Final presentation
Github version

Adam Wuilmart
August Regnell
Eric Bojs
Ludvig Wärnberg Gerdin

Executive summary

Issues

High computational waste due to difficulty determining feasibility in a optimization problem

Method

Machine learning algorithms

Solutions

SVM	Random Forest	Regression	Deep Learning	Dimensionality Reduction

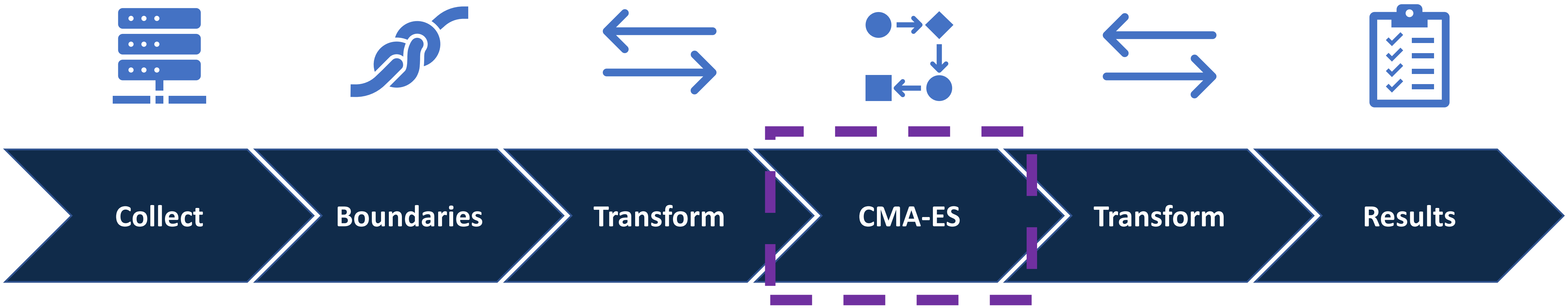
Results

No significant results with current approach due to nature of problem optimization problem

Background

Working with a market technology leader

Client’s Value chain: Algorithms to be used in the CMA-ES step



Scope of Work: Additional modules delivered on top of original SOW

	Original scope	Delivery
Tasks	Test ML-algorithms on classification problem	Tested ML-algorithms on classification problems
Tasks	Explore alternative approaches	Explored dimension reduction
Format	Python code with detailed explanations	Regression approach
Requirement	• Accurate classification algorithm • Time efficiency to improve on current approach • Consistency over different problems	Custom testing library

Problem

Complex non-linear optimization problem

Background: Client solves complex non-linear optimization problems

Problem formulation

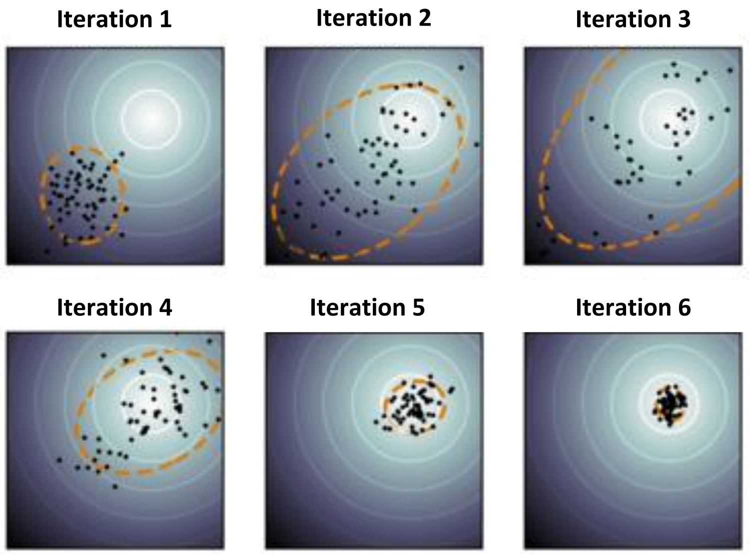
Client finds new trades for a set of banks which will reduce the sum of **CENSORED** required in the network.

$$\min f(x)$$
$$\text{s.t. } x \in D$$

f is proxy for **total CENSORED** in network
$x \in R^{4,000}$ is proxy of **new trades** in the network
D is the **feasible** space, by SIMM & CCP constraints
Both f and D are **non-trivial**

Extremely computer intensive problem to solve

CMA-ES



Each iteration of the evolutionary algorithm **samples 1,024 points** in a region. Next steps **moves towards the optimum** while still being in the feasible region

