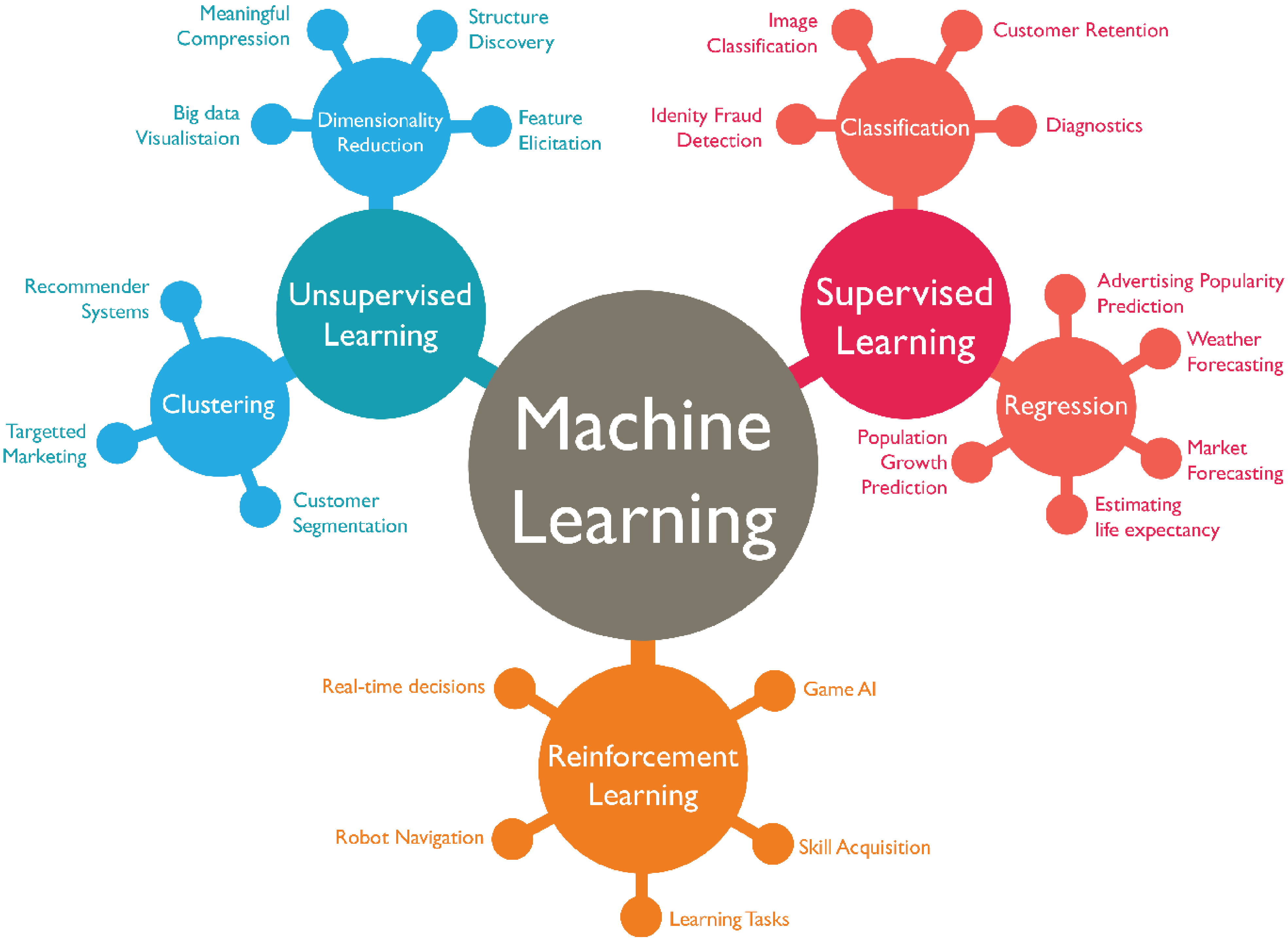
Machine Learning Basics

Prof. Kuan-Ting Lai

2019/3/5

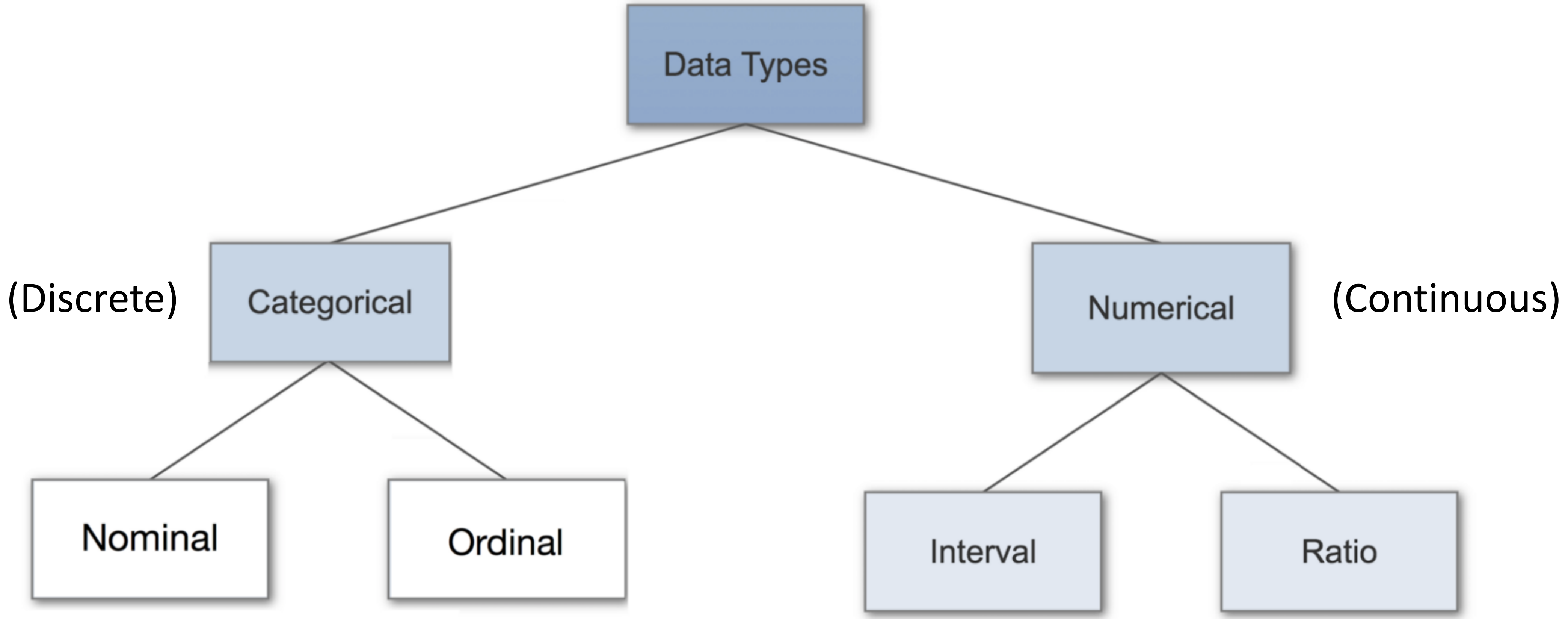


Machine Learning
Unsupervised Learning
Dimensionality Reduction
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Classification
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

Types of Machine Learning

- Supervised Learning
  - Build a mathematical model from labeled data (training data)
  - Each training example has one or more inputs and a desired output
- Semi-supervised Learning:
  - only part of the training data are annotated
- Unsupervised Learning
  - The input data have no labels
- Reinforcement Learning
  - Design a software agent that can self-learn a model to maximize cumulative reward in an environment

Data Types (Measurement Scales)



Nominal Data (Labels)

- Nominal data are labeling variables without any quantitative value
- Encoded by one-hot encoding for machine learning
- Examples:

What is your Gender?

- Female
- Male

What languages do you speak?

- Englisch
- French
- German
- Spanish

Ordinal Data

- Ordinal values represent discrete and ordered units
- The order is meaningful and important

What Is Your Educational Background?

- ○ 1 - Elementary
- ○ 2 - High School
- ○ 3 - Undegraduate
- ○ 4 - Graduate

Interval Data

- Interval values represent **ordered units that have the same difference**
- Problem of Interval: **Don’t have a true zero**
- Example: Temperature Celsius (°C) vs. Fahrenheit (°F)

Temperature?

