Applied Machine Learning for Business Analytics

Lecture 3: Modeling

Lecturer: Zhao Rui

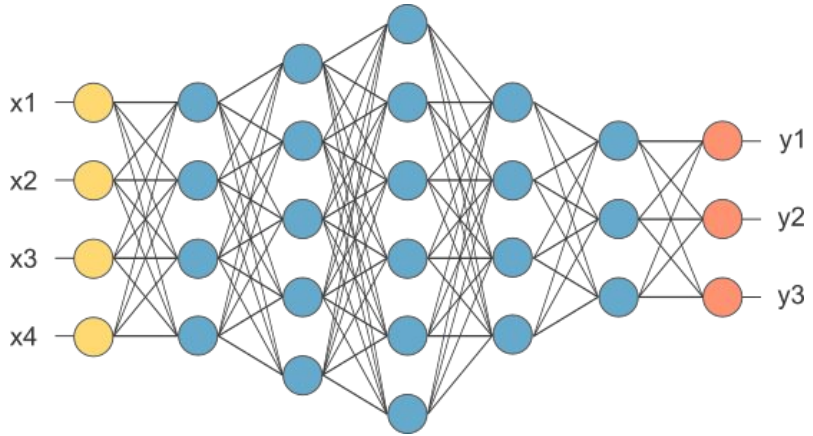
Agenda

1. Understanding Concepts behind ML Models
2. Cross-Validation
3. Hyper-parameter Selection
4. Ensembles
5. Do not sleep on traditional machine learning

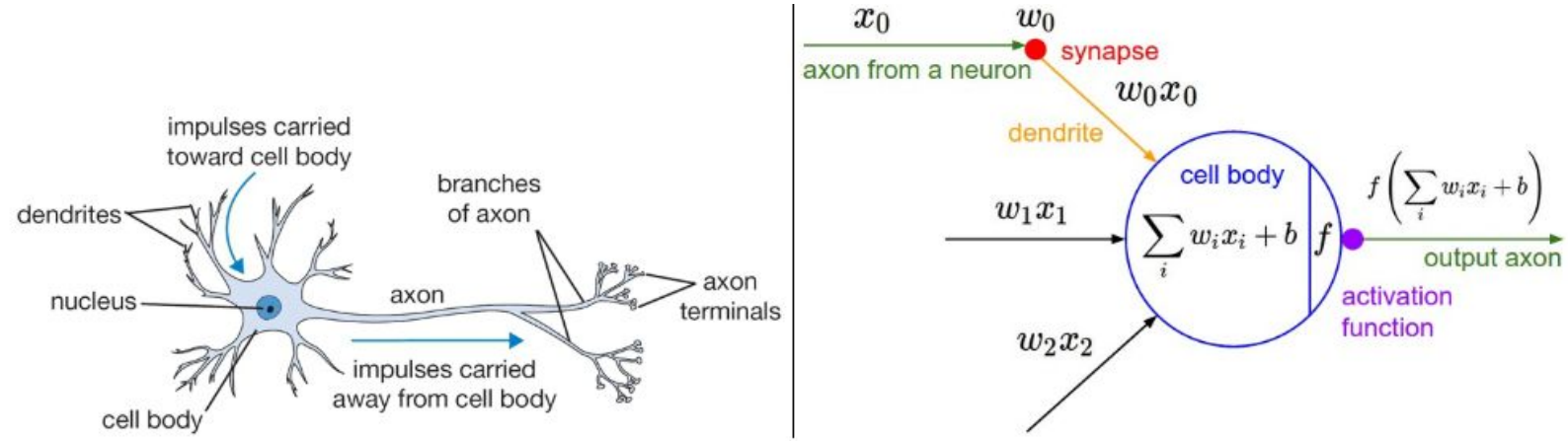
1. Understanding concepts

Take Neural network as an example

- From Wiki:
  - NN is based on a collection of connected units of nodes called artificial neurons which loosely model the neurons in a biological brain.
- From another way:
  - NN is running several ‘logistic regression’ at the same time (expanding at width and depth dimensions).



Neural computation

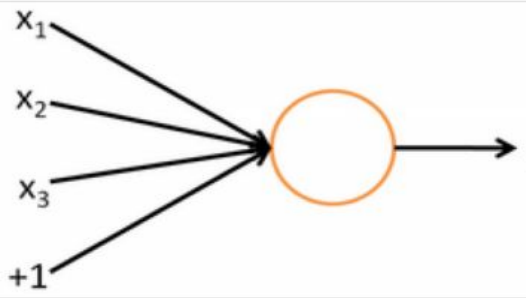


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

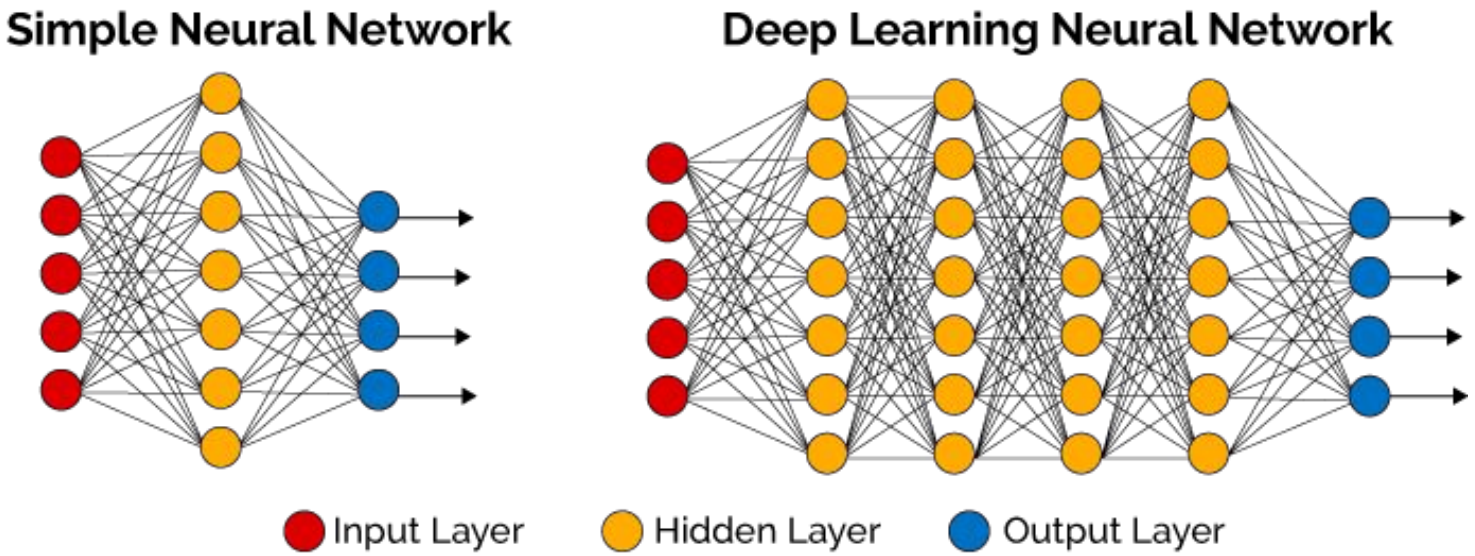
The fact that a neuron is essentially a logistic regression unit:

1 performs a dot product with the input and its weights

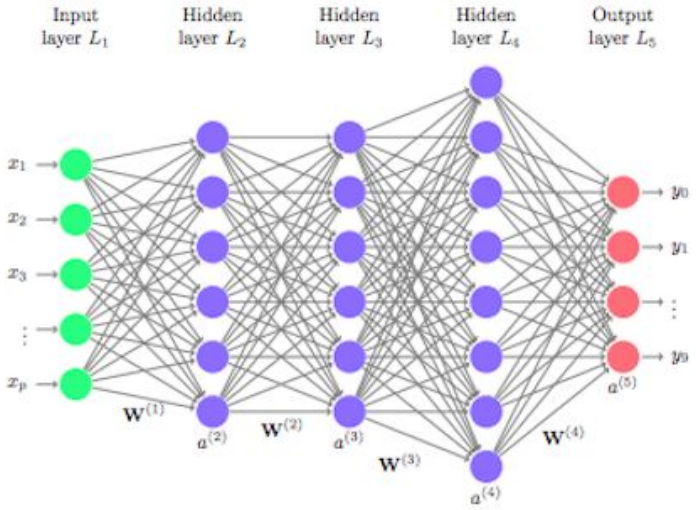
2 adds the bias and apply the non-linearity



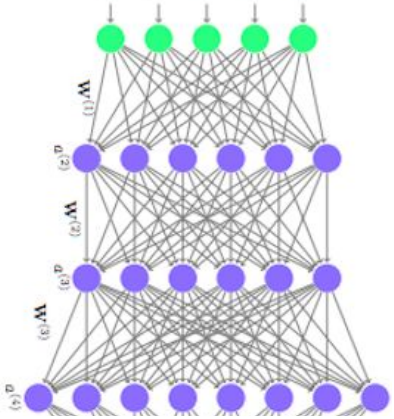
Shallow vs Deep



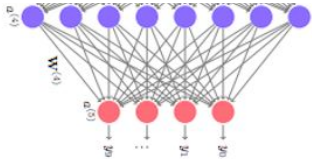
Hidden representation in deep learning



Low-dim, Original Space



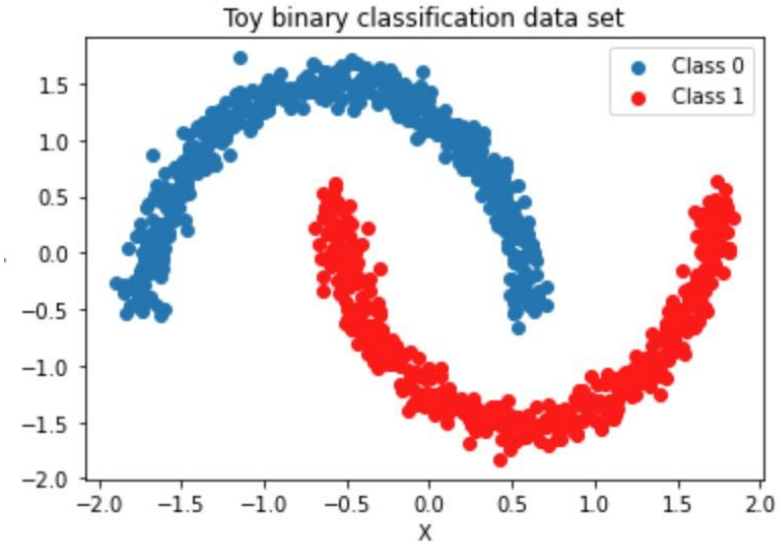
High-dim, **Linearly Separated** Space



Softmax Classifier
(Linear Model)

We want to project the data into the new feature/vector space that data is linearly separated

Moons Dataset




```
# fit a logistic regression model to classify this data set as a benchmark
simple_model = LogisticRegression()
simple_model.fit(X_train, Y_train)
print('Train accuracy:', simple_model.score(X_train, Y_train))
print('Test accuracy:', simple_model.score(X_test, Y_test))
```

```
Train accuracy: 0.89
Test accuracy: 0.88
```

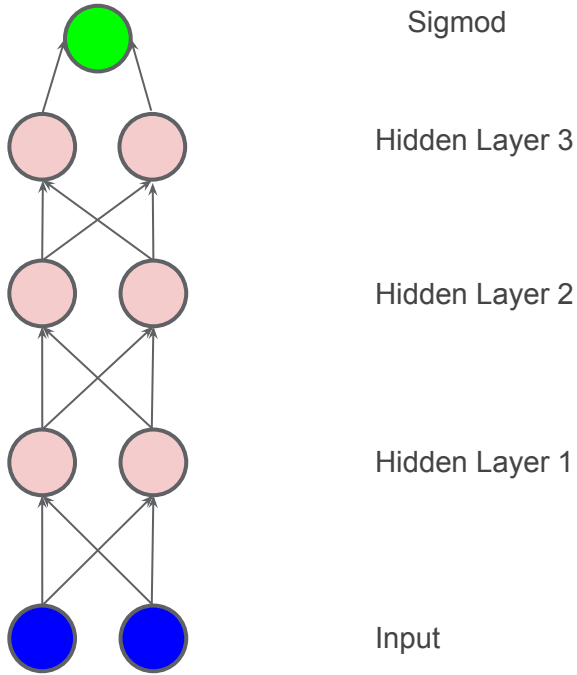
Fully-Connected Neural Network

```
# fix a width that is suited for visualizing the output of hidden layers
H = 2
input_dim = X.shape[1]

# create sequential multi-layer perceptron
model = Sequential()

# Then, use add() to insert layers into the container
model.add(Input(shape=(input_dim,)))
model.add(Dense(H,activation='tanh'))
model.add(Dense(H, activation='tanh'))
model.add(Dense(H, activation='tanh'))
#binary classification, one output
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              metrics=['acc'])
```



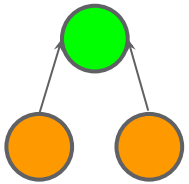
Fully-Connected Neural Network

```
# evaluate the training and testing performance of your model
# note: you should extract check both the loss function and your evaluation metric
score = model.evaluate(X_train, Y_train, verbose=0)
print('Train loss:', score[0])
print('Train accuracy:', score[1])
```

```
Train loss: 0.0007340409210883081
Train accuracy: 1.0
```

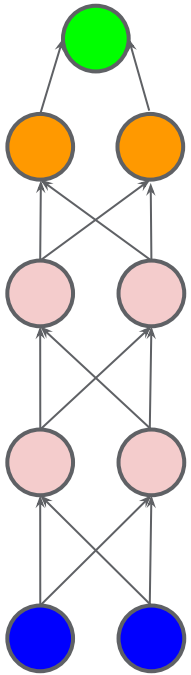
```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.0008793871384114027
Test accuracy: 1.0
```



1. In forward computation, the output of hidden layer 3 is feed into “logistic regression” to predict labels.
2. Since the train and test accuracy are both 1, it means the hidden layer 3’ output are linearly separated.

Let us visualize those outputs!



Sigmod

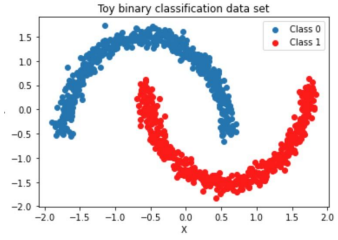
Hidden Layer 3

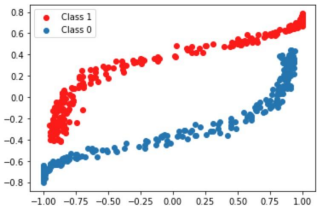
Hidden Layer 2

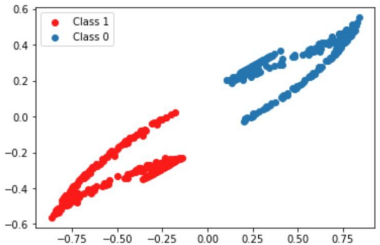
Hidden Layer 1

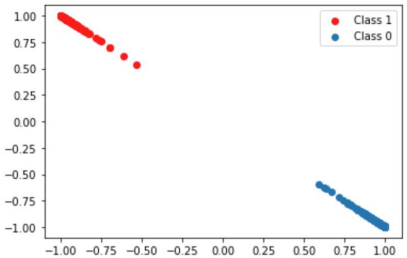
Input

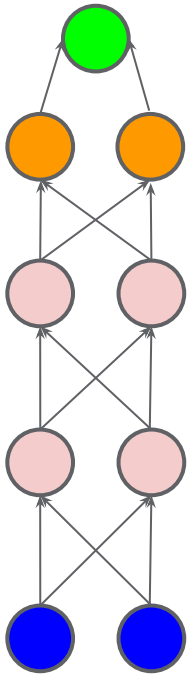
Fully-Connected Neural Network











Sigmod

Hidden Layer 3

Hidden Layer 2

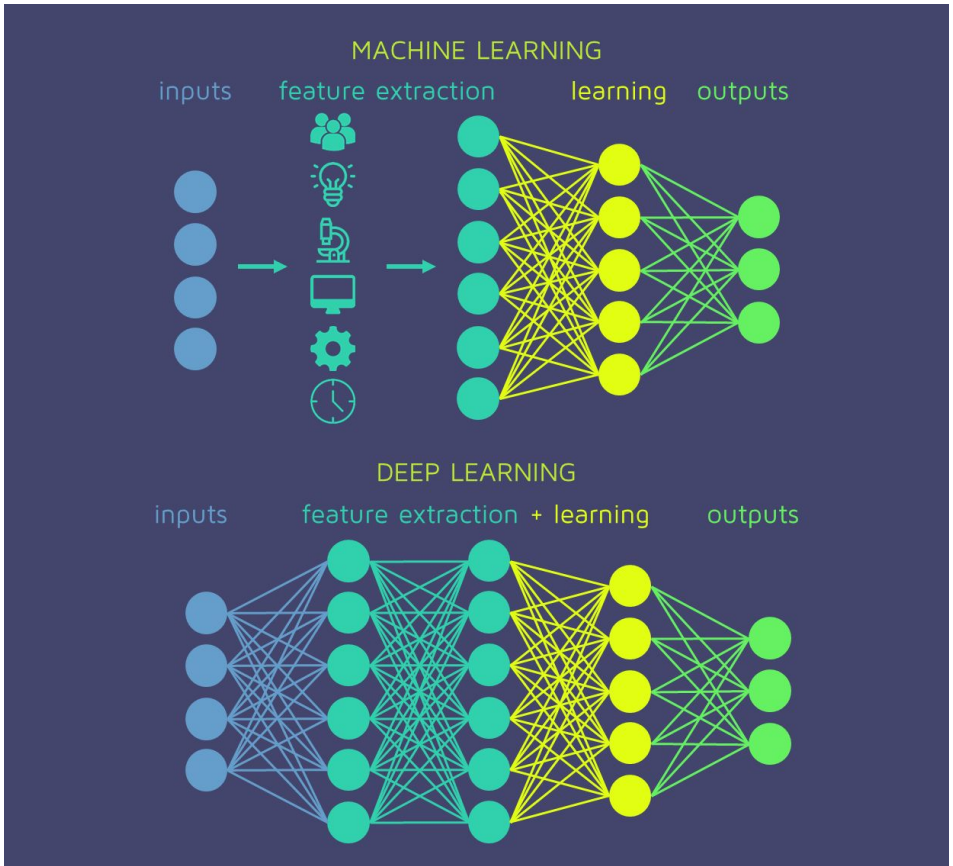
Hidden Layer 1

Input

Representation Learning

https://github.com/rz0718/BT5153_2023/blob/main/codes/lab_lecture03/Representation_Learning.ipynb

End-to-end learning

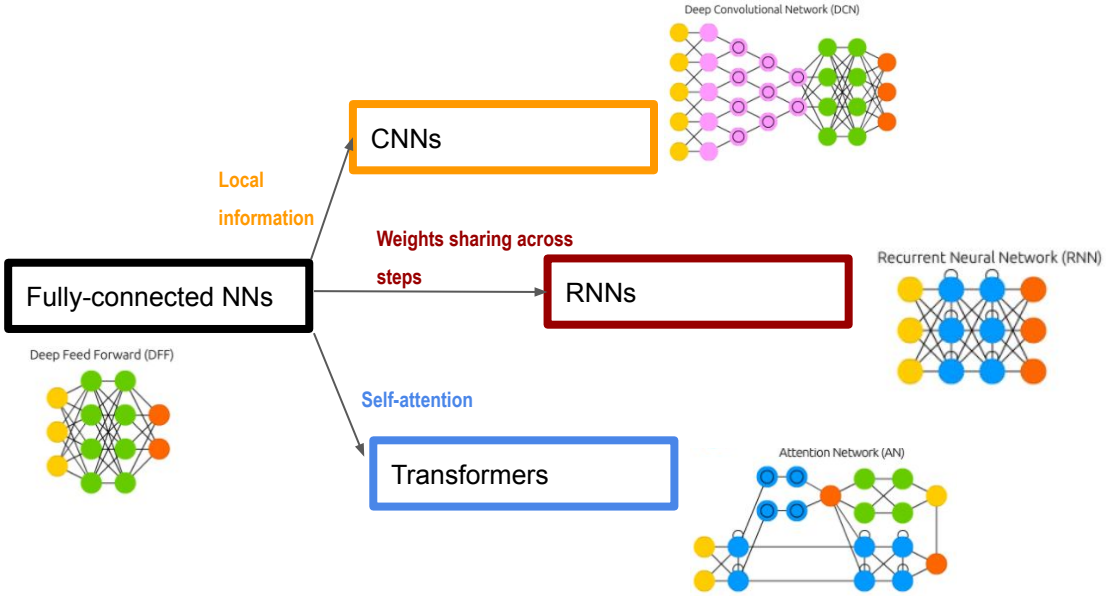


From Aporras

Representation Learning in Neural Networks

- Outputs of each hidden layer of an neural network is a non-linear transformation of the input data into a feature space. Each hidden layer should transform the input so that it is more linearly separable
- we are more interested in learning the latent representation of the data rather than perfecting our performance in a single task (such as classification).
    - We do not need to preprocess the data to add non-linear features. The neural network will learn the most suitable non-linear transformations to the input (to achieve the best classification)

Deep learning structures



https://www.asimovinstitute.org/author/fjodorvanveen/

Understand your model's assumption

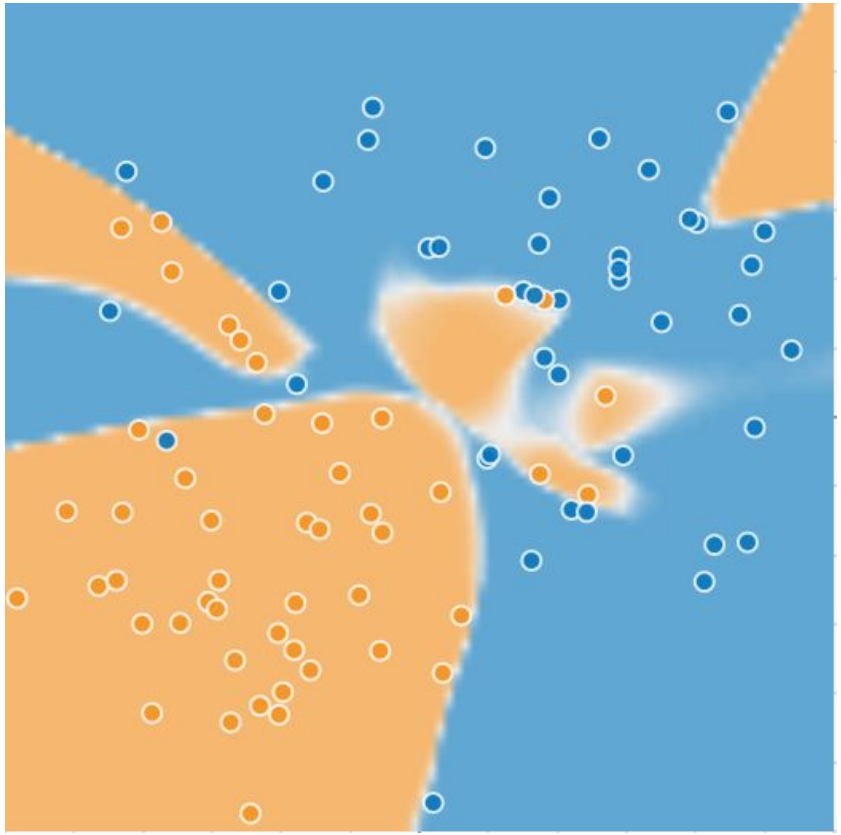
- Independent and Identically Distributed-IID
  - **Neural networks** assume that examples are **independent and identically distributed**
- Smoothness
  - **Supervised algorithms** assume that there's a set of functions that can transform inputs into outputs such that **similar inputs are transformed 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.**

2. Cross-Validation

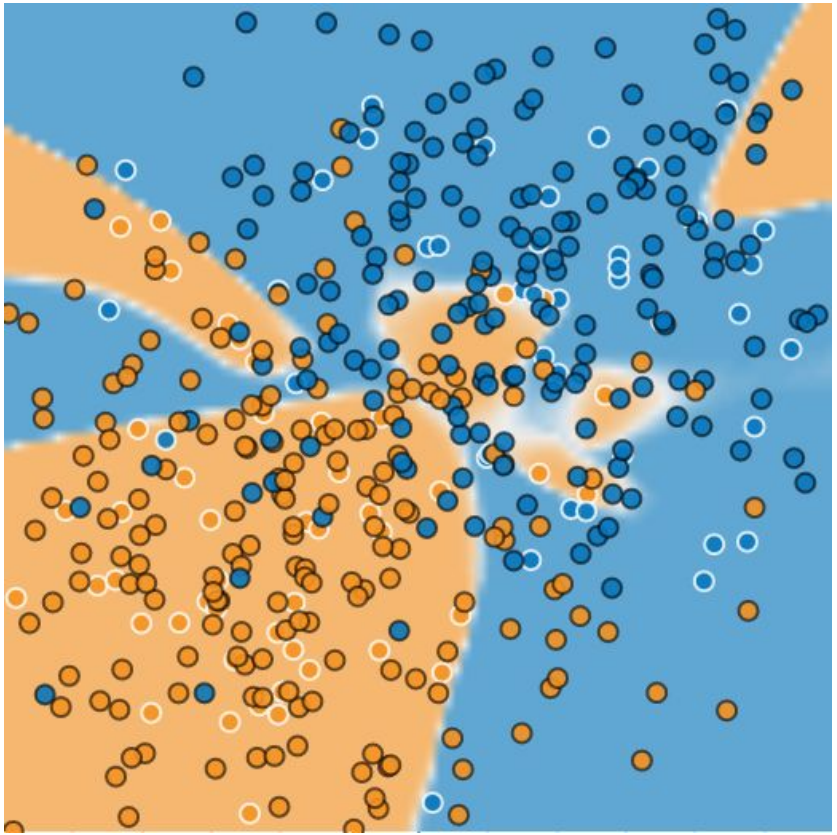
Which measure should we look for model evaluation?

Let’s try to train a model for this problem

How about this model?



More data



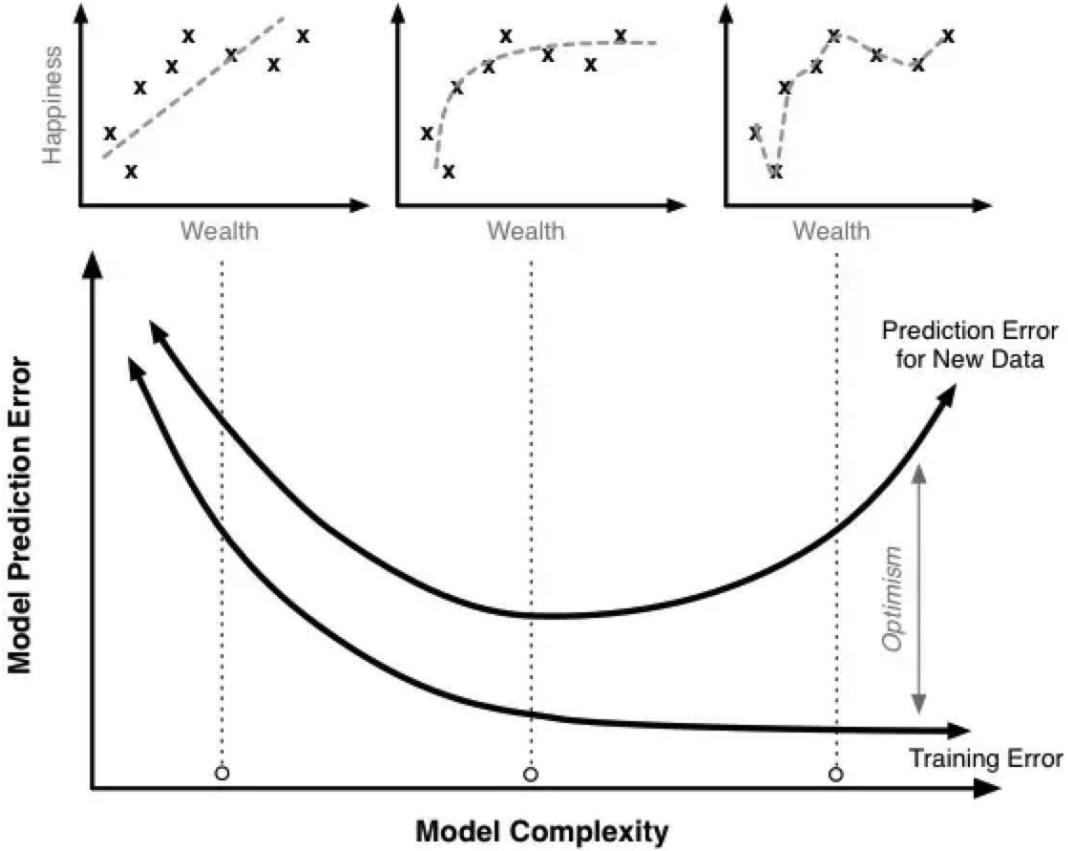
Which measure should we look for model evaluation?

Training performance is not suitable

Generalization

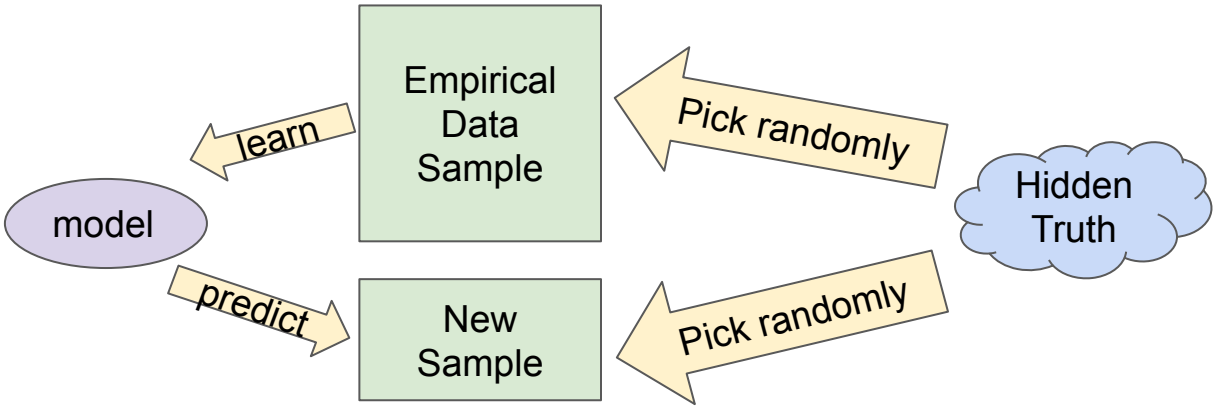
- In ML, a model is used to fit the data
- Once trained, the model is applied upon new data
- Generalization is the prediction capability of the model on live/new data

Model Complexity



The Big Picture

- Goal: predict well on new data drawn from (hidden) true distribution.
- Problem: we don't see the truth.
  - We only get to sample from it.
- If model *h* fits our current sample well, how can we trust it will predict well on other new samples?



Is the model overfitting?

- Intuition: Occam’s Razor principle
  - The less complex a model is, the more likely that a good empirical result is not just due to the peculiarities of our samples.

- Theoretically:
  - Interesting field: generalization theory
  - Based on ideas of measuring model simplicity / complexity

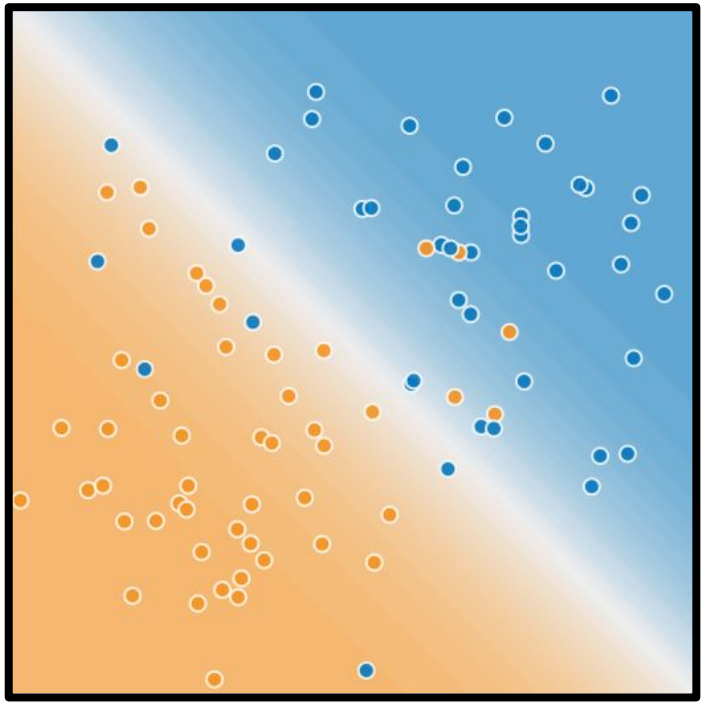
Is the model overfitting?

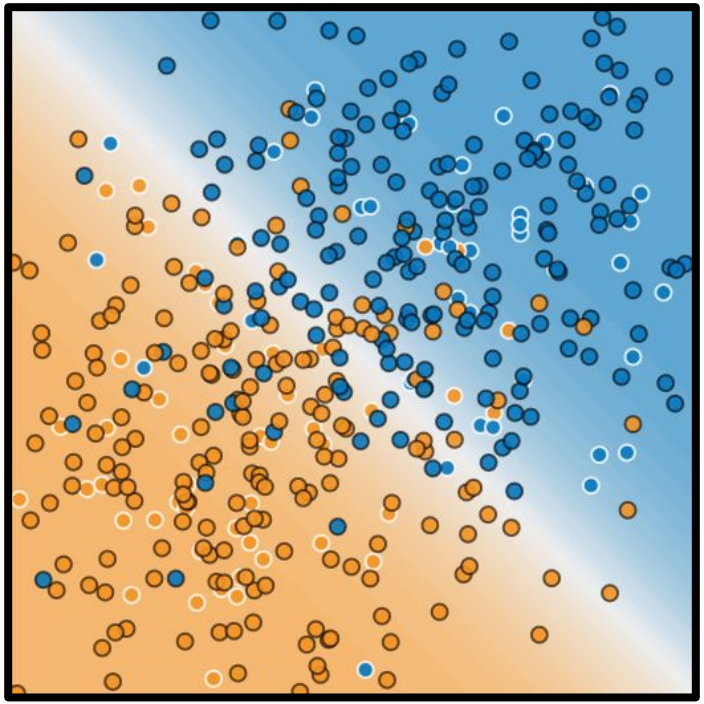
- Empirically:
  - Key point: will our model be good on new samples?
  - Evaluate: get new samples of data (test set)
  - If test set is large enough and we do not cheat by using test set over and over, the good performance on test set can be a useful indicator of model's generalization capability

Training/Test Splitting

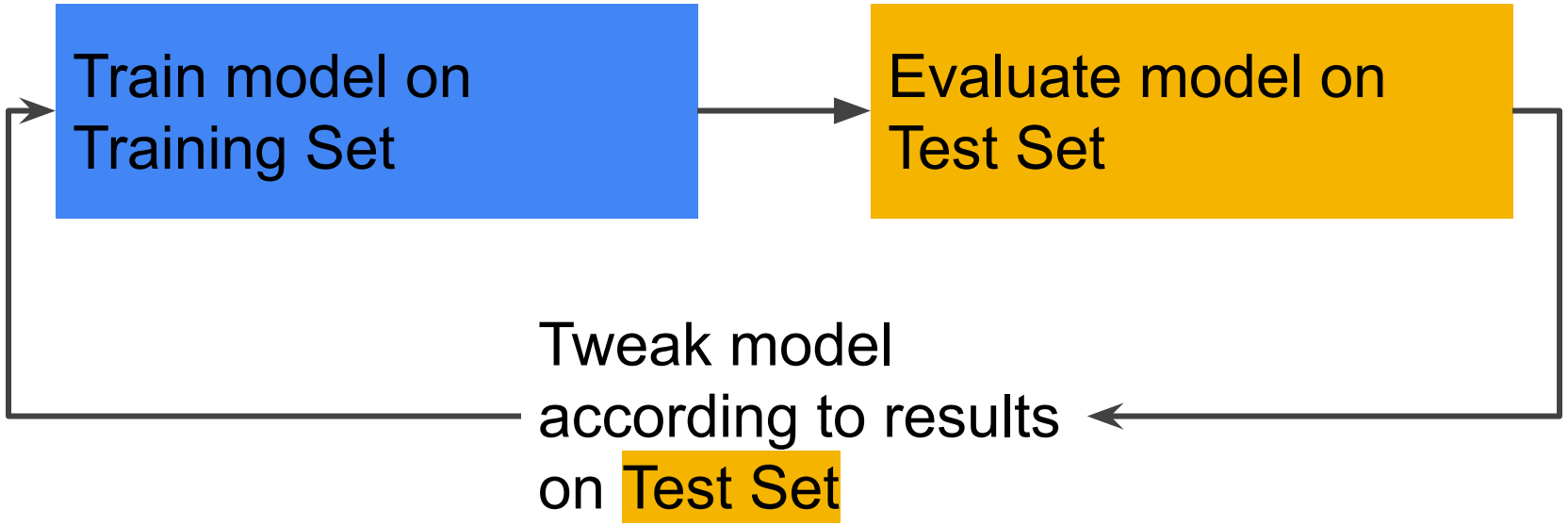
- If models do much better on the training set than the testing set, then models are likely overfitting.
- How do we divide?
  - Randomization for splitting
  - Larger training data size -> better model
  - Larger testing data size -> more confident in model's evaluation
  - One practical rule: 10-15% left for testing, the rest for training

Training vs Test





How about this workflow?



Pick model that does best on Test Set.

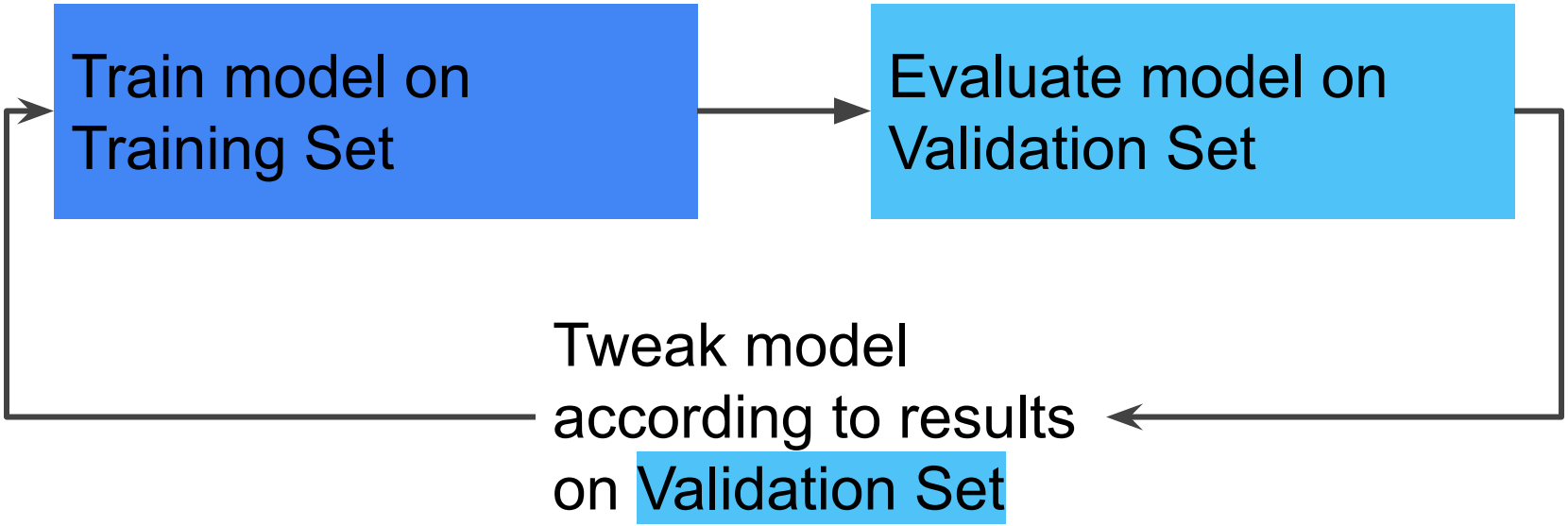
Partition Data Sets

Training Set

Validation Set

Test Set

Better Workflow: Use a validation set



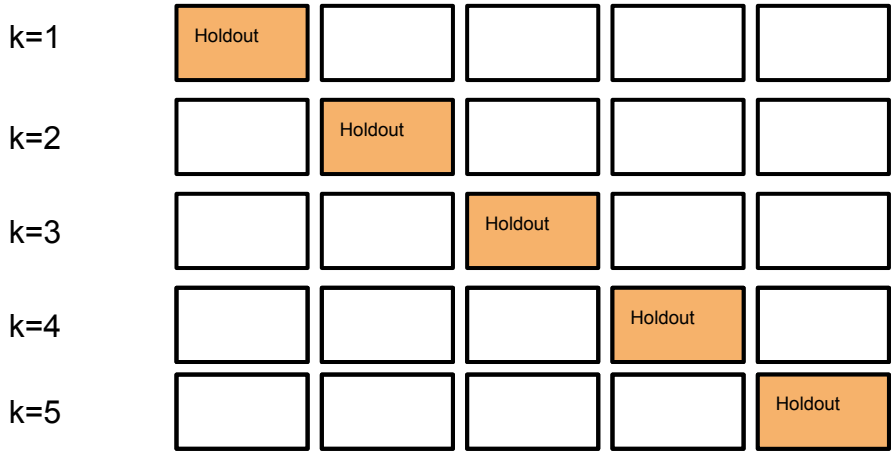
Pick model that does best on Validation Set
Confirm results on Test Set

Cross-Validation

- If we have a small dataset: CV can be used
- Idea is simple but smart:
  - Use your initial training data to generate multiple mini train-test splits. Use these splits to evaluate your model
  - K is a hyper-parameters. K is equal to the number of generated train-test splits.

Cross-Validation

- Partition data into k subsets, i.e., folds
- Iteratively train the model on k-1 folds while using the remaining fold as the test set (hold-out set)
- Compute the average performances over the K folds



CV

- Divide into three sets
  - Training set
  - Validation set
  - Test set
- Classic gotcha: only train the model on training data
  - Getting surprisingly low loss?
  - Check the whole procedure

How to detect overfitting

- After training/testing splitting, training loss is much less than testing loss.
- Start with a simple model as the benchmark
    - When add model complexity, you will have a reference point to see whether the additional complexity is worthy.

How to prevent overfitting

- Train with more data
  - Filter noisy data (outlier)
- Remove features
  - Remove irrelevant features
- Regularization
  - Control model complexity
  - Different machine learning models have their own regularization methods.

sklearn.linear_model.Ridge

class sklearn.linear_model.**Ridge**(*alpha=1.0*, *fit_intercept=True*, *normalize=False*, *copy_X=True*, *max_iter=None*, *tol=0.001*, *solver='auto'*, *random_state=None*) [source]

Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the User Guide.

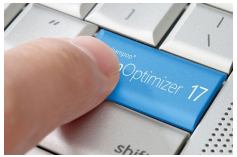
Alpha is the controlling parameter, which is also hyperparameter

3. Hyperparameter Optimization

Hyperparameters

- Machine learning algorithms usually have two kinds of weights:
  - ***Parameters***: learned by data during training such as slope of linear regression, layer weights of neural networks
  - **Hyperparameters**: left to us to select beforehand such as K in KNN, number of layers in neural networks

Hyperparameters



Parameters

Scores

Hyperparameters

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge()
```

Hyperparameters should be passed when you initialize the machine learning model **before training**

Hyperparameters Tuning

- Weaker models with well-tuned hyperparameters can outperform more complex models
- However, each model has so many hyperparameters to be tuned

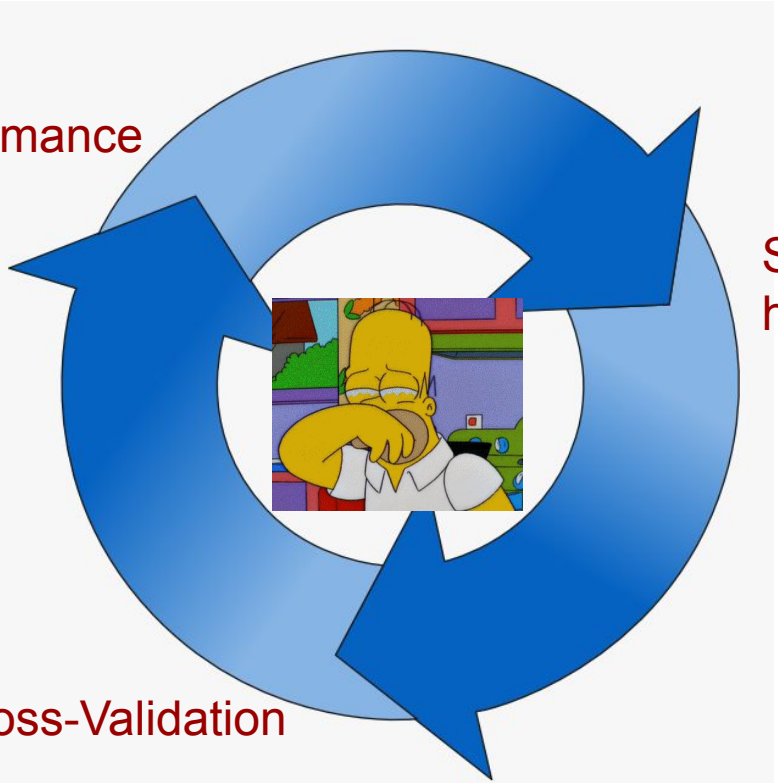
class transformers.LlamaConfig <source>

```
( vocab_size = 32000, hidden_size = 4096, intermediate_size = 11008, num_hidden_layers = 32,
num_attention_heads = 32, num_key_value_heads = None, hidden_act = 'silu', max_position_embeddings =
2048, initializer_range = 0.02, rms_norm_eps = 1e-06, use_cache = True, pad_token_id = None,
bos_token_id = 1, eos_token_id = 2, pretraining_tp = 1, tie_word_embeddings = False, rope_theta =
10000.0, rope_scaling = None, attention_bias = False, **kwargs )
```

Llama from hugginface

Searching is Iterative, then Expensive

Track the performance

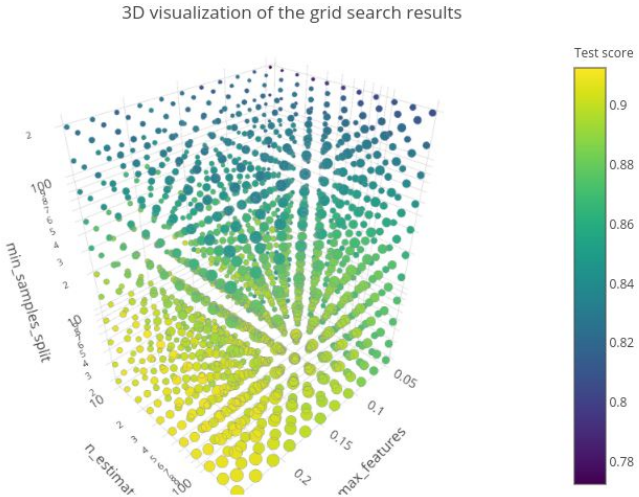


Select one potential hyperparameter set

Run Cross-Validation

Grid Search

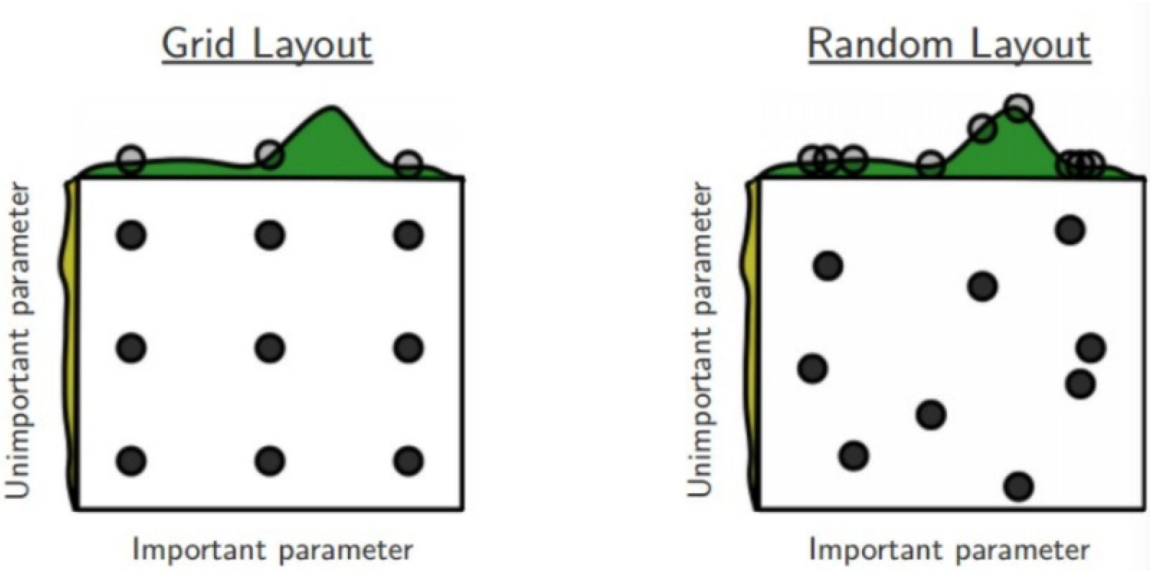
- Define a grid on n-dimensions, where each of these maps for an hyperparameter
- For each dimension, define the range of possible values
- Search for all combinations and select the best one



Inefficient !!!

Random Search

- Randomly pick the point from the configuration space
- The rest is the same as grid search



Good on high-dim spaces

From Bergstra and Bengio

Bayesian optimization

- For grid and random search, the previous trials can not contribute to each new guess.
- Try to model the hyperparameter search as a machine learning task
  - Tree-structured Parzen Estimator
  - Gaussian Process
  - Other bayesian optimization methods

Main idea: based on the distribution of the previous results, decide which set of parameters should be explored firstly

GCP Implementation:
https://cloud.google.com/blog/products/ai-machine-learning/hyperparameter-tuning-cloud-machine-learning-engine-using-bayesian-optimization

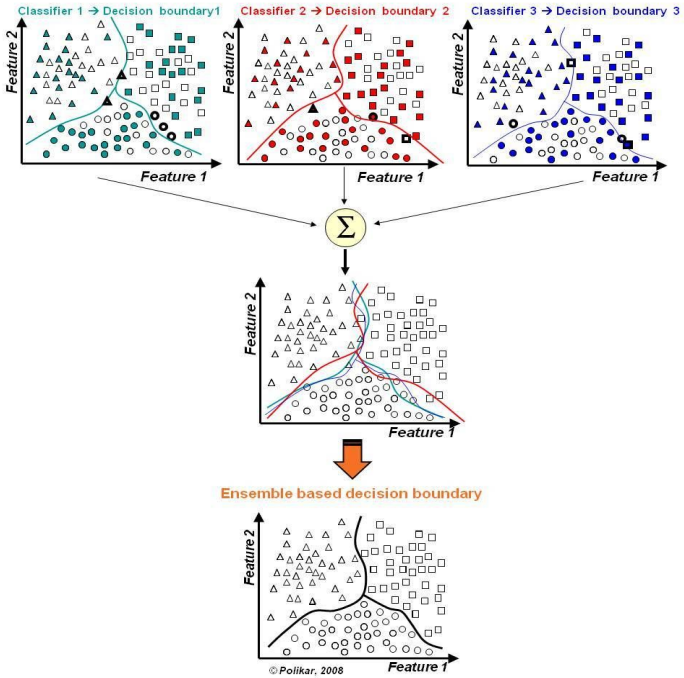
Hyperparameter tuning

- Hyperparam tuning has become a standard part of ML workflows
- Built-in with frameworks
  - Tensorflow: KerasTuner
  - **Scikit-learn**: auto-sklearn
  - **Ray**: Tune
- Distributed optimization library
  - Hyperopt

4. Ensemble

Ensemble

- Creating a strong model from an ensemble of weak models (base learners)



Ensembles: wining leaderboard (Kaggle & SOTA)

Leaderboard

SQuAD2.0 tests the ability of a system to not only answer reading comprehension questions, but also abstain when presented with a question that cannot be answered based on the provided paragraph.

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Sep 18, 2019	ALBERT (ensemble model) *Google Research & TTIC* https://arxiv.org/abs/1909.11942	**89.731**	**92.215**
2 Jul 22, 2019	XLNet + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	88.592	90.859
2 Sep 16, 2019	ALBERT (single model) *Google Research & TTIC* https://arxiv.org/abs/1909.11942	88.107	90.902
2 Jul 26, 2019	UPM (ensemble) *Anonymous*	88.231	90.713
3 Aug 04, 2019	XLNet + SG-Net Verifier (ensemble) *Shanghai Jiao Tong University & CloudWalk* https://arxiv.org/abs/1908.05147	88.174	90.702

1st PLACE - WINNER SOLUTION - Gilberto Titericz & Stanislav Semenov

1st PLACE SOLUTION - Gilberto Titericz & Stanislav Semenov

First, thanks to Organizers and Kaggle for such great competition.

Our solution is based in a 3-layer learning architecture as shown in the picture attached.

-1st level: there are about 33 models that we used their predictions as meta features for the 2nd level, also there are 8 engineered features.

https://www.kaggle.com/c/otto-group-product-classification-challenge/discussion/14335

Why does ensembling work

- Task: credit card fraud detection (Normal/Fraudulent)
- 3 **uncorrelated** models, each with accuracy of 80%
- Ensemble: Majority voting
    - When at least two models are correct, ensemble model would be correct

Why does ensembling work

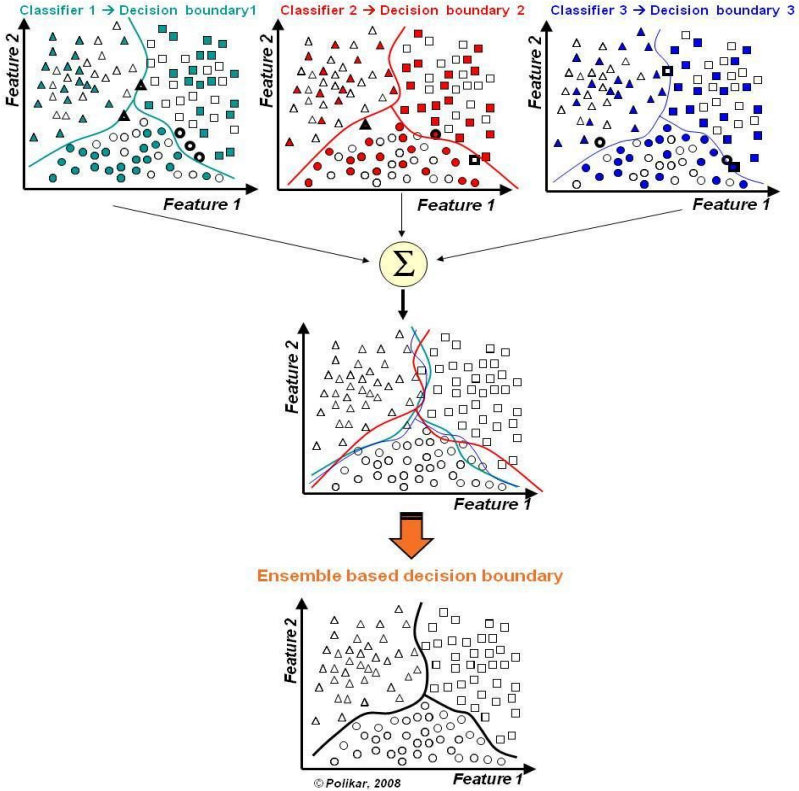
- Ensemble Accuracy:
  - Probability that at least two models are correct:

Outputs of 3 models	**Probability**	**Ensemble's output**
All 3 are correct	0.8 * 0.8 * 0.8 = 0.512	Correct
Only 2 are correct	(0.8 * 0.8 * 0.2) * 3 = 0.384	Correct
Only 1 is correct	(0.2 * 0.2 * 0.8) * 3 = 0.096	Wrong
None is correct	0.2 * 0.2 * 0.2 = 0.008	Wrong

Why does ensembling work

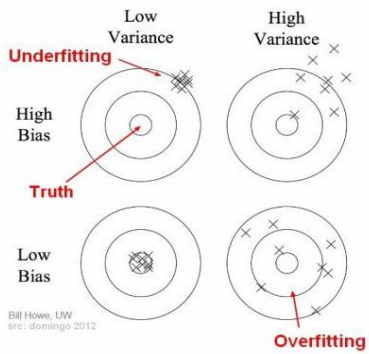
- Reduce Bias
- Reduce Variance

Prediction Error = Bias ^ 2 + Variance + Irreducible Error



Bias-Variance

- Bias:
  - The difference between the average prediction of our model and the correct value which we are trying to predict
- Variance:
  - The variability of model prediction for a given data point or a value which tells us spread of data



Reduce Bias

- Assume a test set of 10 samples and k (assume k is odd) **uncorrelated** binary classifiers, where each classifier has ***p*** accuracy
- The accuracy of ensembling using majority voting
  - The probability that majority of classifiers are correct

$$\sum_{i=0}^{int(\frac{k}{2})} \binom{k}{i} p^{k-i}(1-p)^{i}$$

What is the probability that k choose ***i classifiers*** whose predictions are **wrong** and the rest ***k-i models***' outputs are **correct**.

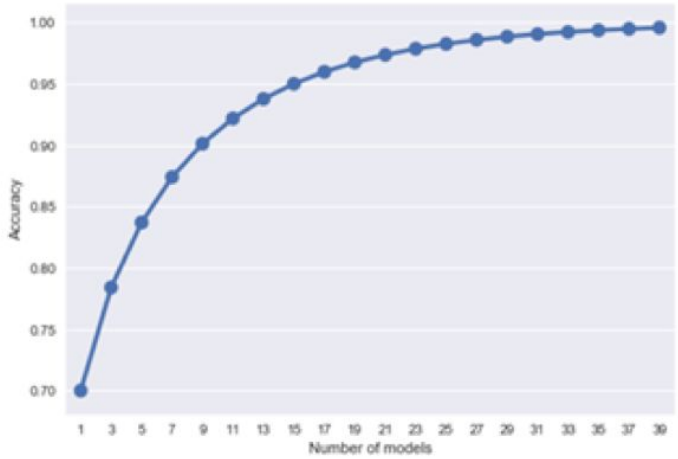
Reduce Bias

- Change the number of models

$$\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{i} p^{k-i} (1-p)^i$$

If p = 0.7, then we have

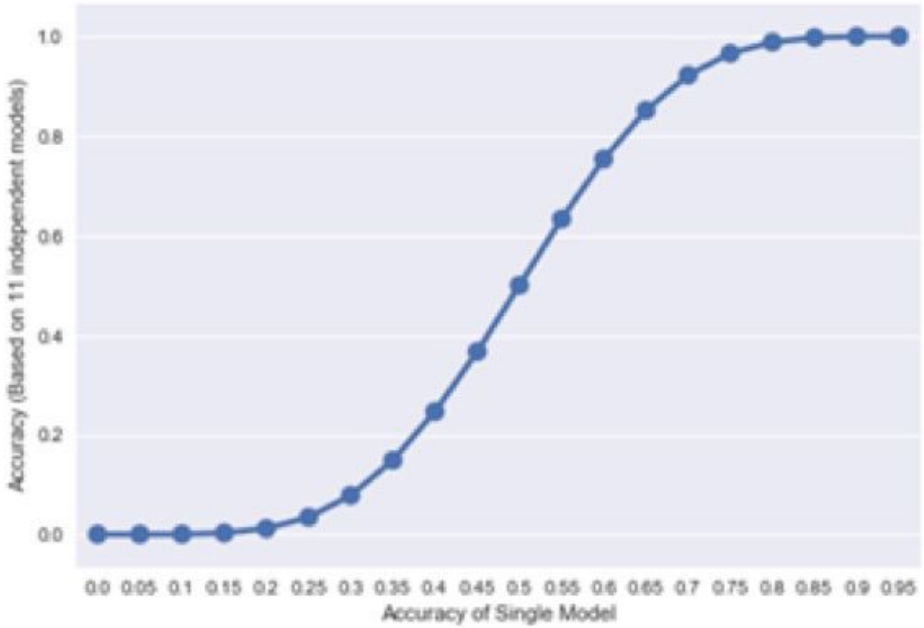
k	Ensemble Accuracy
1	0.7
3	0.784
5	0.83692
11	0.92177520904
101	0.999987057446



Reduce Bias

- Change the accuracy of the base model $\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{i} p^{k-i}(1-p)^i$

Fix # of classifiers to be 11



Reduce Variance

- Suppose we have n **independent** models: M1, M2, …, Mn with the same variance σ ^2
- The ensemble M* constructed from those models using averaging will have the variance as follows:

$$
\begin{aligned}
Var(M^*) &= Var(\frac{1}{n}\sum_i M_i) \\
&= \frac{1}{n^2} Var(\sum_i M_i) \\
&= \frac{1}{n^2} * n * Var(M_i) \\
&= \frac{Var(M_i)}{n}
\end{aligned}
$$

$$\sigma^2 \rightarrow \frac{\sigma^2}{n}$$

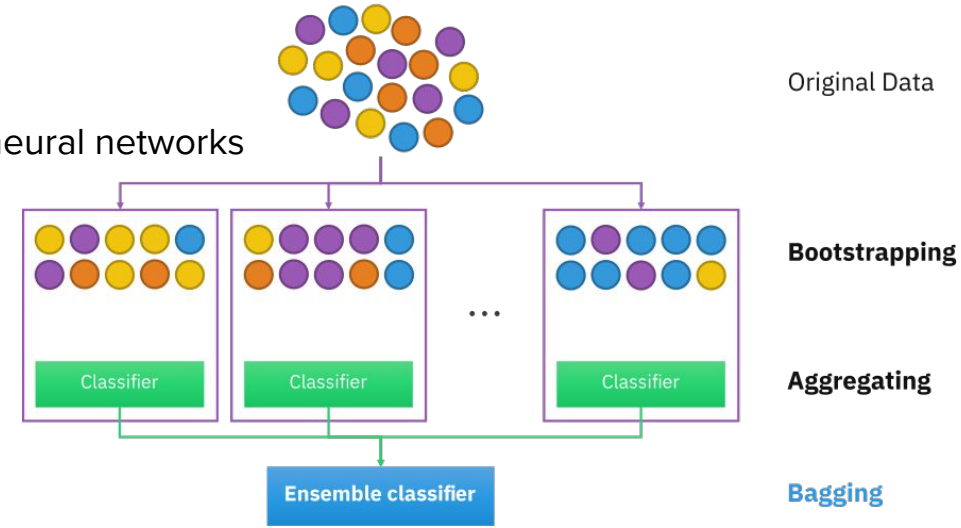
Ensemble

- Bagging: reduce the variance in the model
  - Random Forest
- Boosting: reduce the bias in a model
  - Ada-Boost, XGBoost
- Stacking: increase the prediction accuracy of a model
  - MLxtend

- **The less correlation among base learners, the better**
- **Try to have different model architectures for base learners**

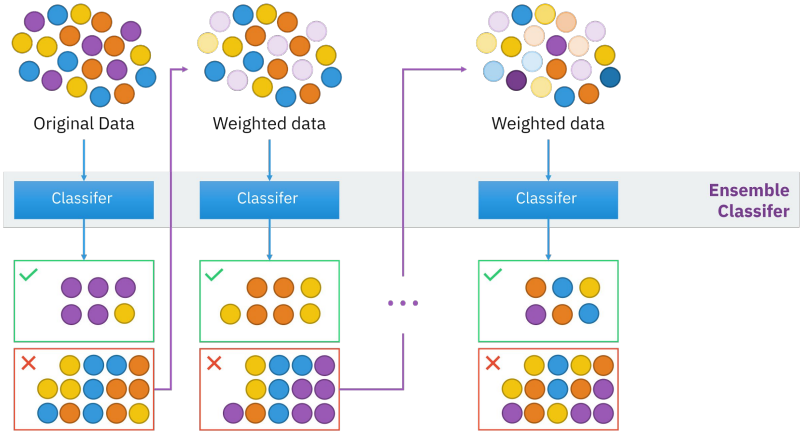
Bagging

- Sample with replacement to create different datasets
- Train a classifier with each dataset
- Aggregate predictions from classifiers
  - Majority Voting:
    - Equal and weighted combinations
- Decreases errors by decreasing the variance
- Can improve unstable methods such as trees, neural networks



Boosting

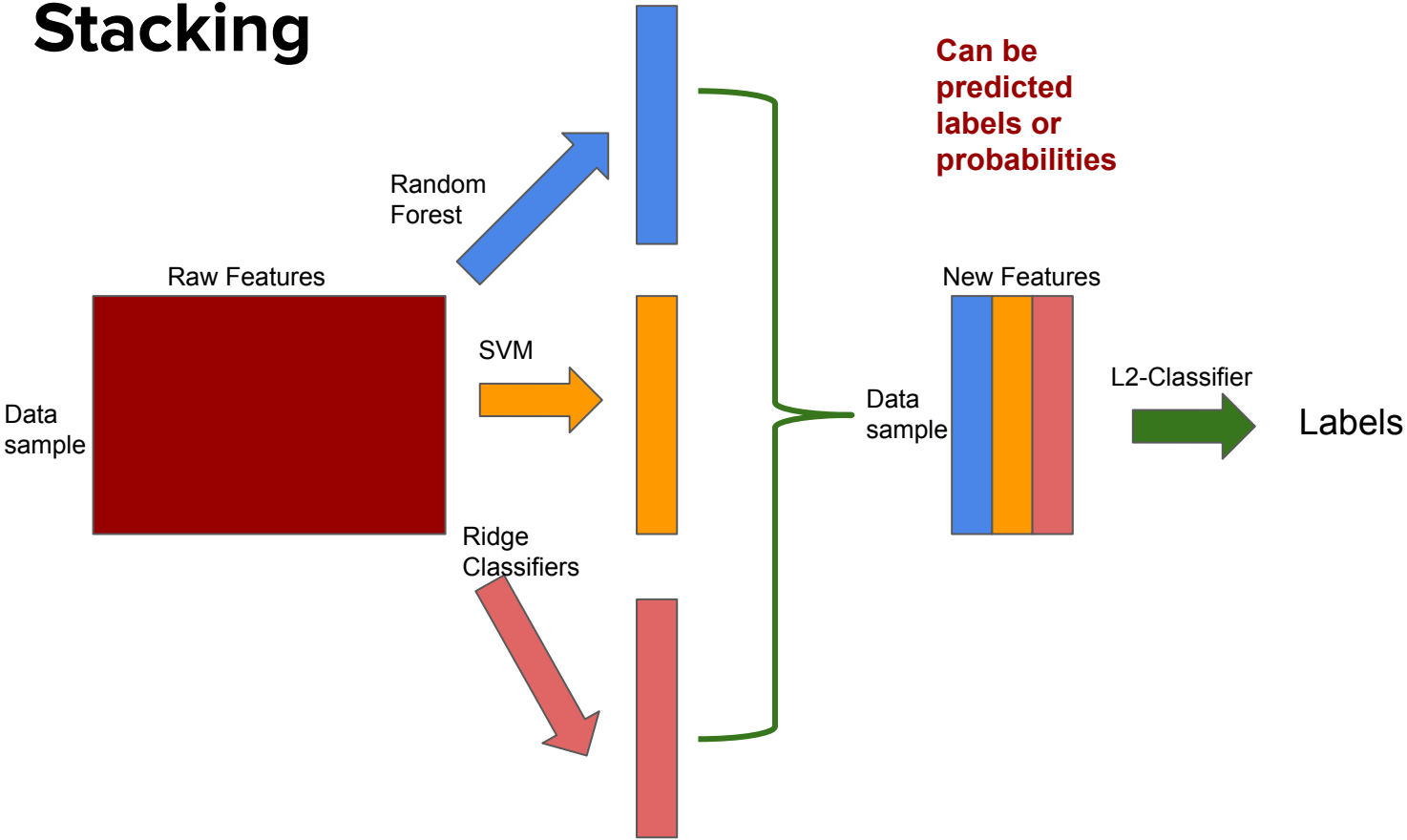
- Train a weak classifier
- Give samples misclassified by weak classifier higher weight
- Repeat (1) on this reweighted data as many iterations as needed
- Final strong classifier: weighted combination of existing classifiers
  - classifiers with smaller training errors have higher weights
- Popular methods for tabular data:
  - Gradient Boosting
  - AdaBoost
  - XGBoost
  - LigtGBM



Stacking

- Core idea: use a pool of base classifiers, then using another classifier (stacker) to combine their prediction for the final decision

Stacking


Random
Forest
Raw Features
Data
sample
SVM
Ridge
Classifiers
Can be
predicted
labels or
probabilities
New Features
Data
sample
L2-Classifier
Labels

Some possible pitfalls of Ensemble

- Exponentially increasing training times and computational requirements
- Increase demand on infra. To maintain and update these models
- Greater chance of data leakage between models or stages in the whole training

5. Do not sleep on traditional machine learning

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
ISIR, CNRS, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

Model comparison

Tree-based Models outperform deep learning on tabular data

Based on 45 middle-sized datasets (10, 000 samples)

From this paper, authors explain:

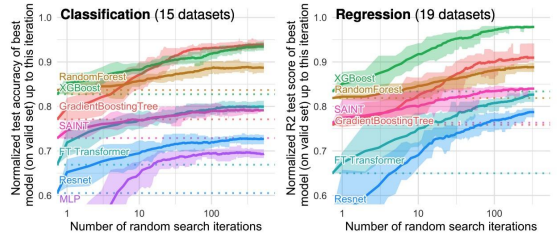


Figure 1: **Benchmark on medium-sized datasets, with only numerical features**. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

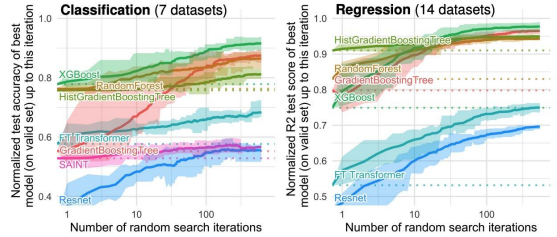


Figure 2: **Benchmark on medium-sized datasets, with both numerical and categorical features**. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

- Deep learning bias to the overly smooth solution, while tree-based models are able to generate irregular decision boundaries
- Deep learning are very sensitive to uninformative features which could be easily spotted in tabular data, while tree-based models are more robust

Deep Learning for time series data

- “Results show that competitive performance can be achieved with a conventional machine learning pipeline consisting of **preprocessing**, **feature extraction**, and a **simple machine learning model**. In particular, we analyze the performance of a linear model and a non-linear (gradient boosting) model”

Table 3
Comparison between the proposed classical machine learning pipeline and other (deep learning) solutions using macro F1-score (MF1), overall accuracy (ACC), and Cohen's Kappa coefficient (κ). The approaches are sorted according to macro F1. The scores in bold represent the best score for each dataset (that are comparable to our approach).

Dataset	Year	System	Technique	LP	MF1	ACC	κ	Signals
Sleep-EDF-SC-20	2021	RobustSleepNet [28]	RNN	FT	0,817	–	–	EEG + EOG
	2022	*This work*	Catboost	DT	**0,810**	**0,866**	**0,816**	EEG + EOG + EMG
	2021	XSleepnet2 [46]	CNN & RNN	LFS	0,809	0,864	0,813	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,809	0,857	0,806	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	DT	0,805	0,863	0,813	EEG + EOG + EMG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,805	0,854	0,800	EEG
	2020	SimpleSleepNet [48]	RNN	LFS	0,805	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,803	0,853	0,800	EEG + EOG
	2022	*This work*	Catboost	LFS	0,802	0,864	0,812	EEG + EOG + EMG
	2022	*This work*	Catboost	LFS	0,797	0,860	0,807	EEG + EOG
	2019	SleepEEGNet [44]	CNN & RNN	LFS	0,797	0,843	0,790	EEG
	2020	SeqSleepNet+ [45]	RNN	FT	0,796	0,852	0,789	EEG
	2021	RobustSleepNet [28]	RNN	LFS	0,791	–	–	EEG + EOG
	2021	RobustSleepNet [28]	RNN	DT	0,791	–	–	EEG + EOG
	2020	DeepSleepNet+ [45]	CNN	FT	0,790	0,846	0,782	EEG + EOG
	2021	DeepSleepNet-Lite [15]	CNN	LFS	0,780	0,840	0,780	EEG
	2019	IITNet [43]	CNN & RNN	LFS	0,776	0,839	0,780	EEG
	2017	DeepSleepNet [41]	CNN & RNN	FT	0,769	0,820	0,760	EEG
Sleep-EDF-SC-78	2022	SleepTransformer [40]	transformer	FT	0,788	0,849	0,789	EEG
	2021	XSleepnet2 [46]	CNN & RNN	LFS	**0,787**	**0,840**	**0,778**	EEG + EOG
	2020	XSleepnet1 [46]	CNN & RNN	LFS	0,784	0,840	0,777	EEG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,781	0,831	0,770	EEG
	2021	RobustSleepNet [28]	RNN	FT	0,779	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	0,775	0,831	0,766	EEG + EOG + EMG
	2022	*This work*	Catboost	LFS	0,772	0,830	0,763	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,771	0,821	0,756	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	LFS	0,768	0,820	0,753	EEG + EOG
	2021	RobustSleepNet [28]	RNN	LFS	0,763	–	–	EEG + EOG
	2021	DeepSleepNet-Lite [15]	CNN	LFS	0,752	0,803	0,730	EEG
	2022	SleepTransformer [40]	transformer	LFS	0,743	0,814	0,743	EEG
	2021	RobustSleepNet [28]	RNN	DT	0,738	–	–	EEG + EOG
	2019	SleepEEGNet [44]	CNN & RNN	LFS	0,736	0,800	0,730	EEG
Sleep-EDF-ST	2021	RobustSleepNet [28]	RNN	FT	0,810	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	**0,795**	**0,836**	**0,765**	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	LFS	0,792	0,829	0,759	EEG + EOG + EMG
	2021	RobustSleepNet [28]	RNN	DT	0,791	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	0,789	0,832	0,758	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,788	0,825	0,754	EEG + EOG
	2021	RobustSleepNet [28]	RNN	LFS	0,786	–	–	EEG + EOG
	2020	DeepSleepNet+ [45]	CNN	FT	0,775	0,815	0,738	EEG
	2020	SeqSleepNet+ [45]	RNN	FT	0,775	0,810	0,734	EEG
MASS SS3	2020	SimpleSleepNet [48]	RNN	LFS	**0,847**	–	–	EEG + EOG
	2021	RobustSleepNet [28]	RNN	FT	0,840	–	–	EEG + EOG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,832	**0,875**	**0,820**	EEG
	2021	RobustSleepNet [28]	RNN	DT	0,822	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	0,817	0,867	0,803	EEG + EOG + EMG
	2017	DeepSleepNet [41]	CNN & RNN	FT	0,817	0,862	0,800	EEG
	2022	*This work*	Catboost	LFS	0,809	0,863	0,797	EEG + EOG
	2021	RobustSleepNet [28]	RNN	DT	0,808	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,807	0,853	0,786	EEG + EOG + EMG
	2019	IITNet [43]	CNN & RNN	LFS	0,805	0,863	0,790	EEG
	2021	U-Sleep [29]	CNN	DT	0,800	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,794	0,845	0,775	EEG + EOG

Do not sleep on traditional machine learning: Simple and interpretable techniques are competitive to deep learning for sleep scoring ☆

Jeroen Van Der Donckt [1] ✉, Jonas Van Der Donckt [1], Emiel Deprost, Nicolas Vandenbussche, Michael Rademaker, Gilles Vandewiele, Sofie Van Hoecke

Show more

Source:
https://www.sciencedirect.com/science/article/abs/pii/S1746809422008837

Deep Learning for unstructured data

- Deep learning are good at capturing high dimensional and spatial patterns/interactions among data
- Therefore, in those domains such as image, video, and text, deep learning is able to achieve huge success especially enough data are present

Next Class: Model Evaluation