Data: Data set in use is heavy and highly mobile yielding problems for ML

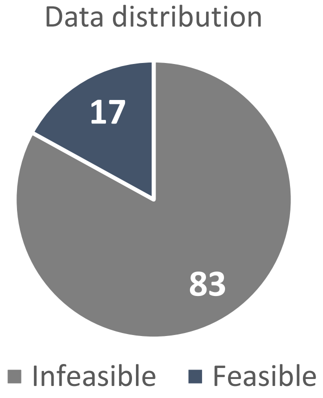
Data Characteristics

4000 Dimensions

1024 Samples

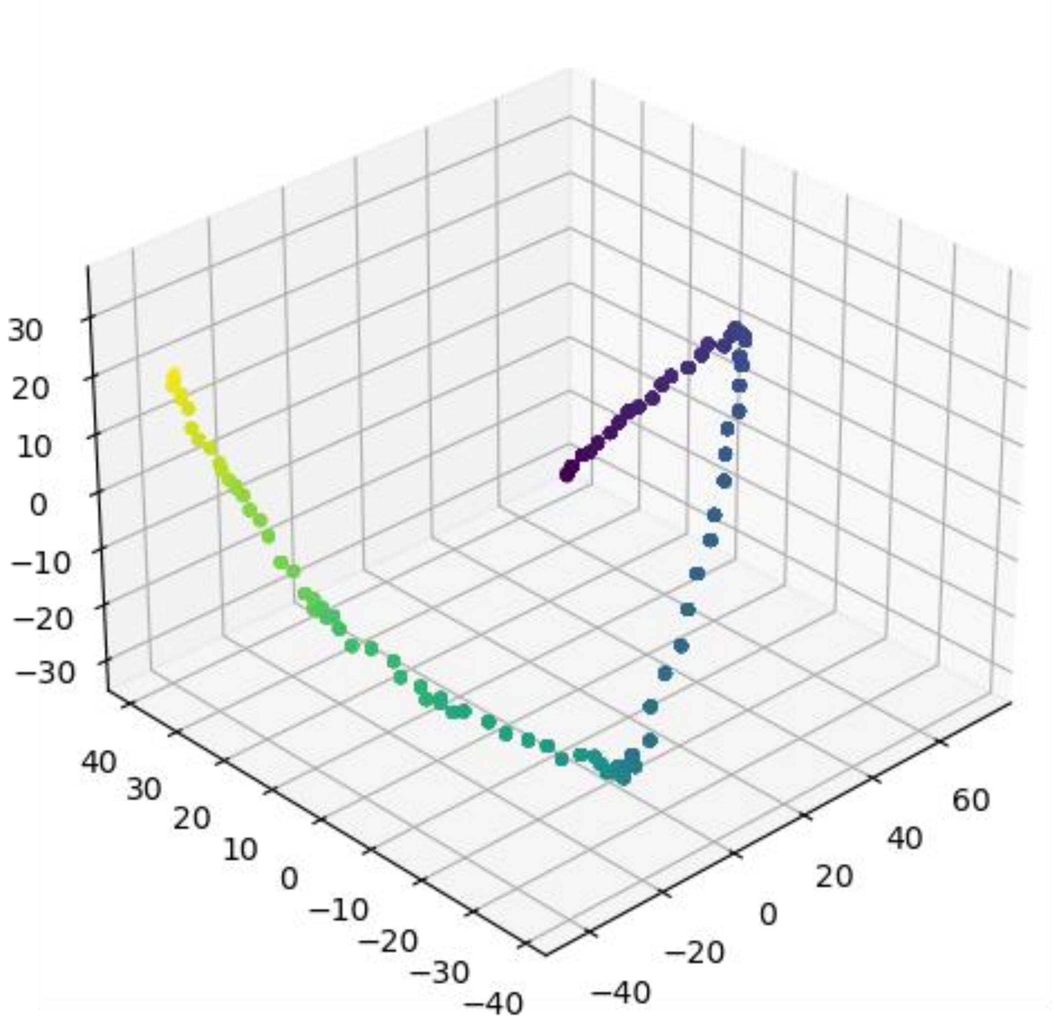
10 000 Iterations

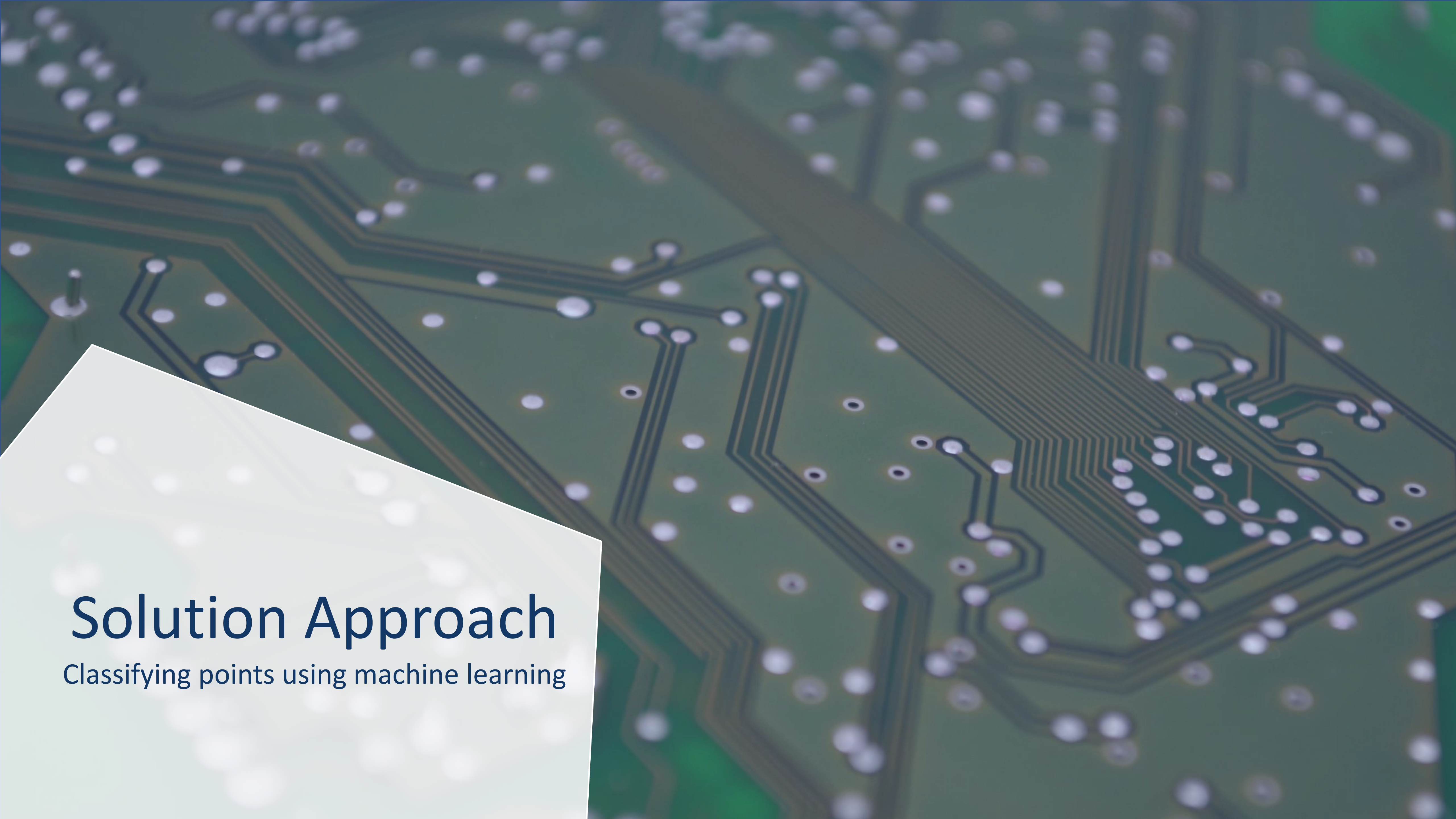
$4.1*10^{11}$ Datapoints



Extremely large and imbalanced dataset

CMA-ES Visualization

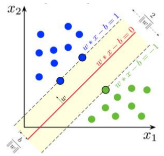
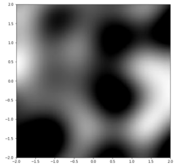
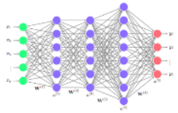
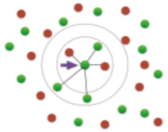
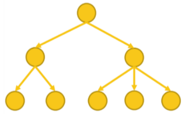




Solution Approach

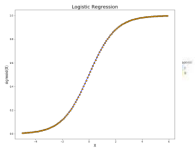
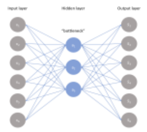
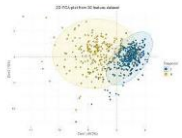
Classifying points using machine learning

Algorithms: Machine learning algorithms (I/II)

Algorithm		Description
	Support Vector Machines	Support vector machines is a supervised ML-algorithm which finds the best fitting hyperplane which linearly divides the data
	Radial Basis Function Sampler	RBFSampler is a way to approximate the RBF kernels in SVM methods, but with a benefit of being very quick. One combines the approximated kernel with a classifier. The extra speed yields better optimization of parameters
	Deep Learning Networks	Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised
	Nearest Neighbor	Nearest Neighbour search, as a form of proximity search, is the optimization problem of finding the point in a given set that is closest, or most similar, to a given point
	Random Forest	Random forest classifiers are an ensemble of decision trees, each giving their own classification. The decision made by the ensemble is determined by voting, where the option with most votes from the trees wins

Source: towardsdatascience.com, Wikipedia

Algorithms: Machine learning algorithms (II/II)

Algorithm		Description
	**Logistic Regression**	Logistic regression fits a cumulative probability density function using the logit function to the data. This can then be used to classify future samples
	**Gradient Boosting Machine**	Gradient Boosting Machines build an ensemble of shallow trees, in contrast to Random Forest models that build an ensemble of deep trees. Trees are added *sequentially,* each new tree learn with respect to current error
	**Autoencoder**	An autoencoder is a type of artificial neural network used to learn efficient coding of unlabelled data. The encoding is validated and refined by attempting to regenerate the input from the encoding
	**Principal Component Analysis**	Principal Component Analysis is the process of computing the principal components and using them to perform a change of basis on the data, using only the first few principal components and ignoring the rest

