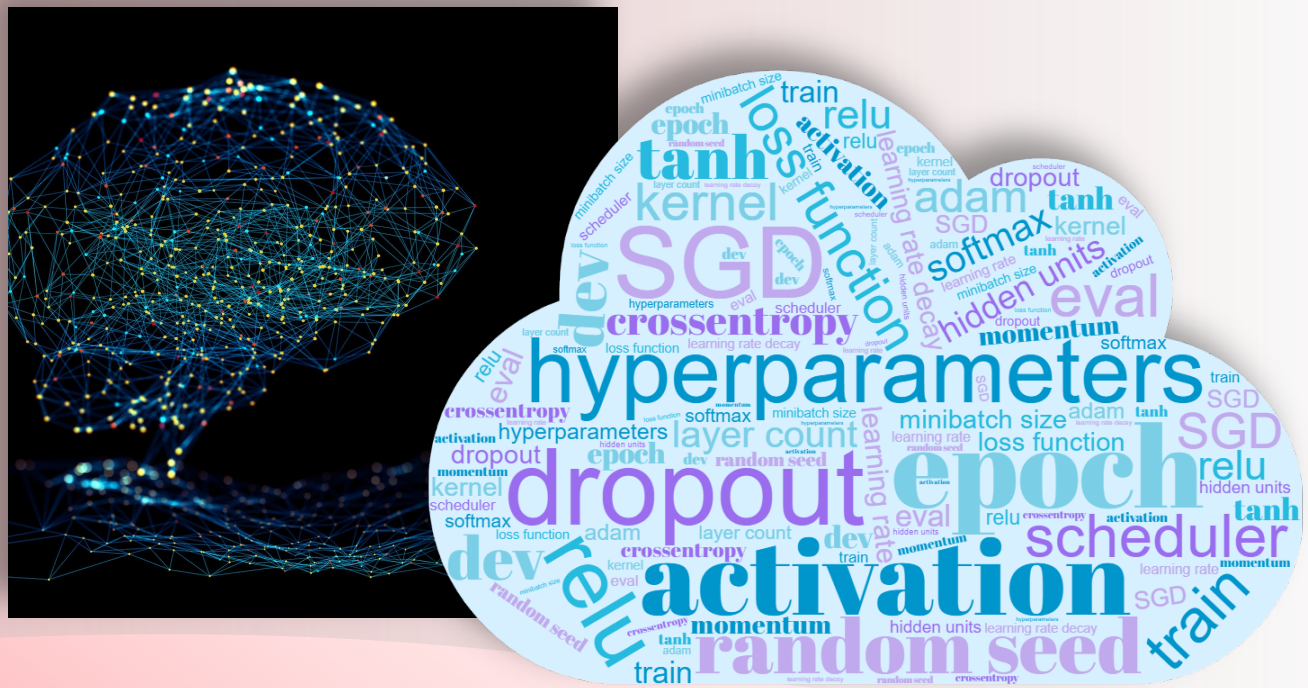
On Automating Hyperparameter Optimization for Deep Learning Applications



N. Shawki[1], R. Rodriguez Nunez[2], I. Obeid[1] and J. Picone[1]

1. Neural Engineering Data Consortium, Temple University
2. Dynamical Systems Lab, Temple University

NEURAL ENGINEERING DATA CONSORTIUM

Abstract

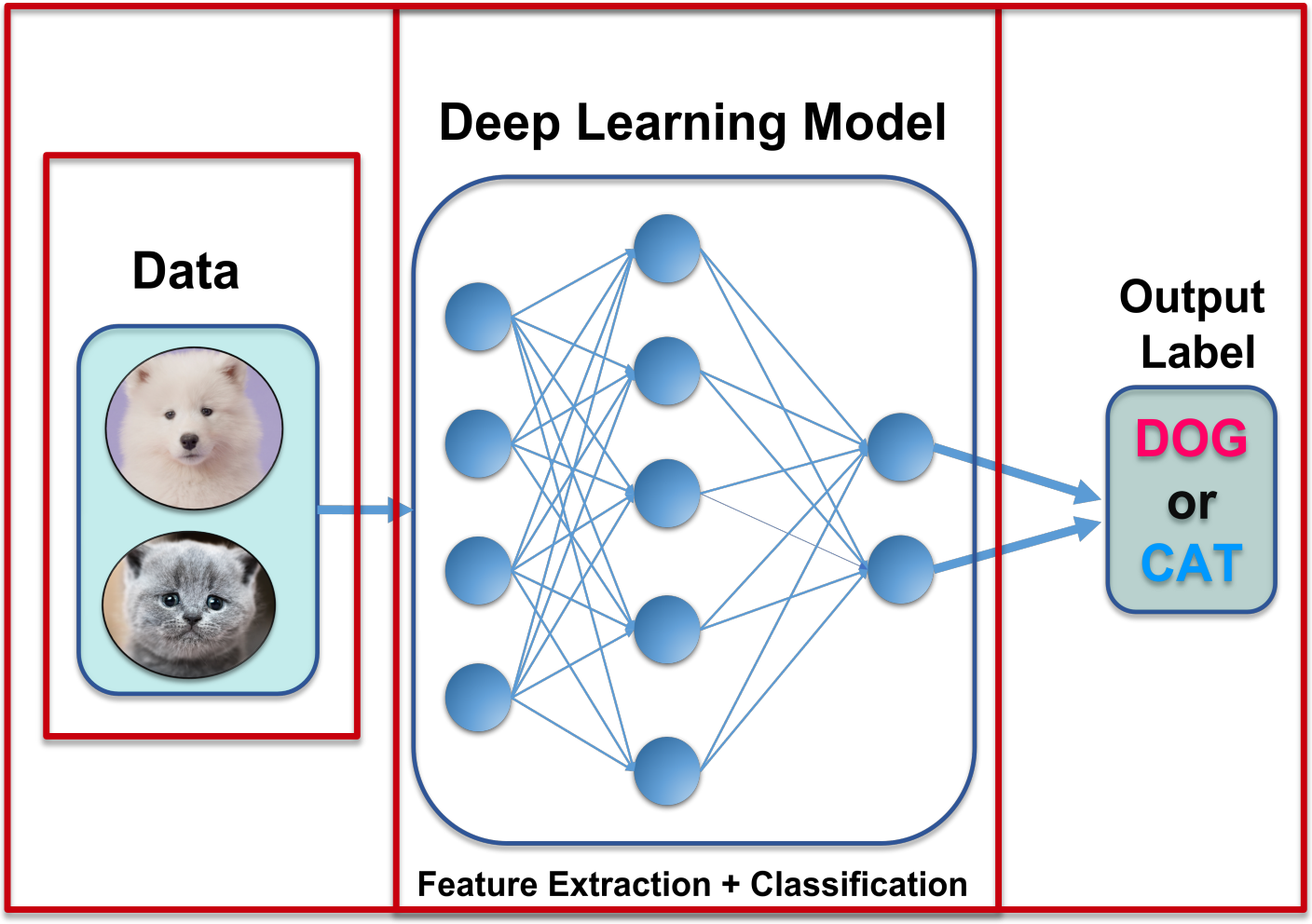
- The parameter search space of a deep learning model is large and populated with a less-than-ideal solutions.
- Ideal solutions can be obtained by manually tuning a deep learning model, but that requires expertise and experience, which is in short supply.
- Automatic hyperparameter tuning algorithms, known as autotuners, automate the training process and increase the accessibility of deep learning technology to different scientific communities and novice users.
- The appeal of autotuning techniques lies not only in the fact that it reduces the trial-and-error period but also provides an off-the-shelf aid for experts in other fields with limited knowledge in machine learning.
- We investigated an autotuner called Keras Tuner and compared its performance to manual tuning on several complex data sets. These data sets were designed to expose flaws in the learning algorithms.
- Experiments show that autotuning performed well on synthetic datasets but was inadequate on real data.
- Autotuning tools are excellent for creating baseline models on new datasets, but they need more attention to formulate optimal solutions for end-users with less background in deep learning.

Introduction

- Deep Learning techniques have been extremely successful for applications that have large amounts of training data (e.g., human language technology, image classification, and social network analysis).
- These techniques can automatically extract features from data. They eliminate the need for handcrafted features.
- With the availability of high-speed computing infrastructure, publicly available datasets, open-source libraries, and pre-trained models, researchers are applying deep learning techniques to a wide range of problems.
- The performance of a deep learning model depends largely on whether an optimal set of hyperparameters can be found during training. This is traditionally achieved by manually searching the parameter space, which requires expertise, experience, and time.
- Automatic hyperparameter tuners, known as autotuners, provide an attractive alternative to manual tuning. We investigate a specific autotuner, Keras Tuner.
- We prepared synthetic datasets and challenging real world data to test the generalization capability of both auto and manual tuning.

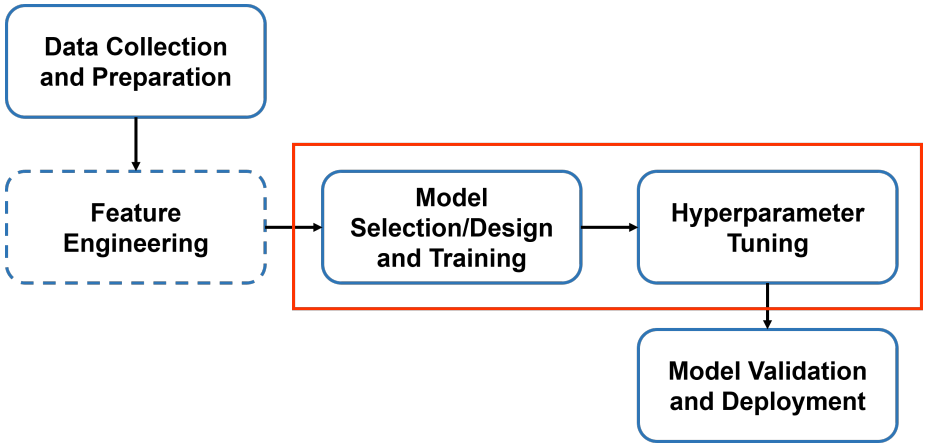
Deep Learning Challenges

- With the abundance of available data, models, and automation techniques, <u>we investigated if achieving a generalized solution has become easier without human intervention.</u>



Challenges in Tuning Hyperparameters

- Solving a problem with deep learning often follows a pipeline that includes feature engineering, model selection, training by tuning hyperparameters, and validation.



- Hyperparameters (HPs) can be divided into two categories:
  - Training-related: learning rate, batch size, dropout rate, and epoch count
  - Model design-related: model structure, regularization, and activation functions
- Due to the number of hyperparameters involved, it is nearly impossible to explore all possible combinations.
- Autotuning is an active research area that involves automated search techniques to find an optimal solution.
- A few popular autotuning algorithms are Grid Search, Random Search, Bayesian Optimization, and Gradient-based Optimization.
- Keras Tuner uses random search for finding a generalized solution.

Hyperparameter Tuning

- The process of selecting hyperparameters is a complex optimization problem.
- Grid search of the hyperparameter space is a popular method which is simple to implement and parallelize, and provides insight into the search space.
- Ongoing research suggests that automated random search optimization is a more efficient alternative that often yields as good or better models than manual methods due to their ability to search larger configuration spaces.
- The problem of hyperparameter tuning ($\lambda^{(*)}$) can be expressed as:

$$\lambda^{(*)} = \boldsymbol{argmin}_{\lambda \epsilon \Lambda}\, \mathbb{E}_{x \sim \mathcal{G}_x}\left[\mathcal{L}\left(x, \mathcal{A}_\lambda\left(X^{(train)}\right)\right)\right] = \boldsymbol{argmin}_{\lambda \epsilon \Lambda} \Psi(\lambda) \equiv \tilde{\lambda},$$

