Machine Learning Systems Design

Lecture 5: Model Selection, Development, and Training

Dr. José Ramón Iglesias
DSP-ASIC BUILDER GROUP
Director Semillero TRIAC
Ingenieria Electronica
Universidad Popular del Cesar

PSET collaboration policy

You can work in a group of 2 people, but **EACH MUST WRITE ANSWERS INDIVIDUALLY**

Agenda

1. Data leakage
2. How to engineer good features
3. Breakout exercise
4. Model selection
5. Ensembles
6. AutoML

Data leakage (ctd.)

Data leakage

- Some form of the label “leaks” into the features
- This same information is not available during inference

Data leakage: example 1

- Problem: detect lung cancer from CT scans
- Data: collected from hospital A
- Performs well on test data from hospital A
- Performs poorly on test data from hospital B

Patient ID	Date	Doctor note	Medical record	Scanner type	CT scan

At hospital A, when doctors suspect that a patient has lung cancer, they send that patient to a higher-quality scanner

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time

Partition: shuffle then split

	Week 1	Week 2	Week 3	Week 4	Week 5
Test split	X11	X21	X31	X41	X51
Valid split	X12	X22	X32	X42	X52
Train split	X13	X23	X33	X43	X53
	X14	X24	X34	X44	X54
	...	...	...	...	...

Aim for similar distributions of labels across splits
e.g. each split has 90% NEGATIVE, 10% POSITIVE

Partition: shuffle then split

	Week 1	Week 2	Week 3	Week 4	Week 5
Test split	X11	[illegible]	[illegible]	X41	X51
Valid split	X12	[illegible]	[illegible]	X42	X52
Train split	X13	[illegible]	[illegible]	X43	X53
	X14	X24	[illegible]	X44	X54
	...	...	...	...	...

⚠ Not representative of real-world usage! ⚠

Partition: shuffle then split

	Week 1	Week 2	Wee[illegible]	Week 4	Week 5
Test split	X11	[illegible]	[illegible]	X41	X51
Valid split	X12	X[illegible]	[illegible]2	X42	X52
Train split	X13	[illegible]	[illegible]	X43	X53
	X14	X24	X3[illegible]	X44	X54
	...	...	...	...	...

A source of data leakage. Examples:

- stock price prediction
- song recommendation

A better partition

Train split

Week 1	Week 2	Week 3	Week 4	Week 5	
X11	X21	X31	X41	X51	**Valid split**
X12	X22	X32	X42	X52	
X13	X23	X33	X43	X53	**Test split**
X14	X24	X34	X44	X54	
...	...	...	...	...	

Solution: split data by time

Train split

Week 1	Week 2	Week 3	Week 4	Week 5	
X11	X21	X31	X41	X51	Valid split
X12	X22	X32	X42	X52	
X13	X23	X33	X43	X53	Test split
X14	X24	X34	X44	X54	
...	...	...	...	...	

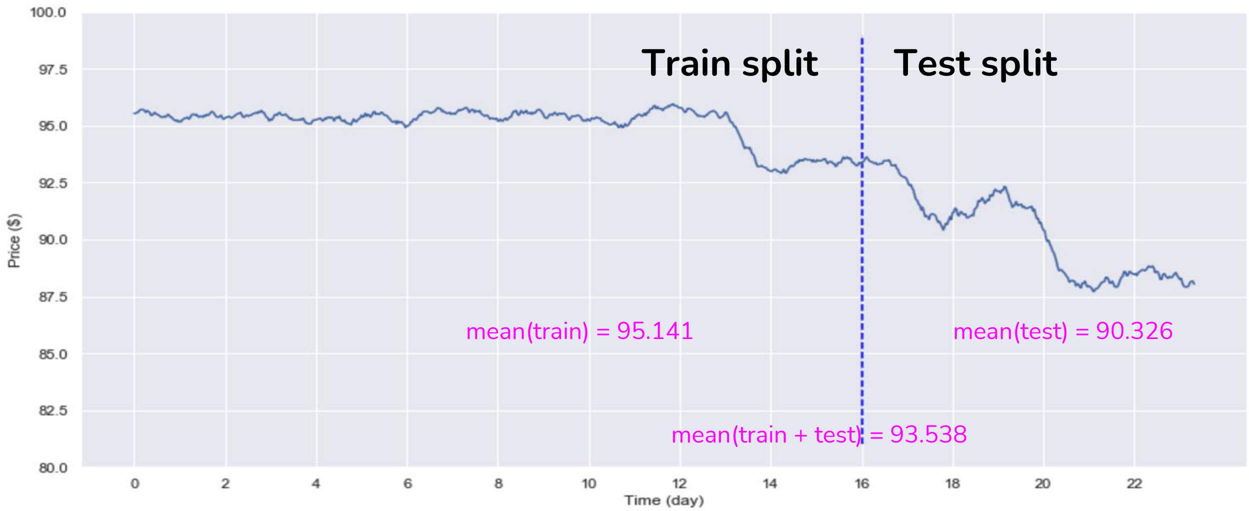
Also forces you to think about the cold-start problem

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
    a. Use the whole dataset (including valid/test) to generate global statistics/info

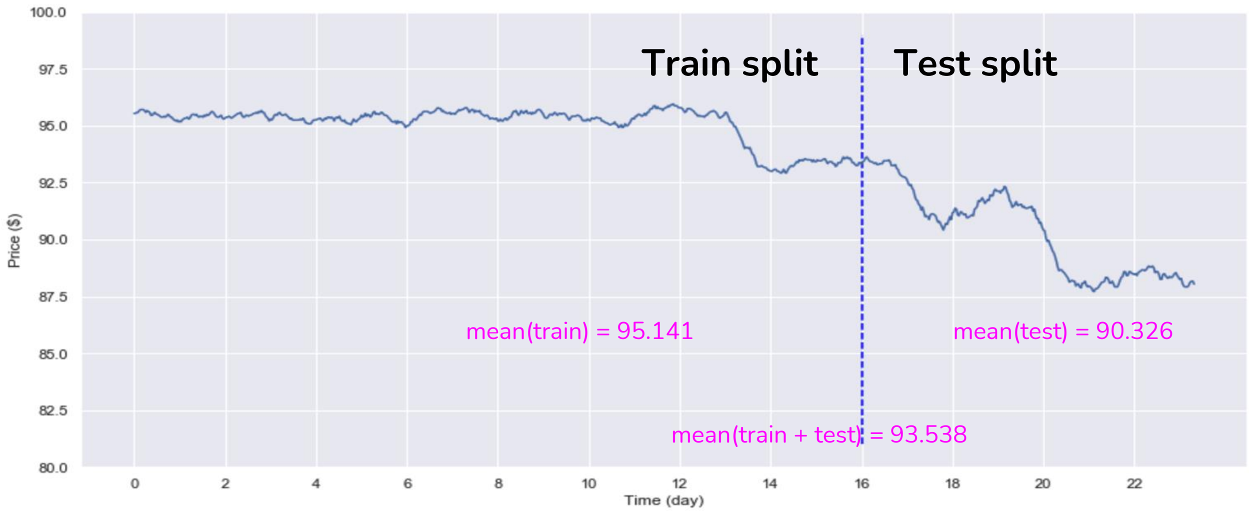
2. Data processing before splitting

- Use the whole dataset (including valid/test) to generate global statistics/info
  - mean, variance, min, max, n-gram count, vocabulary, etc.
- Statistics are then used to process test data
  - scale, fill in missing values, etc.



