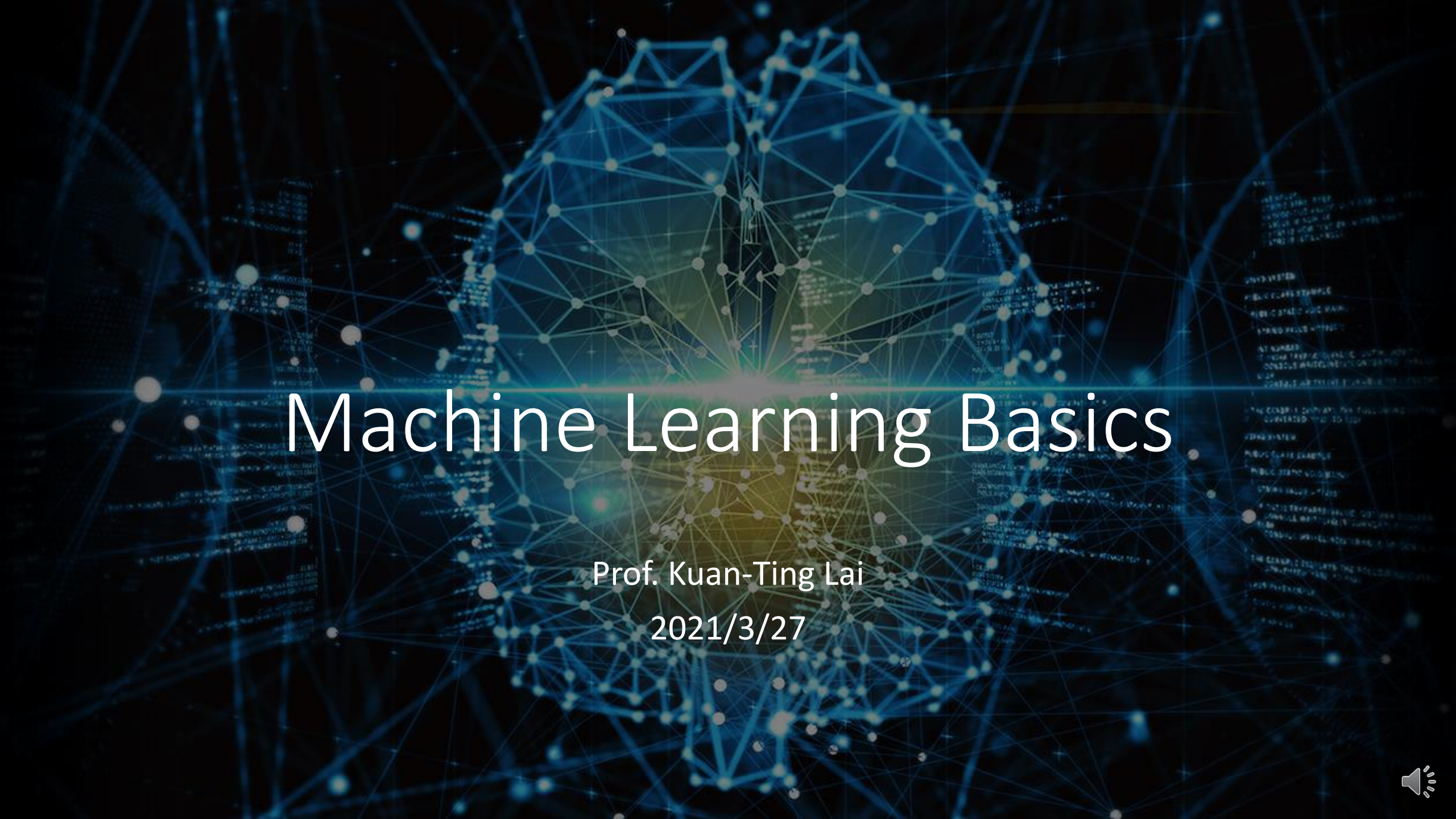
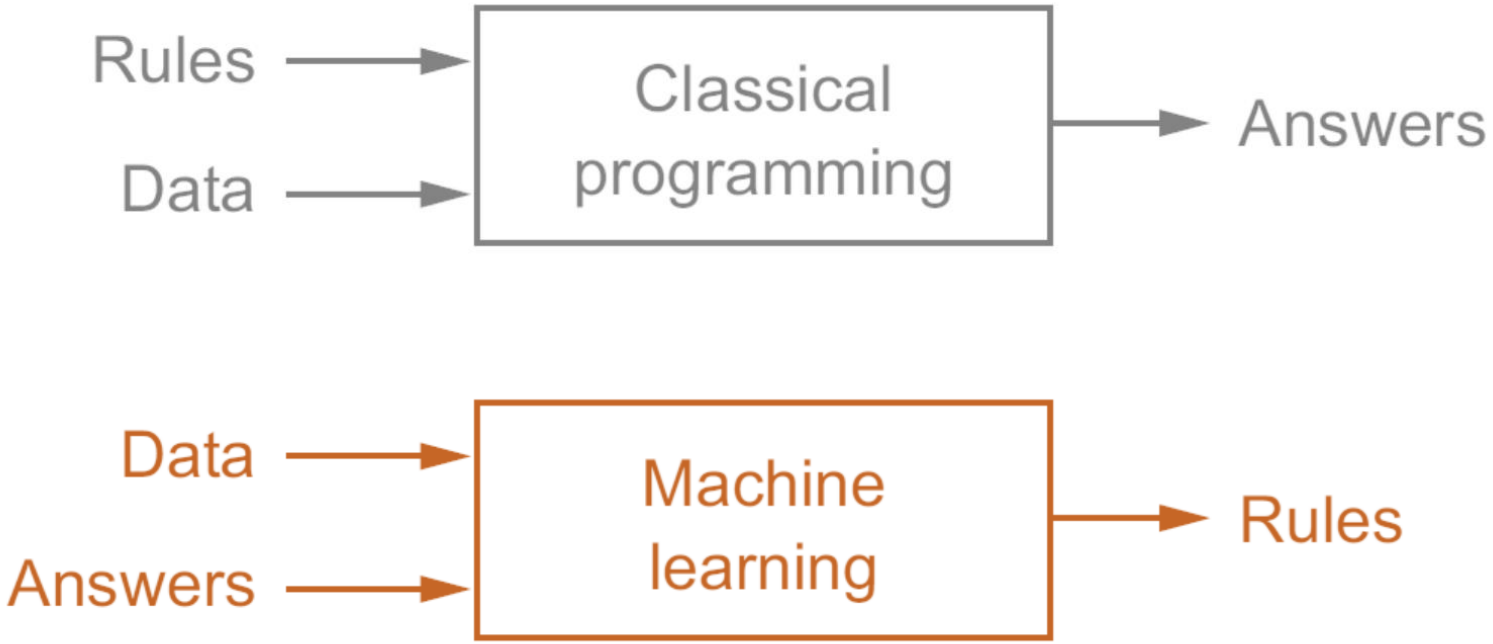


Machine Learning Basics

Prof. Kuan-Ting Lai

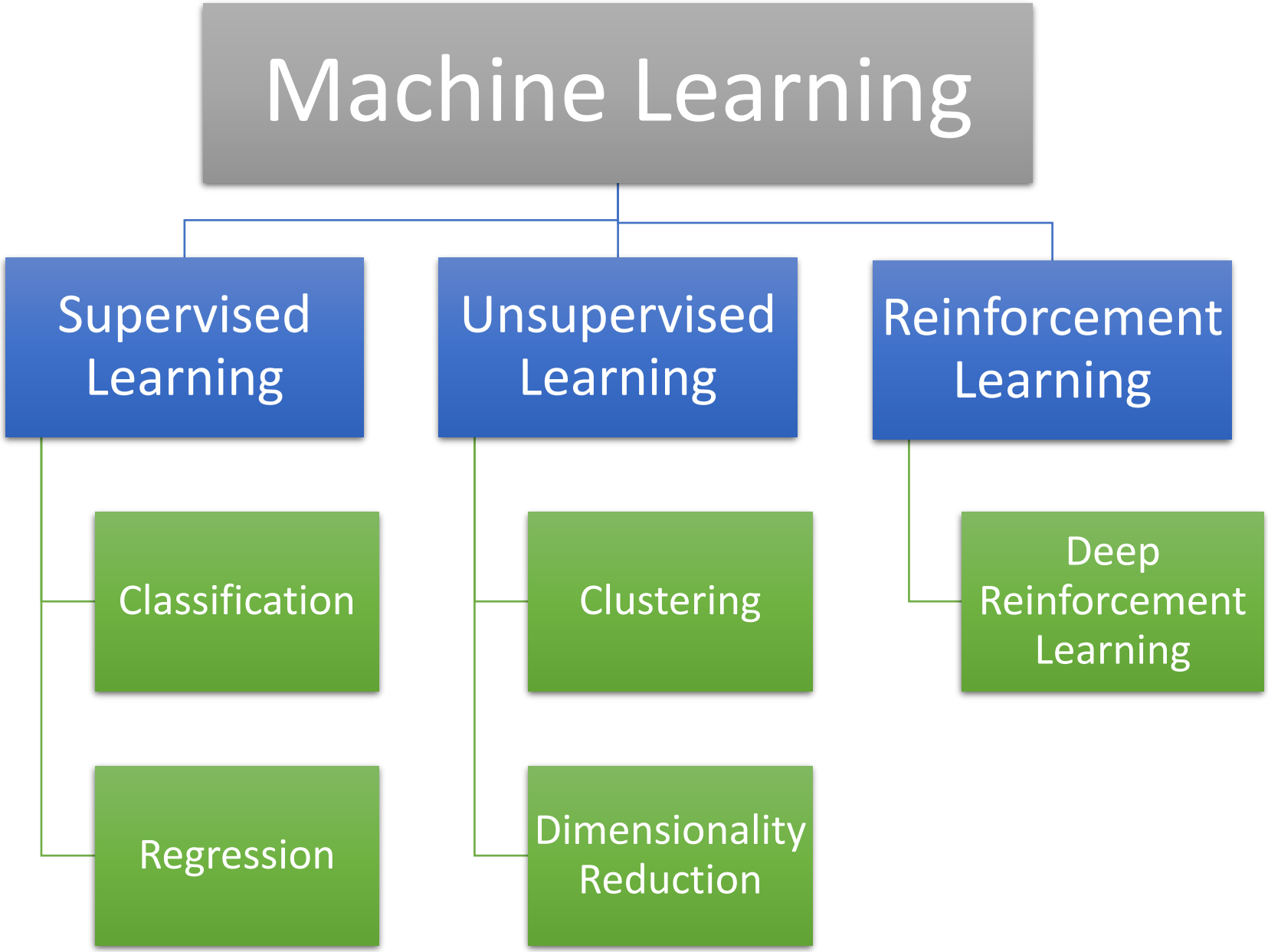
2021/3/27

Machine Learning



Francois Chollet, “Deep Learning with Python,” Manning, 2017

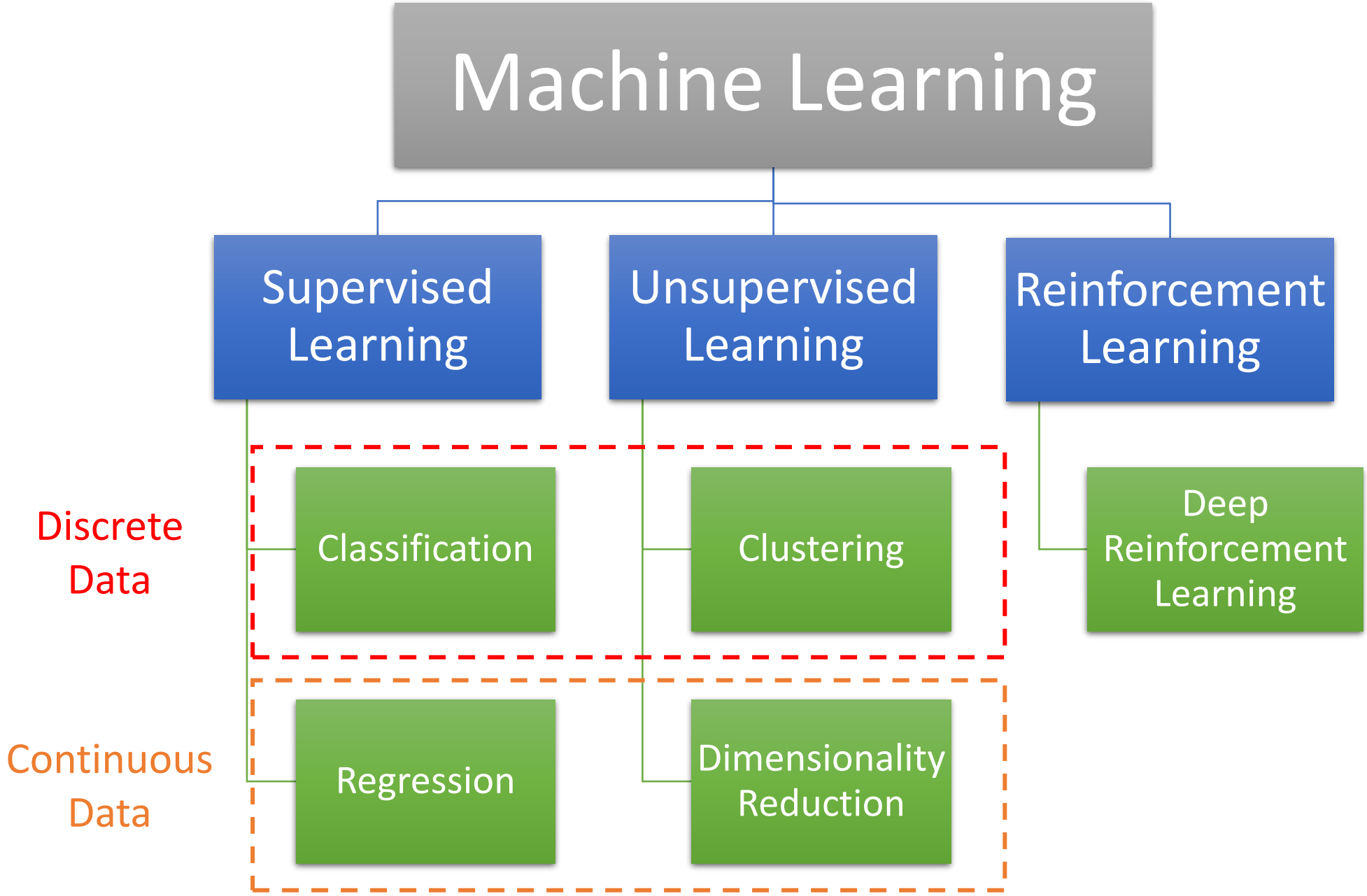
Machine Learning Flow

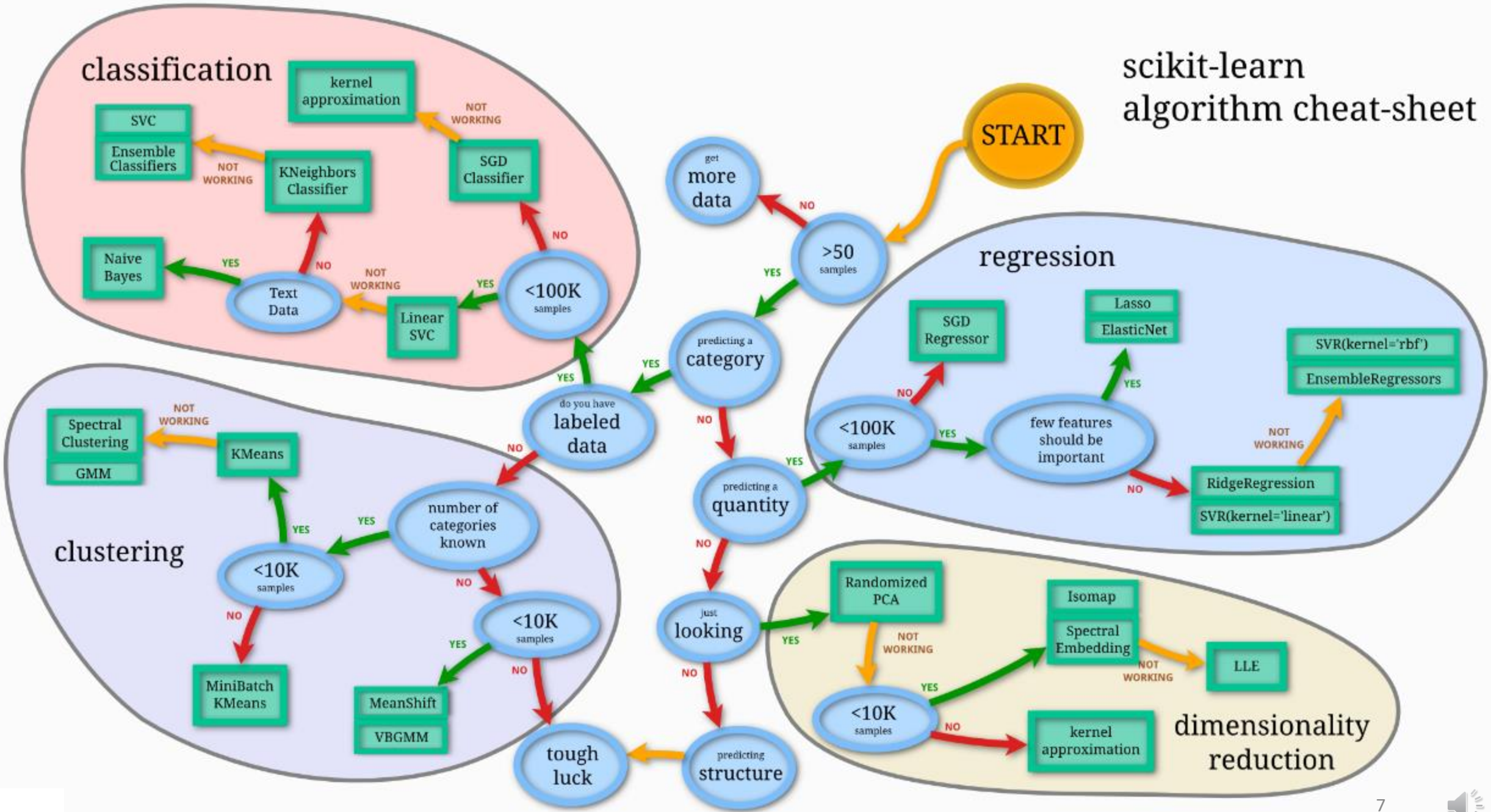

Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Classification
Regression
Clustering
Dimensionality Reduction
Deep Reinforcement Learning

Machine Learning

Has a teacher to label data!

- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
  - Clustering
  - Dimensionality Reduction
- Reinforcement Learning
  - Deep Reinforcement Learning


Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Discrete Data
Classification
Clustering
Deep Reinforcement Learning
Continuous Data
Regression
Dimensionality Reduction

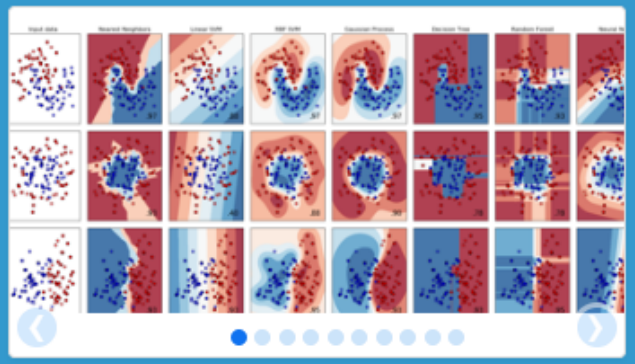

scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
YES
Linear SVC
NOT WORKING
Text Data
YES
Naive Bayes
NO
KNeighbors Classifier
NOT WORKING
Ensemble Classifiers
SVC
NOT WORKING
kernel approximation
NO
SGD Classifier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

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.

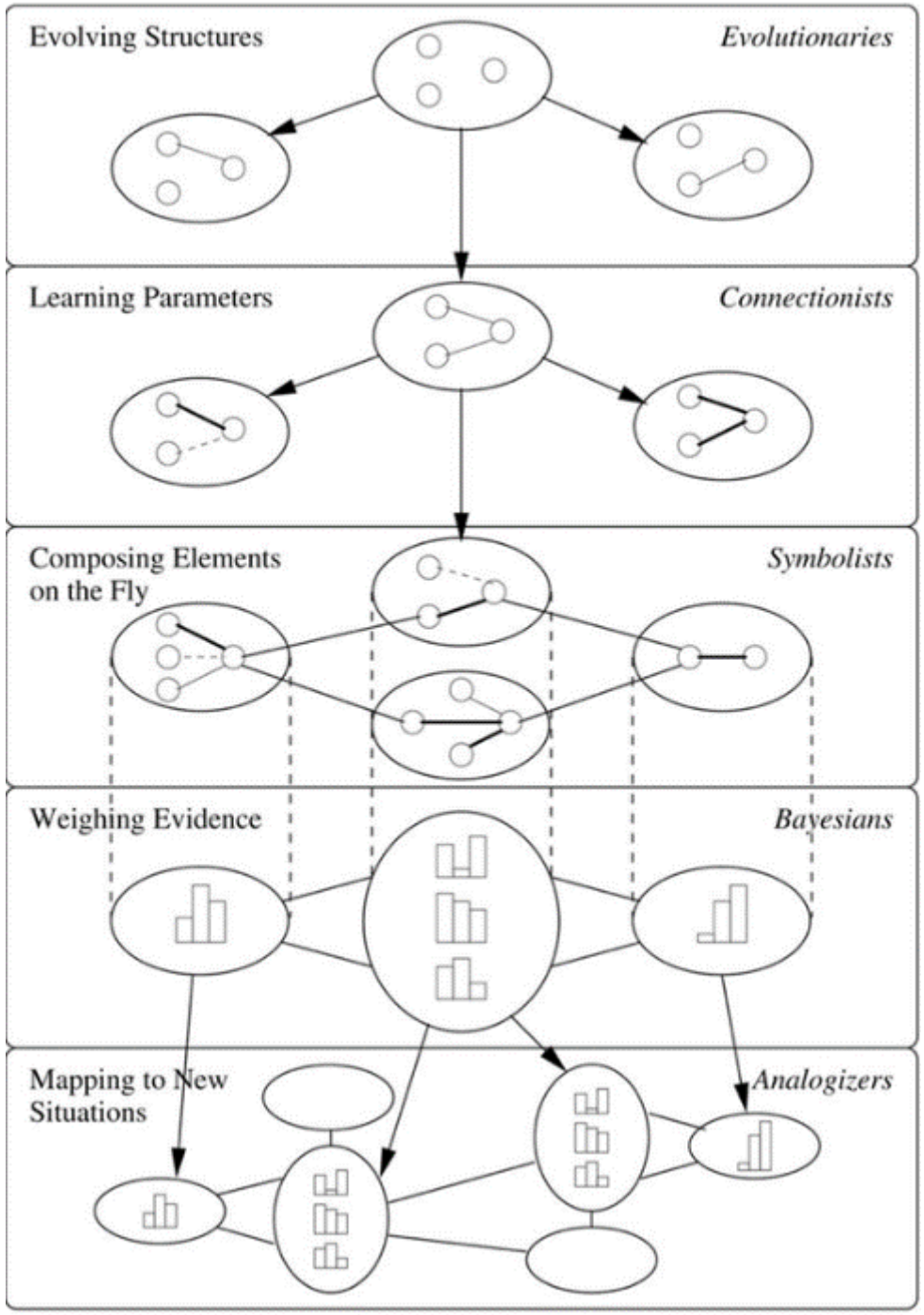
Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

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

— Examples



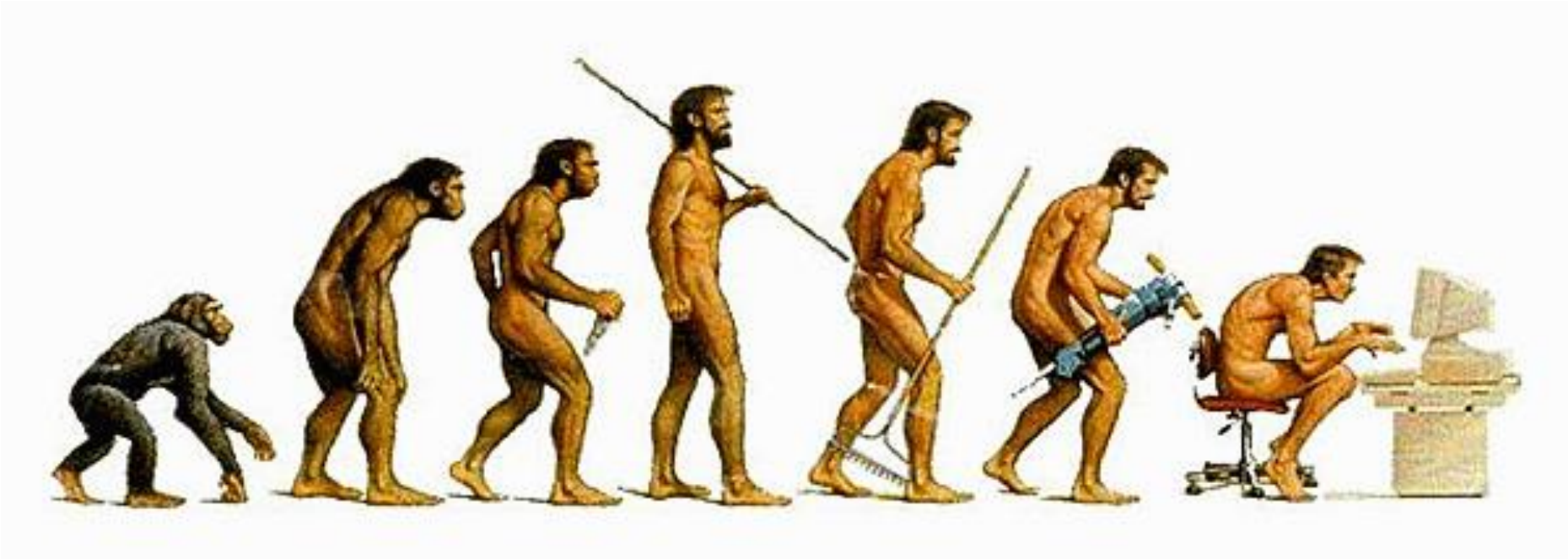
Five Tribes of Machine Learning

- Evolutionaries
- Connectionists
- Symbolists
- Bayesians
- Analogizers

Pedro Domingos, “The Master Algorithm,” 2015

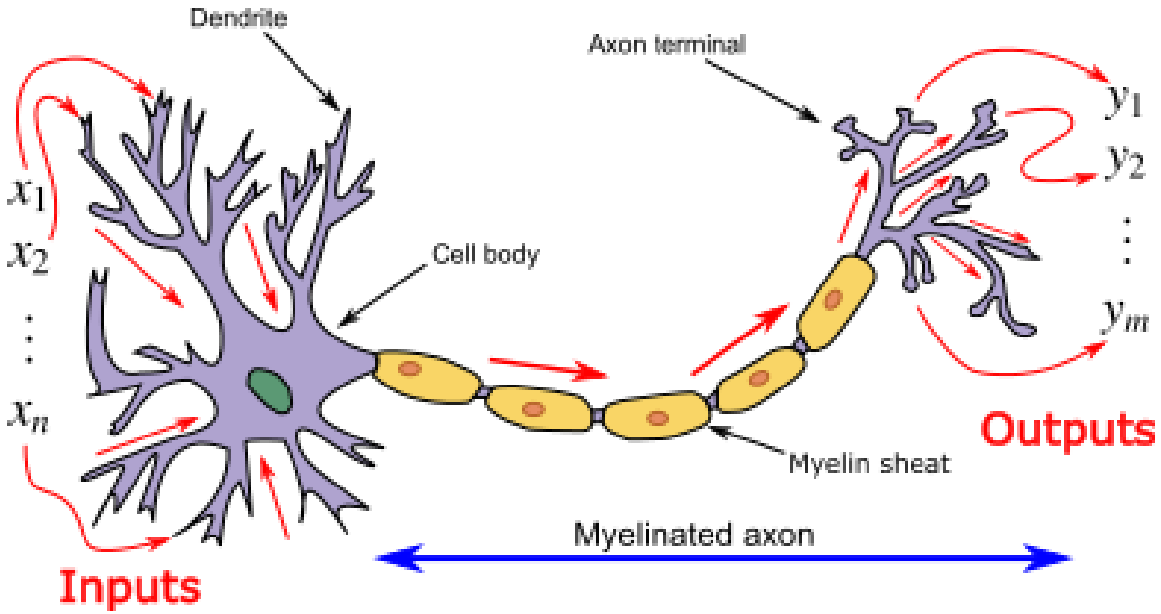
Evolutionaries: Survival of the Fittest

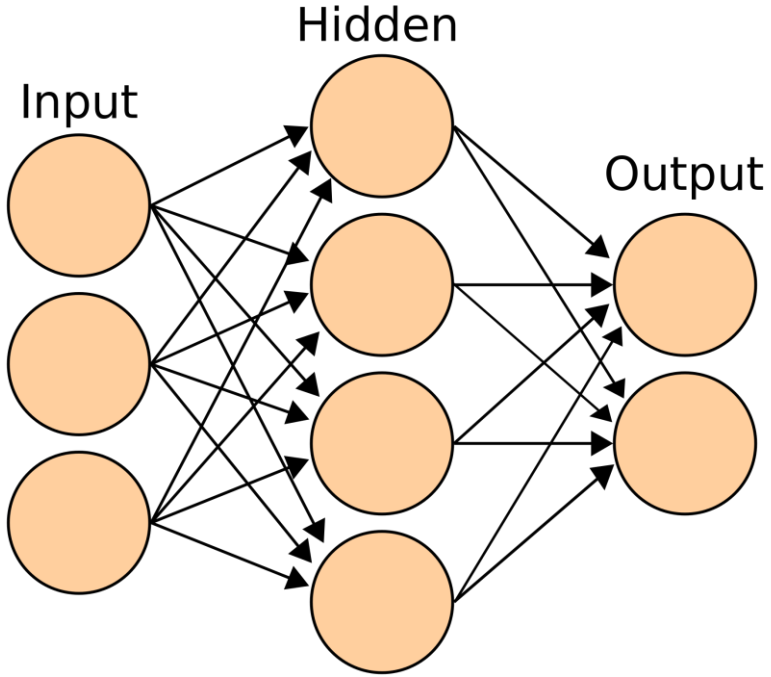
- Genetic algorithms, evolutionary algorithms
  - Crossover, mutation



Connectionists: Neural Networks

- Neural Networks, deep learning



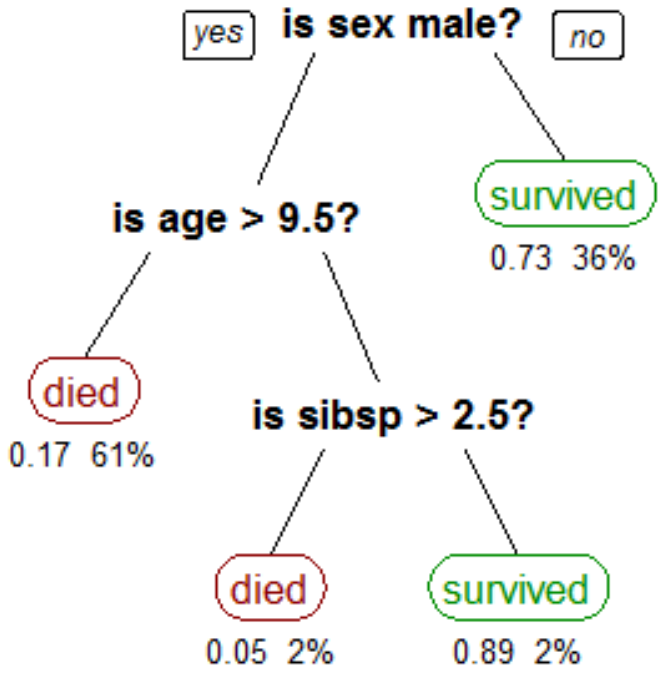


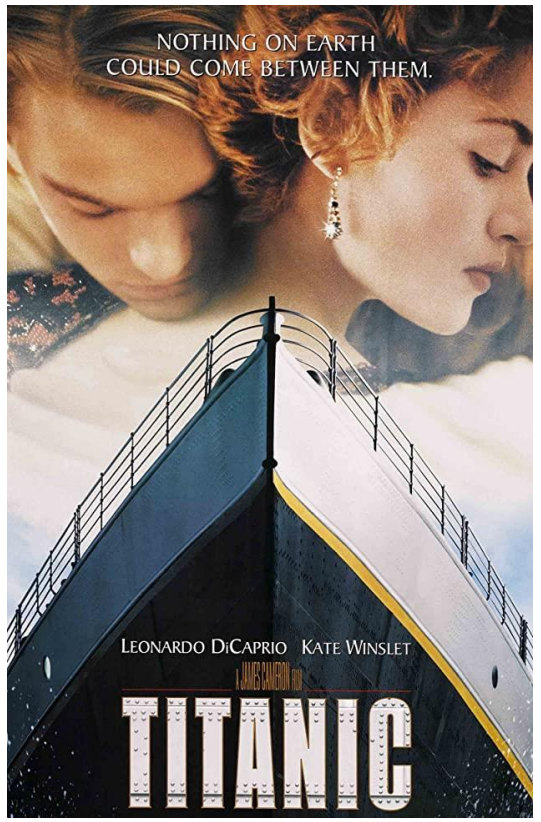
https://en.wikipedia.org/wiki/Artificial_neural_network

Symbolists: Finding the Rules

- Decision Tree, expert system, rule-based system

Survival rate of Passengers on Titanic

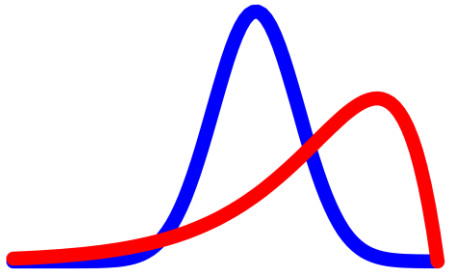


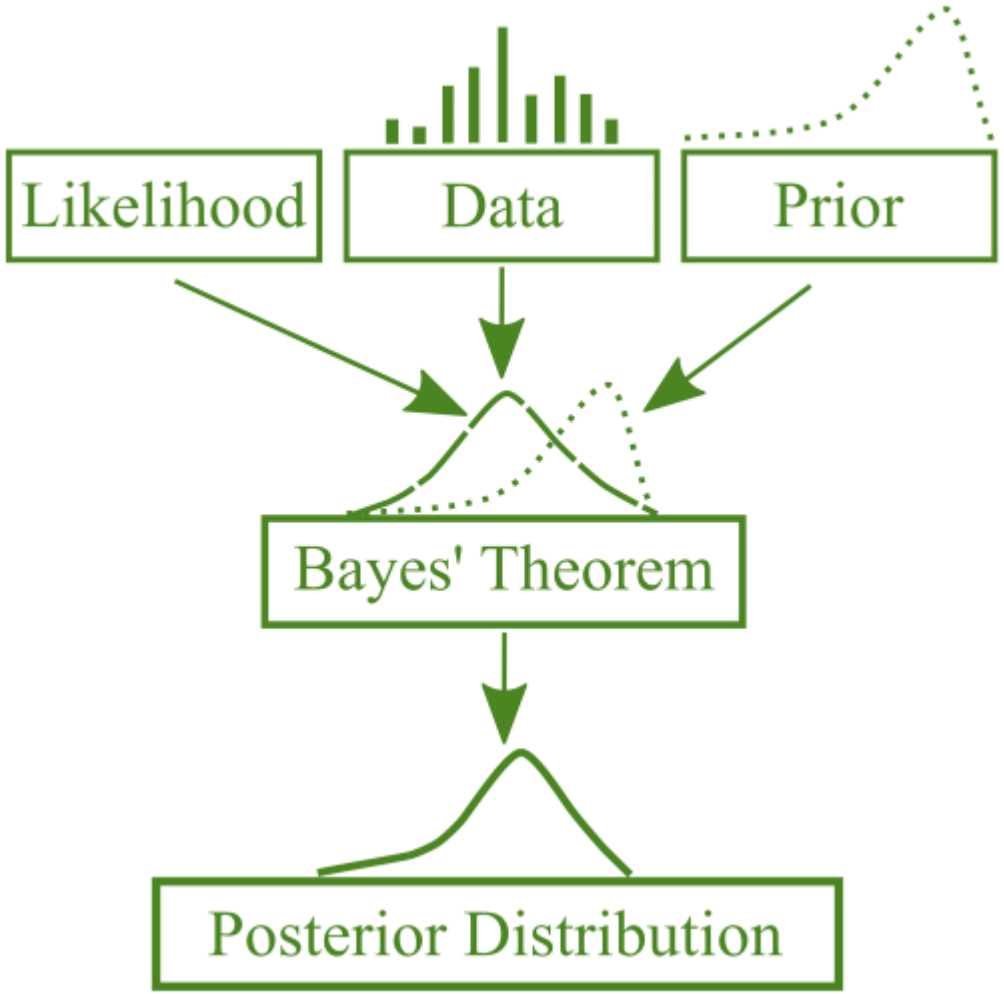


Bayesians: Anything happens, just the Probability

- *Bayes' theorem*

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y)p(y)}{P(\boldsymbol{x})}$$

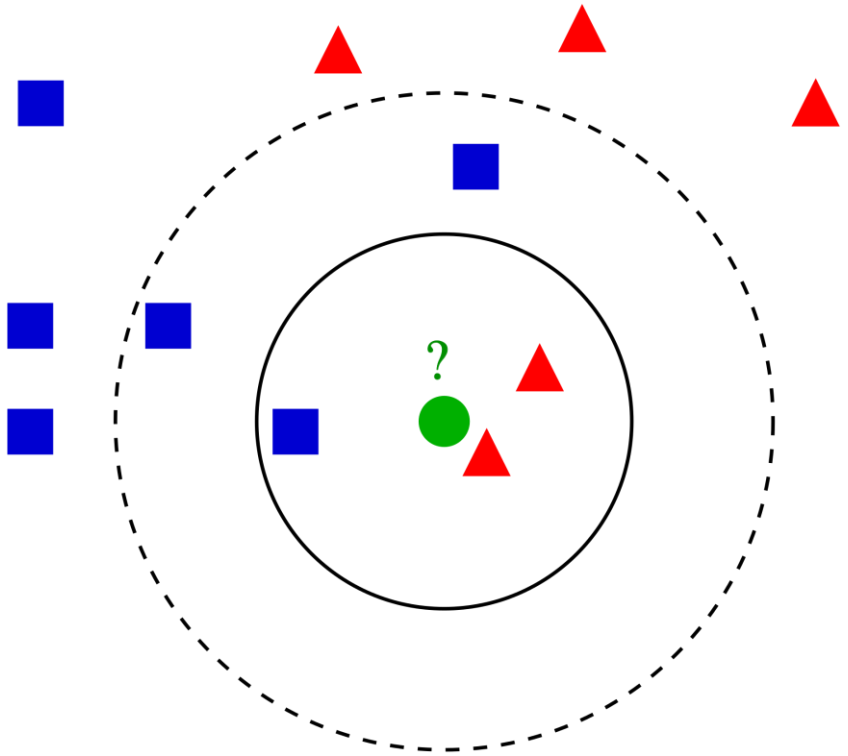




https://medium.com/@rathi.ankit/bayesian-statistics-for-data-science-45397ec79c94

Analogizers: Like Father Like Son

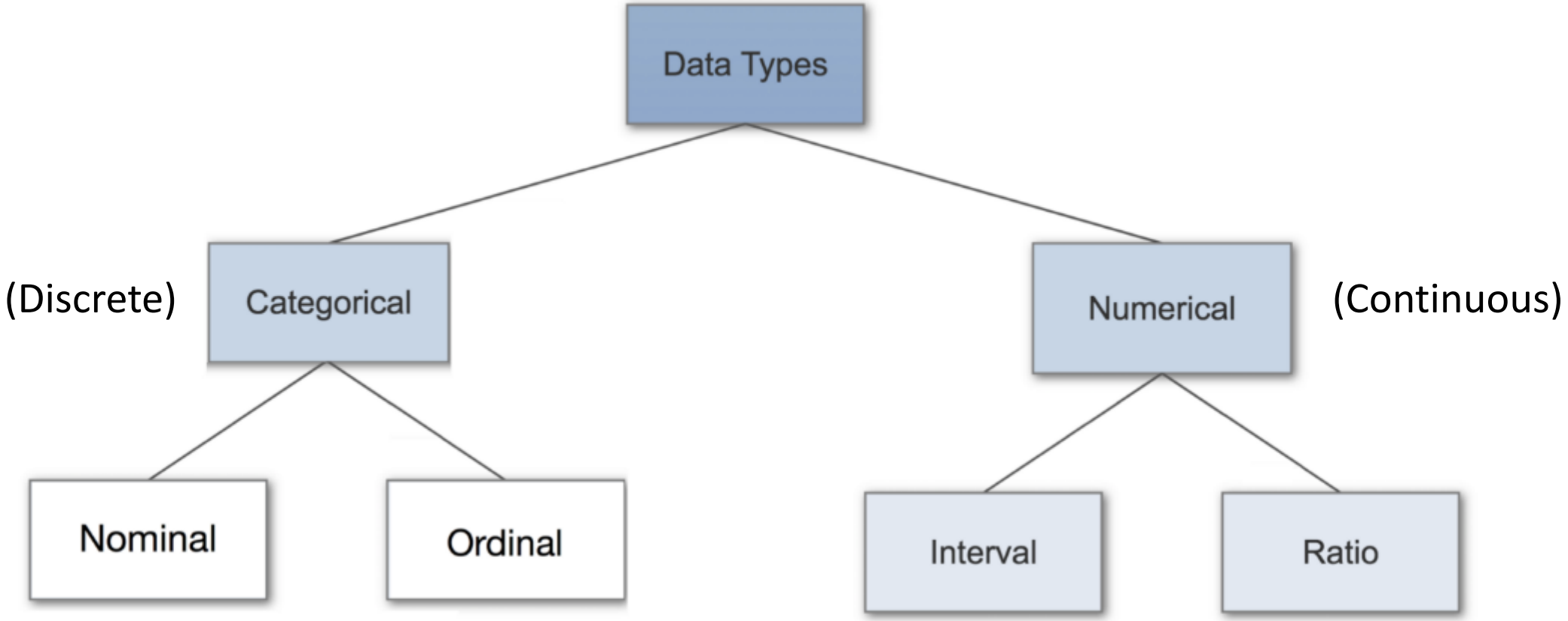
- k Nearest-Neighbors (kNN), SVM



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Types of Data

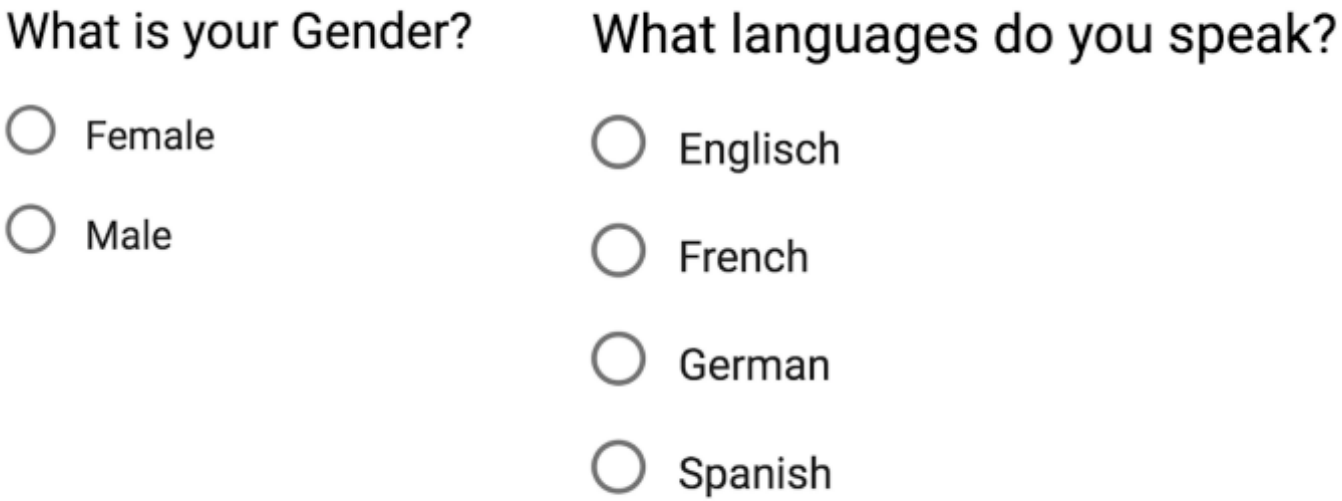
Data Types (Measurement Scales)



https://towardsdatascience.com/data-types-in-statistics-347e152e8bee

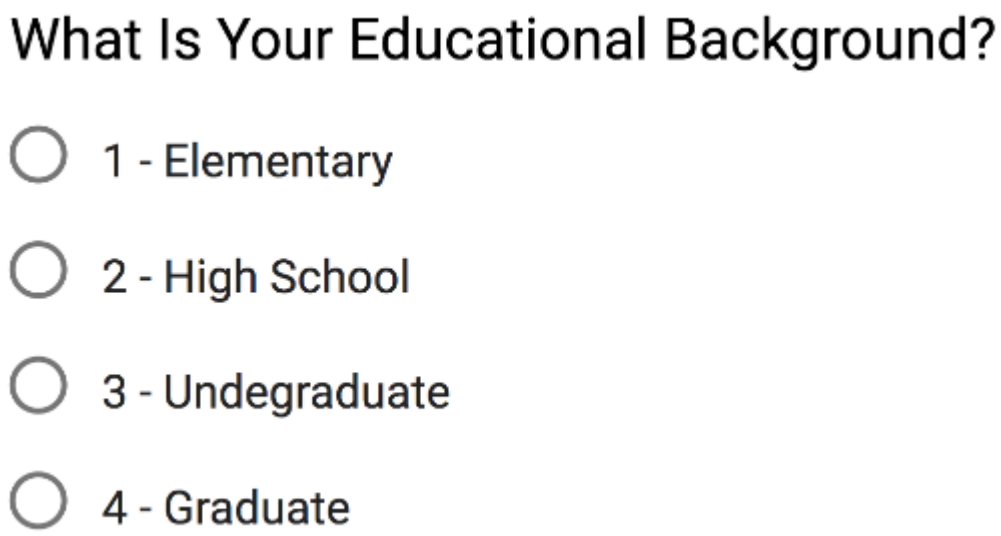
Nominal Data (Labels)

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



Ordinal Data

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



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 Data

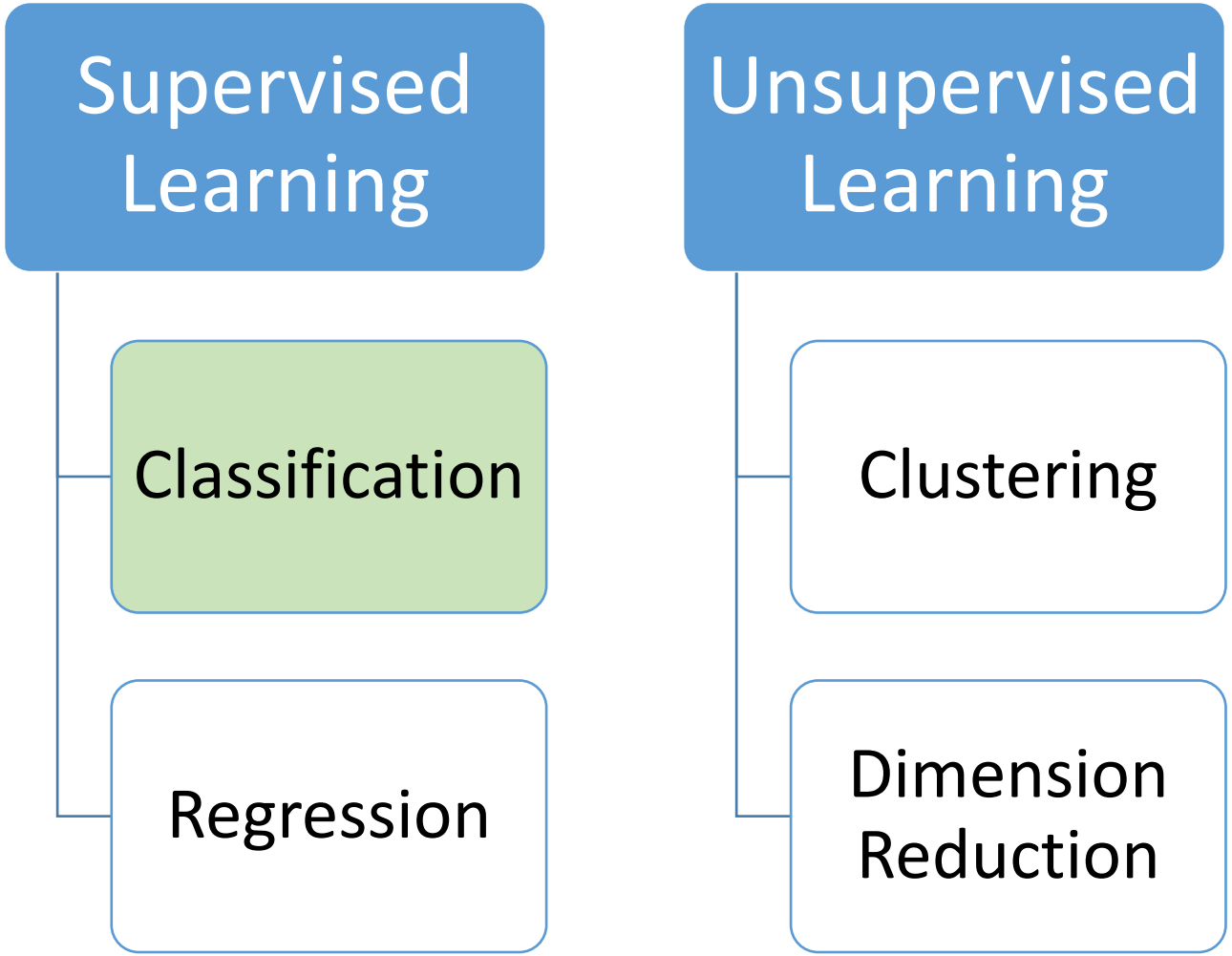
- Same as interval data but have absolute zero
- Can be applied to both descriptive and inferential statistics
- Example: weight & height

Machine Learning vs. Statistics

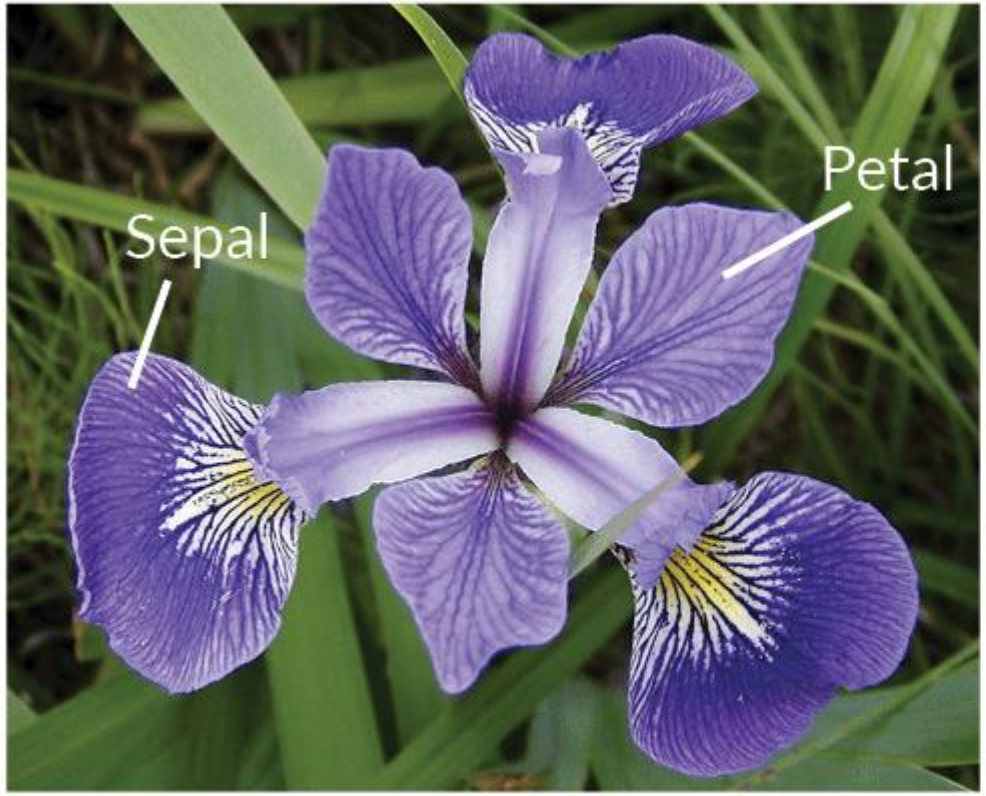
- https://www.r-bloggers.com/whats-the-difference-between-machine-learning-statistics-and-data-mining/

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

Supervised and Unsupervised Learning



Iris Flower Classification



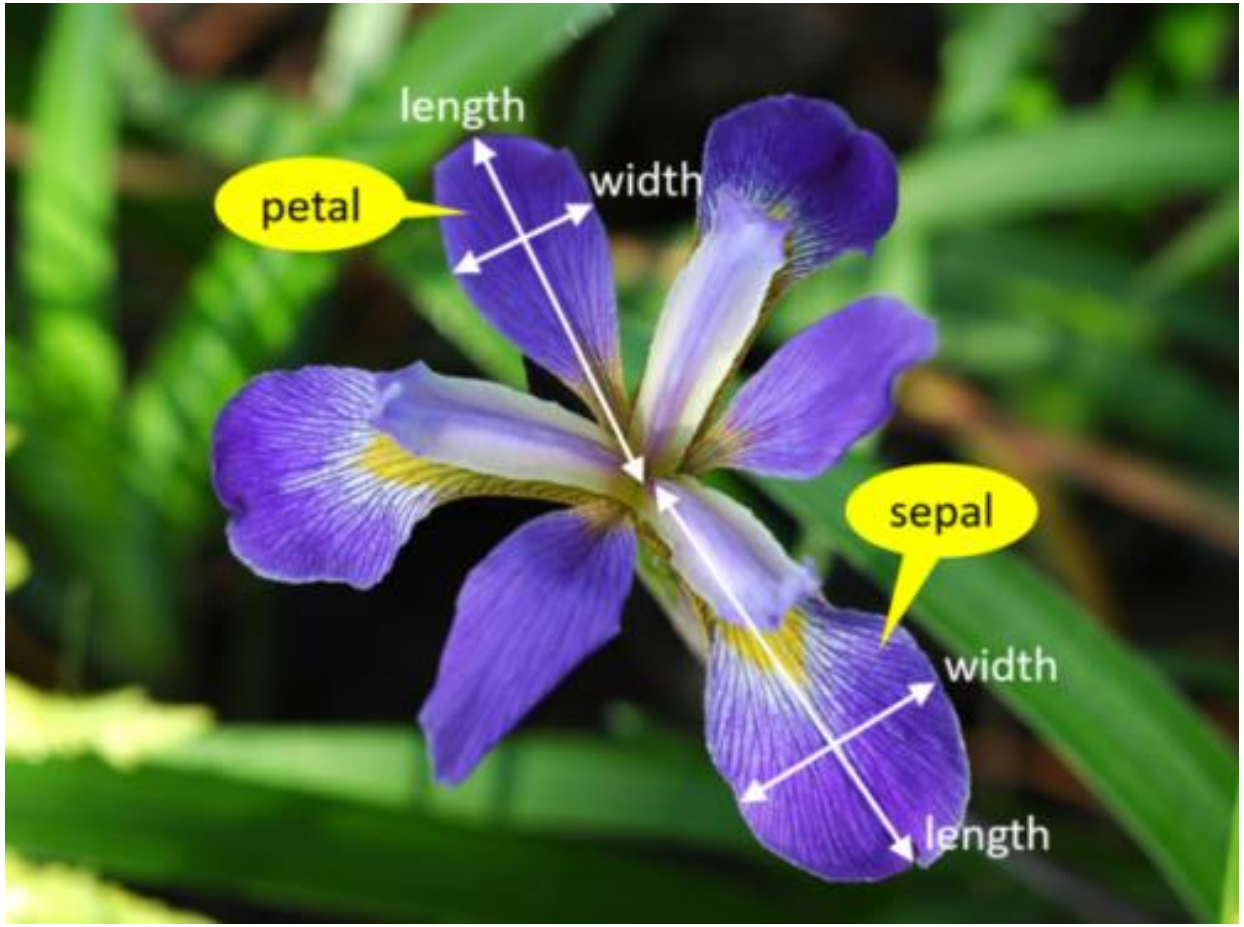
Iris Versicolor

Iris Setosa

Iris Virginica

Extracting Features of Iris

- Width and Length of Petal and Sepal



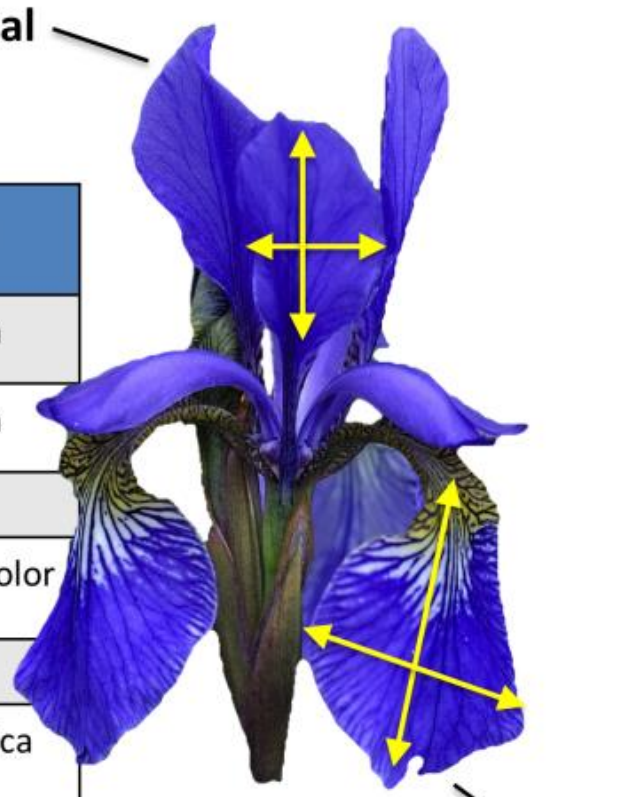
Iris Flower Dataset

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Features
(attributes, measurements, dimensions)

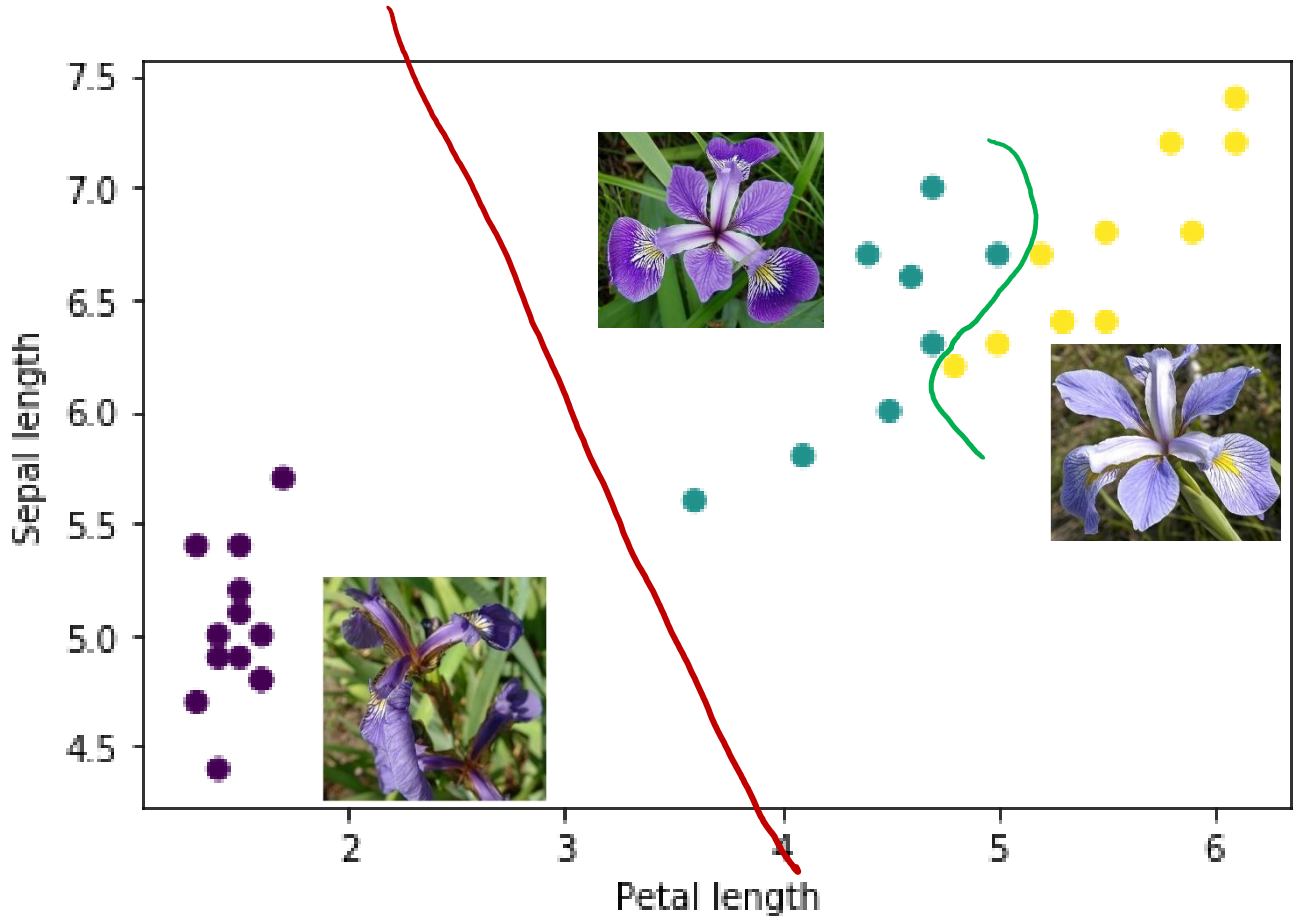
Class labels
(targets)



[Jebaseelan Ravi @ Medium]

Classify Iris Species via Petals and Sepals

- Iris versicolor and virginica are not linearly separable



https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough

Binary Linear Classifier

$$w_1 x_1 + w_2 x_2 + b = 0$$

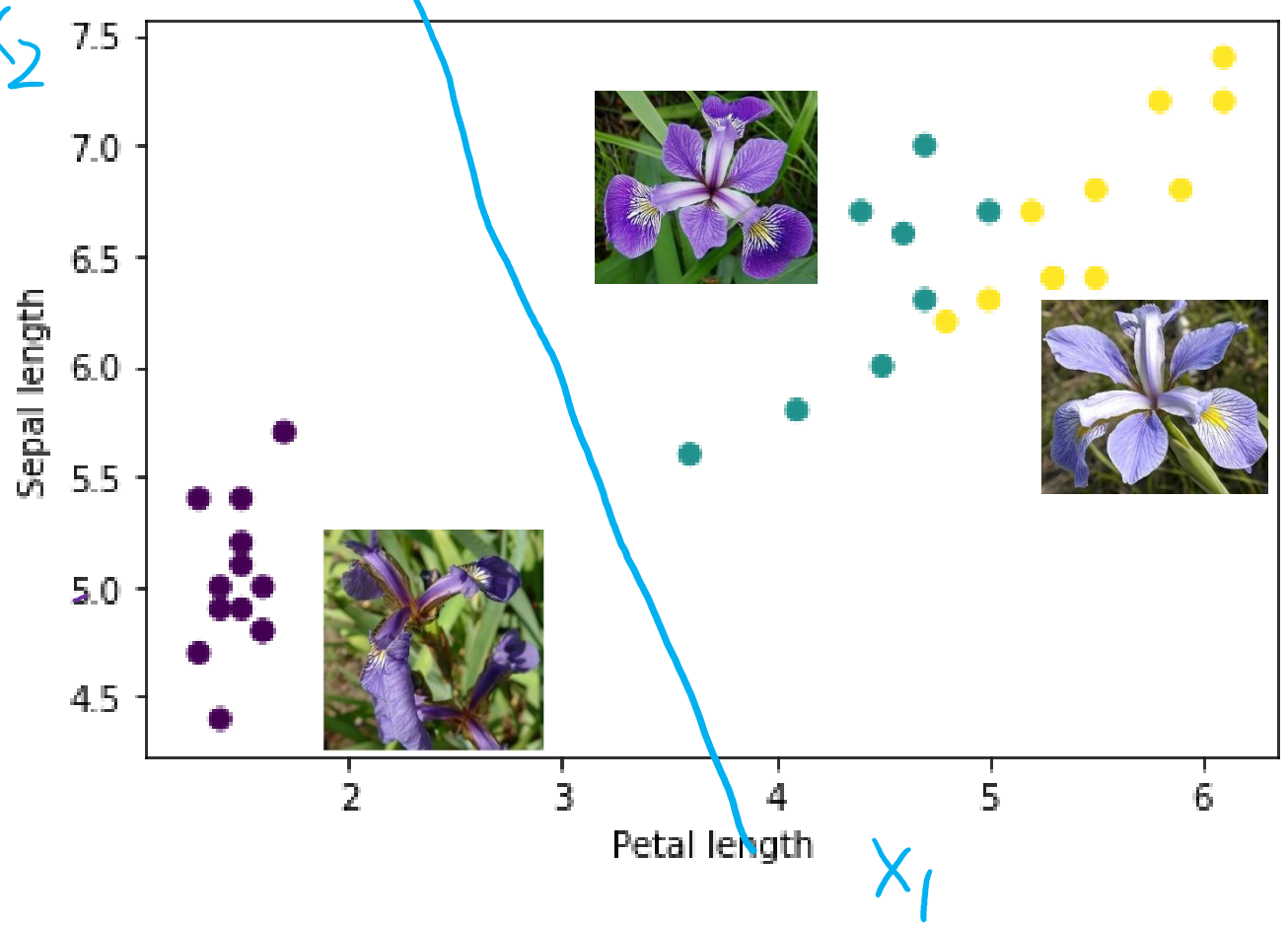
$$\Downarrow$$

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = 0$$

$$y' = \mathbf{w}^T \mathbf{x} + b$$

$$y' = w^T x + b \leq 0$$

$$w^T x + b > 0$$



Training Linear Classifier

- Perceptron

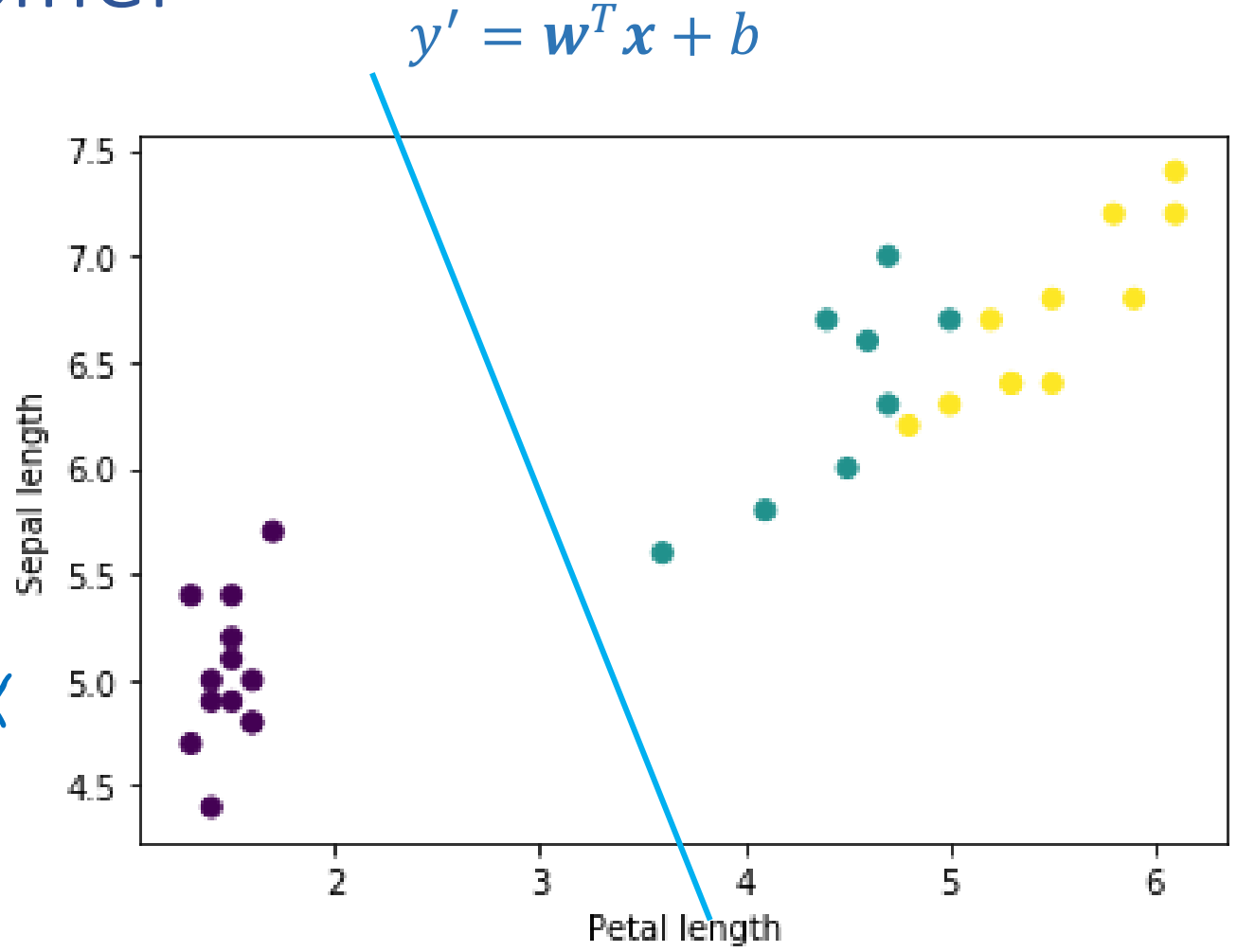
$$\ell = (y - y')^2$$

$$\nabla \ell = \Delta = y - y'$$

Loop {

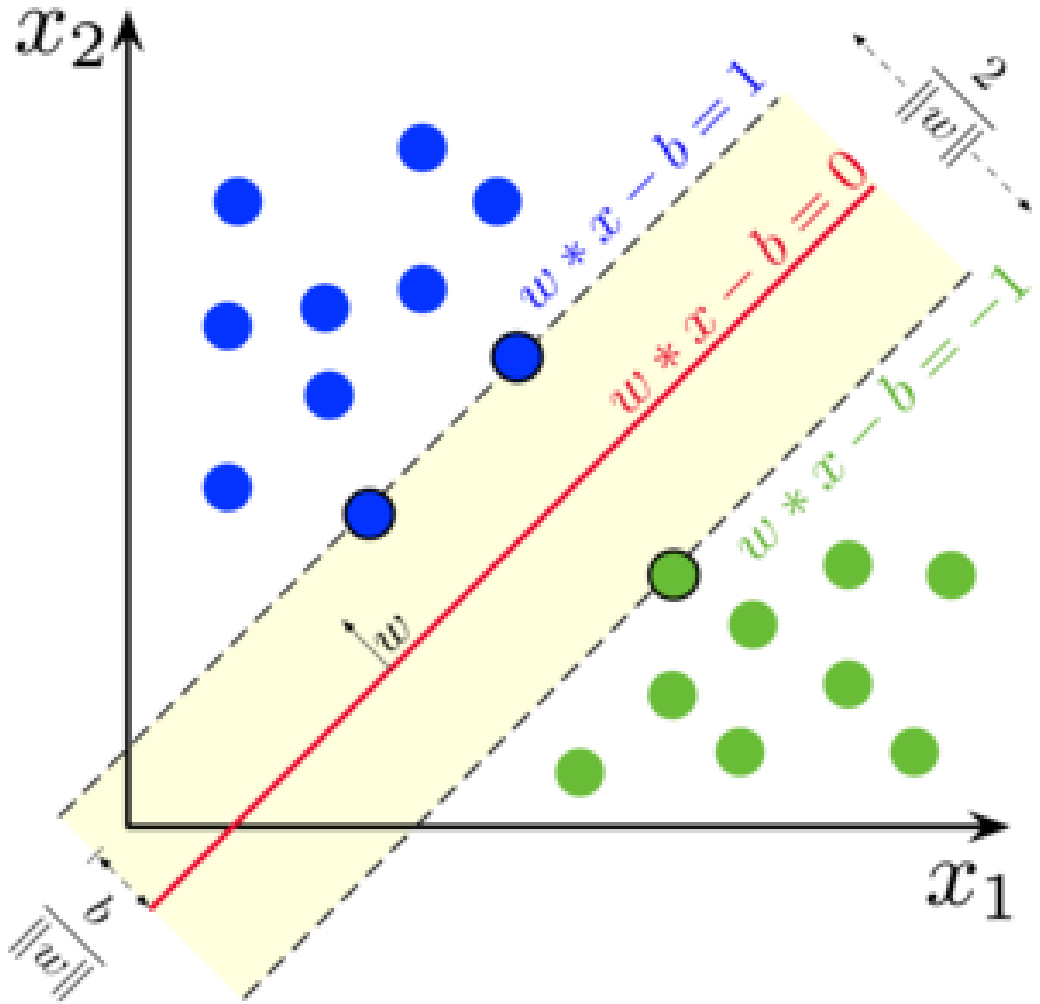
$$W_{t+1} \leftarrow W_t + \alpha (y - y') x$$

} Until $\Delta \approx 0$



Support Vector Machine (SVM)

- Choose the hyperplanes that have the largest separation (margin)



Loss Function of SVM

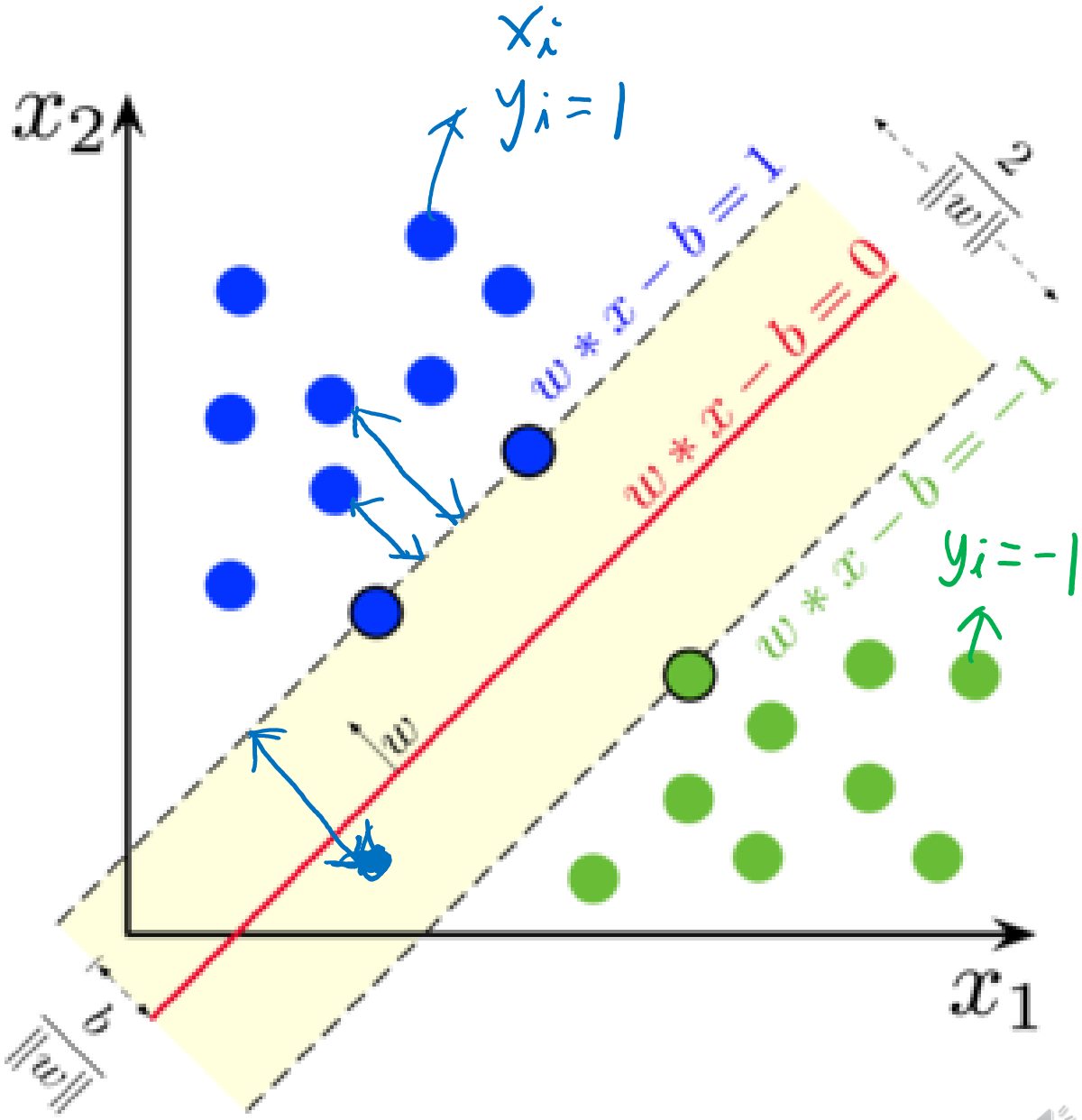
- Hinge Loss

$$y_i' = w^T x_i - b \geq 1$$
$$y_i' = w^T x_i - b \leq -1$$

$$y_i(w^T x_i - b) \geq 1$$

$$loss = \max(0, 1 - y_i(w^T x_i - b))$$

Hinge Loss



SVM Optimization

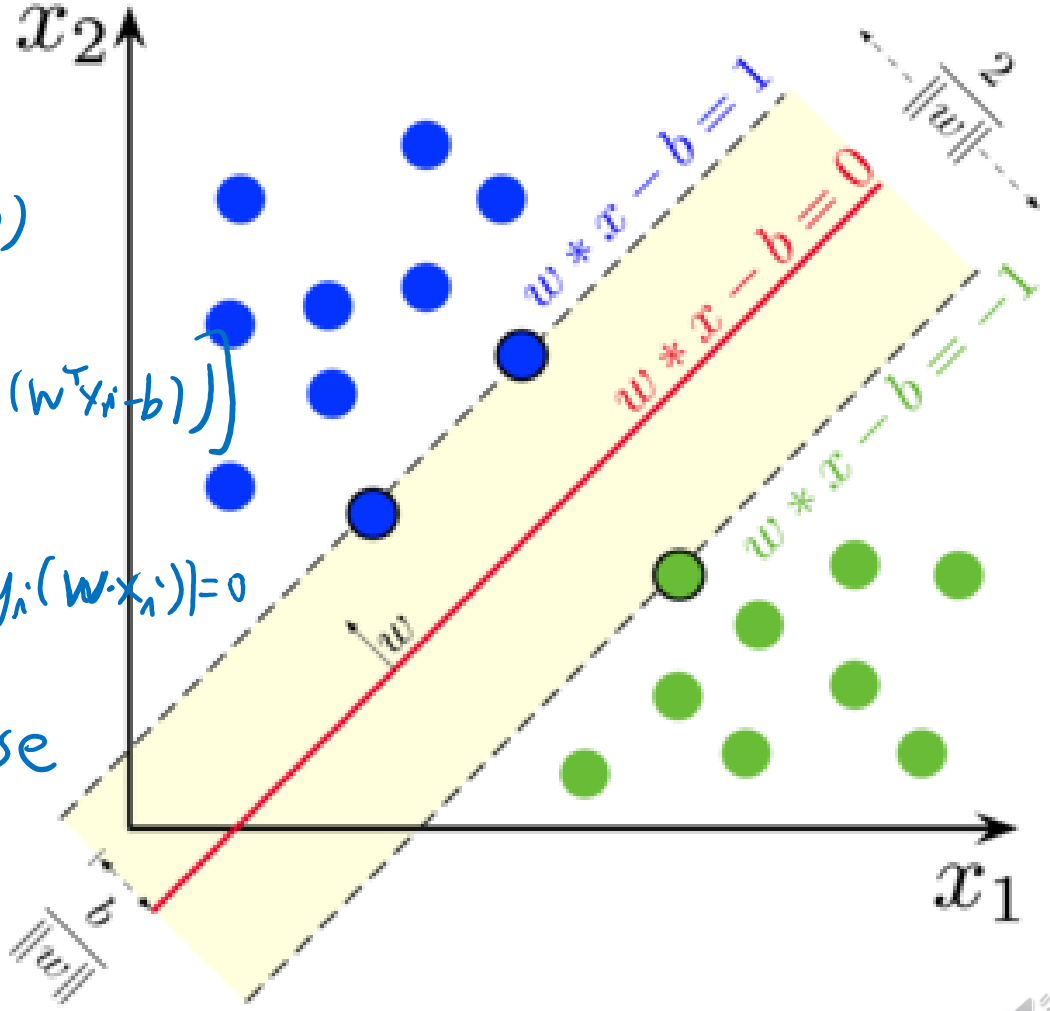
- Maximize the margin while reducing hinge losses

$$\min. \ \frac{1}{2}\|w\|^2$$

$$s.b.t. \ \frac{1}{N}\sum_{i=0}^{N} \max(0, 1 - y_i(w^T x_i - b))$$

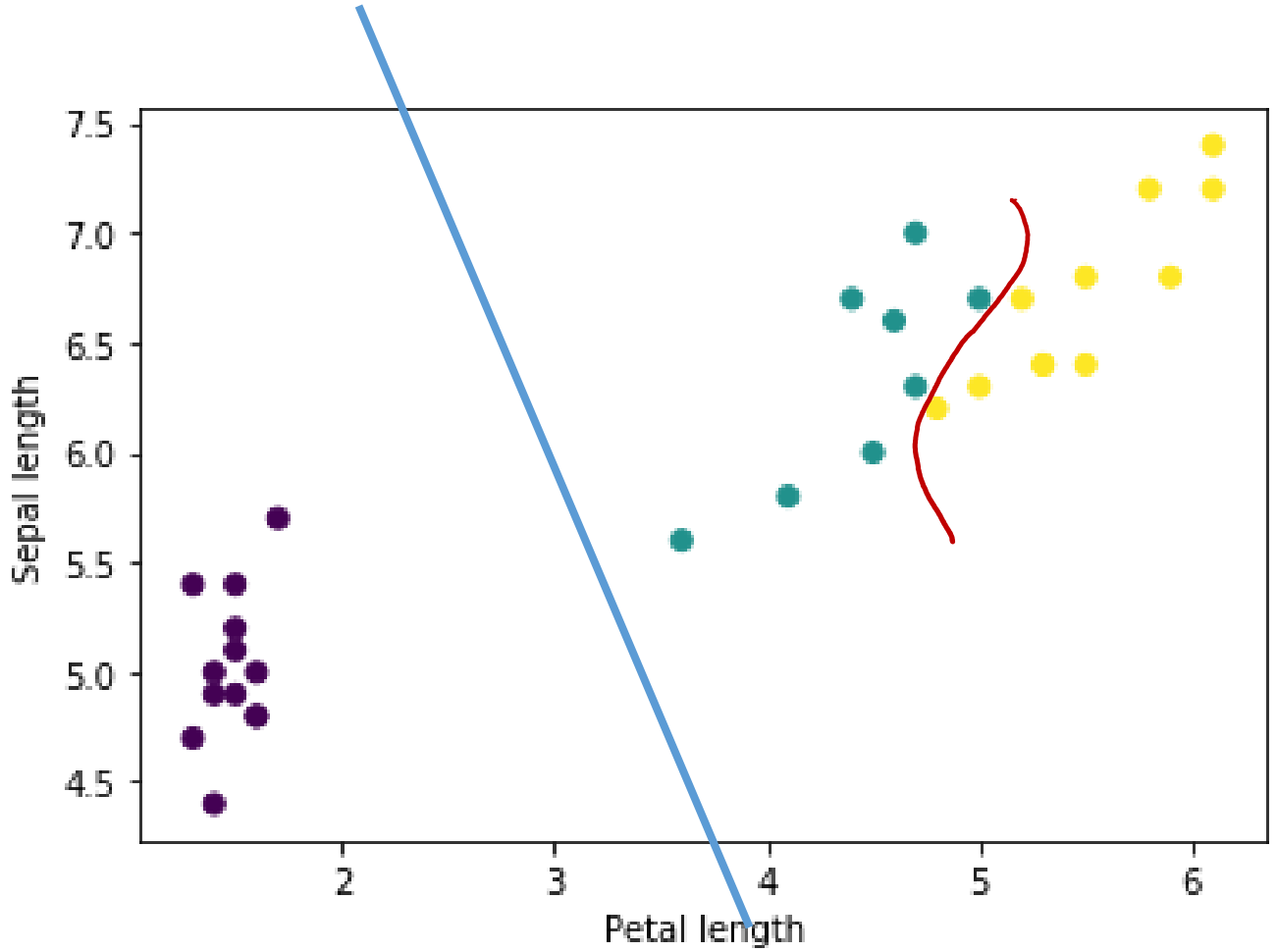
$$J(w) = \frac{1}{2}\|w\|^2 + C\left[\frac{1}{N}\sum_{i=0}^{N} \max(0, 1 - y_i(w^T x_i - b))\right]$$

$$\frac{\partial J(w)}{\partial w} = \frac{1}{N}\sum_{i} \begin{cases} w & \text{if } \max(0, 1 - y_i(w \cdot x_i)) = 0 \\ w - C y_i x_i, & \text{otherwise} \end{cases}$$



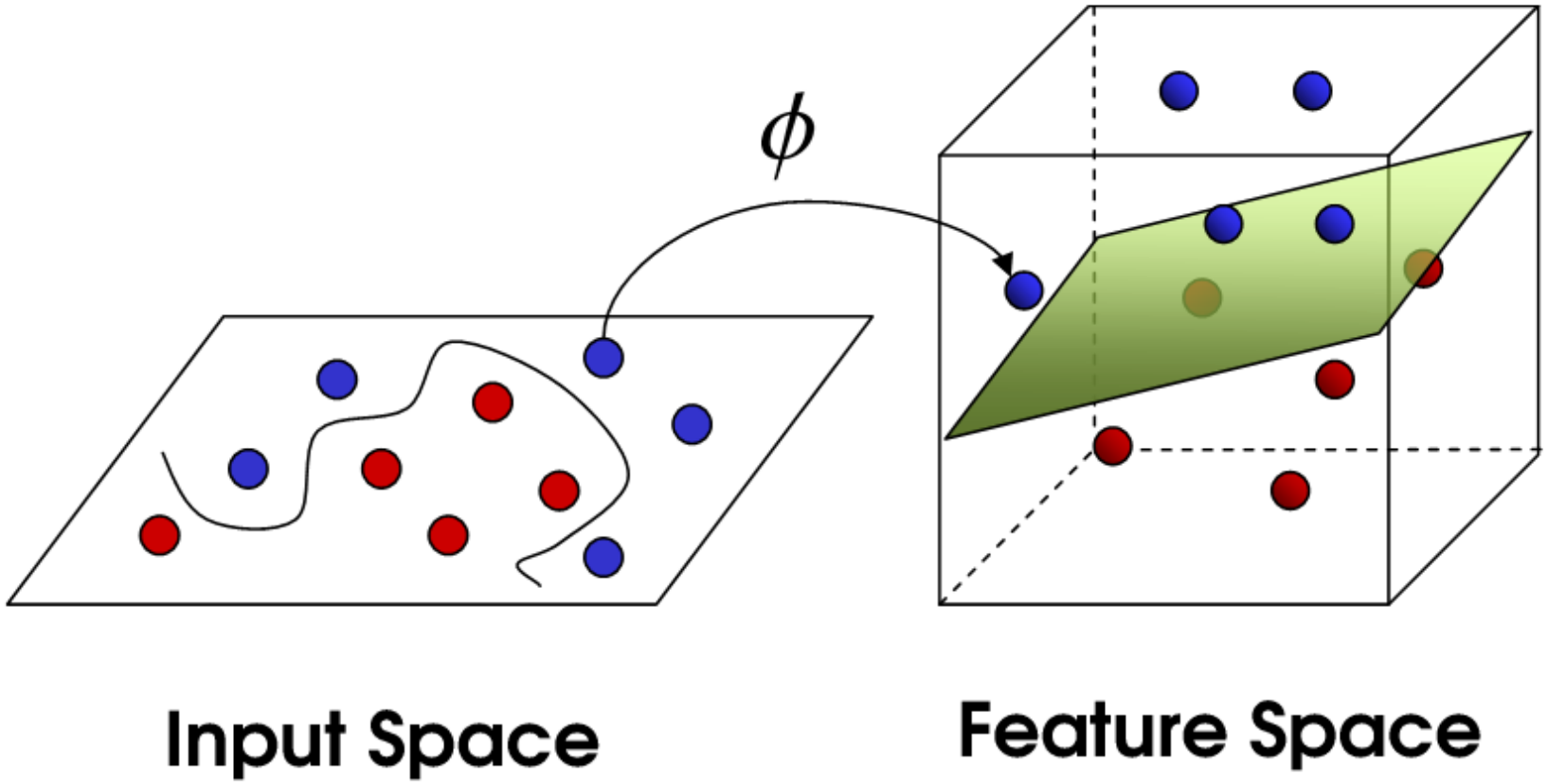
Nonlinear Problem for SVM?

- How to separate Versicolor and Virginica?



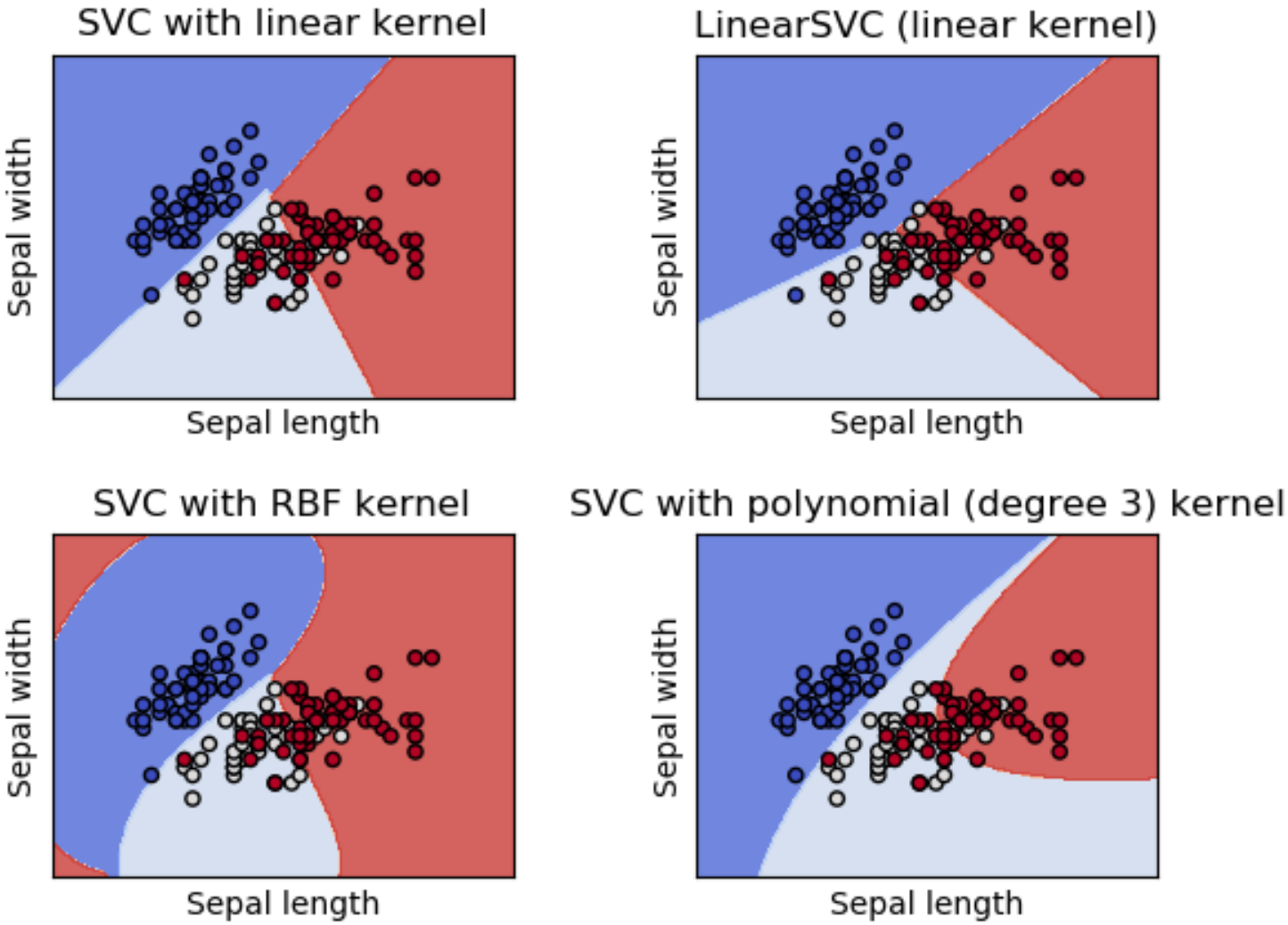
SVM Kernel Trick

- Project data into higher dimension and calculate the inner products



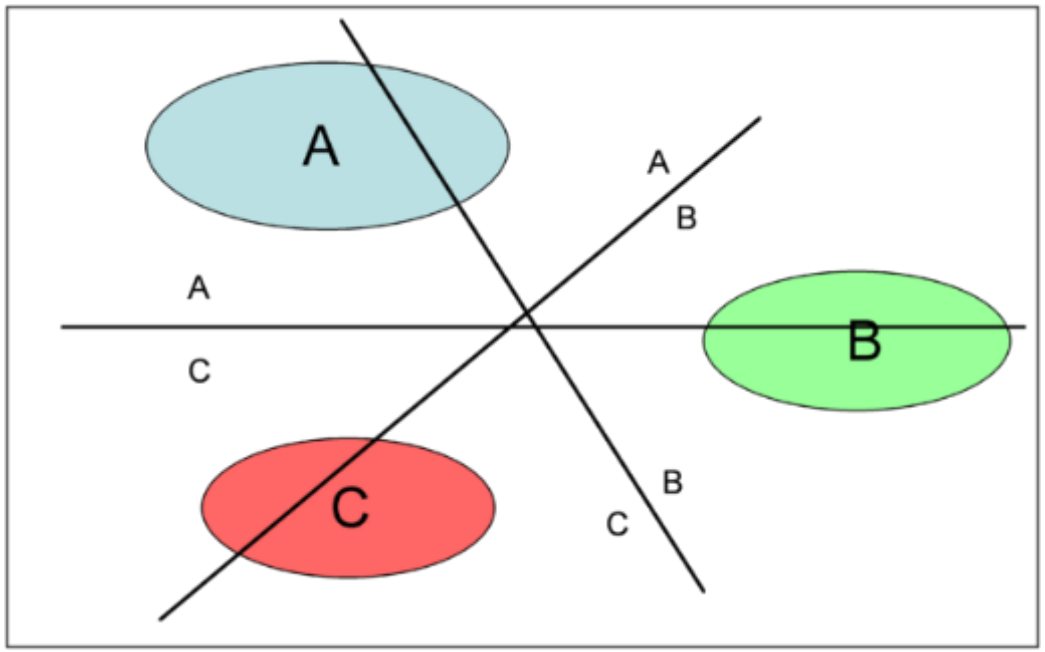
https://datascience.stackexchange.com/questions/17536/kernel-trick-explanation

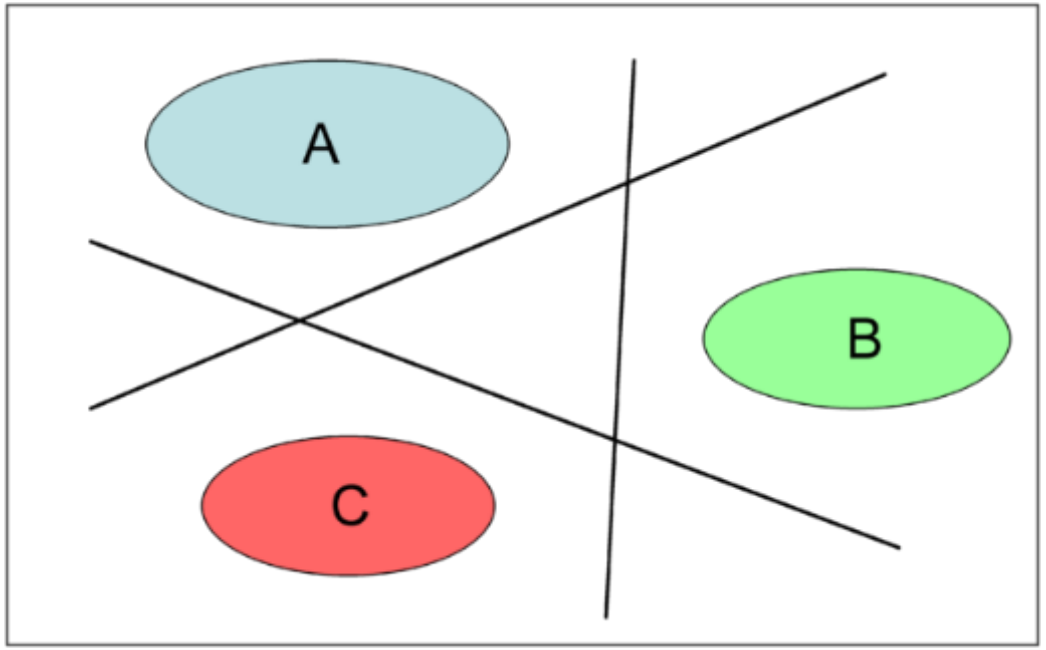
Nonlinear SVM for Iris Classification



Multi-class SVM

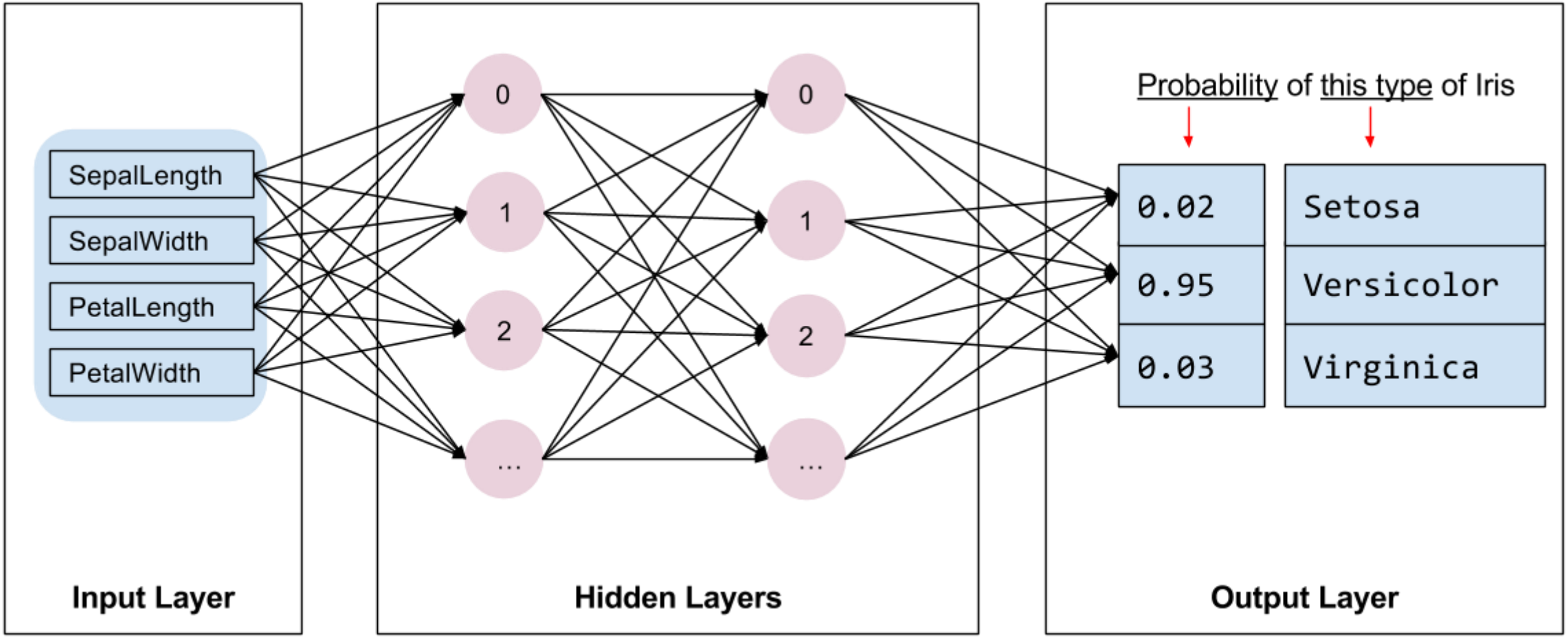
- One-against-One C_2^n
- One-against-All N





https://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/

Using Neural Network



https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough

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

Naïve Bayes

- Probabilistic classifier based on Bayes' theorem

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

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

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
```

Evaluating All Classifiers on Iris Flower Dataset

```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import *
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
```

Notebook Link

```
[2] iris = datasets.load_iris()
    X = iris.data
    y = iris.target
    print('There are {} training samples in the Iris dataset'.format(len(X)))
```

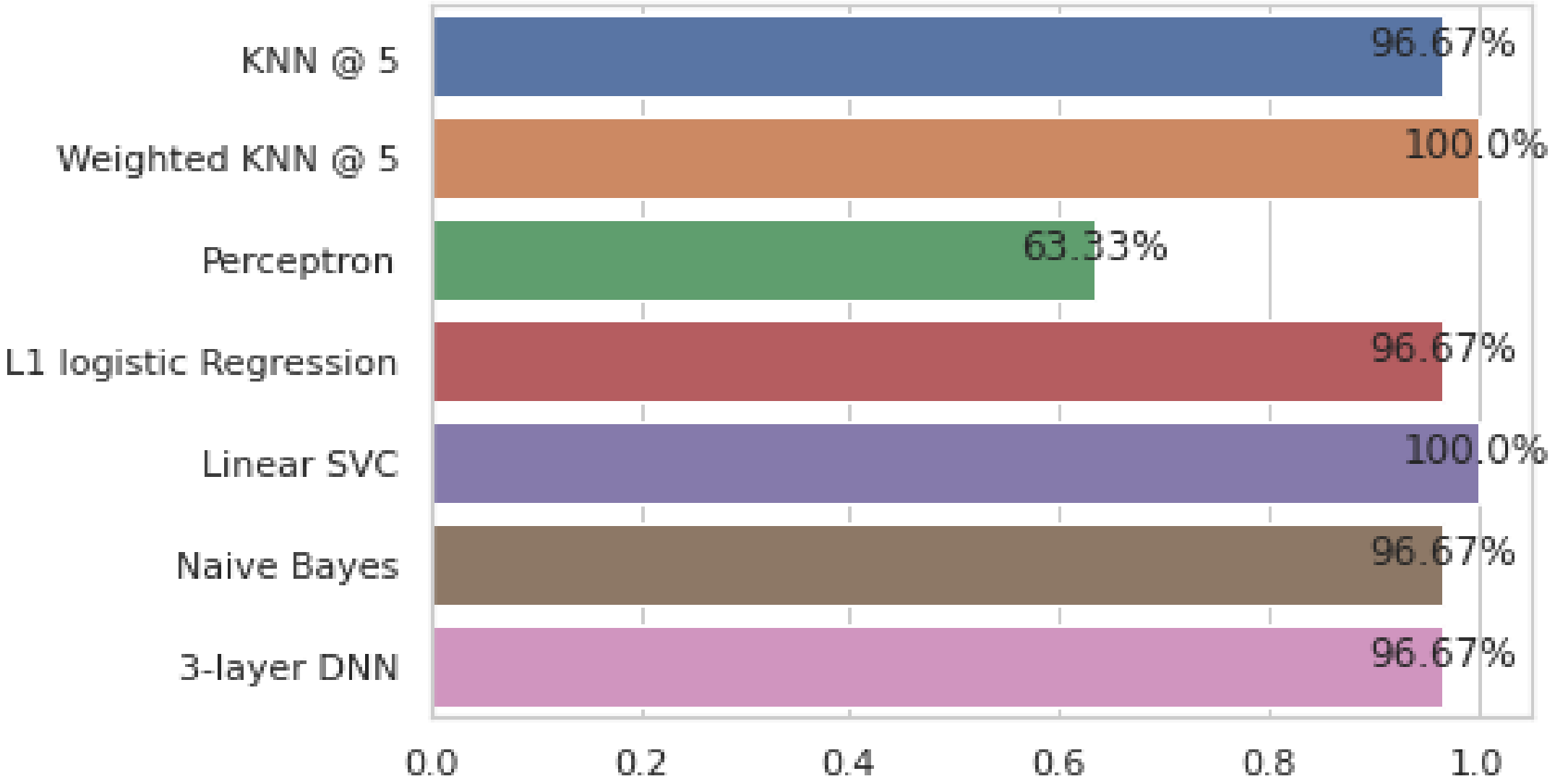
```
There are 150 training samples in the Iris dataset
```