where,

λ = the hyperparameters,

Λ = the search space,

$\mathcal{L}$ = the loss function,

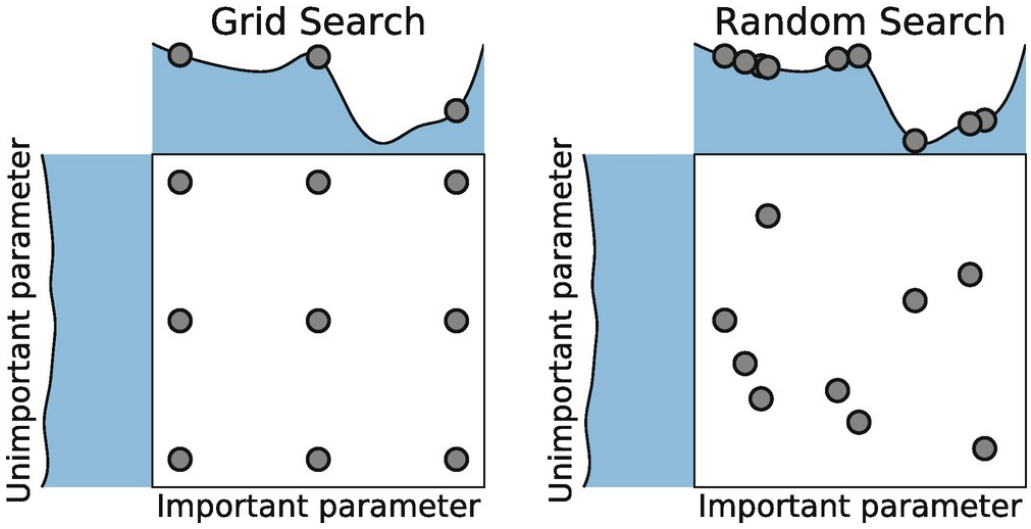
$\mathcal{G}$ = the ground truth

$\mathcal{A}$ = the learning algorithm,

Ψ = the hyperparameter response function.

Grid and Random Search

- Grid search is a popular technique for hyperparameter tuning which performs an exhaustive search with every possible combination of the HPs. It becomes computationally expensive as the number of HPs increases.
- Random search draws independent sets from the HP search space using statistical distributions of the HPs. When the number of HPs is high, the random search can effectively search a larger space compared to grid search.
- In the example shown, with grid search, the model explore only three distinct hyperparameter sets when the random search explore more distinct combinations.
- Keras Tuner uses an evolutionary random search algorithm to draw hyperparameter sets from the search space Λ where the new sets are influenced by the performance of the previous sets.



Manual and Autotuning

- In manual tuning, the optimal solution is obtained by selecting a set of hyperparameters, evaluating the response surface ($\psi(\lambda)$) and selecting the best performing hyperparameter set.
- While this helps to gain intuition into the decision surface and identify promising regions in the search space, this involves trial-and-error and is computationally expensive. It requires that researchers have domain expertise for the given data and experience working with ML/DL techniques.
- Autotuning methods such as the Keras Tuner tool providea user-friendly platform for the automated search of optimal hyperparameter combinations.
- An autotuner defines a hyperparameter search space that is broad enough include all reasonable combinations of λ and a hypermodel of the selected deep learning model that supports variation of hyperparameters, including ones that can change the architecture of the chosen model.
- The response surface ($\psi(\lambda)$) is evaluated on each of the randomly drawn HP sets and the best performing set of HPs is selected as the optimal solution.
- However, with autotuners such as Keras Tuner, a model can get trapped in a local optimum or draw a bad combination that terminates the process.

Experimental Data

- To investigate the generalization capability of the autotuning process, we generated three synthetic datasets (#08-#10).
- Two more datasets (#11 and #12) with real-world data were created:
  - Dataset #11 contains two types of cancer image patches.
  - Dataset #12 consists of background and seizure signal from Electroencephalograph or EEG records.
- These datasets are publicly available at:

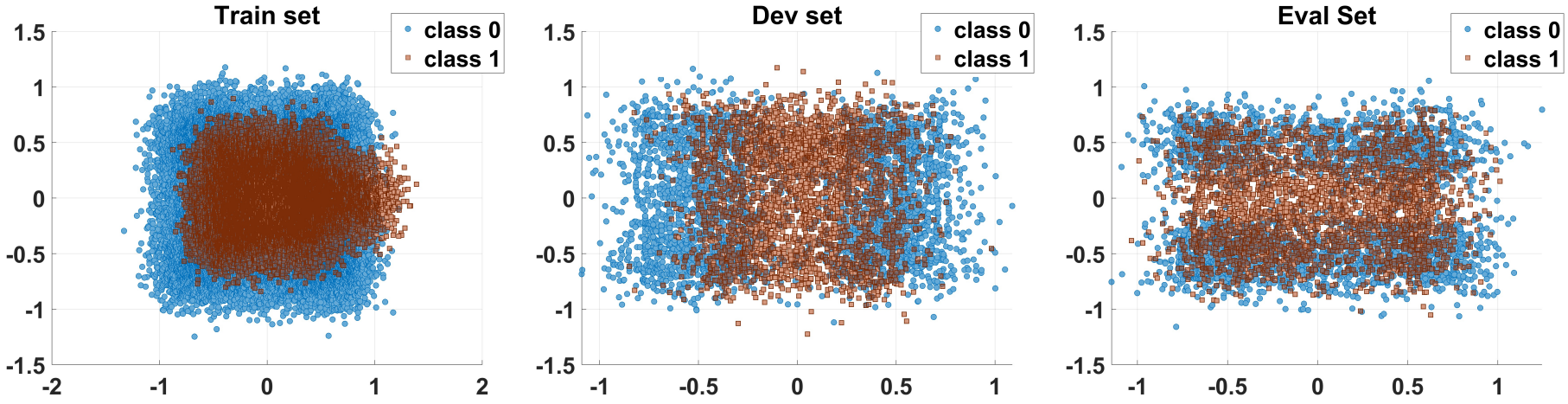
  www.isip.piconepress.com/courses/temple/ece_8527/resources/data/

Distribution of the Datasets

#	Description	No. Classes	Train	Dev	Eval
08	Synthetic data	3	300,000	15,000	15,000
09	Synthetic data	2	500,000	250,000	250,000
10	Synthetic data	2	100,000	10,000	10,000
11	Cancer Images	2	10,000	2,500	2,500
12	Electroencephalograph (EEG) signals	2	10,000	2,500	2,500

Synthetic Datasets

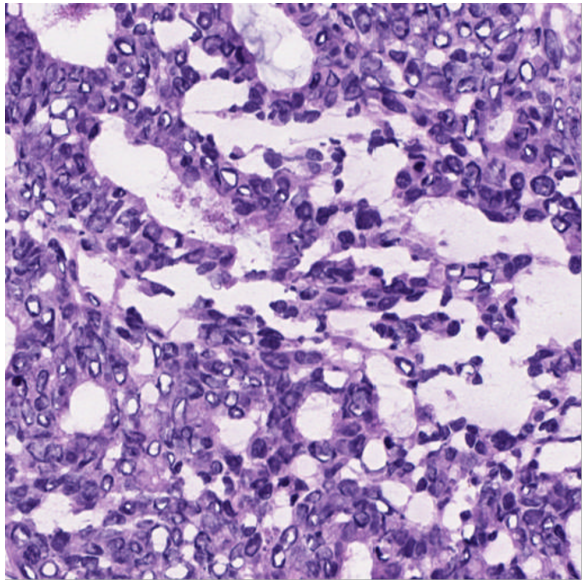
- The synthetic datasets were created using a Python tool developed in-house called IMLD (see our IEEE SPMB 2021 poster).
- Each sample in these datasets (#08-#10) contains two features so that they are easy to visualize using plotting tools available in Python and MATLAB.
- Dataset #10 was generated manually in IMLD and later augmented using Gaussian white noise.
- The subsets contain complex distribution that require the models to formulate an optimal non-linear decision surface.