Source: towardsdatascience.com, Wikipedia

Method: Input from multiple sources yielding standardized testing

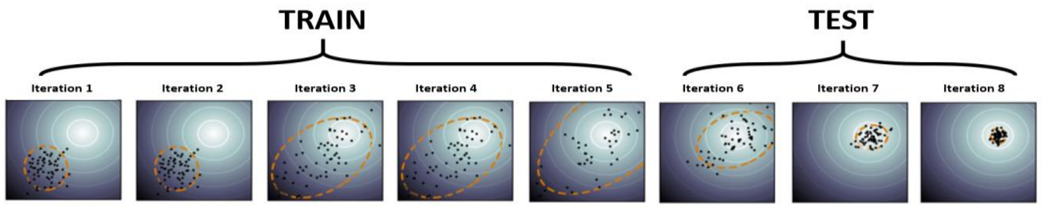
Client suggestions

+

ML-expert suggestions

1. At least twice as many training samples as features
2. 75% train, 25% test
3. Dimensionality reduction infeasible
4. More data, 100k samples is not enough
5. The problem is very difficult

Standardized model tester



We developed a **python library** with:

1. Fixed number of training samples (ca. 8000)
2. Fixed number of test samples (ca. 2000)
3. Fixed lag between training and test samples
4. Hyperparameter tuning
5. Several epochs to get confidence intervals

Output:

variable	average	std	95.0% CI	min	max
weighted accuracy [%]	49.65	1.2	(48.79, 50.51)	47.23	51.73
duration [s]	2.12	0.03	(2.1, 2.14)	2.09	2.18
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	35.62	14.46	(25.27, 45.96)	14.84	54.79
feasible_recall [%]	63.8	14.92	(53.12, 74.47)	42.11	85.45
feasible_precision [%]	17.26	2.56	(15.43, 19.09)	13.74	22.65
infeasible_recall [%]	35.5	14.41	(25.19, 45.81)	14.91	54.86
infeasible_precision [%]	82.33	2.47	(80.56, 84.09)	76.92	86
AUC	0.5	0.01	(0.49, 0.51)	0.47	0.52

Results

ML-algorithms unfit for problem

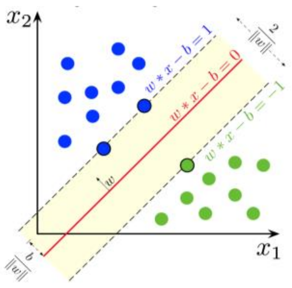
Results: RBFSampler + Classifier showing the most promising results

Algorithm	Balanced Accuracy	95% Prediction Interval
Support Vector Machine	50.2%	(49.31%, 51.03%)
RBFSampler + Classifier	**51.3%**	(50.30%, 52.20%)
Nearest Neighbor	50.3%	(49.51%, 51.01%)
Logistic Regression	50.4%	(49.89%, 50.97%)
Random Forest Classifier	50.4%	(49.75%, 50.58%)
Deeplearning (kNN)	50.2%	(50.04%, 50.30%)
Gradient Boosting Machine (GBM)	50.6%	(49.69%, 51.55%)
Autoencoder + GBM	50.1%	(49.82%, 50.33%)

Support Vector Machine: Slow algorithm with poor results

Model

Support vector machines are supervised ML-algorithms which finds the best fitting hyperplane which linearly divides the data.



Hyper parameters

Kernel function	Transforms hyperspace since the original is not linearly seperable
Regularization parameter	Amount of missclassifications allowed on train data
Kernel coefficient	Specific to kernel chosen

Results

Weighted accuracy: 50.17%

Precision (inf./feasible): 82.42% / 17.63%

Recall (inf./feasible): 46.55% / 53.79%

Speed: 51.69 s

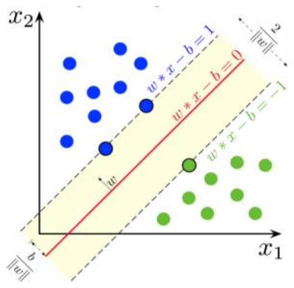
2/5

variable	average	std	95.0% CI	min	max
weighted accuracy [%]	50.17	1.2	(49.31, 51.03)	47.82	51.91
duration [s]	51.69	0.13	(51.59, 51.78)	51.56	51.93
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	46.5	23.08	(29.99, 63.02)	3.56	79.93
feasible_recall [%]	53.79	22.77	(37.51, 70.08)	23.2	96.14
feasible_precision [%]	17.63	2.35	(15.95, 19.31)	14.66	22.68
infeasible_recall [%]	46.55	23.17	(29.97, 63.12)	3.48	80.51
infeasible_precision [%]	82.42	3.27	(80.08, 84.76)	75.34	85.9
AUC of ROC	0.5	0.02	(0.48, 0.52)	0.46	0.54

Support Vector Machine: Slow algorithm with poor results

Model

Support vector machines are supervised ML-algorithms which finds the best fitting hyperplane which linearly divides the data.