2. Data processing before splitting

- Use the whole dataset (including valid/test) to generate global statistics/info
- Solution:
  - Split your data before scaling/filling in missing values
  - Split even before any EDA to ensure you're blind to the test set

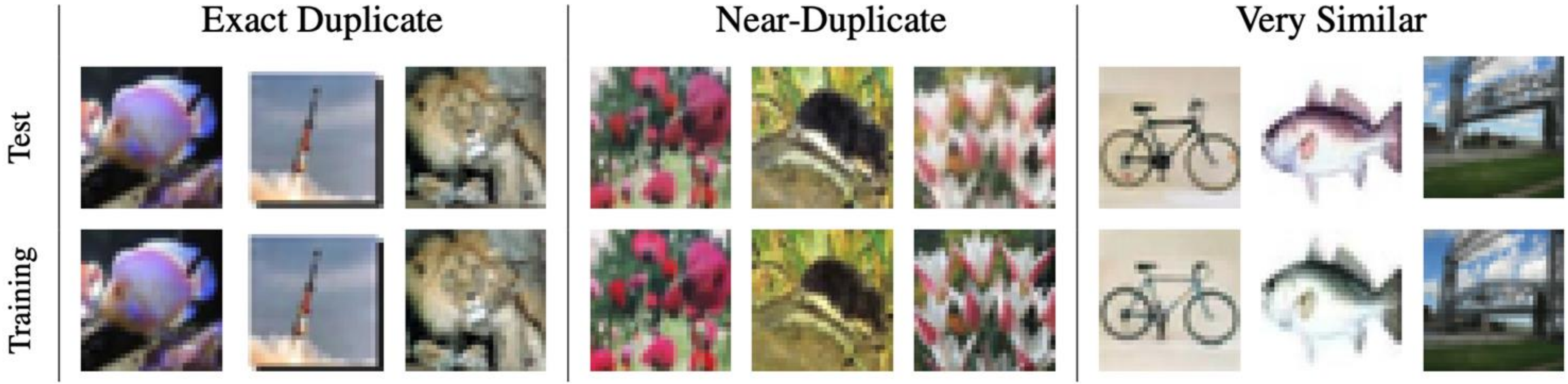


Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
    a. Test set includes data from the train set

3. Poor handling of data duplication before splitting

- Datasets come with duplicates & near-duplicates
  - 3.3% CIFAR-10 and 10% CIFAR-100 test images have dups in training set
  - Removing dups increases errors 17.05% -> 19.38% on CIFAR-100 [PyramidNet-272-200}



Do we train on test data? Purging CIFAR of near-duplicates (Barz & Denzler, 2019)

3. Poor handling of data duplication before splitting

- Datasets come with duplicates & near-duplicates
- Oversampling can cause duplications

3. Poor handling of data duplication before splitting

- Test set includes data from the train set
- Solution:
  - Deduplicate data before splitting
  - Oversample after splitting

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
    a. A group of examples have strongly correlated labels but are divided into different splits

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
   a. A group of examples have strongly correlated labels but are divided into different splits
   b. Example: CT scans of the same patient a week apart
   c. Solution: Understand your data and keep track of its metadata

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
5. Leakage from data generation & collection process
    a. Example: doctors send high-risk patients to a better scanner
    b. Solution: Data normalization + subject matter expertise

Causes of data leakage

1. Splitting time-correlated data randomly instead of by time
2. Data processing before splitting
3. Poor handling of data duplication before splitting
4. Group leakage
5. Leakage from data generation & collection process

How to detect leakage?

How to detect leakage?

1. Measure correlation of a feature with labels
    a. A feature alone might not cause leakage, but 2 features together might

How to detect leakage?

1. Measure correlation of a feature with labels
2. Feature ablation study
    a. If removing a feature causes the model performance to decrease significantly, figure out why.

How to detect leakage?

1. Measure correlation of a feature with labels
2. Feature ablation study
3. Monitor model performance as more features are added
    a. Sudden increase: either a very good feature or leakage!

How to engineer good features

Evaluating a feature

1. Feature importance
2. Feature generalization

Measuring a feature’s importance

How much the model performance deteriorates
if a feature or a set of features containing that feature
is removed from the model?

Measuring a feature’s importance

- XGBoost

get_score(*fmap='', importance_type='weight'*)

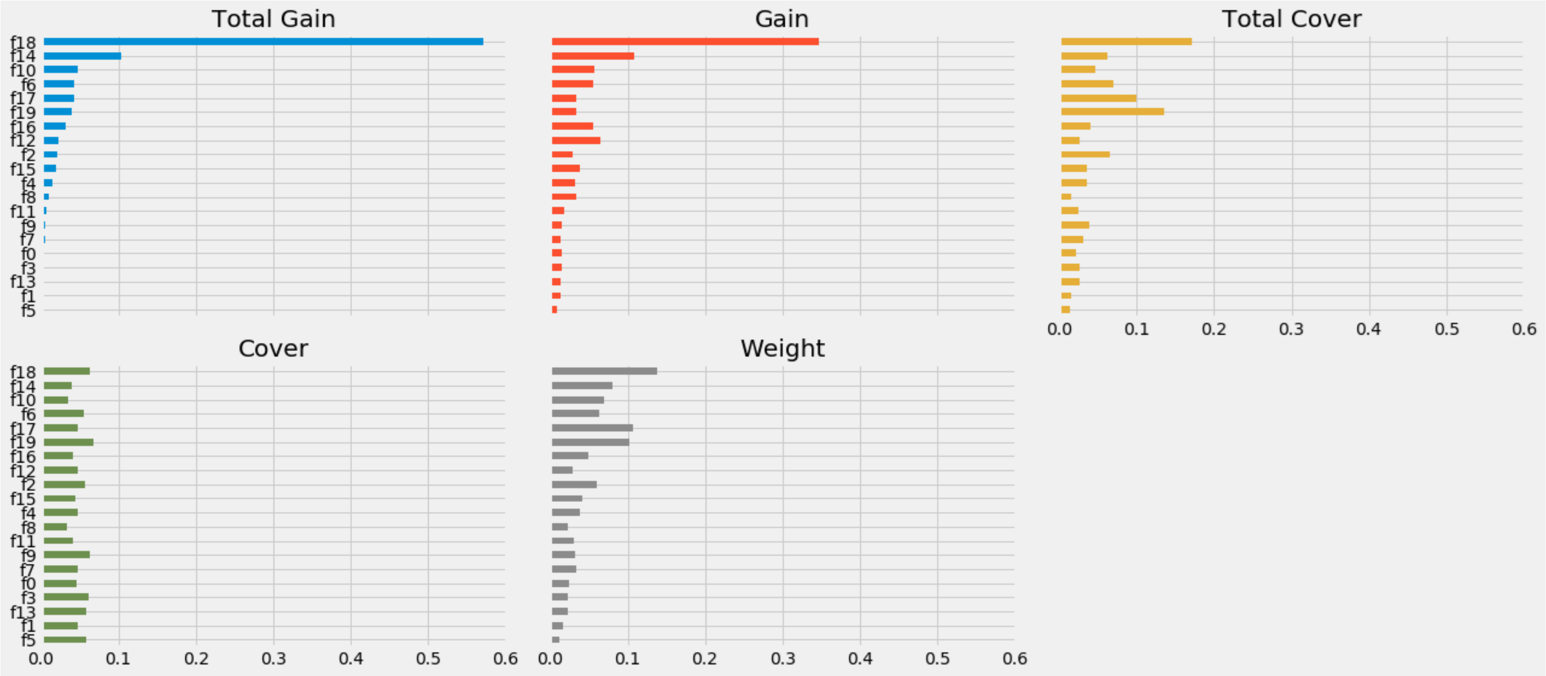
Get feature importance of each feature. For tree model Importance type can be defined as:

- ‘weight’: the number of times a feature is used to split the data across all trees.
- ‘gain’: the average gain across all splits the feature is used in.
- ‘cover’: the average coverage across all splits the feature is used in.
- ‘total_gain’: the total gain across all splits the feature is used in.
- ‘total_cover’: the total coverage across all splits the feature is used in.

https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.Booster.get_score

Measuring a feature’s importance

- XGBoost



The Multiple faces of ‘Feature importance’ in XGBoost (Amjad Abu-Rmileh, 2019)

SHAP: SHapley Additive exPlanations

- Measuring a feature's contribution to a single prediction

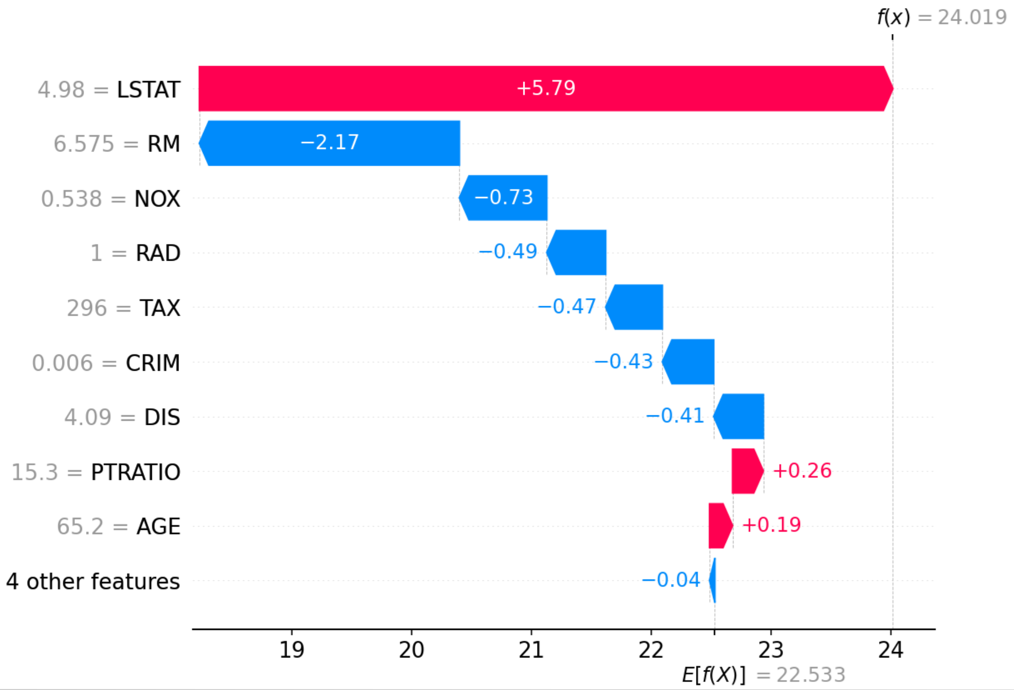


Image by Scott Lundberg from https://github.com/slundberg/shap

SHAP: SHapley Additive exPlanations

- Measuring a feature's contribution to the entire model

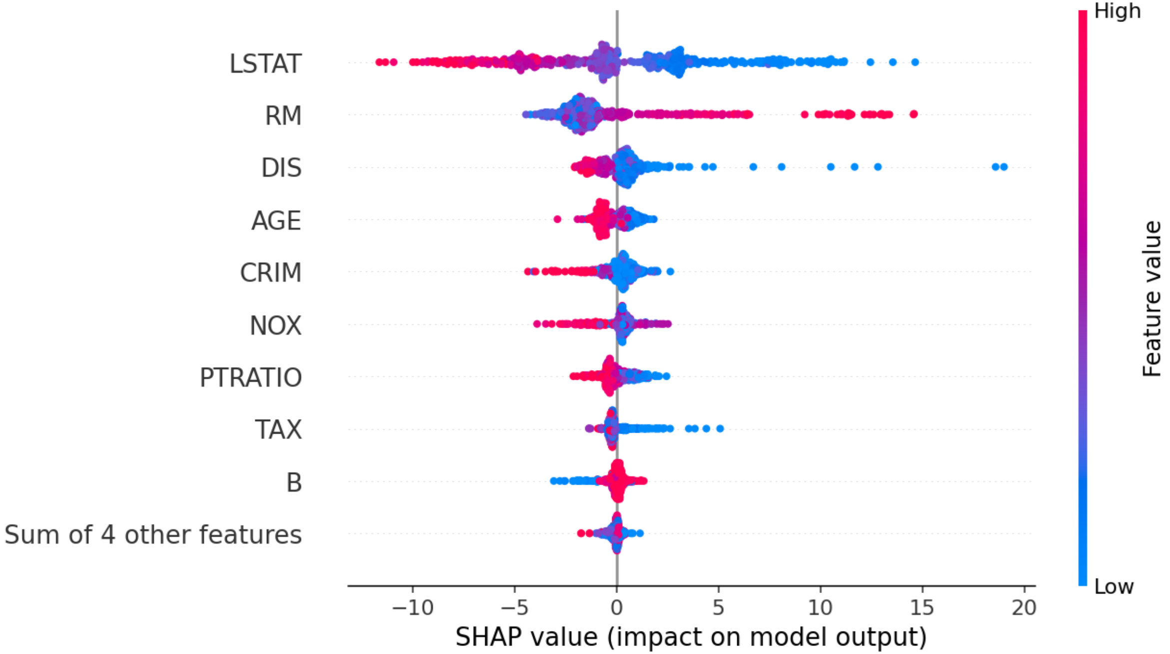
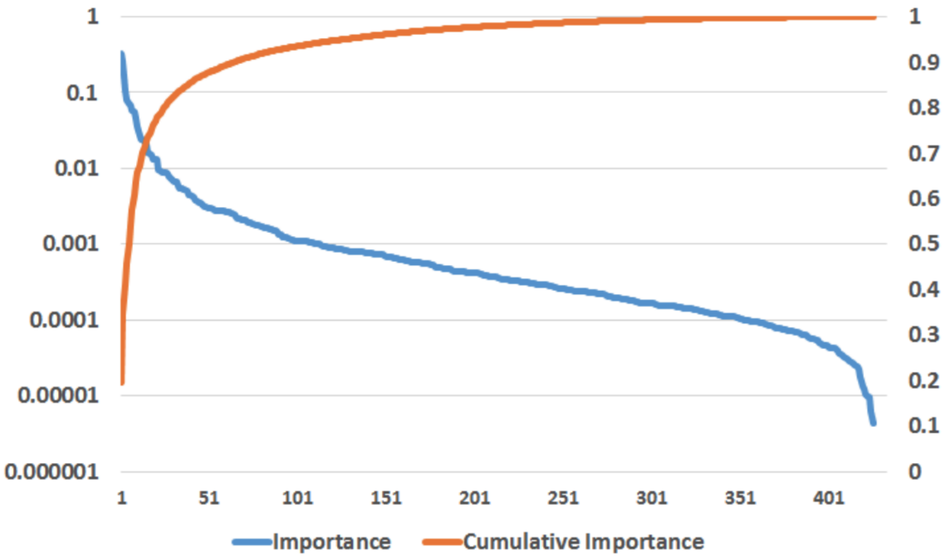


Image by Scott Lundberg from https://github.com/slundberg/shap

Measuring feature importance @ Facebook

- Top 10 features: 50% total feature importance
- Bottom 300 features: <1% total feature importance



[Practical Lessons from Predicting Clicks on Ads at Facebook](#) (He et al., 2014) https://github.com/interpretml/interpret

```
ebm = ExplainableBoostingClassifier()
```

Feature engineering: the more the better?

- Adding more features tends to improve model performance

How can having too many features be bad?

Too many features can be bad …

- Training:
  - Overfitting
  - More features, more opportunity for data leakage

Too many features can be bad ...

- Training:
  - Overfitting
  - More features, more opportunity for data leakage
- Inference
  - Increase inference latency with online prediction
  - Might cause increased memory usage -> more expensive instance required

Too many features can be bad ...

- Training:
  - Overfitting
  - More features, more opportunity for data leakage
- Inference
  - Increase inference latency with online prediction
  - Might cause increased memory usage -> more expensive instance required
- Stale features become technical debts
  - E.g. if zip codes are no longer allowed for predictions, all features that use zip codes will need to be updated

Too many features can be bad ...

- Training:
  - Overfitting
  - More features, more opportunity for data leakage
- Inference
  - Increase inference latency with online prediction
  - Might cause increased memory usage -> more expensive instance required
- Stale features become technical debts

Solution:

- Clean up stale / ineffective features
- Store features in case you want to reuse them
  - Feature management

9 best practices for feature engineering

1. Split data by time instead of doing it randomly.
2. If you oversample your data, do it after splitting.
3. Use statistics/info from the train split, instead of the entire data, for feature engineering: scaling, normalizing, handling missing values, creating n-gram count, item encoding, etc.
4. Understand how your data is generated, collected, and processed. Involve domain experts if necessary.
5. Keep track of data lineage.
6. Understand feature importance to your model.
7. Measure correlation between features and labels.
8. Use features that generalize well.
9. Remove stale features from your models.

Breakout exercise

10 minutes, group of 5

Duolingo is a platform for language learning. When a student is learning a new language, Duolingo wants to recommend increasingly difficult stories to read.

1. What features would you use to measure the difficulty level of a story?
2. Given a story, how would you edit it to make it easier or more difficult?

Model selection

ML algorithm

- Function to be learned
  - E.g. model architecture, number of hidden layers
- Objective function to optimize (minimize)
  - Loss function
- Learning procedure (optimizer)
  - Adam, Momentum

6 tips for evaluating ML algorithms

1. Avoid the state-of-the-art trap

- SOTA's promise
  - Why use an old solution when a newer one exists?
  - It's exciting to work on shiny things
  - Marketing

1. Avoid the state-of-the-art trap

- SOTA's reality
  - SOTA on research data != SOTA on your data
  - Cost
  - Latency
  - Proven industry success
  - Community support

2. Start with the simplest models

- Easier to deploy
  - Deploying early allows validating pipeline
- Easier to debug
- Easier to improve upon

2. Start with the simplest models

- Easier to deploy
  - Deploying early allows validating pipeline
- Easier to debug
- Easier to improve upon
- Simplest models != models with the least effort
  - BERT is easy to start with pretrained model, but not the simplest

3. Avoid human biases in selecting models

- A tale of human biases
  - Papers proposing LSTM variants show that the variants improve upon the vanilla LSTM.
  - Do they?

3. Avoid human biases in selecting models

- A tale of human biases
  - Papers proposing LSTM variants show that the variants improve upon the vanilla LSTM.
  - Do they?

LSTM: A Search Space Odyssey

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber

We conclude that the most commonly used LSTM architecture (vanilla LSTM) performs reasonably well on various datasets. None of the eight investigated modifications significantly improves performance.

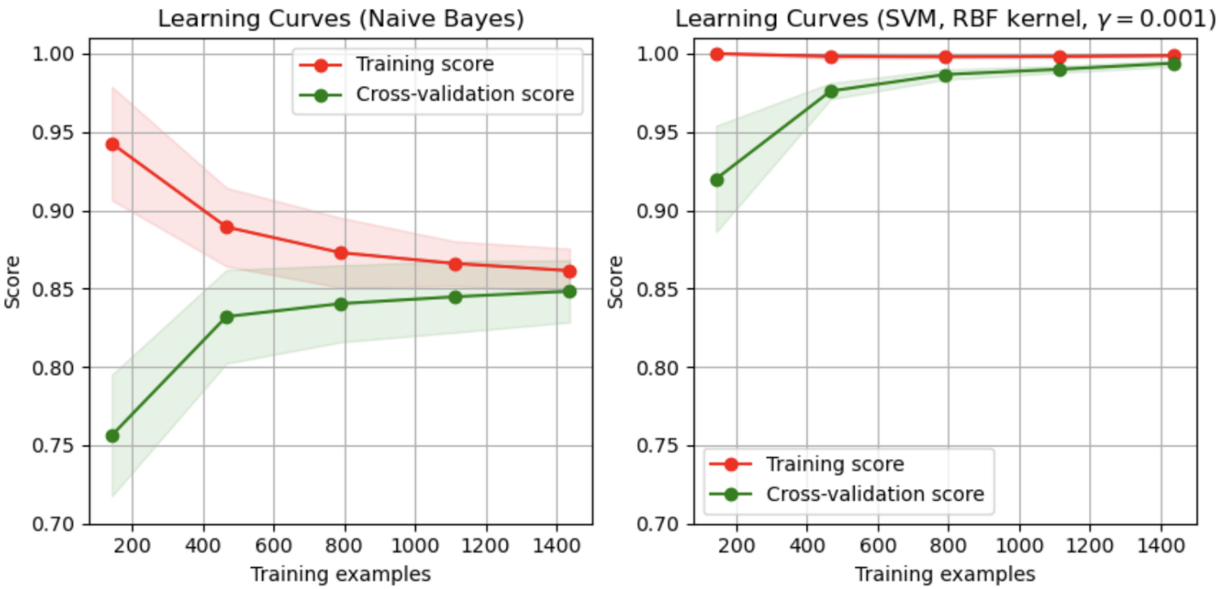
3. Avoid human biases in selecting models

- It’s important to evaluate models under comparable conditions
  - It’s tempting to run more experiments for X because you’re more excited about X
- Near-impossible to make blanketed claims that X is *always* better than Y

Better now vs. better later

- Best model now != best model in 2 months
  - Improvement potential with more data
  - Ease of update

Learning curve



Good for estimating if performance can improve with more data

https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

5. Evaluate trade-offs

- False positives vs. false negatives
- Accuracy vs. compute/latency
- Accuracy vs. interpretability

6. Understand your model's assumption

- IID
  - **Neural networks** assume that examples are **independent and identically distributed**
- Smoothness
  - **Supervised algorithms** assume that there's a set of functions that can transform inputs into outputs such that **similar inputs are transformed into similar outputs**
- Tractability
  - Let X be the input and Z be the latent representation of X. **Generative models** assume that it's tractable to compute P(Z|X).
- Boundaries
  - **Linear classifiers** assume that **decision boundaries are linear**.
- Conditional independence
  - **Naive Bayes classifiers** assume that the **attribute values are independent of each other given the class.**

6 tips for evaluating ML algorithms

1. Avoid the state-of-the-art trap
2. Start with the simplest models
3. Avoid human biases in selecting models
4. Evaluate good performance now vs. good performance later
5. Evaluate trade-offs
6. Understand your model’s assumptions

Ensembles

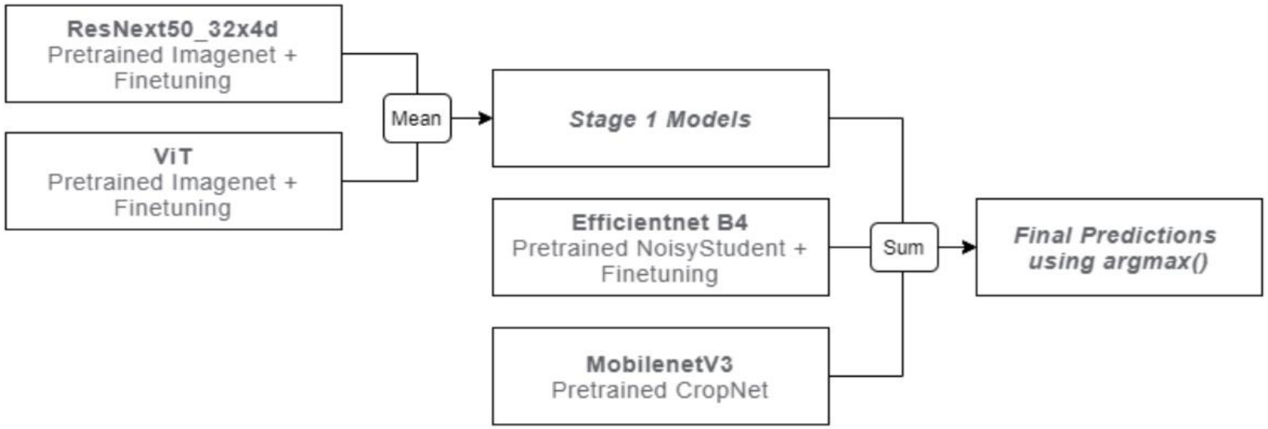
Ensemble

- Creating a strong model from an ensemble of weak models

Base learners

Ensembles: extremely common in leaderboard style projects

- 20/22 winning solutions on Kaggle in Jan - Aug 2021 use ensembles
- One solution uses 33 models!



https://farid.one/kaggle-solutions

Why does ensembling work?

- Task: email classification (SPAM / NOT SPAM)
- 3 uncorrelated models, each with accuracy of 70%
- Ensemble: majority vote of these 3 models
  - Ensemble is correct is at least 2 models are correct

Why does ensembling work?

- 3 models, each with 70% accuracy
- Ensemble is correct if at least 2 models are correct
- Probability at least 2 models are correct: 34.3% + 44.1% = 78.4%

Outputs of 3 models	Probability	Ensemble's output
All 3 are correct	0.7 * 0.7 * 0.7 = 0.343	Correct
Only 2 are correct	(0.7 * 0.7 * 0.3) * 3 = 0.441	Correct
Only 1 is correct	(0.3 * 0.3 * 0.7) * 3 = 0.189	Wrong
None is correct	0.3 * 0.3 * 0.3 = 0.027	Wrong

Why does ensembling work?

- 3 models, each with 70% accuracy
- Ensemble is correct if at least 2 models are correct
- Probability at least 2 models are correct: 34.3% + 44.1% = 78.4%

- The less correlation among base learners, the better
- Common for base learners to have different architectures

Ensemble

- Bagging
- Boosting
- Stacking

Bagging

- Sample with replacement to create different datasets
- Train a classifier with each dataset
- Aggregate predictions from classifiers
    - e.g. average, majority vote

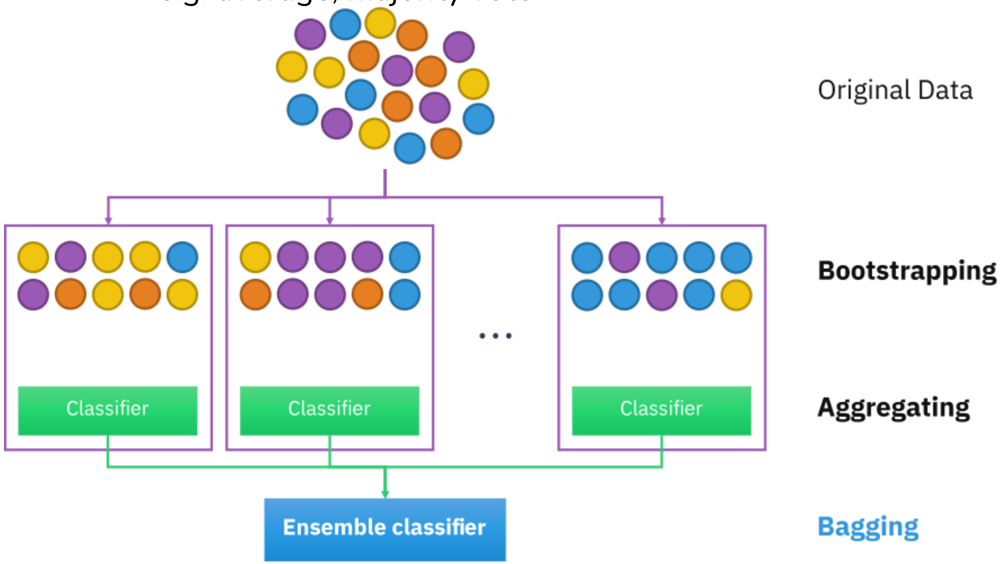
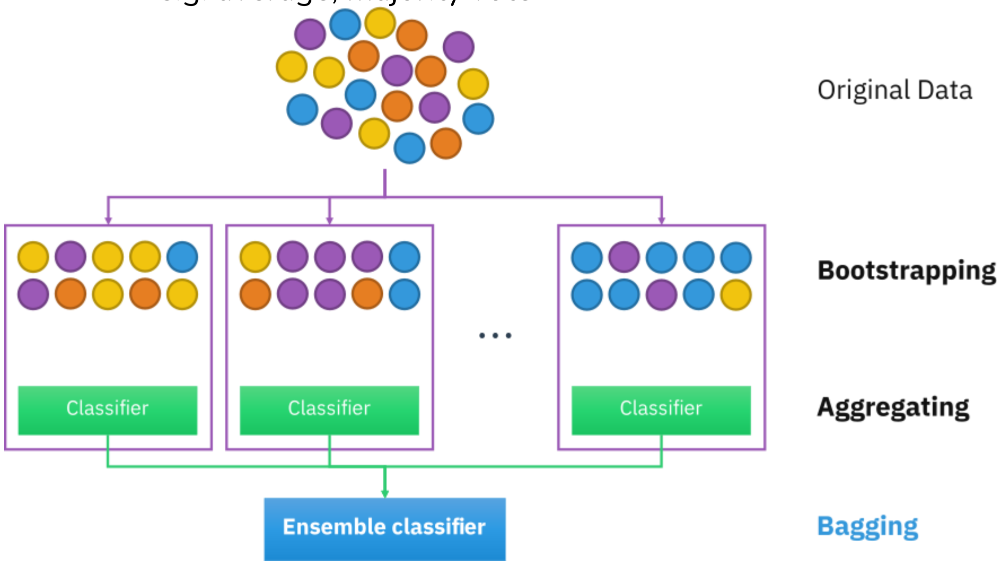


Illustration by [Sirakorn]

Bagging

- Sample with replacement to create different datasets
- Train a classifier with each dataset
- Aggregate predictions from classifiers
    - e.g. average, majority vote



- Generally improves unstable methods e.g. neural networks, trees
- Can degrade stable methods e.g. kNN

Bagging Predictors (Leo Breiman, 1996)

Illustration by Sirakorn

Boosting

1. Train a weak classifier
2. Give samples misclassified by weak classifier higher weight
3. Repeat (1) on this reweighted data as many iterations as needed
4. Final strong classifier: weighted combination of existing classifiers
    a. classifiers with smaller training errors have higher weights

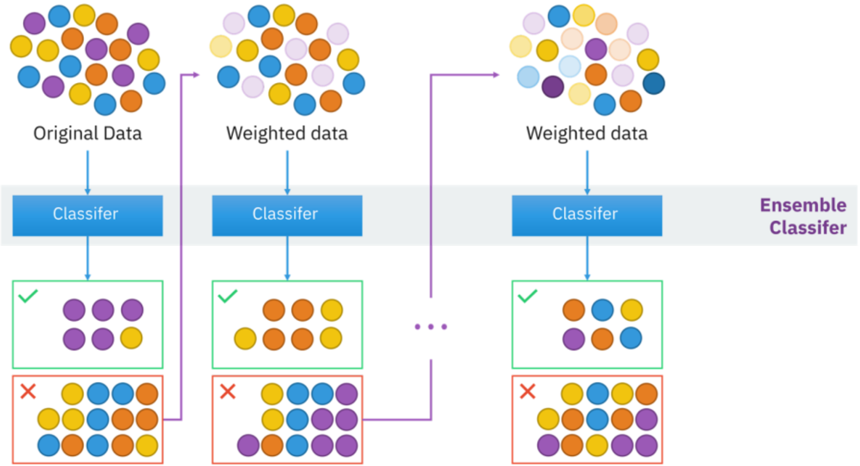


Illustration by [Sirakorn]

Boosting

1. Train a weak classifier
2. Give samples misclassified by weak classifier higher weight
3. Repeat (1) on this reweighted data as many iterations as needed
4. Final strong classifier: weighted combination of existing classifiers
    a. classifiers with smaller training errors have higher weights

Extremely popular:

- XGBoost
- LightGBM

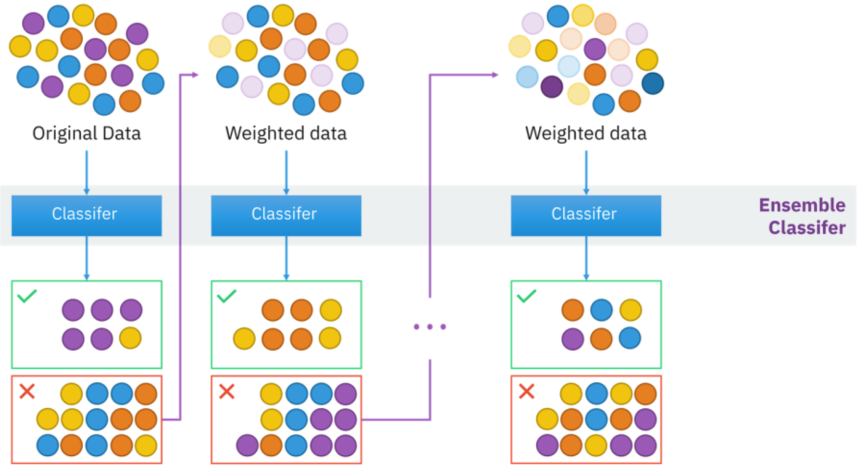
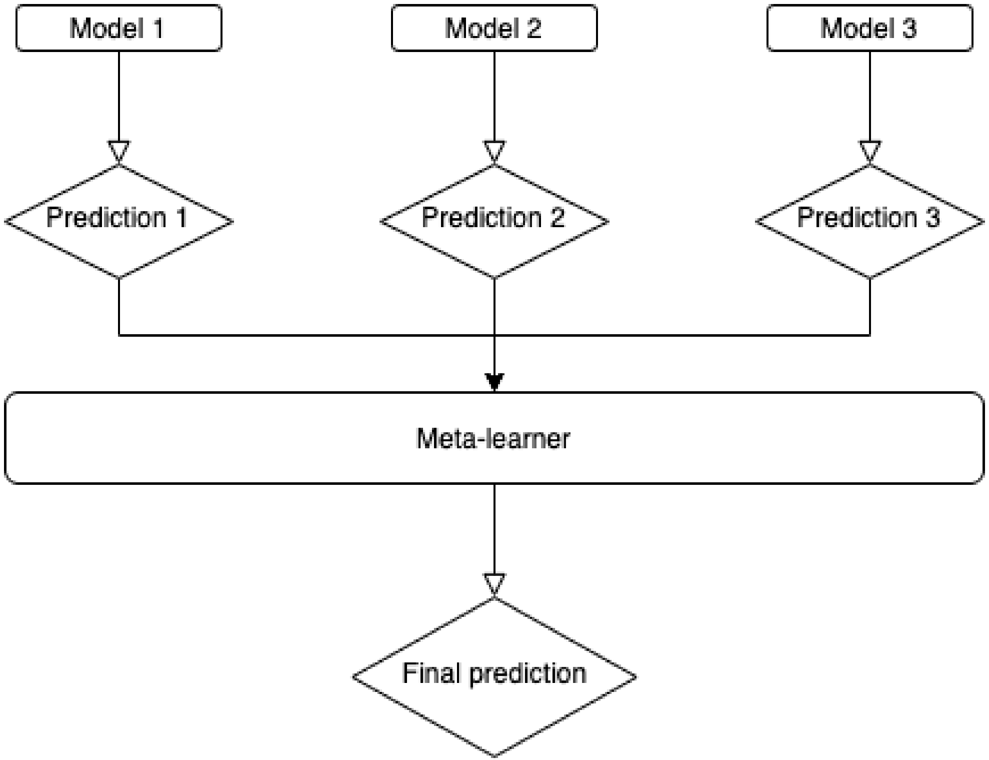


Illustration by Sirakorn

Stacking

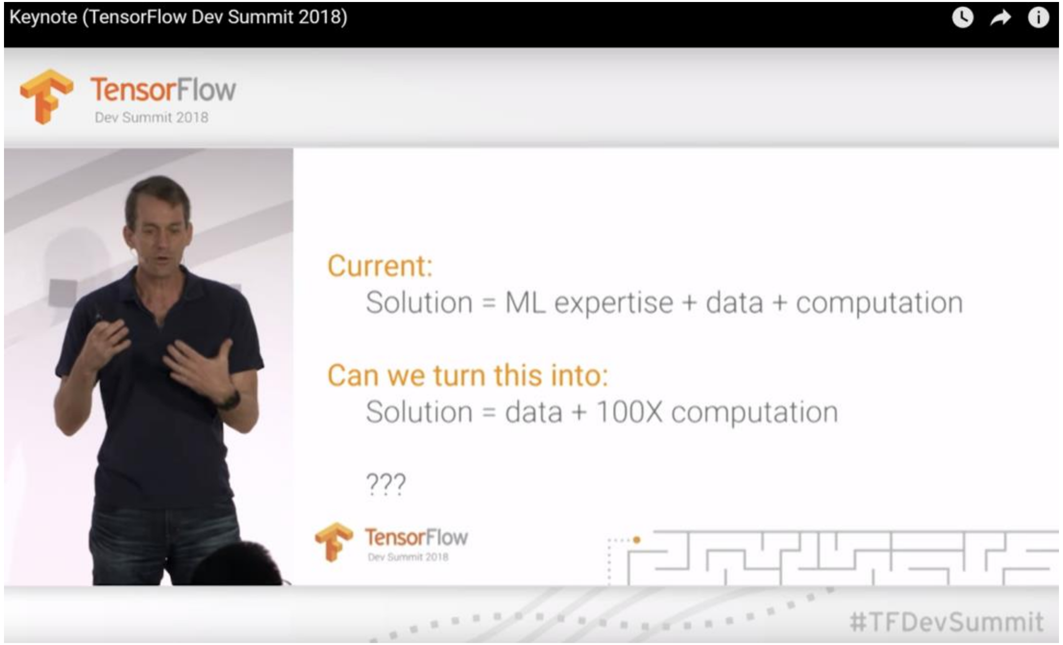


Majority vote, logistic regression, simple NN

AutoML

AutoML

- A good ML researcher is someone who will automate themselves out of job
- Google: what if we replace ML experts with 100x compute?



AutoML

- Soft AutoML:
  - hyperparameter tuning
- Hard AutoML
  - neural architecture search
  - learned optimizer

More computationally expensive

Soft AutoML: Hyperparameter tuning

- Weaker models with well-tuned hyperparameters can outperform fancier models
  - On the State of the Art of Evaluation in Neural Language Models (Melis et al. 2018)

Soft AutoML: Hyperparameter tuning

- Many hyperparameters to tune

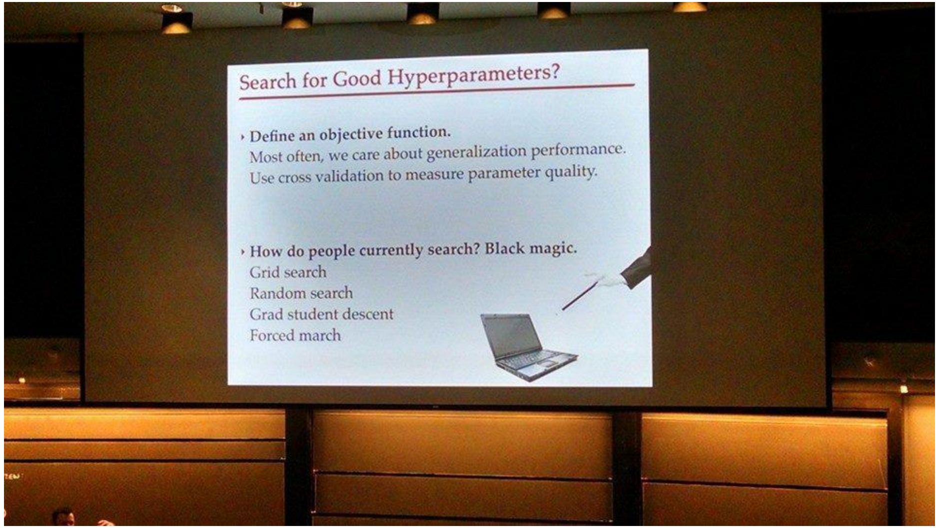
```
model_type = "bert"

def __init__(
    self,
    vocab_size=30522,
    hidden_size=768,
    num_hidden_layers=12,
    num_attention_heads=12,
    intermediate_size=3072,
    hidden_act="gelu",
    hidden_dropout_prob=0.1,
    attention_probs_dropout_prob=0.1,
    max_position_embeddings=512,
    type_vocab_size=2,
    initializer_range=0.02,
    layer_norm_eps=1e-12,
    pad_token_id=0,
    position_embedding_type="absolute",
    use_cache=True,
    classifier_dropout=None,
    **kwargs
):
```

https://github.com/huggingface/transformers/blob/master/src/transformers/models/bert/configuration_bert.py

Soft AutoML: Hyperparameter tuning

- Graduate Student Descent (GSD)
  - A graduate student fiddles around with the hyperparameters until the model works



https://twitter.com/GuyZys/status/592847074170896384/photo/1

Soft AutoML: Hyperparameter tuning

- Hyperparam tuning has become a standard part of ML workflows
- Built-in with frameworks
    - TensorFlow: Keras Turner
    - scikit-learn: auto-sklearn
    - Ray Tune
- Popular algos:
    - Random search
    - Grid search
    - Bayesian optimization

NAS: Neural architecture search

- **Search space**
  - Set of operations
    - e.g. convolution, fully-connected, pooling
  - How operations can be connected

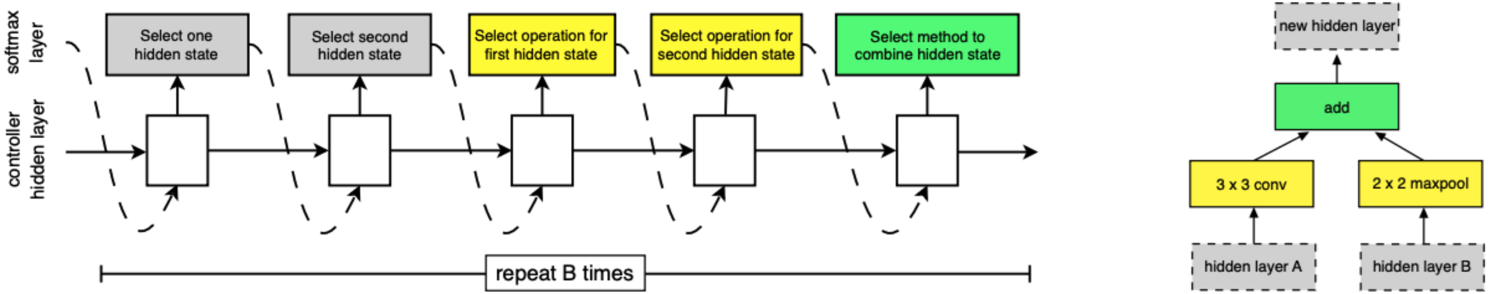


Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

Learning Transferable Architectures for Scalable Image Recognition (Zoph et al., 2017)

NAS: Neural architecture search

- Search space
- **Performance estimation strategy**
  - How to evaluate **many** candidate architectures?
  - Ideal: should be done without having to re-construct or re-train them from scratch.

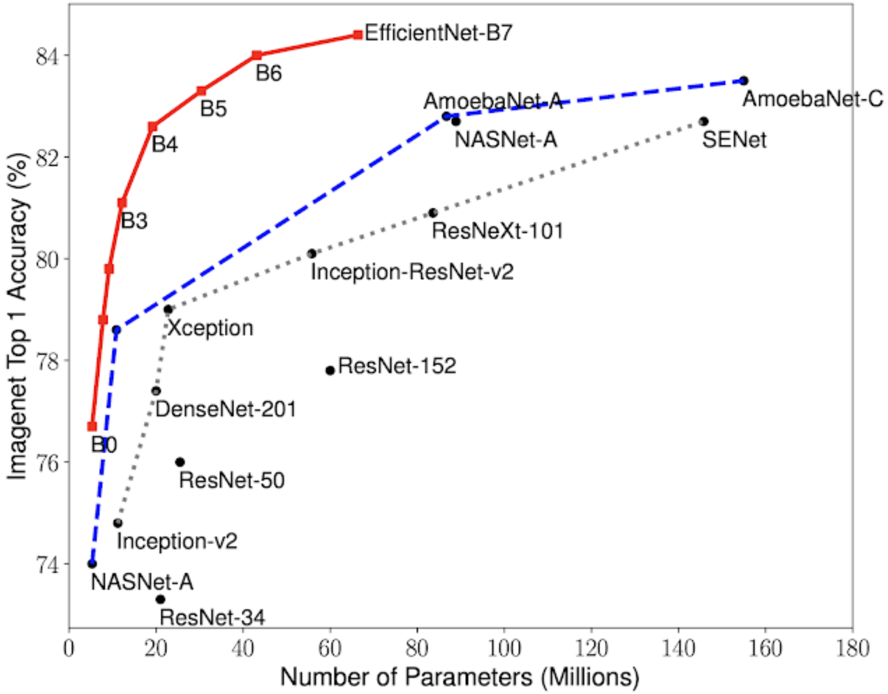
NAS: Neural architecture search

- **Search space**
- **Performance estimation strategy**
- **Search strategy**
  - Random
  - Reinforcement learning
    - reward the choices that improve performance estimation
  - Evolution
    - mutate an architecture
    - choose the best-performing offsprings
    - so on

NAS: Neural architecture search

- Search space
- Performance estimation strategy
- Search strategy

Very successful



EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan et Le., 2019)

Learning: architecture + learning algorithm

- Learning algorithm:
  - A set of functions that specifies how to update the weights.
  - Also called **optimizers**
    - Adam, Momentum, SGD

Learned optimizer

Deep learning

engineering features → learning features

SIFT (Lowe et. al. 1999)
HOG (Dalal et. al. 2005)

LeNet (LeCun et. al. 1998)
AlexNet (Krizhevsky et. al. 2012)

Meta learning

engineering to learn → learning to learn

SGD (Robbins et. al. 1951, Bottou 2010)
Autoencoders (Hinton et. al. 2006)

Learning To Learn (Hochreiter et. al. 2001)
Learned Optimizers (Andrychowicz et. al. 2016, Li et. al. 2016, Wichrowska et. al. 2017, Metz et. al. 2018, 2019)

Slide courtesy of Luke Metz

Learned optimizer

- Learn how to learn on a set of tasks
- Generalize to new tasks

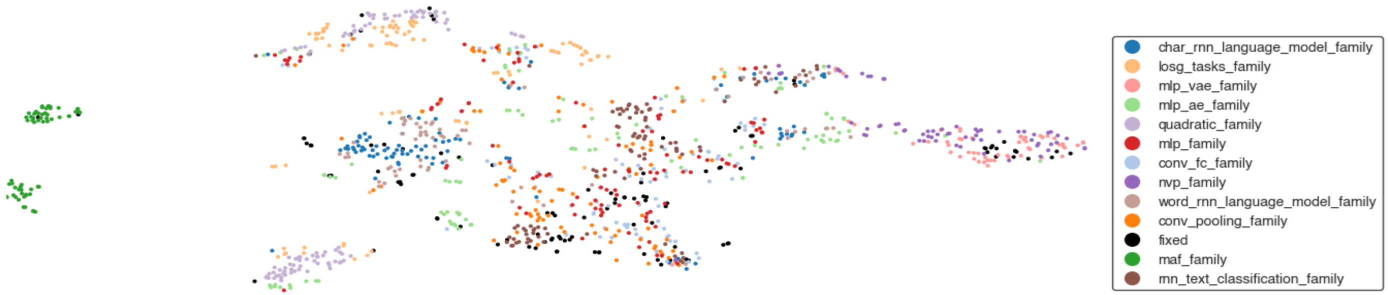
Using a thousand optimization tasks to learn hyperparameter search strategies

Luke Metz [1] **Niru Maheswaranathan** [1] **Ruoxi Sun** [1] **C. Daniel Freeman** [1] **Ben Poole** [1] **Jascha Sohl-Dickstein** [1]

Slide courtesy of Luke Metz

Learned optimizer

- Learn how to learn on a set of tasks
- Generalize to new tasks
- The learned optimizer can then be used to train a better version of itself!



Using a thousand optimization tasks to learn hyperparameter search strategies

Luke Metz [1] Niru Maheswaranathan [1] Ruoxi Sun [1] C. Daniel Freeman [1] Ben Poole [1] Jascha Sohl-Dickstein [1]

Slide courtesy of Luke Metz

Machine Learning Systems Design

Next class: Model Offline Evaluation

Universidad
Popular del Cesar