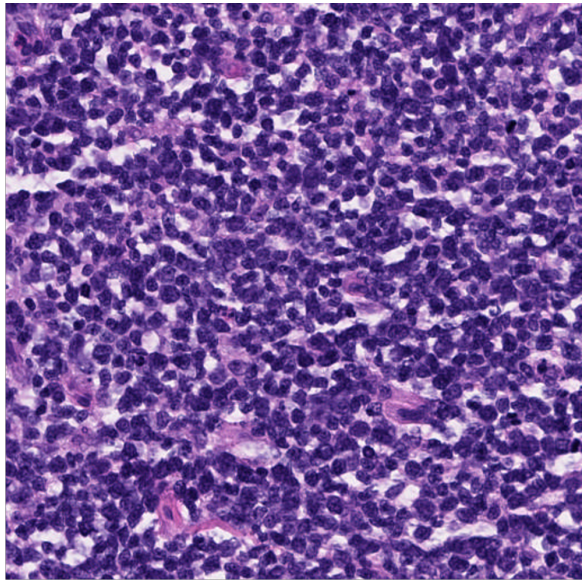
The train, dev, and eval sets for dataset #10

Dataset #11

- The cancer images for dataset #11 were extracted from Temple University Digital Pathology (TUDP) database, which contains whole slide images (WSI) of cancerous and non-cancerous tissues.
- The breast cancer slides in TUDP can contain two types of cancer: invasive ductal carcinoma in situ (indc) and ductal carcinoma in situ (dcis).
- The image patches are of size 512×512 pixels.
- The WSIs for each individual set were selected before the patch extraction so that overlaps among training, validation, and testing are avoided.



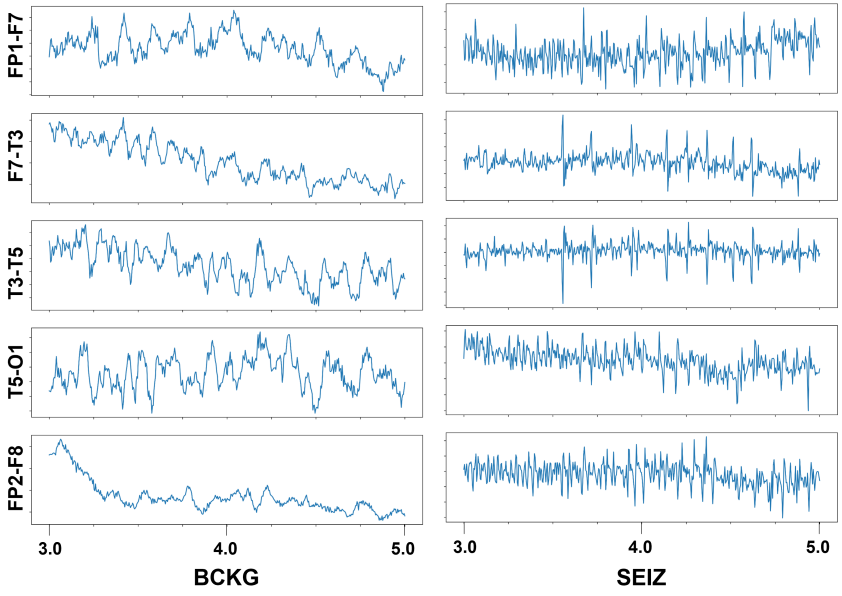
indc



dcis

Dataset #12

- Dataset #12 contains time series data collected from electroencephalograph (EEG) records in Temple University Hospital EEG Seizure Corpus (TUSZ).
- Seizure detection is a very challenging problem because the signals are noisy and contain numerous artifacts.
- The files are annotated with two classes which are seizure (SEIZ) and background (BCKG).
- We selected 20 channels from the EEG records and separated the segments that were more than 15 seconds long.
- The segments that were 10-seconds long were extracted from these by avoiding overlaps in the longer segments.
- Like Dataset #11, the files for the subsets were selected beforehand so that there were no overlap.



Five channels from typical 2-second, 20-channel background and seizure samples

Preprocessing

- The synthetic datasets required no preprocessing.
- For creating non-deep learning baselines and establishing the Bayes Error Rate (BER) on the synthetic datasets, we used k-nearest neighbors (KNN) and random forests (RNF).
- The images in dataset #11, the images were preprocessed by random cropping, jittering, rotating, and normalizing.
- The EEG signals in dataset #12 were filtered within the range $0.5 - 35$ Hz using a bandpass filter and then downsampled to 50 Hz.
- We then applied framing and windowing to extract 0.1 second frames with 0.3 second windows.
- Both autotuned and manual tuned models were the fed the same preprocessed data.

Hardware Description

- Google Colab was used for the first three datasets for autotuning experiments (using an Intel Xeon 2.30 Hz processor and NVIDIA K80 and T4 GPUs).
- The manual tuning experiments for those datasets were executed on AMD Opteron 2.40 GHz and Intel Xeon 2.20 GHz processors.
- The experiments on the EEG and cancer datasets for both manual and autotuning were performed on GPUs (NVIDIA RTX 2080).

Experimental Design

- The manual tuning was performed using PyTorch.
- The autotuning experiments used the Keras Tuner Toolkit where the setup process began with the definition of the hyperparameter search space Λ and a hypermodel to better encapsulate the search space.
- The hypermodel uses a randomly drawn set of hyperparameters to build a trainable model that includes the architecture (e.g., number of layers and neurons per layer) and the learning process (e.g., loss function and optimizer).
- For the synthetic dataset, both auto and manual tuned models used multilayer perceptron (MLP).
- We used convolutional neural network models (ConvNets) for dataset #11.
- For dataset #12, the manual method tuned a hybrid model containing both convolutional and long short term memory layers while the autotuning method used LSTM layers.