Hyper parameters

Kernel function	Transforms hyperspace since the original is not linearly seperable
Regularization parameter	Amount of missclassifications allowed on train data
Kernel coefficient	Specific to kernel chosen

Results

Weighted accuracy: 50.17%

Precision (inf./feasible): 82.42% / 17.63%

Recall (inf./feasible): 46.55% / 53.79%

Speed: 51.69 s

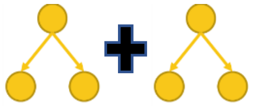
2/5

variable	average	std	95.0% CI	min	max
weighted accuracy [%]	50.17	1.2	(49.31, 51.03)	47.82	51.91
duration [s]	51.69	0.13	(51.59, 51.78)	51.56	51.93
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	46.5	23.08	(29.99, 63.02)	3.56	79.93
feasible_recall [%]	53.79	22.77	(37.51, 70.08)	23.2	96.14
feasible_precision [%]	17.63	2.35	(15.95, 19.31)	14.66	22.68
infeasible_recall [%]	46.55	23.17	(29.97, 63.12)	3.48	80.51
infeasible_precision [%]	82.42	3.27	(80.08, 84.76)	75.34	85.9
AUC of ROC	0.5	0.02	(0.48, 0.52)	0.46	0.54

Gradient Boosting Machine: Poor results due to overfitting

Model

Ensemble of shallow trees. Trees are added *sequentially,* each new tree learn with respect to current error.



Hyper parameters

Maximum tree depth	Adjusts the maximum allowed depth of each tree in ensemble
Number of leaves	Maximum number of leaves in one tree
Regularization parameter	Amount of regularization to apply. That is, to shrink the coefficients.

Results

Weighted accuracy: 50.62%

Precision (inf./feasible): 81.69% / 18.16%

Recall (inf./feasible): 96.88% / 4.36%

Speed: 13.87 s

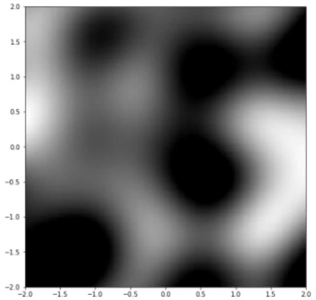
2/5

variable	average	std	95.0% PI	min	max
weighted accuracy [%]	50.62	0.75	(49.69, 51.55)	49.93	51.87
duration [s]	13.87	0.61	(13.12, 14.62)	13.25	14.99
infeasible_percentage [%]	81.47	2.67	(78.17, 84.78)	77.25	85.06
infeasible_guessed_percentage [%]	96.62	3.05	(92.83, 100.41)	91.5	99.95
feasible_recall [%]	4.36	4.19	(-0.84, 9.57)	0	11.5
feasible_precision [%]	18.16	11.49	(3.89, 32.42)	0	32.5
infeasible_recall [%]	96.88	2.78	(93.42, 100.34)	92.23	99.94
infeasible_precision [%]	81.69	2.49	(78.6, 84.79)	77.64	85.04

RBFSampler: Fast algorithm with most promising results

Model

The RBFSampler is a way to approximate the RBF kernels in an efficient way. One combines the approximated kernel with a classifier.



Hyper parameters

Length scale (smoothness)	Adjusts the influence of single data points
Number of components	How many Monte Carlo samples that are used per feature
Parameters of the classifier	Loss-function, penalty-type and penalty-strength: depends on the chosen model

Results

Weighted accuracy: 51.25%

Precision (inf./feasible): 83.23% / 18.25%

Recall (inf./feasible): 63.31% / 39.19%

Speed: 3.74 s

3/5

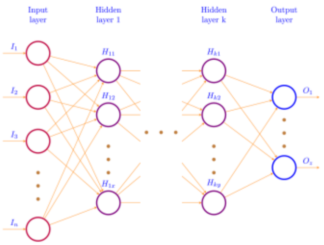
variable	average	std	95.0% CI	min	max
weighted accuracy [%]	51.25	1.32	(50.3, 52.2)	49.01	53.18
duration [s]	3.74	0.81	(3.16, 4.32)	3.21	6.04
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	62.89	12.49	(53.95, 71.82)	45.61	84.33
feasible_recall [%]	39.19	13.93	(29.23, 49.16)	17.24	59.44
feasible_precision [%]	18.25	2.17	(16.7, 19.8)	14.72	21.2
infeasible_recall [%]	63.31	12.19	(54.59, 72.03)	46.55	84.62
infeasible_precision [%]	83.23	2.57	(81.39, 85.06)	76.8	86.07

Note: See appendix for remaining algorithms

Deeplearning kNNs: Key takeaway from the models results

k-layer Neural Networks

Using multiple layers of connected neurons features are extracted from the input in a similar way to human learning.



Hyper parameters

k	The number of hidden layers in the network
Learning rate	How fast the network learns from every iteration
Optimizer	How and in which direction the learning algorithm moves

Results

Weighted accuracy: 50.17%

Precision (inf./feasible): 82.62% / 21.09%

Recall (inf./feasible): 92.51% / 7.82%

Speed: 1.24 s

1/5

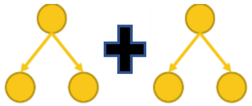
variable	average	std	95.0% CI	min	max
weighted accuracy [%]	50.17	0.18	(50.04, 50.3)	50	50.58
duration [s]	1.24	0.18	(1.1, 1.37)	1.05	1.64
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	92.45	11.58	(84.17, 100.74)	68.07	100
feasible_recall [%]	7.82	11.86	(-0.66, 16.31)	0	32.92
feasible_precision [%]	21.09	28.63	(0.61, 41.57)	0	100
infeasible_recall [%]	92.51	11.52	(84.27, 100.75)	68.25	100
infeasible_precision [%]	82.62	2.38	(80.92, 84.33)	77.38	85.45
auc of roc	0.49	0.01	(0.48, 0.5)	0.46	0.51

Note:
Source:

Gradient Boosting Machine: Poor results due to overfitting

Model

Ensemble of shallow trees. Trees are added *sequentially,* each new tree learn with respect to current error.



Hyper parameters

Maximum tree depth	Adjusts the maximum allowed depth of each tree in ensemble
Number of leaves	Maximum number of leaves in one tree
Regularization parameter	Amount of regularization to apply. That is, to shrink the coefficients.

Results

Weighted accuracy: 50.62%

Precision (inf./feasible): 81.69% / 18.16%

Recall (inf./feasible): 96.88% / 4.36%

Speed: 13.87 s

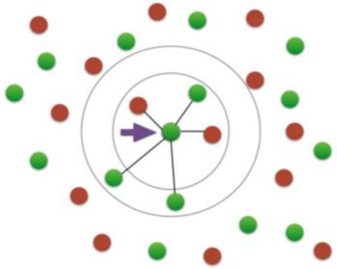
2/5

variable	average	std	95.0% PI	min	max
weighted accuracy [%]	50.62	0.75	(49.69, 51.55)	49.93	51.87
duration [s]	13.87	0.61	(13.12, 14.62)	13.25	14.99
infeasible_percentage [%]	81.47	2.67	(78.17, 84.78)	77.25	85.06
infeasible_guessed_percentage [%]	96.62	3.05	(92.83, 100.41)	91.5	99.95
feasible_recall [%]	4.36	4.19	(-0.84, 9.57)	0	11.5
feasible_precision [%]	18.16	11.49	(3.89, 32.42)	0	32.5
infeasible_recall [%]	96.88	2.78	(93.42, 100.34)	92.23	99.94
infeasible_precision [%]	81.69	2.49	(78.6, 84.79)	77.64	85.04

Nearest Neighbor: Key takeaway from the models results

Nearest Neighbor

Classifying new points be performing a plurality vote of its *k* nearest neighbors.



Hyper parameters

k	The number of neighbors considered.
Distance metric	How the distances should be calculated, e.g., l2-norm

Results

Weighted accuracy: 50.26%

Precision (inf./feasible): 82.48% / 17.65%

Recall (inf./feasible): 50.58% / 49.94%

Speed: 1.11 s

1/5

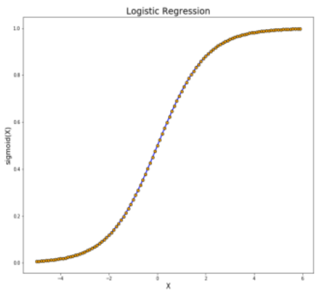
variable	average	std	95.0% CI	min	max
weighted accuracy [%]	50.26	1.04	(49.51, 51.01)	48.35	52.12
duration [s]	1.11	0.2	(0.96, 1.25)	0.79	1.43
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	50.51	36.55	(24.36, 76.66)	7.37	96.29
feasible_recall [%]	49.94	35.84	(24.3, 75.57)	2.65	92.75
feasible_precision [%]	17.65	2.85	(15.62, 19.69)	11.84	22.6
infeasible_recall [%]	50.58	36.7	(24.33, 76.84)	7.4	96.08
infeasible_precision [%]	82.48	2.7	(80.55, 84.41)	76.81	85.85
auc of roc	0.5	0.01	(0.5, 0.51)	0.48	0.52

Note:
Source:

Logistic Regression: Key takeaway from the models results

Logistic Regression

Logistic regression fits a cumulative probability density function using the logit function to the data. This can then be used to classify future samples.



Hyper parameters

Penalty	The loss function assigned to missclassifications
Regularization parameter	Amount of missclassifications allowed on train data
Fit interception	Weather or not we should assume that $\beta_0=0$ or not

Results

Weighted accuracy: 50.43%

Precision (inf./feasible): 82.79% / 17.65%

Recall (inf./feasible): 36.19% / 64.68%

Speed: 496.05 s

1/5

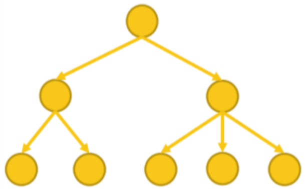
variable	average	std	95.0% CI	min	max
weighted accuracy [%]	50.43	0.75	(49.89, 50.97)	49.02	51.65
duration [s]	496.05	182.34	(365.61, 626.49)	286.23	707.68
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	36.05	9.6	(29.18, 42.92)	22.61	49.71
feasible_recall [%]	64.68	8.87	(58.33, 71.02)	51.31	77.66
feasible_precision [%]	17.65	2.3	(16.01, 19.3)	15.05	22.48
infeasible_recall [%]	36.19	9.76	(29.2, 43.17)	22.5	50.06
infeasible_precision [%]	82.79	2.69	(80.87, 84.71)	76.52	86.3
AUC of ROC	0.49	0.01	(0.48, 0.5)	0.47	0.51

Note:
Source:

Random Forest:

Random forest

Random forest classifiers ensembles of decision trees, each giving a classification. Decision made by the ensemble is determined by voting, where the option with most votes from the trees wins



Hyper parameters

Estimators	Number of trees used in the forest
Criterion	How each split is determined
Max depth	Maximum number of levels allowed in the tree

Results

Weighted accuracy: 50.01%

Precision (inf./feasible): 82.58% / 3.32%

Recall (inf./feasible): 98.2% / 1.9%

Speed: 7 minutes

1/5

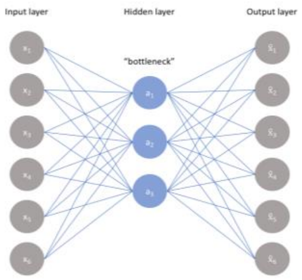
The model is slow and gives poor accuracy. There will be a high need for hyper-parameter tuning in the case that the model works to prevent it form overfitting the data. This will significantly vary from case to case. Thereby, we do not recommend the use of this model for the problem

Note:
Source:

Autoencoder + GBM: Key takeaway from the models results

Autoencoder

Neural network used to denoise data. The performance measured by squared cell-wise difference between denoised and original data.



Hyper parameters

Number of layers	Adjust the complexity of the model, I.e. how well-fitted to sample
Learning rate	Distance to jump in each iteration
Activation function	Transformation of output from each layer

Results

Weighted accuracy: 50.08%

Precision (inf./feasible): 82.67% / 12.08%

Recall (inf./feasible): 92.68% / 7.48%

Speed: 198.17 s

1/5

variable	average	std	95.0% PI	min	max
weighted accuracy [%]	50.08	0.36	(49.82, 50.33)	49.69	51.08
duration [s]	198.17	20.73	(183.34, 213.0)	170.08	223.45
infeasible_percentage [%]	82.56	2.4	(80.85, 84.28)	77.25	85.45
infeasible_guessed_percentage [%]	92.64	20.25	(78.15, 107.12)	31.93	100
feasible_recall [%]	7.48	20.77	(-7.39, 22.34)	0	69.74
feasible_precision [%]	12.08	11.07	(4.16, 20.0)	0	28.57
infeasible_recall [%]	92.68	20.1	(78.3, 107.05)	32.43	100
infeasible_precision [%]	82.67	2.14	(81.14, 84.2)	78.44	85.43

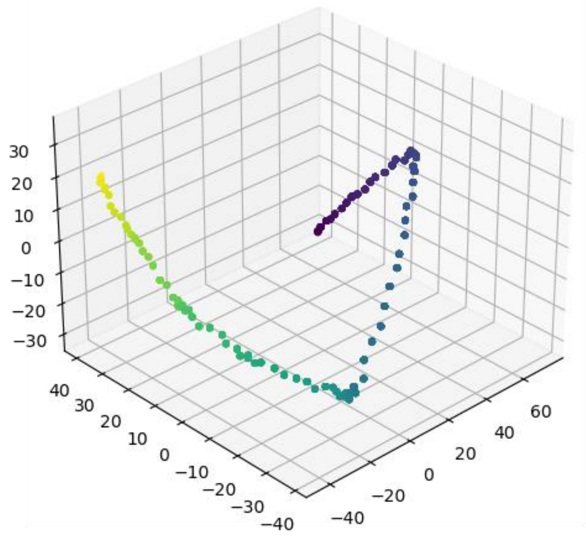
Note:
Source:

Underlying Causes

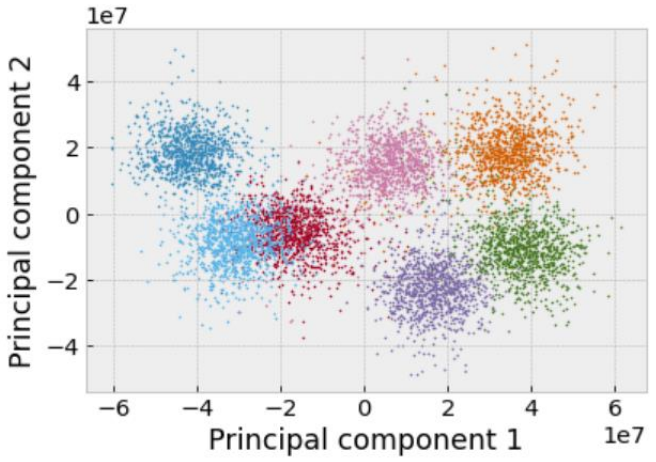
Data, dimensionality & mobility

Outcome analysis: Mobility of CMA-ES causing difficulties for ML

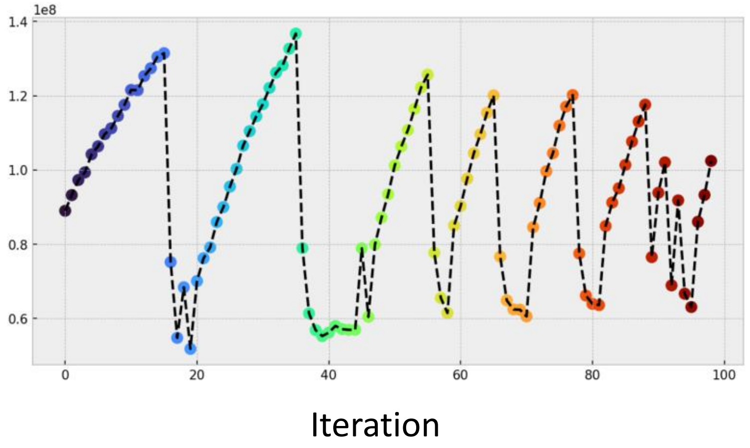
3D PCA on all iterations



2D PCA on 7 iterations



Distance between iterations



Jumps between iterations poses a very difficult challenge for ML-algorithms due to extrapolation

Outcome analysis: Infeasibility of feature reduction due to boundaries

Stability of influential features

Data

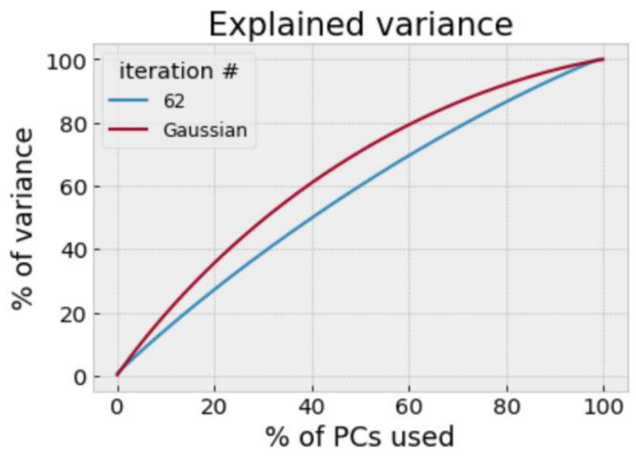
50% | **50%**

$$y = \beta_1 X + \epsilon \qquad y = \beta_2 X + \epsilon$$

If parameter β_i is influential, we would expect it to show up in both sets. However, this was **not found**

Principal Component Analysis

We perform PCA on the in-/feasible solutions seperatly and see if there is redundant information

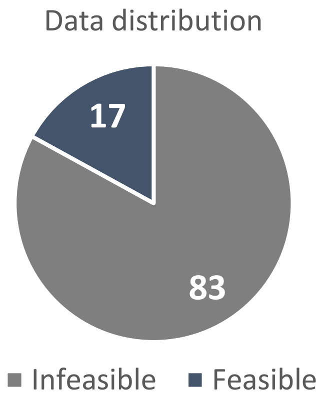


Feature reduction could **not be done**

ML-algorithms have trouble training on few samples in the vicinity of test set which yields insufficient information

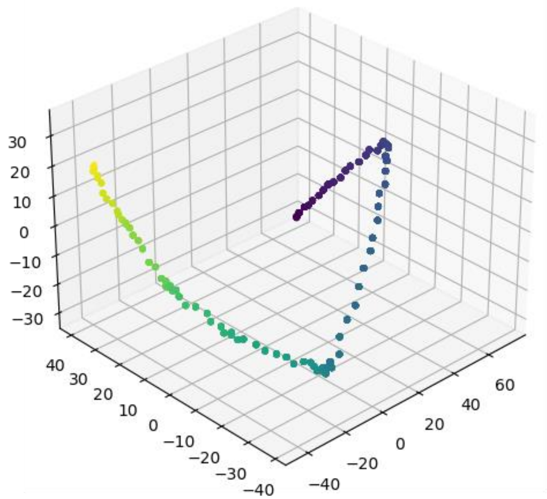
Outcome analysis: Insufficient data for algorithms to work with

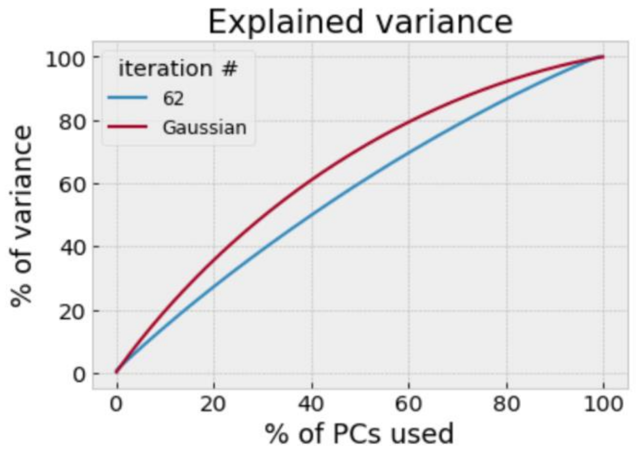
Underlying problems	Potential solution	Issue
4000 vs **8000** Dimensions vs Samples	More data necessary	**Higher CMA-ES runtime**



Outcome analysis: Results due to mobility, information and data size

CMA-ES mobility	Information density	Data size





4000 vs **8000**

Dimensions Samples

Insignificant results from ML algorithms

Next steps: Two potential keys to unlock the problem

Transforms and adaptation possible with algorithm understanding

Train on larger datasets or change algorithm approach