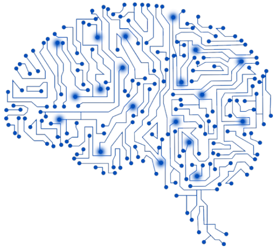
Model Performance Improvements

Vuong Thi Hai Yen, VNU-UET



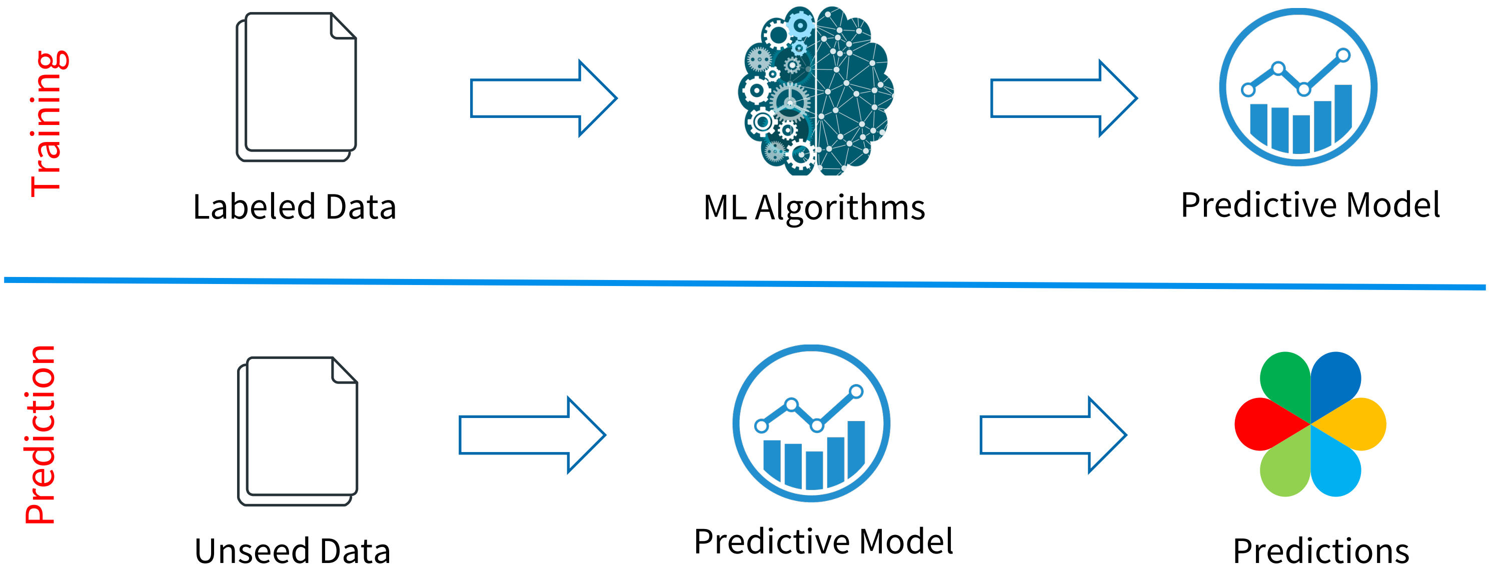
Outline

- The Problem of Model Generalization
  - Underfitting, Overfitting, Good Fit
  - Ways to Deal with Underfitting & Overfitting
- Overfitting Reduction for Deep Learning Models
  - Regularization
  - Early stopping
  - Cross validation
  - Ensembles: Bagging, Boosting, Stacking
- Improvements for Imbalanced Data Problem
  - Subsampling, Oversampling
  - Weighted Samples

1.

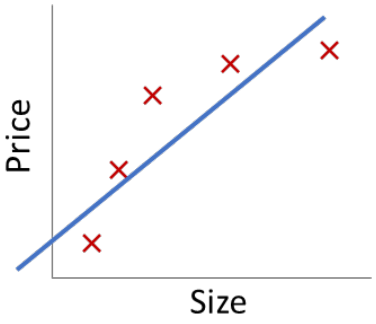
The Problem of Model Generalization

Two Phases of Model Development



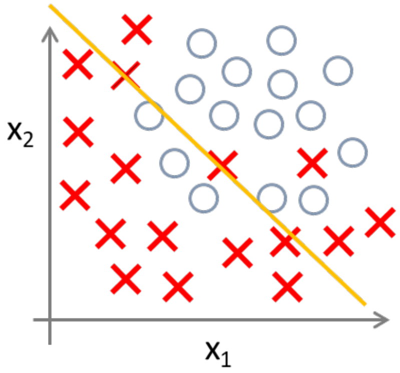
Underfitting

Regression



$$w_0 + w_1 x$$

Classification

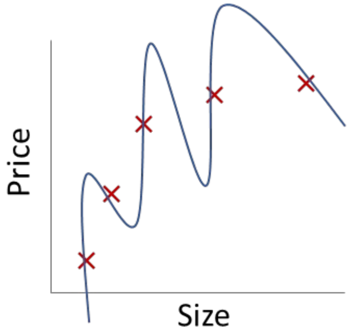


$$h(x) = g(w_0 + w_1 x_1 + w_2 x_2)$$

Underfitting: A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on holdout/validation samples.

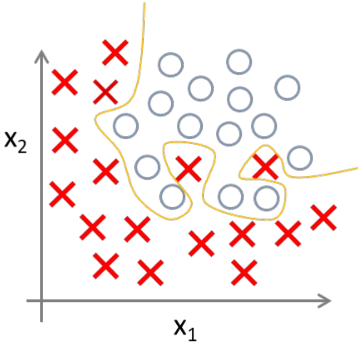
Overfitting

Regression



$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

Classification

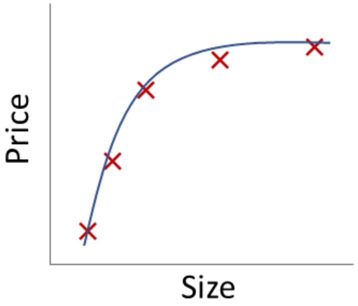


$$g(w_0 + w_1 x_1 + w_2 x_1^2 \\ + w_3 x_1^2 x_2 + w_4 x_1^2 x_2^2 \\ + w_5 x1^2 x_2^3 + w_6 x1^3 x_2 + \ldots)$$

Overfitting: A model that learns the training dataset too well, performing well on the training dataset but does not perform well on holdout samples.

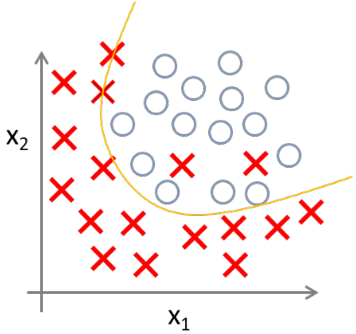
Good Fit

Regression



$$w_0 + w_1 x + w_2 x^2$$

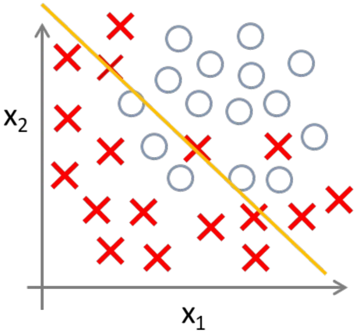
Classification



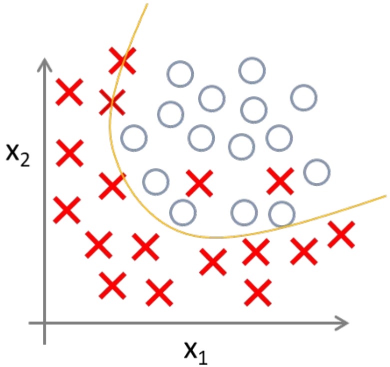
$$g(w_0 + w_1 x_1 + w_2 x_2 \\ + w_3 x_1^2 + w_4 x_2^2 \\ + w_5 x1 x_2)$$

Good Fit: A model that suitably learns the training dataset and generalizes well to the holdout dataset.

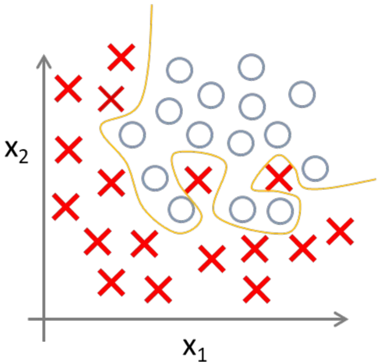
Underfitting, Overfitting & Goodfit



$$h(x) = g(w_0 + w_1x_1 + w_2x_2)$$



$$g(w_0 + w_1x_1 + w_2x_2 \\ + w_3x_1^2 + w_4x_2^2 \\ + w_5x1x_2)$$

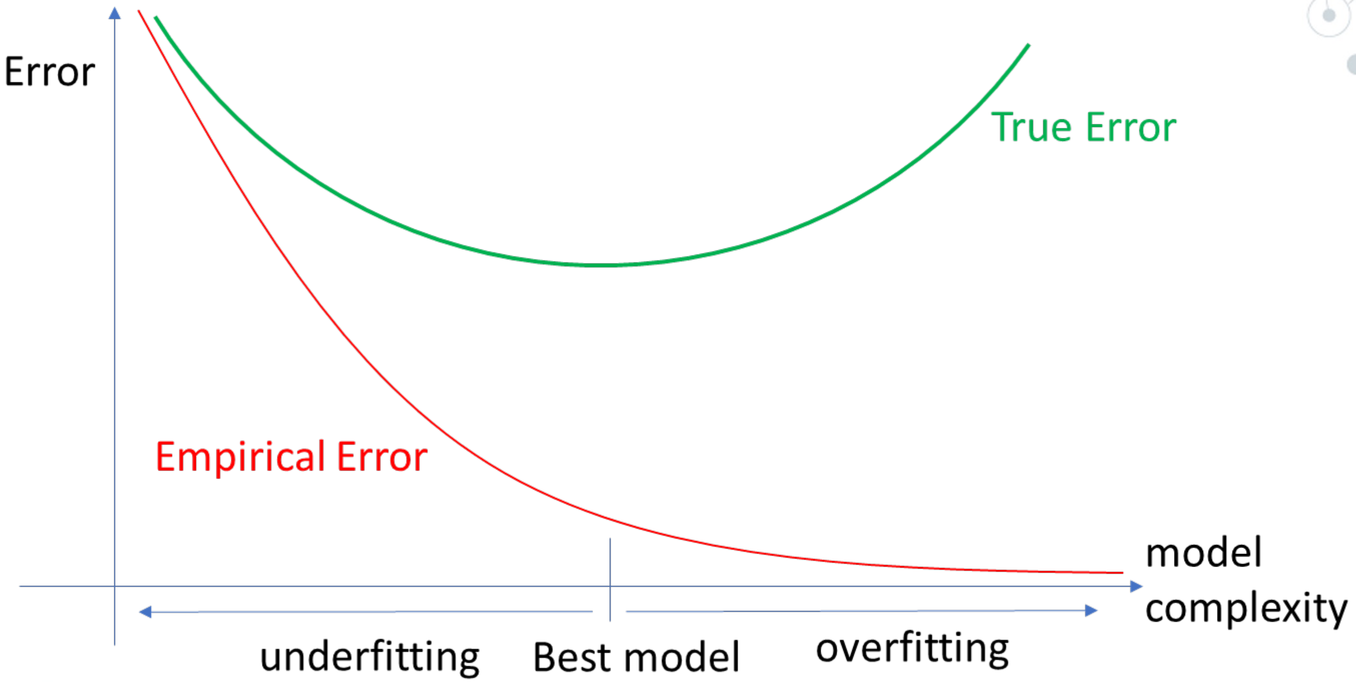


$$g(w_0 + w_1x_1 + w_2x_1^2 \\ + w_3x_1^2x_2 + w_4x_1^2x_2^2 \\ + w_5x1^2x_2^3 + w_6x1^3x_2 + \ldots)$$

"Underfitting"

"Overfitting"

The Problem of Model Generalization



*"One should **not increase**, beyond what is necessary, **the number of entities** required to **explain** anything."*

Occam's Razor

Ways to Deal with Underfitting, Overfitting

Underfitting, Overfitting Summary

Underfitting	Overfitting
• High empirical errors on both training and holdout sets. • Model is too simple • High bias • Maybe not converged yet	• Low empirical error on train but high empirical error on the holdout set. • Model is too complex • High variance • May pass the ‘gold’ converged point

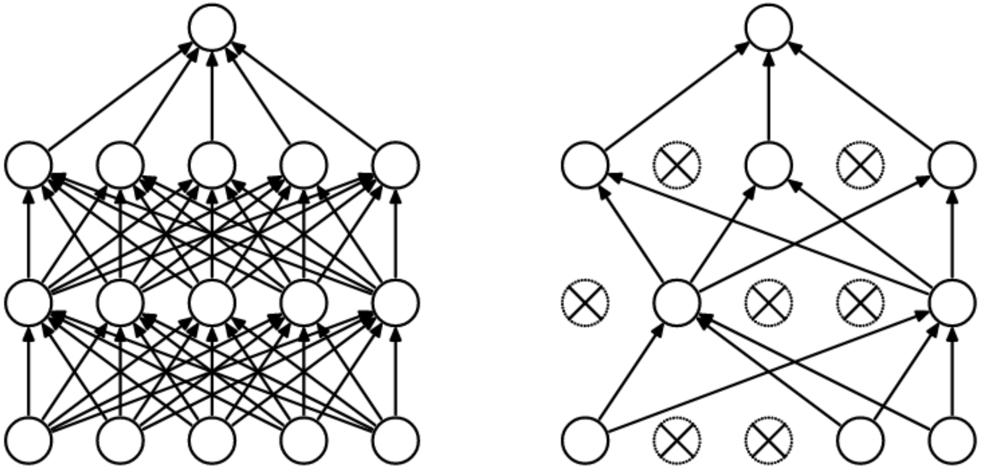
Ways to Deal with Underfitting, Overfitting

Underfitting	Overfitting
• Increase the complexity of the model by adding more features • Increase the number of the training epochs • Remove noise from data • Try more complex models	• Increase the training data size • Reduce the model complexity by feature selections • Regularization • Cross-validation • Early stopping • Ensembles

2.

Overfitting Reduction for Deep Learning Models

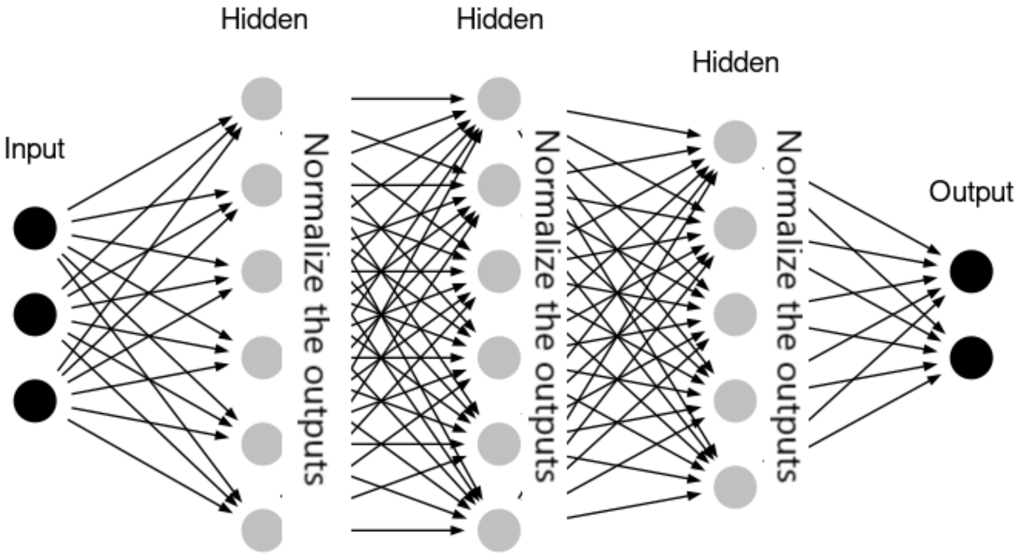
Regularization – Dropout (1)



(a) Standard Neural Net (b) After applying dropout.

Dropout is a **technique** reduce the model complexity whereby randomly selected neurons are ignored during training. They are “dropped-out” randomly with probability *p*.

Regularization – Batch Normalization (1)



Batch normalization is a **technique** that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks

Regularization – Batch Normalization (2)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

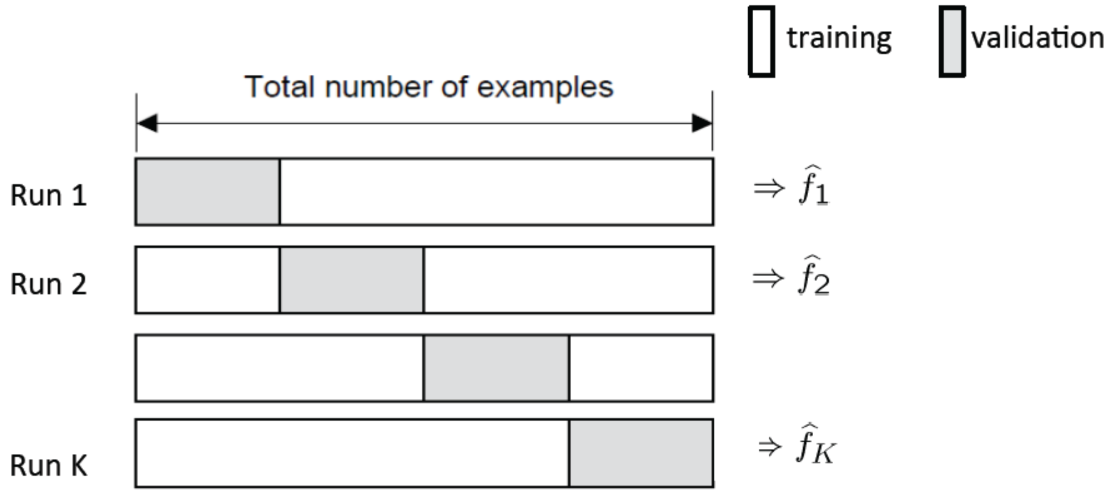
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

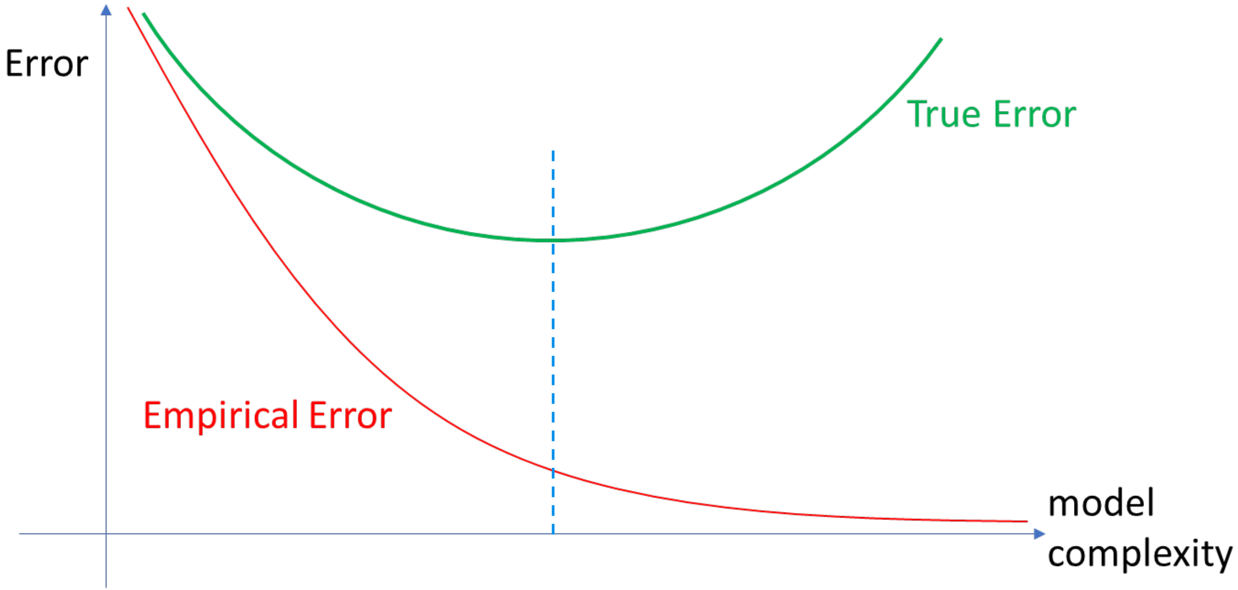
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Cross Validation



K-fold cross validation is a **technique** where randomly create K-fold partition of the dataset. Form K hold-out predictors, each time using one partition as validation and rest (K-1) as training datasets

Early Stopping

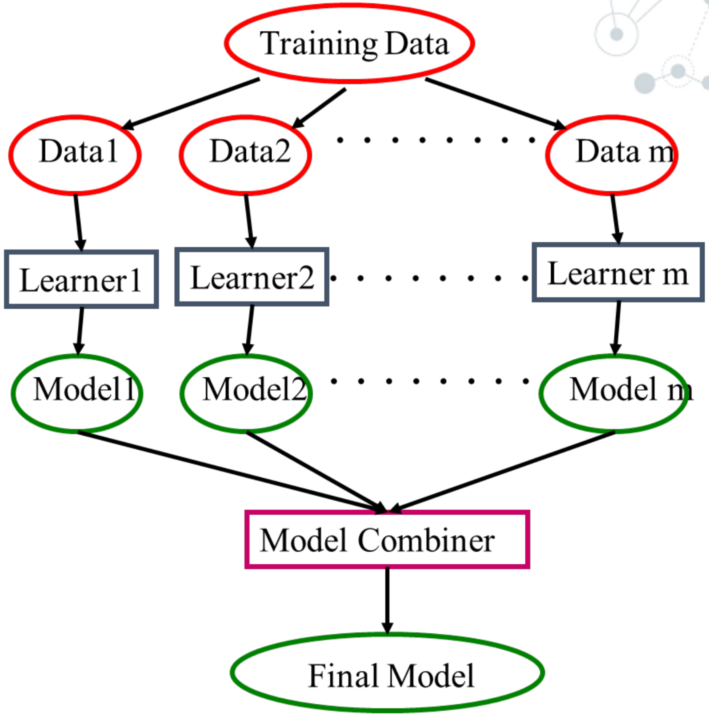


Early stopping is a **technique** where try to stop training as soon as the validation error has stopped reducing

Ensemble Models

Ensemble Learning

- **Main Idea**: Instead of learning one mode, learning several and combine them
- Typically improves the accuracy, often by a lot

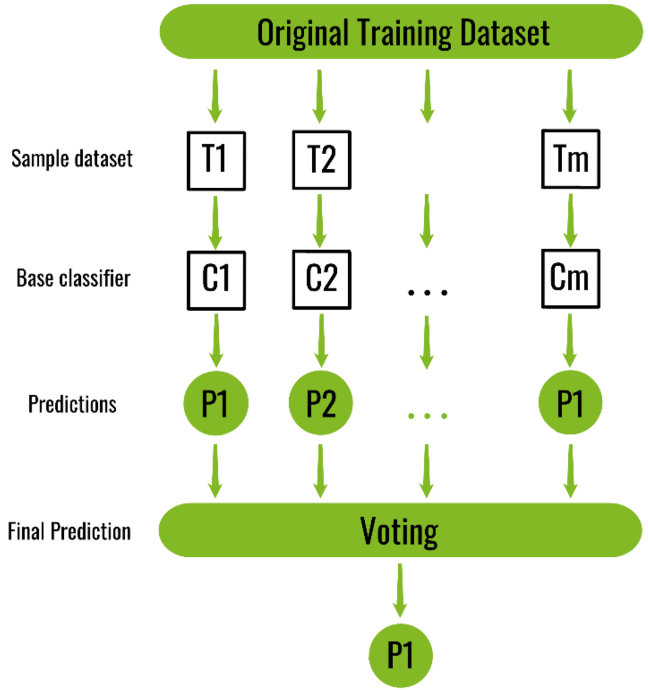


Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction (i.e., *13 out of the 25 classifiers misclassified*):

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Bagging



Main Idea: Model generalization via sampling training examples

Bagging Algorithm

Create — Create **k** training samples **D1, D2,…, Dk**

Train — Train distinct classifier **h_i** on each **D_i**

Classify — Classify new instance by classifier vote with equal weights

$$c^*(\mathbf{x}) = \arg\max_c \sum_{i=1}^{k} p(c|h_i, \mathbf{x})$$

The Problem of Bagging

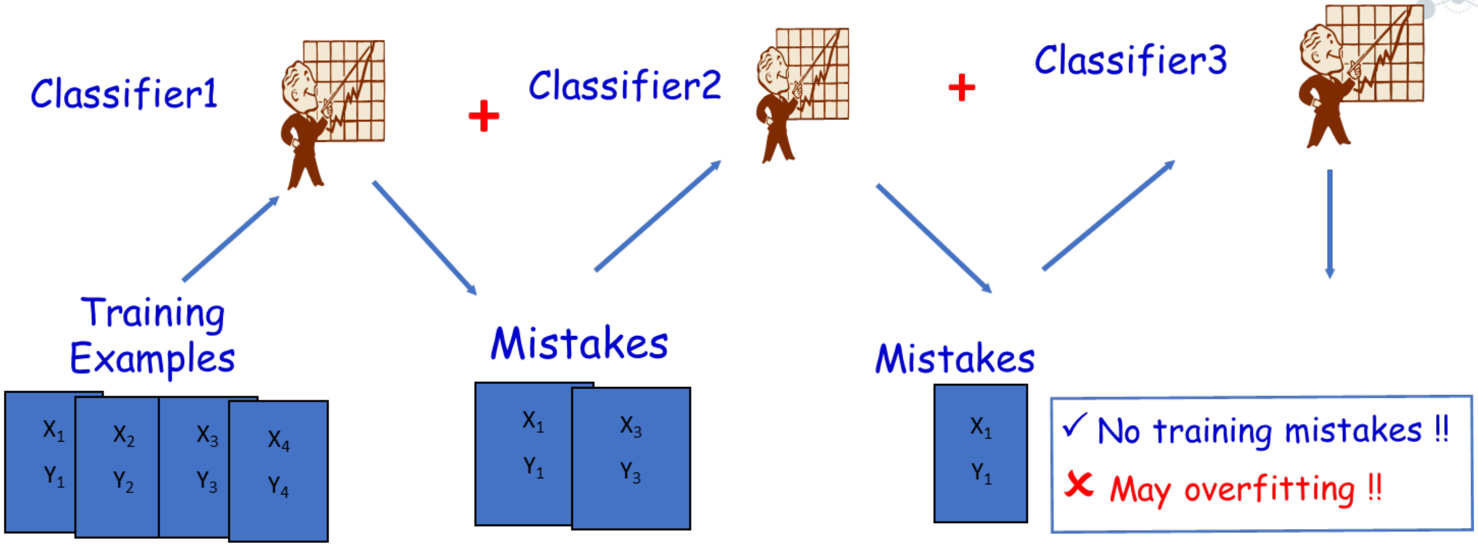
- **Inefficient sampling:**
  - Every example has equal chance to be sampled
  - No distinction between “easy” examples and “difficult” examples

- **Inefficient model combination**
  - A constant weight for each classifier
  - No distinction between accurate classifiers and inaccurate classifiers

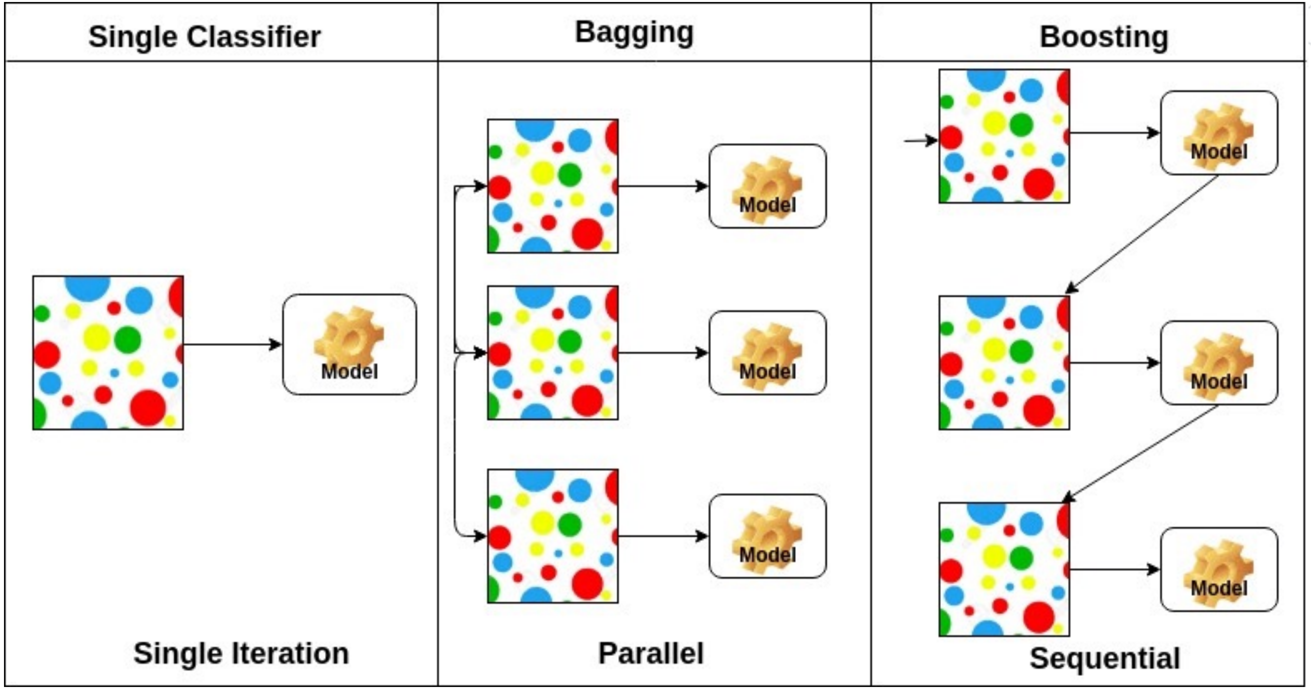
Improve the Efficiency of Bagging

- **Better sampling strategy:**
  - Focus on the examples that are difficult to classify
- **Better combination strategy**
  - Accurate model should be assigned larger weights

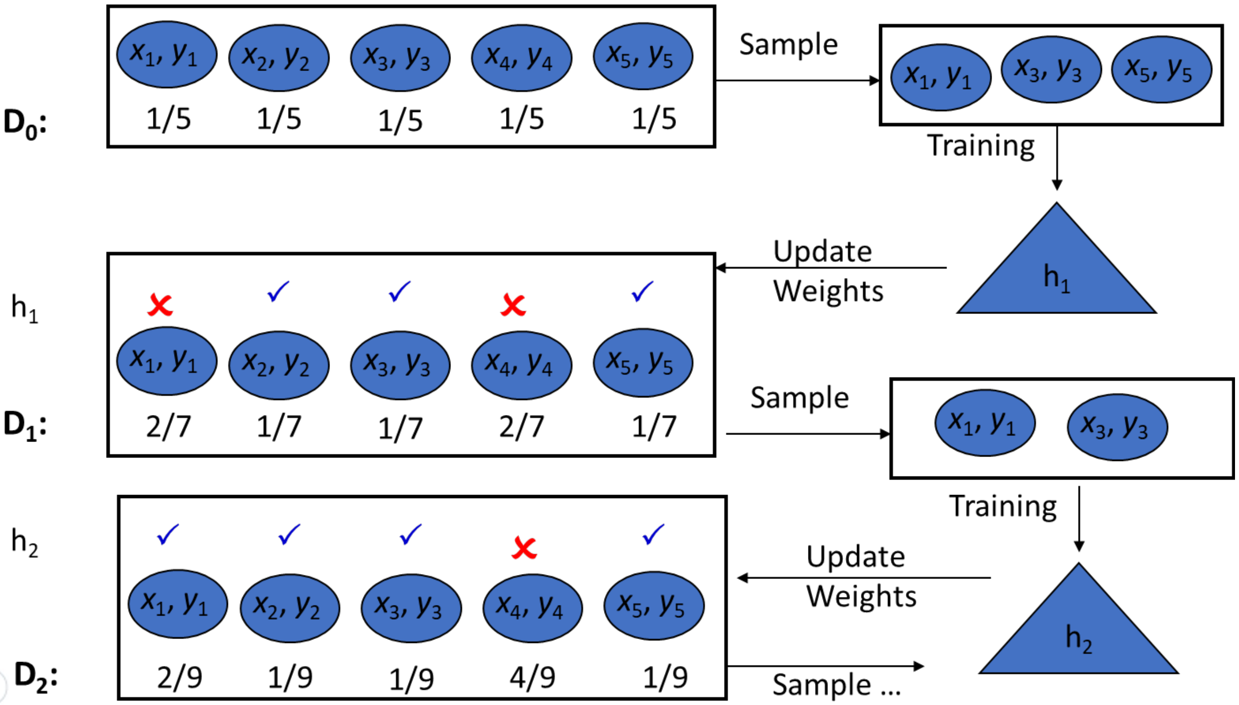
Boosting: Intuition



Bagging vs Boosting

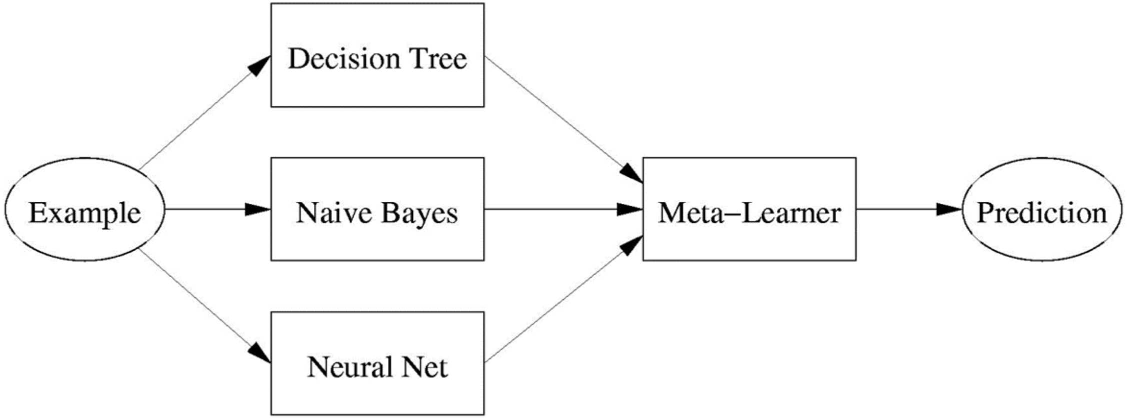


AdaBoost Example



Stacking

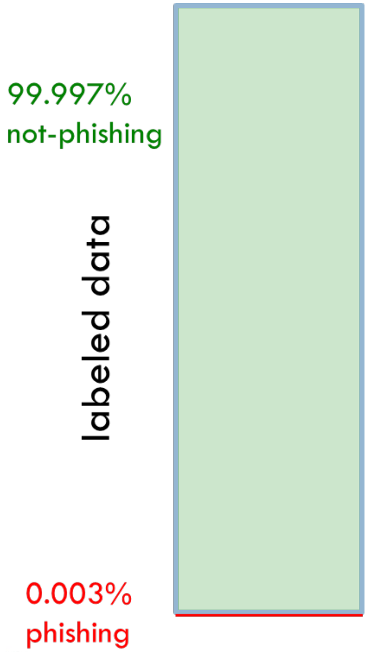
- Apply multiple base learners (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation: Meta-L. inputs = Predictions on left-out examples



3.

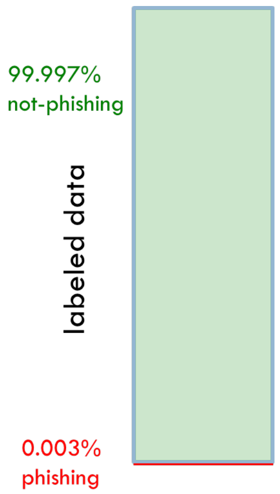
Improvements for Imbalanced Data Problem

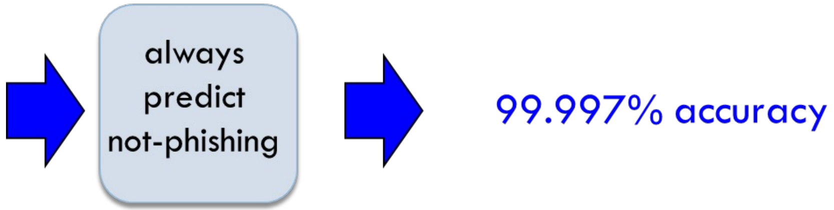
Imbalanced Data



- **Dataset:** 1M emails collected randomly
- **Problem**: phishing detection problem
- **Imbalance**: 99.997% emails labeled as "not-phishing" while 0.003% as "phishing"

Imbalanced Data Problem





Is it too simple? Do we have an ideal classifier?

Imbalanced Problem Domains

- Medical diagnosis
- Predicting faults/failures (e.g., hard-drive failures, mechanical failures, etc.)
- Predicting rare events (e.g., earthquakes)
- Detecting fraud (credit card transactions, internet traffic)

Why does the Problem happen?

Imbalance

A large discrepancy between the number of examples for class labels with different importances.

Evaluation

Accuracy is not right measure of classifier performance for the imbalanced problem.

Imbalanced Problem Identification

How to Identify the Problem

- View the task as trying to find/identify “positive” examples (i.e., the rare events)

Precision: proportion of test examples *predicted* as positive that are correct

$$\frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

Recall: proportion of test examples *labeled* as positive that are correct

$$\frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

Precision and Recall

data	label	predicted
	0	0
	0	1
	1	0
	1	1
	0	1
	1	1
	0	0

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

$$\text{precision} = \frac{2}{4}$$

$$\text{recall} = \frac{2}{3}$$

Maximizing Precision

data	label	predicted
	0	0
	0	0
	1	0
	1	0
	0	0
	1	0
	0	0

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

Don’t predict anything as positive!

Maximizing Recall

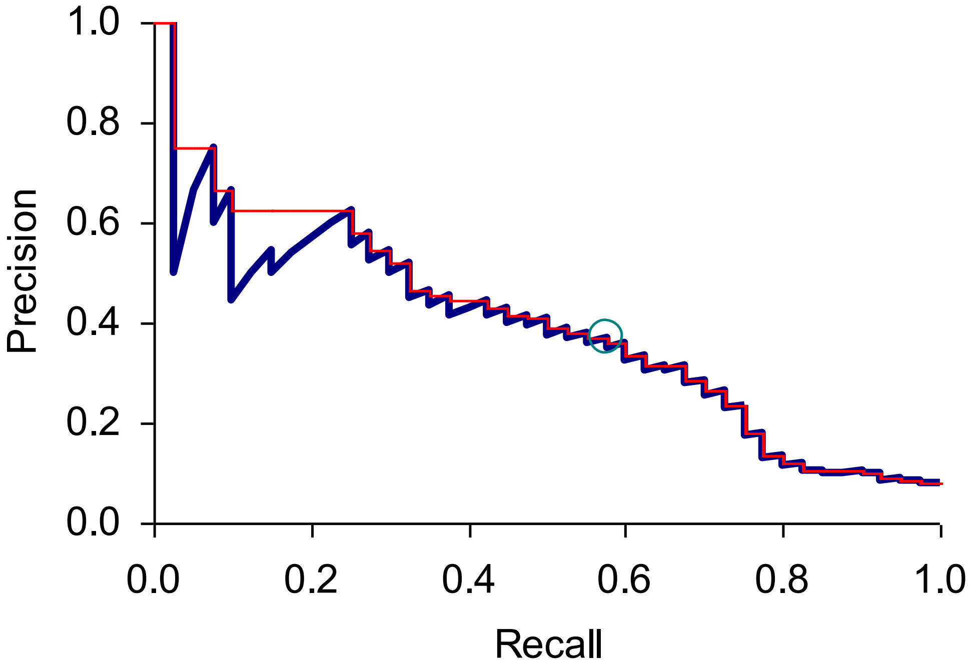
data	label	predicted
	0	1
	0	1
	1	1
	1	1
	0	1
	1	1
	0	1

$$\text{precision} = \frac{\text{\# correctly predicted as positive}}{\text{\# examples predicted as positive}}$$

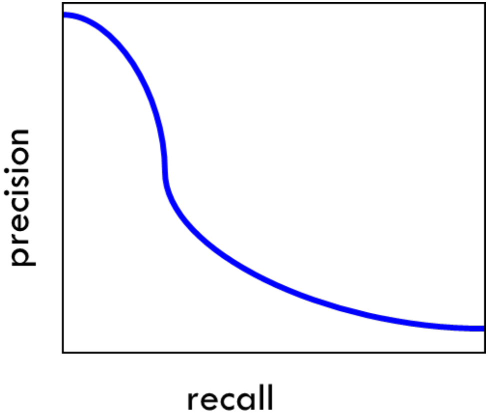
$$\text{recall} = \frac{\text{\# correctly predicted as positive}}{\text{\# positive examples in test set}}$$

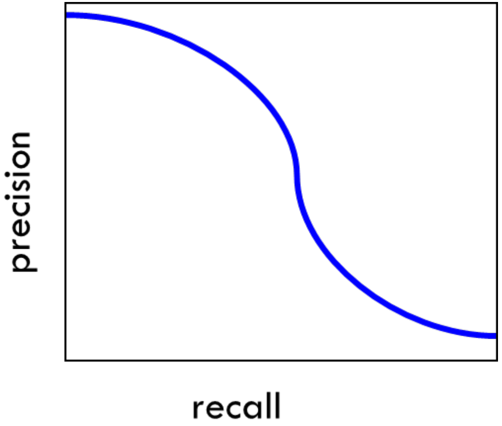
Predict everything as positive!

Precision Recall Trade-off (1)



Precision Recall Trade-off (2)





Which one is better?

A Combined Measure: F

Combined measure that assesses precision/recall tradeoff is **F measure** (weighted harmonic mean):

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

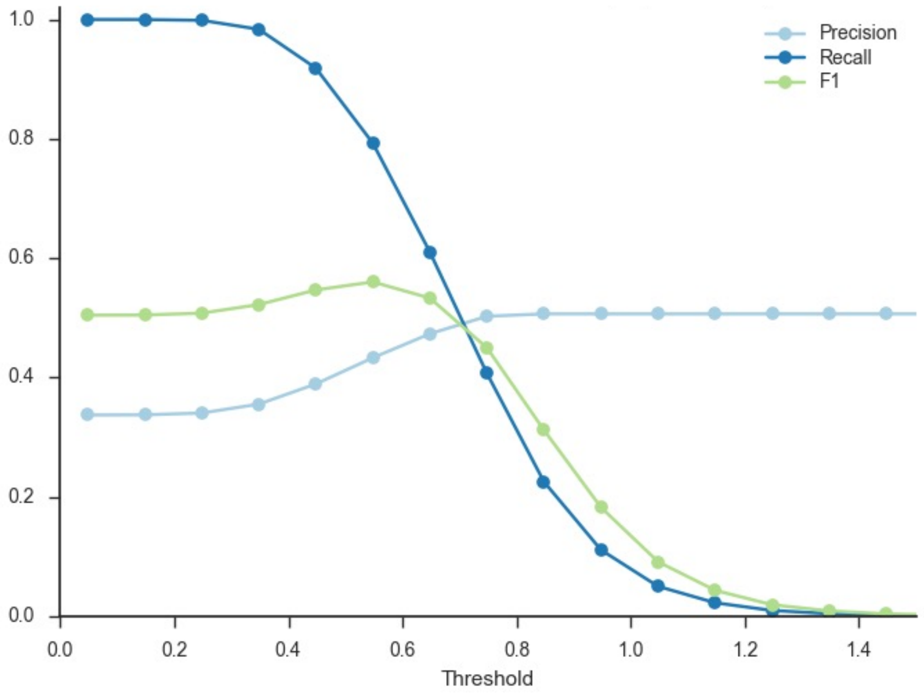
F1-Measure

Most common **α=0.5**: equal importance between precision and recall:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

$$F1 = \frac{1}{0.5\frac{1}{P} + 0.5\frac{1}{R}} = \frac{2PR}{P+R}$$

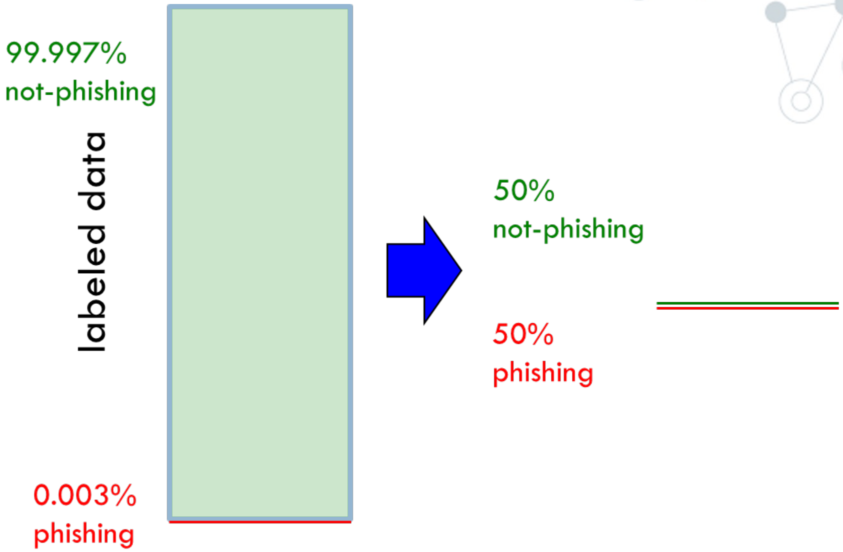
F1-Measure Visualization



Imbalanced Problem Handling

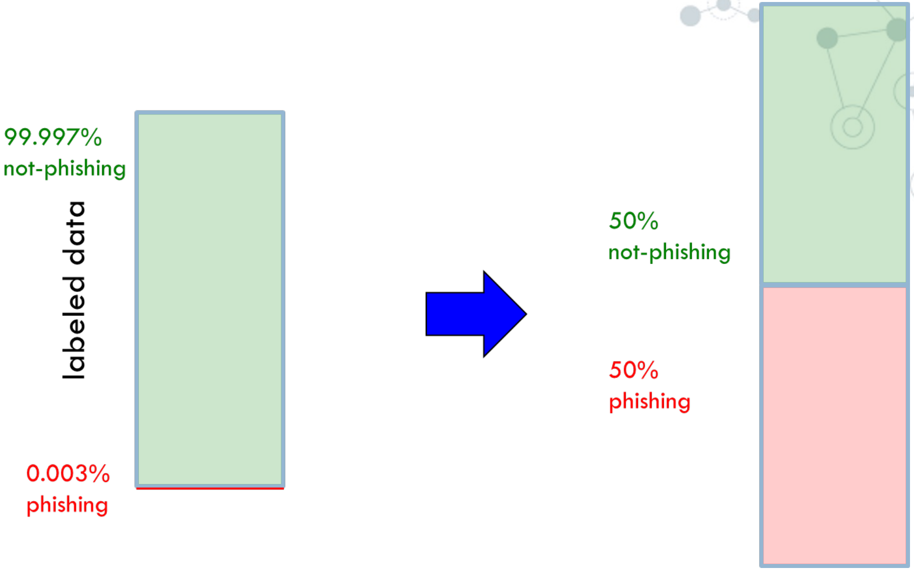
Solution 1: Subsampling

- **Main idea**: Create a new training data by:
  - including all *k* “positive” examples
  - randomly picking *k* “negative” examples
- **Pros**:
  - Easy to implement
  - Smaller training set
- **Cons**:
  - Data/information lost

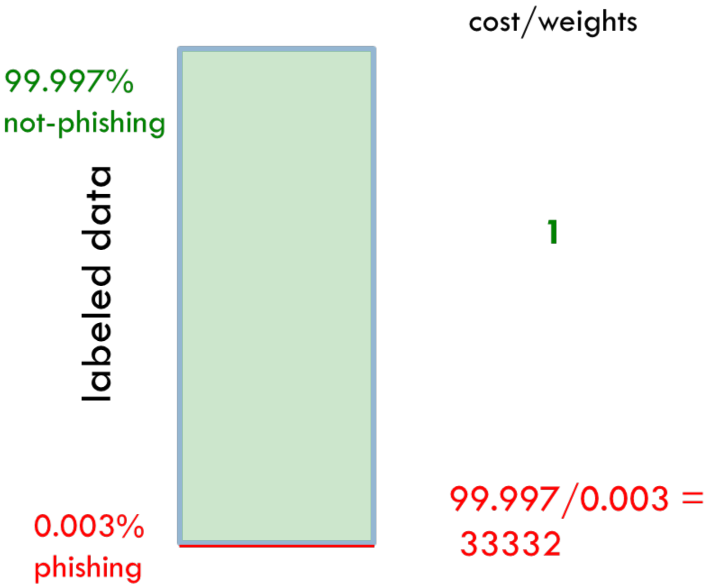


Solution 2: Oversampling

- **Main idea**: Create a new training data by:
  - including all *m* "*negative*" examples
  - sample *m* "*positive*" examples by:
    - Repeating
    - Sample with replacement
- **Pros**:
  - Easy to implement
  - Utilizes all of the training data
  - Better subsampling
- **Cons**:
  - Computational cost



Solution 3: Weighted examples



- **Main idea**: Assign different "important" weights to "positive" and "negative" labels in the training phase.
- **Pros**:
  - Good as oversampling
  - Utilizes all of the training data
- **Cons**:
  - Weighting strategy

```
tf.nn.weighted_cross_entropy_with_logits(
    labels, logits, pos_weight, name=None
)
```

Summary

- The Problem of Model Generalization
  - Underfitting, Overfitting, Good Fit
  - Ways to Deal with Underfitting & Overfitting
- Overfitting Reduction for Deep Learning Models
  - Regularization
  - Early stopping
  - Cross validation
  - Ensembles: Bagging, Boosting, Stacking
- Improvements for Imbalanced Data Problem
  - Subsampling, Oversampling
  - Weighted Samples

Thanks!

Any questions?