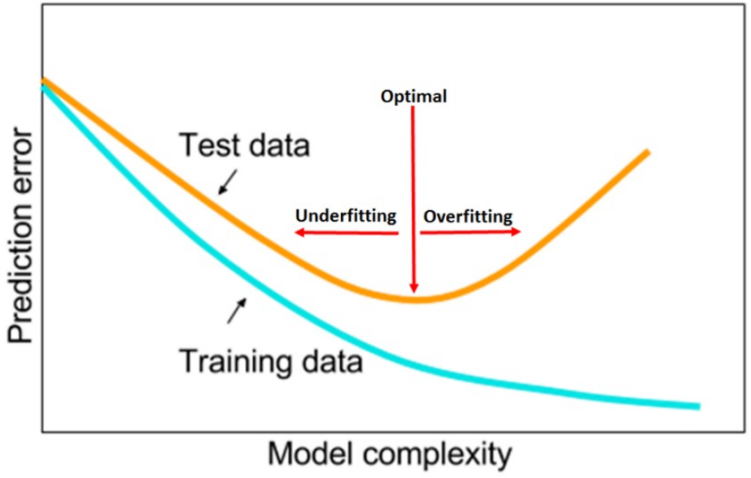
Results

- We used error rate as the performance comparison metric.
- BER estimate of the lowest achievable error rate for that set that was calculated by performing closed-loop training with KNN and RNF for the closed-loop training and testing on train, dev, and eval sets independently.
- MLP-M denotes manually tuned models and MLP-A stands for autotuned models.
- For dataset #10, MLP-M1 and MLP-A1 denote models without dropout and MLP-M2 and MLP-A2 denote models with dropout.
- The error rates for the manually tuned models were lower than those for the autotuned models (shaded in red).

DS	System	Train	Dev	Eval
#08	KNN	23.48	26.62	64.18
	RNF	29.23	29.45	59.77
	MLP-M	32.56	32.29	56.39
	MLP-A	36.42	30.70	57.38
	BER	23.48	24.45	20.07
#09	KNN	2.11	3.81	16.63
	RNF	2.06	3.82	18.32
	MLP-M	2.21	3.91	12.87
	MLP-A	3.70	5.30	14.22
	BER	2.06	2.30	1.85
#10	KNN	7.63	38.83	33.44
	RNF	2.15	39.74	33.28
	MLP-M1	8.74	38.52	32.80
	MLP-M2	9.91	40.88	32.00
	MLP-A1	28.95	26.95	42.07
	MLP-A2	12.42	40.11	32.34
	BER	2.15	11.70	13.74
#11	Manual	12.17	12.56	20.88
	Auto	35.28	24.28	41.55
#12	Manual	24.85	27.32	36.08
	Auto	50.00	50.00	50.00

Comparison of Model Architectures for Dataset #10

- With 4 layers, MLP-M1 achieved 9.27% less error rate on the eval set compared to the MLP-A1 which contains 6 layers.
- The performances of MLP-M2 and MLP-A2 are comparable which the former achieved with 3 layers whereas the latter contains 5 layers.
- This reduction in computational complexity suggests that the manually tuned models are less prone to overfitting and generalizes better to unseen data.

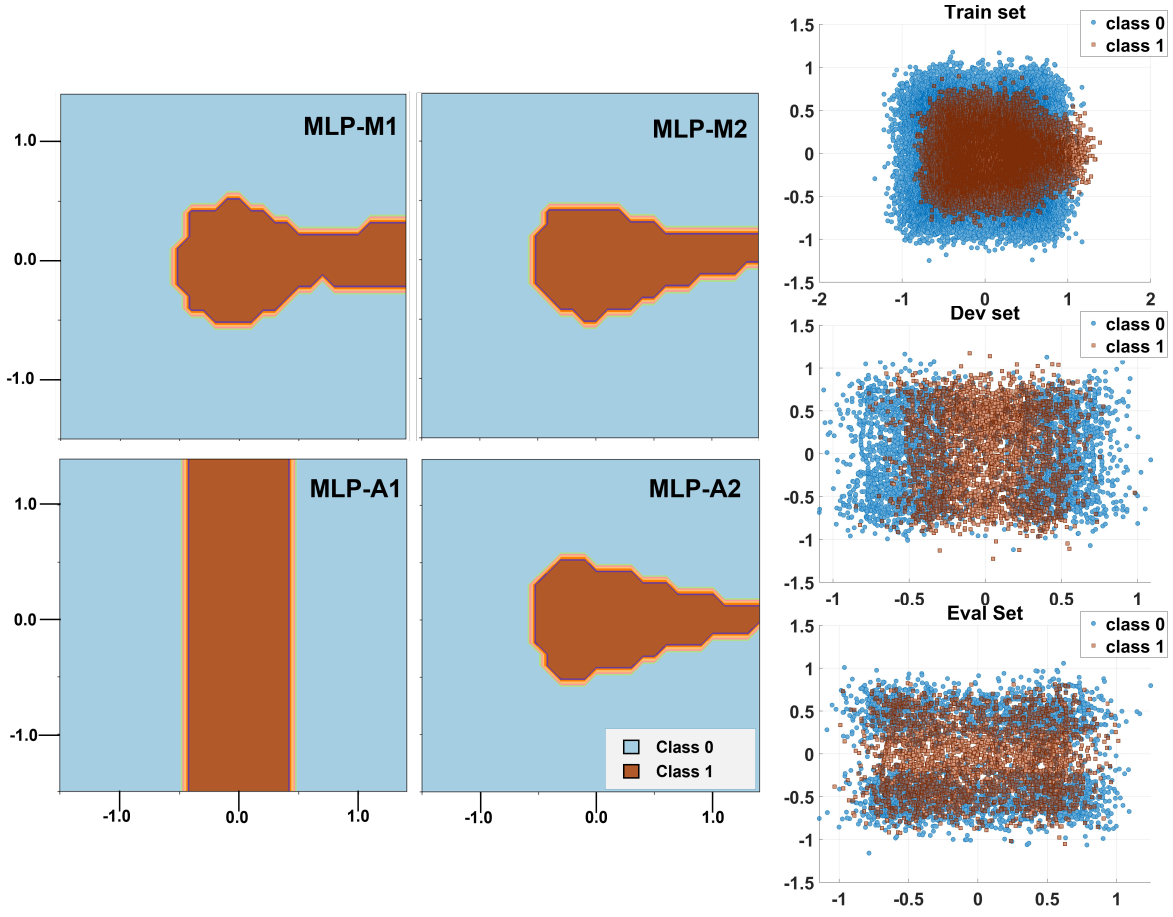


System	Train	Dev	Eval
MLP-M1	8.74	38.52	32.80
MLP-M2	9.91	40.88	32.00
MLP-A1	28.95	26.95	42.07
MLP-A2	12.42	40.11	32.34

