Psychoinformatics & Neuroinformatics

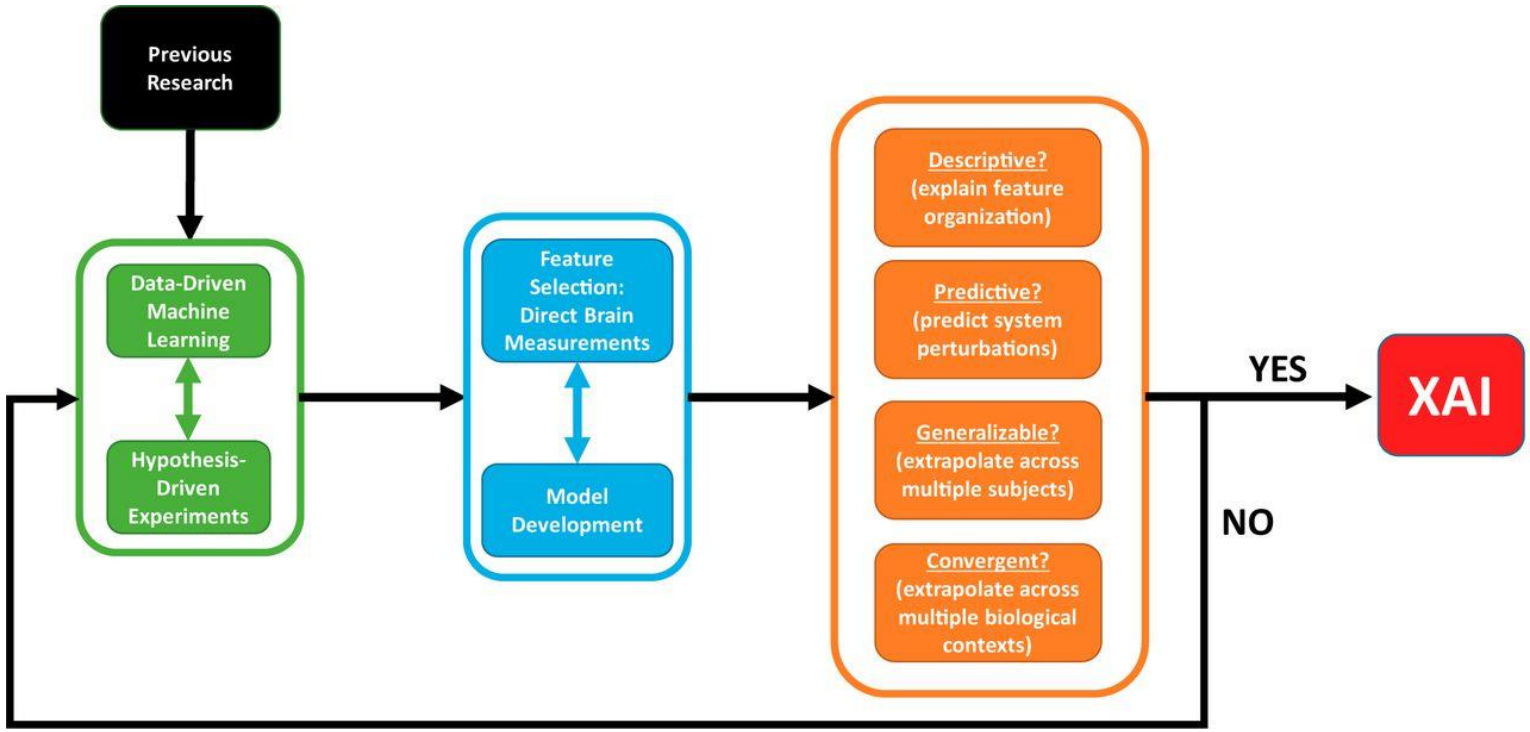
Week 10

Machine Learning (2/3)

by Tsung-Ren (Tren) Huang 黃從仁

A Shared Vision for Machine Learning in Neuroscience

Mai-Anh T. Vu,[1] Tülay Adalı,[8] Demba Ba,[9] György Buzsáki,[10] David Carlson,[3,4] Katherine Heller,[5] Conor Liston,[11] Cynthia Rudin,[6,7] Vikaas S. Sohal,[12] Alik S. Widge,[13] Helen S. Mayberg,[14] Guillermo Sapiro,[6] and Kafui Dzirasa[1,2]



We need to get a well-working model first!

Goals for today

Model selection criteria

Which model is better?

Hyperparameter tuning

The ones that specify model complexity

Hybrid/mixed learning

Machines' crowd intelligence

Goals for today

Model selection criteria

Which model is better?

Hyperparameter tuning

The ones that specify model complexity

Hybrid/mixed learning

Machines' crowd intelligence

Common Pitfalls (3/3)

Why getting a **better**-than-chance accuracy if there is no relationship between X_i and Y_i?

```
from sklearn.model.selection import *
from sklearn.metrics import *
x=random.rand(100,3) # 3-d random features
y=random.permutation([0]*90+[1]*10) # 2 categories
clf=svm.SVC(); cv=KFold(100)
yp=cross_val_predict(clf,x,y,cv=cv)# leave-1-out
print('Accuracy:',mean(y==yp)) # mean accuracy =0.9
print('C. Matrix:\n',confusion_matrix(y,yp)) # c. matrix
```

Accuracy is not the best target

Say we are predicting who are leaving the job (Y=1).
This is the result from a focus on prediction **accuracy**:

	Actual 0 (staying)	Actual 1 (leaving)
Predict 0 (staying)	a: 90	b: 10
Predict 1 (leaving)	c: 0	d: 0

Model **precision**: actual1/predict1=d/(c+d)=0 here
Model **recall**: predict1/actual1=d/(b+d)=0 here

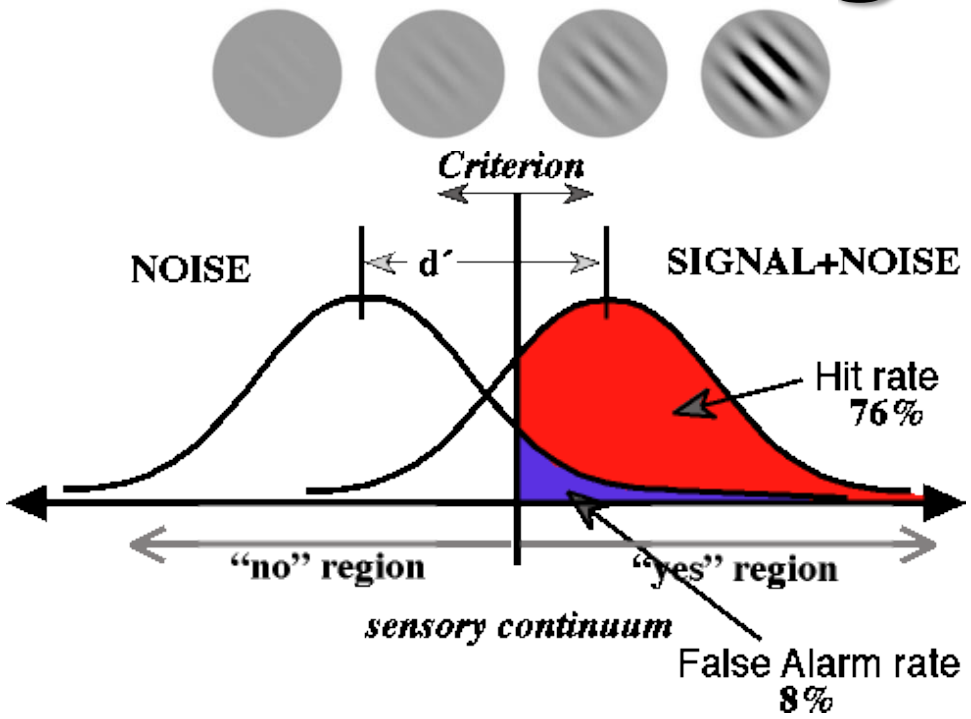
Beyond Accuracy

		True condition		
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive** (Type I error)	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Test outcome positive}}$
	Predicted condition negative	**False negative** (Type II error)	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Test outcome negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	True positive rate (TPR), Sensitivity, Recall $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$	

F1=harmonic mean=2*Precision*Recall/(Precision+Recall)

G=geometric mean=sqrt(Precision*Recall)

Receiver Operating Characteristic (ROC)

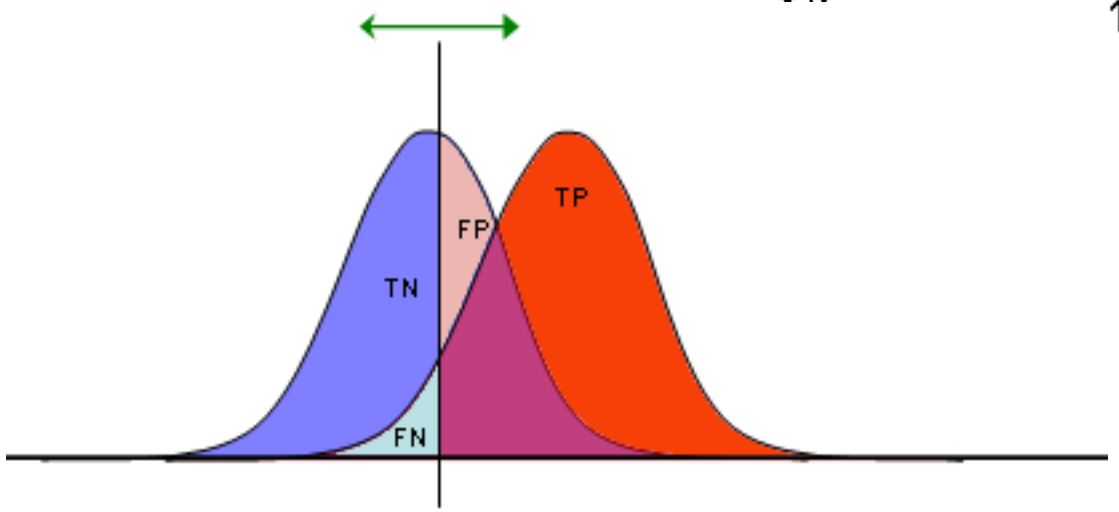


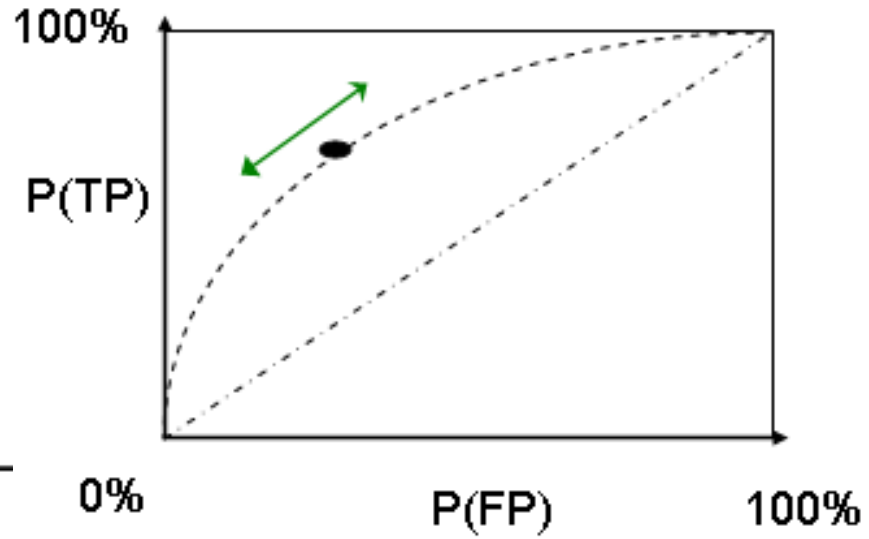

```
knn.predict_proba(X_test)[:9]
```

```
array([[1. , 0. ],
       [0.6, 0.4],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0.6, 0.4],
       [0.6, 0.4],
       [1. , 0. ],
       [0. , 1. ]])
```

```
knn.predict(X_test)[:9]
```

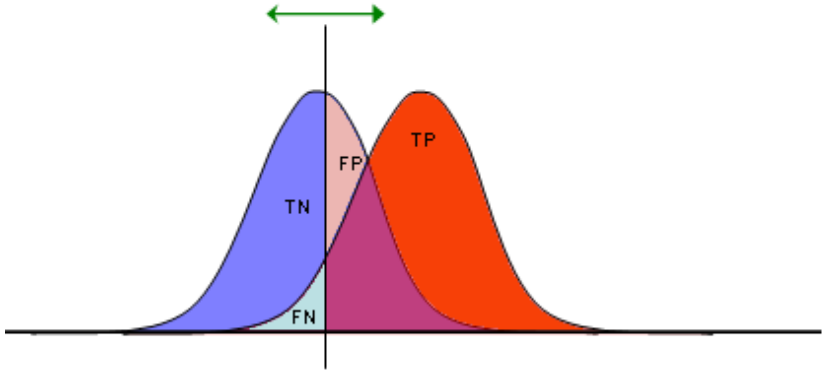
```
array([0, 0, 0, 0, 1, 0, 0, 0, 1], dtype=uint8)
```

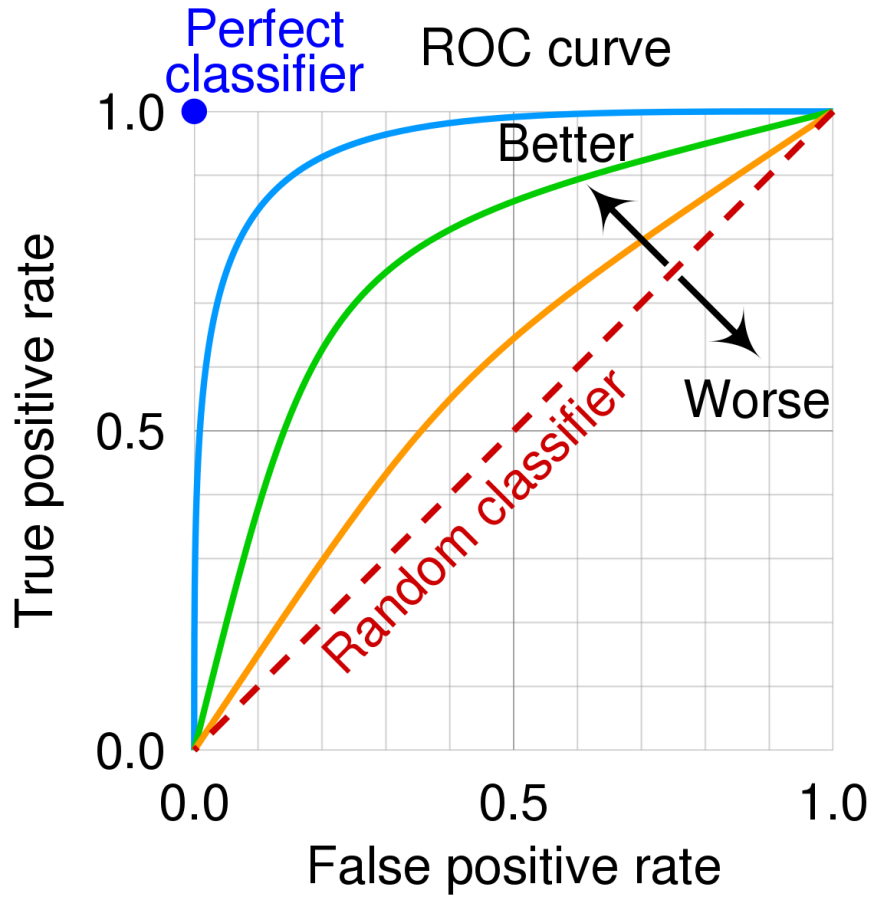




Receiver Operating Characteristic (ROC)

Area Under Curve (AUC) characterizes the discriminability of a classifier:





Goals for today

Model selection criteria

Which model is better?

Hyperparameter tuning

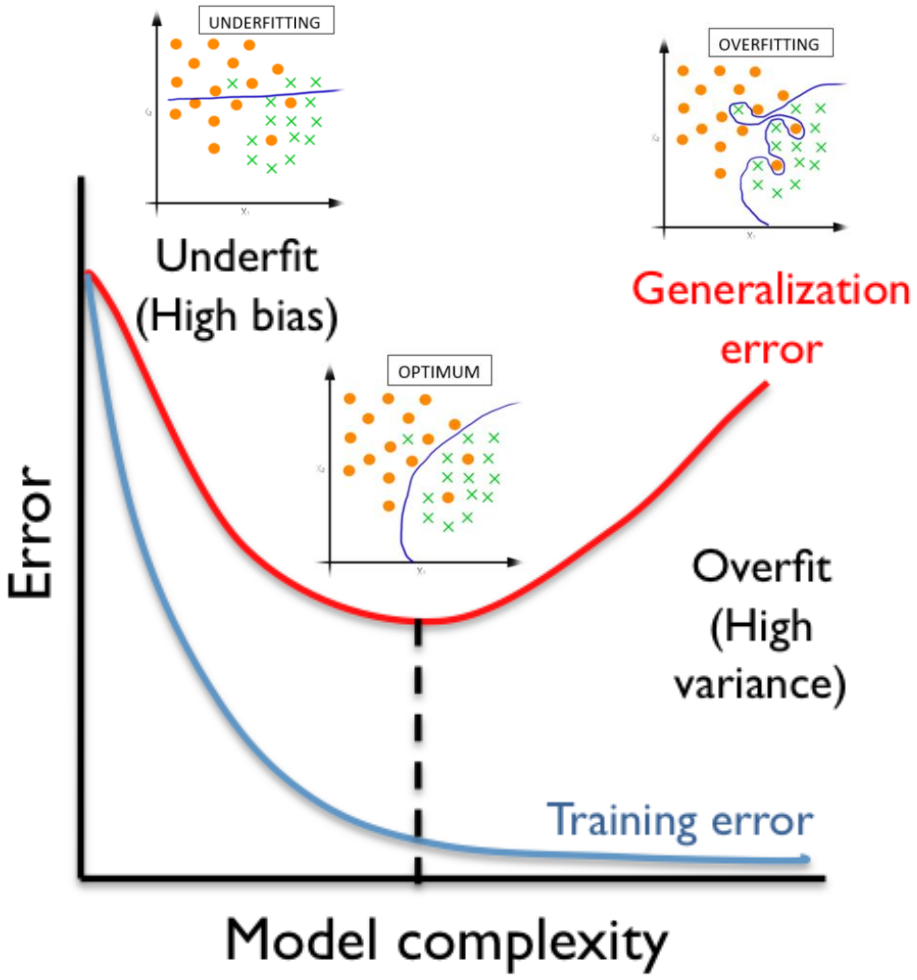
The ones that specify model complexity

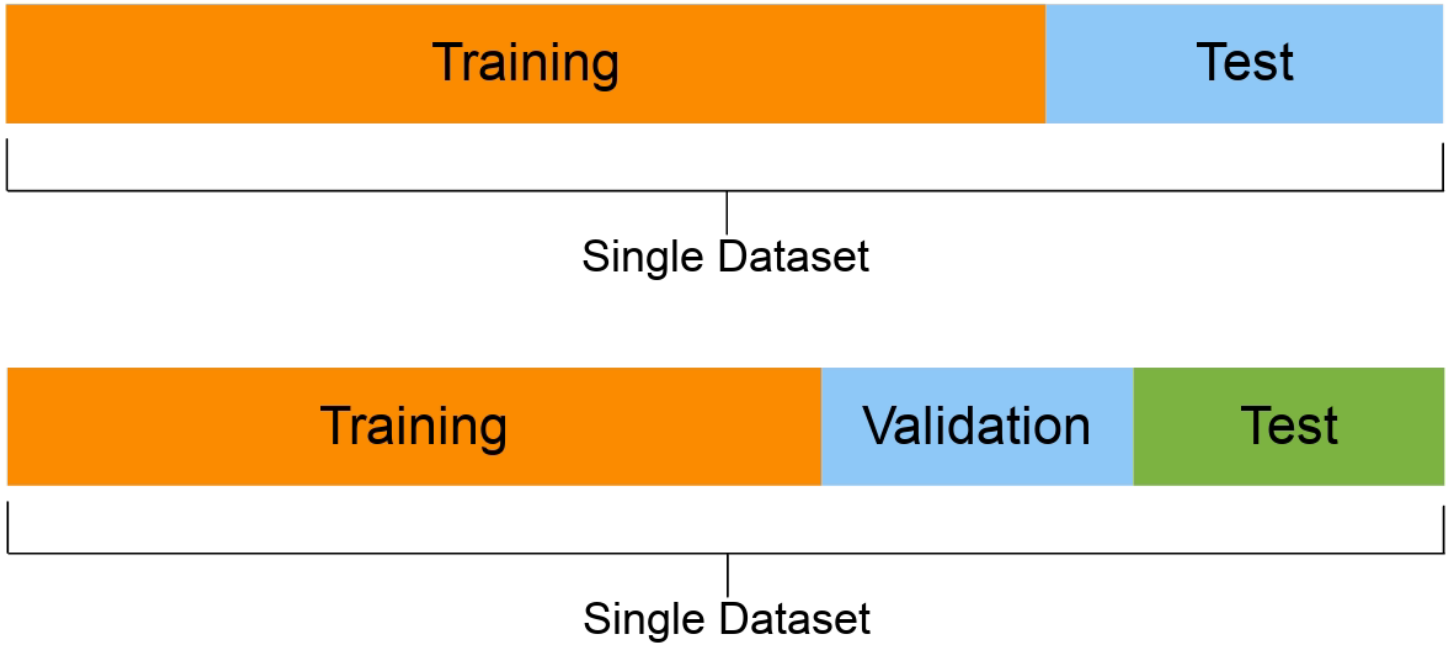
Hybrid/mixed learning

Machines' crowd intelligence

Performance=f(Data, Model Complexity)

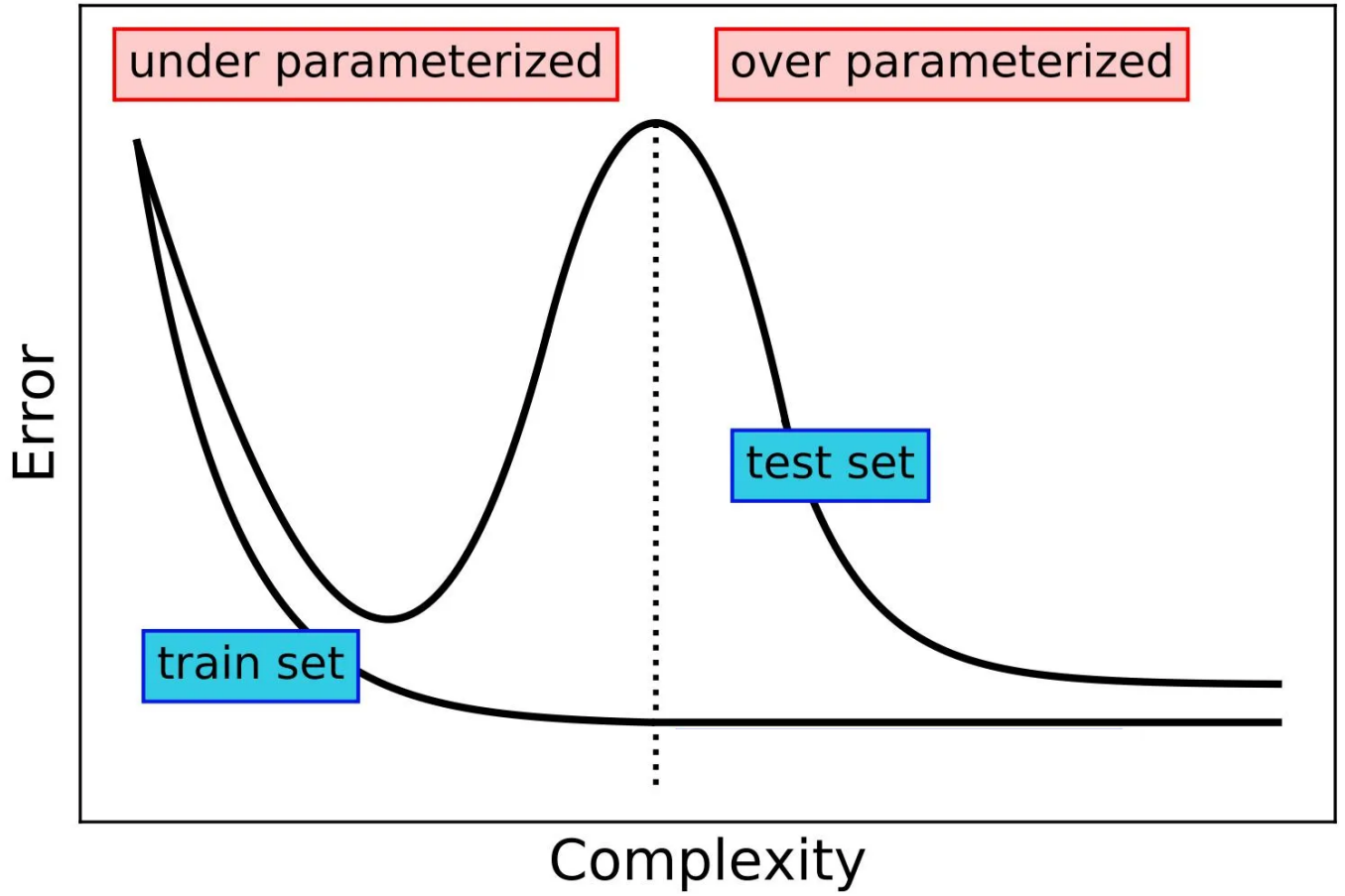
Changing model complexity to maximize Perf(Test Data)





Double Descent : What?

Classical vs. Modern Machine Learning (i.e., Deep Learning)

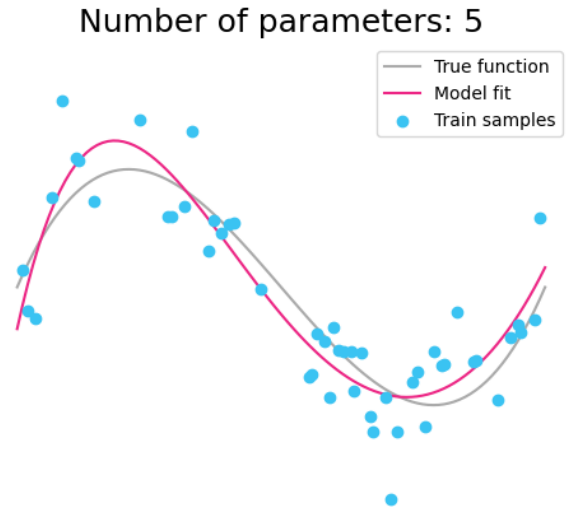


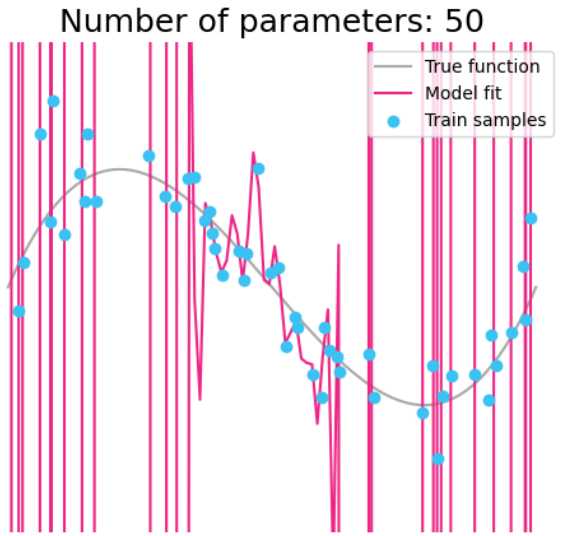
Modern ML: ~~Bigger models are always better~~

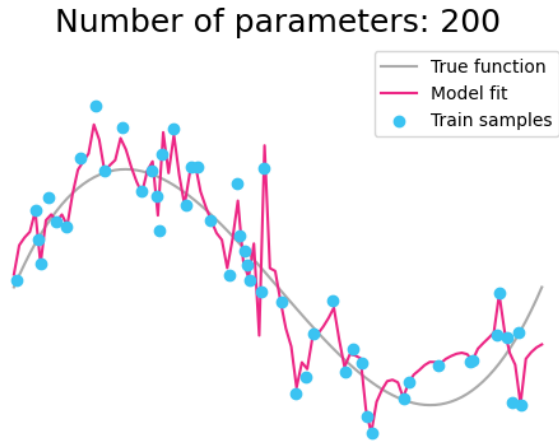
Classical ML: ~~Too large models are worse~~

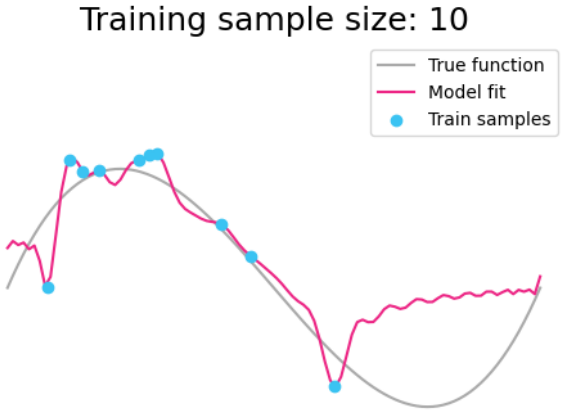
(Data) Double Descent : Examples

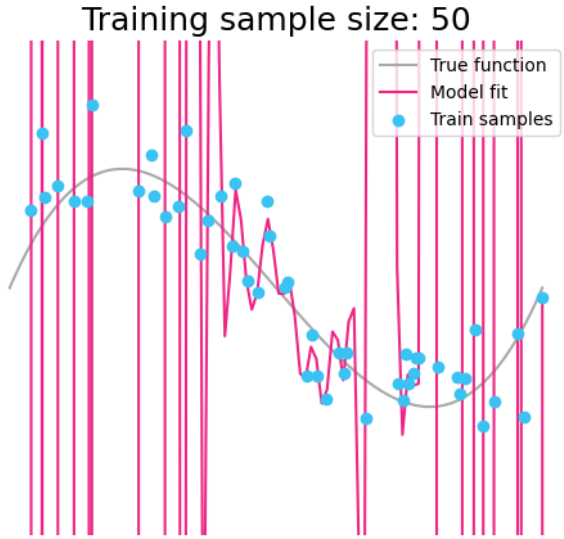
Big data: ~~More data is always better~~





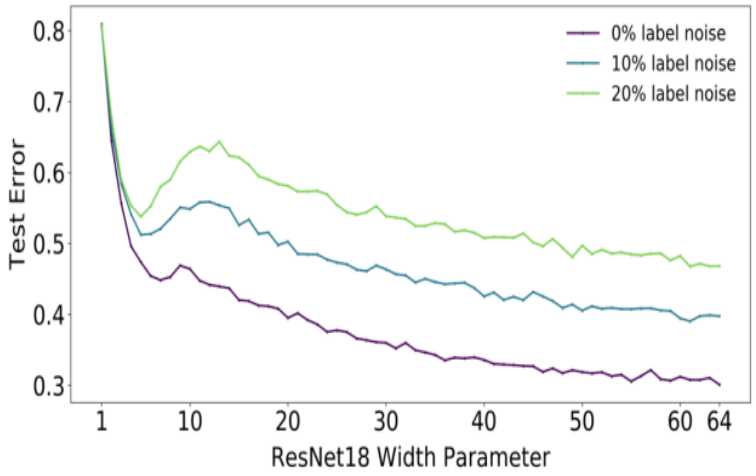


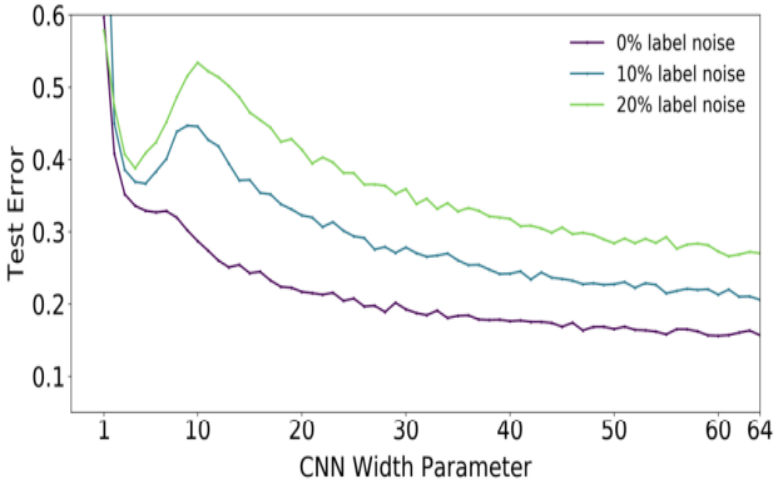


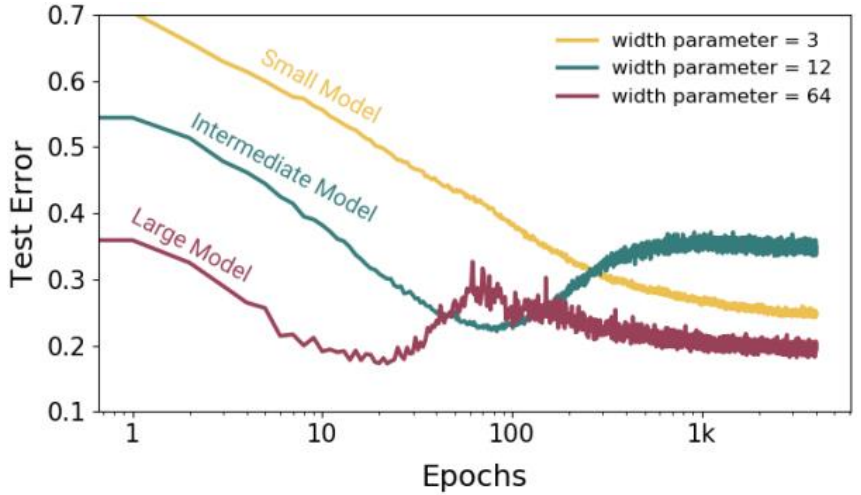


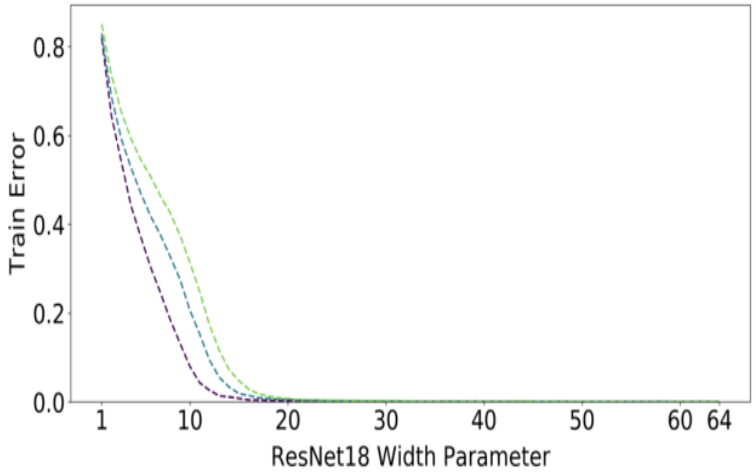
Model- & Epoch-Wise Double Descent

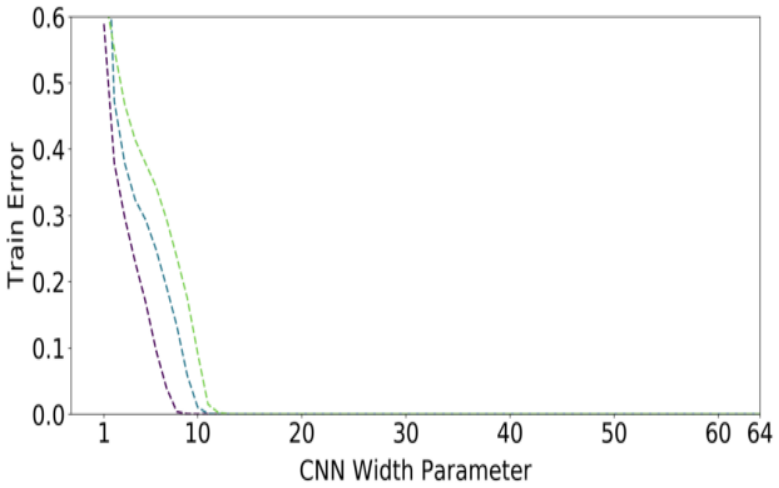
It occurs in various models (ResNet, CNN, Transformer, …)

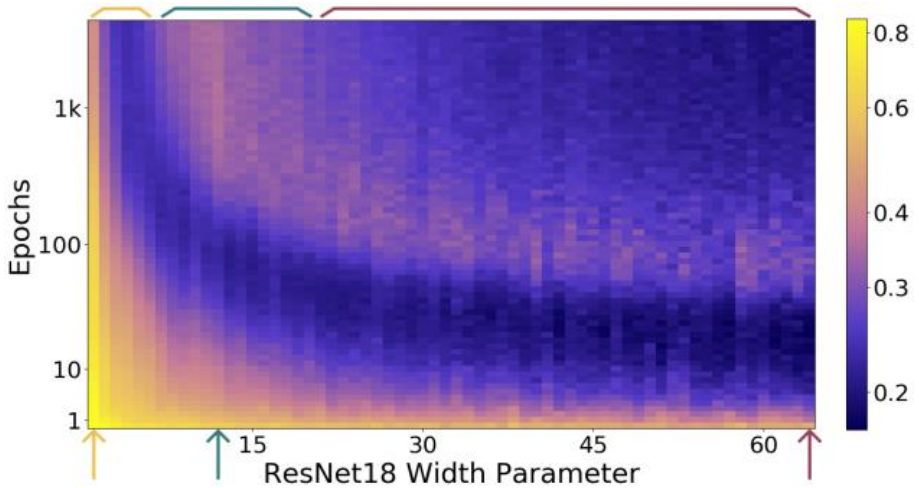






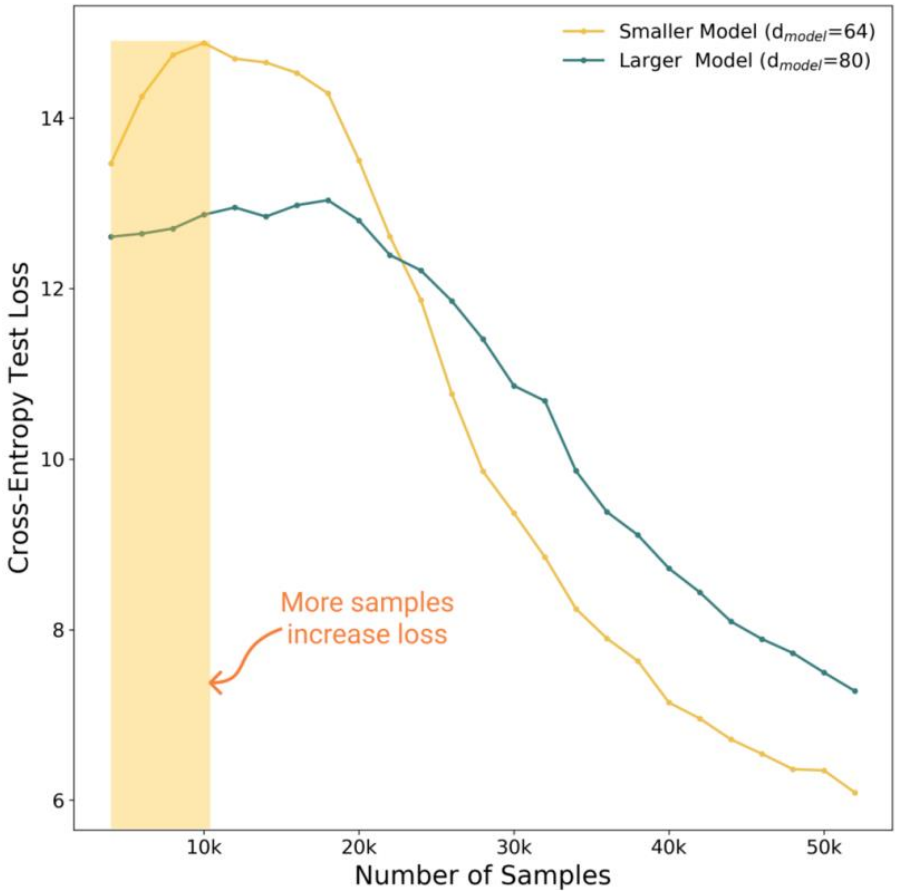


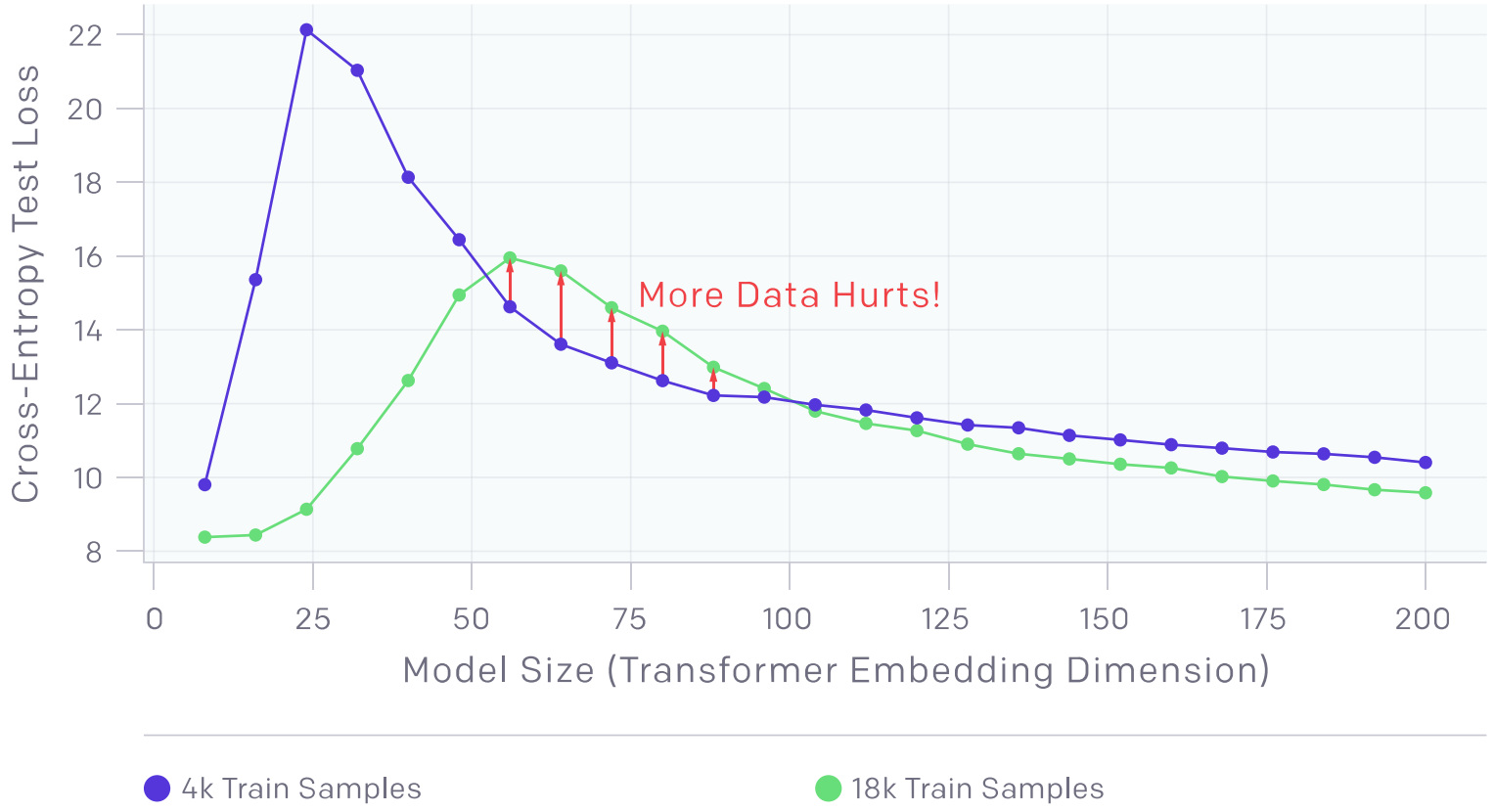




Sample-wise Double Descent

Super big data tends to be better





Double Descent: Why

Regularization picks the *simplest* model of all possibilities

A Why Gradient Descent Implicitly Regularizes

This is a sketch of why gradient descent implicitly regularizes. Suppose we have a model Xw for a

space norm. Choosing the smoothest function that perfectly fits observed data is a form of Occam's razor: The simplest explanation compatible with the observations should be preferred (cf. refs. 7 and 8). By considering larger function classes, which contain more candidate predictors compatible with the data, we are able to find interpolating functions that have smaller norm and are thus "simpler." Thus, increasing function class capacity improves performance of classifiers.

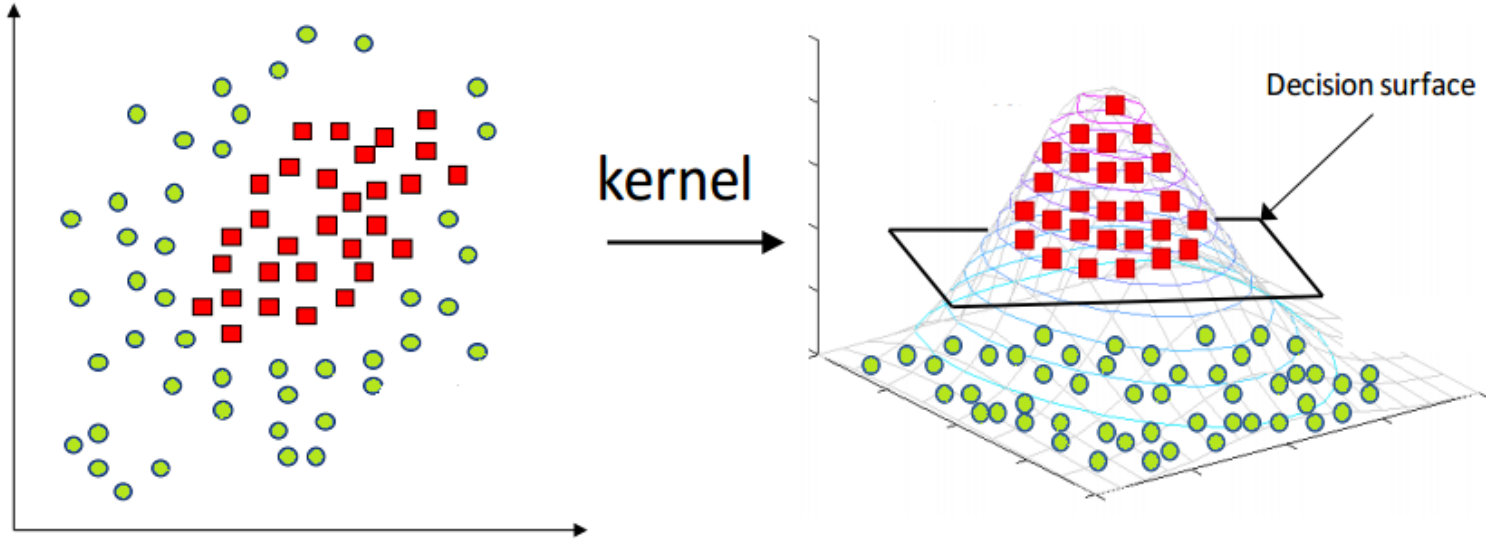
c.f. **Explanatory** models vs. **Predictive** models

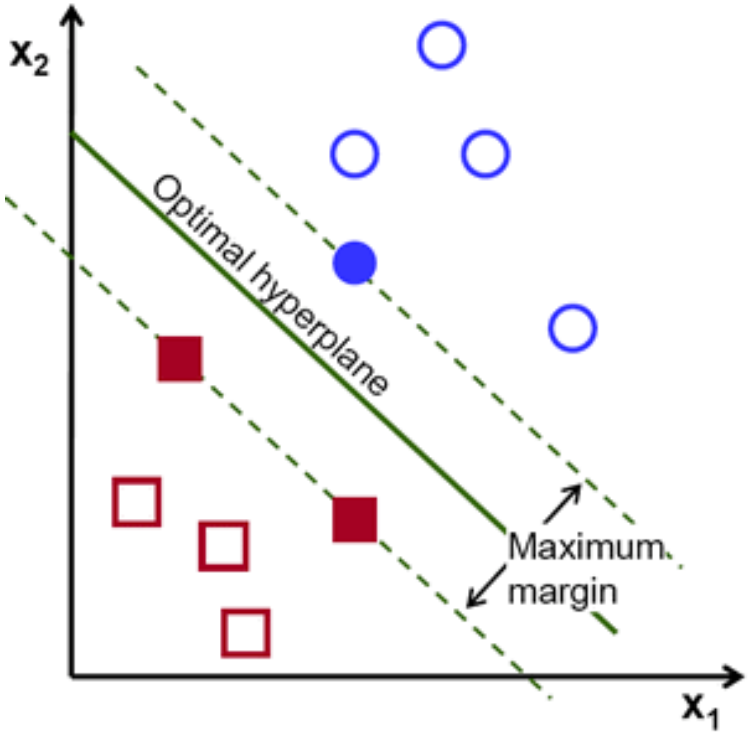
Under-parameterized: Optimization under a *fixed* model structure

Over-parameterized: Optimization by finding the *best/simplest* model structure

Revisiting Support Vector Machine

Using kernel trick + maximum margin to set boundary:

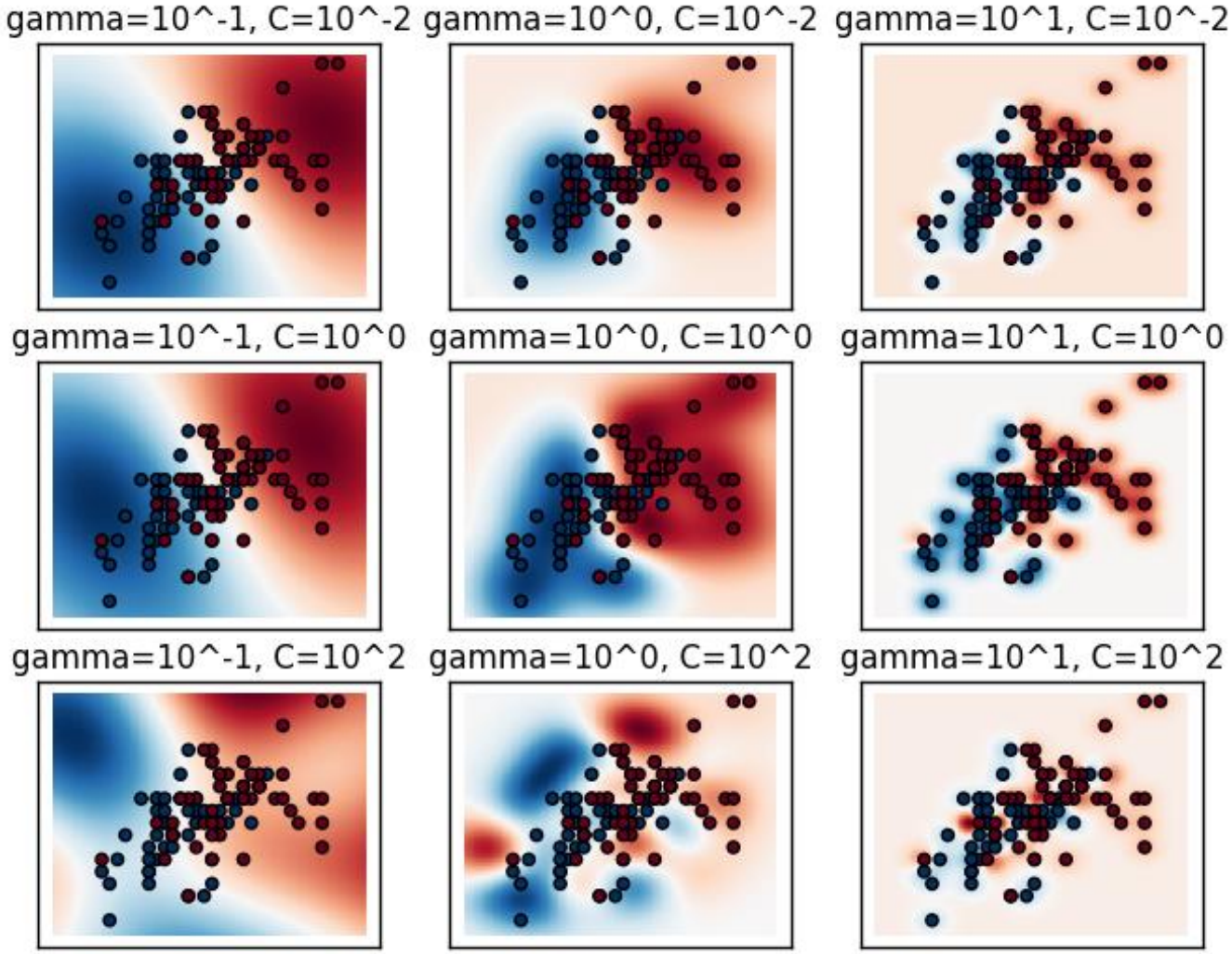





```
clf=svm.SVC(kernel='rbf', C=1.0, gamma='scale')
clf.fit(X,Y) #training
print(np.mean(clf.predict(X)==Y)) #testing
```

Tuning a RBF SVM by Grid Search

Systematically cross-validate SVM on a grid of (C, γ)



$$exp(-\gamma \left|\left| x - x^{'} \right|\right|^2), \gamma > 0$$

Support vector machines (SVM) (Boser et al., 1992) are useful for data classification. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l,\ \boldsymbol{x}_i \in R^n,\ y_i \in \{-1, +1\}$, SVM requires the solution of the following unconstrained optimization problem:

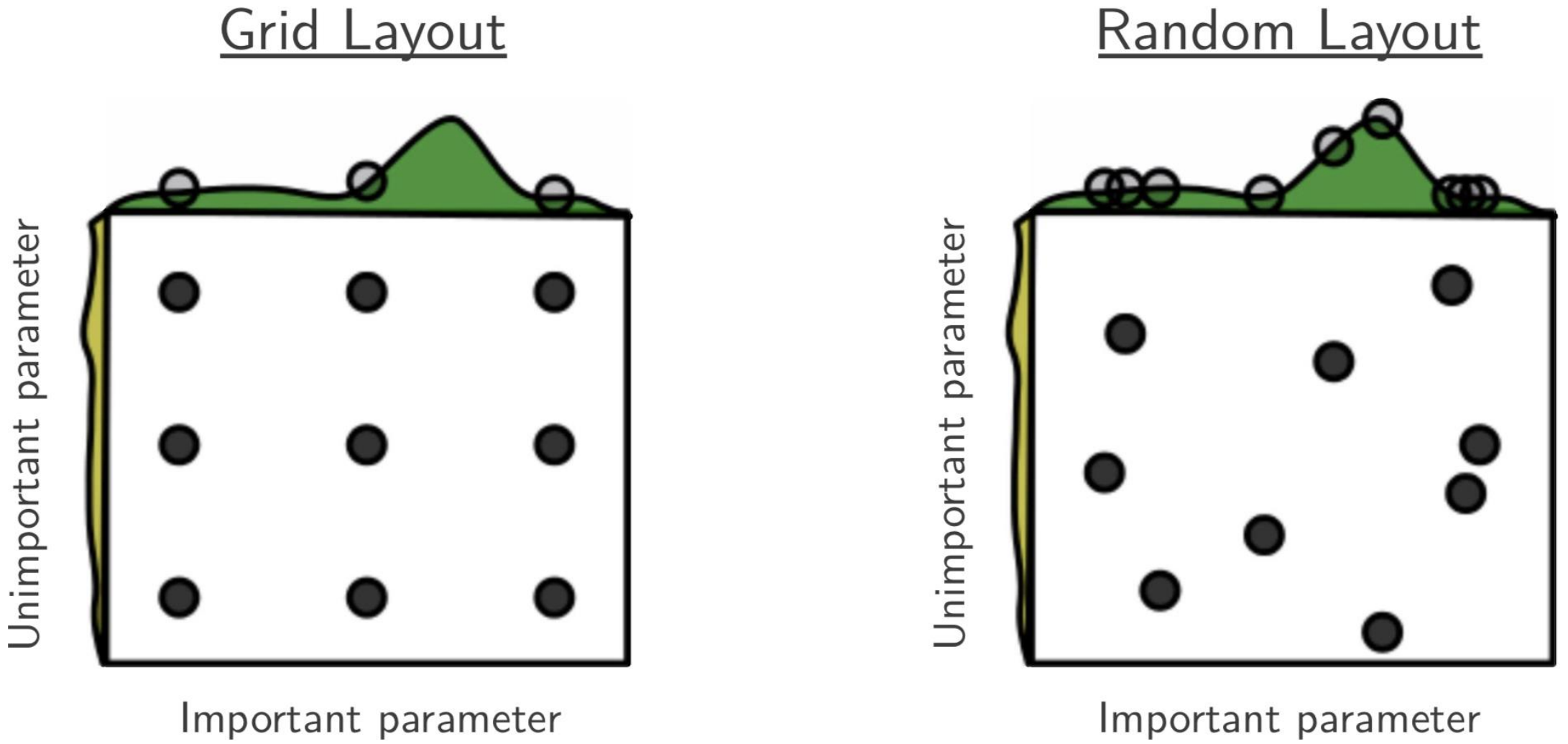
$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \qquad (1)$$

where $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$ is a loss function, and $C \geq 0$ is a penalty parameter. Two common loss functions are:

$$\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0) \text{ and } \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2. \qquad (2)$$

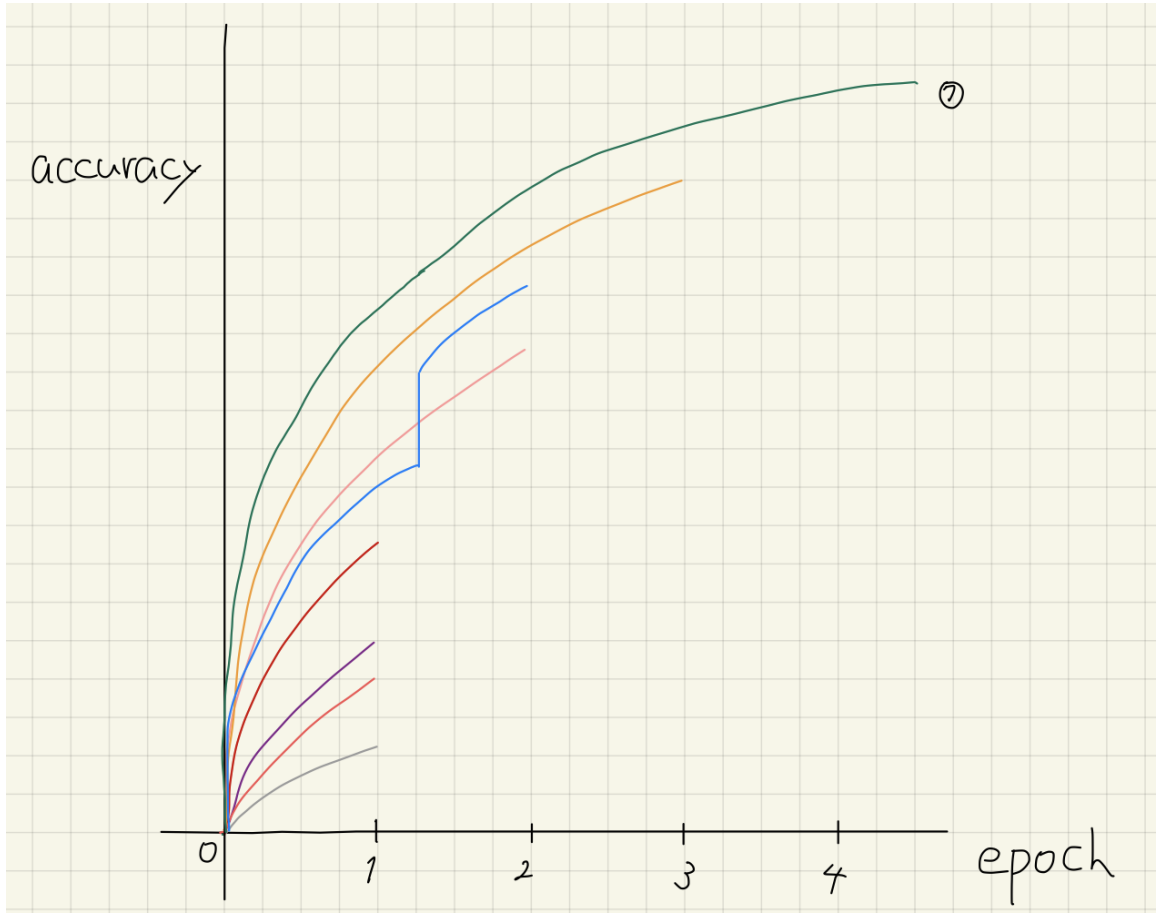
Performance Tuning by Random Search

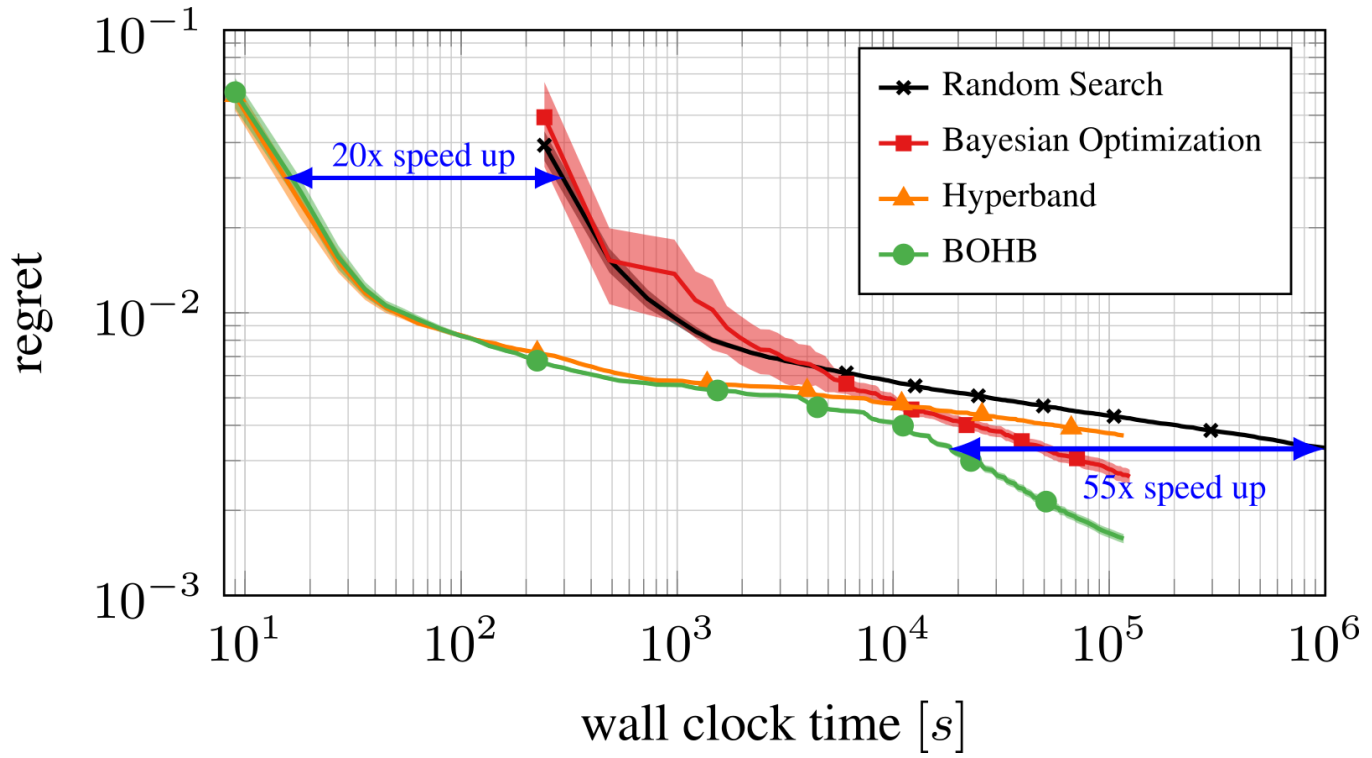
Rand search is actually more efficient than grid search



Performance Tuning by HyperBand

Bandit-based resource/time allocation
in hyperparameter optimization:





Goals for today

Model selection criteria

Which model is better?

Hyperparameter tuning

The ones that specify model complexity

Hybrid/mixed learning

Machines' crowd intelligence

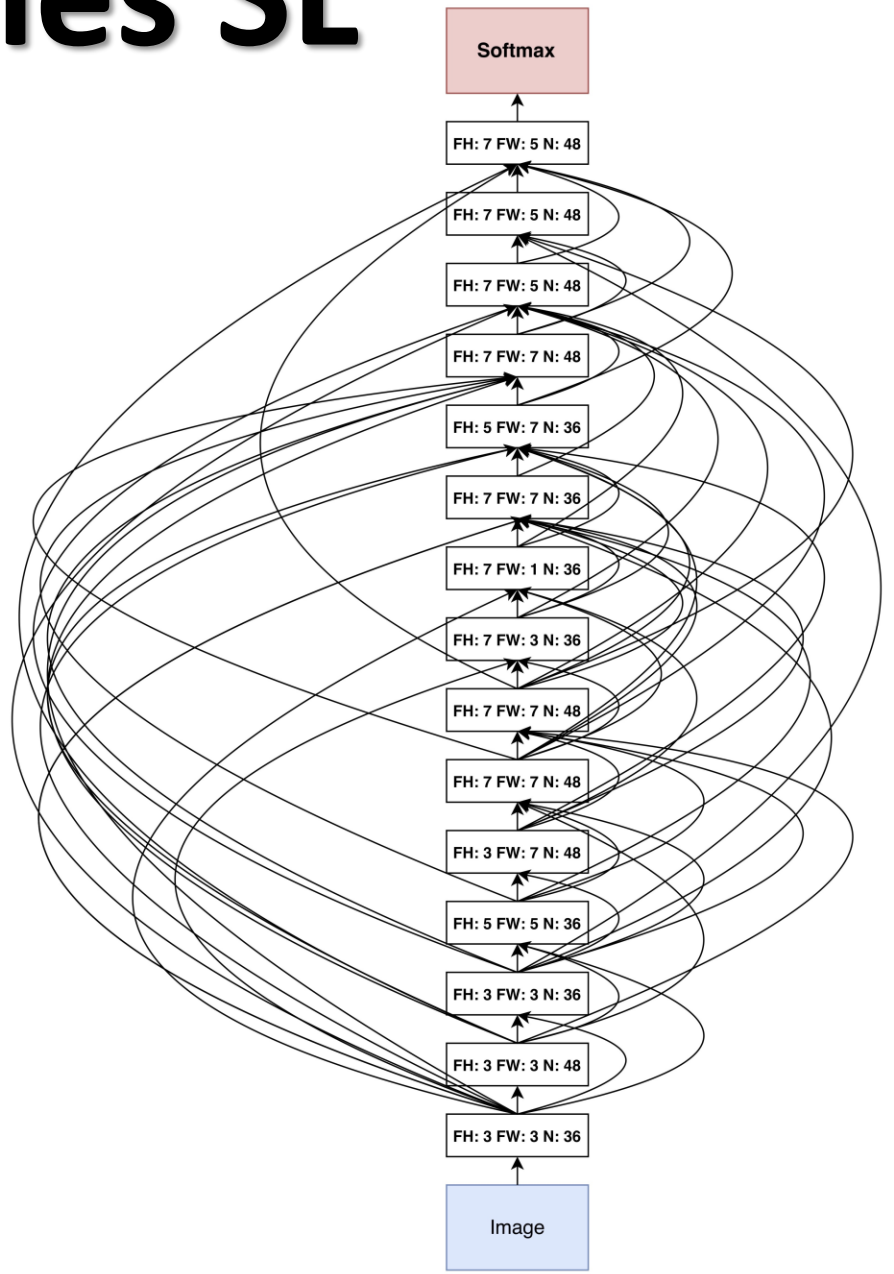
Hybrid Example 1: RL tunes SL

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

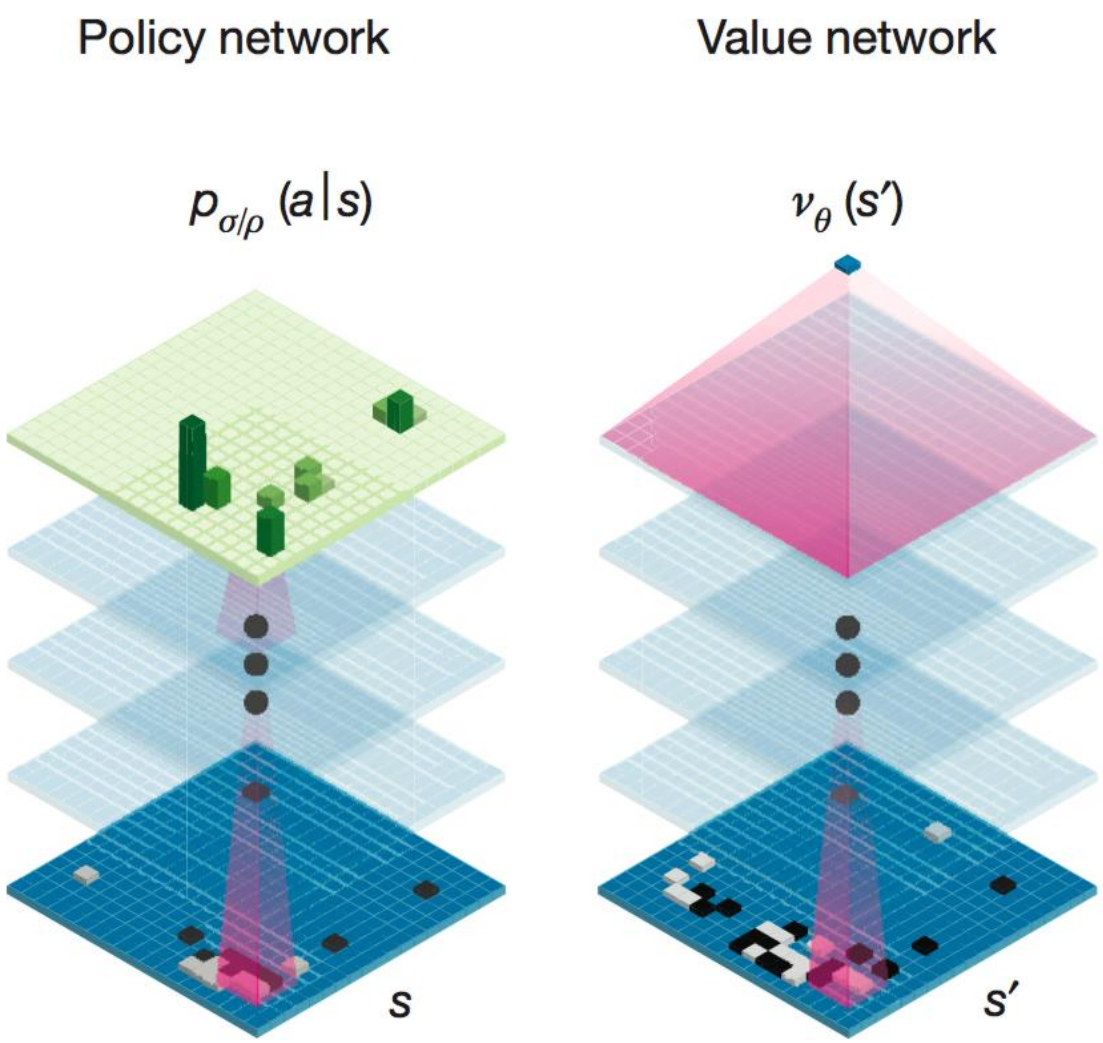
Barret Zoph,* **Quoc V. Le**
Google Brain
{barretzoph,qvl}@google.com

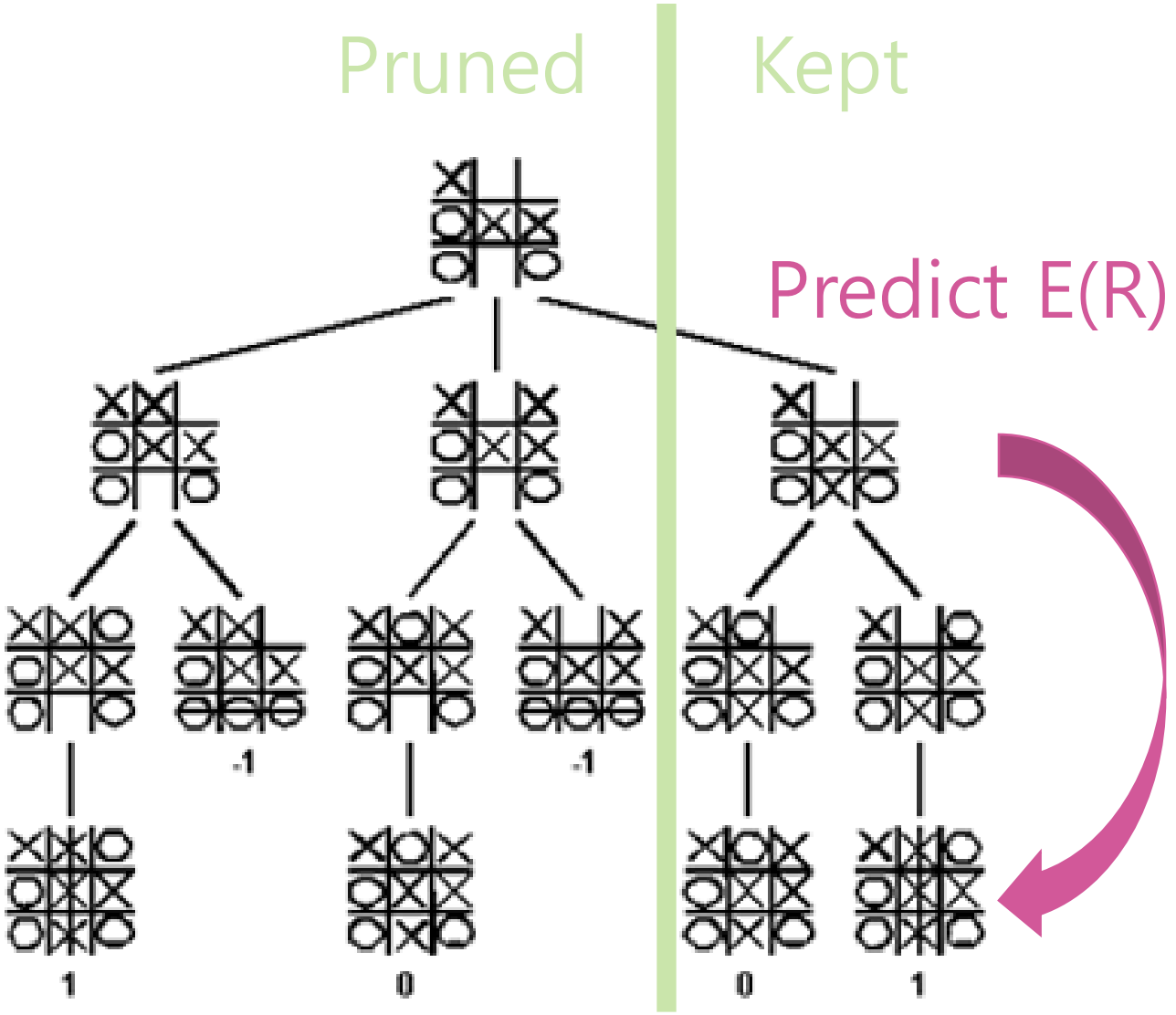
ABSTRACT

Neural networks are powerful and flexible models that work well for many difficult learning tasks in image, speech and natural language understanding. Despite their success, neural networks are still hard to design. In this paper, we use a recurrent network to generate the model descriptions of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. On the CIFAR-10 dataset, our method, starting from scratch, can design a novel network architecture that rivals the best human-invented architecture in terms of test set accuracy. Our CIFAR-10 model achieves a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous state-of-the-art model that used a similar architectural scheme. On the Penn Treebank dataset, our model can compose a novel recurrent cell that outperforms the widely-used LSTM cell, and other state-of-the-art baselines. Our cell achieves a test set perplexity of 62.4 on the Penn Treebank, which is 3.6 perplexity better than the previous state-of-the-art model. The cell can also be transferred to the character language modeling task on PTB and achieves a state-of-the-art perplexity of 1.214.



Hybrid Example 2: RL+SL





Ensemble Learning

Using a collection of models to improve performance

Common Types of Ensemble Methods

Bagging

- Reduces variance and increases accuracy
- Robust against outliers or noisy data
- Often used with Decision Trees (i.e. Random Forest)

Boosting

- Also reduces variance and increases accuracy
- Not robust against outliers or noisy data
- Flexible – can be used with any loss function

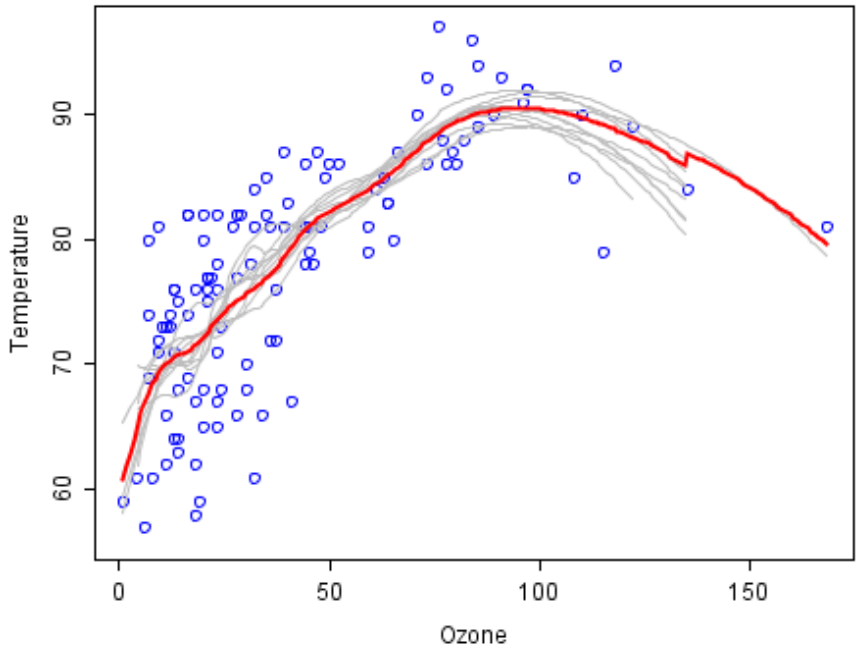
Stacking

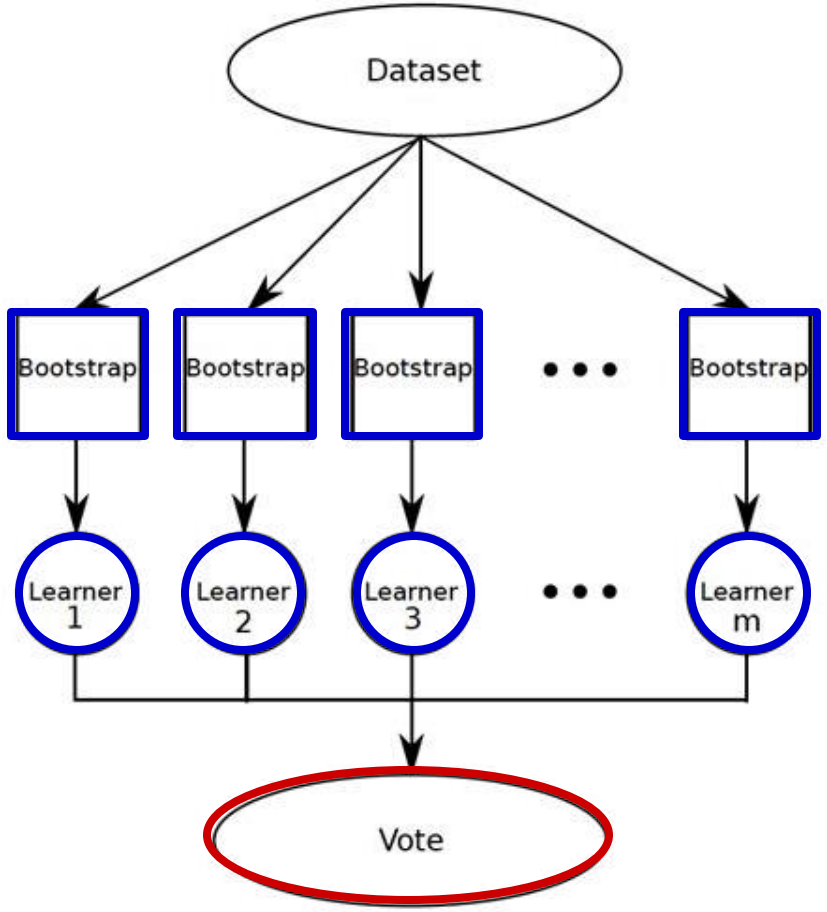
- Used to ensemble a diverse group of strong learners
- Involves training a second-level machine learning algorithm called a "metalearner" to learn the optimal combination of the base learners

Bagging: Bootstrap Aggregating

Applying the same model on different samples of data

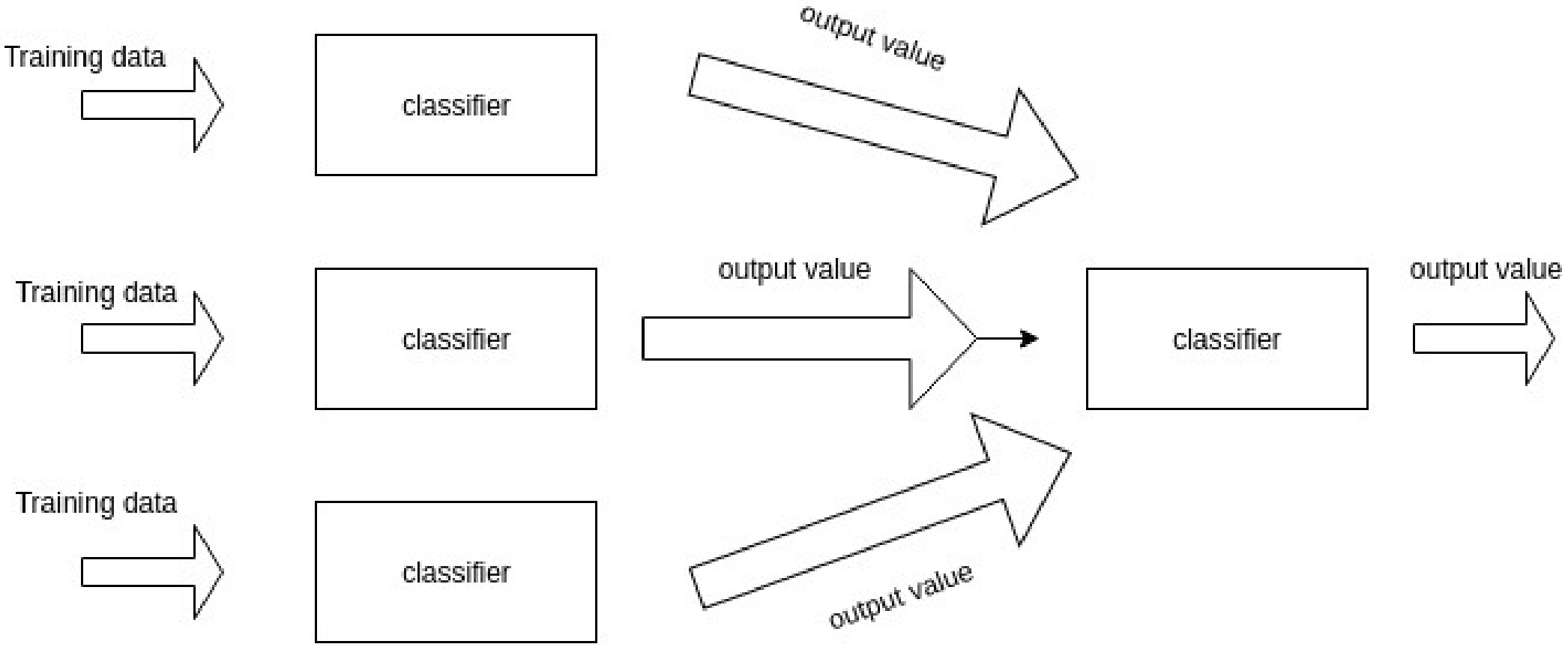
Different decisions due to different life experiences:





Stacking: Stacking Supervised Learners

Hierarchically using a master learner as an arbitrator



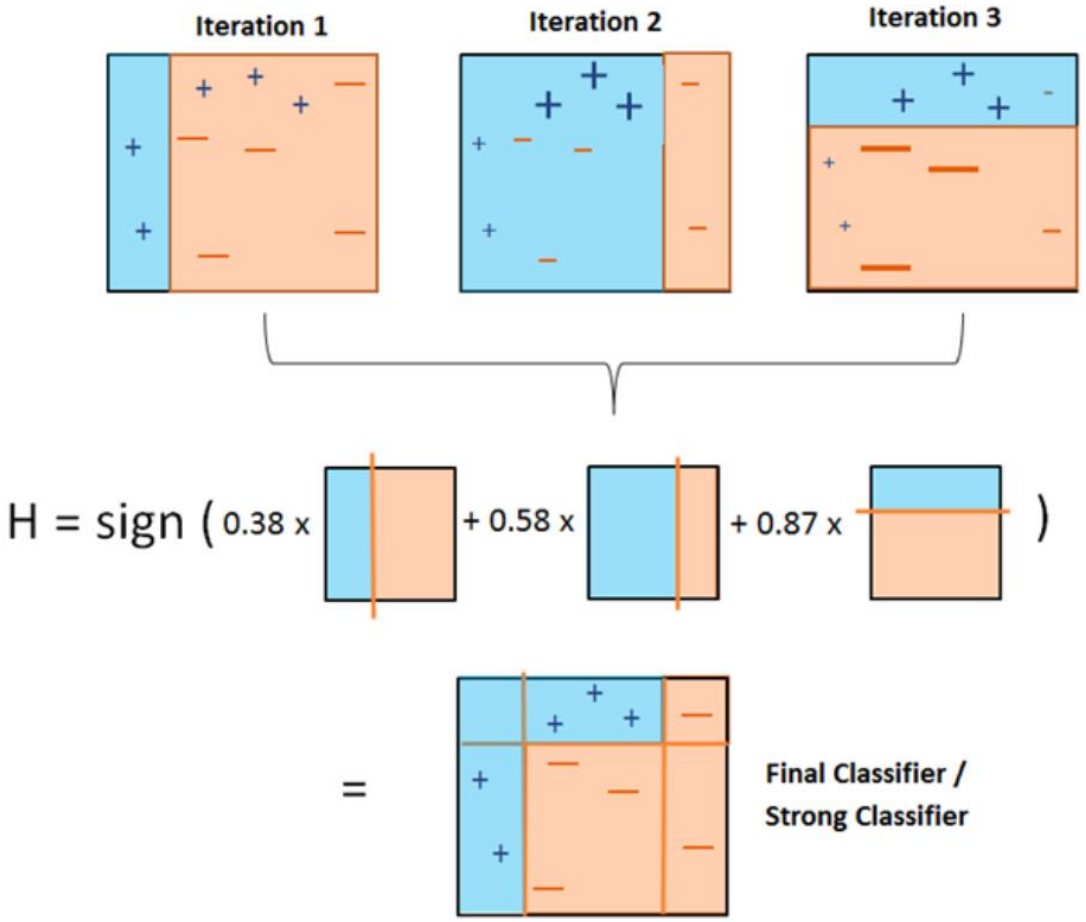
Boosting (1/3): Sequential Enhancement

AdaBoost is a special case of GradientBoost

AdaBoost	GradientBoost
Both AdaBoost and Gradient Boost use a base weak learner and they try to boost the performance of a weak learner by iteratively shifting the focus towards problematic observations that were difficult to predict. At the end, a strong learner is formed by addition (or weighted addition) of the weak learners.	
In AdaBoost, shift is done by up-weighting observations that were misclassified before.	Gradient boost identifies difficult observations by large residuals computed in the previous iterations.
In AdaBoost "shortcomings" are identified by high-weight data points.	In Gradientboost "shortcomings" are identified by gradients.
Exponential loss of AdaBoost gives more weights for those samples fitted worse.	Gradient boost further dissect error components to bring in more explanation.
AdaBoost is considered as a special case of Gradient boost in terms of loss function, in which exponential losses.	Concepts of gradients are more general in nature.

Boosting (2/3): AdaBoost

Increasing sample weights for mis-classified samples:



$$H(x) = \sum_t \rho_t \, h_t(x)$$

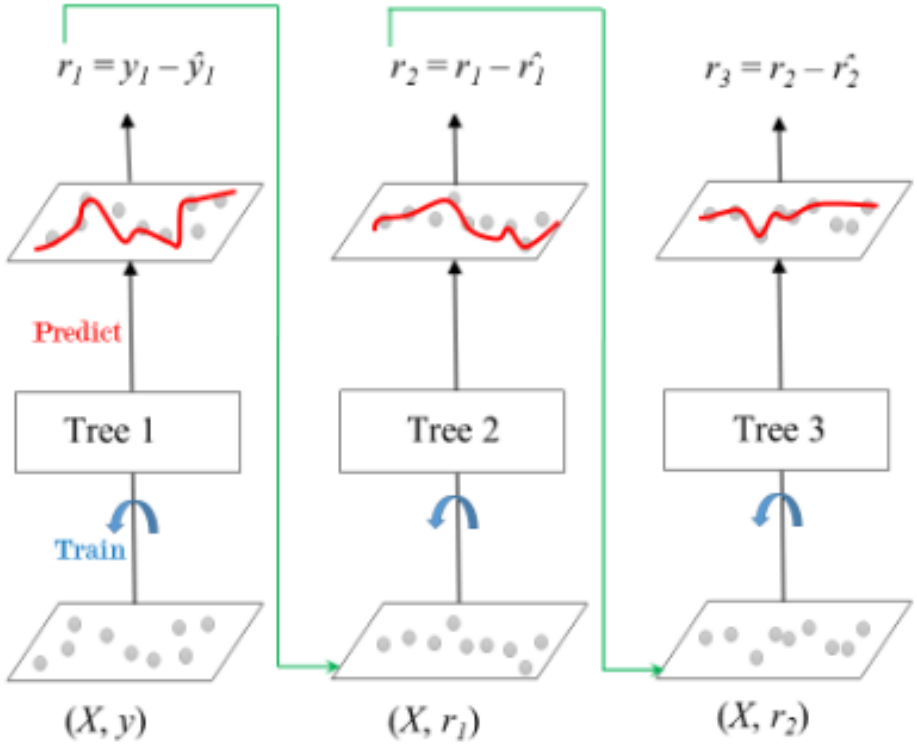
$$Amount\ of\ say = \frac{1}{2}\log\left(\frac{1-total\ error}{total\ error}\right)$$

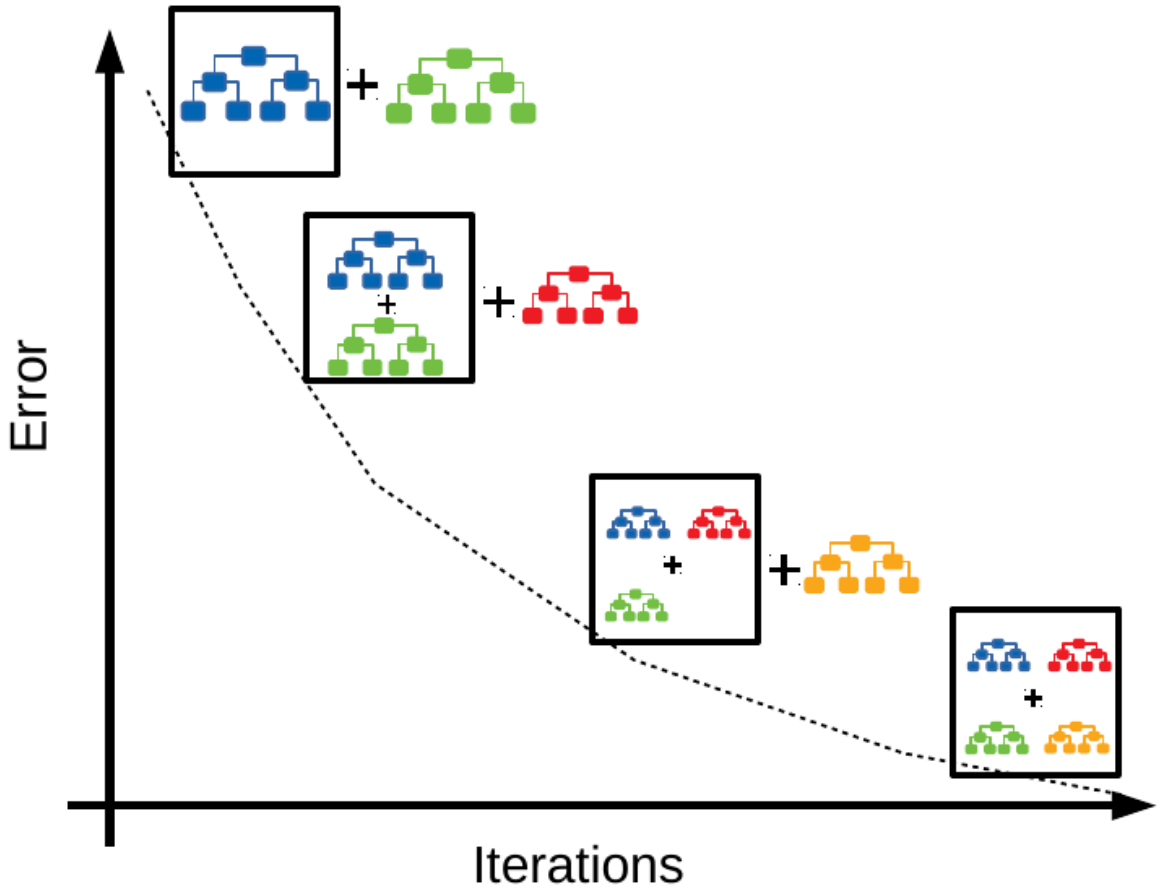
where total error
=Σ(weights of mis-classified samples)

Boosting (3/3): GradientBoost

Subsequent models focus on fitting the residuals

$$y = \hat{y}_1 + r_1 = \hat{y}_1 + \hat{y}_2 + r_2 = \hat{y}_1 + \hat{y}_2 + \hat{y}_3 + r_3$$





Goals for today

Model selection criteria

Which model is better?

Hyperparameter tuning

The ones that specify model complexity

Hybrid/mixed learning

Machines' crowd intelligence

Game Over