- ○ - 10
- ○ -5
- ○ 0
- ○ + 5
- ○ + 10
- ○ + 15

Ratio

- Same as interval data but have absolute zero
- Can be applied to both descriptive and inferential statistics
- Example: weight & height

Machine Learning vs. Statistics

- https://www.r-bloggers.com/whats-the-difference-between-machine-learning-statistics-and-data-mining/

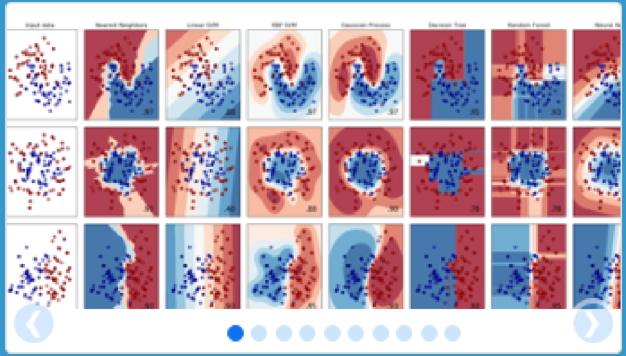
Machine learning	Statistics
network, graphs	model
weights	parameters
learning	fitting
generalization	test set performance
supervised learning	regression/classification
unsupervised learning	density estimation, clustering
large grant = $1,000,000	large grant = $50,000
nice place to have a meeting: Snowbird, Utah, French Alps	nice place to have a meeting: Las Vegas in August

scikit-learn.org

scikit learn

Home Installation Documentation Examples

Google Custom Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

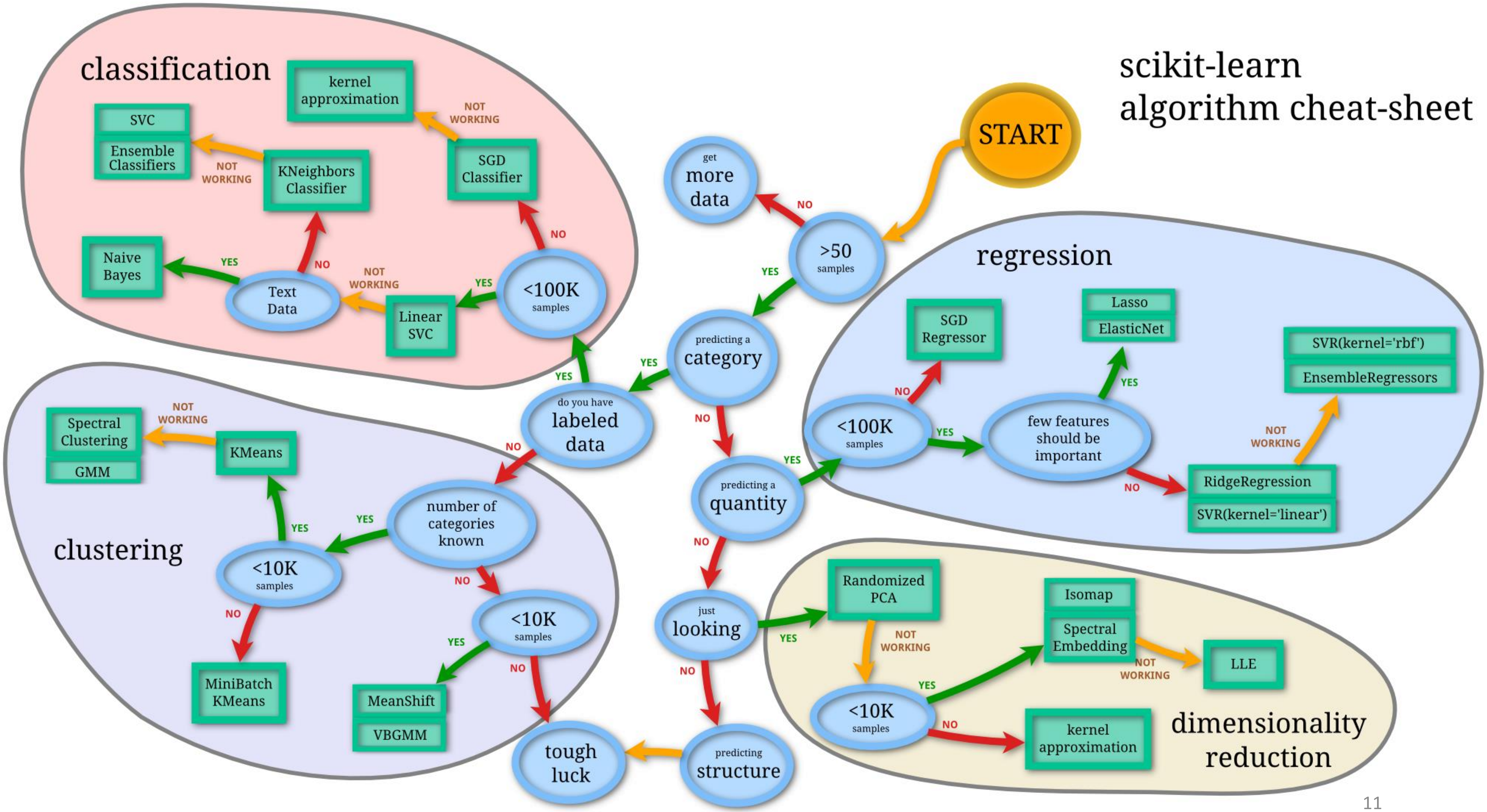
Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples



scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
<100K samples
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC
Ensemble Classifiers
SGD Classifier
kernel approximation
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering
GMM
MiniBatch KMeans
<10K samples
MeanShift
VBGMM
regression
<100K samples
SGD Regressor
few features should be important
Lasso
ElasticNet
RidgeRegression
SVR(kernel='linear')
SVR(kernel='rbf')
EnsembleRegressors
dimensionality reduction
Randomized PCA
<10K samples
Isomap
Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING

Supervised and Unsupervised Learning

- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Dimension Reduction

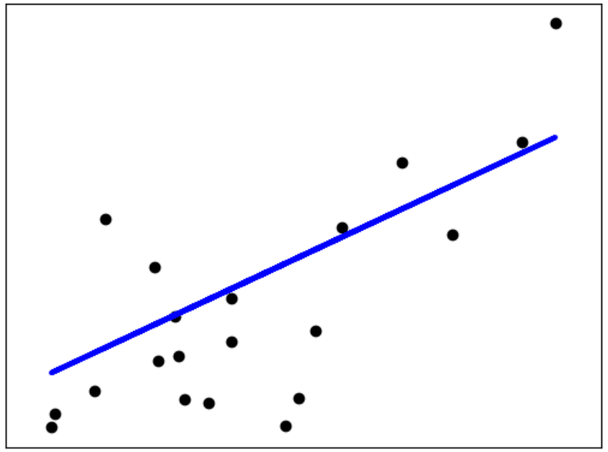
Supervised and Unsupervised Learning

- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Dimension Reduction

Linear Regression

- Least Squares

$$\min_{w} \left\| \boldsymbol{w}^T \boldsymbol{x} - y \right\|_2^2$$

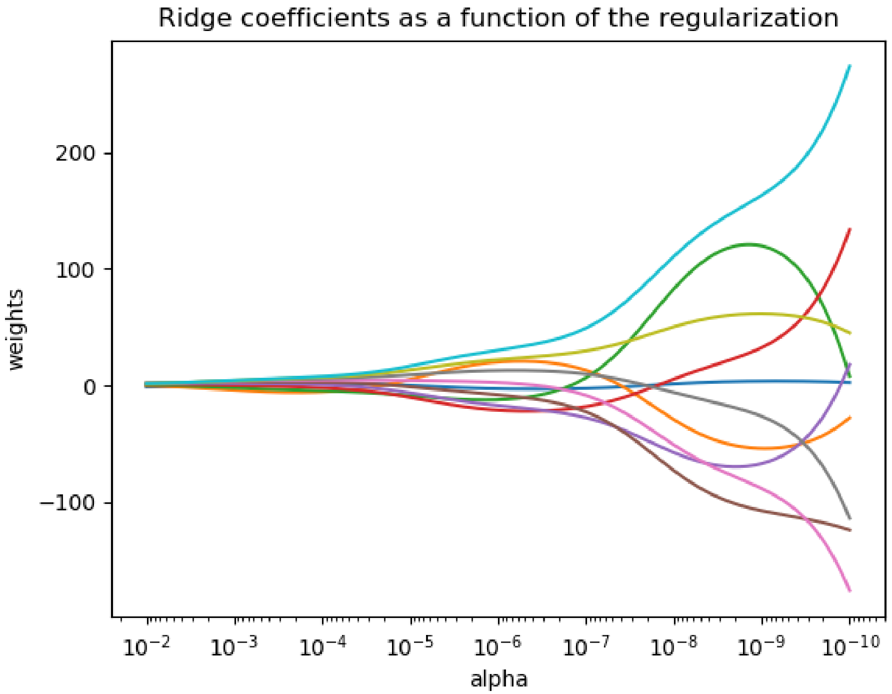


```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
diabetes_X_train = diabetes_X[:-20]
diabetes_y_train = diabetes.target[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue',
linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Ridge Regression

- Impose a penalty on the size of coefficients

$$\min_{w} \|\boldsymbol{w}^T\boldsymbol{x} - y\|_2{}^2 + \alpha\|w\|_2{}^2$$




```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
# X is the 10x10 Hilbert matrix
X = 1. / (np.arange(1, 11) + np.arange(0, 10)[:, np.newaxis])
y = np.ones(10)
###############################################################
# Compute paths
n_alphas = 200
alphas = np.logspace(-10, -2, n_alphas)
coefs = []
for a in alphas:
    ridge = linear_model.Ridge(alpha=a, fit_intercept=False)
    ridge.fit(X, y)
    coefs.append(ridge.coef_)
###############################################################
# Display results
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the
regularization')
plt.axis('tight')
plt.show()
```

Supervised and Unsupervised Learning

- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Dimension Reduction

Naïve Bayes

- Probabilistic classifier based on Bayes' theorem

$$\mathrm{P}(y|\boldsymbol{x}) = \mathrm{P}(y|x_1, x_2, \dots, x_n)$$

$$\mathrm{P}(y|\boldsymbol{x}) = \frac{\mathrm{P}(y)\mathrm{P}(\boldsymbol{x}|y)}{\mathrm{P}(\boldsymbol{x})} \qquad Posterior = \frac{Prior \times Liklihood}{Evidence}$$

- Bayes assumes features x_i are conditional independent

$$\mathrm{P}(\boldsymbol{x}|y) = \mathrm{P}(x_1|y)\,\mathrm{P}(x_2|y) \cdots \mathrm{P}(x_n|y) = \prod_{i=1}^{n} \mathrm{P}(x_i|y)$$

$$\Rightarrow \mathrm{P}(y|\boldsymbol{x}) = \frac{\mathrm{P}(y)\prod_{i=1}^{n} \mathrm{P}(x_i|y)}{\mathrm{P}(\boldsymbol{x})} \propto \mathrm{P}(y)\prod_{i=1}^{n} \mathrm{P}(x_i|y)$$

$$\Rightarrow \hat{y} = arg\max_{y} \mathrm{P}(y)\prod_{i=1}^{n} \mathrm{P}(x_i|y)$$

Gaussian Naive Bayes in Scikit

- $P(x_i|y_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{\left(-\frac{(x-\sigma\mu_k)^2}{2\sigma_k{}^2}\right)}$

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)
>>> print("Number of mislabeled points out of a total %d points : %d"
... % (iris.data.shape[0],(iris.target != y_pred).sum()))
Number of mislabeled points out of a total 150 points : 6
```

Logistic Regression (3-1)

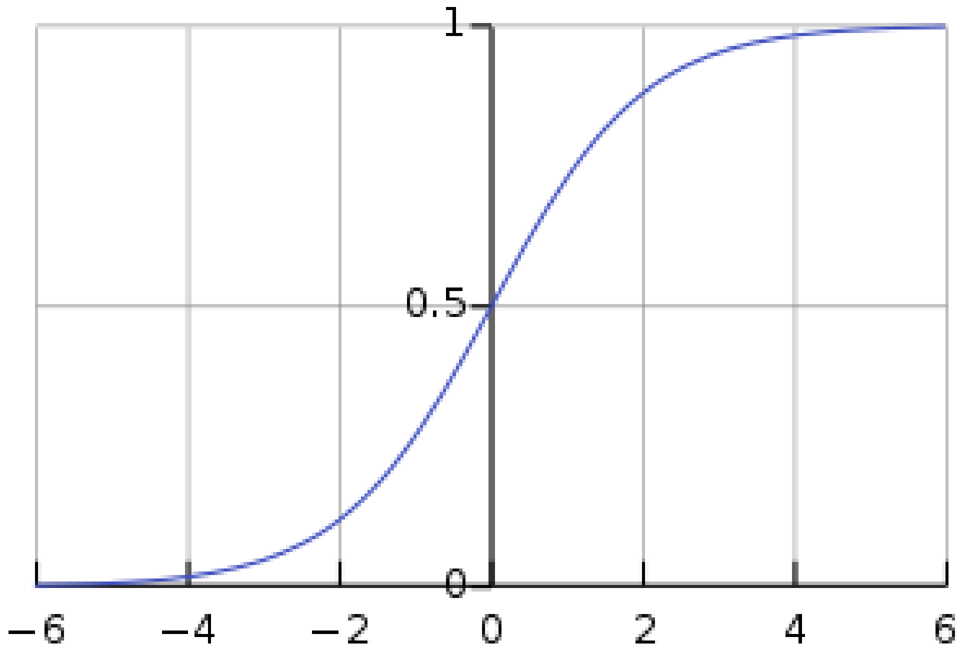
- Sigmoid function

$$S(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$$

- Derivative of Sigmoid

$$S(x) = S(x)(1\text{-}\, S(x))$$

S-shaped curve



https://en.wikipedia.org/wiki/Sigmoid_function

Logistic Regression (3-2)

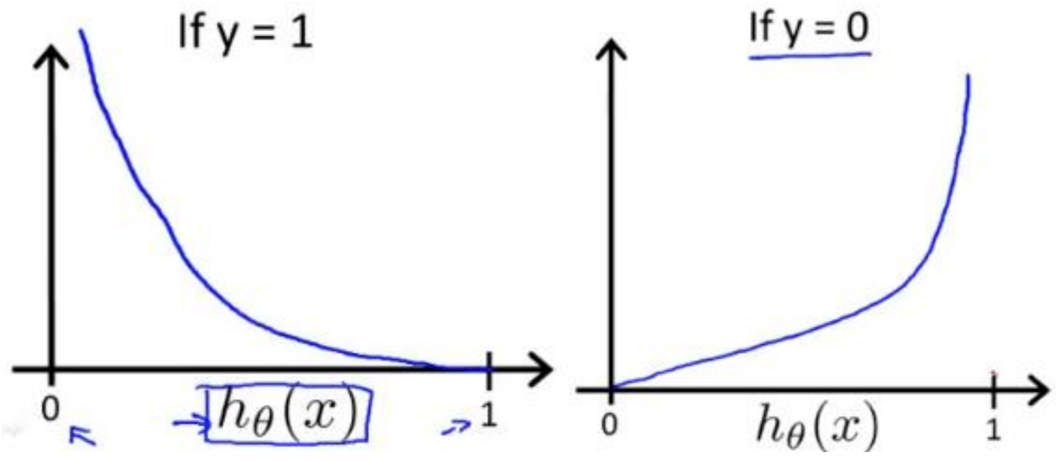
- Binary classification with decision boundary t

$$y = P(x, w) = P_\theta(x) = \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x} + b)}}$$

$$y = \begin{cases} 0, & x < t \\ 1, & x \geq t \end{cases}$$

Logistic Regression (3-3)

- Loss function: cross entropy



$$\text{loss}=\begin{cases}-\log(1-P_\theta(x)),\ \ if\ y=1\\ -\log(P_\theta(x)),\ \ if\ y=0\end{cases}$$

$$\Rightarrow L_\theta(\mathrm{x}) = -y\log(1-P_\theta(x)) + \text{-}(1-\mathrm{y})\log(1-P_\theta(x))$$

$$\nabla L_W(\mathrm{x}) = -(y-P_\theta(x))x$$

Logistic Regression Example

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
 >>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

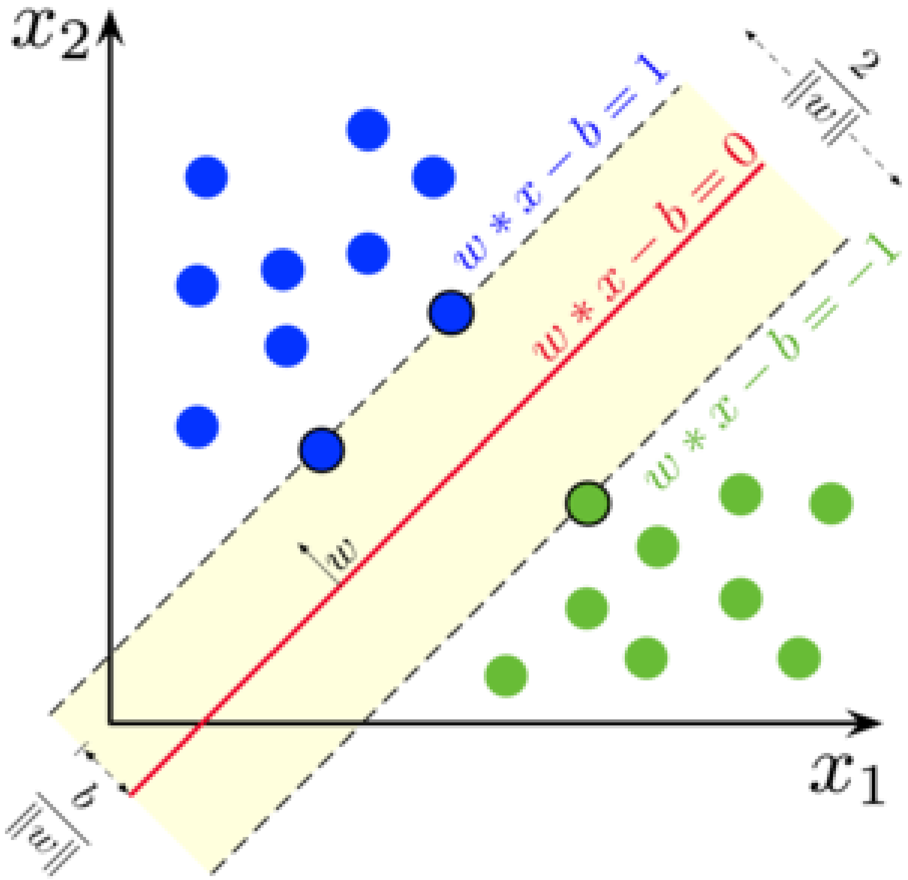
Support Vector Machine (SVM)