Layout	MLP-M1 (PyTorch)	MLP-M2 (PyTorch)	MLP-A1 (Keras)	MLP-A2 (Keras)
Input Layer	Linear (2, 64), ReLU	Linear (2, 64), ReLU, Dropout (0.3)	Dense (52), sigmoid	Dense (97), tanh, dropout (0.25)
Layer 1	Linear (64, 32), ReLU,	Linear (64, 32), ReLU, dropout (0.3)	Dense (92), sigmoid	Dense (47), sigmoid, dropout (0.25)
Layer 2	Linear (32, 16), ReLU,	–	Dense (62), sigmoid	Dense (62), exponential, dropout (0.25)
Layer 3	–	–	Dense (72), sigmoid	Dense (37), tanh, dropout (0.25)
Layer 4	–	–	Dense (67), tanh	–
Output Layer	Linear (16,2)	Linear (32,2)	Dense (2), softmax	Dense (2), sigmoid

Decision Surface Analysis for Dataset #10

- The autotuned model without dropout, MLP-A1, overfitted the dev set while the manually tuned model, MLP-M1, produced a more generalized solution.
- The autotuner selected a model by aggressively reducing the error on the validation set, which caused a dip in performance on the eval set.
- During manual tuning, such performance issues were avoided by carefully reviewing both the error rates and the decision surfaces on the train and dev sets.



Comparison of Model Architectures for Dataset #11

- The manually tuned model for the cancer dataset is simpler compared to the autotuned model.
- However, the autotuner tended to fail when the random search drew a problematic set of parameters.
- The errors seemed to be difficult to resolve for users with limited knowledge in this field.
- The parameter search space for the autotuner was refined to ensure a complete exploration without running into bad combinations.

Layout	Manual (PyTorch)	Auto (Keras)
Input Layer	Conv2d (3, 16, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,8), tanh, MaxPool2d (5,5)
Layer 1	Conv2d (16, 32, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,5), tanh, MaxPool2d (3,3) Dropout (0.25)
Layer 2	Conv2d (32, 64, 5), ReLU, MaxPool2d (2, 2)	Conv2d (1,5), tanh MaxPool2d (3,3) Dropout (0.25)
Layer 3	Linear (9216, 128), ReLU, dropout (0.25)	Dense (25), ReLU
Layer 4	-	Dense (25), ReLU
Layer 5	-	Dense (25), ReLU
Output Layer	Linear (128, 2)	Dense (2), softmax
Optimizer	SGD	Adam
LR	0.001	0.001
Epoch	15	10
bsize	8	32

Observations on Dataset #12

- The EEG dataset, #12, was the most difficult dataset among all these datasets because it requires modeling of both spatial and temporal contexts.
- For manual tuning, we used a hybrid model with two convolutional layers to capture the spatial context and two LSTM layers to capture the temporal context.
- It had been difficult to tune models with only LSTM layers for autotuning.

DS	System	Train	Dev	Eval
#12	Manual	24.85	27.32	36.08
	Auto	50.00	50.00	50.00

- The autotuner developed many solutions (hyperparameter sets), however, due to the high complexity of the EEG dataset the developed models were unable to converge to a good solution. This resulted in the best performing model only achieving 50% accuracy by predicting the same label for all samples in the balanced subsets.

Summary

- Obtaining a generalized solution of a deep learning problem is a complex task.
- Autotuners were developed with the goal of democratizing state-of-the-art machine learning approaches and increasing their accessibility to different scientific communities.
- We investigated automatic hyperparameter tuning using the Keras Tuner toolkit.
- For synthetic data, the performance of auto and manually tuned models are comparable.
- For more complex datasets, it was difficult to perform autotuning, as it failed when it reached a problematic combination of hyperparameters.
- Our observations suggest that limited knowledge on deep learning processes (e.g., lack of components like dropout) can lead to degraded performance due to the use of incomplete hyperparameter search spaces.
- Future work will involves experimenting with more autotuning tools such as Ray Tune, Optuna, Google Vizier, and Microsoft Neural Network Intelligence (NNI) to provide a better analysis of autotuning techniques.

Acknowledgments

This material is supported by the National Science Foundation under grants nos. CNS-1726188 and 1925494. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Open-source libraries that were used to develop the IMLD are: NumPy v1.15.4, PyQT5 v5.13.1, SkLearn v0.20.0, Scipy v1.1.0, and Matplotlib v3.0.2.

References

- M. Feurer and F. Hutter, "Hyperparameter Optimization," in *Automated Machine Learning: Methods, Systems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer International Publishing, 2019, pp. 3-33. *https://link.springer. com/chapter/10.1007/978-3-030-05318-5_1*.

- J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *J. Mach. Learn. Res.,* vol. 13, pp. 281-305, Feb. 2012. ˙ *https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a*.

- S. Pouyanfar *et al.*, "A Survey on Deep Learning: Algorithms, Techniques, and Applications," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 1-36, Sep. 2018. ˙ *https://dl.acm.org/doi/abs/10.1145/ 3295748*.

- C. Dossman, "Top 6 Errors Novice Machine Learning Engineers Make," *AI: Theory, Practice, Business*, 2021. [Online]. Available: *https://medium.com/ai^3-theory-practice-business/top-6-errors-novice-machine-learning-engineers-make-e82273d394db*. [Accessed: 13-Oct-2021].

- T. O'Malley et al., "Keras Tuner," 2019. [Online]. Available: *https://keras.io/keras_tuner*. [Accessed: 29-Jul-2021].

- T. Cap, A. Kreitzer, M. Miranda, D. Vadimsky, and J. Picone, "IMLD: A Python-Based Interactive Machine Learning Demonstration," in *Proceedings of the IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2021, pp. 1-4. *https://www.isip.piconepress.com/publications/conference_presentations/2021/ieee_spmb/imld/*.

- N. Shawki et al., "The Temple University Digital Pathology Corpus," in *Signal Processing in Medicine and Biology: Emerging Trends in Research and Applications*, 1st ed., I. Obeid, I. Selesnick, and J. Picone, Eds. New York City, New York, USA: Springer, 2020, pp. 67-104. ˙ *https://www.springer.com/gp/book/ 9783030368432*.

