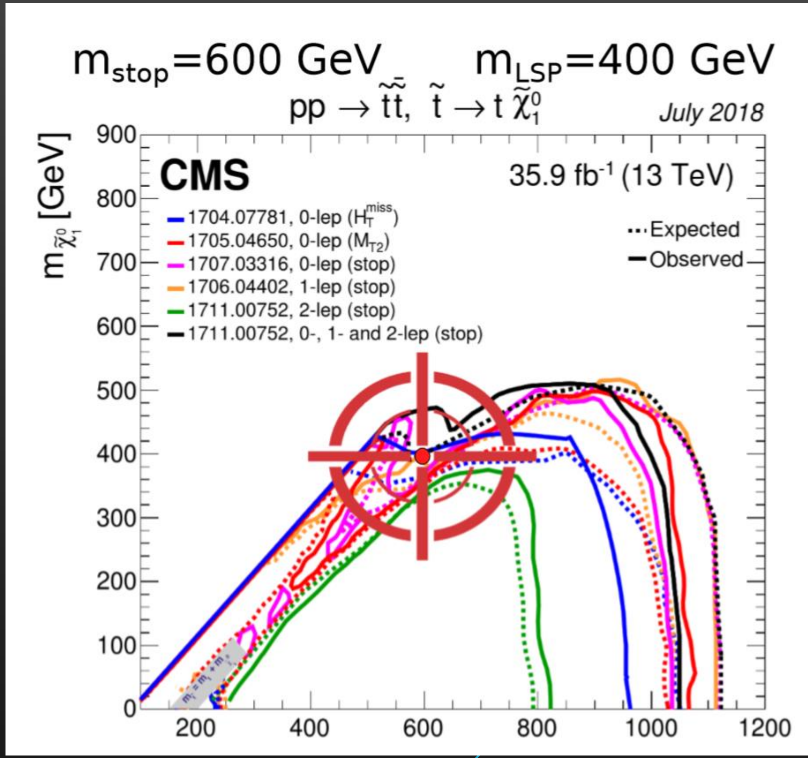
$$g_i = -Z_A^{-2}(\frac{\partial Z_A}{\partial s} W_s y_i + \frac{\partial Z_A}{\partial b} W_b(1 - y_i))$$

Monte Carlo

- To test approach look at stop SUSY model close to edge of exclusion at 30 fb-1 of 13TeV of LHC data
- Currently using compressed model data. Code can also be used with uncompressed data
- Taken 1M events of signal and background with Pythia and Delphes with basic selection criteria: 1 lepton pT≥40 GeV, 4 jets pT≥30 GeV, at least 1 b-tagged jet



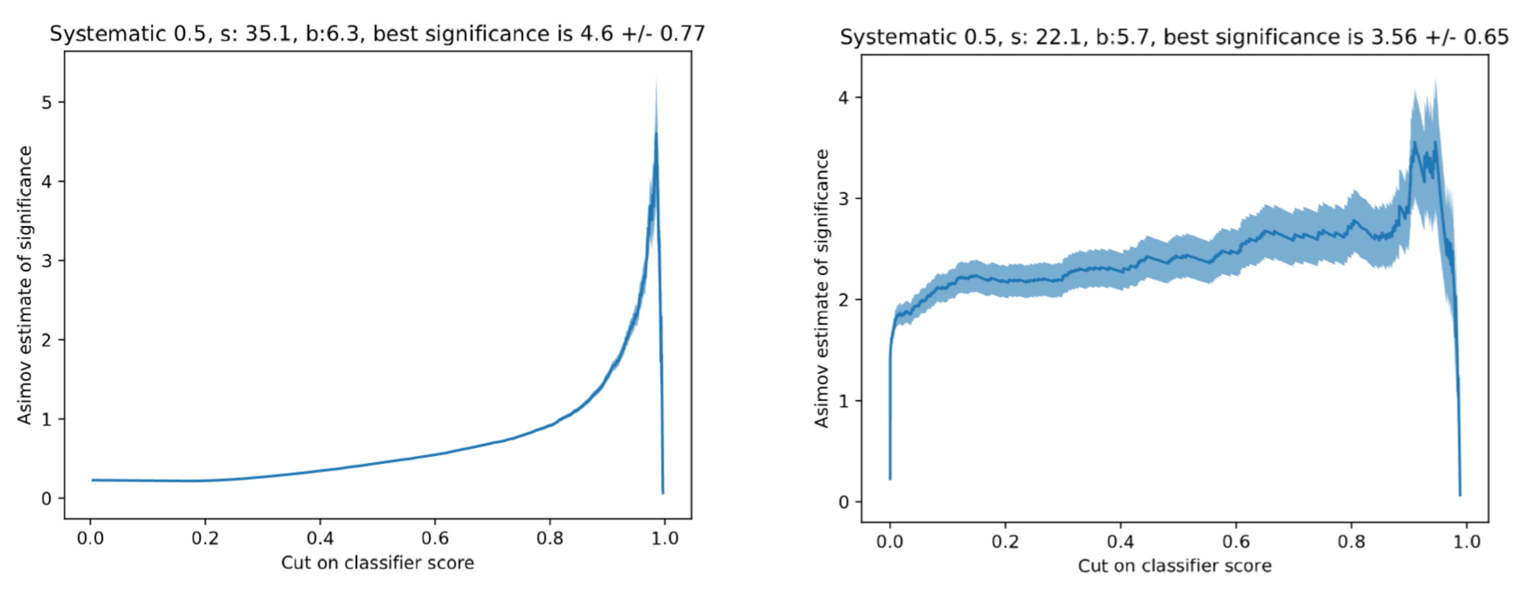




[3]https://indico.desy.de/indico/event/21116/contribution/0/material/slides/0.pdf

Results

- Best results from Mykyta for XGBoost
- Compared to best results from the paper for DNN
- Asimov nearly comparable to cross entropy for high uncertainty XGBoost.
- DNN still the best performance
- Asimov performs better than cross entropy at high systematic uncertainty for compressed model



[3]XGBoost results: Cross entropy vs Asimov loss

Comp. mod.	s	b	$Z_A(10\%)$	s	b	$Z_A(30\%)$	s	b	$Z_A(50\%)$
Loss:									
cross entropy	74.4	18.2	10.7 ± 0.3	44.0	7.7	6.8 ± 0.3	40.5	6.8	4.8 ± 0.3
ℓ_{Asimov}	78.4	19.4	10.8 ± 0.3	25.9	3.2	6.8 ± 0.4	11.9	0.5	6.2 ± 0.6

[4]DNN Results: Best Asimov significance for cross entropy optimization vs direct significance optimization

[4]A.Elwood, D.Krücker, M.Shchedrolosiev, "Direct optimization of the discovery significance in machine learning for new physics searches in particle colliders"

My Aims

- Two separate scripts – run.py and exampleScript.py
- Combine the two to create a more legible, easier to run script with XGBoost implementation ready for paper publication
- Create new environment with XGB modules to run the script in
- Push to github

Data Frames

- Original data frame sourcing and converting from root file to pickle '/nfs/dust/cms/group/susydesy/marco/training_sample_new'
- Taken from run.py, data processing altered in exampleScript.py
- Easier variable choice, allowing for exclusion of high level variables
- Defined splitting of data once, not separately for each class

```
#Either make the dataframes fresh from the trees or just read them in
if makeDfs:
    print "Making DataFrames"

    signalFile = []#'/nfs/dust/cms/group/susy-desy/marco/training_sample_new/stop_sample_0.root'
    bkgdFile = []#'/nfs/dust/cms/group/susy-desy/marco/training_sample_new/top_sample_0.root'

    for i in range(nInputFiles):
        signalFile.append(' /nfs/dust/cms/group/susy-desy/marco/leonid/stop_600_400/stop_samples_'+str(i)+'.root')
        bkgdFile.append('/nfs/dust/cms/group/susy-desy/marco/training_sample_new/top_sample_'+str(i)+'.root')

    signal = convertTree(signalFile,signal=True,passFilePath=True,tlVectors = ['selJet','sel_lep'])
    bkgd = convertTree(bkgdFile,signal=False,passFilePath=True,tlVectors = ['selJet','sel_lep'])

    # #Expand the variables to 1D
    signal = expandArrays(signal)
    bkgd = expandArrays(bkgd)

    if saveDfs:
        print 'Saving the dataframes'
        # Save the dfs?
        if not os.path.exists('dfs'): os.makedirs('dfs')
        print 'signal size:',len(signal)
        signal.to_pickle('dfs/signal'+appendInputName+'.pkl')
        print 'bkgd size:',len(bkgd)
        bkgd.to_pickle('dfs/bkgd'+appendInputName+'.pkl')
else:
    print "Loading DataFrames"

    signal = pd.read_pickle('dfs/signalleonid.pkl')
    bkgd = pd.read_pickle('dfs/bkgdleonid.pkl')
```

Data frame sourcing

```
#======== Choosing variables ========#
#Now carry out machine learning (with some algo specific diagnostics)
    #Choose the variables to train on

    chosenVars = {

            # #The 4 vectors only, if don't want HL variables
            # 'fourVector':['signal',
            # 'sel_lep_pt','sel_lep_eta','sel_lep_phi','sel_lep_m',
            # 'selJet_phi','selJet_pt','selJet_eta','selJet_m','MET'],
            #
            # 'fourVectorBL':['signal','lep_type','selJetB',
            # 'sel_lep_pt','sel_lep_eta','sel_lep_phi','sel_lep_m',
            # 'selJet_phi','selJet_pt','selJet_eta','selJet_m','MET'],
            #
            # 'fourVectorMT':['signal',
            # 'sel_lep_pt','sel_lep_eta','sel_lep_phi','sel_lep_m',
            # 'selJet_phi','selJet_pt','selJet_eta','selJet_m','MET','MT'],
            #
            # 'fourVectorMT2W':['signal',
            # 'sel_lep_pt','sel_lep_eta','sel_lep_phi','sel_lep_m',
            # 'selJet_phi','selJet_pt','selJet_eta','selJet_m','MET','MT2W'],
            #
            # 'fourVectorHT':['signal',
            # 'sel_lep_pt','sel_lep_eta','sel_lep_phi','sel_lep_m',
            # 'selJet_phi','selJet_pt','selJet_eta','selJet_m','MET','HT'],
            #
```

Variable choice

Grid Search

- Implement a grid search to scan a range of hyperparameters
- Avoids having to do full hyperparameter optimization
- Still option to just use default configurations
- Grid search does not automatically plot with new parameters
- Not all options implemented yet

```
#======== If carrying out grid search ========#
#If doing the grid search
def hiddenLayerGrid(nLayers,nNodes):
    hlg=[]
    for nn in nNodes:
        for nl in nLayers:
            hlg.append([nn for x in range(nl)])
        pass
    return hlg

dnnGridParams = dict(
        mlp__epochs=[10,20,50],
        mlp__batch_size=[32,64],
        mlp__hiddenLayers=hiddenLayerGrid([1,2,3,4,5],[2.0,1.0,0.5]),
        mlp__dropOut=[None,0.25,0.5],
        # mlp__activation=['relu','sigmoid','tanh'],
        # mlp__optimizer=['adam','sgd','rmsprop'],
        ## NOT IMPLEMENTED YET:
        # mlp__learningRate=[0.5,1.0],
        # mlp__weightConstraint=[1.0,3.0,5.0]
        )

bdtGridParams = dict(
        base_estimator__max_depth=[3,5],
        base_estimator__min_samples_leaf=[0.05,0.2],
        n_estimators=[400,800]
        )
```

Setting up grid search

```
Best: 0.795610 using {'mlp__dropOut': None, 'mlp__batch_size': 32}

0.795610 (0.001938) with: {'mlp__dropOut': None, 'mlp__batch_size': 32}
0.783410 (0.001315) with: {'mlp__dropOut': 0.25, 'mlp__batch_size': 32}
0.779960 (0.001371) with: {'mlp__dropOut': 0.5, 'mlp__batch_size': 32}
0.785860 (0.005014) with: {'mlp__dropOut': None, 'mlp__batch_size': 64}
0.783880 (0.002477) with: {'mlp__dropOut': 0.25, 'mlp__batch_size': 64}
0.778050 (0.002813) with: {'mlp__dropOut': 0.5, 'mlp__batch_size': 64}
```

DNN grid search results

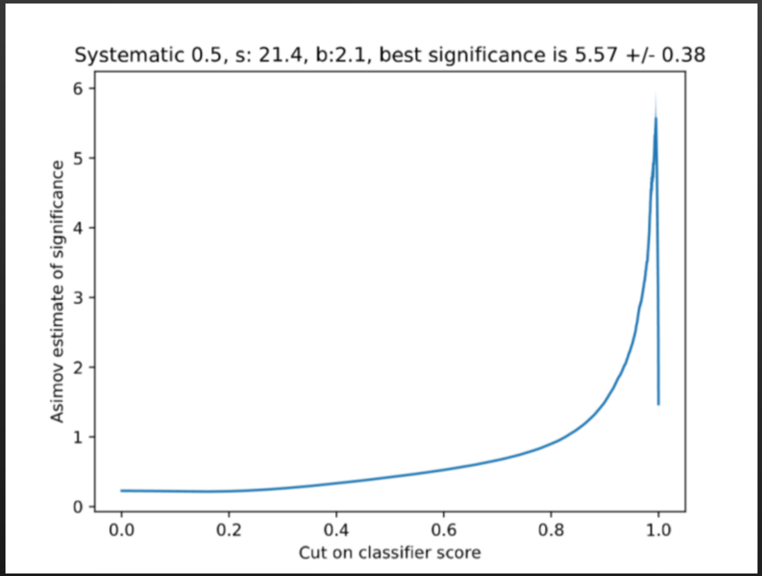
```
Best: 0.519770 using {'n_estimators': 400, 'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 0.05}

0.519770 (0.023822) with: {'n_estimators': 400, 'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 0.05}
0.519390 (0.023285) with: {'n_estimators': 800, 'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 0.05}
0.519770 (0.023822) with: {'n_estimators': 400, 'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 0.2}
0.519390 (0.023285) with: {'n_estimators': 800, 'base_estimator__max_depth': 3, 'base_estimator__min_samples_leaf': 0.2}
0.519770 (0.023822) with: {'n_estimators': 400, 'base_estimator__max_depth': 5, 'base_estimator__min_samples_leaf': 0.05}
0.519390 (0.023285) with: {'n_estimators': 800, 'base_estimator__max_depth': 5, 'base_estimator__min_samples_leaf': 0.05}
0.519770 (0.023822) with: {'n_estimators': 400, 'base_estimator__max_depth': 5, 'base_estimator__min_samples_leaf': 0.2}
0.519390 (0.023285) with: {'n_estimators': 800, 'base_estimator__max_depth': 5, 'base_estimator__min_samples_leaf': 0.2}
```

XGBoost grid search results

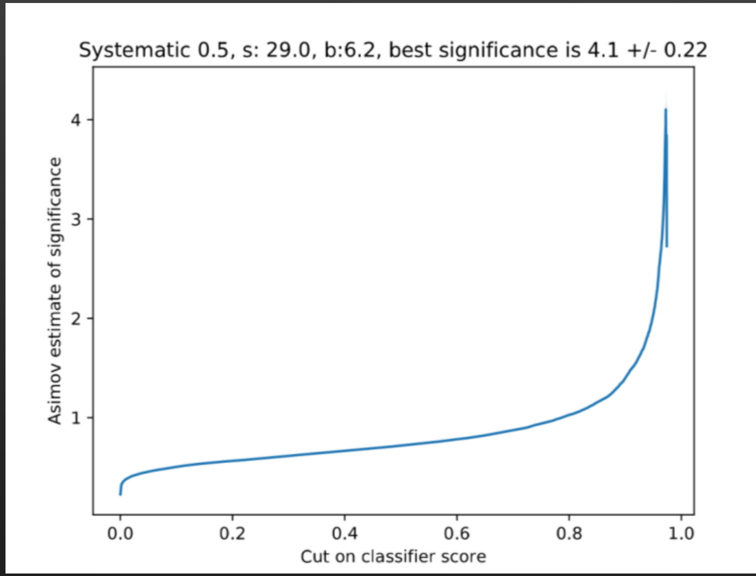
Other Alterations

- General tidying up and formatting
- Make sure all variables are defined properly for each classification
- Successfully running Plotting, DNN, XGBoost, Regression
- Bdt with Adaboost classifier not working due to error in Bdt code from original source, not so relevant to the paper
- Combined functionality of run.py plus exampleScript.py (total lines > 1100)
- Reduced to < 600 lines of code

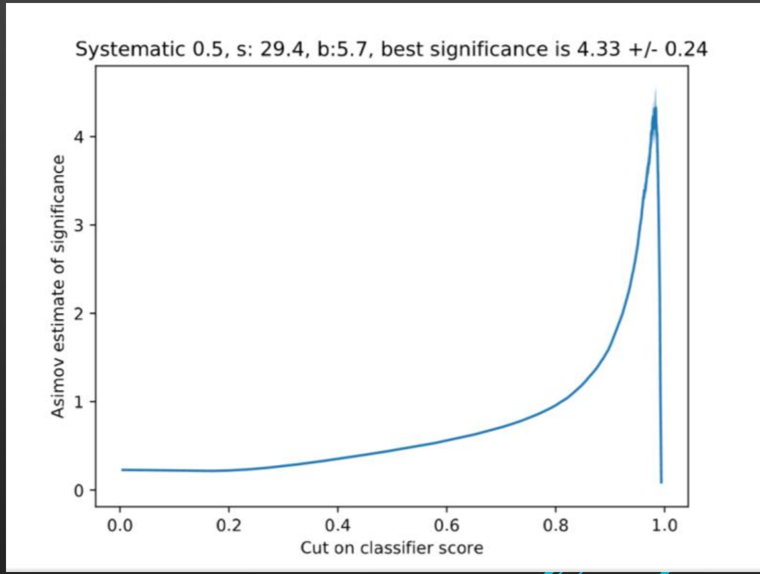
Run_xgb.Py Results



DNN with Asimov loss function



XGBoost with Asimov loss function



XGBoost with binary cross-entropy loss function

The results show a good replication of expected behaviour

Summary And Outlook

- Asimov loss demonstrates comparable significance at high systematics for XGBoost with compressed model
- Better significance at high systematic for DNN with compressed model
- I have cleaned up code, including grid search and data frame sourcing
- Good replication of expected behaviour
- To do: grid search with automatic plotting, fix bdt, reproduce results from paper with correct configs

What Have I Learnt?

- Some introductory supersymmetry
- The basics of how the CMS detector works
- Python and Numpy
- Linux
- Git
- What machine learning is and a few different classes
- Bdts, XGBoost, DNNs
- Worked with tensorflow and keras
- Worked on CMS servers
- More generally: how a site like DESY runs, how groups like CMS and SUSY work
- Attended some interesting discussions and colloquiums

THANKS!