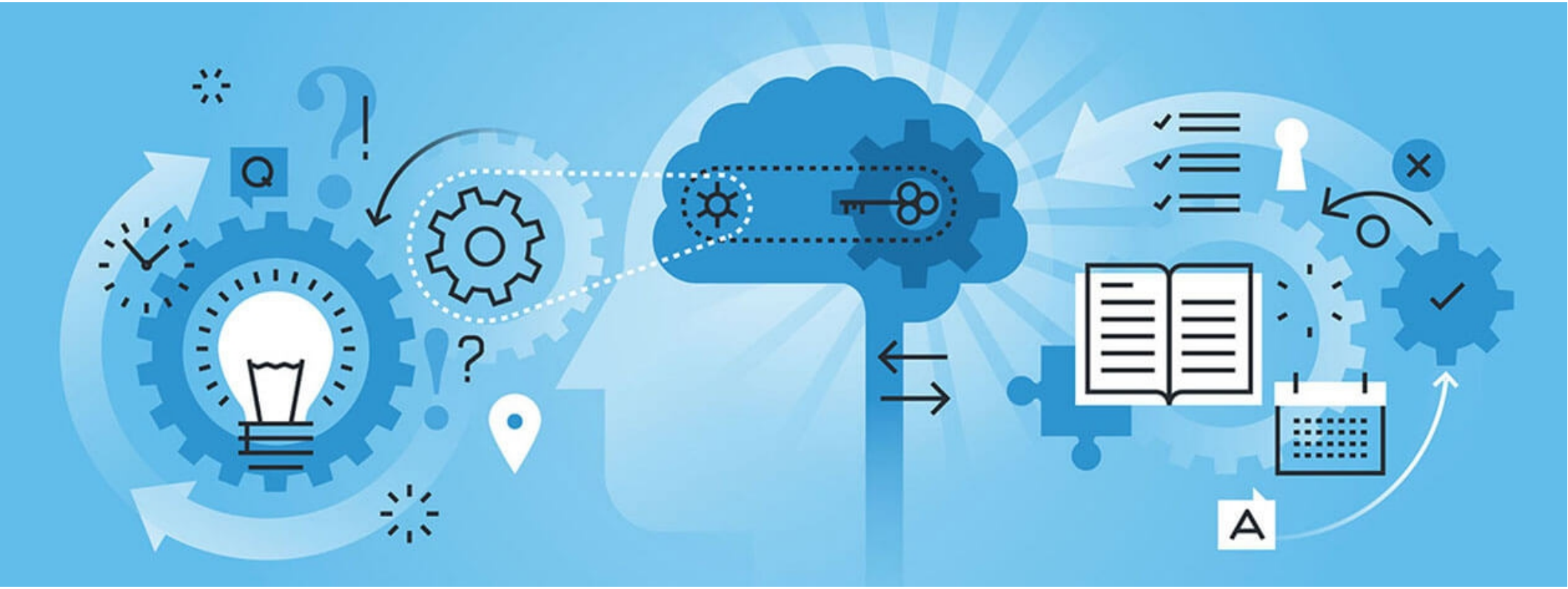
Status report on AI-assisted Computing WTC-Console



Caltech Machine Learning Meeting

Christian Contreras, Jean-Roch Vlimant, Daniel Abercrombie

Overview

- Central MC production manages thousands of “workflow” tasks each with thousands of jobs.

- Common issues are errors in grid jobs may be due to missing/corrupt input files, high memory usage, etc.

- Currently all handled manually in Operations

    - An operator must look at the error codes and decide on what appropriate actions to take on the workflow

    - Some classes of errors are easy to understand, and there are automated responses

    - An algorithm that encompasses all possible patterns that can be anticipated would be difficult to write and - most important - impossible to maintain

Machine learning stands as a very natural solution

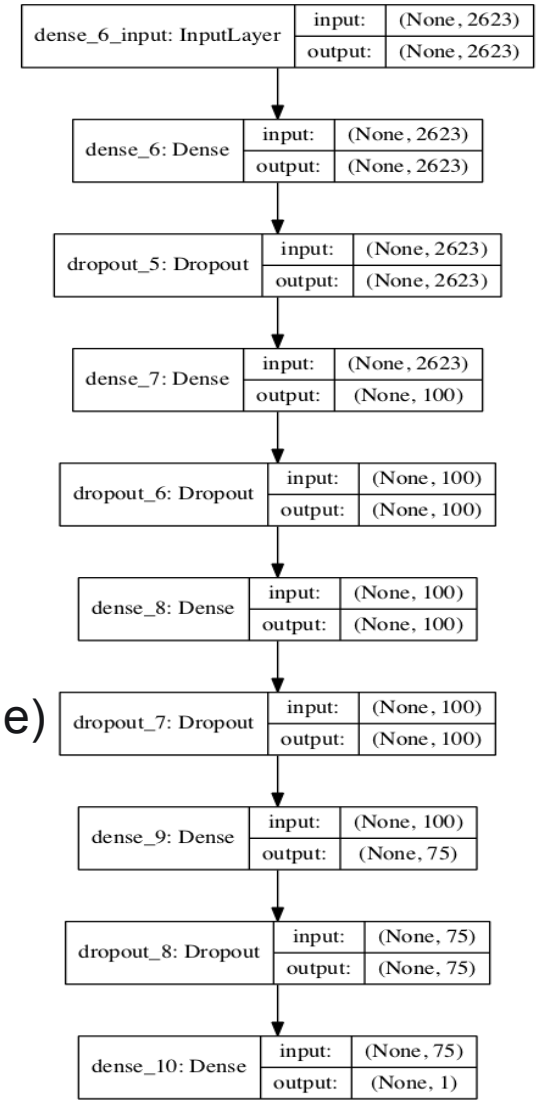
General goal:

• Deliver error handling predictions and mature towards moving manual operator intervention into automated actions

Goal and Strategy

Deliver a CMS workflow failures recovery panel towards AI assisted operations

- Use supervised learning to build classifier that predict recovery action (kill, clone, resubmit, recover as appropriate)
- Pull in data to build a model from CMS data service
  - Use the “exit codes - site status” as input information
- We've establish workflow pipeline for development
- Study enhancement techniques for data and model tuning
- Scale features using statistics that are robust to outliers
  - Scales according to the quantile range (25th quantile & 75th quantile)
- Data resampling methods to handle class imbalance
  - Synthetic Minority Over-sampling, Under- and Over-Sampling technique
- Classifier performance measurements:
  - ROC, accuracy, precision, recall, log-loss, F1-score, KS-test metrics
  - Use of Cross-Validation method for more robust measurements



Workflow handling

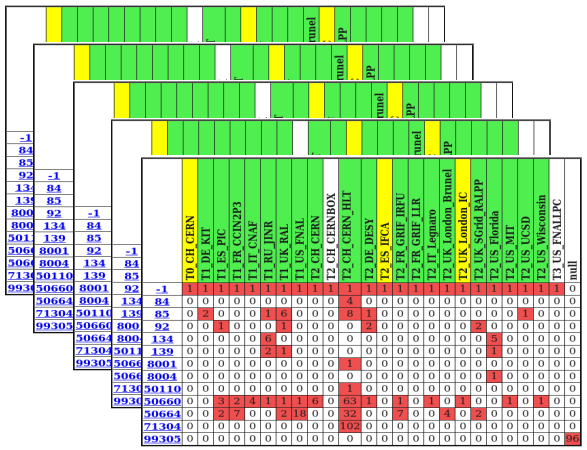
- For each task (workflow+campaign), we know the number of times each possible error code is thrown at each site.
- This leads to a simple matrix of numbers of error codes per sites, with each element being the number of matching errors
  - Features considering the site status at time of the error (enabled, disabled, or in drain) not considered.
  - Currently we just split these into “good” and “bad” site status.
- Good/bad site, refers to the site availability, maintained by the site support team, at the time the workflow was reported as needing assistance.
- Information stored in JSON files
  - Acquired using Workflow Team Web Tools that extracts information from CMS Site-Readiness report

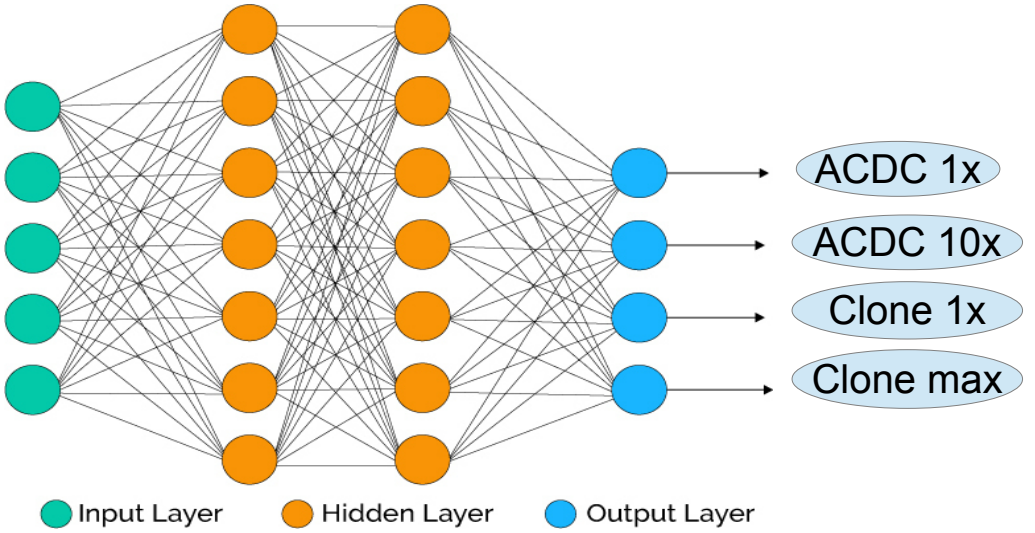
Data representation & Pipeline

- Workflow task sample data in the form of exit code vs site table
  - **Good site** 48 rows × 62 columns
  - **Bad site** 41 rows× 122 columns
  - Built from status key hierarchy
- 7582 total tasks
  - Data split 80%/20% for training & testing
  - Additionally training data further split for training & validation

To top

	T0_CH_CERN	T1_DE_KIT	T1_ES_PIC	T1_FR_CCIN2P3	T1_IT_CNAF	T1_RU_JINR	T1_UK_RAL	T1_US_FNAL	T2_CH_CERN	T2_CH_CERNBOX	T2_CH_CERN_HLT	T2_DE_DESY	T2_ES_IFCA	T2_FR_GRIF_IRFU	T2_FR_GRIF_LLR	T2_IT_Legnaro	T2_UK_London_Brunel	T2_UK_London_IC	T2_UK_SGrid_RALPP	T2_US_Florida	T2_US_MIT	T2_US_UCSD	T2_US_Wisconsin	T3_US_FNALLPC	null
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
84	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85	0	2	0	0	0	1	6	0	0	0	8	1	0	0	0	0	0	0	0	0	0	1	0	0	0
92	0	0	1	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	2	0	0	0	0	0	0
134	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
139	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
8001	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
50110	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50660	0	0	3	2	4	1	1	1	6	0	63	1	0	1	0	1	0	1	0	0	1	0	1	0	0
50664	0	0	2	7	0	0	2	18	0	0	32	0	0	7	0	0	4	0	2	0	0	0	0	0	0
71304	0	0	0	0	0	0	0	0	0	0	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0
99305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96



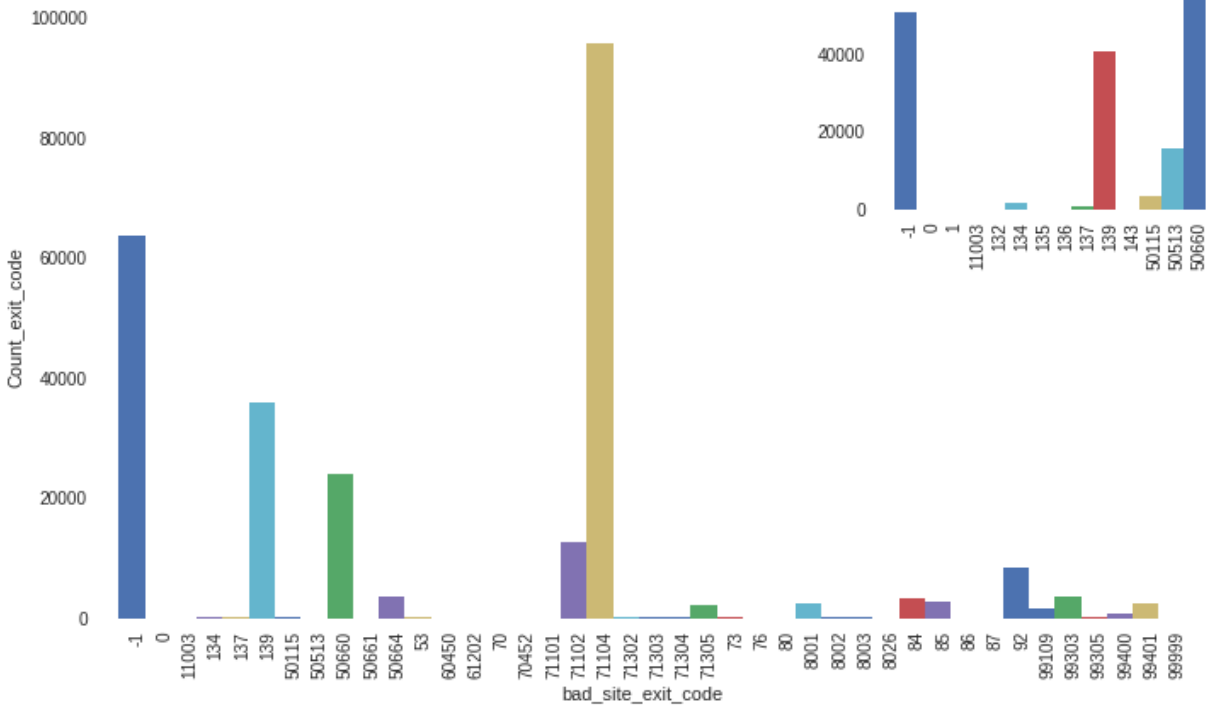


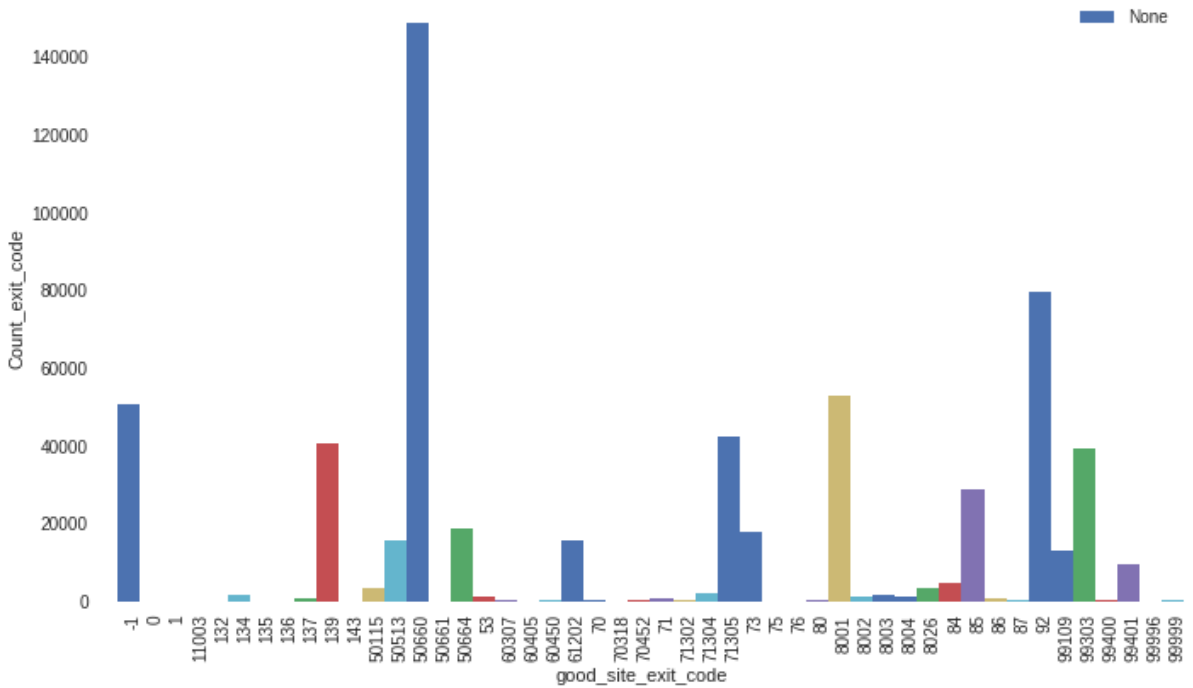
Good & Bad site description

- Good and bad site, refers to the site availability, maintained by the site support team, at the time the workf bw was reported as needing assistance.
- The site support team runs multiple tests over all of the sites.
- SAM and HammerCloud tests are the ones they often report.
- One tests if f les are accessible, and I'm not sure about the other.
- The main thing is that production jobs should not be expected to run successfully at "bad" sites,
  while they should be able to run at "good" sites that have the correct f les.

Good & bad site exit code counts profile across all production campaign

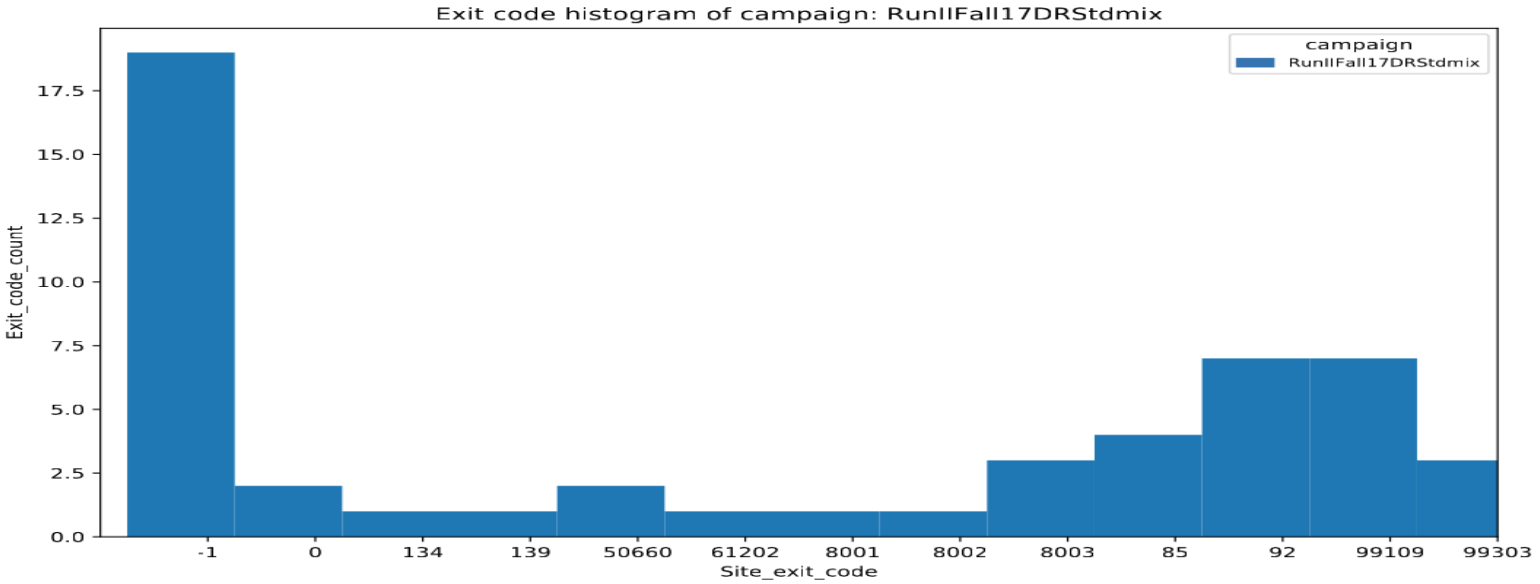
- Most common exit codes are 50660, 7104, -1, 92, 8001, and 139

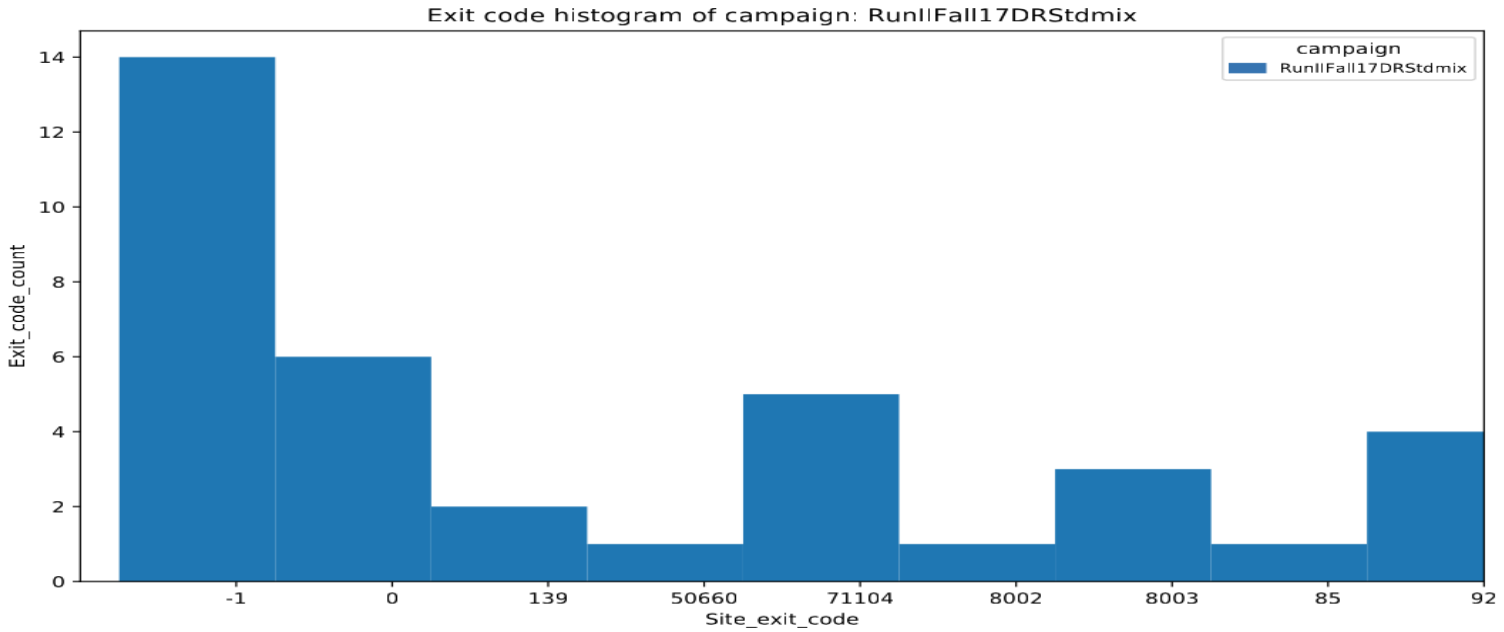




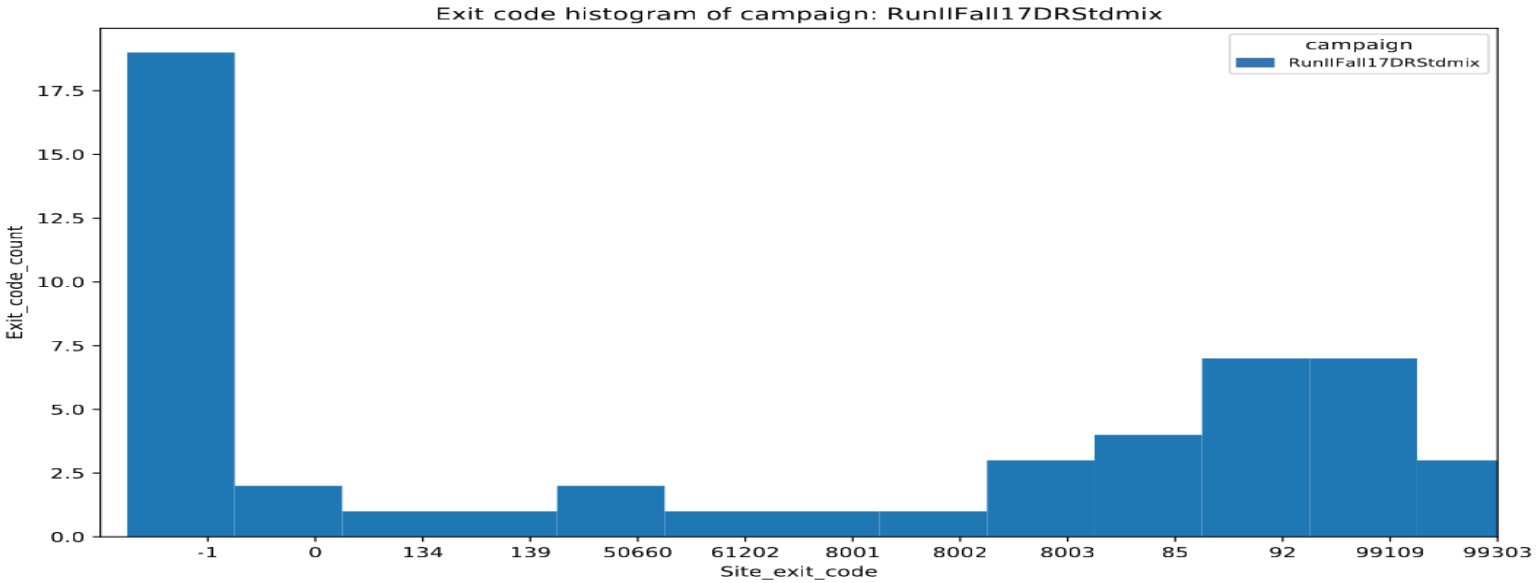
- Exit code may be related to workflow, site, or merge issues, etc.

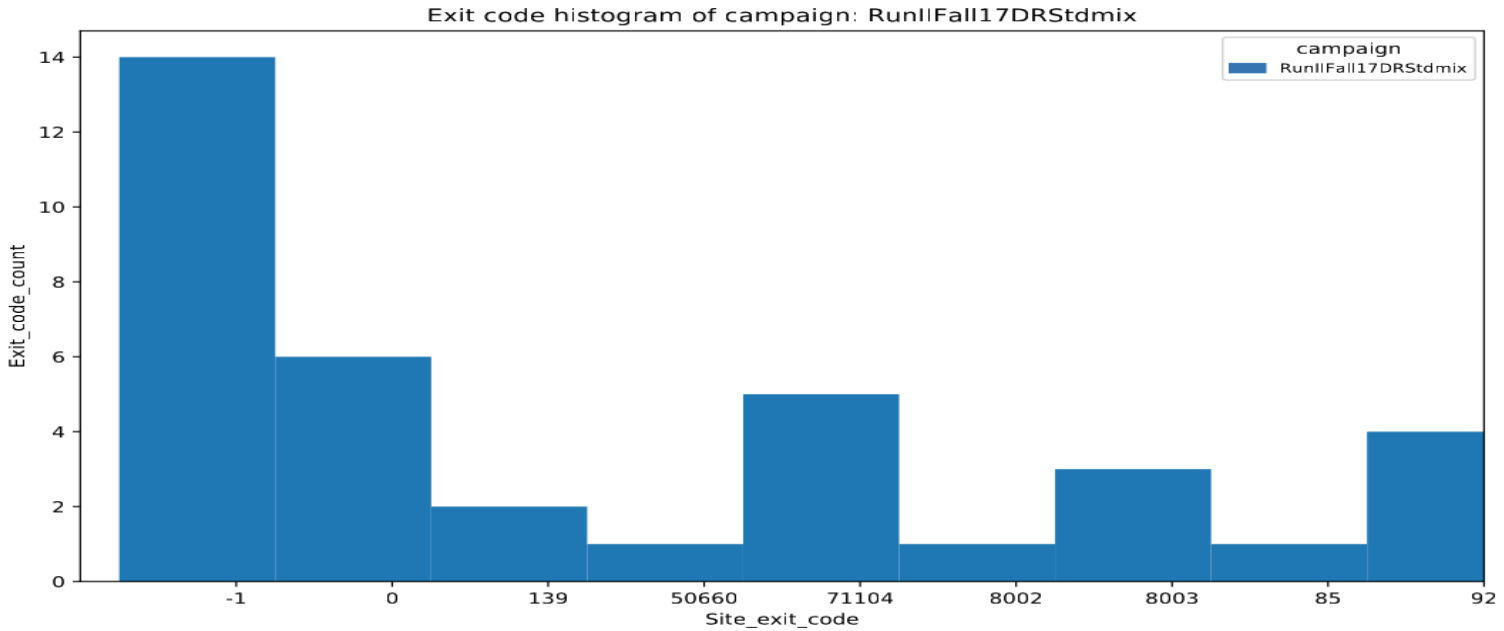
Example exit code counts for Run Fall 17 DR Stdmix campaign





Example exit code counts for Run Fall 17 DR Stdmix campaign

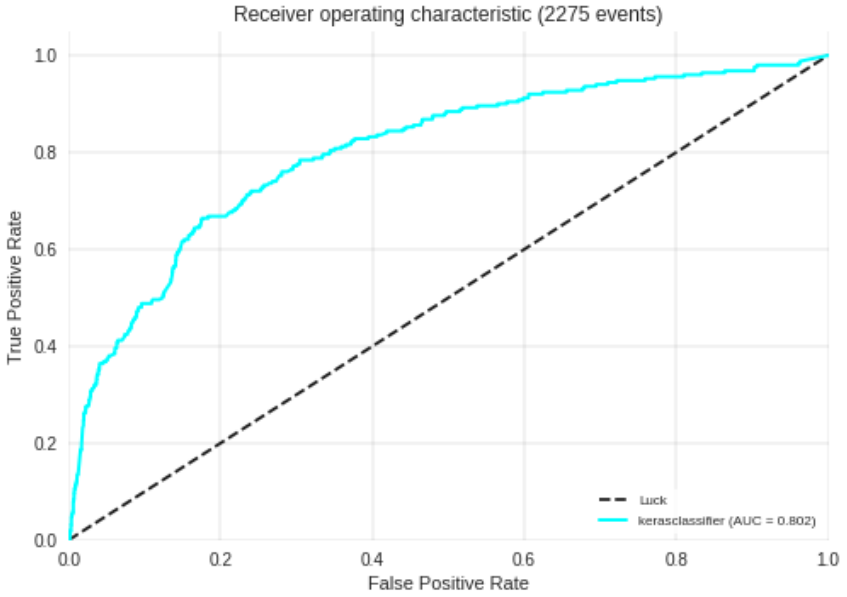


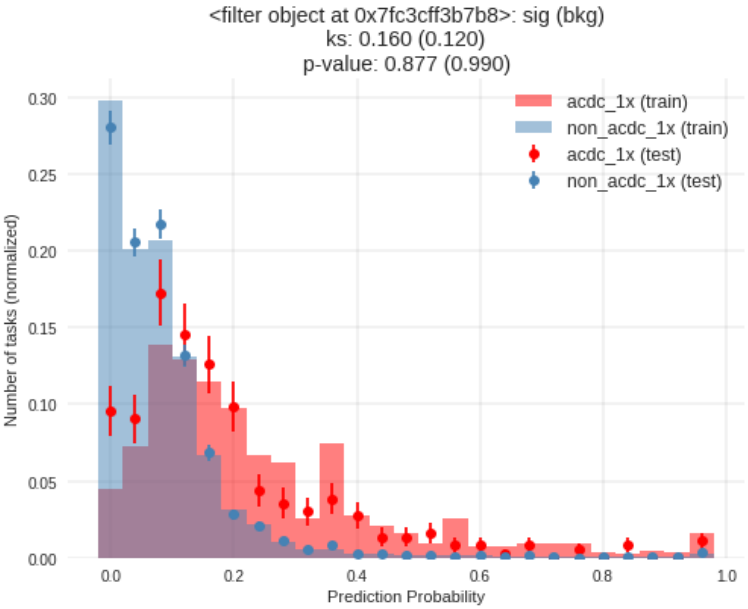


Combine Good & Bad sites:

AUC metrics to check model performance

- Binary classification was the initial approach taken (acdc vs non-acdc)
- Inclusion of both good and bad site information in the modeling leads to a boost in ROC AUC performance
  - Scaling included
- Evaluated general model performance on the remaining 20% of tasks
  - **Left:** ROC and
  - **Right:** Over-training plots



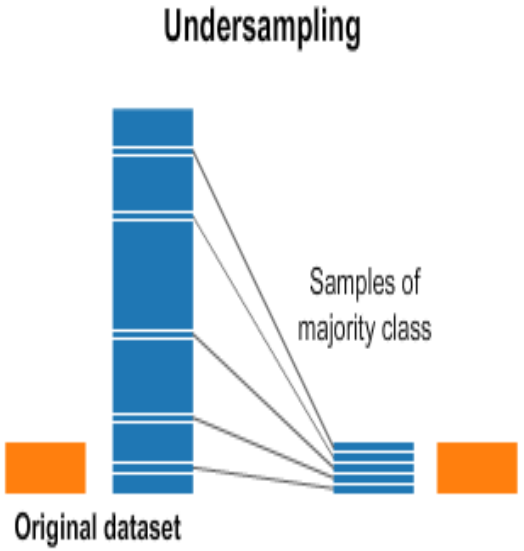


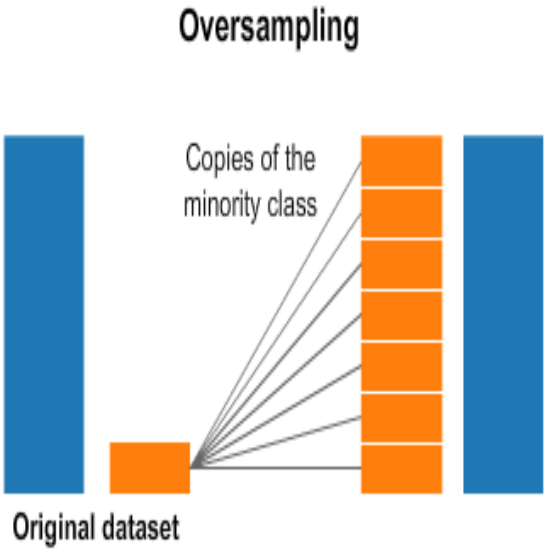
Class labels

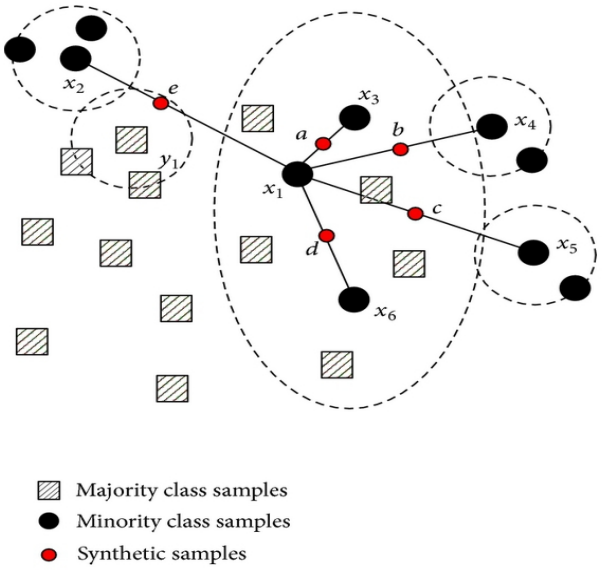
- Action labels: ACDC, clone, and special
- The "special" action just has a few simple options for "action" under it:
  - https://github.com/dabercro/WorkflowWebTools/blob/master/runserver/static/js/makeparams.js#L175-L182
- special' action options: by-pass, force-complete, and on-hold
- Also use job split as class label to predict on
  - Job split options: 1x, 2x, 10x, 20x, 50x, and max
- Examples of
  - Binary classification: acdc and non-acdc
  - Multiclass classification: acdc_50x, clone_10x, acdc_max (not a complet list here)

Handling Imbalanced Class Distribution

- A dataset is imbalanced if the classification categories are not approximately equally represented.
- We studied the use of Synthetic Minority Over-sampling resampling method to deal with highly unbalanced datasets. It consists of under-sampling the majority class and adding more synthetic examples from the minority class.



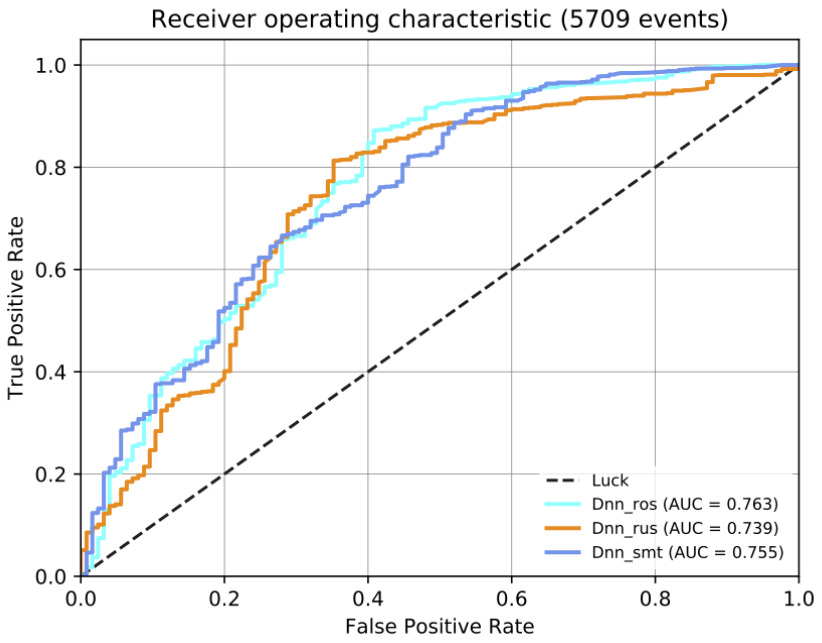




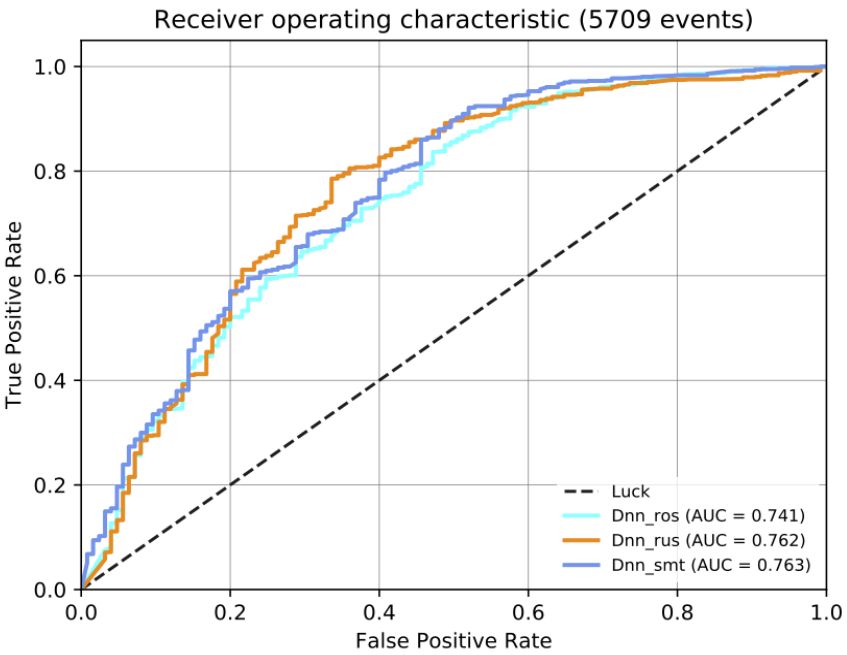
Comparison between w/o scaling applied while using resampling

- Older plot based on smaller number of samples (not to be compared with previous plots)

Resampling and scaling applied

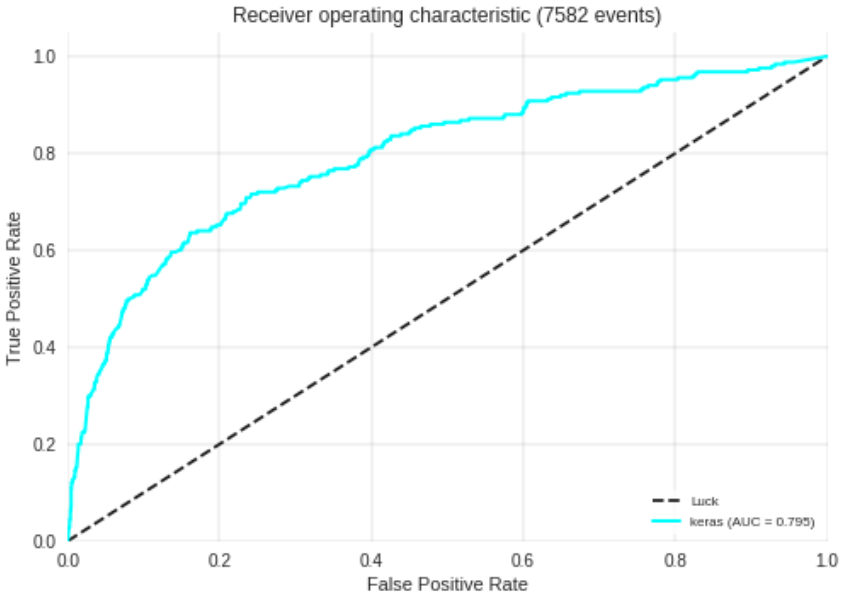


Without resampling and scaling applied



Resampling technique applied

- SMOTE method applied + robust scaling
  - Ratio set to 'minority'
  - Kind set to 'svm'
  - m_neighbors set to 3



- Works by creating synthetic samples from the minor class instead of creating copies.
- The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances

Combine Good & Bad sites: CV ROC plots

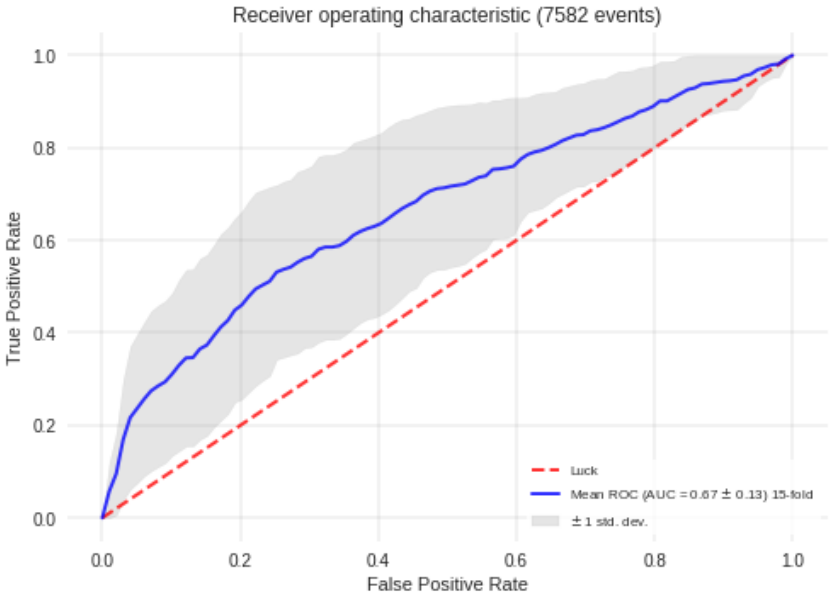
Cross-validated (15-fold) ROC curves to check model prediction variability given the amount of data used to train the model

Additionally,

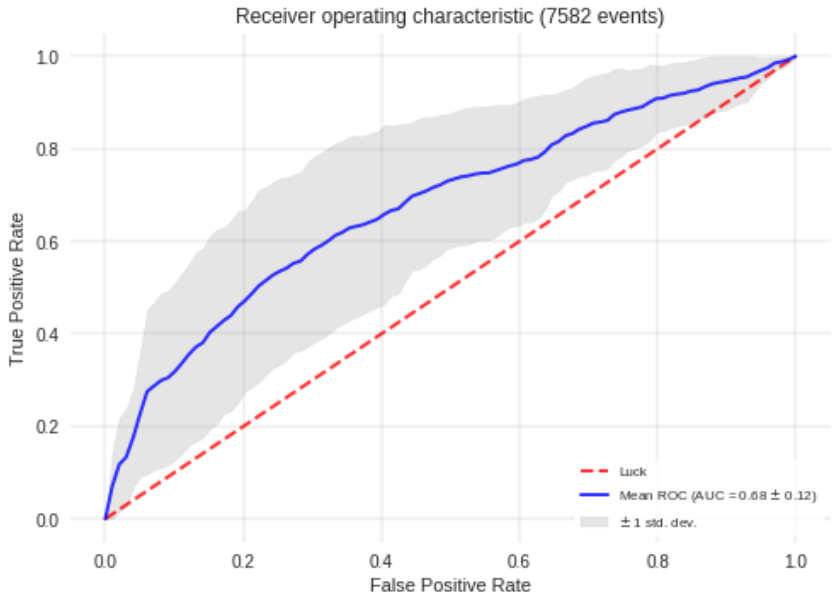
- Applied feature scaling
- Used data resampling SMOTE method

We observe about a 11-13% variability

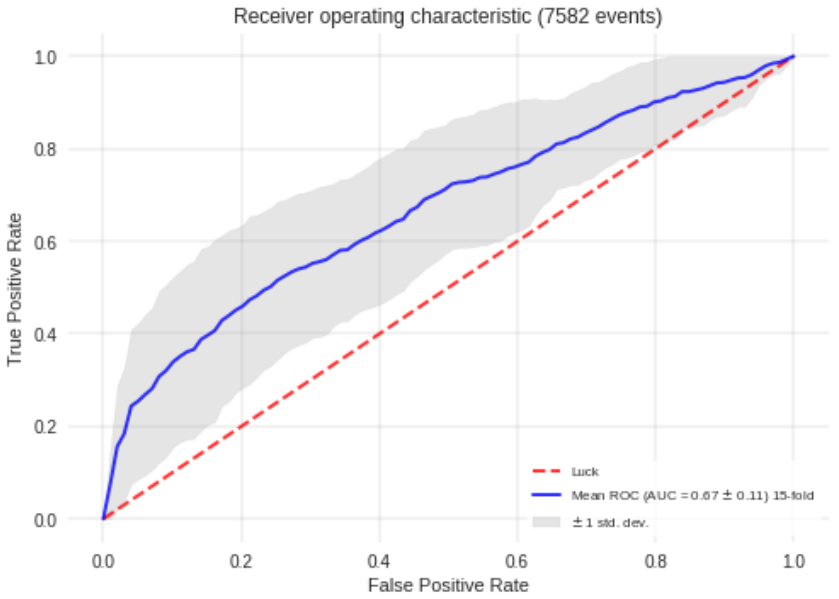
Without scaling or resampling applied



With resampling +scaling applied

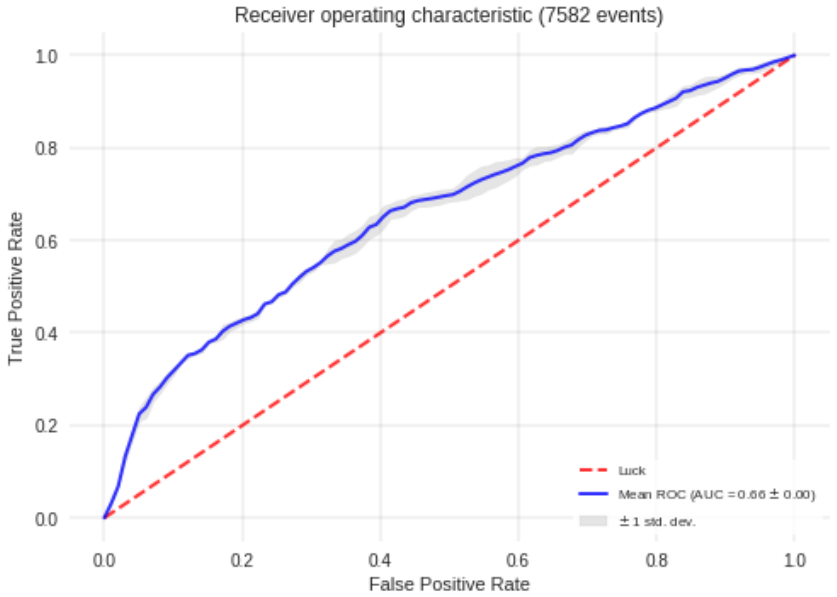


With only scaling applied

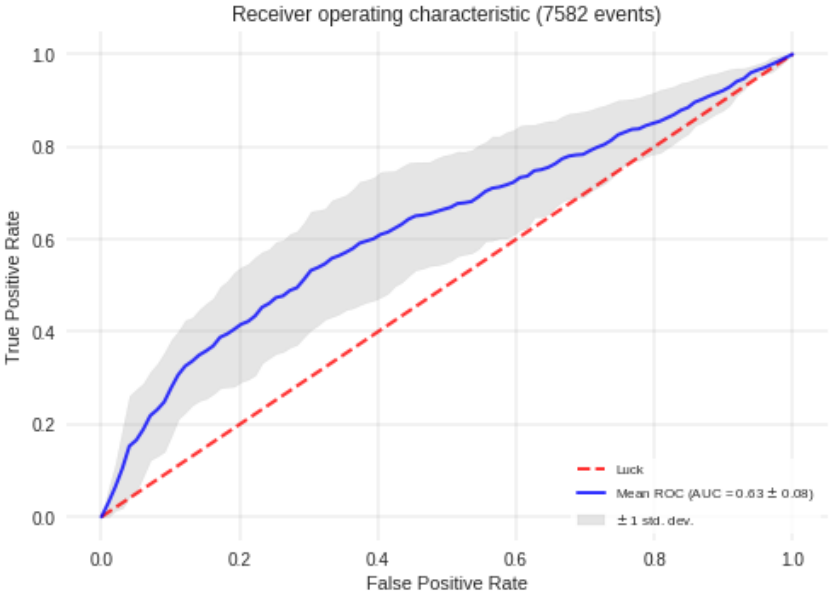


Several K-folds comparison + scaling

- 2-fold

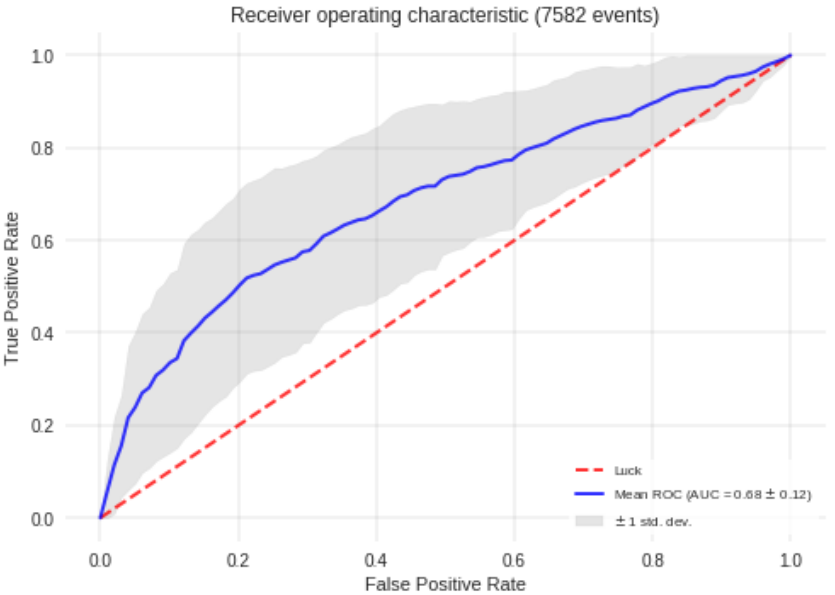


- 5-fold



- 10-fold

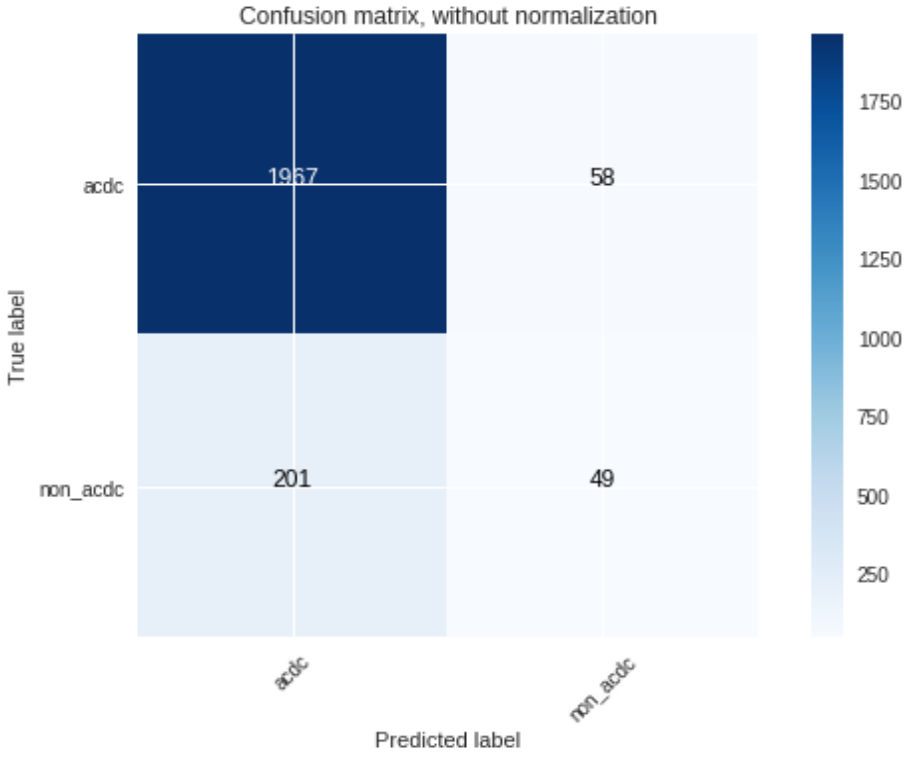
- 15-fold



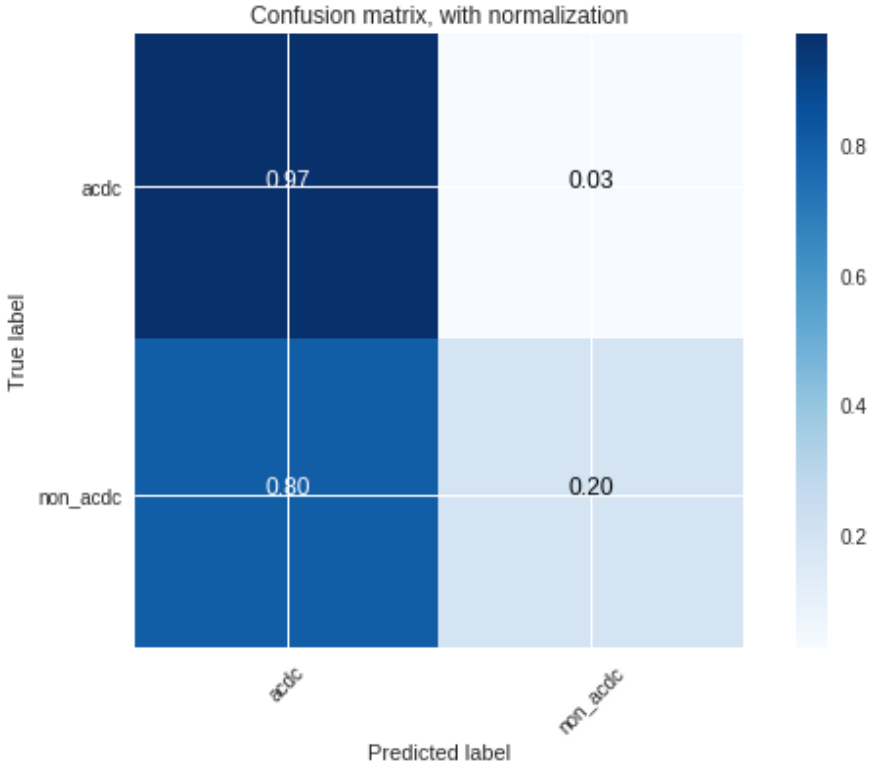
Combine Good & Bad sites: Confusion matrix

- Model performance on test dataset

Un-normalized



Normalized



Model performance across several metrics

- Accuracy based on stratified training data yields ~0.8

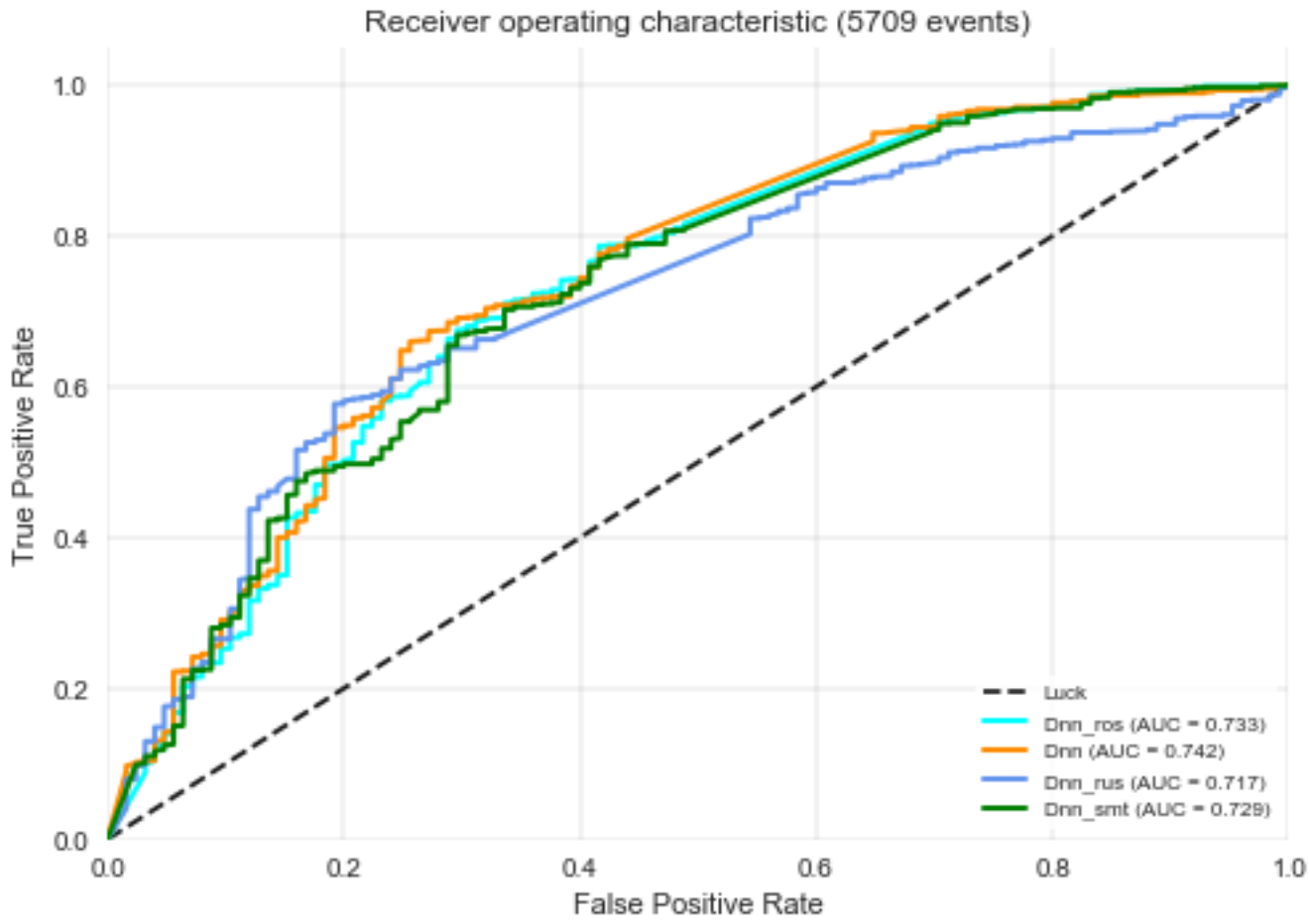
Model Name	Test Log-Loss Score	Test Accuracy Score	Test Recall Score	Test Precision Score	Train Log-Loss Score	Train Accuracy Score	ROC AUC
Good sites	0.403	0.88	0.88	0.86	0.229	0.927	0.745
Bad sites	0.306	0.89	0.89	0.87	0.303	0.901	0.759
Combined good+bad sites	0.395	0.89	0.89	0.86	0.284	0.917	0.80

- Use the log-loss metric for hyper-parameter tuning in binary classification since it heavily penalizes confident prediction but were wrongly classified.
- **Log Loss** takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model.

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1-y_i)\log(1-p_i)].$$

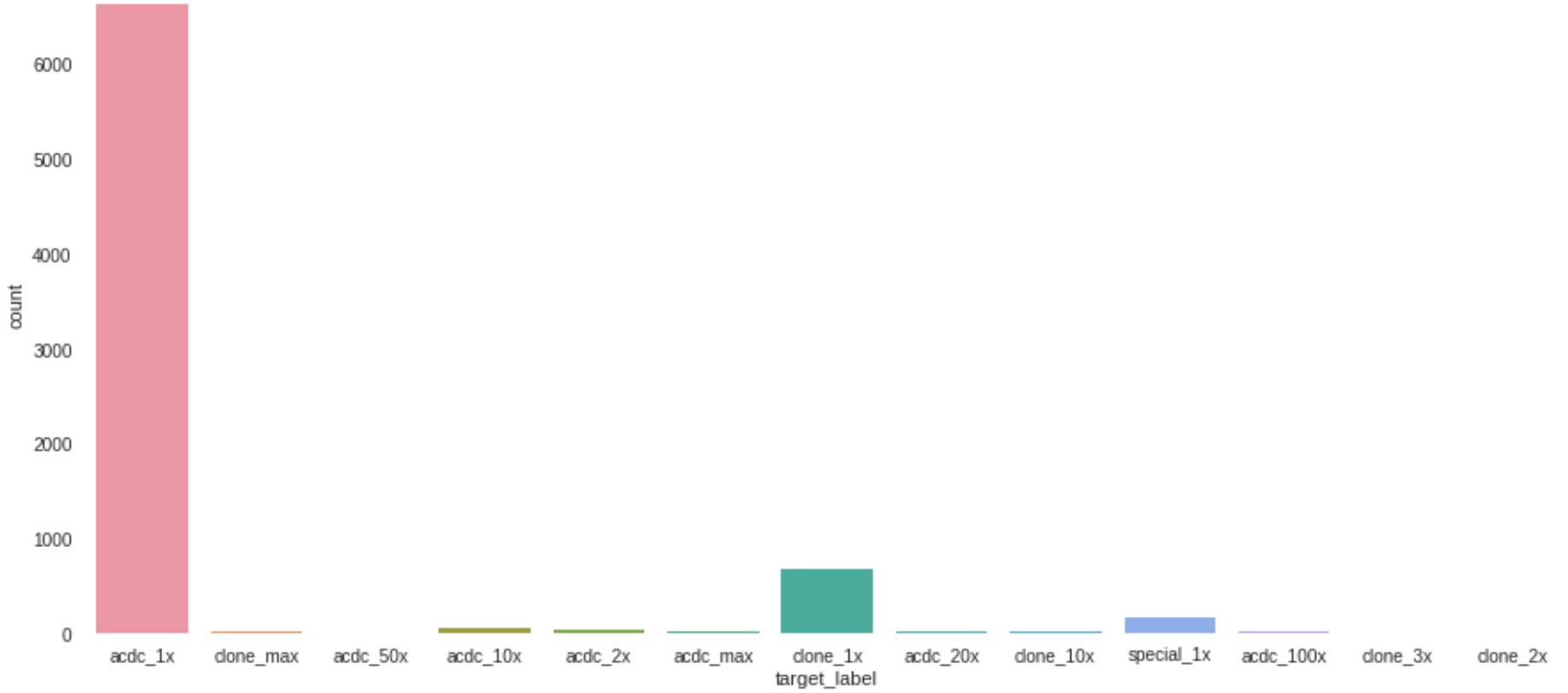
Comparing Machine Learning Models ROC performance

- Comparing ROC performance between model with resampling applied vs non-resampling



Multi-class Classification approach

- Composition of class distribution based on both action (acdc, clone, special) and split (1x, 2x, etc)
- Huge class imbalance
  - Acdc 1x most frequent, followed by clone 1x, and special 1x

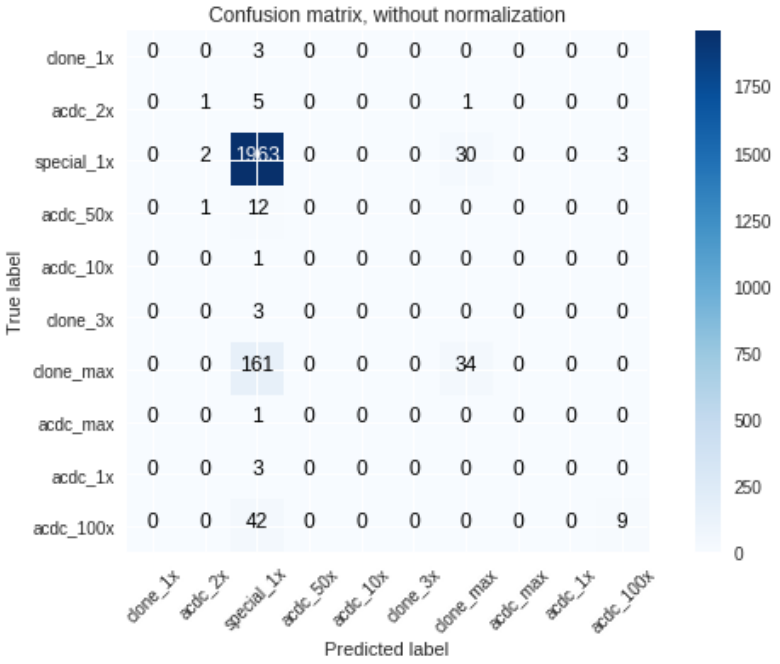


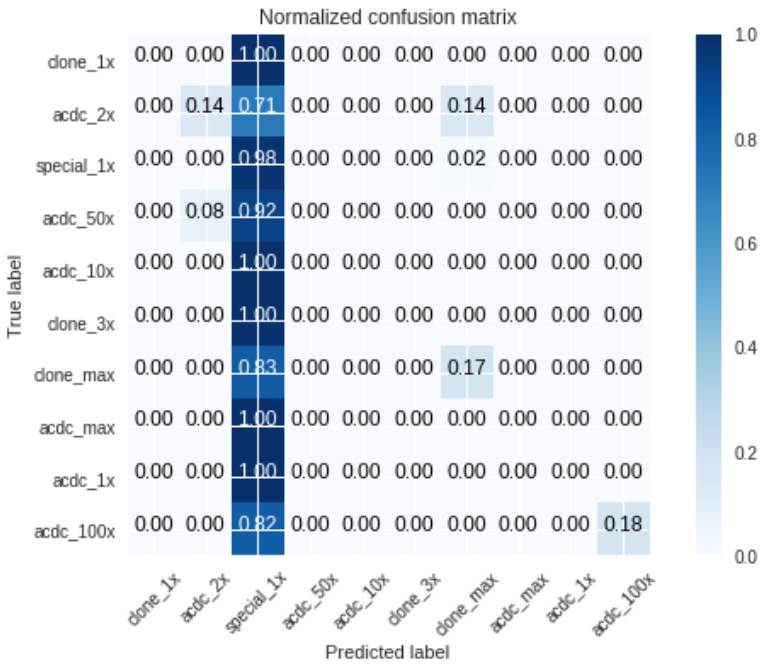
Measure performance for muti-class classification

- Global metrics such as F1-score can be mis-leading
- The confusion matrix is a good way of looking at how good our classifier is performing when presented with new data.
- Provides the yield in true positive, true negative, false positive, and false negative rates.

Confusion Matrix: Multi-class classification

- **Left:** without normalization
- **Right:** with normalization

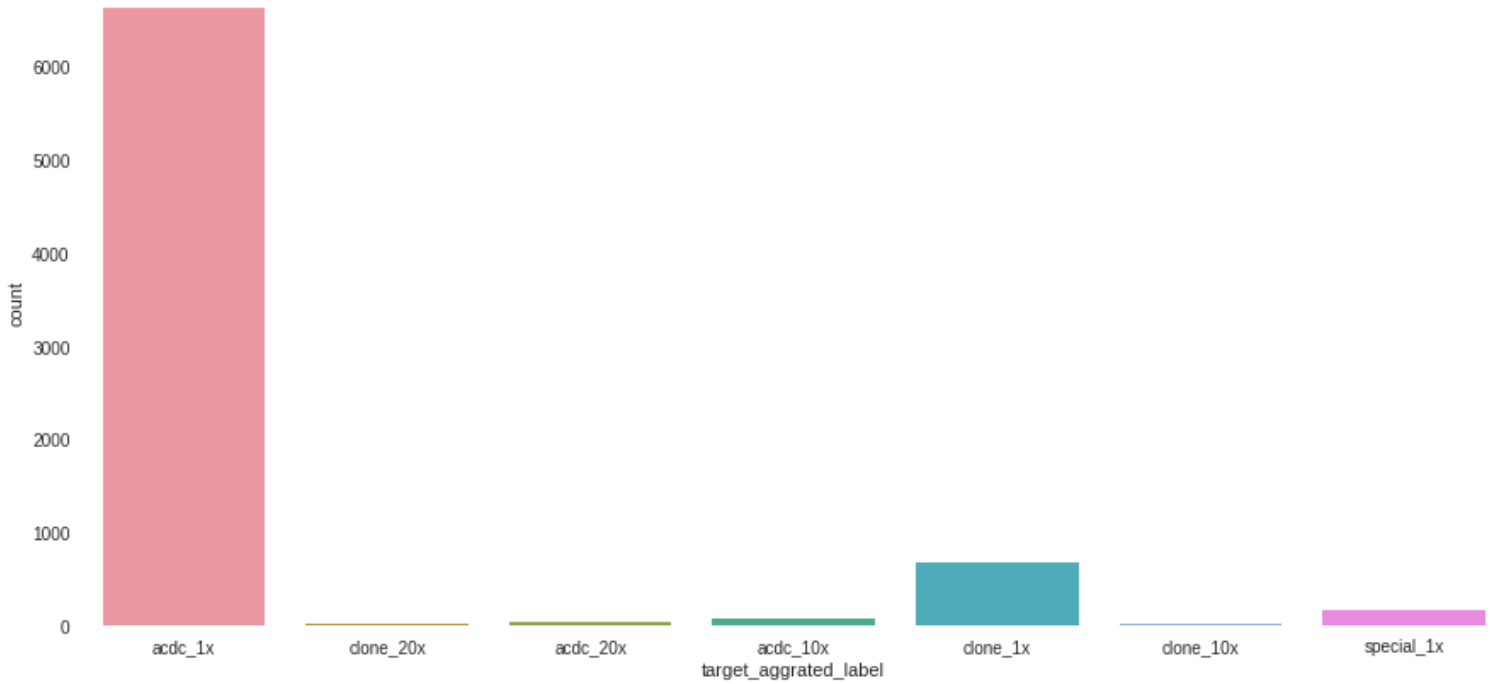




Mislabeled (acdc 1x most prevalent)

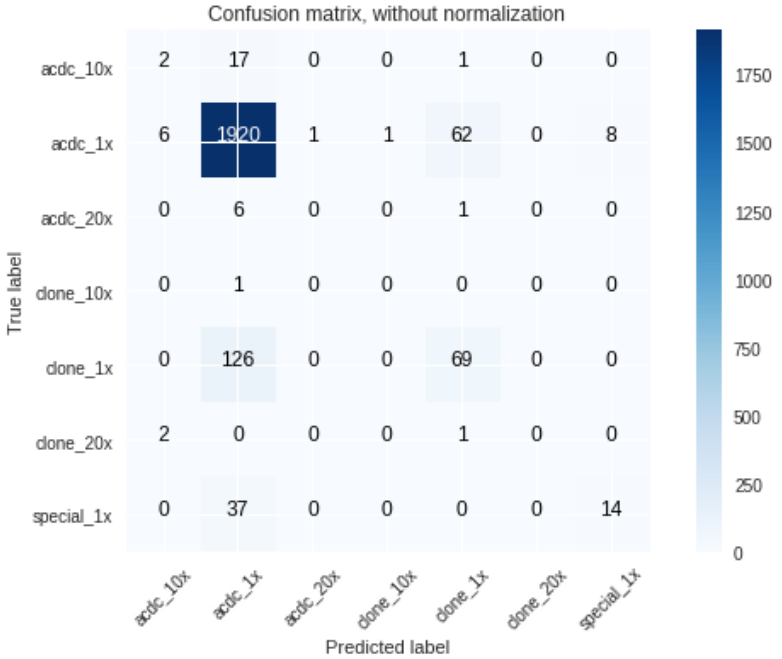
Aggregated Multi-class labels

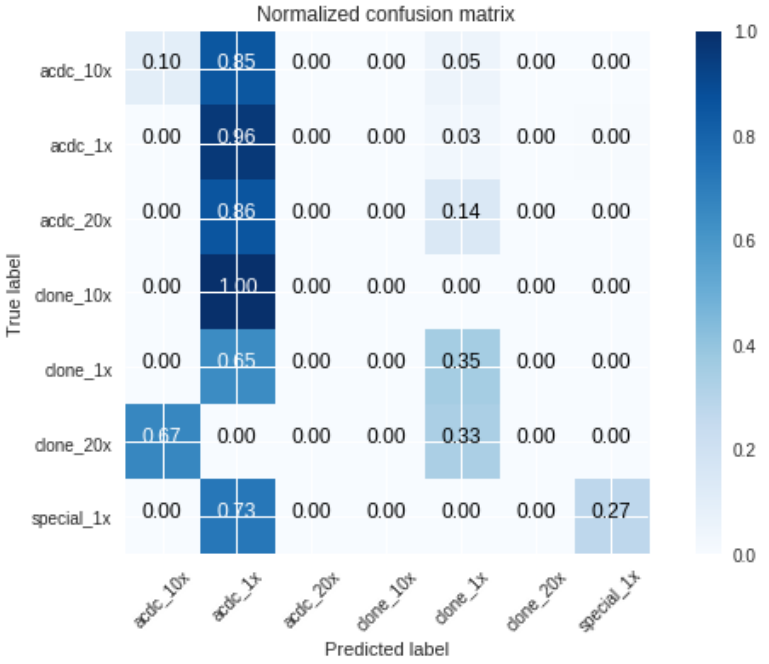
- Classes with 20x label or higher group as one class



Aggregated Multi-target labels

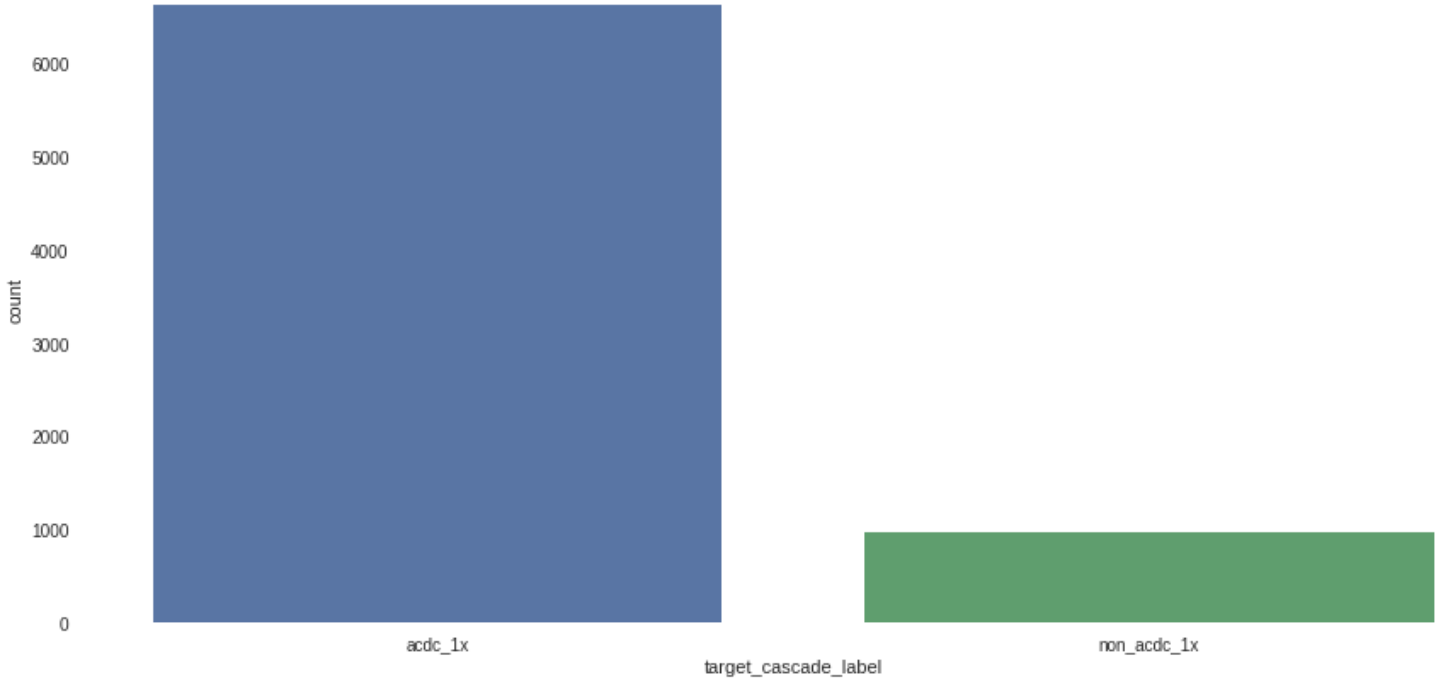
- **Left:** without normalization
- **Right:** with normalization





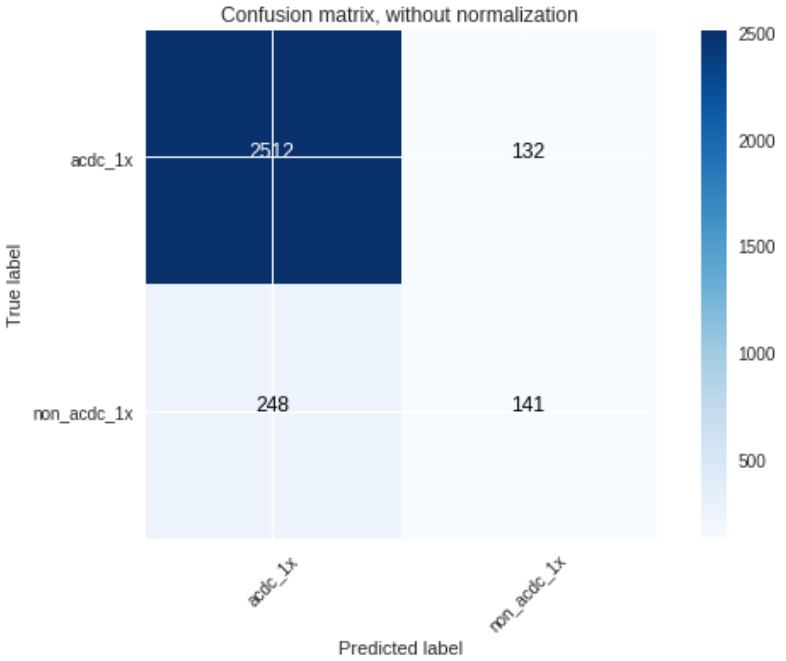
Cascade modeling approach

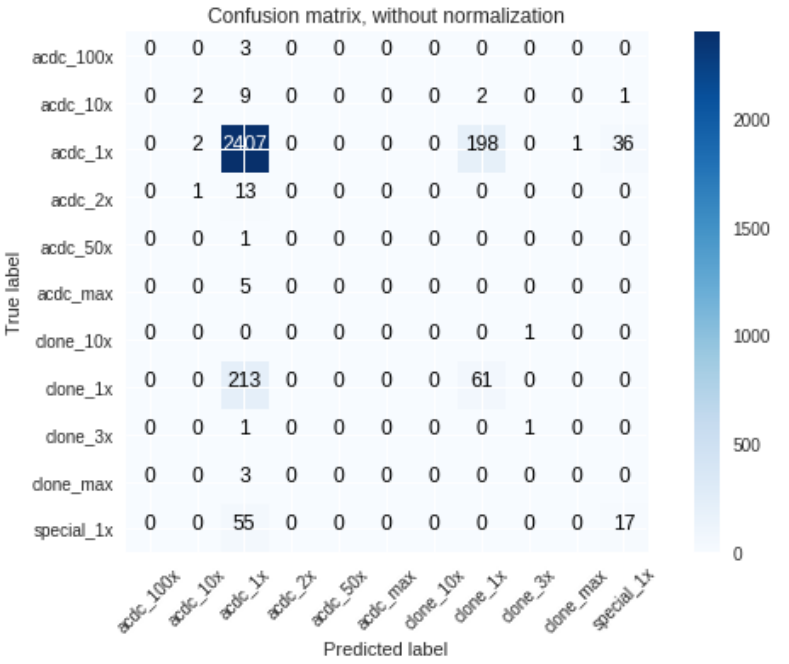
- We study a two stage approach
  - 1st model employes binary classification between acdc 1x and non-acdc 1x targets
  - 2nd model relies on a multi-class classification of all class targets
- Possibly improve performance by accounting for class imbalance



Cascade model: confusion matrix

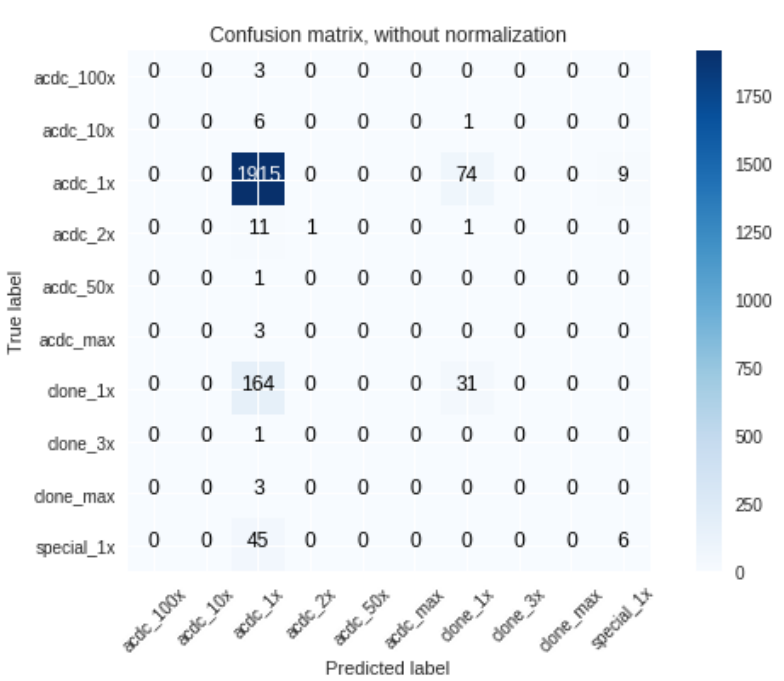
- A total 273 tasks available to train the 2nd model
- **Left:** binary classification model
- **Right:** multi-class classification model

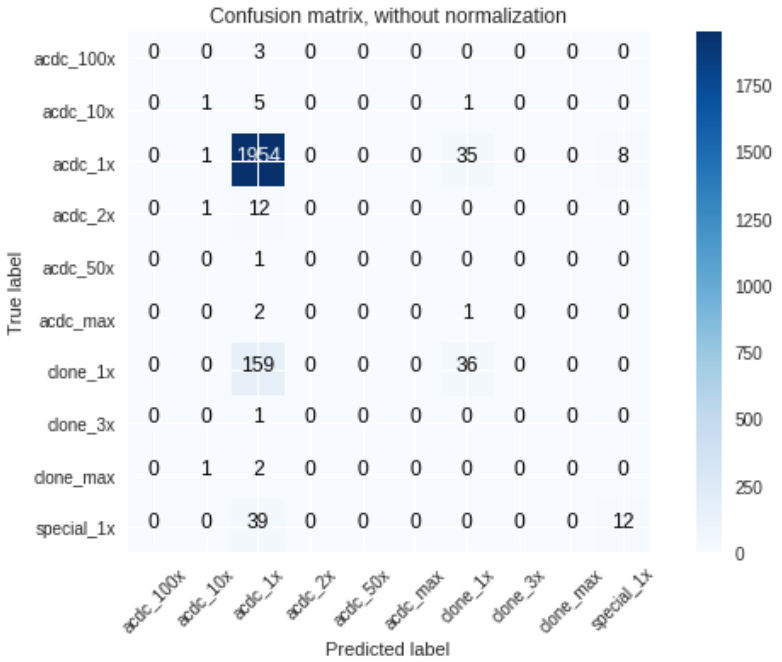




Model comparisons

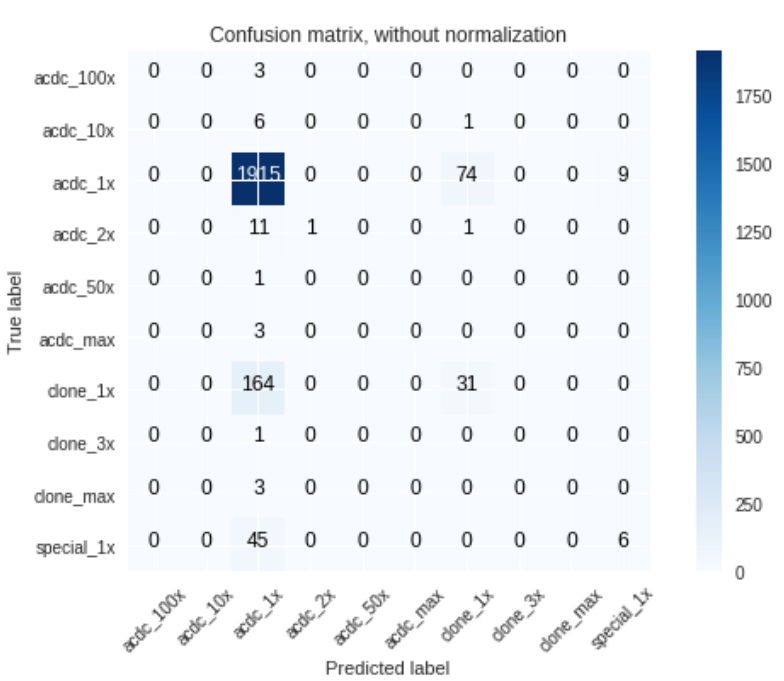
- Compare between original and cascade multi-class classification approach
- **Left:** based on cascade modeling
- **Right:** original baseline approach
- We observe some some improvement in predicting the less frequent classes

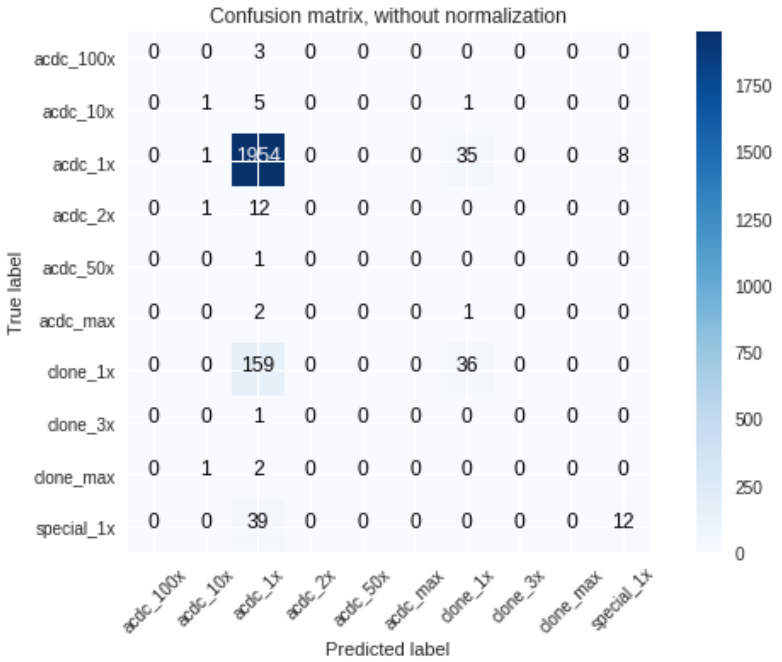




Model comparisons

- Compare between original and cascade multi-class classification approach
- **Left:** based on cascade modeling
- **Right:** original baseline approach
- We observe some some improvement in predicting the less frequent classes



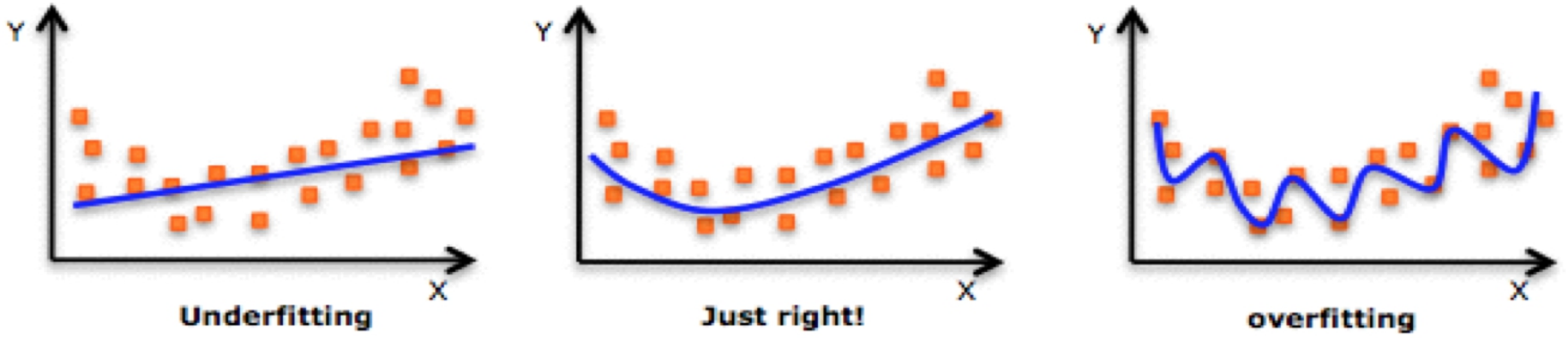


Why hyper-parameter tuning?

- Machine learning models involve careful tuning of learning parameters & algorithm hyper-parameters

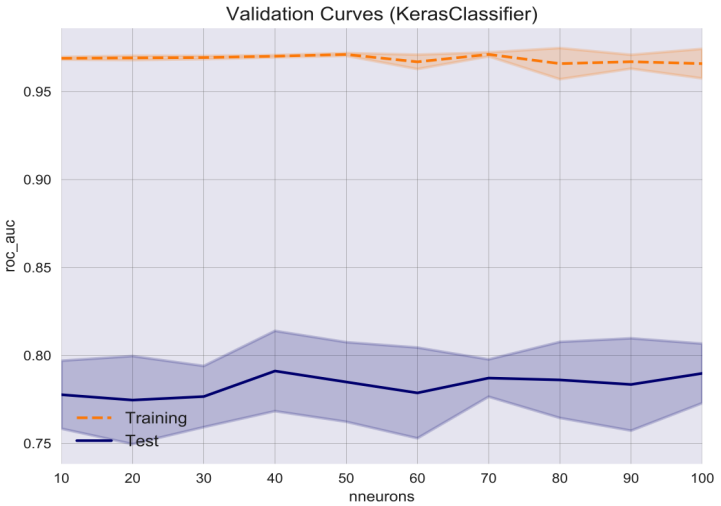
- This tuning is often a “black art” requiring
    - Experience, rules of thumb, or sometimes brute-force search

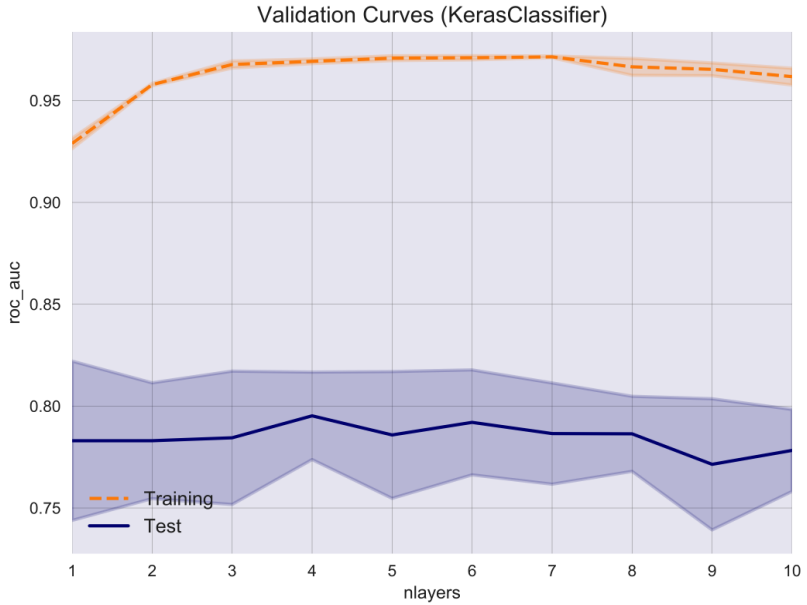
- Tuning prevent **under- or over-fitting** a model
    - The purpose is to generalize well to new data

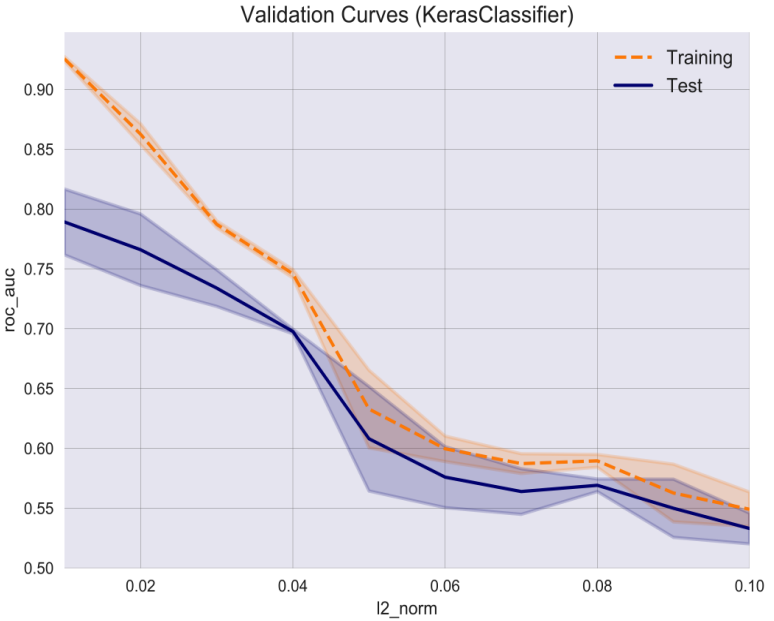


Parameters available to tune

- Neural network have several parameters available to optimize
- AUC ROC used the objective function
  - Performance is cross validated (error band)
  - Compare variational gap between training and test curves







Bayesian optimization

Uses a distribution over functions to build a surrogate model of the unknown function being optimized and then apply some active learning strategy to select the query points that provides most potential interest or improvement

Optimization steps

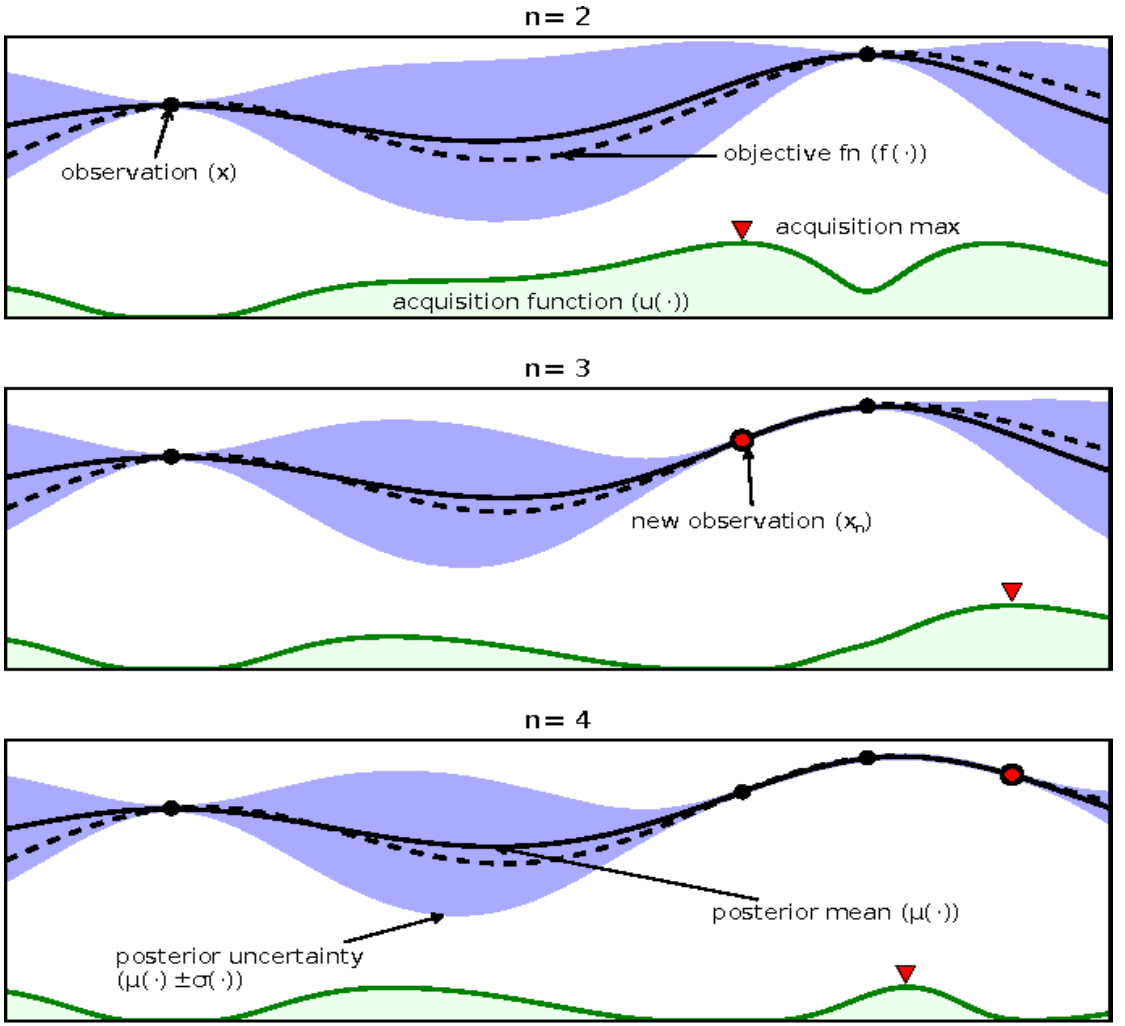
- **Build a probabilist model** for the objective
- **Compute the posterior** predictive distribution
  - Integrate out all the possible true functions
  - Make use of **Gaussian process** regression
- **Optimize a cheap proxy function** instead
  - The model is much cheaper than the true objective

Source: scikit-optimize

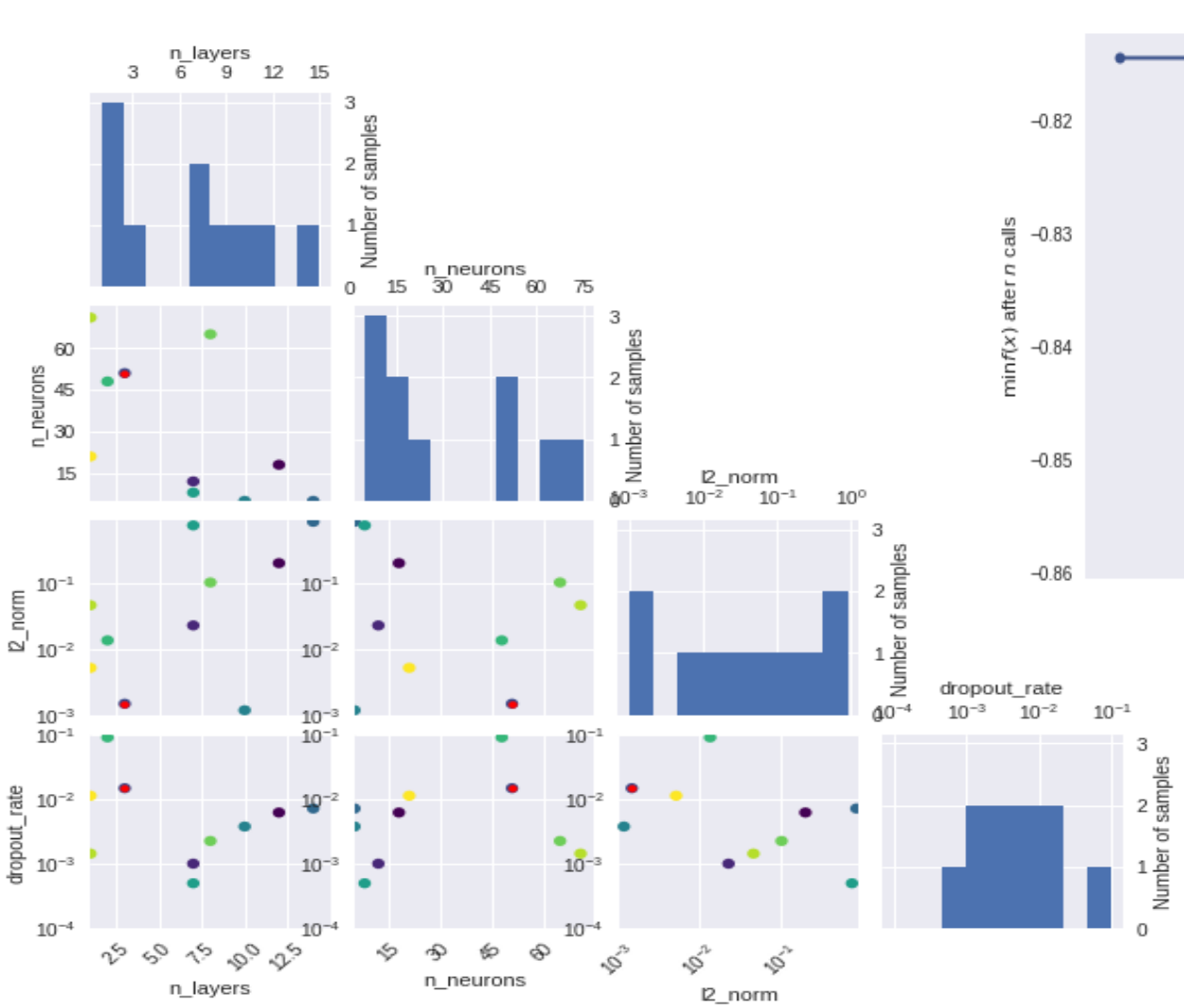
Main insight of Bayesian optimization

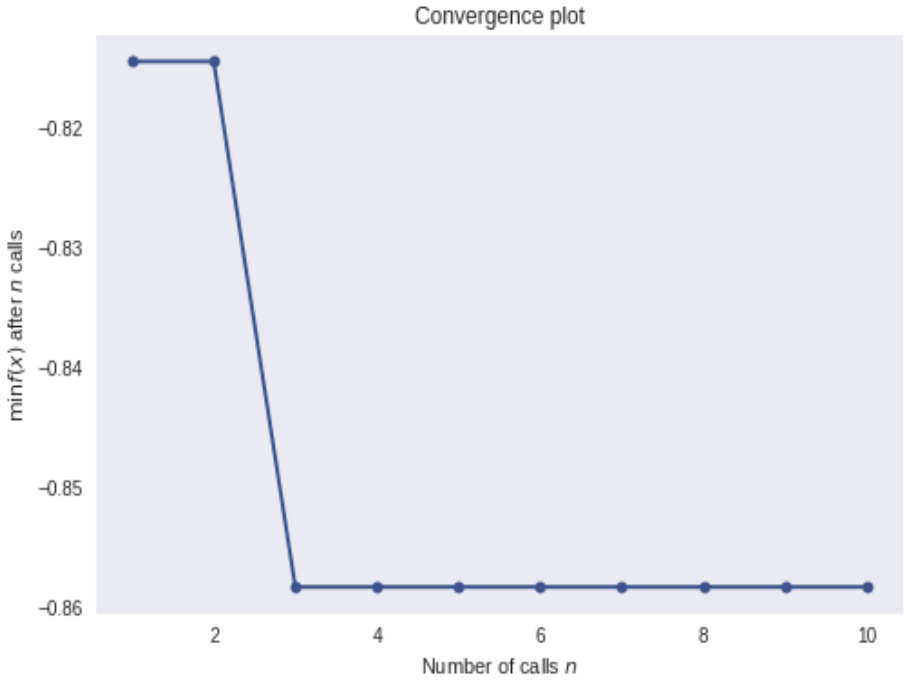
Make the proxy function exploit uncertainty to balance **exploration** against **exploitation**

- **Exploration**: seeks places with high variance
- **Exploitation**: seeks places with low mean



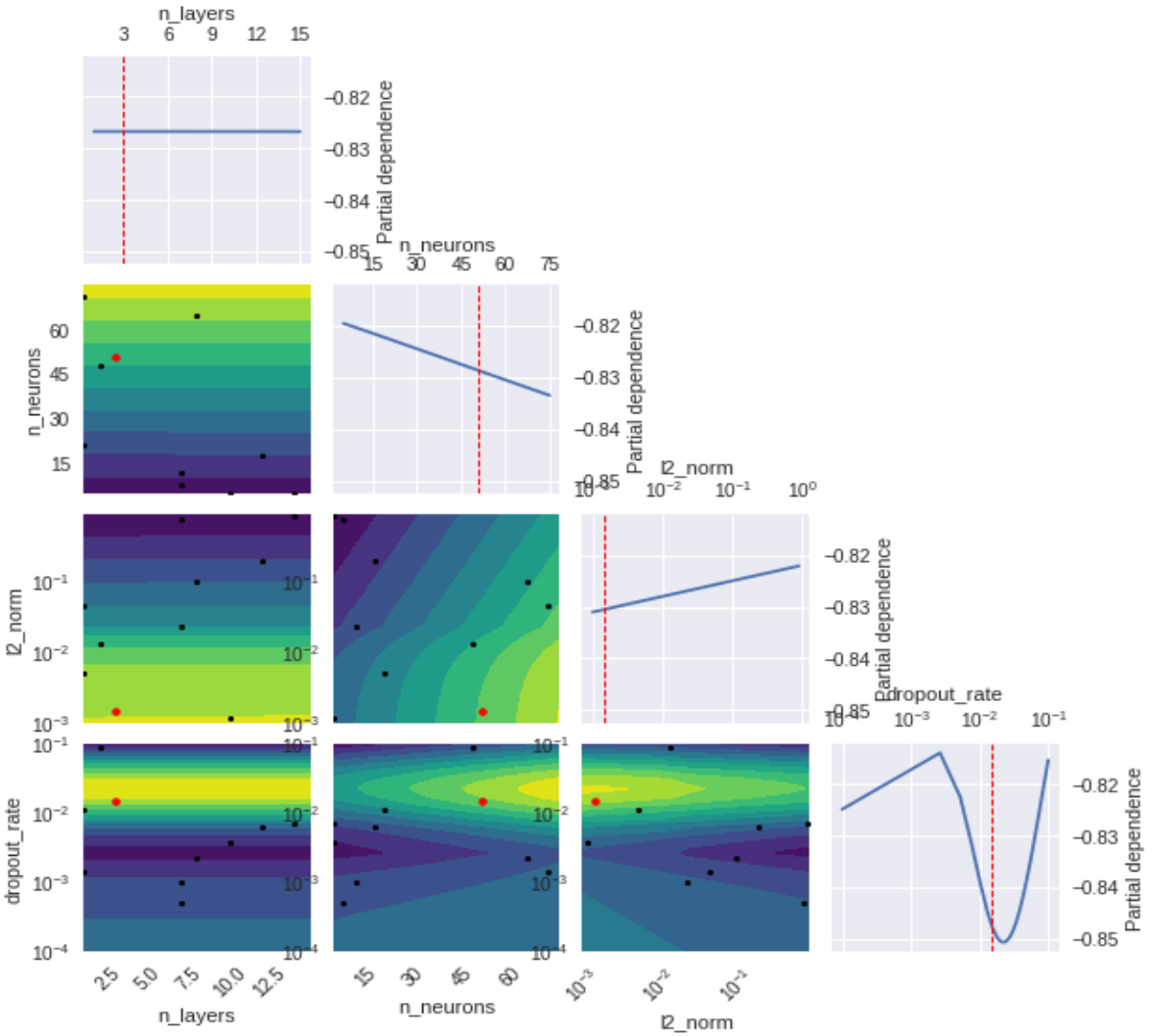
Evaluation & Convergence





Plot of the partial dependence of the objective

- Partial Dependence plots are only approximations (via surrogate model) of the modeled fitness function
  - which in turn is only an approximation of the true objective function in fitness, for each dimension and as pairs of the input dimension
  - Partial dependence plots helped us understand interactions between the variables that is driving the prediction.

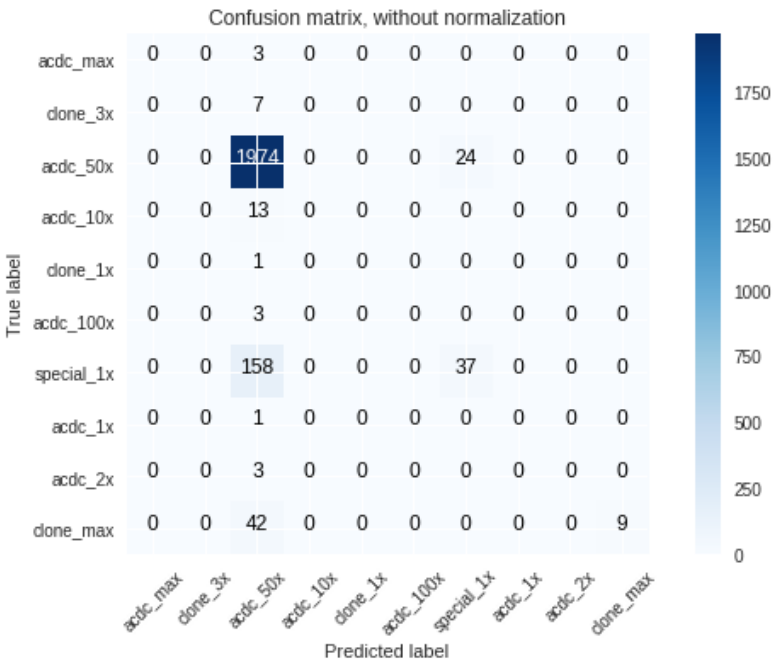


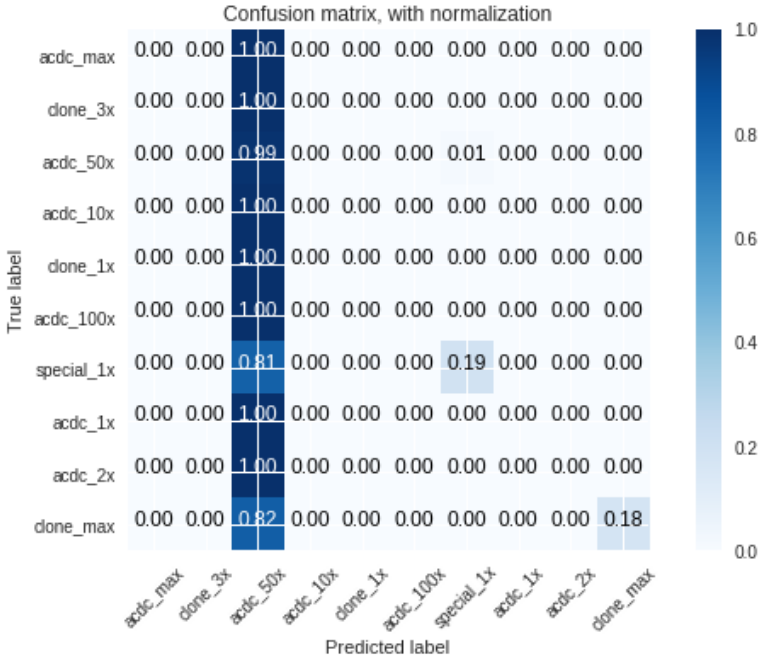
Example of optimal parameters found

- Bayesian optimization using Gaussian Processes using F1-score (weighted) as the objective function
- Best Expected Improvement score = 0.5352
- Expected Improvement (EI) best parameters:
  - Number layers = 3
  - Number neurons = 51
  - l2_norm = 0.13475
  - dropout_rate = 0.0051
  -
- **Note:** we apply early stopping if after 5 epochs performance does not improve then we stop training

XGBoost multiclass classification modeling

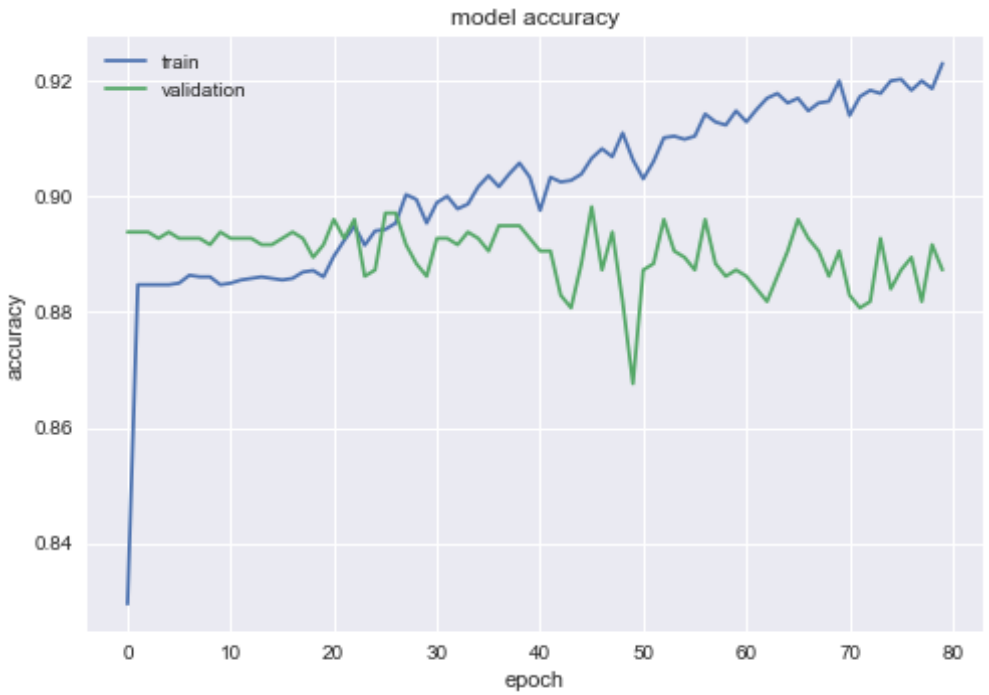
- Number of boosting trees: 100
- Max depth of trees: 4
- Objective function: multi:softprob



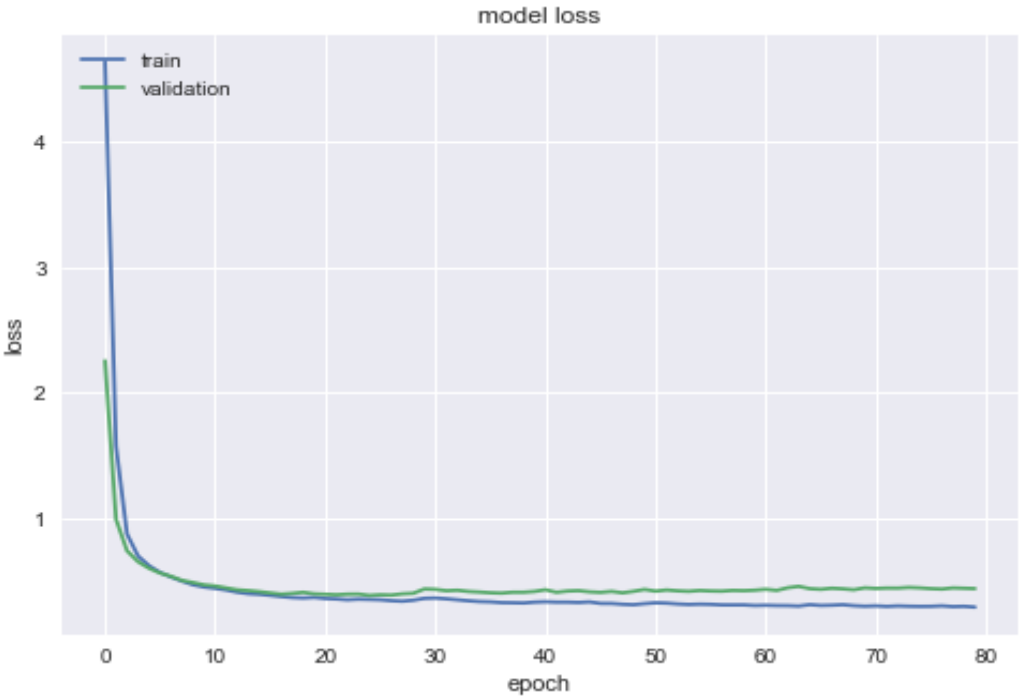


Monitoring training

- Accuracy vs epoch



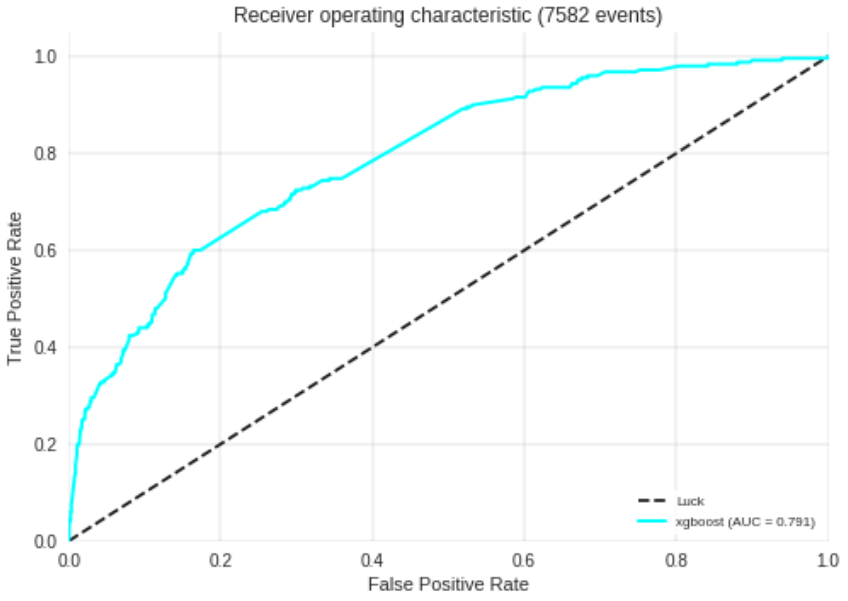
- Log-loss vs epoch

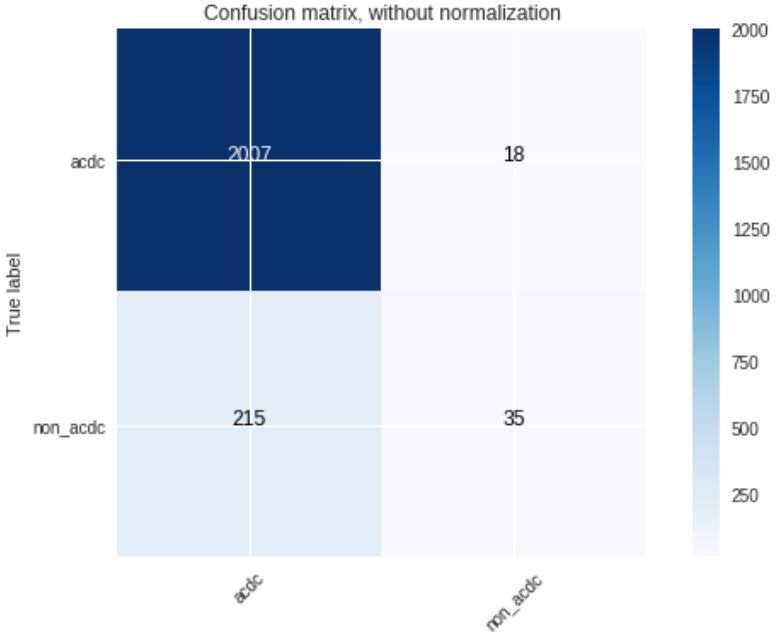


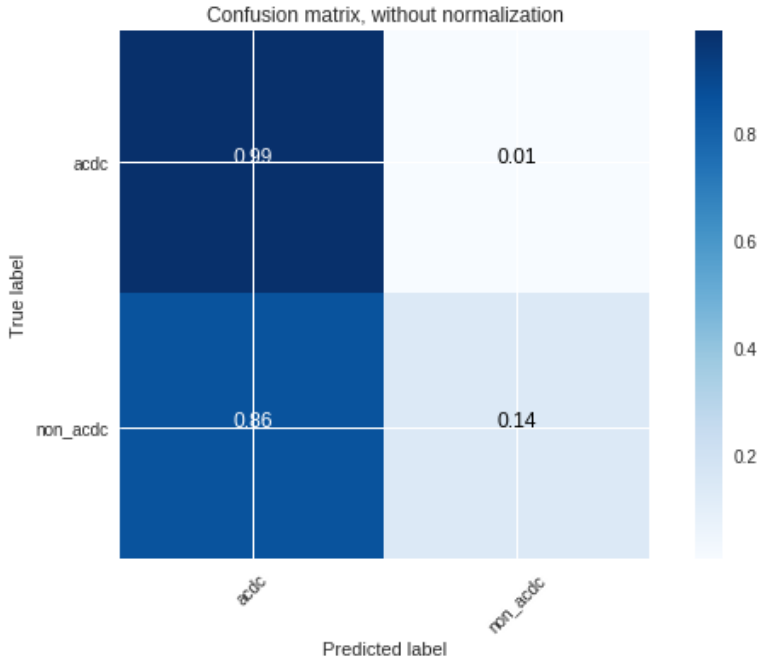
- The model will train until the validation score stops improving
- Validation error needs to decrease at least every 5 epoch rounds to continue training to avoid over-fitting to the training data

XGBoost binary classification modeling

- Number of boosting trees: 100
- Max depth of trees: 3
- Objective function: binary:logistic

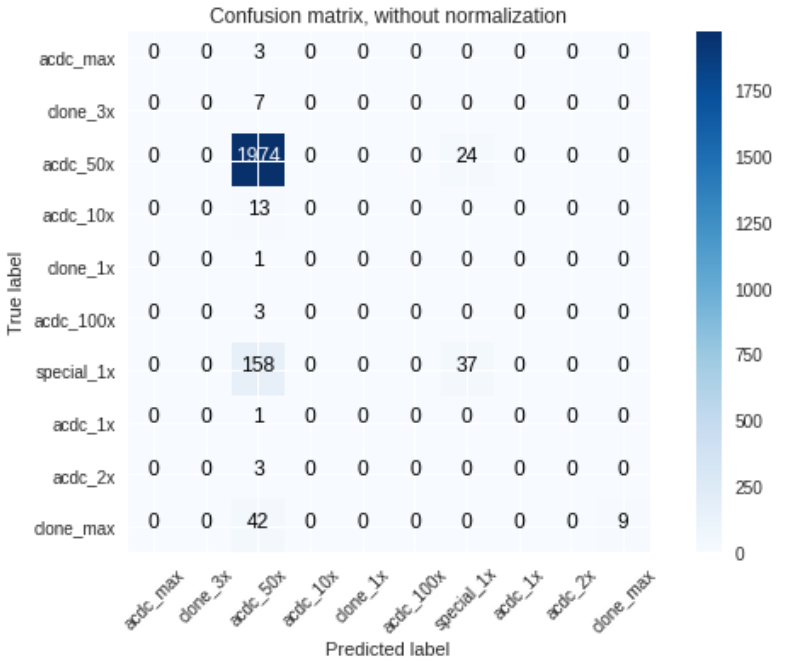


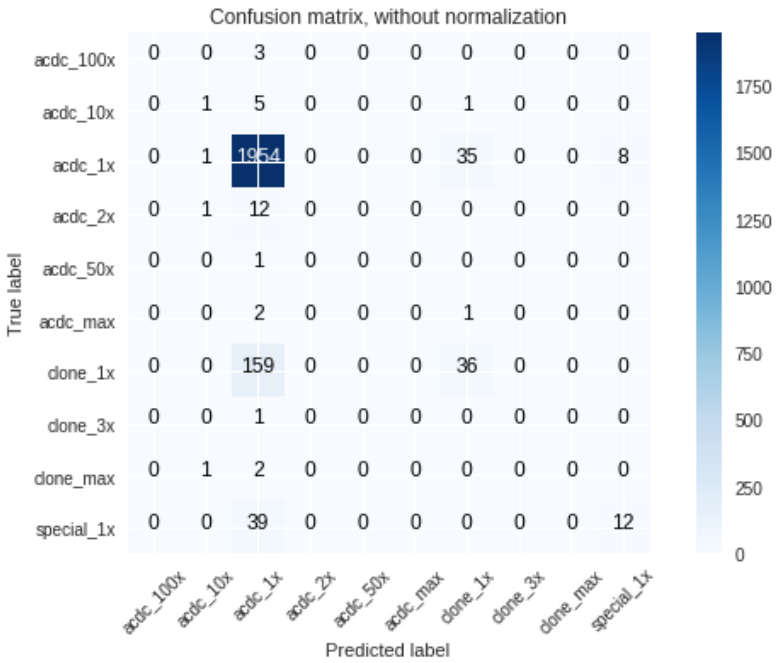




Compare XGBoost and DeepNet multi-class classification performance

- **Right:** original multi-class performance
- **Left:** XGBoost multi-class performance





Summary

Discussed today

- A first pass for the supervised learning, we have been building a sparse matrix of errors for error codes and sites, flattening these matrices, and then sending them through various ML packages.

- Explore multi-class classification for action with the various corresponding sub-options
    - Further parameters to target on are:
        Job splitting, enabling XrootD, and requested memory
- Challenge might be end up with too little examples for certain labels to train on

Future plans

- Looking into include additional information into the modeling such as campaign information
- Comparing models studied so far against more advanced architecture such as Long-Short Term Memory (LSTM) recurrent neural networks

Summary

Discussed today

- A first pass for the supervised learning, we have been building a sparse matrix of errors for error codes and sites, flattening these matrices, and then sending them through various ML packages.

- Explore multi-class classification for action with the various corresponding sub-options
    - Further parameters to target on are:
        Job splitting, enabling XrootD, and requested memory
- Challenge might be end up with too little examples for certain labels to train on

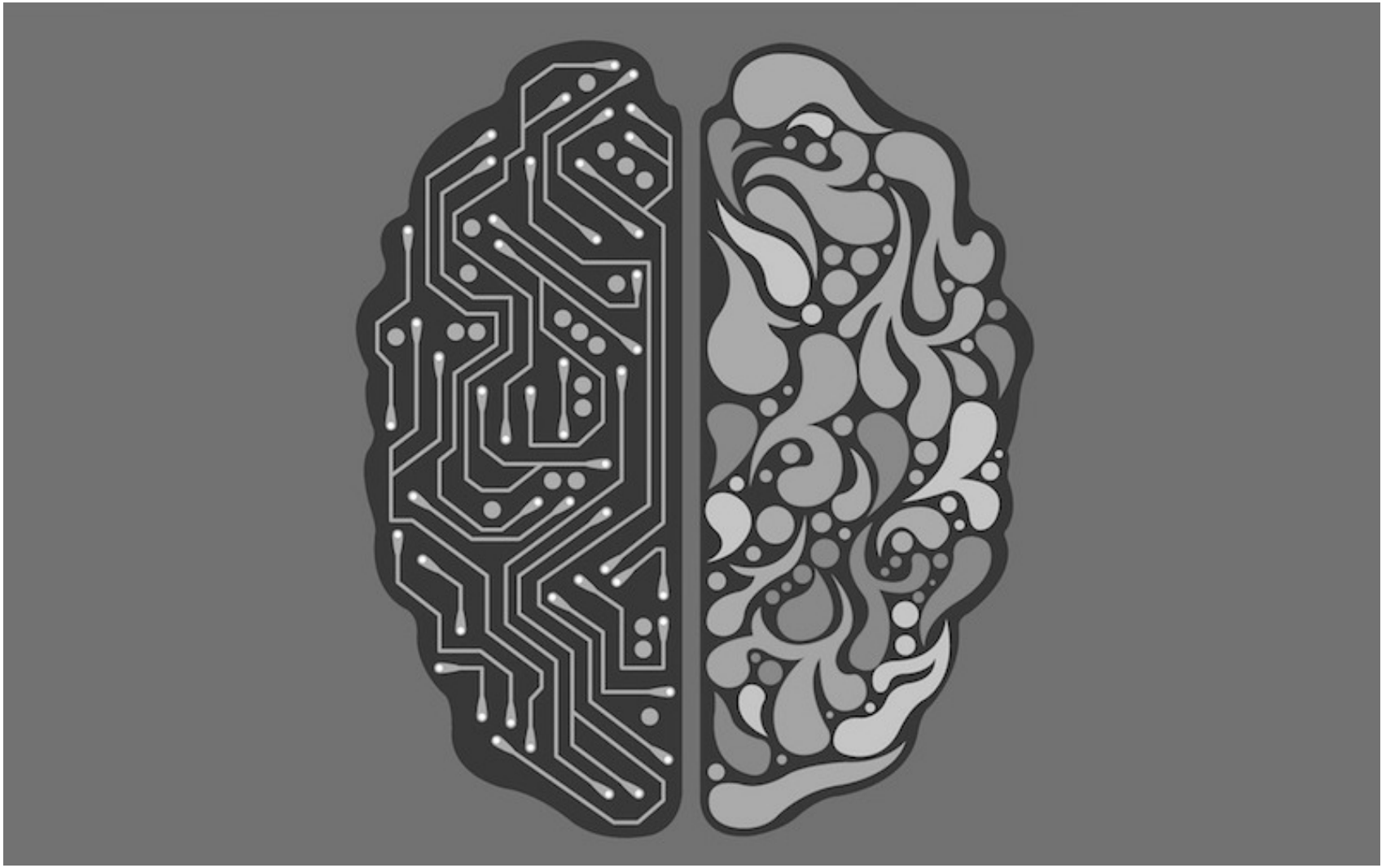
Future plans

- Looking into include additional information into the modeling such as campaign information
- Comparing models studied so far against more advanced architecture such as Long-Short Term Memory (LSTM) recurrent neural networks

Good & Bad site description

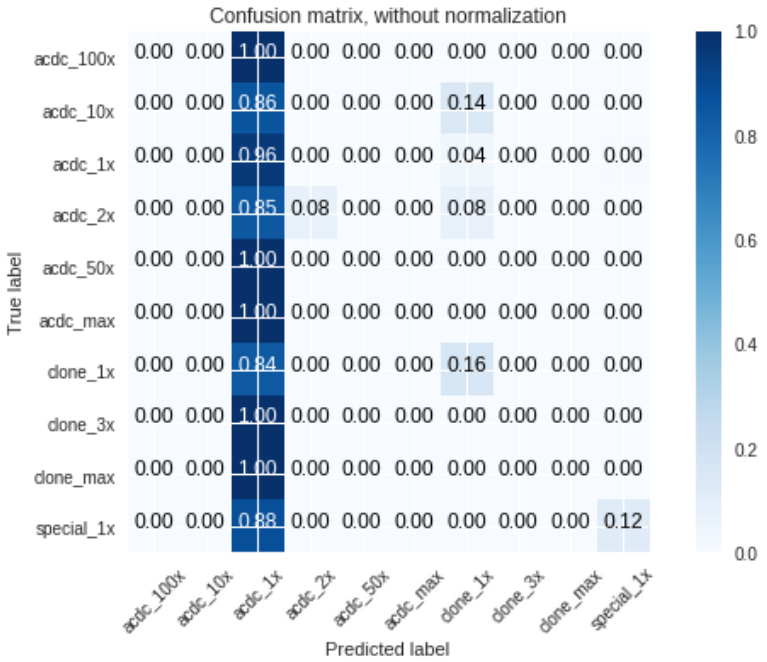
- Good and bad site, refers to the site availability, maintained by the site support team, at the time the workf bw was reported as needing assistance.
- The site support team runs multiple tests over all of the sites.
- SAM and HammerCloud tests are the ones they often report.
- One tests if f les are accessible, and I'm not sure about the other.
- The main thing is that production jobs should not be expected to run successfully at "bad" sites,
  while they should be able to run at "good" sites that have the correct f les.

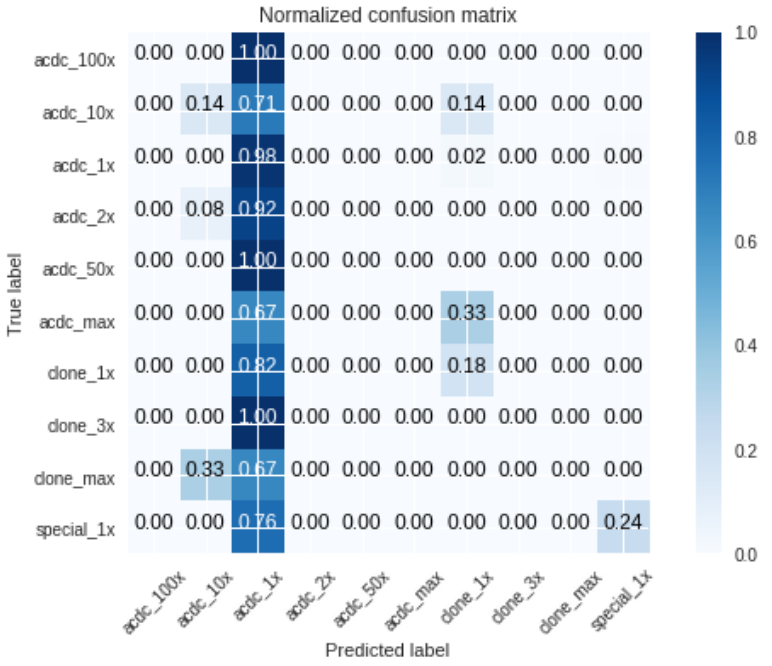
Backup



Comparison between baseline and Cascade model Confusion Matrix performance

- Normalized yields
- Right: original baseline
- Left: cascade model

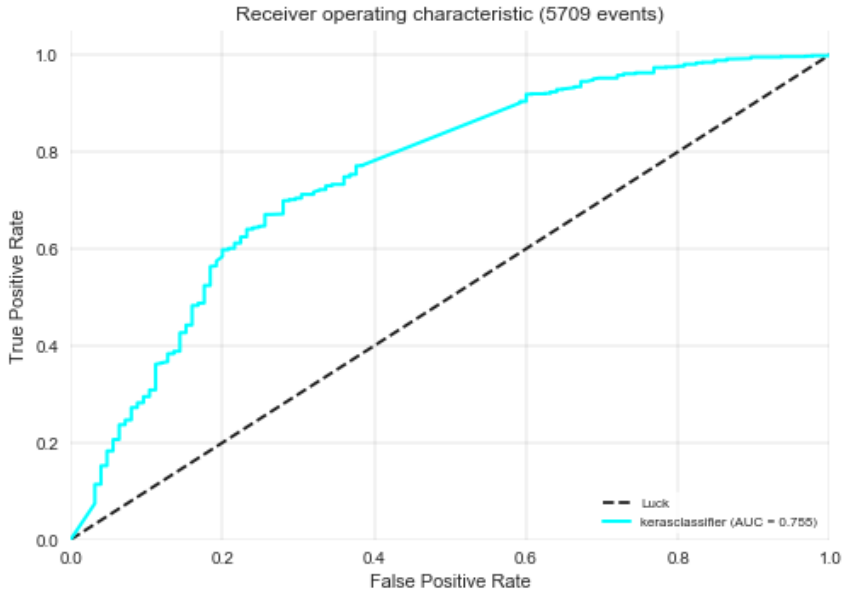


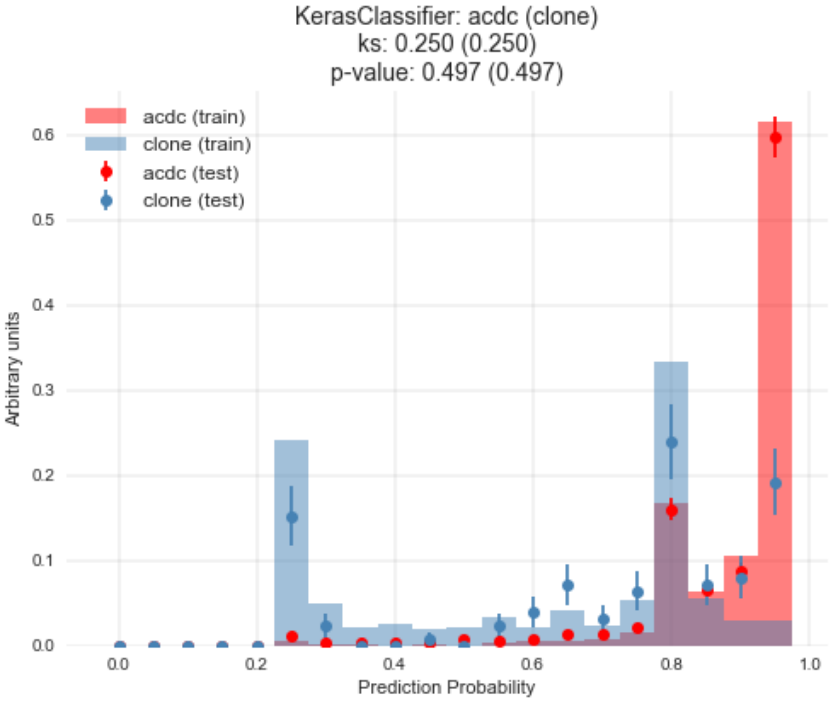


Good sites:

AUC metrics to check model performance

- Trained DNN model on 4500 workflow based on Good Site information
- Evaluated general model performance on the remaining 1200 workflow
  - ROC and Over-training plots
- No optimization techniques applied here, e.g. resampling, and feature scaling

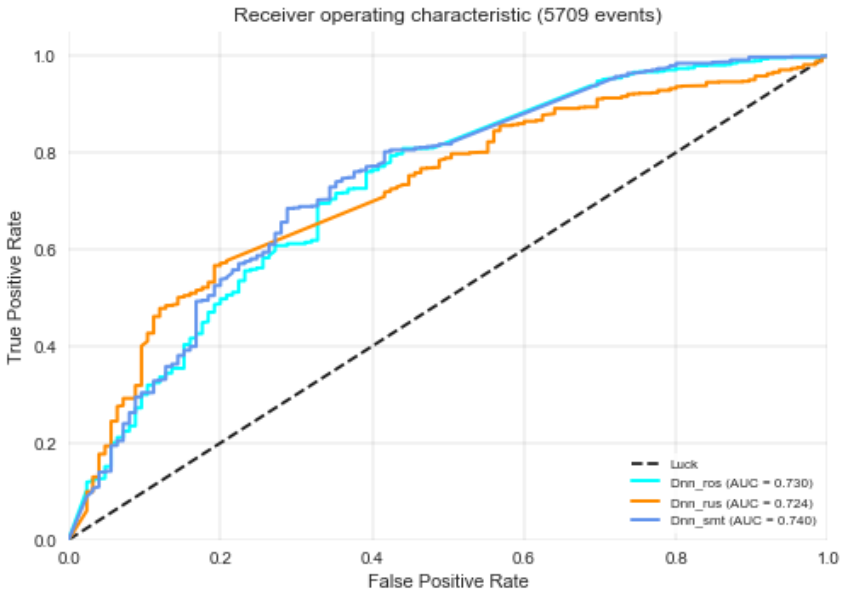




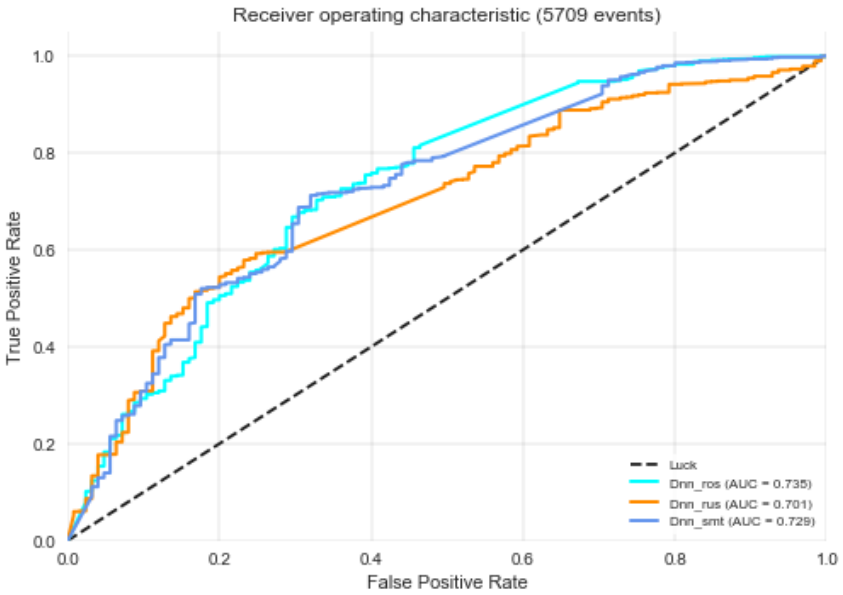
Good sites: compare data resampling methods

- Including feature scaling only marginally improves model performance
- Resampling + feature scaling show slight decrease performance compared to 0.76 (without resampling and scaling)

Resampling and scaling applied



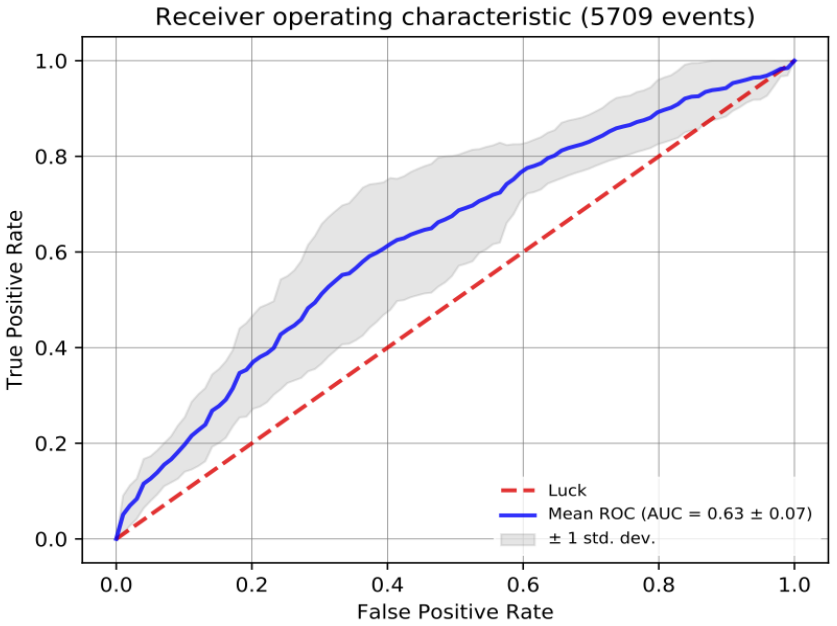
Without scaling applied



Good sites: use SMOTE

- Cross-validated (5-fold) ROC curves to check model prediction variability given the amount of data used to train the model
- Additionally,
  - Applied feature standardization
  - Used data resampling SMOTE method
- A first attempt, already (63±7)% accurate (AUC)
  - can be further optimized, cured for over-training

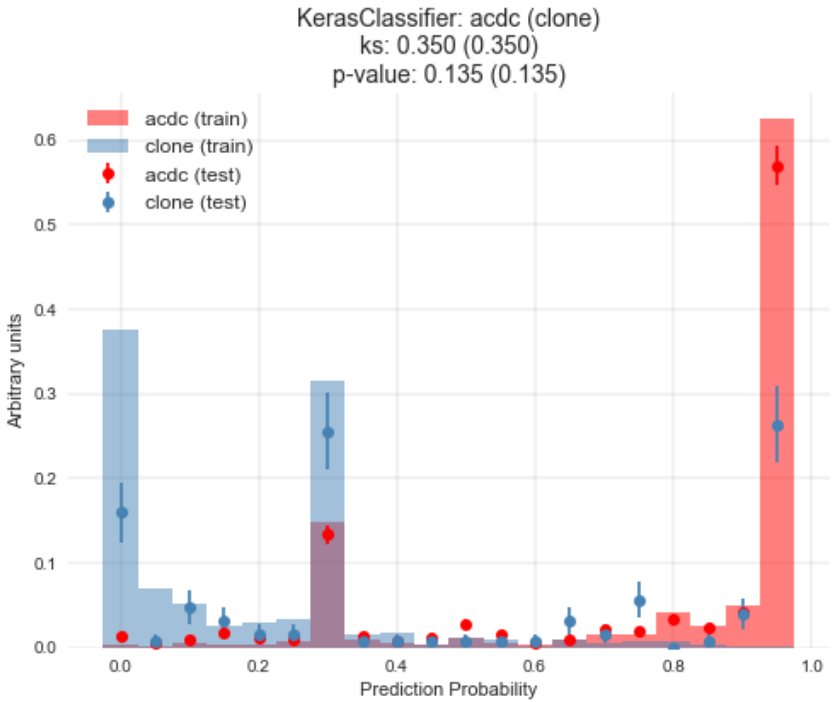
Resampling and scaling applied



Without resampling and scaling applied

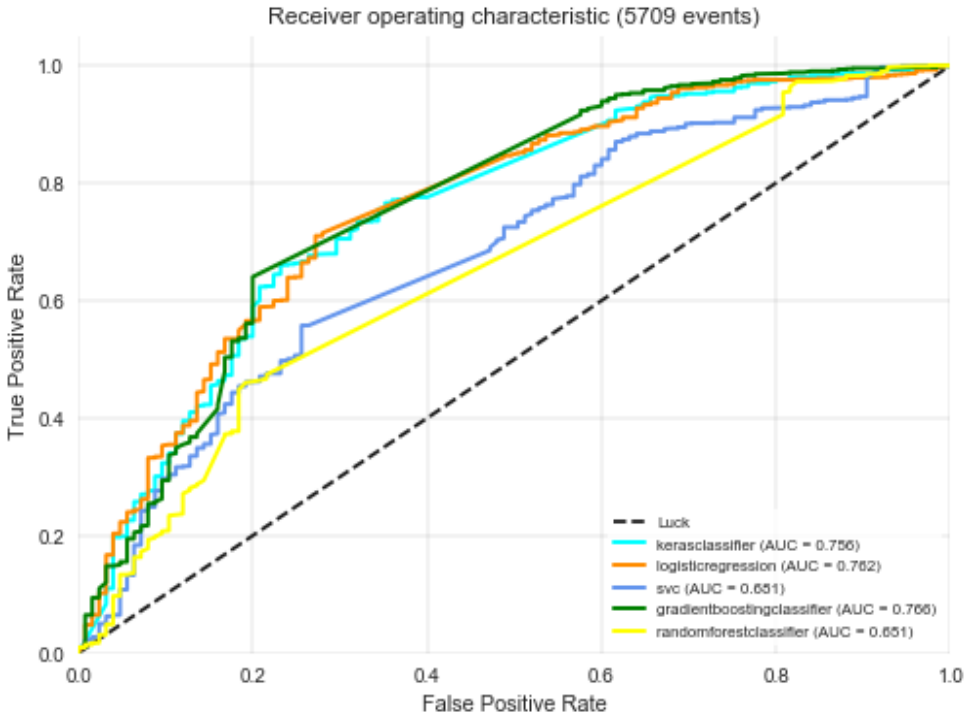
Good sites: user SMOTE

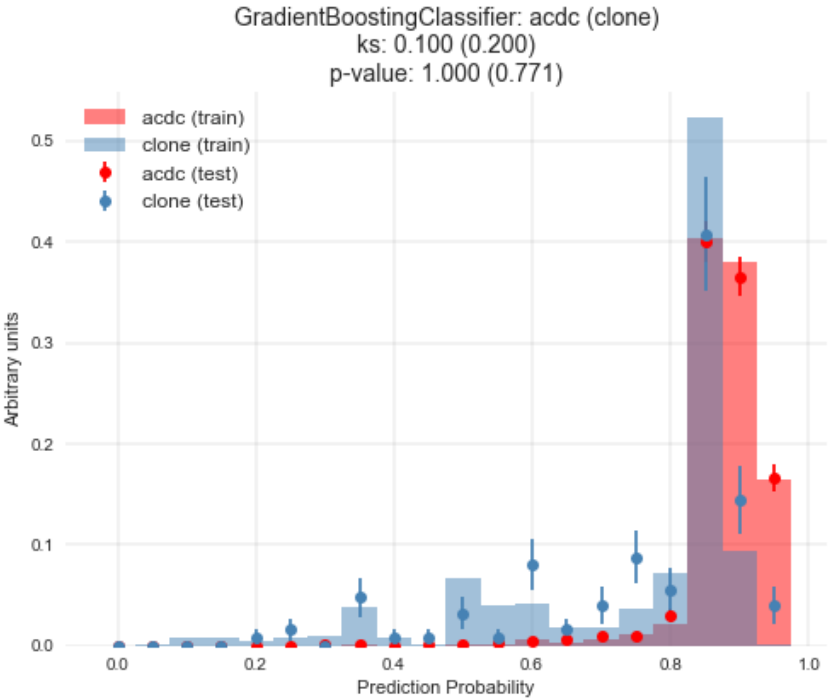
- Resampling with SMOTE method and feature scaling applied
- Noticeable change in discriminant distribution



Classifier comparisons

- Compare several ML algorithms
  - DNN, Logistic Regression, Support vector machine, Gradient boosting, and Random forrest
  - Train them out-of-the box with default settings





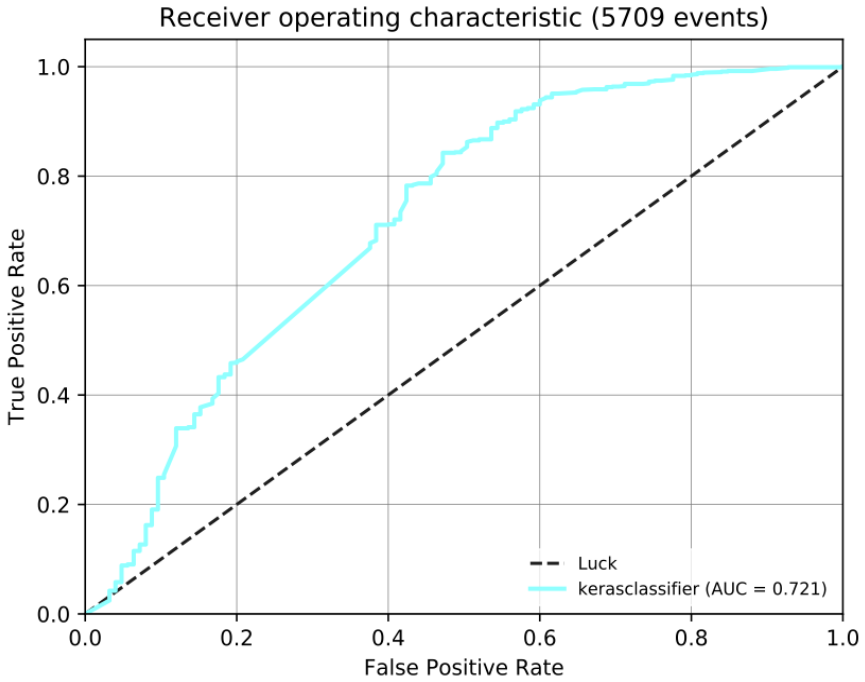
Model evaluation

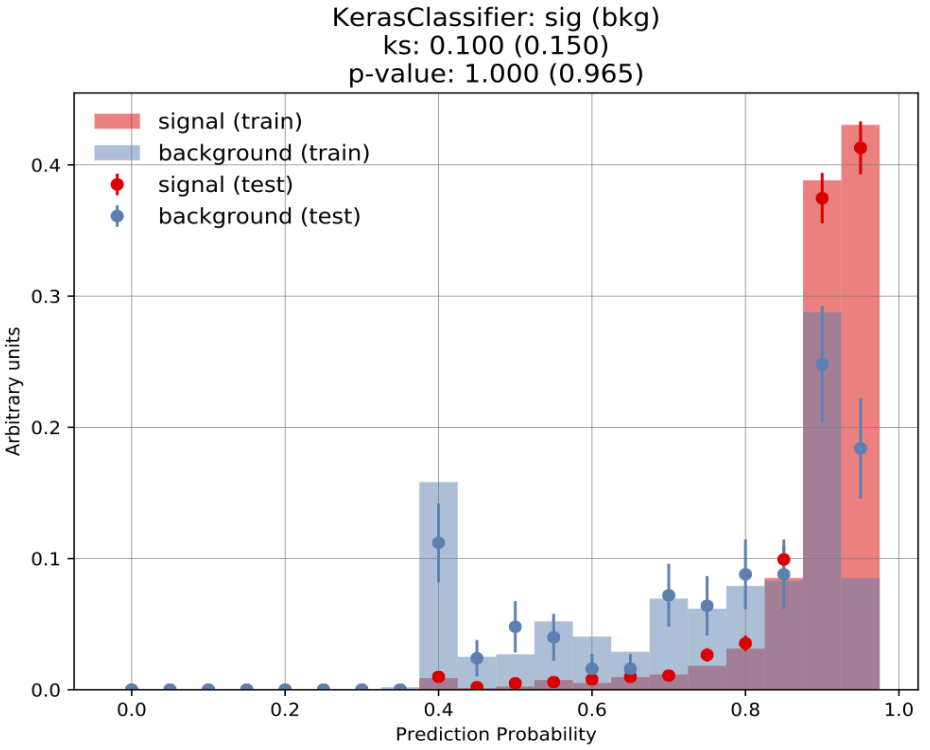
- Checking performance across several metrics for different re-sampling methods

Resampling Type	Test Log-Loss Score	Test Accuracy	Test Recall	Test Precision	Train Log-Loss	Train Accuracy
Under Sampling	1.248	0.508	0.467	0.959	1.297	0.519
Over Sampling	0.487	0.872	0.937	0.921	0.336	0.902
Synthetic Sampling	0.599	0.763	0.793	0.931	0.299	0.817

Bad sites: model performance

Similar performance compared to using Good sites info

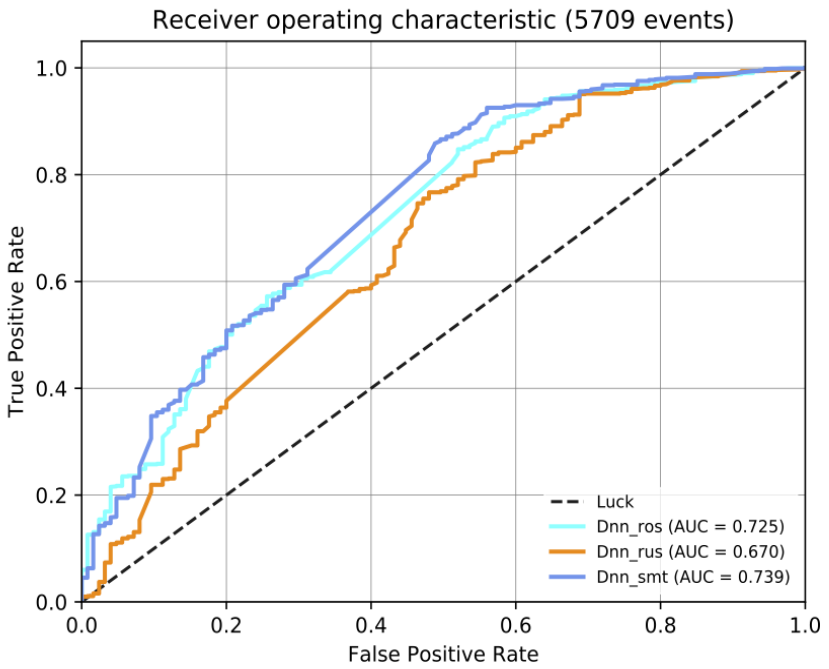




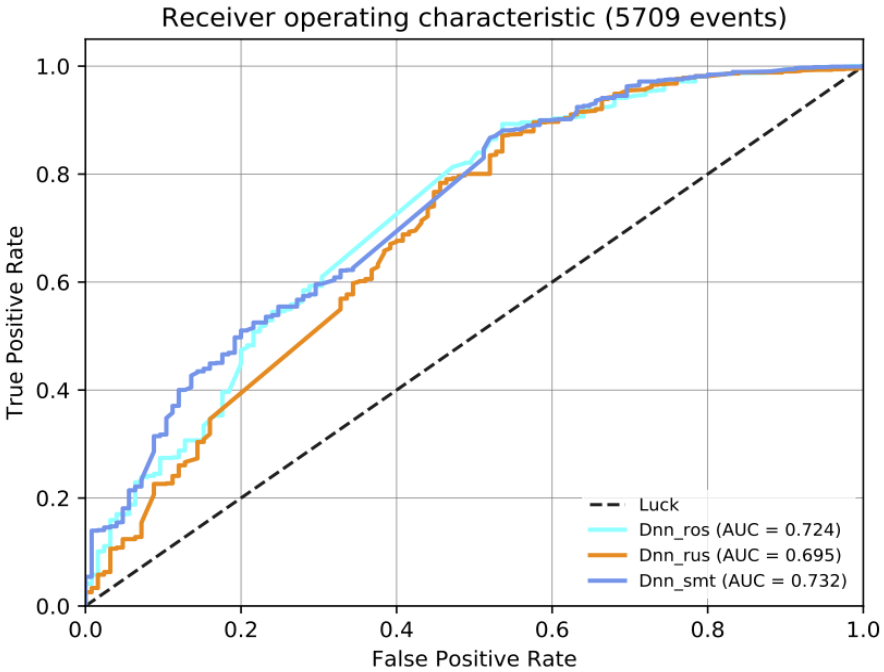
Bad sites:
Apply data resampling method

Adding feature scaling and resampling only marginally improves ROC performance

Resampling and scaling applied



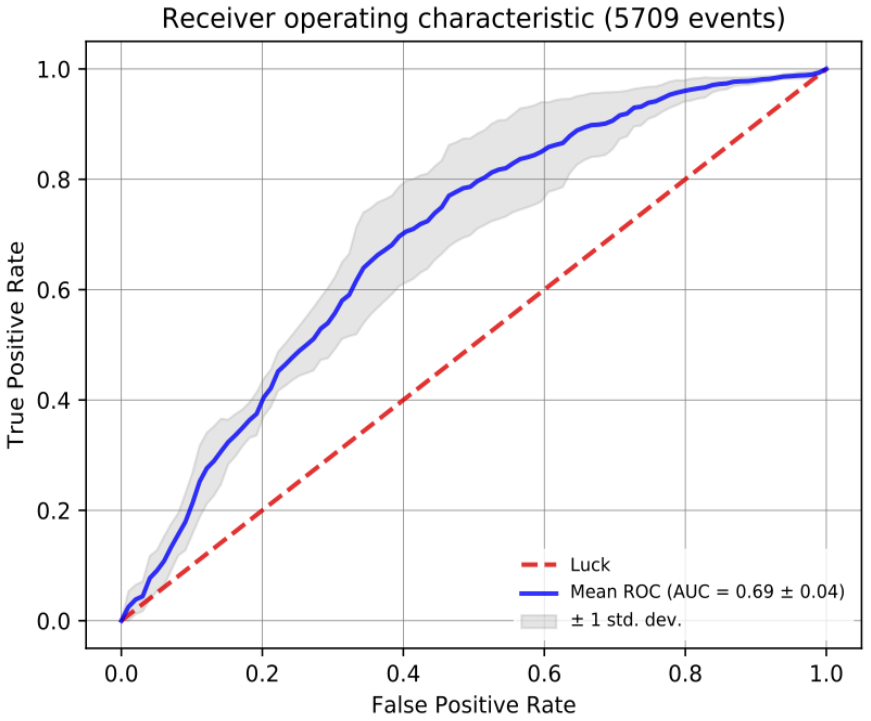
Without resampling and scaling applied



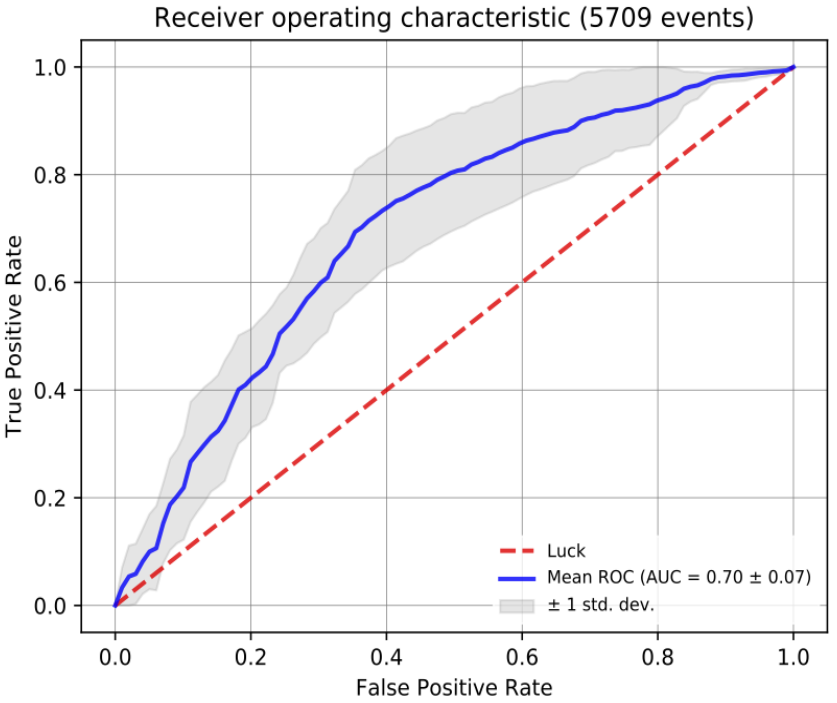
Bad sites: Cross-validation ROC

Using SMOTE and feature scaling shows slight decrease in error band

Resampling and scaling applied

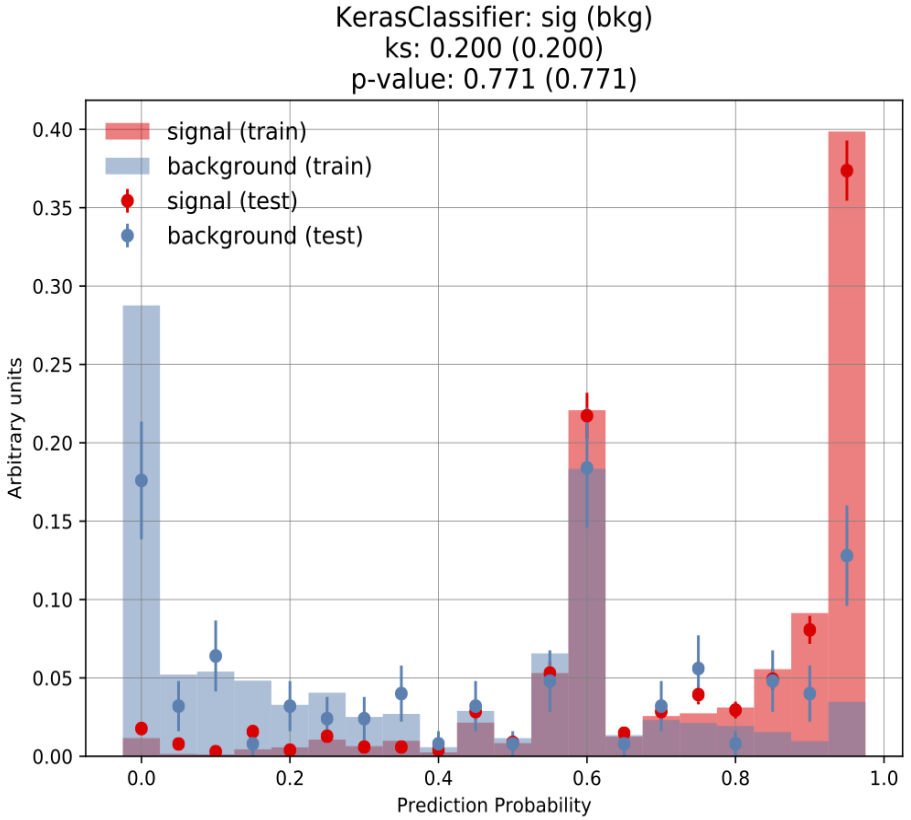


Without resampling and scaling applied



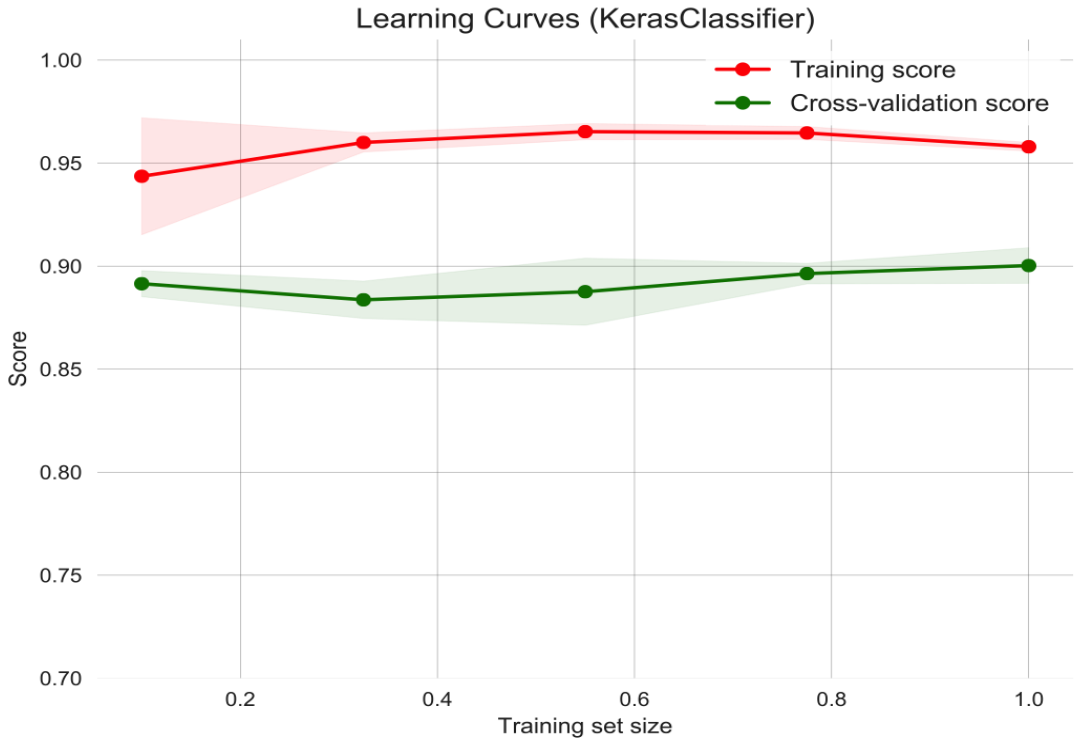
Bad sites: Over-training

Overtraining plot with SMOTE resample and scale applied



Increasing sample size

- Learning curve to check if model performance would benefit from more samples



Additional information

• currently: errors thrown (WMStats), site statuses (SiteDB and CMS Site Support team)

- next: more workflow parameters (Request Manager), job splitting, XRootD enabled or not, requested memory, ..

• basically, a matrix: # times each error codes occurs per site, times site status at the time of the error (e.g. enabled, disabled, drain). Collapsing into good or bad site statuses to limit complexity to x2 only

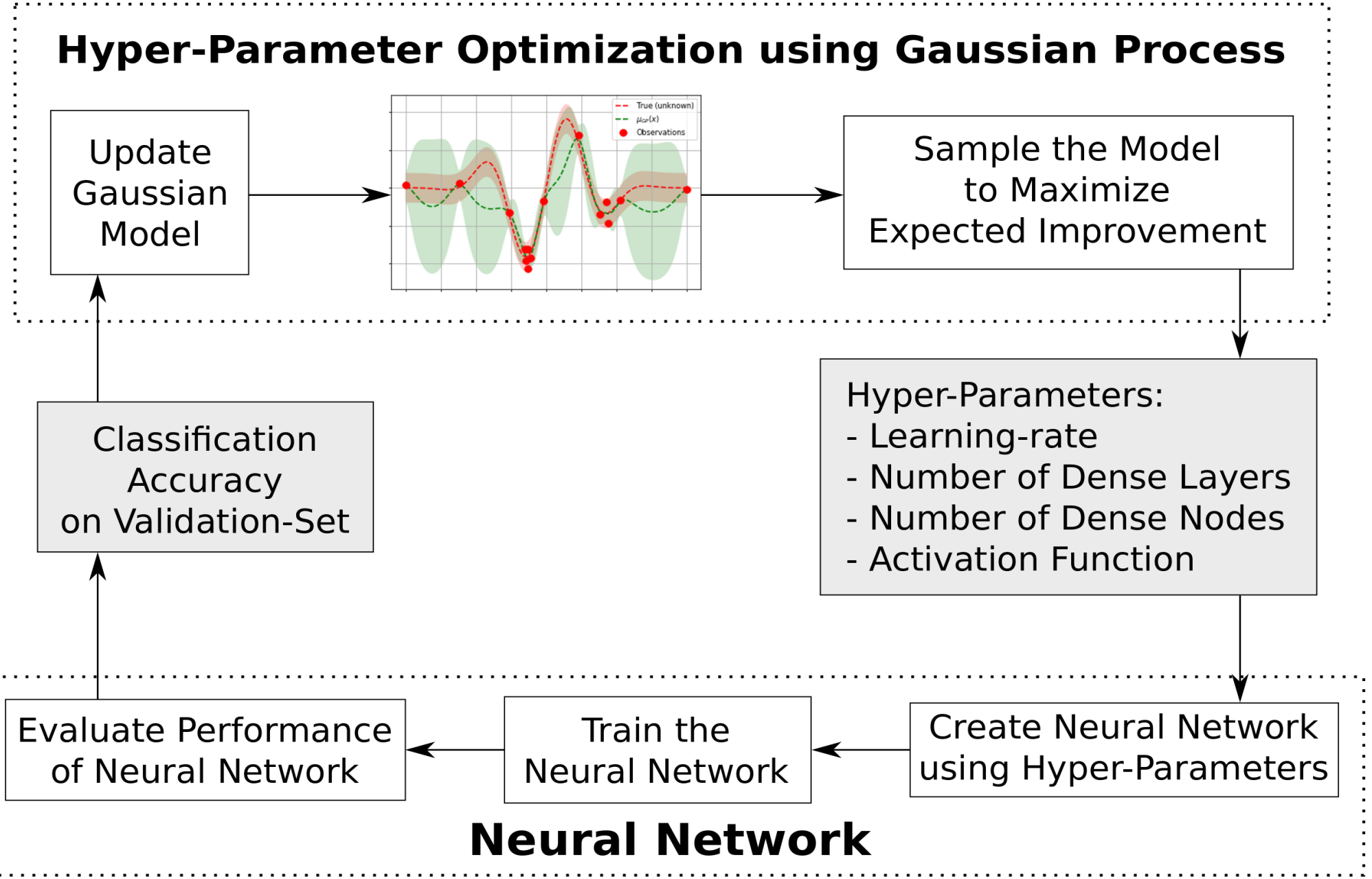
• stored workflow error patterns and site statuses over the past few months

• in progress: techniques to compress errors and sites phase space in order to group similar errors better, and to fight against sparse matrices

We ourselves to simple feed-forward multilayer perceptrons composed of a series of dense layers with relatively small numbers of nodes.

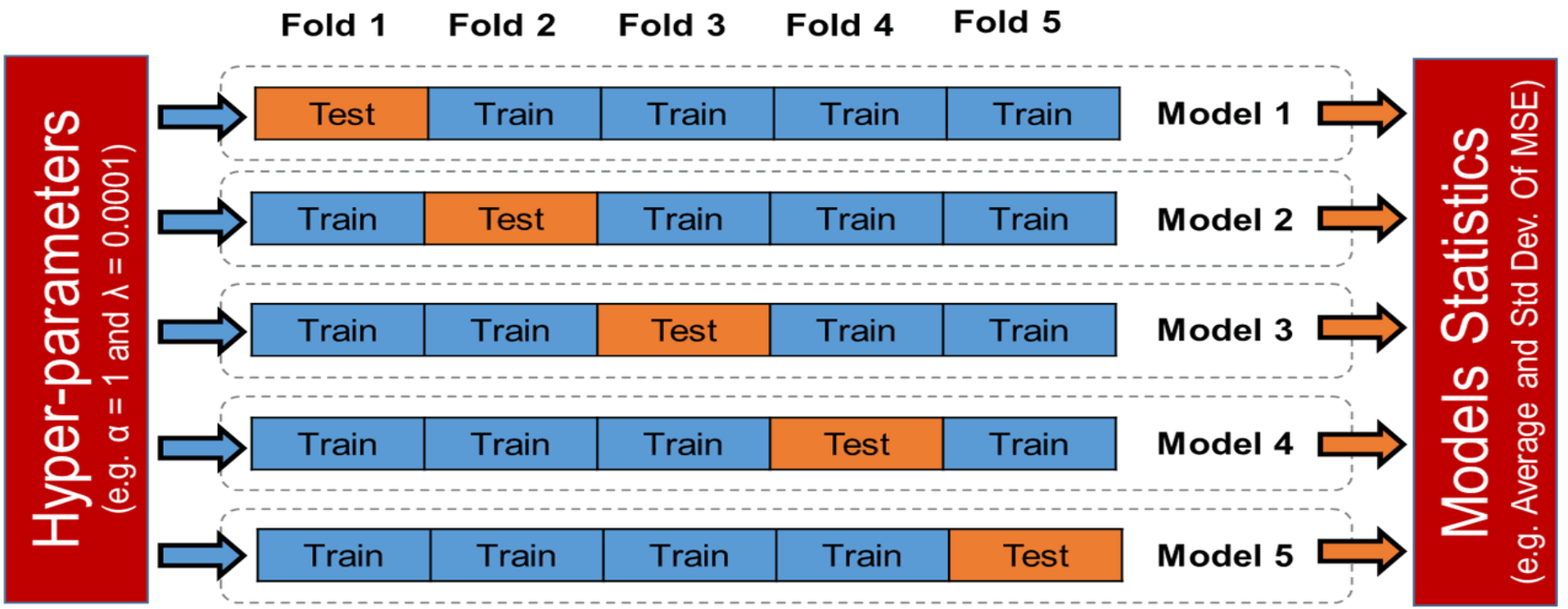
- Error detection task is an imbalanced classification problem: we have two classes we need to identify—acdc and not acdc—with one category representing the overwhelming majority of the data points.
- The positive class—acdc or not acdc (clone)—is greatly outnumbered by the negative class.
- Unrepresentative Data Sample
- This means that the sample size is too small or the examples in the sample do not effectively “cover” the cases observed in the broader domain.
- This can be obvious to spot if you see noisy model performance results. For example:
  - A large variance on cross-validation scores.
  - A large variance on similar model types on the test dataset.
  - In addition, you will see the discrepancy between train and test scores.
  - Look for a large variance in sample means and standard deviation.

Flow of the algorithm is roughly:



Cross-Validation

- Cross-validation is a way to test several combinations of hyper-parameters to identify their optimal values.
  - Method is performed on the training sample sample only (not on the held-out test samples)

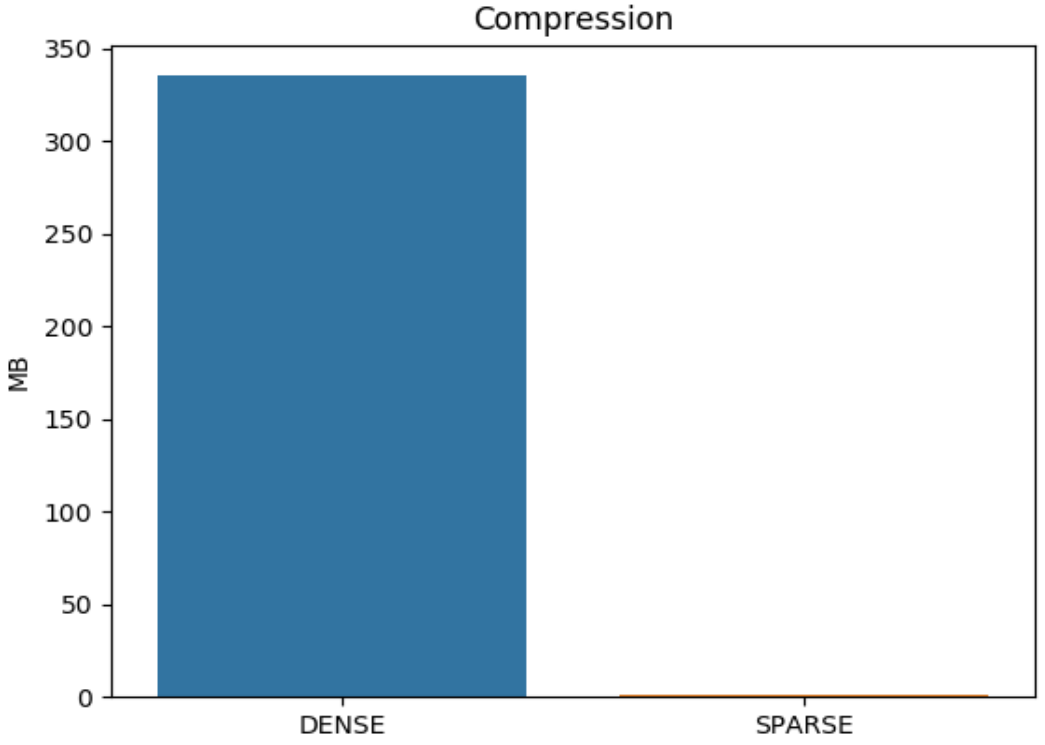


Software specification

- Libraries used with the **Python** programming language:
  - **Pandas** is an open source library providing high-performance, easy-to-use data structures and data analysis tools for handling tabular data in “series” and in “data frames”.
  - **Matplotlib** is a data visualization package which produces publication-quality figures.
  - **Keras** is a high-level neural network library running on top of **TensorFlow** (run on GPU) developed with a focus on enabling fast experimentation.
  - **Scikit-Learn** is an efficient tool for machine learning, data mining, and data analysis *(handle model building pipeline)*.

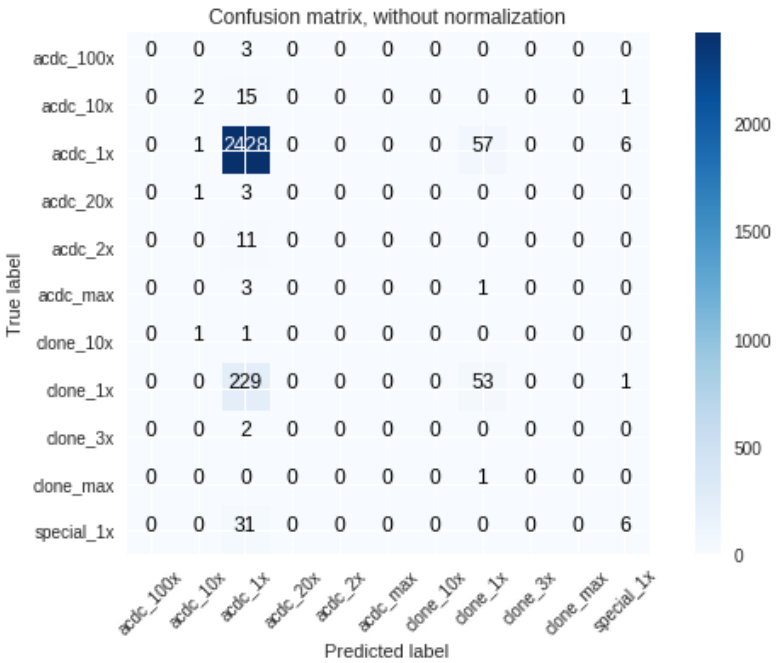
Combined sites: data compression

- Ran into memory issues while training model
  - Handled by employing sparse matrix representation of the data

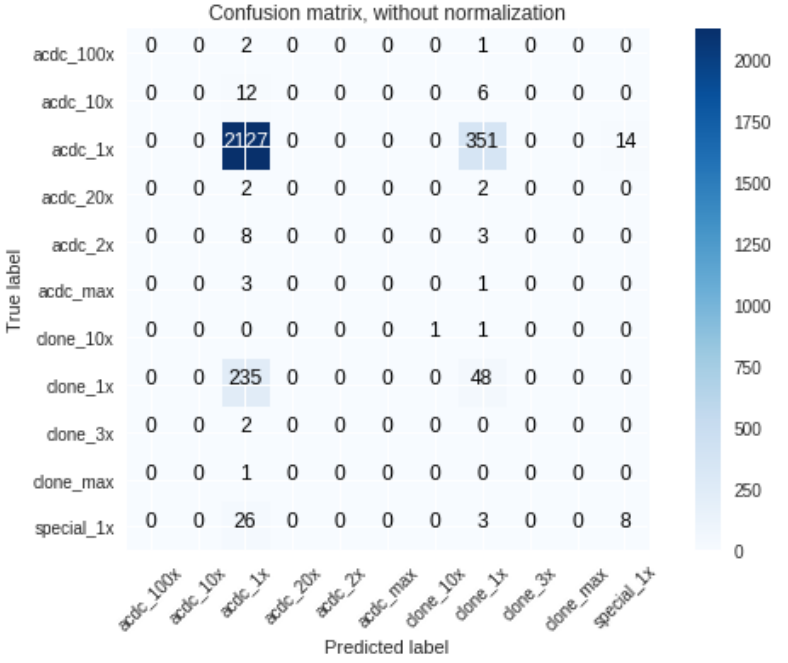


Comparison based on prediction yield

- TBA
  - Multi-class



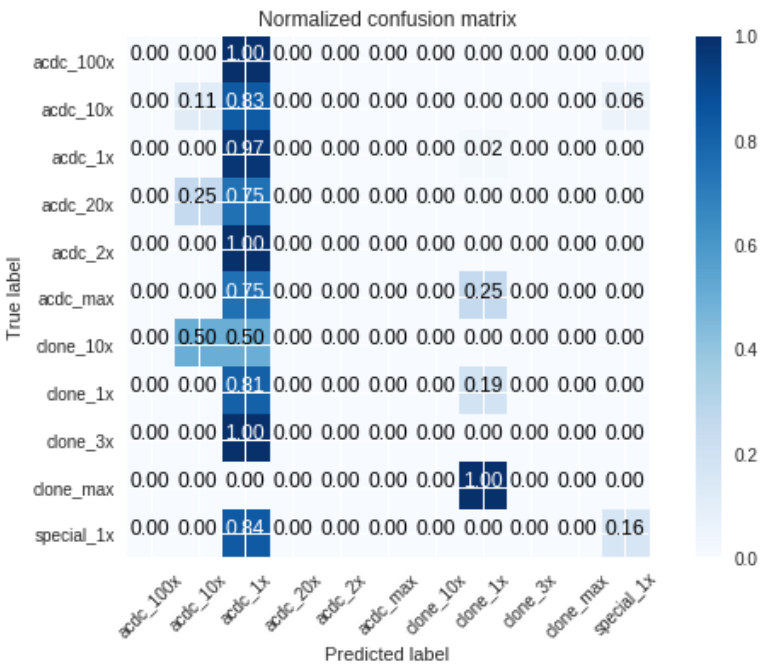
  - Multi-class based on Cascading



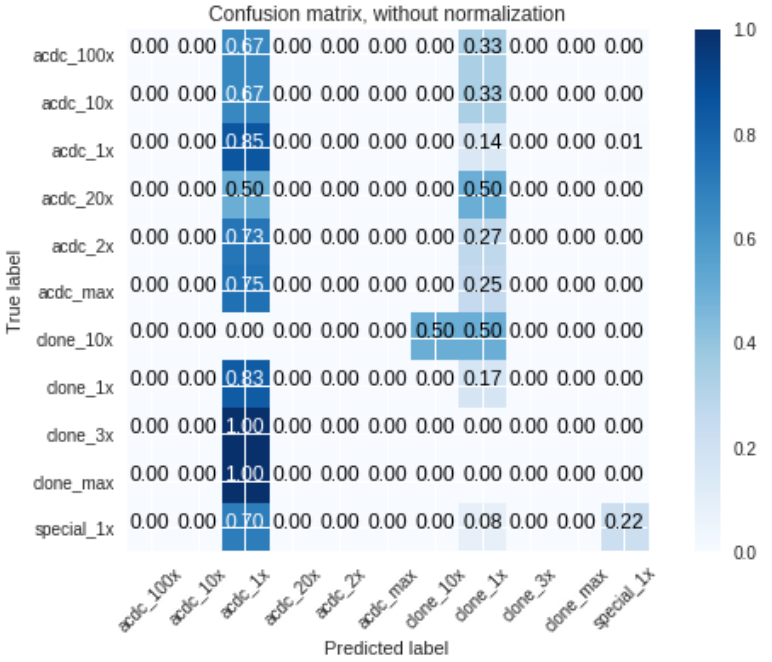
Comparison based on prediction yield percentage

- TBA

- Multi-class

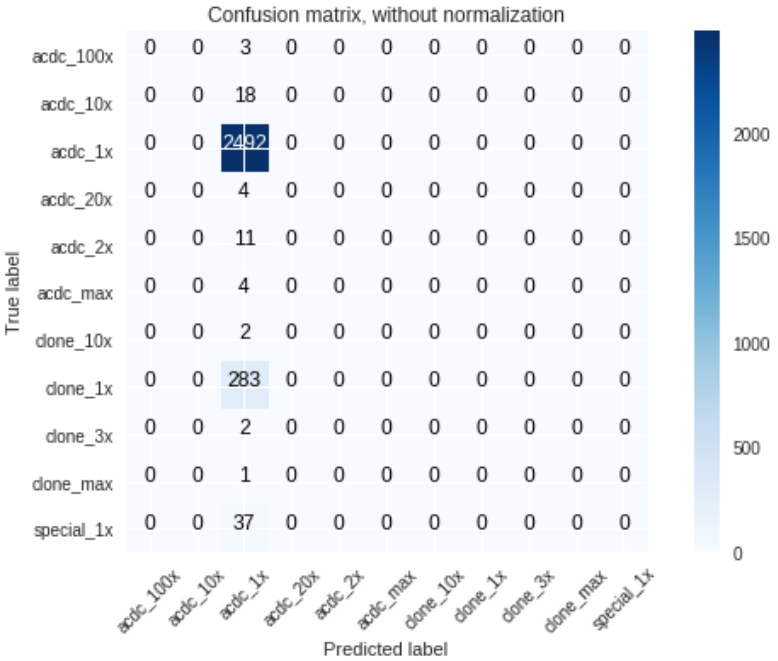


- Multi-class based on Cascading



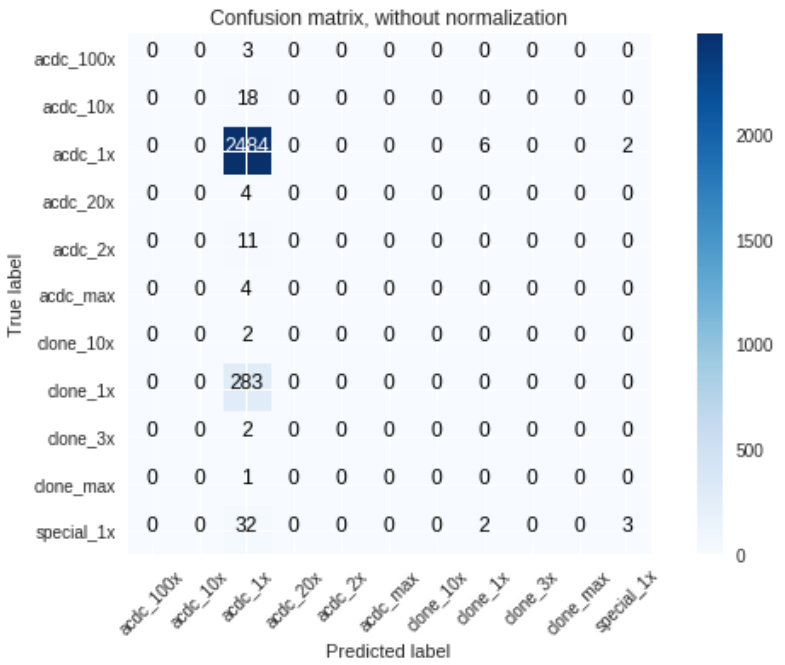
Based on log-loss

- Positive np.mean of cross_val_score
- Best score=-1.0740 (EI)
- Expected Improvement (EI) best parameters:
  - nlayers= 14
  - nneurons= 5
  - l2_norm= 0.8535590410250283
  - dropout_rate= 0.017177621112338382
- Model performance:
  - Negative log-likelihood: 1.3032932080378012
  - Accuracy: 0.8722436121806091
  - Recall: 0.8722436121806091
  - Precision: 0.7608089189898767
  - F1-score: 0.8127242780161069
  - Geometric mean: 0.3338183535857972



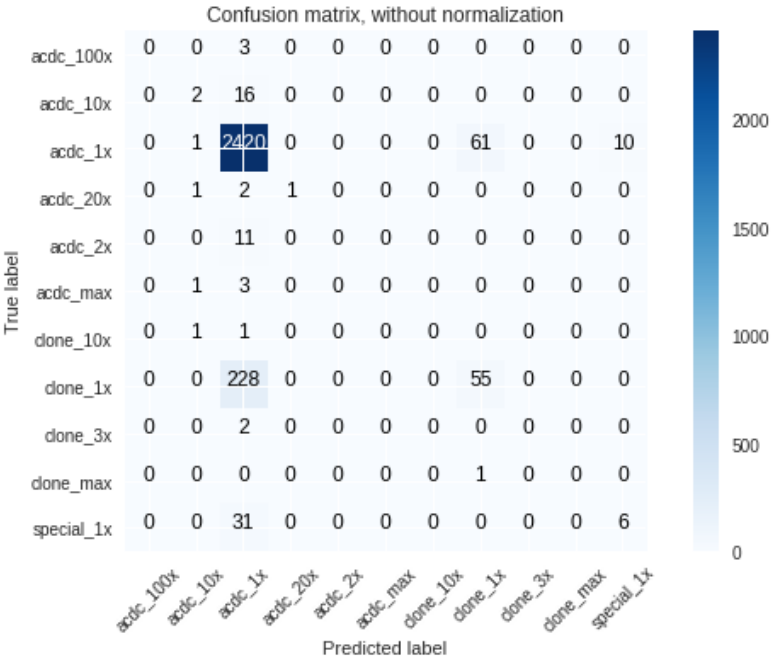
Based on log-loss

- Negative np.mean of cross_val_score
- Best score=0.5078 (EI)
- Expected Improvement (EI) best parameters:
  - nlayers= 2
  - nneurons= 48
  - l2_norm= 0.013486071964912737
  - dropout_rate= 0.0925618992066886
- Model performance:
  - Negative log-likelihood: 0.519546298111967 4
  - Accuracy: 0.8704935246762339
  - Recall: 0.8704935246762339
  - Precision: 0.7696034168797047
  - F1-score: 0.8139389728107095
  - Geometric mean: 0.3483335314463356



Based on F1-score

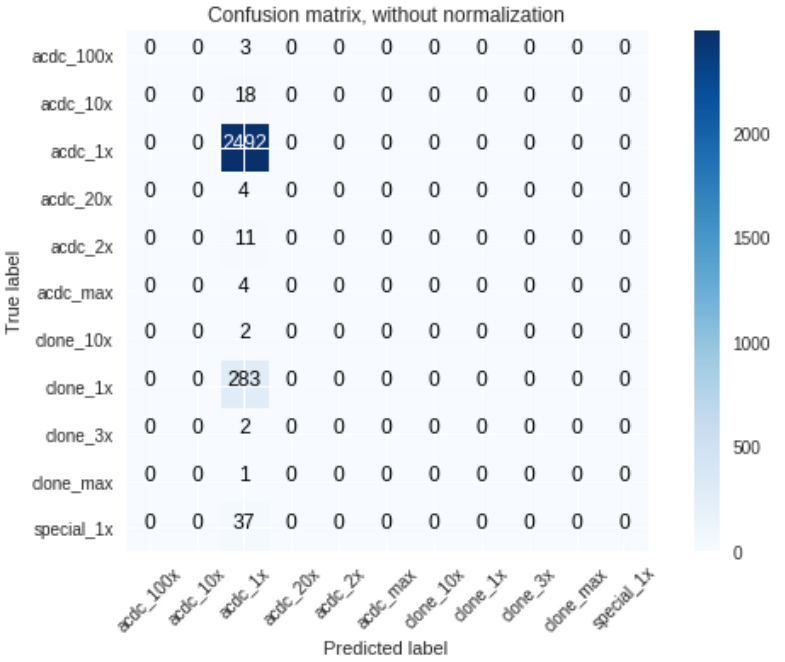
- Negative np.mean of cross_val_score
- Best score=-0.8527 (EI)
- Expected Improvement (EI) best parameters:
  - nlayers= 3
  - nneurons= 51
  - l2_norm= 0.0014677547170664112
  - dropout_rate= 0.027796975515266827
- Model performance:
  - Negative log-likelihood: 0.6643870197520889
  - Accuracy: 0.8694434721736087
  - Recall: 0.8694434721736087
  - Precision: 0.8318181063124231
  - F1-score: 0.8422372203044928
  - Geometric mean: 0.5002395350305183



Model type	Log-Loss	Accuracy	Recall	Precision	F1-score
Binary	**0.395**	**0.89**	0.89	0.86	0.80
Multi-class	**0.66**	**0.87**	**0.86**	**0.83**	**0.84**

Based on F1-score

- Positive np.mean of cross_val_score
- Best score=-0.8527 (EI)
- Expected Improvement (EI) best parameters:
  - nlayers= 12
  - nneurons= 18
  - l2_norm= 0.2010926962787662
  - dropout_rate= 0.01562069367563987
- Model performance:
  - Negative log-likelihood:0.588263257934282
  - Accuracy: 0.588263257934282
  - Recall: 0.8722436121806091
  - Precision: 0.760808918989876 7
  - F1-score: 0.812724278016106 9
  - Geometric mean: 0.3338183535857972 6



Based on Geometric-mean

- Negative np.mean of cross_val_score
- Best score=-0.5411 (EI)
- Expected Improvement (EI) best parameters:
  - Nlayers= 3
  - Nneurons= 51
  - l2_norm= 0.0014677547170664112
  - dropout_rate= 0.027796975515266827
- Model performance: