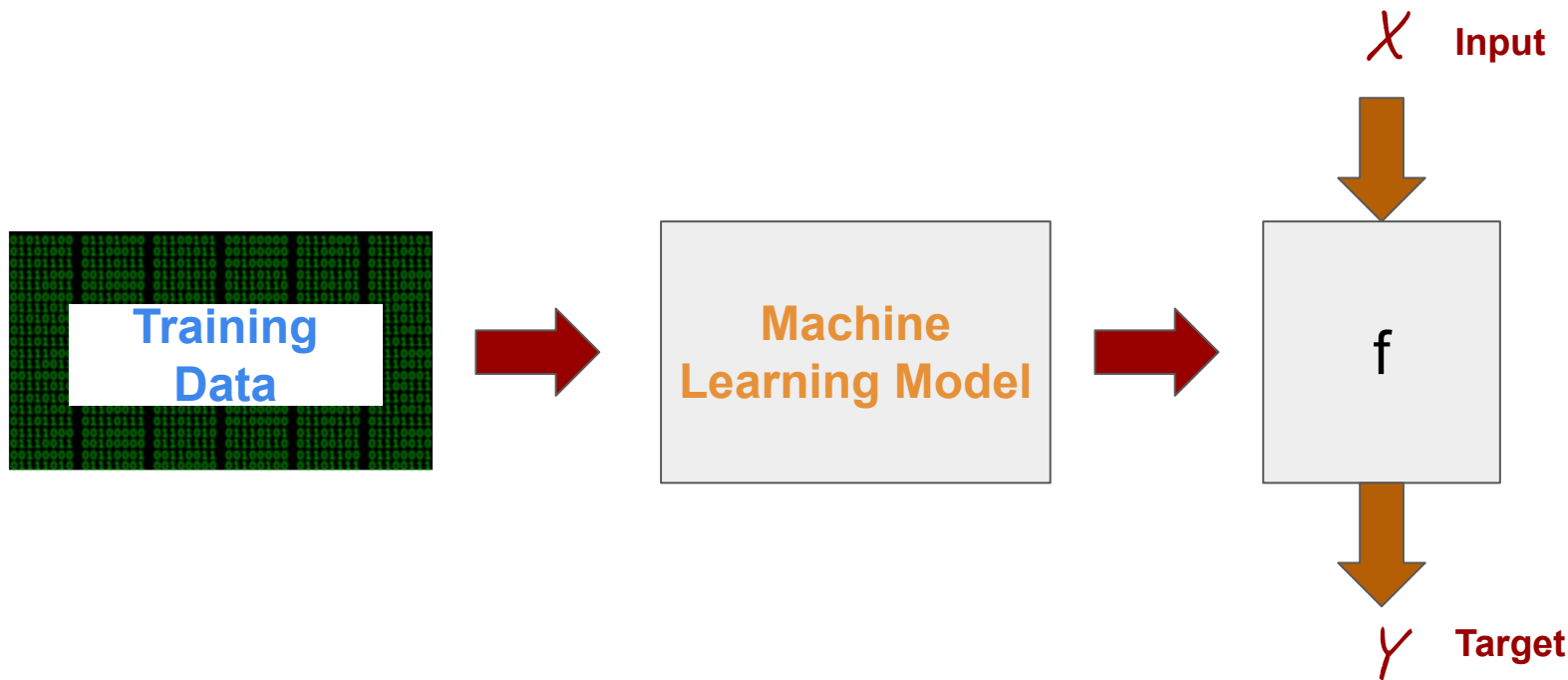


# Machine Learning Practices

# Late Policy

- Fill in the group information table (**group name!**) before 11:59 pm Jan 31.  
**Penalty is 2 scores for late filing.**
- **Late Policy for all assignments:**
  - **Without any reasonable justification, penalty is 25% for each additional late day.**

# Framework (supervised)



# A “Real” Machine Learning Task

- Example task: predict  $y$ , whether a string  $x$  is an email address
  - $x$ : “diszr@nus.edu.sg”  $y:1$
  - $x$ : “nusmsba”  $y:0$
  - $x$ : “@trump”  $y:0$
- If we are going to using ML models?
  - What are T, P, E?

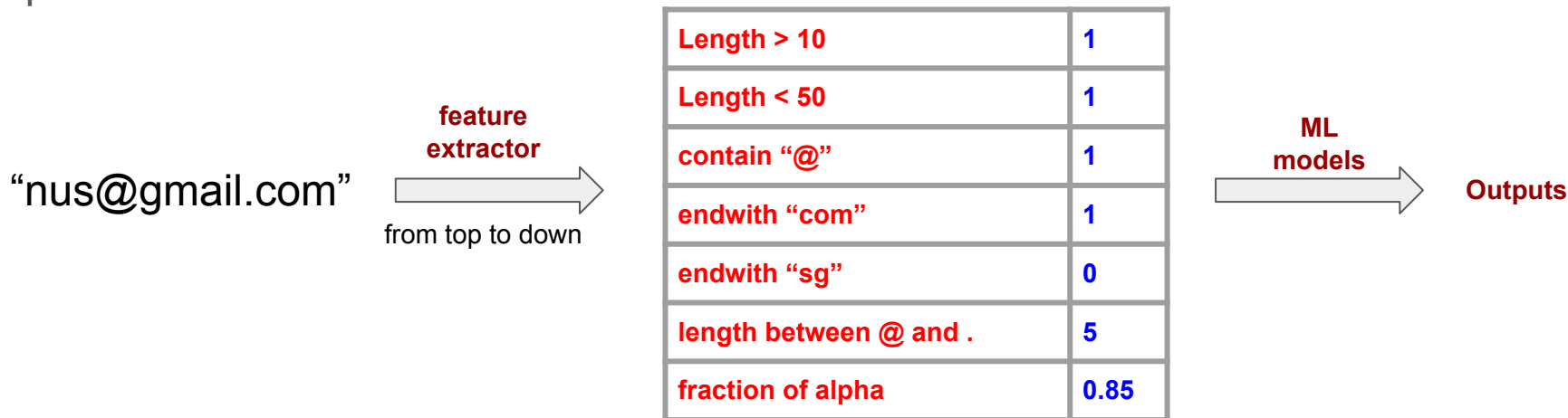
# Feature Extraction

- **Question:** what properties of x **might be** relevant for predicting y?
- **Feature extractor:** Given input x, output a set of (**feature name**, **feature value**) pairs.

“nus@gmail.com”  A **fixed-length** vector

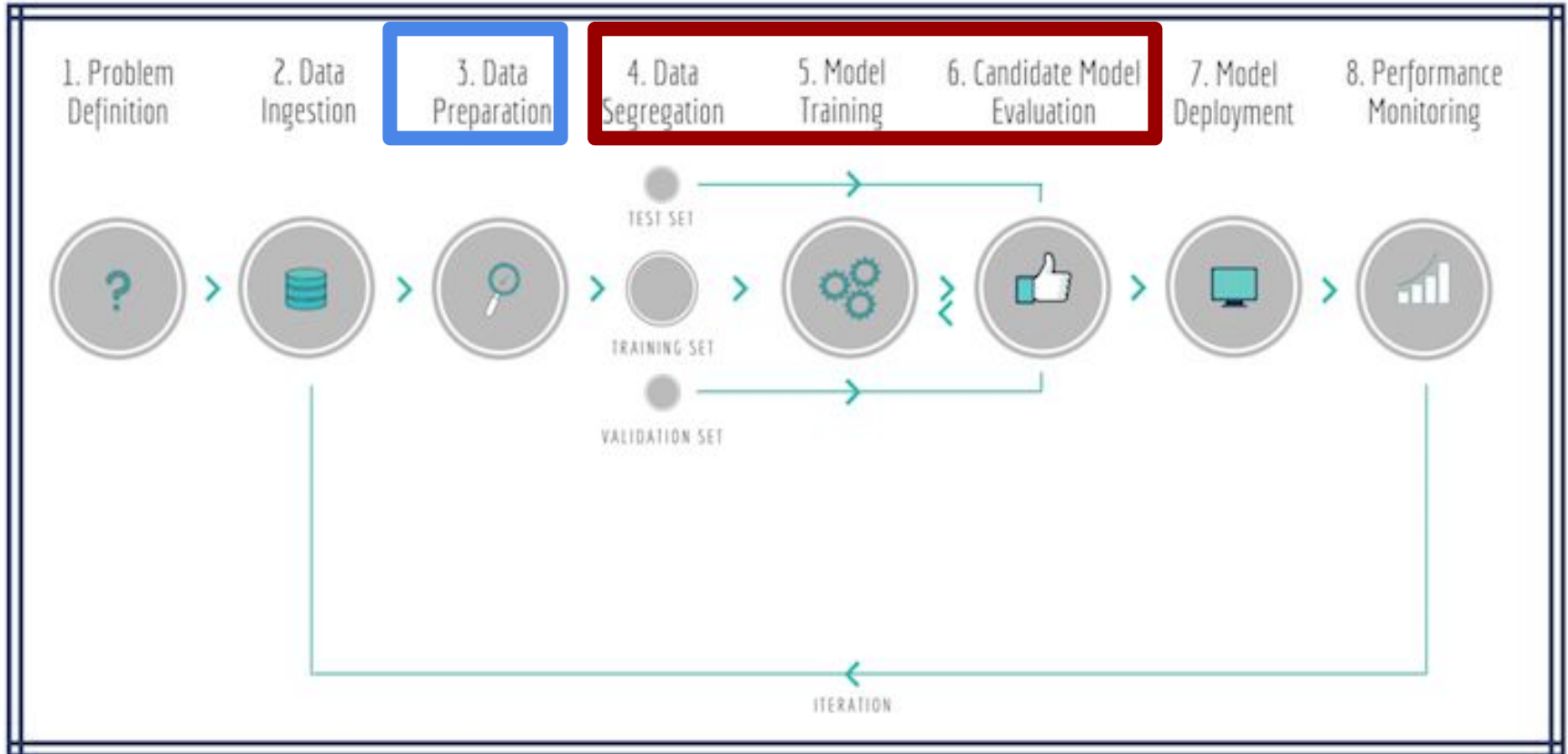
# Feature Extraction

- **Question:** what properties of x **might be** relevant for predicting y?
- **Feature extractor:** Given input x, output a set of (**feature name**, **feature value**) pairs.



- \*Exploration
- \*Transformation
- \*Fea. Engineering

## Modelling



# Agenda

1. Feature Engineering
2. Model Selection: Cross-validation
3. Hyperparameter Search
4. Hands-on Project: Cuisine Categorization



# Feature Engineering

Recall that computers only understand  
numbers (binary)

# What is Feature Engineering

- Features never fully describe the situation
  - All models are wrong, but some are useful. The practical question is how wrong do they have to be to not be useful.



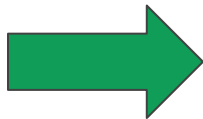
*George Box*

- Feature Engineering
  - How to represent examples by feature vectors
  - Suppose I would like to use examples from past to predict, at the current moment, which students will get an A in BT5153?
  - Some features may be useful: eg. GPA, prior programming experience
  - Other might cause me to overfit, e.g, birth month, weight.

# Example: YouTube Impression

```
0 : {  
  user_info : {  
    age: 42  
    ip_address: "192.168.0.1"  
    watch_history: [ "abc123",  
"def456" ]  
  }  
  impression {  
    ui_position: 0  
    video_id: "xyz789"  
    was_clicked: true  
    watch_time: 1.304  
  }  
}
```

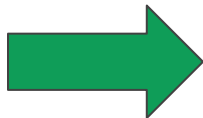
Real valued features can  
be copied over directly



**age\_feature = [ 42.0 ]**

# From Data to Features

```
0 : {  
  user_info : {  
    age: 42  
    ip_address: "192.168.0.1"  
    watch_history: [ "abc123",  
"def456" ]  
  }  
  impression {  
    ui_position: 0  
    video_id: "xyz789"  
    was_clicked: true  
    watch_time: 1.304  
  }  
}
```



Logs Data don't come  
to us a feature vectors

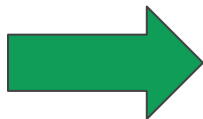
```
[  
  42.0,  
  1.0,  
  0.0,  
  0.0,  
  0.0,  
  9.321,  
  -2.20,  
  1.01,  
  0.0,  
  ...,  
]
```

Process of creating  
features from data is  
Feature Engineering

# From String to Features

```
0 : {  
  user_info : {  
    age: 42  
    ip_address:  
    watch_history:  
    "def456" ]  
  }  
  impression {  
    ui_position: 0  
    video_id: "xyz789"  
    was_clicked: true  
    watch_time: 1.304  
  }  
}
```

String Features can be handled with one-hot encoding



`impression_video_id_feature =`

[ 0,  
...,  
0,  
1,  
0,  
...,  
0 ]

$K$ : number of  
unique  
videos

One-hot encoding. This has a 1 for "xyz789", 0 for all others

# Feature Engineering

- Question u ask:

What properties of  $x$  **might be** relevant for predicting  $y$ ?

**This step is really important for linear models.**

# Properties of a Good Feature

Feature values should appear with **non-zero value** more than a small handful of times in the dataset



**my\_device\_id:8SK982ZZ1242Z**



**device\_model:galaxy\_s6**



# Properties of a Good Feature

Feature names should clearly describe the value's meaning, as well as indicate unit of measurement (where appropriate)



**price\_usd:29.99**



**feature\_7: -10**

# Properties of a Good Feature

Features shouldn't take on “magic” values

(Use an additional boolean feature like `is_watch_time_defined` instead!)



**`watch_time: -1.0`**



**`watch_time: 1.023`**



**`watch_time_is_defined: 1.0`**

# Properties of a Good Feature

The definition of a feature shouldn't change over time.

(Beware of depending on other ML systems!)



**city\_id:“br/sao\_paulo”**

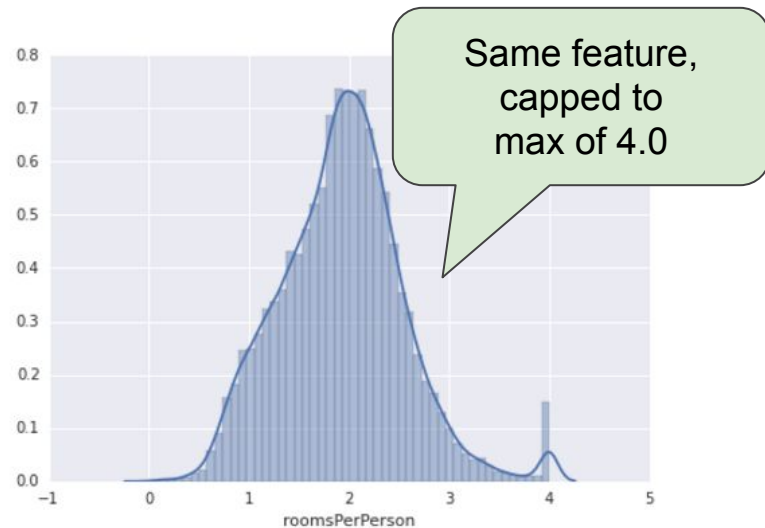
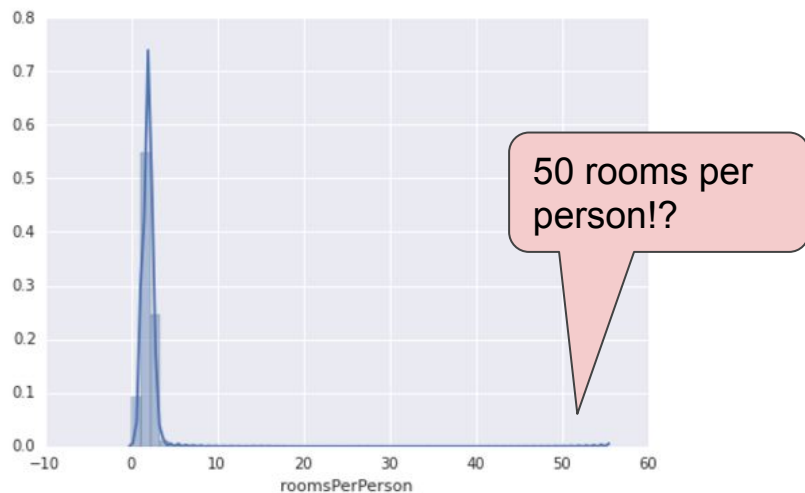


**inferred\_city\_cluster\_id:219**

# Properties of a Good Feature

## Distribution should not have crazy outliers

Ideally all features transformed to **a similar range**, like  $(-1, 1)$  or  $(0, 5)$ .



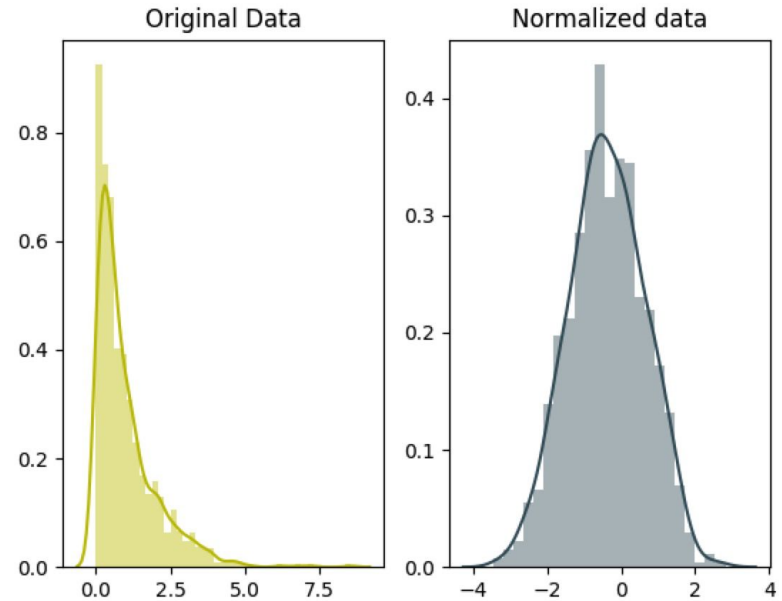
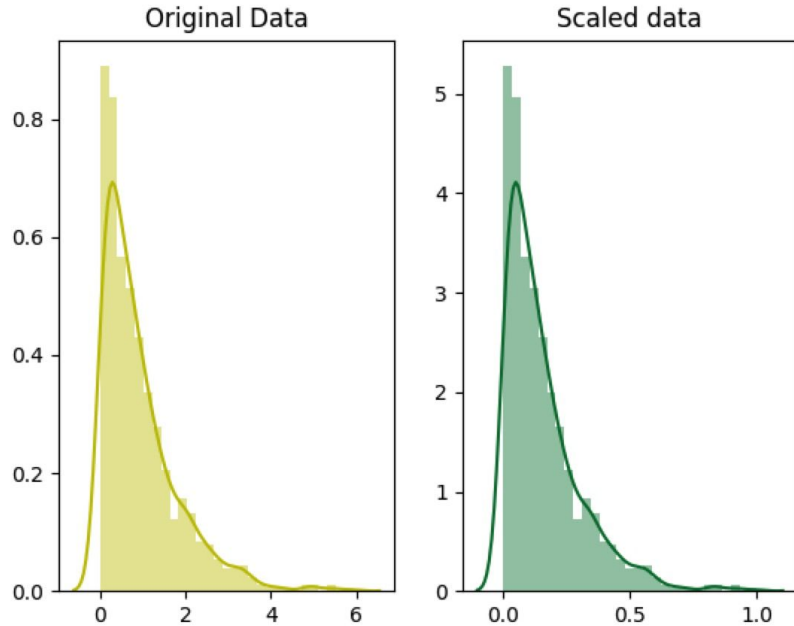
# Why Scaling?

- Range of values of data may vary widely
- Lots of machine learning algorithms are very sensitive to scales of features
  - K-nearest Neighbors
  - K-means
  - Logistic Regression
  - SVMs
  - Gradient Descent
  - PCA
  - Etc

**When distances of feature vector matter**

# Scaling Vs Normalization

Two ways to change the features of input data



# Scaling Vs Normalization

- Feature Scaling: change the range of your data

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

When distances measures are used in ML algorithms under the assumption that all features contribute equally

- Feature Normalization: change the shape of distribution of your data

$$\hat{x} = \frac{x - x_{mean}}{\sigma}$$

When machine learning models that assume that data is normally distributed e.g. linear regression, PCA, Gaussian Naive Bayes

# Good Habits: Know your data!!

- **Visualize:** Plot histograms, rank most to least common value.
- **Debug:** Duplicate examples? Missing values? Outliers? Data agrees with dashboards? Training and Validation data similar?
- **Monitor:** Feature quantiles, number of examples over time?



How about Unstructured Data ?

# Bag-of-Words

- Steps

- Build vocab i.e., set of all the words in the corpus
- Count the occurrence of words in each document

The cat and the dog play
The cat is on the mat

*corpus*

and, the, cat, dog, play, on, mat, is
--

*vocab.*

1	2	1	1	1	0	0
1	2	0	0	1	1	1

*countVec*

# N-gram model

- Steps

- Build vocab, which set of all n-gram in the corpus
- Count the occurrence of n-gram in each document

The cat and the dog play
The cat is on the mat

*corpus*

The cat, cat and, and the, the dog, dog play, cat is, is on, on the, the mat
--

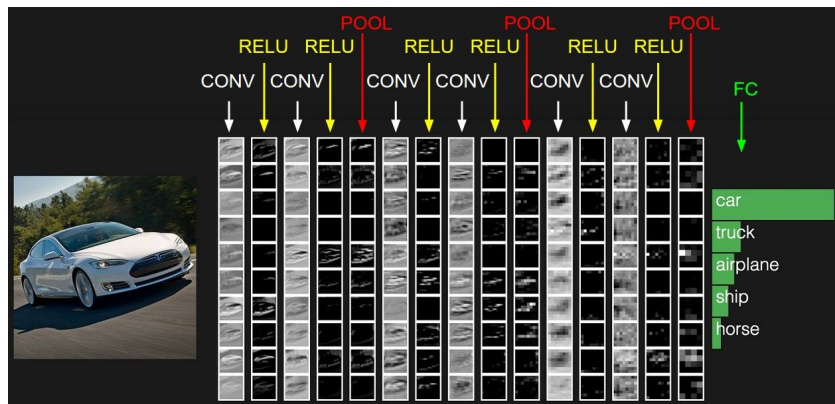
*vocab.*

1	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---

1	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---

# More Advanced Approach

- Using Deep learning
  - CNN, RNN, Attention Model
  - Learn representations from text, image, video, audio signal

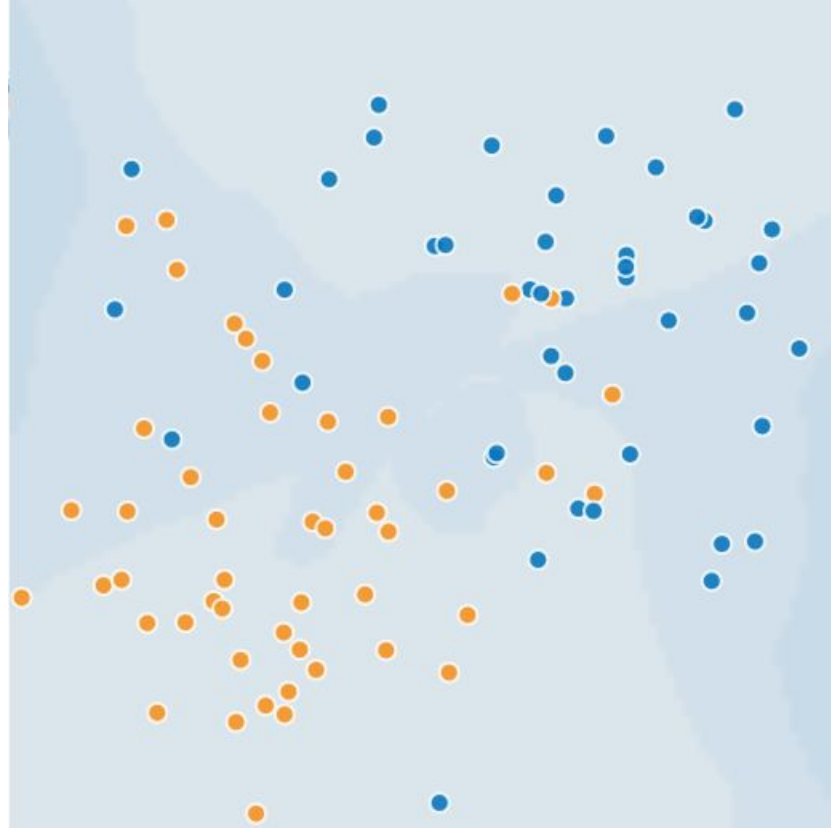


<http://cs231n.github.io/convolutional-networks/>

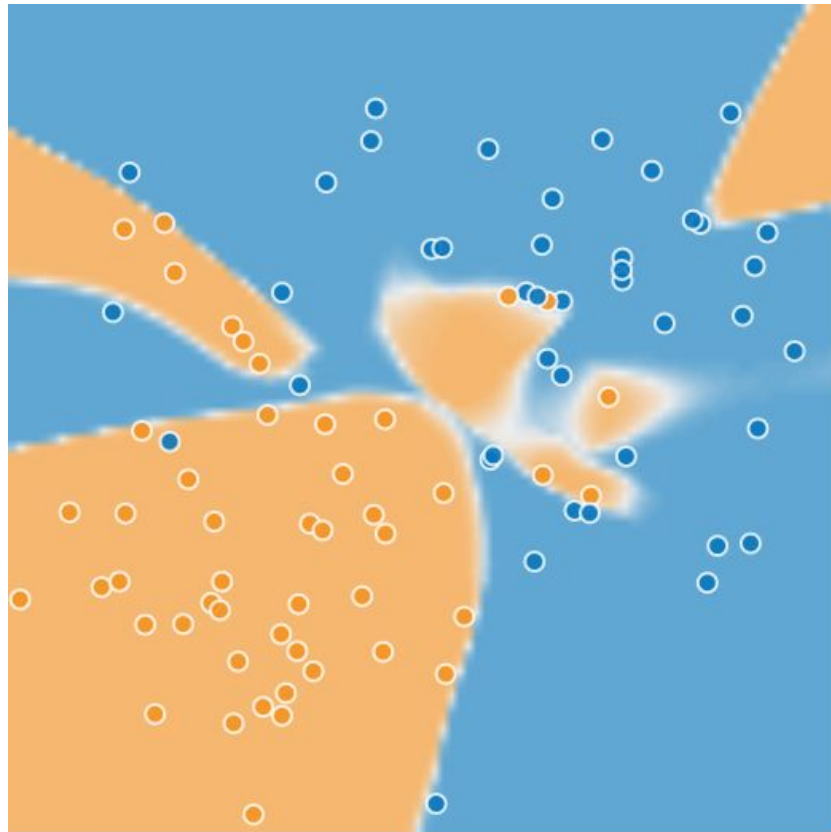
# Cross-Validation

Which measure should we look for  
model evaluation?

Let's try to train a model for this problem

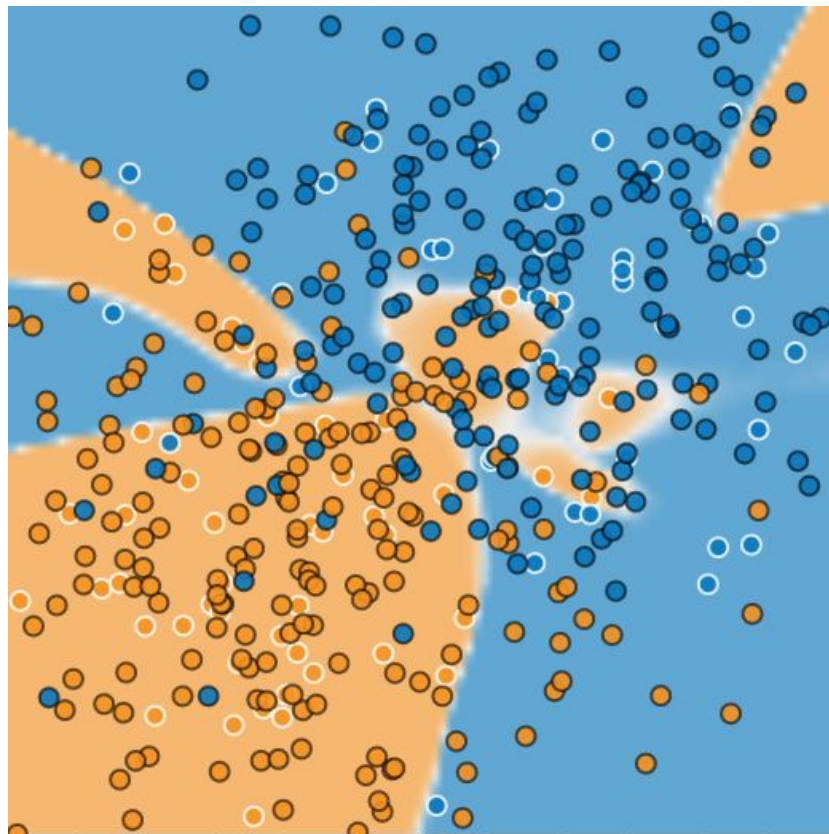


How about this model?





# Overfitting



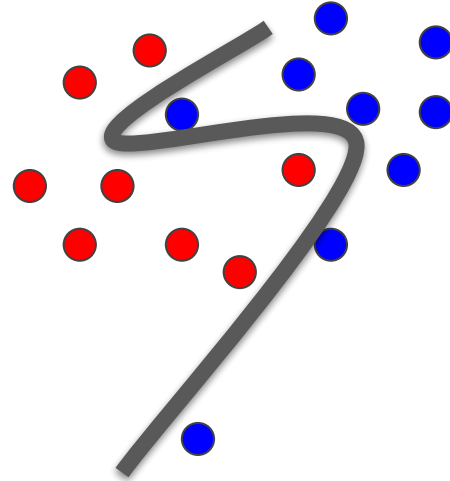
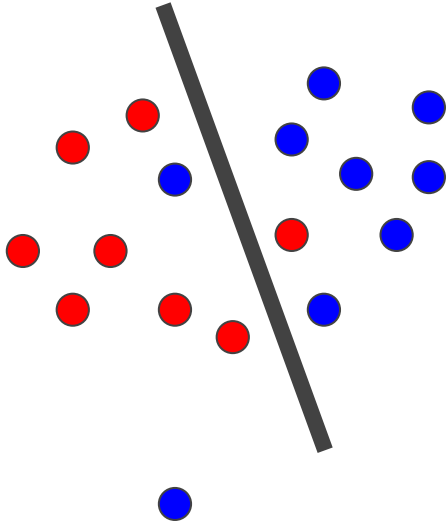
# Which measure should we look for model evaluation?

**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

# Which model is better?

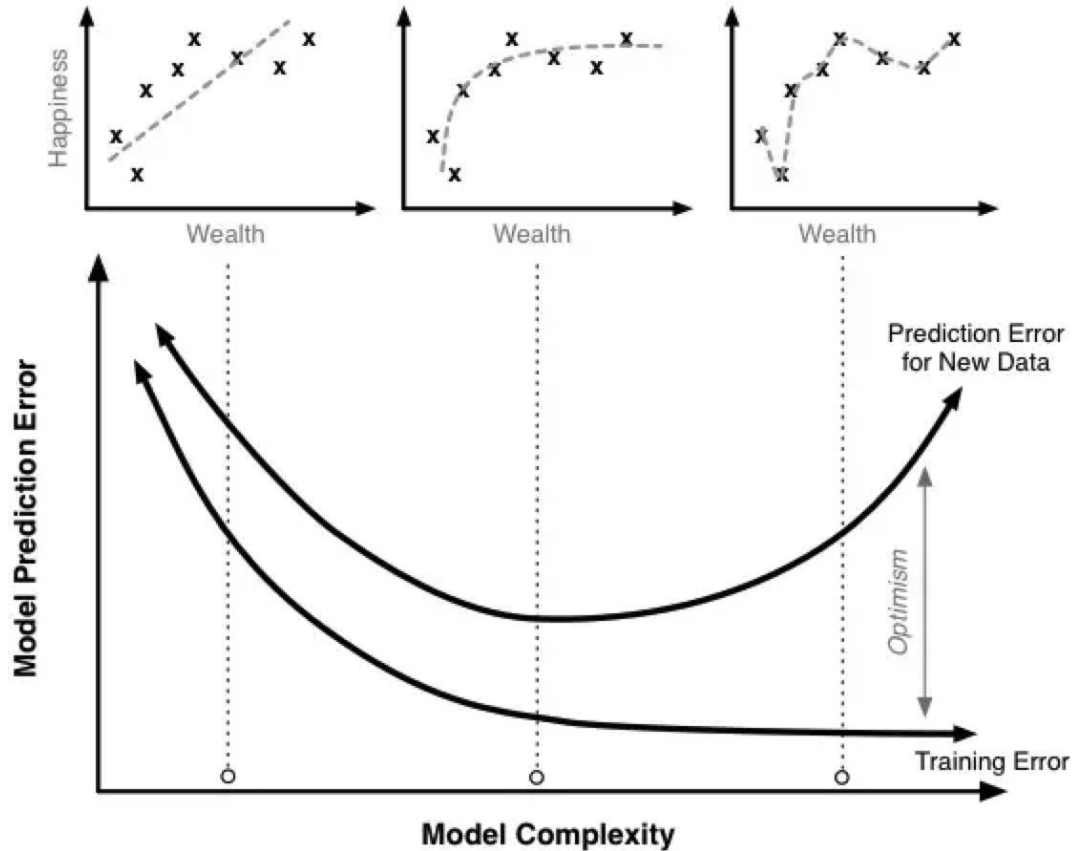


**SPAM** VS **Not SPAM**

# Model Complexity

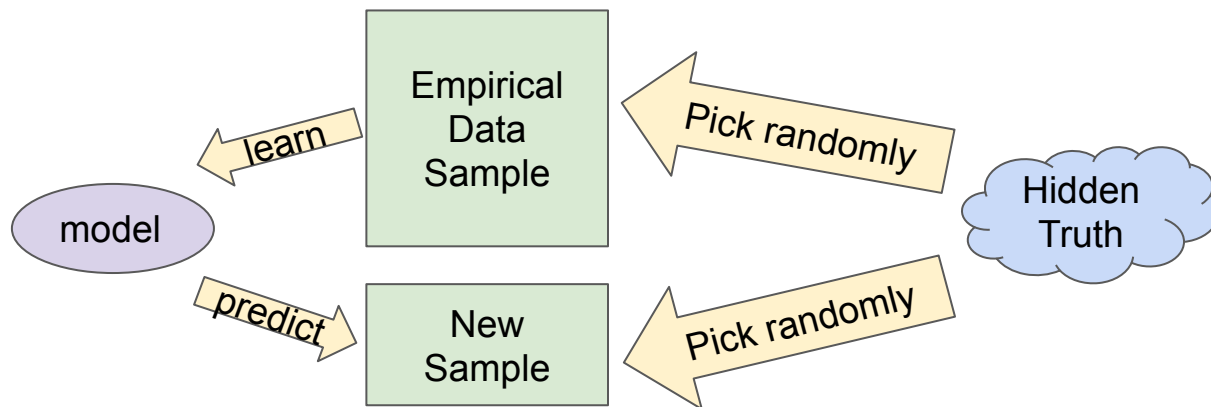
- Complex model easily overfits the training data
- Then, the trained model is unable to generalize on testing data
- overfitting vs underfitting
  - overfitting: small training error but large testing error
  - underfitting: large training and testing errors

# 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



*William of Occam*



# Is the model overfitting?

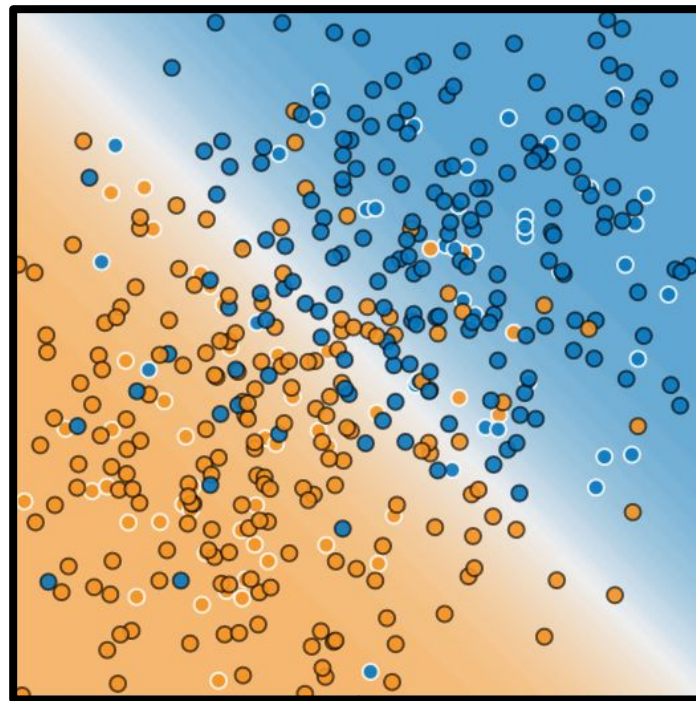
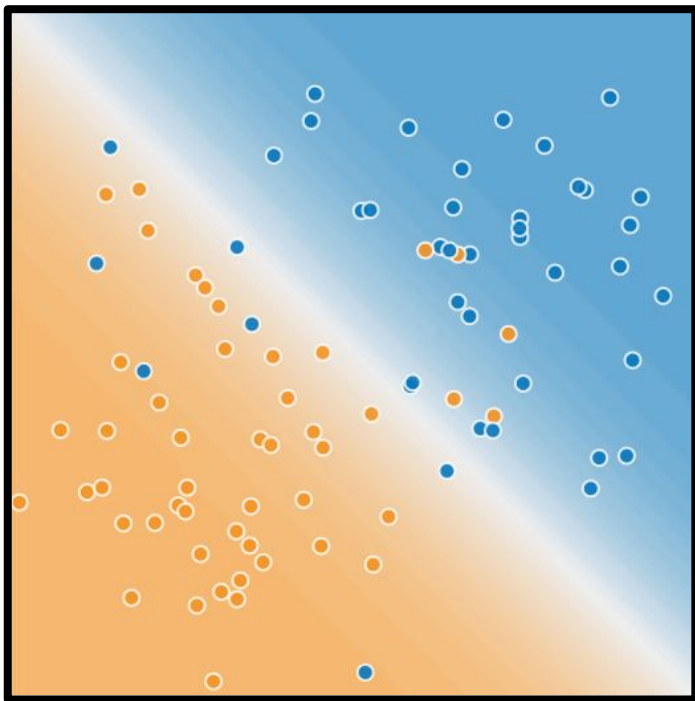
- Empirically:
  - Key point: will our model be good on new samples?
  - Evaluate: get new samples of data (test set)
  - If test set is large enough and we do not cheat by using test set over and over, the good performance on test set can be a useful indicator of model's generalization capability

# Training/Test Splitting

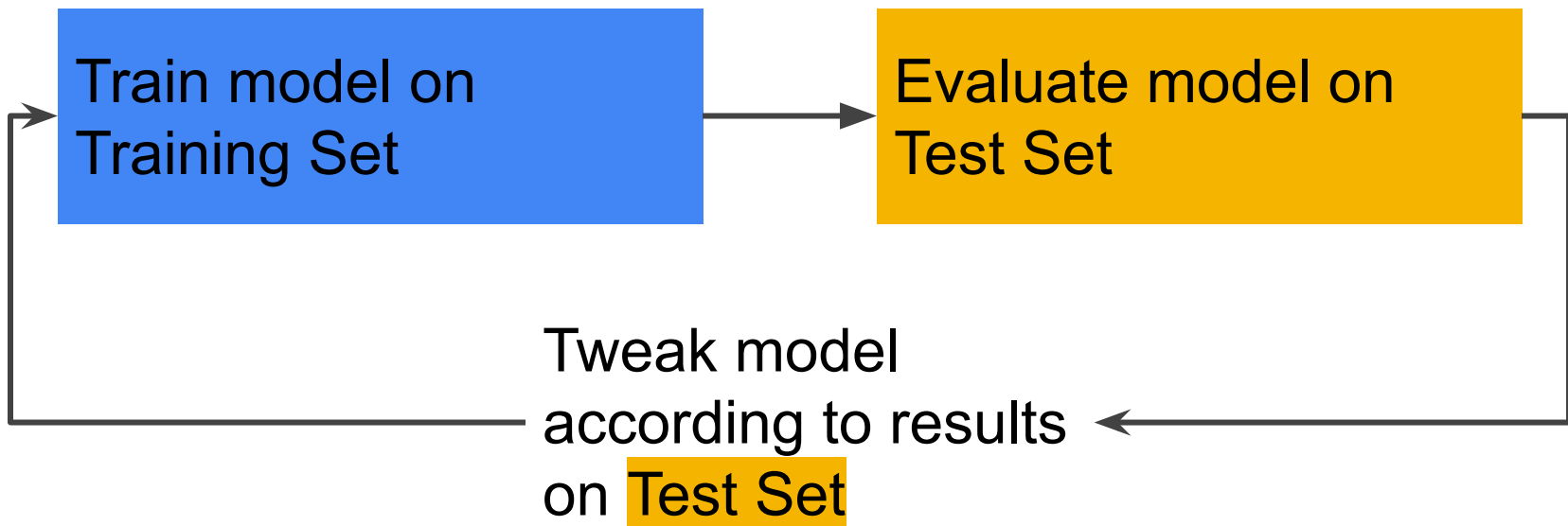


- If models do much better on the training set than the testing set, then models are likely overfitting.
- How do we divide?
  - Randomization for splitting
  - Larger training data size -> better model
  - Larger testing data size -> more confident in model's evaluation
  - One practical rule: 10-15%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

# Training Evaluation v.s. Test Evaluation



# How about this workflow?



Pick model that does best on Test Set.

# Partition Data Sets

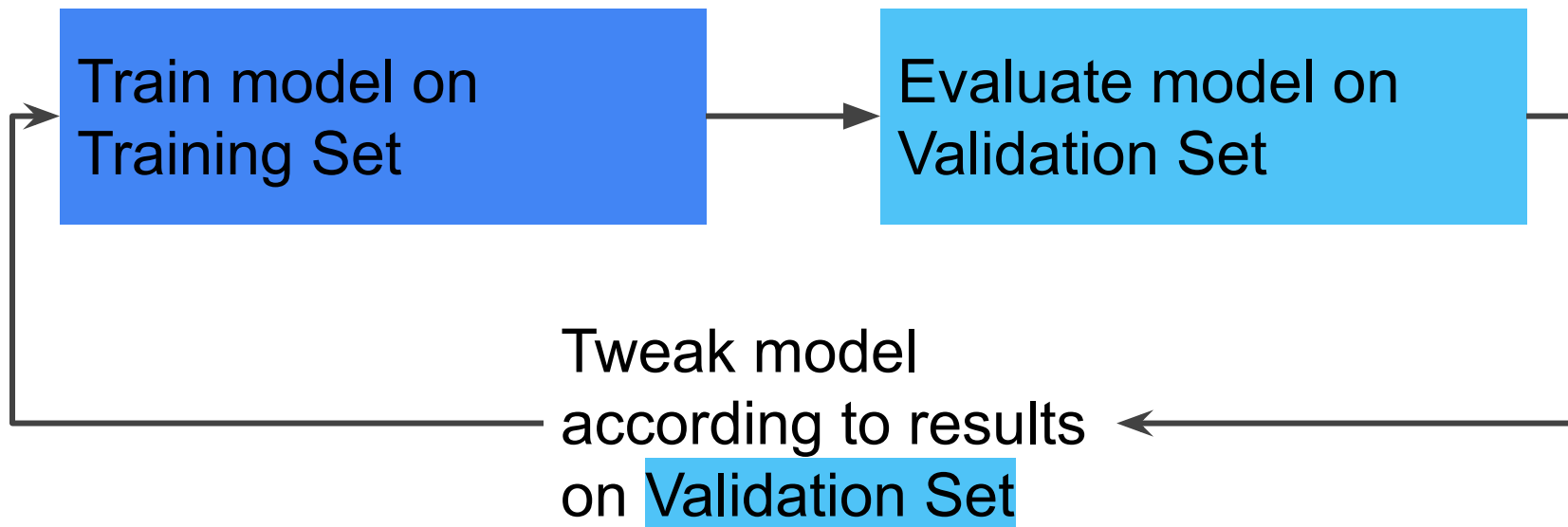


Training Set

Validation Set

Test Set

## Better Workflow: Use a validation set



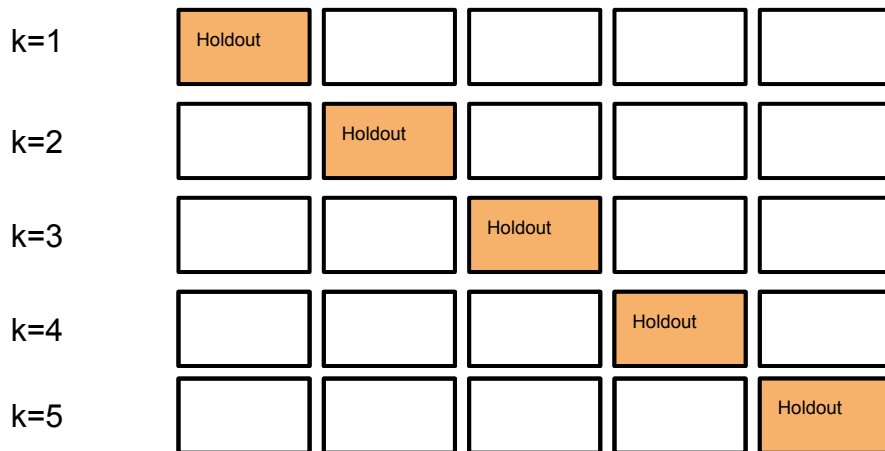
Pick model that does best on Validation Set  
Confirm results on Test Set

# Cross-validation

- If we have a small dataset: CV can be conducted
- 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





# Summary

- 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.

# As a Summary

- What should be the targets set for Machine Learning
  - Minimize loss on the training set
  - Minimize loss on the validation set
  - Minimize loss on the test set
  - Minimize loss on unseen future examples

## 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:

$$\|y - Xw\|^2 + \alpha * \|w\|^2$$

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**

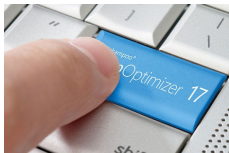
# Hyperparameter Optimization

# 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**

⚙️  
n\_layers = 3  
n\_neurons = 512  
learning\_rate = 0.1



Weights  
optimization



85%

⚙️  
n\_layers = 3  
n\_neurons = 1024  
learning\_rate = 0.01



Weights  
optimization



80%

# 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**

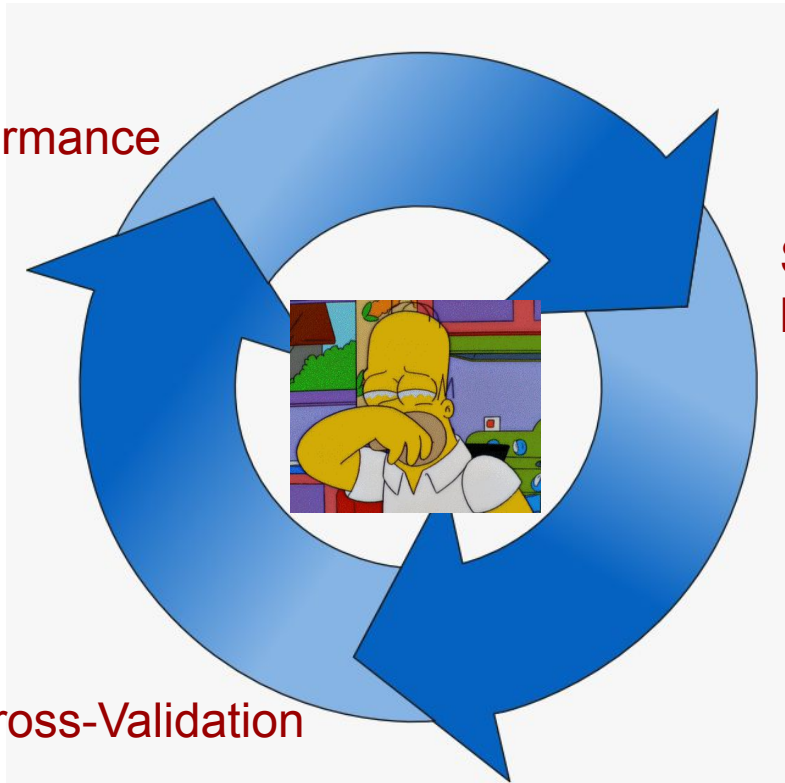


# Searching is Iterative, then Expensive

Track the performance

Select one potential  
hyperparameter set

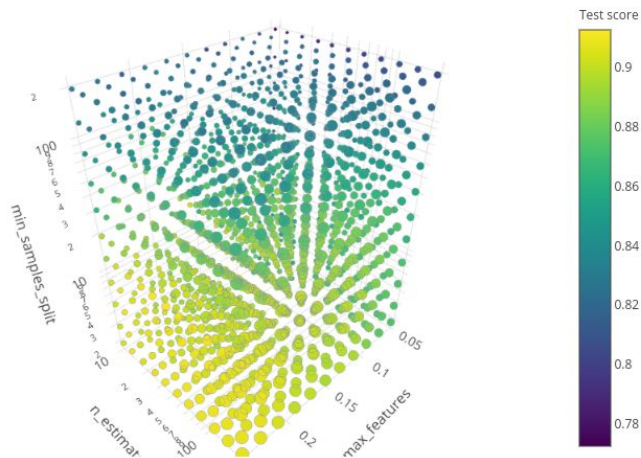
Run Cross-Validation



# 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

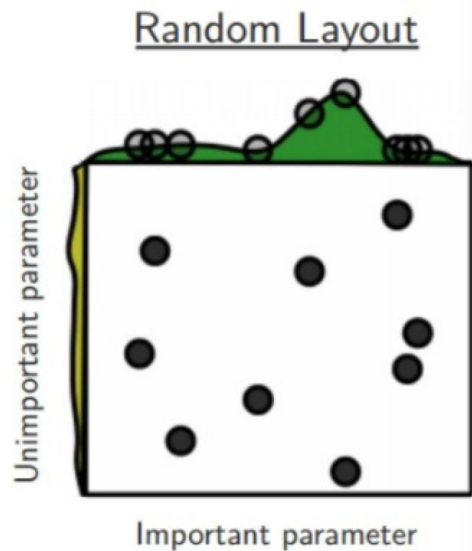
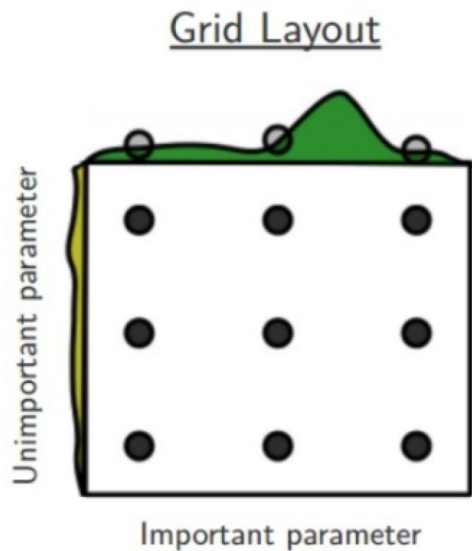
3D visualization of the grid search results



**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*

# Advanced Search Algorithms

- 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**