- SVM builds a model to assign each example to one category or the other
- Non-probabilistic binary linear classifier
- Hard margin

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \qquad \text{for all } 1 \leq i \leq n.$$

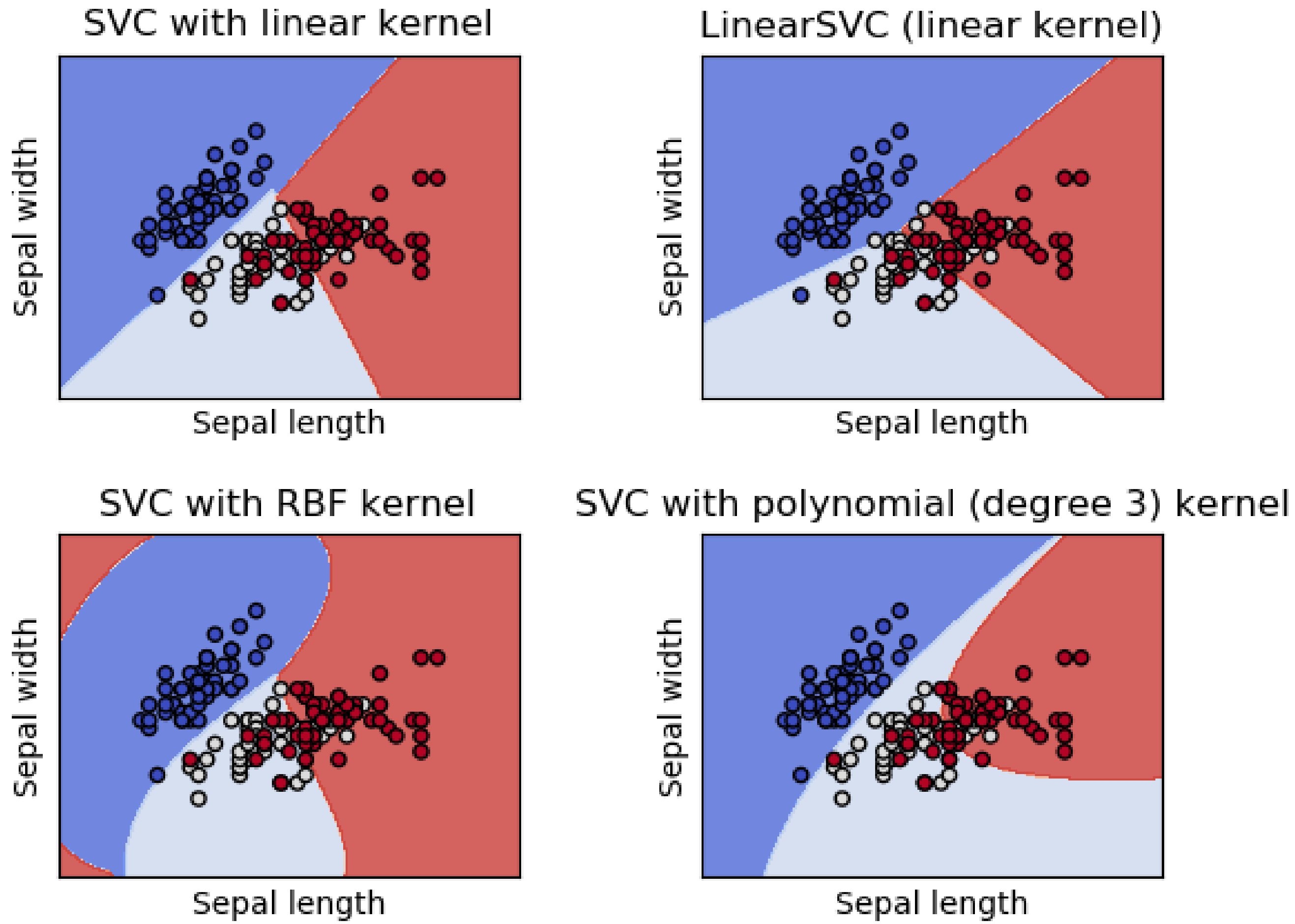
- Soft margin

$$\left[\frac{1}{n}\sum_{i=1}^{n} \max\left(0, 1 - y_i(\vec{w} \cdot \vec{x}_i - b)\right)\right] + \lambda \|\vec{w}\|^2$$



SVM with Kernel

- Handle non-linear data



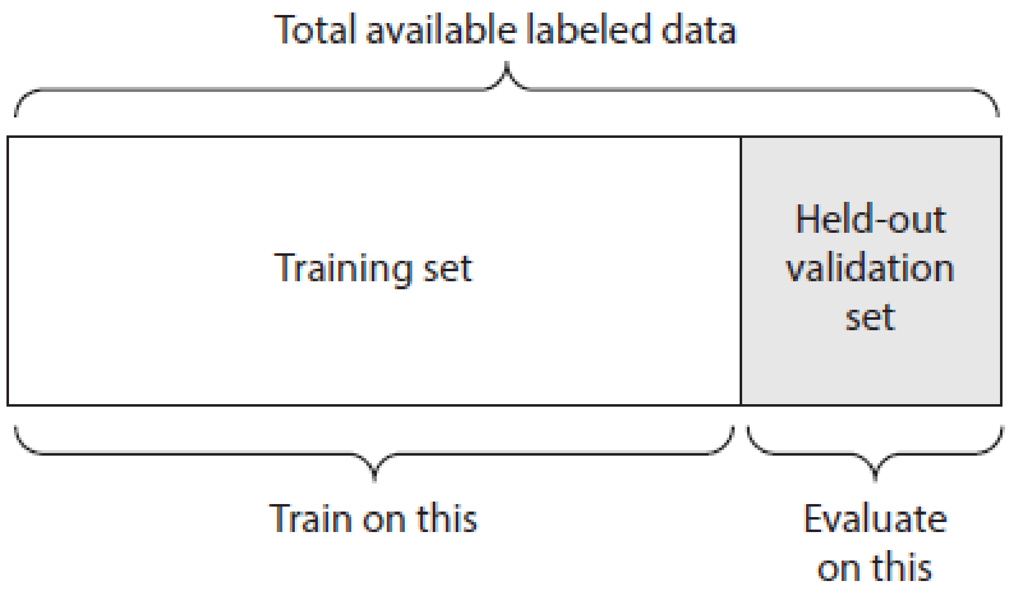
Support Vector Classification (SVC)

- Multi-class SVM
- One-against-One
- One-against-All

```
>>> X = [[0], [1], [2], [3]] >>>
Y = [0, 1, 2, 3]
>>> clf = svm.SVC(gamma='scale',
decision_function_shape='ovo’)
>>> clf.fit(X, Y)
SVC(C=1.0, cache_size=200, class_weight=None,
coef0=0.0,
 decision_function_shape='ovo', degree=3,
gamma='scale', kernel='rbf’,
 max_iter=-1, probability=False, random_state=None,
shrinking=True,
 tol=0.001, verbose=False)
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes: 4*3/2 = 6 6
>>> clf.decision_function_shape = "ovr"
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes 4
```

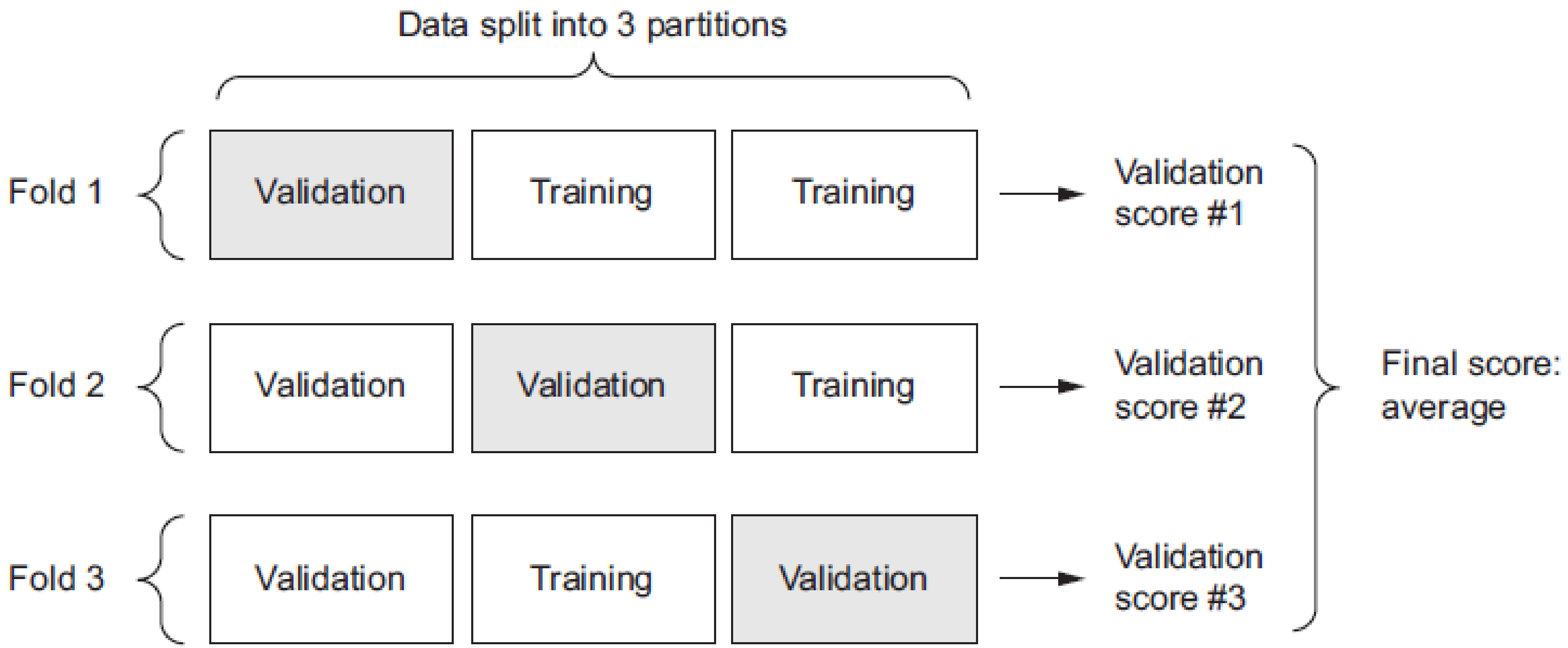
Training, Validation, Testing

- Tuning the *hypterparameters* of our model
- The information of test data should not be leaked into our model
- Better generalize the model to future unseen data



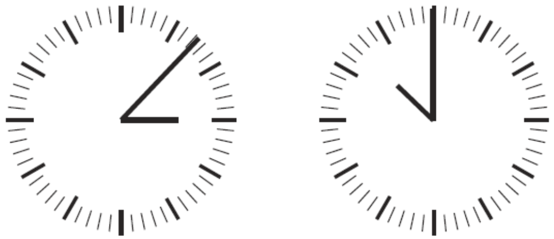
K-Fold Cross Validation

- Lower the variance of validation set



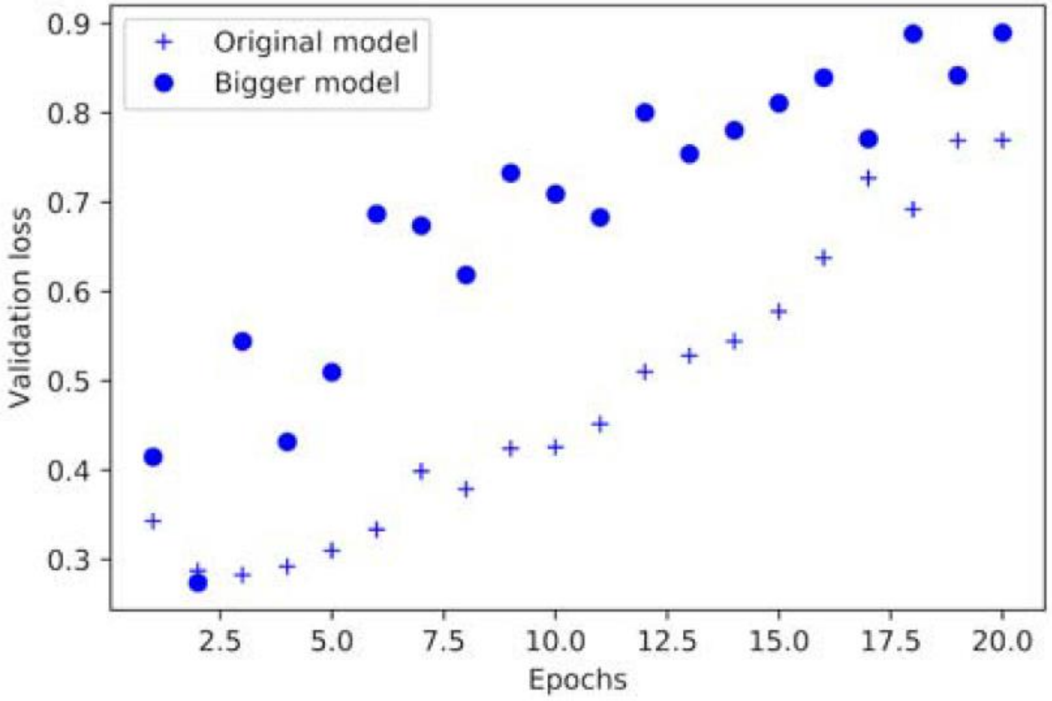
Feature Engineering & Feature Learning

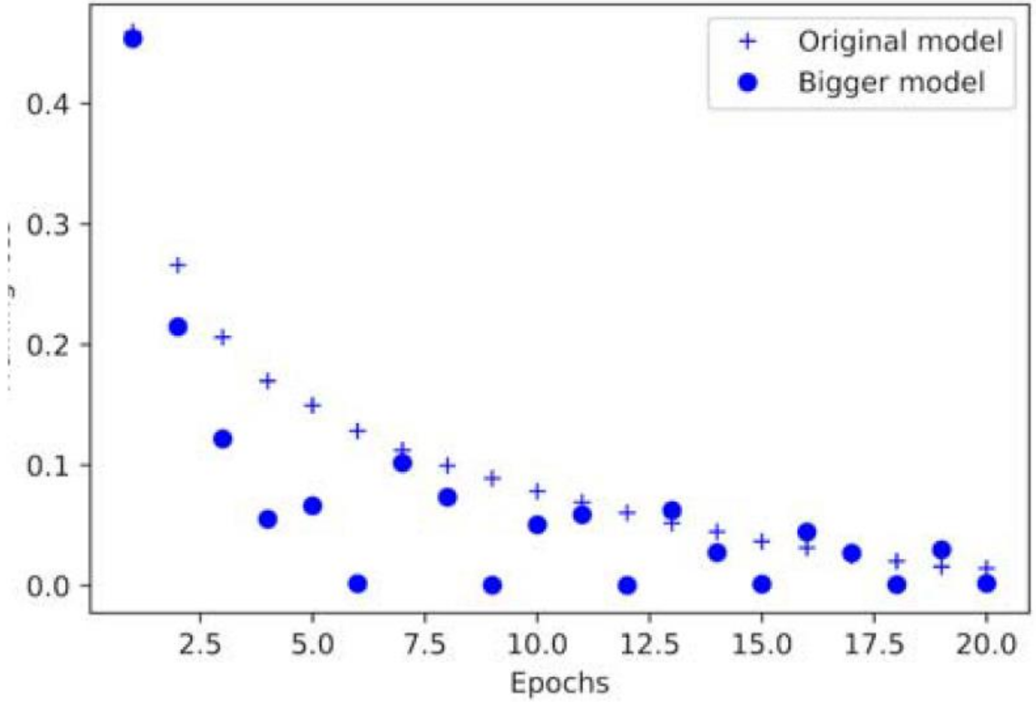
- Data preprocessing
  - Vectorization
  - Normalization
  - handling missing value
- Feature engineering
  - Good features still make learning easier

Raw data: pixel grid		
Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

Capacity, Overfitting and Underfitting

- The more capacity the network has, the more quickly it can model the training data, but the more susceptible it is to overfitting

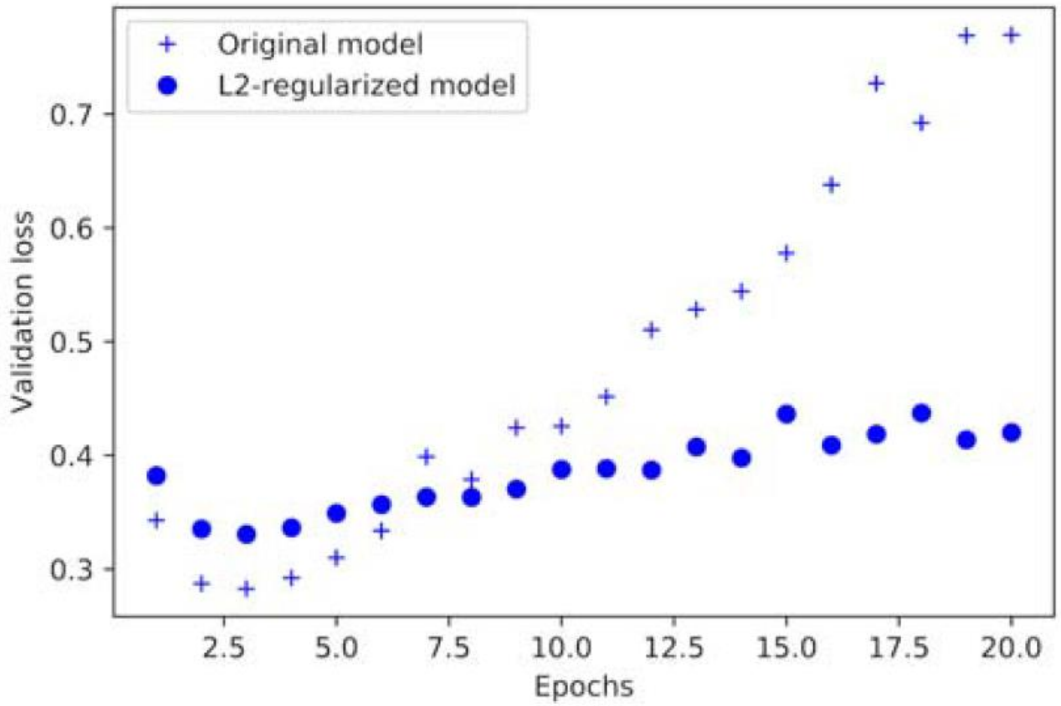




Regularization

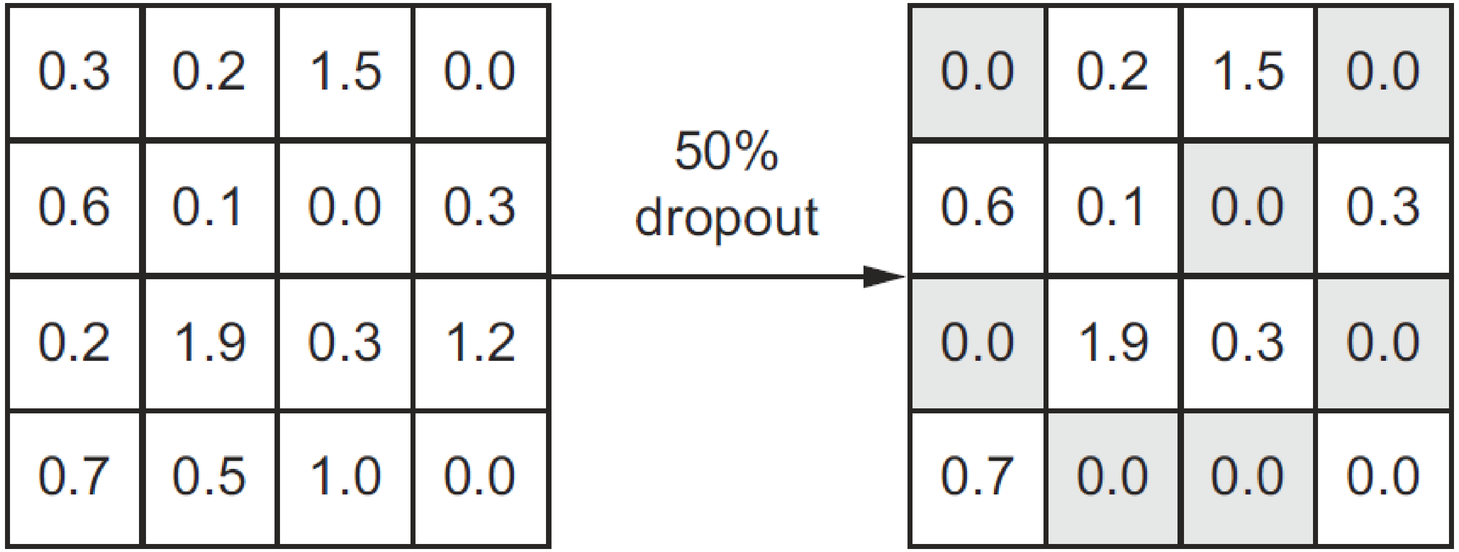
- Weight regularization – L1 and L2 norm

```
from keras import regularizers
model = models.Sequential()
model.add(layers.Dense(16,
kernel_regularizer=regularizers.l2(0.001),
       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16,
       kernel_regularizer=regularizers.l2(0.001),
       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



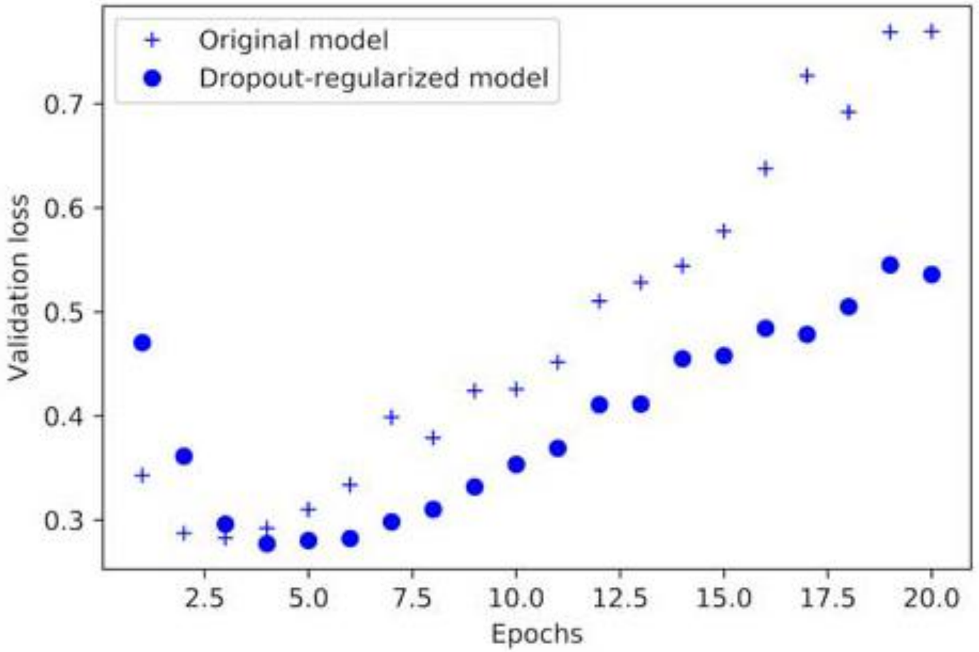
Dropout

- randomly dropping out (setting to zero) a number of output features of the layer during training
- Dropout applied to an activation matrix at training time
- At test time, the activation matrix is unchanged



Adding Dropout to the IMDB Network

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```



Machine Learning Workflow

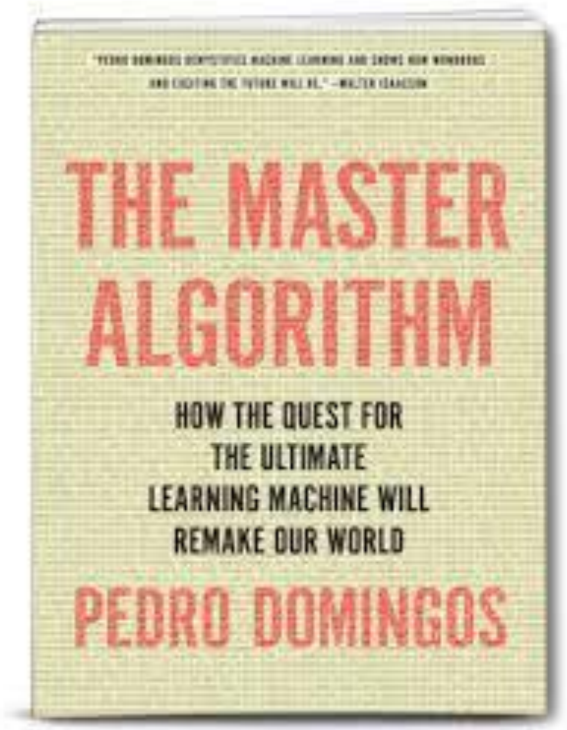
1. Defining the problem and assembling a dataset
2. Choosing a measure of success
3. Deciding on an evaluation protocol
4. Preparing your data
5. Developing a model that does better than a baseline
6. Scaling up: developing a model that overfits
7. Regularizing your model and tuning your hyperparameters

Deep Learning for Classification & Regression

- Choosing the right last-layer activation and loss function

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Pedro Domingos – Things to Know about Machine Learning



Useful Things to Know about Machine Learning

1. It's generalization that counts
2. Data alone is not enough
3. Overfitting has many faces
4. Intuition fails in high dimensions
5. Theoretical guarantees are not what they seem
6. More data beats a cleverer algorithm
7. Learn many models, not just one

Pedro Domingos, "A Few Useful Things to Know about Machine Learning," Commun. ACM, 2012

Learning = Representation + Evaluation + Optimization

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

It’s Generalization that Count

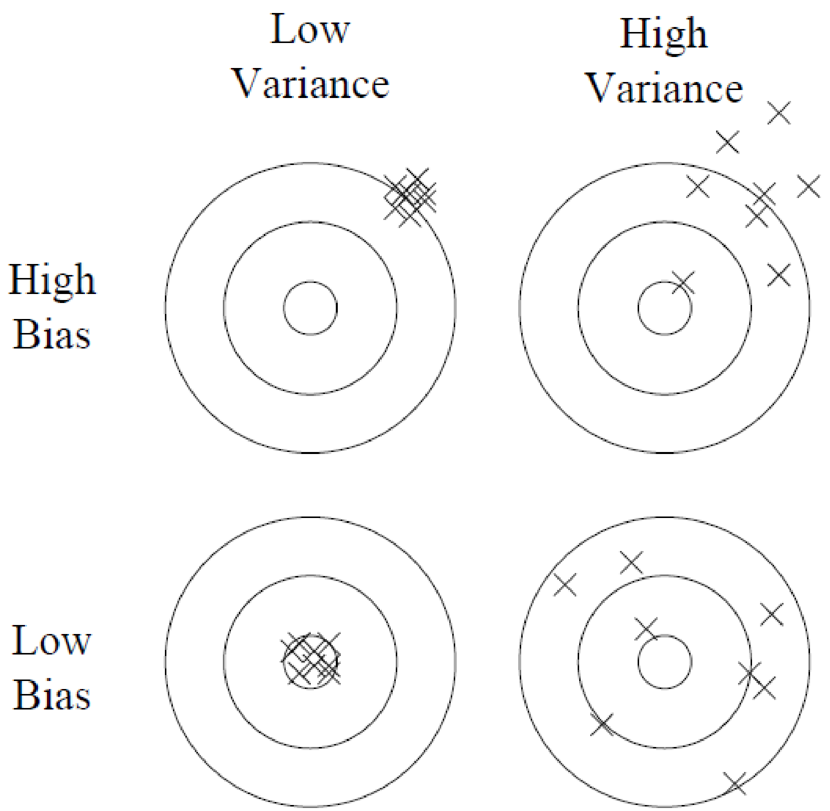
- The goal of machine learning is to *generalize* beyond the examples in the training set
- Don’t use test data for training
- Use cross validation to verify your model

Data Alone Is Not Enough

- No free lunch theorem (Wolpert)
  - Every learner must embody some knowledge or assumptions beyond the data
- Learners combine knowledge with data to grow programs

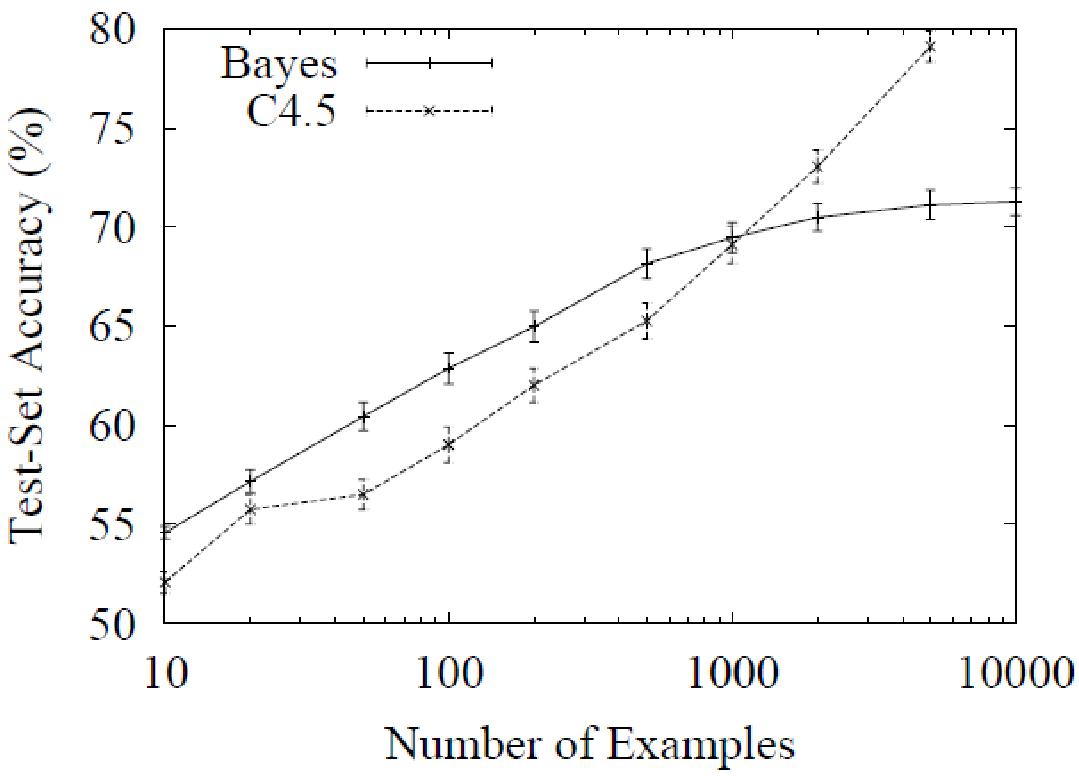
Overfitting Has Many Faces

- Overfitting – get very good results on training data but very bad results on test data
- Overfitting has many forms. Example: bias & variance
- Combat overfitting
  - Cross validation
  - Add Regularization term to avoid overfitting



Overfitting Has Many Faces – Cont'd

- Strong false assumptions can be better than weak true ones
- Example: Naive Bayes can outperform a state-of-the-art rule learner (C4.5rules) even when the true classifier is a set of rules



Intuition Fails in High Dimensions

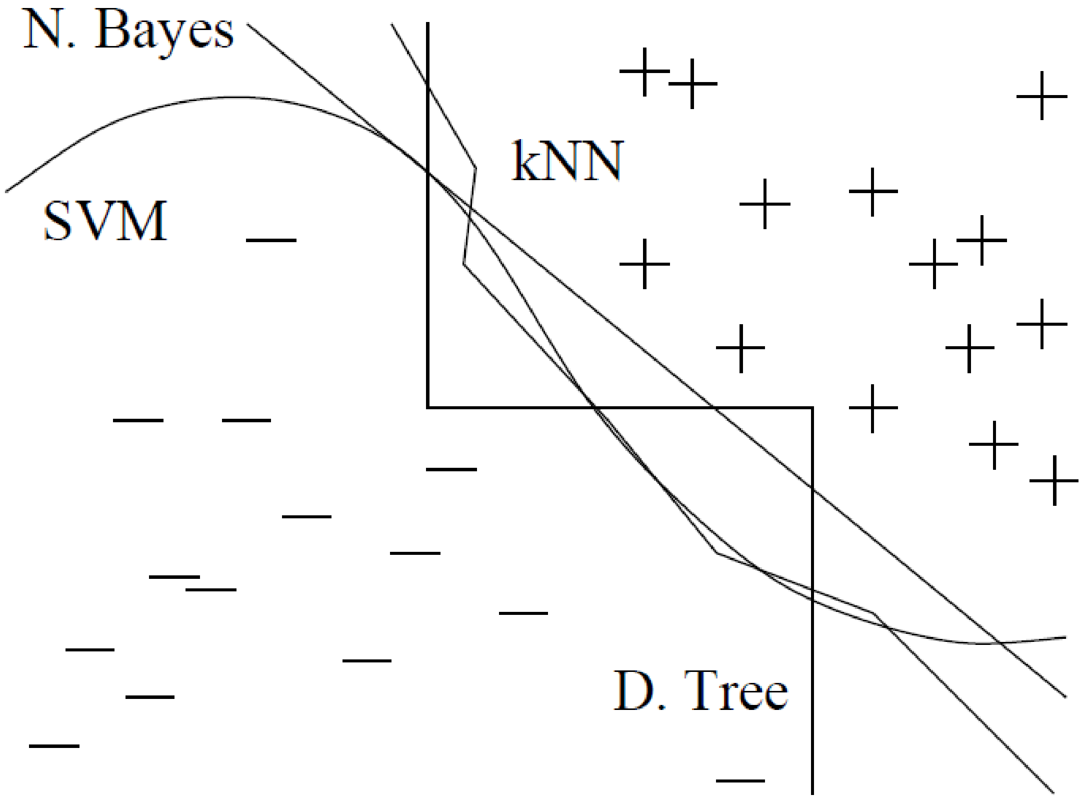
- Curse of Dimensionality
- Algorithms that work fine in low dimensions become intractable when the input is high-dimensional
- Generalizing correctly becomes exponentially harder as the dimensionality (number of features) of the examples grows
- Our intuition only comes from 3-dimension

Theoretical Guarantees Are Not What They Seem

- Theoretical bounds are usually very loose
- The main role of theoretical guarantees in machine learning is to help understand and drive force for algorithm design

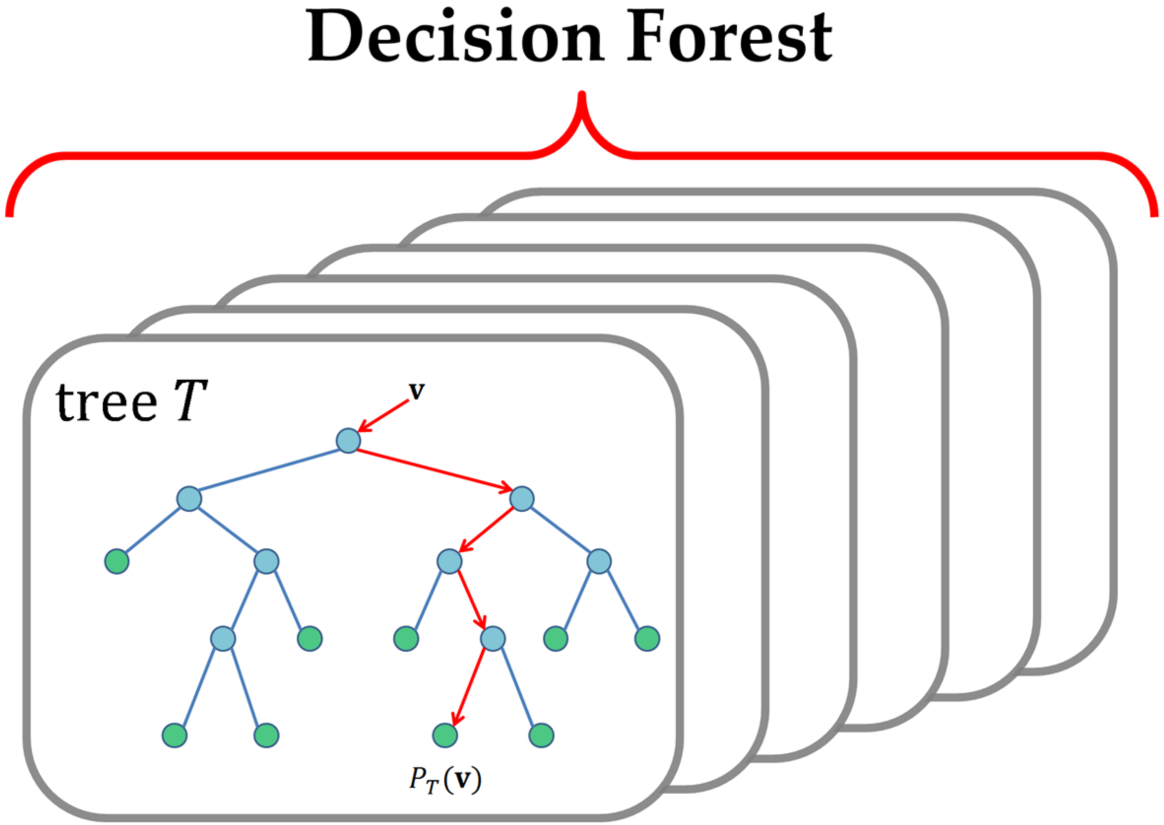
More Data Beats a Cleverer Algorithm

- Try simplest algoritm first



Learn Many Models, Not Just One

- Ensembling methods: Random Forest ,XGBoost, Late Fusion
- Combining different models can get better results



Metrics

https://en.wikipedia.org/wiki/Confusion_matrix

		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score = $\frac{1}{\frac{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}{2}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

Popular Metrics

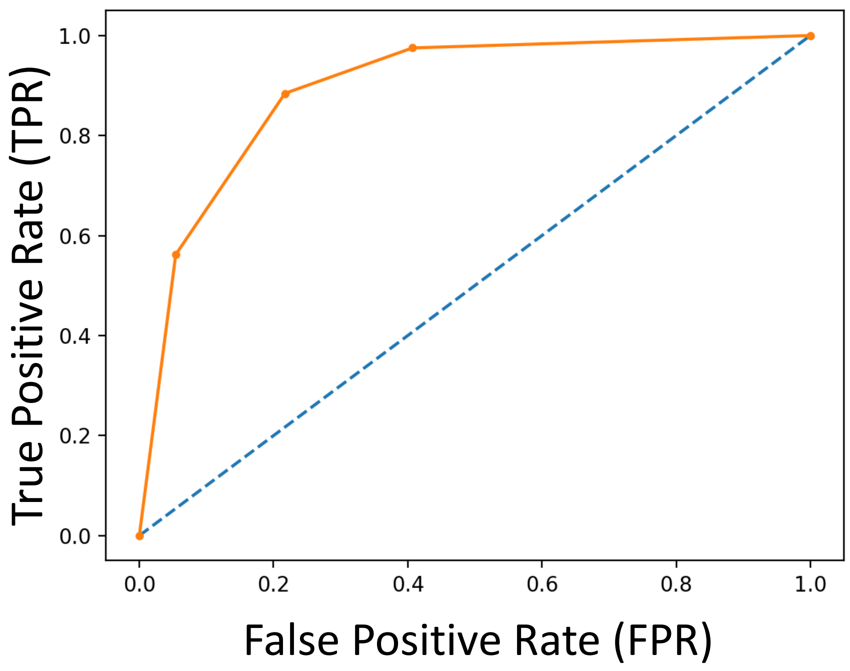
- The notations represent the number of
  - P: positive samples, N: negative samples, P': predicted positive samples, TP: true positives, TN: true negatives
- Recall = $\frac{\mathrm{TP}}{\mathrm{P}}$
- Precision = $\frac{\mathrm{TP}}{\mathrm{P'}}$
- Accuracy = $\frac{\mathrm{TP+TN}}{P+\mathrm{N}}$


Total Recall

ROC (Receiver Operating Characteristic)

- Evaluate binary classifier's ability
- Plot the true positive rate (TPR) against the false positive rate (FPR) at various thresholds (decision boundaries)
- Use area under curve (AUC) to evaluate performance

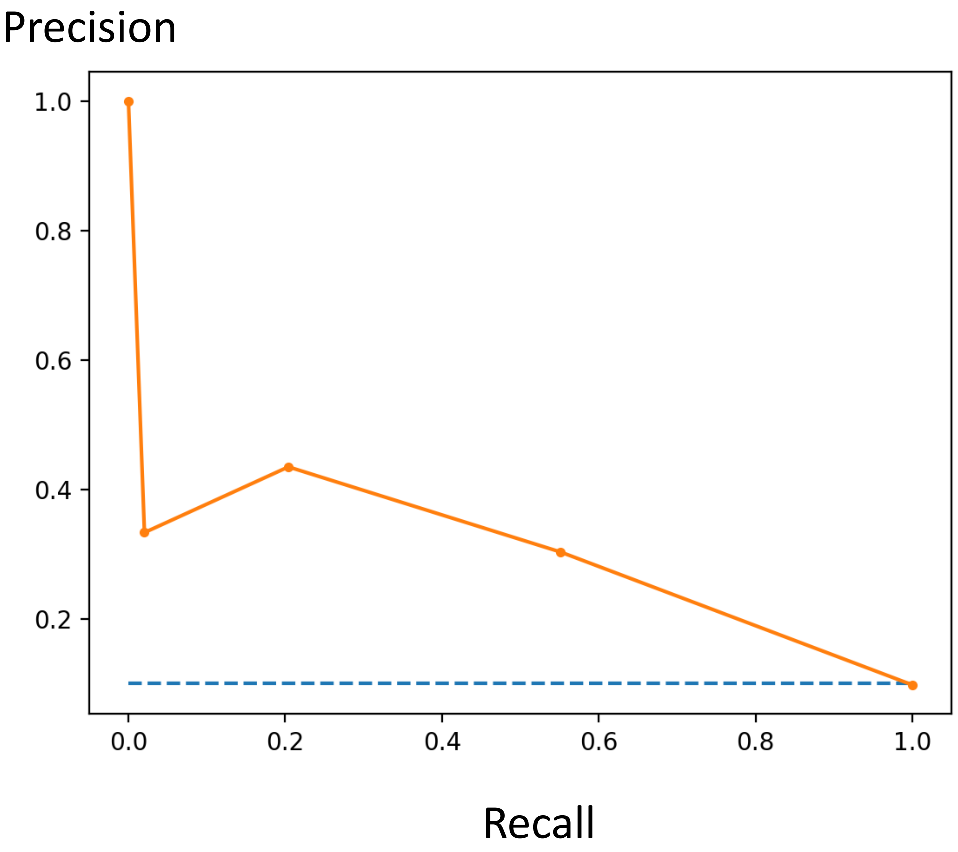
```python
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, weights=[1,1],
random_state=1)
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
model = KNeighborsClassifier(n_neighbors=3)
model.fit(trainX, trainy)
probs = model.predict_proba(testX) # predict probabilities
probs = probs[:, 1] # keep probabilities for the positive outcome only
# calculate AUC
auc = roc_auc_score(testy, probs)
print('AUC: %.3f' % auc)
fpr, tpr, thresholds = roc_curve(testy, probs) # calculate roc curve
pyplot.plot([0, 1], [0, 1], linestyle='--')
pyplot.plot(fpr, tpr, marker='.')
pyplot.show()
```



Precision-Recall (PR) Curve

- Plot Precision vs. Recall
- Popular in information retrieval

```
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2,
weights=[0.9,0.09], random_state=1)
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5,
random_state=2)
# fit a model
model = KNeighborsClassifier(n_neighbors=3)
model.fit(trainX, trainy)
probs = model.predict_proba(testX)[:, 1]
# predict class values
yhat = model.predict(testX)
# Calculate precision recall curve
precision, recall, thresholds = precision_recall_curve(testy, probs)
pyplot.plot(recall, precision, marker='.')
pyplot.show()
```



ROC Curve & PR Curve

- ROC curves should be used when there are roughly equal numbers of observations for each class
- Precision-Recall curves should be used when the data are imbalance and we only care about the positive class
- Note! ROC may be harmful
  - ROC curves can present an overly optimistic view of an algorithm's performance if there is a large skew in the class distribution

References

- Francois Chollet, “Deep Learning with Python”, Chapter 4
- Pedro Domingos, “A Few Useful Things to Know about Machine Learning,” Commun. ACM, 2012
- https://ml-cheatsheet.readthedocs.io/en/latest/index.html
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
- https://towardsdatascience.com/data-types-in-statistics-347e152e8bee
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier