CSC510 SOFTWARE ENGINEERING

AI + SE

November 1, 2023

Rahul Yedida

Department of Computer Science
NC State University

WHAT YOU WILL LEARN TODAY

→ Slightly fast-paced: we have a lot of ground to cover!
→ The importance of theory in applied ML, such as in SE
→ The temporal delay between AI venues (NeurIPS, ICLR, ICML, JMLR, etc.) and SE venues (ICSE, TSE, JSS, etc.)
→ The ML4SE process
→ A gentle introduction to SE4ML

OVERVIEW

1. The varying perspectives

2. Examples

3. The AI4SE process

4. SE4ML

THEORETICAL VS APPLIED ML

→ Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.
  - What is the generalization error of my model? PAC bounds.

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.
  - What is the generalization error of my model? PAC bounds.
  - How well does my model tolerate noise? Rademacher complexity. See Zhang et al. [20]

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.
  - What is the generalization error of my model? PAC bounds.
  - How well does my model tolerate noise? Rademacher complexity. See Zhang et al. [20]
  - What properties does my loss function have? Smoothness, Lipschitzness, convexity, strong convexity

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.
  - What is the generalization error of my model? PAC bounds.
  - How well does my model tolerate noise? Rademacher complexity. See Zhang et al. [20]
  - What properties does my loss function have? Smoothness, Lipschitzness, convexity, strong convexity
- Applied ML focuses on results, with lesser focus on why something works.
  - Some papers, sadly, still resort to “kitchen-sink" learning

THEORETICAL VS APPLIED ML

- Theoretical ML uses strong mathematical foundations to provide guarantees and results about ML algorithms.
  - What is the representation capacity of the class of learning algorithms I'm using? VC dimension.
  - What is the generalization error of my model? PAC bounds.
  - How well does my model tolerate noise? Rademacher complexity. See Zhang et al. [20]
  - What properties does my loss function have? Smoothness, Lipschitzness, convexity, strong convexity
- Applied ML focuses on results, with lesser focus on why something works.
  - Some papers, sadly, still resort to “kitchen-sink" learning
  - However, some papers use intuitions (not backed by theory) to understand certain behaviors of algorithms, and devise ways to circumvent or prevent it [18].

AI FOR SE VS. AI FOR SE

- → A slightly philosophical approach: which do you treat as a first-class citizen?
- → More precisely, how much domain knowledge do you use in your learning?

THEORETICAL AI IN SE

→ Monolithic architecture to microservices: Desai et al. [6]

$$\min_{W,O,M,C} \mathcal{L}_{total} = \alpha_1 \mathcal{L}_{str} + \alpha_2 \mathcal{L}_{attr} + \alpha_3 \mathcal{L}_{clus}$$

$$\text{such that } \sum_{i \in V} O_{si} = \sum_{i \in V} O_{ai} = 1$$

$$M \in \{0, 1\}^{|V| n \times K}, O_{si}, O_{ai} > 0$$

THEORETICAL AI IN SE

- Monolithic architecture to microservices: Desai et al. [6]

$$\min_{W,O,M,C} \mathcal{L}_{total} = \alpha_1 \mathcal{L}_{str} + \alpha_2 \mathcal{L}_{attr} + \alpha_3 \mathcal{L}_{clus}$$
$$\text{such that } \sum_{i \in V} O_{si} = \sum_{i \in V} O_{ai} = 1$$
$$M \in \{0,1\}^{|V| n \times K}, O_{si}, O_{ai} > 0$$

- Feedforward networks work very well: Yedida and Menzies [18]:
  - From Hornik [10]: If ψ is unbounded and nonconstant, then $\mathfrak{R}(\psi)$ is dense in $L^p(\mu)$ for all finite measures μ on $\mathbb{R}^k$.
  - From Montufar et al. [14] The maximal number of linear regions of the functions computed by a neural network with n_0 input units and L hidden layers, with $n_i \geq n_0$ rectifiers at the i-th layer, is lower bounded by