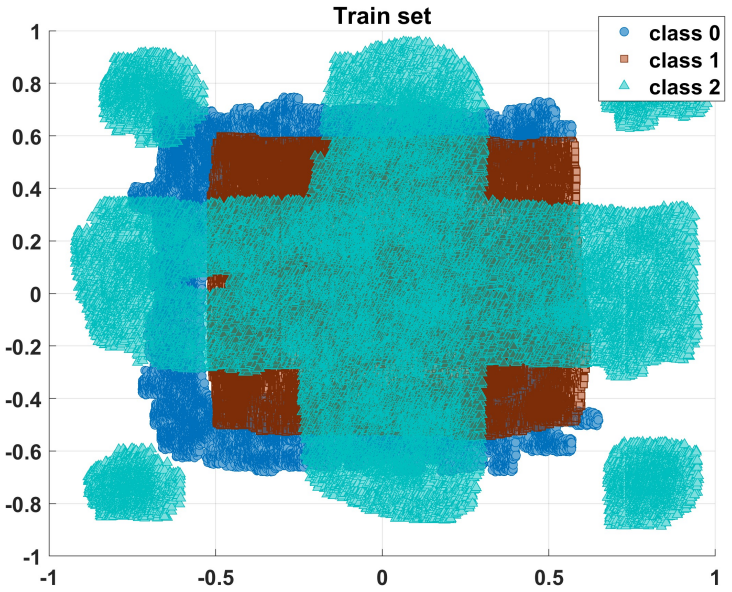
- V. Shah et al., "The Temple University Hospital Seizure Detection Corpus," *Front. Neuroinform.,* vol. 12, pp. 1-6, 2018. ˙ *https://www.frontiersin.org/articles/10.3389/fninf.2018.00083/full*.

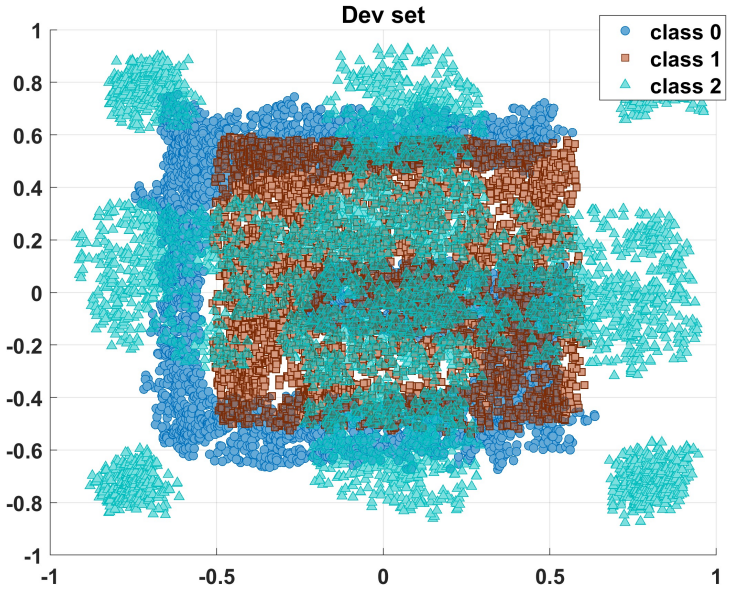
- K. Tumer and J. Ghosh, "Estimating the Bayes error rate through classifier combining," in *Proceedings of 13th International Conference on Pattern Recognition*, 1996, vol. 2, pp. 695-699. *https://ieeexplore.ieee.org/document/546912*.

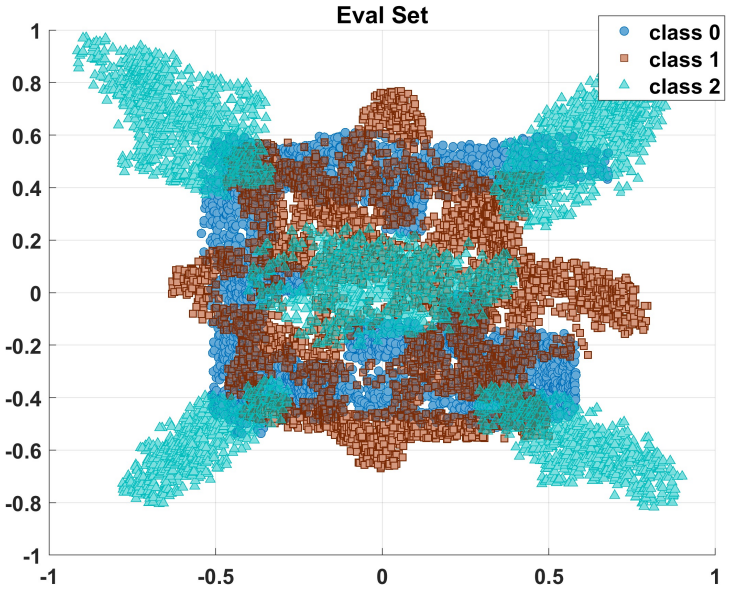
- L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi, "Sherpa: Robust hyperparameter optimization for machine learning," *SoftwareX*, vol. 12, p. 100591, 2020. ˙ *https://www.sciencedirect. com/ science/article/pii/S2352711020303046*.

Dataset #08

- Dataset #08 contains three classes with complex distributions requiring a model to find an optimal non-linear decision surface.
- The subsets were manually created using IMLD.
- Each class contains two features.

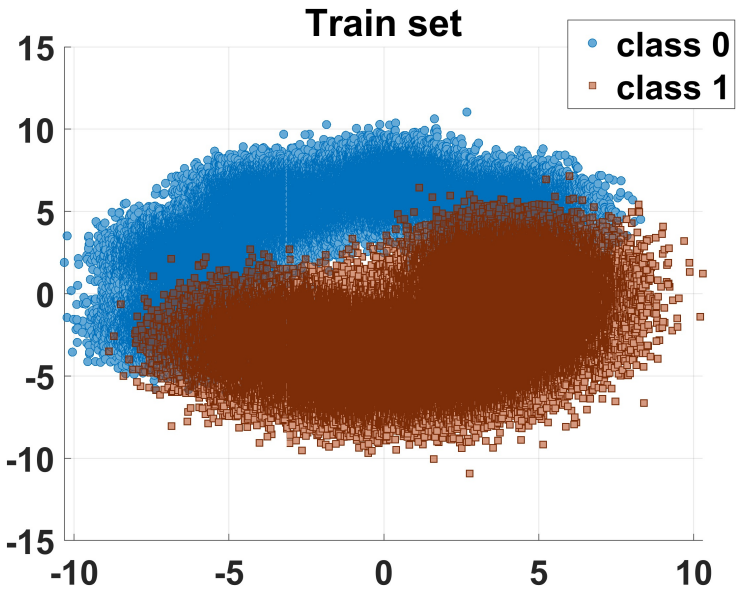


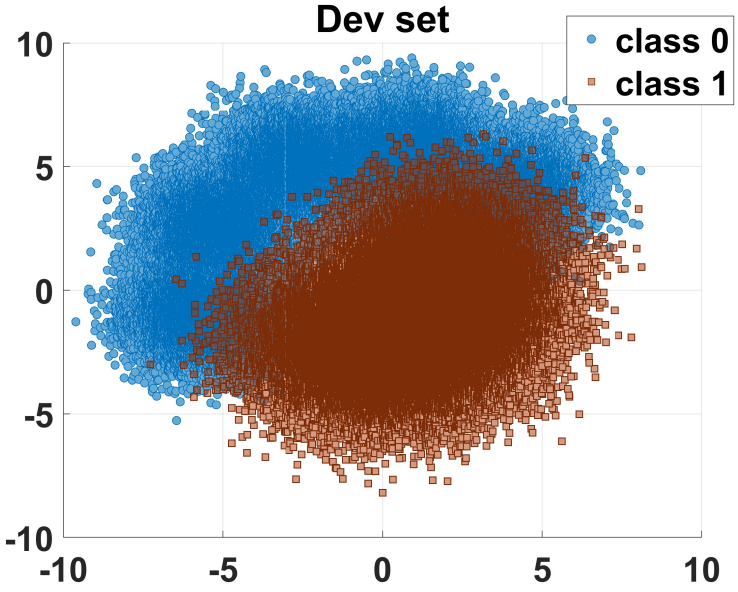


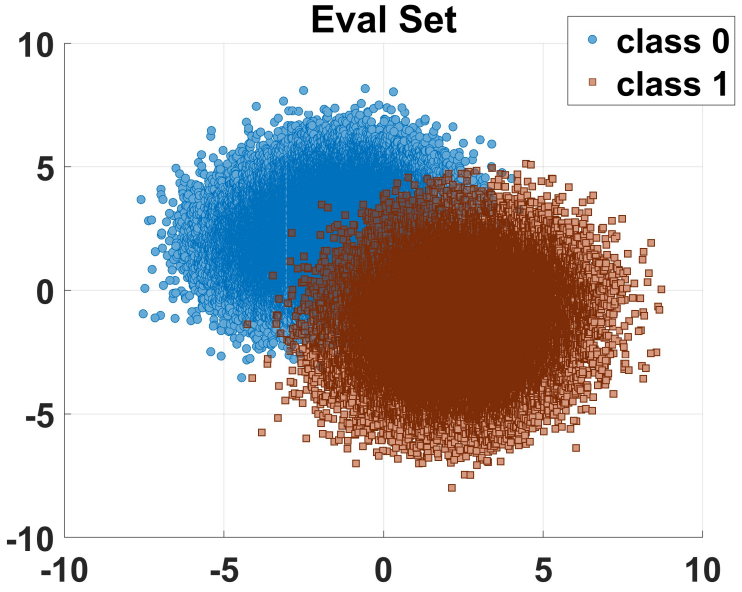


Dataset #09

- The train set for dataset #09 was created with five tight equally-spaced Gaussian distributions.
- The dev and eval sets each contain two such Gaussian distributions.
- Each class contains two features.

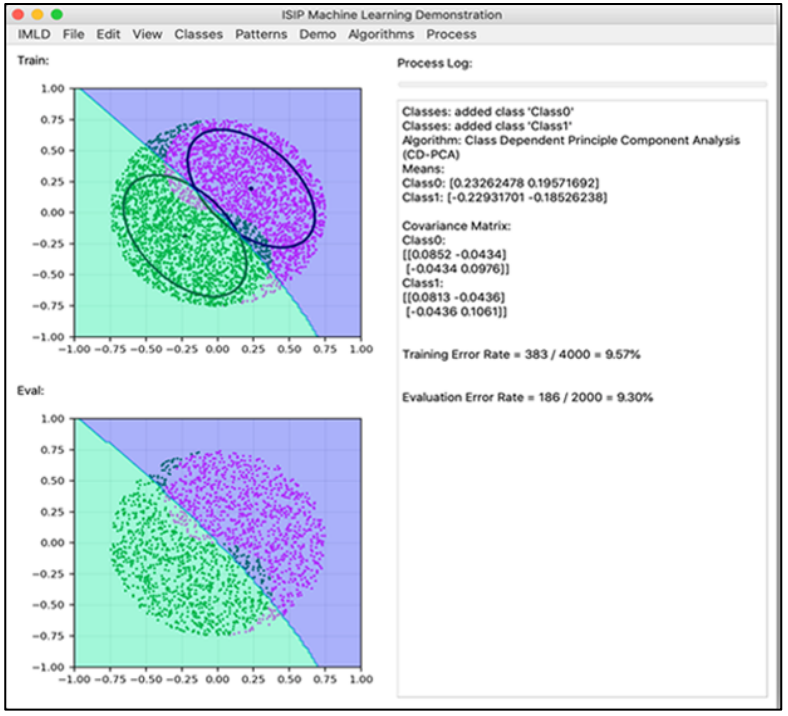






ISIP Machine Learning Demo - IMLD

- The ISIP Machine Learning Demo (IMLD) is a robust, visual, and interactive tool designed to assist in machine learning education.
- Generate Data: Users have the option to create data by drawing with the cursor or importing data using predefined statistical functions (e.g., Gaussian, toroidal and yin-yang distributions).
- Import/Export Data: Users have the option to import and export datasets through a comma separated value (CSV) file format.
- Customize Dataset: Users are given the opportunity to set the total number of data points as well as key parameters of each generator (e.g., mean, covariance).
- Apply Algorithms: Algorithms can be applied to datasets and the results are plotted to the desired canvas. IMLD will calculate and display the means, covariance matrices of all classes and the corresponding error rates.



T. Cap, A. Kreitzer, M. Miranda, D. Vadimsky, and J. Picone, "IMLD: A Python-Based Interactive Machine Learning Demonstration," in *Proceedings of the IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, 2021, pp. 1-4. *https://www.isip.piconepress.com/publications/conference_presentations/2021/ieee_spmb/imld/*.