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0} \right) \sum_{j=0}^{n_0} \binom{n_L}{j}$$

THEORETICAL AI IN SE

- Monolithic architecture to microservices: Desai et al. [6]

$$\min_{W,O,M,C} \mathcal{L}_{total} = \alpha_1 \mathcal{L}_{str} + \alpha_2 \mathcal{L}_{attr} + \alpha_3 \mathcal{L}_{clus}$$

$$\text{such that } \sum_{i \in V} O_{si} = \sum_{i \in V} O_{ai} = 1$$

$$M \in \{0,1\}^{|V|n \times K}, O_{si}, O_{ai} > 0$$

- Feedforward networks work very well: Yedida and Menzies [18]:
  - From Hornik [10]: If ψ is unbounded and nonconstant, then $\mathfrak{R}(\psi)$ is dense in $L^p(\mu)$ for all finite measures μ on $\mathbb{R}^k$.
  - From Montufar et al. [14] The maximal number of linear regions of the functions computed by a neural network with n_0 input units and L hidden layers, with $n_i \geq n_0$ rectifiers at the i-th layer, is lower bounded by

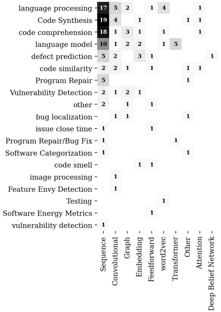
$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{n_0} \right\rfloor^{n_0} \right) \sum_{j=0}^{n_0} \binom{n_L}{j}$$

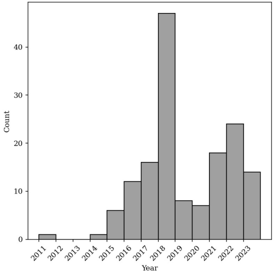
- Our current work: hyper-parameter optimization using smoothness of the loss function.

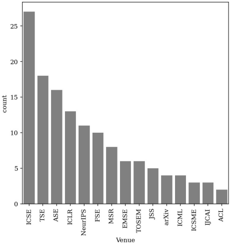
APPLIED AI IN SE

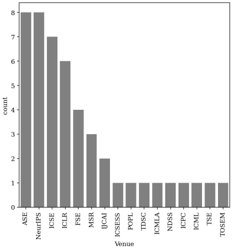
- → 95% of papers in AI for SE
- → Xu et al. [17]: Stack Overflow posts to vectors
- → Prenner and Robbes [16]: Transformer models are competitive when used for small SE datasets, especially for natural language.
- → Agrawal et al. [1]: A novel hyper-parameter optimization that relies on the $\epsilon-$domination principle.

DL FOR SE: A BROAD VIEW









DATA COLLECTION

- Two main ways: static attributes and automated methods
- Static methods:
  - Example: the PROMISE repository [13]
  - Rather out of fashion lately
- Automated methods:
  - Source-code based, using Transformer architectures [7, 8, 12]
  - Graph neural networks: [15, 19]

LEARNING ALGORITHMS I

- Gaussian Discriminant Analysis: when your data comes from Gaussian distributions.
  - A fun example from computational neuroscience: decision making in the visual cortex uses a likelihood ratio to discriminate between signals and noise (why? see Neyman-Pearson lemma) [3, 5]
  -

$$
\begin{aligned}
y &\sim \text{Bernoulli}(\phi) \\
\mathrm{x}|y = 0 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}) \\
\mathrm{x}|y = 1 &\sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})
\end{aligned}
$$

LEARNING ALGORITHMS II

- Support Vector Machines: when you can take advantage of kernel learning and want a maximum-margin classifier.

$$\max_{\boldsymbol{\lambda}} \sum_{i=1}^{m} \lambda_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y^{(i)} y^{(j)} \left\langle x^{(i)}, x^{(j)} \right\rangle$$

$$\text{s.t. } \lambda_i \geq 0$$

$$\sum_{i=1}^{m} \lambda_i y^{(i)} = 0$$

- Decision trees: When you need interpretable decision-making
  - Very useful in the context of knowledge distillation: Liu et al. [11]
- Neural networks: when you want automated feature extraction, want to make use of pre-trained models, or want a higher VC-dimension.

LEARNING ALGORITHMS III

- → Deep ReLU networks have a much higher VC-dimension than classical models: Bartlett et al. [2], Harvey et al. [9]: VC-dimension is $\mathcal{O}(WL \log W)$ or $\Theta(WU)$
- → They can require much more computational power, however.

ANALYSIS

- Statistics!
  - Your results mean nothing without stats to back them up.
  - Examples: t-test (parametric, assumes t-distributed), Kruskal-Wallis (non-parametric, independence), ANOVA (parametric, homogeneity of variance), Mann-Whitney (non-parametric, independence),
  - Mantel-Haenszel: for testing the association between two categorical variables (example: smoking/not smoking vs. lung cancer/cancer-free), when controlling for a confounding variable (example: age)
- Pretty graphs
  - Even peer-reviewers want something pretty to look at.
  - Common tools: Matplotlib, Seaborn, plot.ly

AN END-TO-END EXAMPLE: DEFECT PREDICTION

- Data: We will use the PROMISE repository.
  - Simple, tabular.

AN END-TO-END EXAMPLE: DEFECT PREDICTION

- Data: We will use the PROMISE repository.
  - Simple, tabular.
- Preprocessing: Some standard, some not.
  - Normalizing, standardizing, min-max scaling, etc.
  - SMOTE [4]
  - Still not enough!
    - Let's fast-forward to 300 awkward failed attempts later...
    - Fuzzy sampling and "ultrasampling" (see Yedida and Menzies [18])

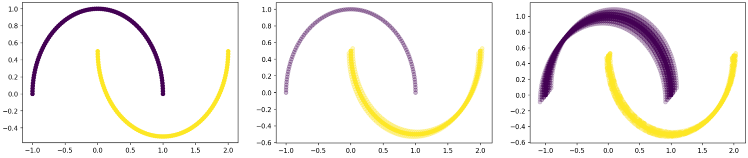
AN END-TO-END EXAMPLE: DEFECT PREDICTION

- Data: We will use the PROMISE repository.
  - Simple, tabular.
- Preprocessing: Some standard, some not.
  - Normalizing, standardizing, min-max scaling, etc.
  - SMOTE [4]
  - Still not enough!
    - Let's fast-forward to 300 awkward failed attempts later...
    - Fuzzy sampling and "ultrasampling" (see Yedida and Menzies [18])
- Learning system: Feedforward networks
  - As discussed previously, we can exploit theoretical results (Hornik [10], Montufar et al. [14])
  - Fast to train and iterate on.

AN END-TO-END EXAMPLE: DEFECT PREDICTION

- Data: We will use the PROMISE repository.
  - Simple, tabular.
- Preprocessing: Some standard, some not.
  - Normalizing, standardizing, min-max scaling, etc.
  - SMOTE [4]
  - Still not enough!
    - Let's fast-forward to 300 awkward failed attempts later...
    - Fuzzy sampling and "ultrasampling" (see Yedida and Menzies [18])
- Learning system: Feedforward networks
  - As discussed previously, we can exploit theoretical results (Hornik [10], Montufar et al. [14])
  - Fast to train and iterate on.
- Compute: Slurm cluster

FUZZY SAMPLING



(a) Original data (b) Fuzzy sampling (c) Ultrasampling

HYPER-PARAMETER OPTIMIZATION —

Theorem (Smoothness for feedforward networks)

For a feedforward network with ReLU activations in the hidden layers and a softmax activation at the last layer, the $\beta-$smoothness of the cross-entropy loss is given by

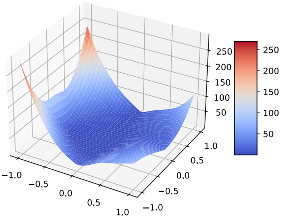
$$\|\nabla_W^2 E\| \leq \frac{k-1}{km} \max \frac{\|a_j^{L-1}\|}{\|W^{[L]}\|} \tag{2}$$

There is, of course, some theoretical backing for using the maximum smoothness: Let $f(X,h) = \|\nabla^2 \mathcal{L}(\cdot)\|$

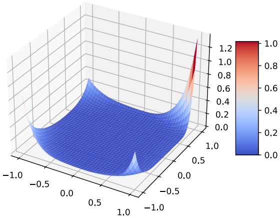
$$\max_h \min_X f(X,h) \leq \min_h \max_X f(X,h) \leq \max_h \max_X f(X,h) \tag{1}$$

where the left part is the $\mu-$strong convexity (exercise: prove this).

MAXIMIZING SMOOTHNESS



(a) Before HPO



(b) After HPO

STATISTICS

- → We will compare with the state-of-the-art, along with other SOTA HPO methods
- → Kruskal-Wallis test (group medians), followed by pairwise Mann-Whitney (Wilcoxon rank-sum) tests, and adjust p-values using Benjamini-Hochberg correction to adjust for Type-I errors

INTRODUCING: MLOPS

- → Software processes have evolved longer than ML coding practices
- → As such, a lot of ML code is...subpar (ironically, this seems especially true in SE venues)
- → MLOps abstracts model and data versioning, experiment tracking, dashboarding, and more
- → We will look at a small example using MLFlow: **https://github.com/yrahul3910/mlflow-demo**

USING MLFLOW

→ Set up an experiment:

```
# Create an experiment
experiment_id = mlflow.create_experiment('mnist')
experiment = mlflow.get_experiment(experiment_id)
print('Artifact location:', experiment.artifact_location)
```

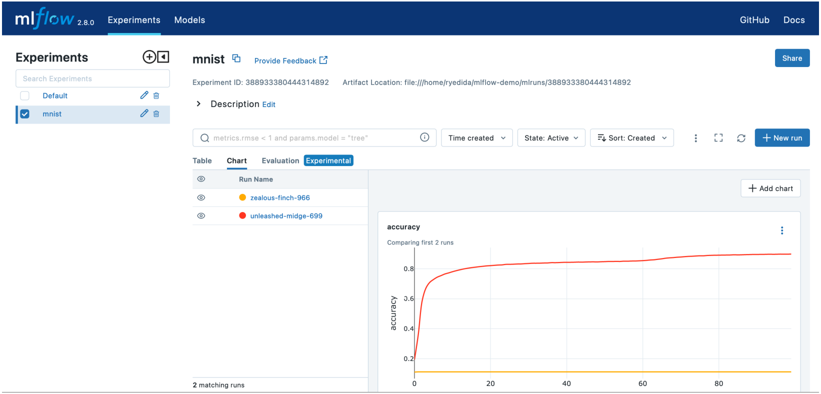
→ Your main loop:

```
with mlflow.start_run(experiment_id=experiment_id):
    # Get the config
    config = get_hyperparams(choices)

    # Log parameters
    for param, val in config.items():
        mlflow.log_param(param, val)

    val_acc = run_experiment(X_train, X_test, y_train, y_test, config)
    mlflow.log_metric('val_acc', val_acc)
```

→ Start the UI server

```
mlflow ui --host 0.0.0.0 --port 5000
```

MLFLOW UI


mlflow 2.8.0
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
mnist
mnist
Provide Feedback
Share
Experiment ID: 388933380444314892
Artifact Location: file:///home/ryedida/mlflow-demo/mlruns/388933380444314892
Description Edit
metrics.rmse < 1 and params.model = "tree"
Time created
State: Active
Sort: Created
New run
Table
Chart
Evaluation
Experimental
Run Name
zealous-finch-966
unleashed-midge-699
2 matching runs
Add chart
accuracy
Comparing first 2 runs
accuracy
0.8
0.6
0.4
0.2
0
20
40
60
80

CLOUD COMPUTING

- Model training: With deeper models, cloud computing may be more cost-effective.
  - Example: on Google Cloud, an A100 with 85GB RAM and 12 vCPUs costs about $89/day.
  - Niceties such as multi-GPU training and flexible instance scaling makes training very convenient.
  - Notebooks can be used too: AWS SageMaker
- Data sharing:
  - When your data size gets excessive, it might be better to store it on the cloud.
  - Example: AWS S3, Google Cloud Storage

BIBLIOGRAPHY I

[1] Amritanshu Agrawal, Wei Fu, Di Chen, Xipeng Shen, and Tim Menzies. How to dodge complex software analytics. IEEE Transactions on Software Engineering, 47(10):2182–2194, 2019.

[2] Peter L Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. The Journal of Machine Learning Research, 20(1):2285–2301, 2019.

[3] Kenneth H Britten, Michael N Shadlen, William T Newsome, and J Anthony Movshon. The analysis of visual motion: a comparison of neuronal and psychophysical performance. Journal of Neuroscience, 12(12):4745–4765, 1992.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16:321–357, 2002.

BIBLIOGRAPHY II

[5] Peter Dayan and Laurence F Abbott. Theoretical neuroscience: computational and mathematical modeling of neural systems. MIT press, 2005.

[6] Utkarsh Desai, Sambaran Bandyopadhyay, and Srikanth Tamilselvam. Graph neural network to dilute outliers for refactoring monolith application. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 72–80, 2021.

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155, 2020.

[8] Shuzheng Gao, Cuiyun Gao, Yulan He, Jichuan Zeng, Lunyiu Nie, Xin Xia, and Michael Lyu. Code structure–guided transformer for source code summarization. ACM Transactions on Software Engineering and Methodology, 32(1):1–32, 2023.

BIBLIOGRAPHY III

[9] Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension bounds for piecewise linear neural networks. In Conference on learning theory, pages 1064–1068. PMLR, 2017.

[10] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251–257, 1991.

[11] Xuan Liu, Xiaoguang Wang, and Stan Matwin. Improving the interpretability of deep neural networks with knowledge distillation. In 2018 IEEE International Conference on Data Mining Workshops (ICDMW), pages 905–912. IEEE, 2018.

[12] Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. Studying the usage of text-to-text transfer transformer to support code-related tasks. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pages 336–347. IEEE, 2021.

BIBLIOGRAPHY IV

[13] Tim Menzies, Bora Caglayan, Ekrem Kocaguneli, Joe Krall, Fayola Peters, and Burak Turhan. The promise repository of empirical software engineering data, 2012.

[14] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. Advances in neural information processing systems, 27, 2014.

[15] Aravind Nair, Avijit Roy, and Karl Meinke. funcgnn: A graph neural network approach to program similarity. In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–11, 2020.

[16] Julian Aron Prenner and Romain Robbes. Making the most of small software engineering datasets with modern machine learning. IEEE Transactions on Software Engineering, 48(12): 5050–5067, 2021.

BIBLIOGRAPHY V

[17] Bowen Xu, Thong Hoang, Abhishek Sharma, Chengran Yang, Xin Xia, and David Lo. Post2vec: Learning distributed representations of stack overflow posts. IEEE Transactions on Software Engineering, 48(9):3423–3441, 2021.

[18] Rahul Yedida and Tim Menzies. On the value of oversampling for deep learning in software defect prediction. IEEE Transactions on Software Engineering, 48(8):3103–3116, 2021.

[19] Rahul Yedida, Rahul Krishna, Anup Kalia, Tim Menzies, Jin Xiao, and Maja Vukovic. An expert system for redesigning software for cloud applications. Expert Systems with Applications, 219:119673, 2023.

[20] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. Communications of the ACM, 64(3): 107–115, 2021.