```
[3] # Create different classifiers.
    knn = 5
    Cost = 10 # For regularization

    classifiers = {
            'KNN @ ' + str(knn): KNeighborsClassifier(knn, 'uniform'),
            'Weighted KNN @ ' + str(knn): KNeighborsClassifier(knn, 'distance'),
            'Perceptron': Perceptron(),
            'L1 logistic Regression': LogisticRegression(C=Cost, penalty='l1', solver='saga', multi_class='multinomial',max_iter=10000),
            'Linear SVC': SVC(kernel='linear', C=Cost, probability=False, random_state=0),
            'Naive Bayes': GaussianNB()
    }
```

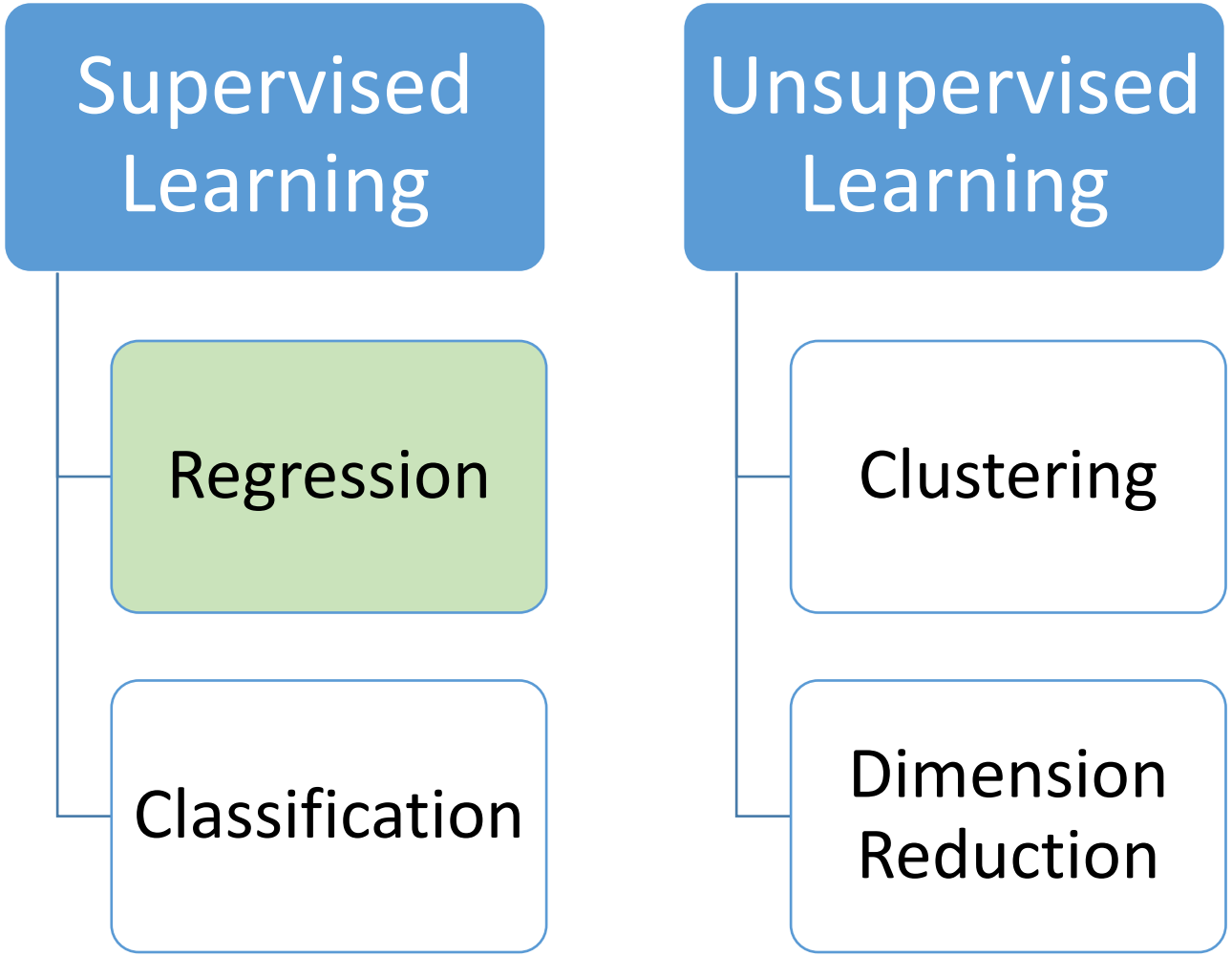
```
[4] # Training
    for index, (name, classifier) in enumerate(classifiers.items()):
            classifier.fit(X, y)

            y_pred = classifier.predict(X)
            accuracy = accuracy_score(y, y_pred)
            print("Accuracy (train) for %s: %0.1f%% " % (name, accuracy * 100))
```

Classifier Evaluation on Iris dataset

https://colab.research.google.com/drive/1CK7NFp6qX0XoGZWqryCDzdHKc3N4nD4J

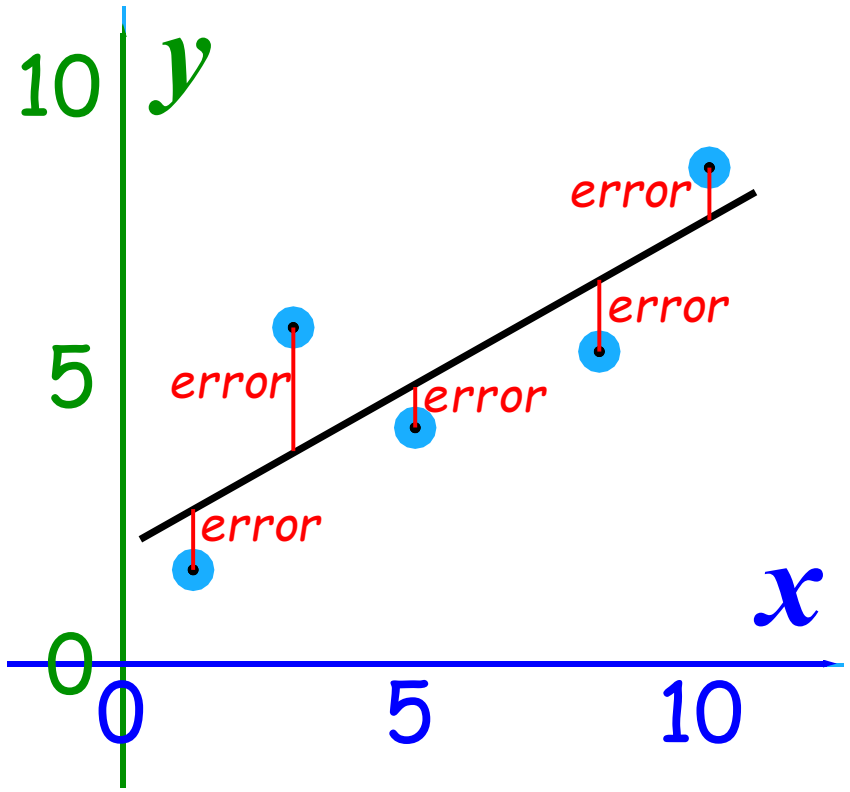


Supervised and Unsupervised Learning



Linear Regression (Least squares)

- Find a "line of best fit“ that minimizes the total of the square of the errors



$$y' = w \cdot x + b$$

$$\min. \sum_{i=0}^{n} (y_i - y'_i)^2$$

$$y = x^T w + b$$

$$= [x \;\; 1] \begin{bmatrix} w \\ b \end{bmatrix}$$

$$= \tilde{x} \cdot w$$

$$w = (\tilde{x}^T \tilde{x})^{-1} \tilde{x}^T y$$

Least Squares Solution

- Pseudo Inverse: $w = (\boldsymbol{x}^T \boldsymbol{x})^{-1} \boldsymbol{x}^T y$

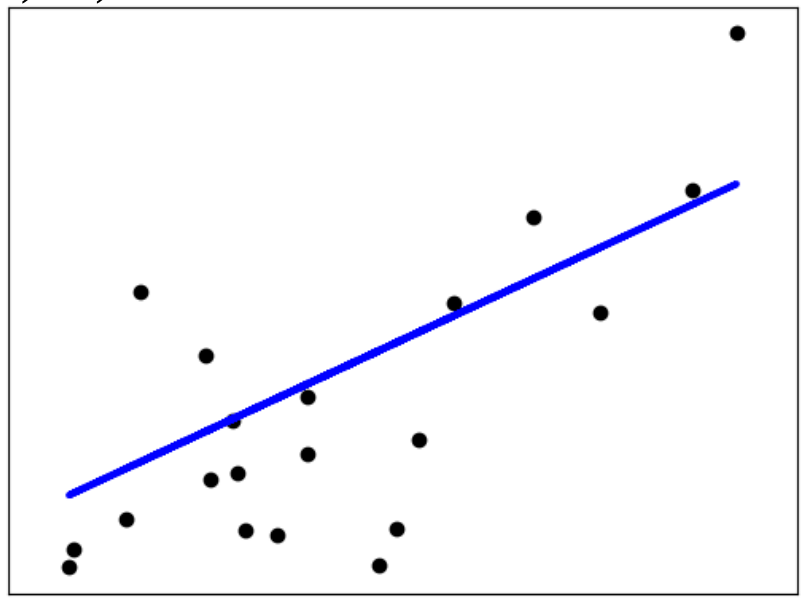
- $\boldsymbol{xw} = y$ *has a unique solution if and only if*
  - *The columns of* $\boldsymbol{x}$ *are linearly independent.*
  - $\boldsymbol{x}^T \boldsymbol{x}$ *is invertible.*

http://textbooks.math.gatech.edu/ila/least-squares.html

Scikit-Learn Diabetes Dataset

- 10 features: age, sex, body mass index, average blood pressure, and six blood serum measurements
- Obtained from n = 442 diabetes patients
- Normalized to have mean 0 and squared length = 1 (sum(x^2)=1)

Total sample	442
Dimensionality	10
Features	real, -.2 < x < .2
Targets	integer 25 - 346



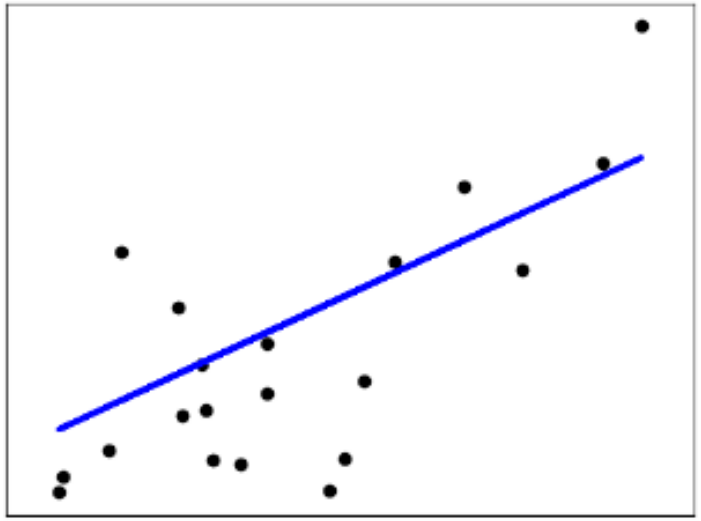
BMI

https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

Linear Regression

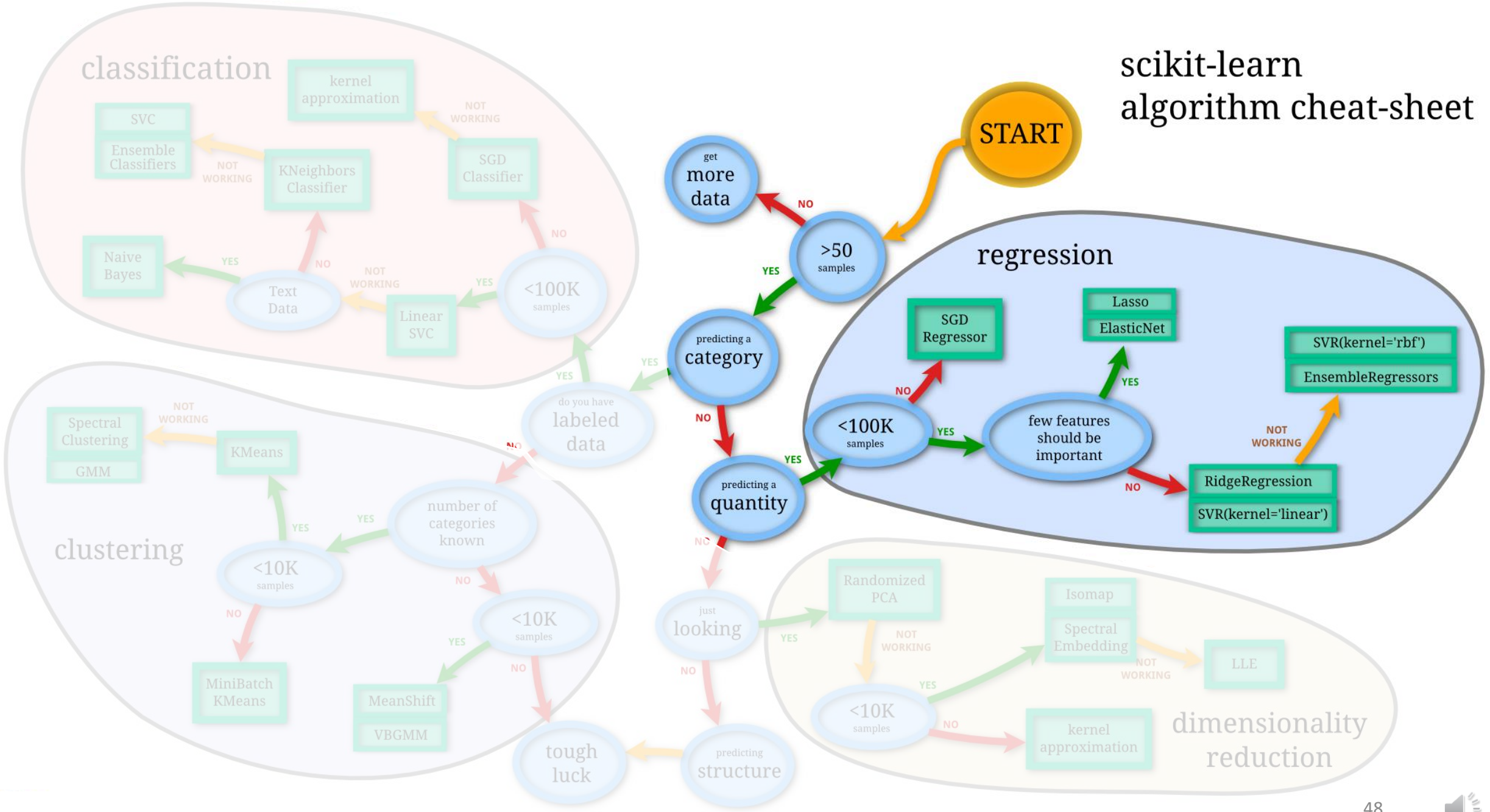
- Least Squares

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



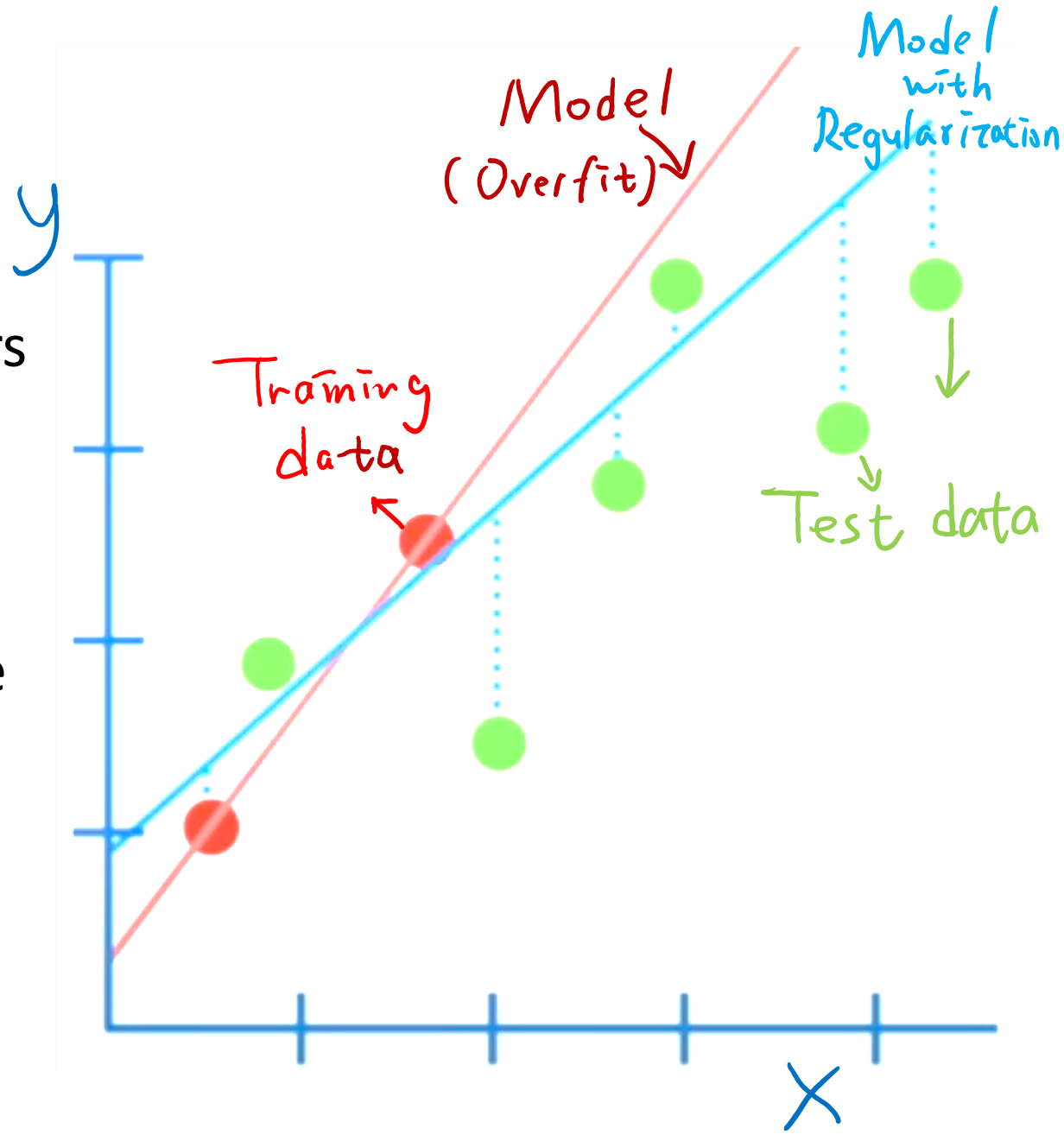
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
diabetes_X_train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

scikit-learn algorithm cheat-sheet



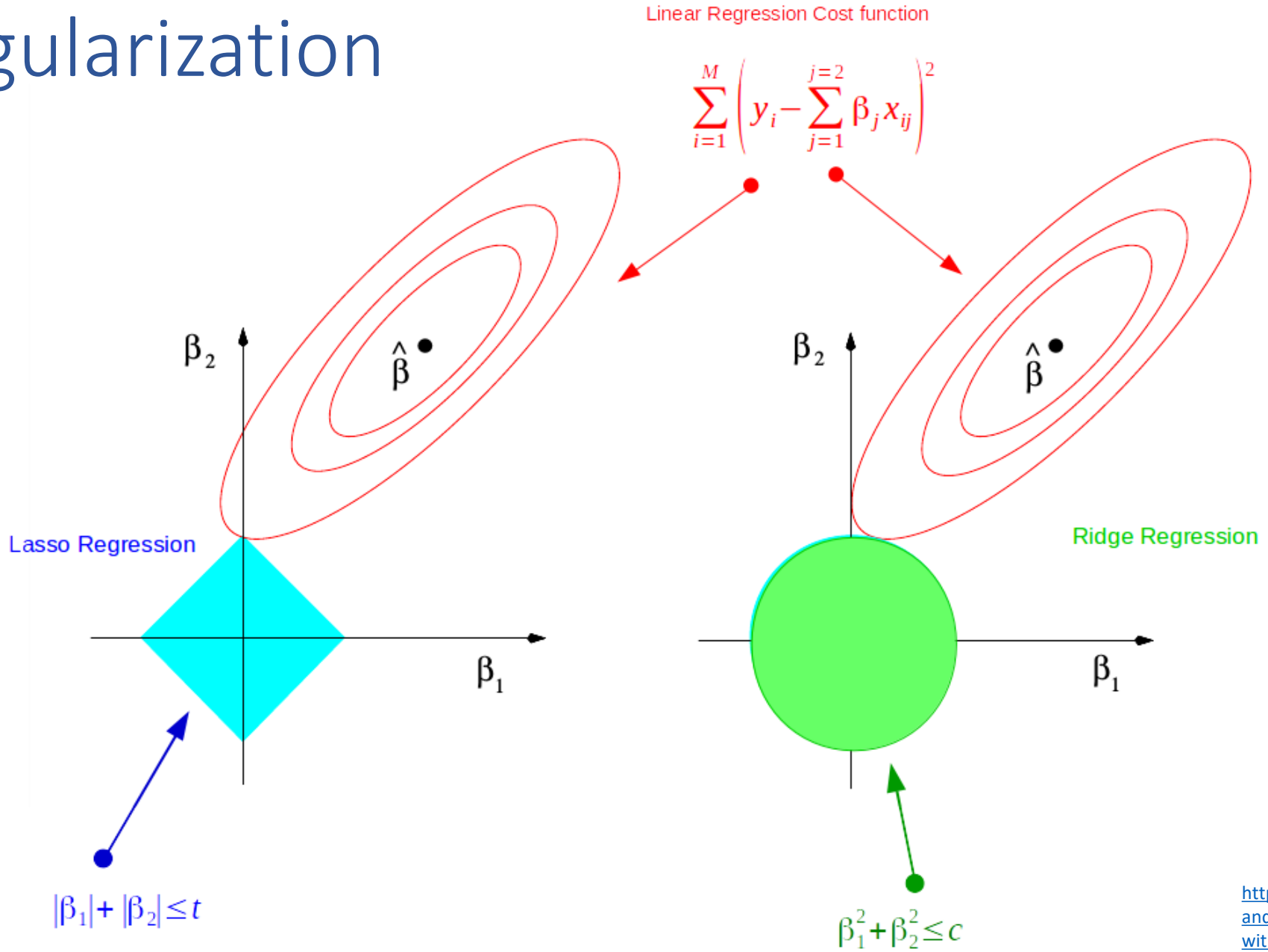
Regularization

- Linear regression is sensitive to outliers and easy to overfit training data
- Regularization add small amount of bias during training to reduce variance on test data



StatQuest: https://www.youtube.com/watch?v=Q81RR3yKn30

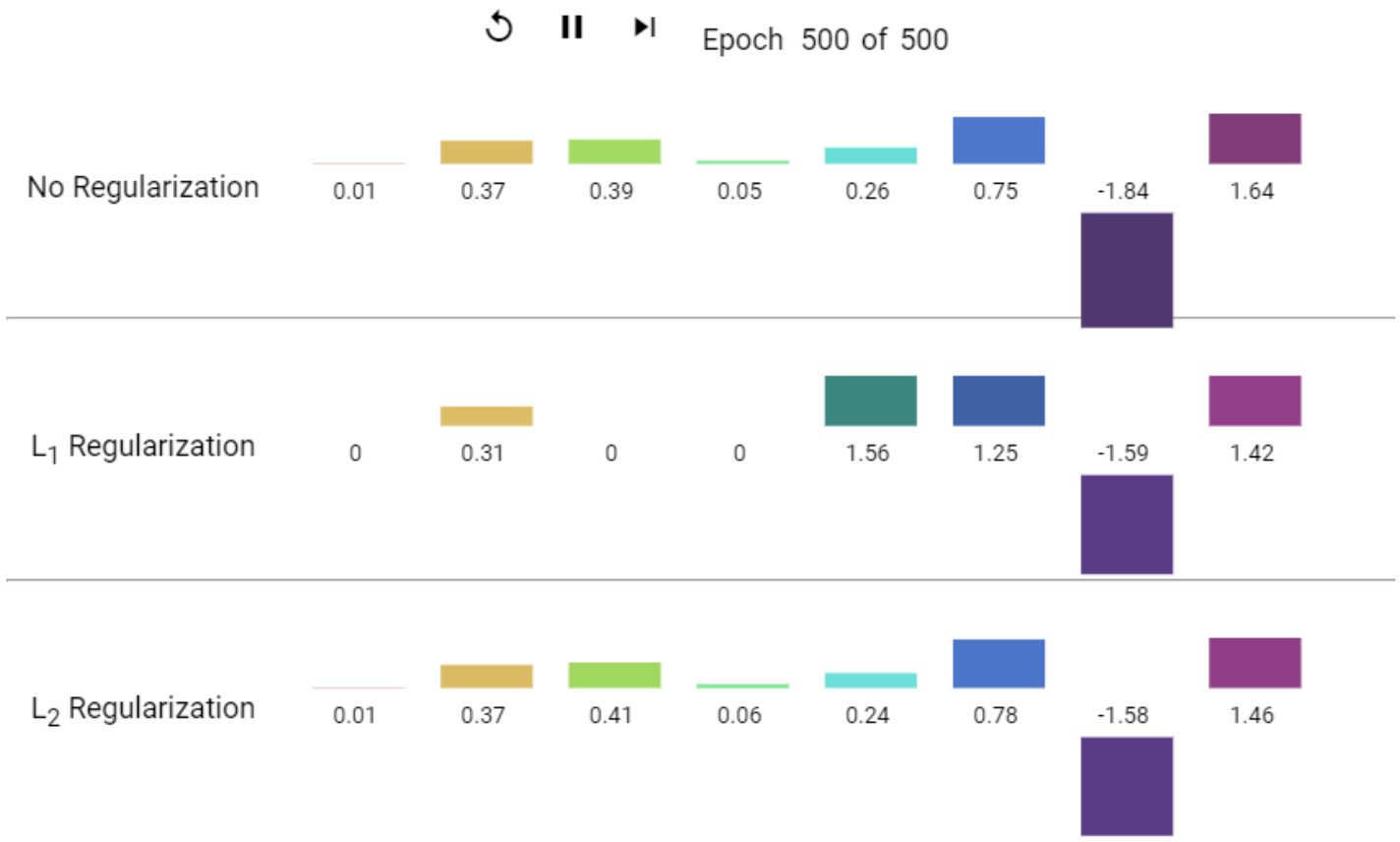
Regularization



https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

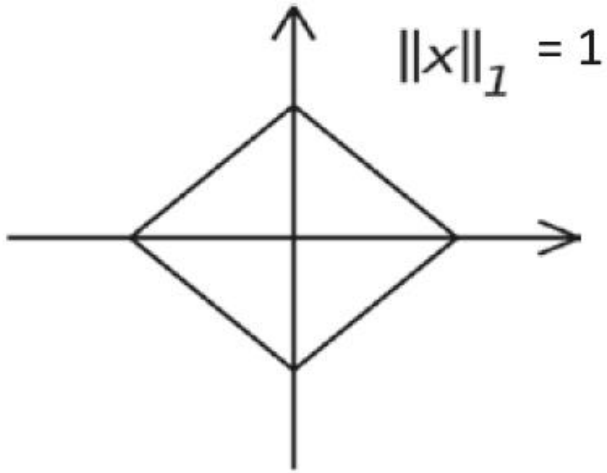
Regularization

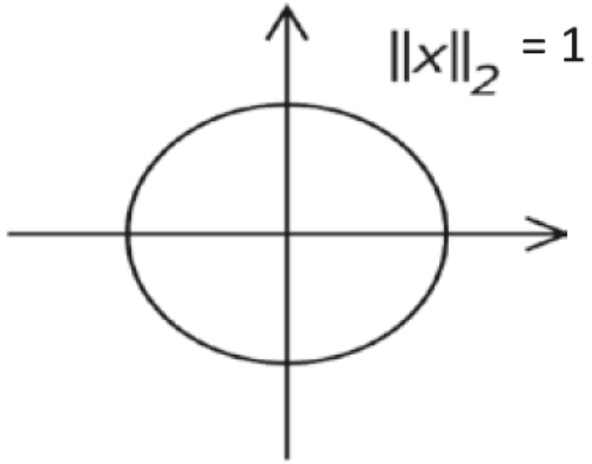
- https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization

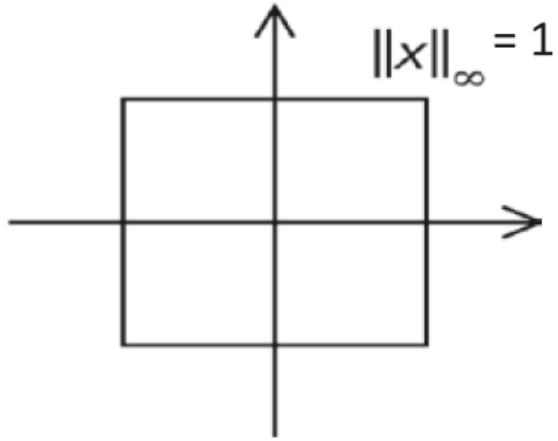


Unit norms in 2D Vectors

- The set of all vectors of norm 1 in different 2D norms







Ridge, Lasso and ElasticNet

- Ridge regression: $\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$
- Lasso regression: $\min_w \frac{1}{2n_{\text{samples}}}||Xw - y||_2^2 + \alpha||w||_1$
- Elastic Net: $\min_w \frac{1}{2n_{\text{samples}}}||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$

Predicting Boston House Prices

Boston House Price Dataset

- Objective: predict the median price of homes
- Small dataset with 506 samples and 13 features
  - https://www.kaggle.com/c/boston-housing

1	crime	per capita crime rate by town.	8	dis	weighted mean of distances to five Boston employment centres.
2	zn	proportion of residential land zoned for lots over 25,000 sq.ft.	9	rad	index of accessibility to radial highways.
3	indus	proportion of non-retail business acres per town.	10	tax	full-value property-tax rate per $10,000.
4	chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).	11	ptratio	pupil-teacher ratio by town.
5	nox	nitrogen oxides concentration	12	black	1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
6	rm	average number of rooms per dwelling.	13	lstat	lower status of the population (percent).
7	age	proportion of owner-occupied units built prior to 1940.			

Normalize the Data

- Make features center around 0 and have unit standard deviation
- Note that the quantities (mean, std) used for normalizing the test data are computed using the training data!

```
# Nomalize the data
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```


Regressor Comparison

+ 程式碼 + 文字 Notebook link...

```
[1] from sklearn.metrics import mean_squared_error, mean_absolute_error
    from sklearn.linear_model import *
    from sklearn import datasets
```

```
data = datasets.load_boston()
X = data.data
y = data.target
y_mean = y.mean()
print('There are {} training samples in the Boston House dataset. The average price is {}'.format(len(X), y_mean))
```

```
There are 506 training samples in the Boston House dataset. The average price is 22.532806324110677
```

```
[3] mean = X.mean(axis=0)
    X -= mean
    std = X.std(axis=0)
    X /= std
```

```
[4] regressors = {
        'Linear Regression': LinearRegression(),
        'Ridge Regression': Ridge(alpha=.5),
        'LASSO': Lasso(alpha=0.1),
        'ElasticNet': ElasticNet(alpha=0.1)
    }
```

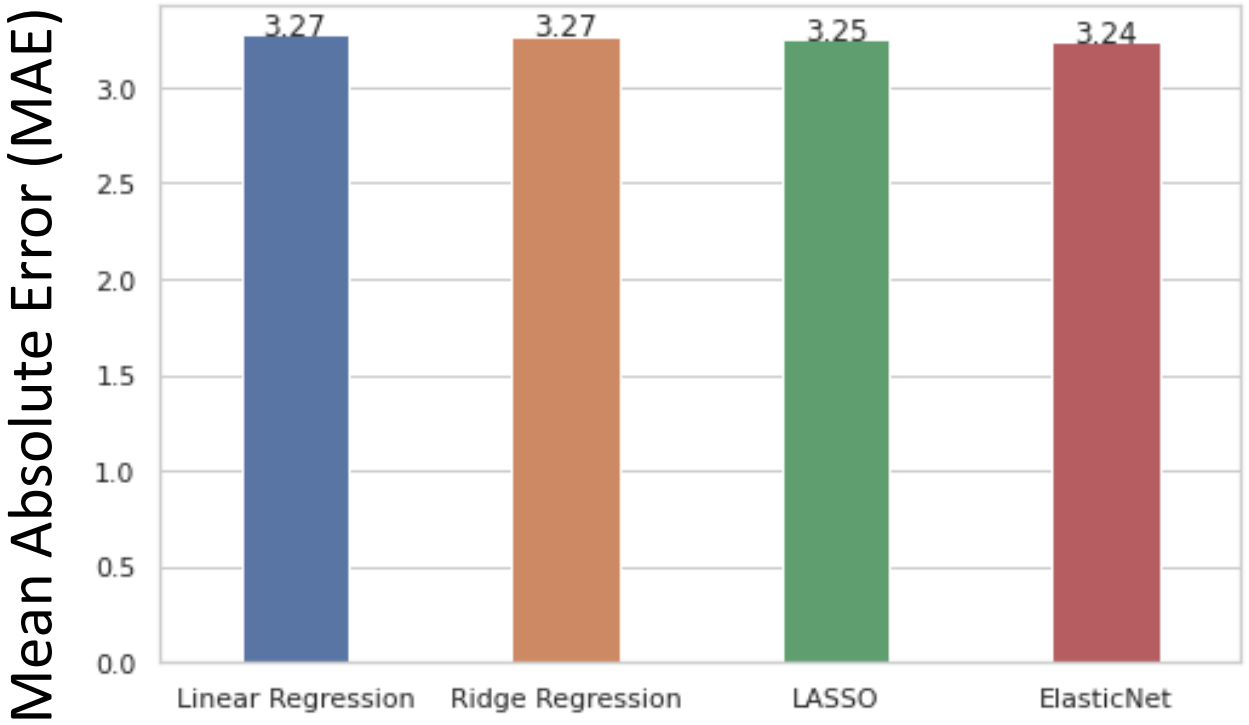
```
[5] # Training
    mae_list = []
    for index, (name, regressor) in enumerate(regressors.items()):
        regressor.fit(X, y)

        y_pred = regressor.predict(X)
        mse = mean_absolute_error(y, y_pred)
        print("The Mean Absolute Error for %s: %0.4f " % (name, mse))
        mae_list.append(mse)
```

```
The Mean Absolute Error for Linear Regression: 3.2709
The Mean Absolute Error for Ridge Regression: 3.2691
The Mean Absolute Error for LASSO: 3.2464
The Mean Absolute Error for ElasticNet: 3.2387
```

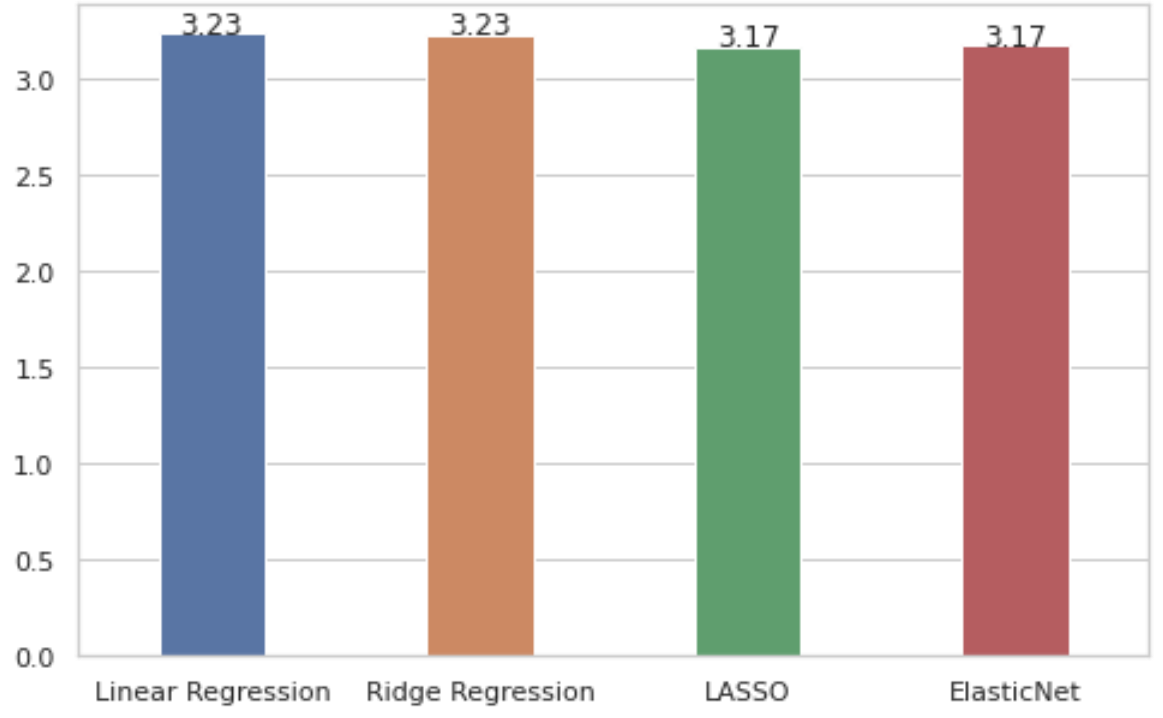
0 秒 完成時間：上午10:41

Comparison of Regularization Methods

Training Data (506 samples)



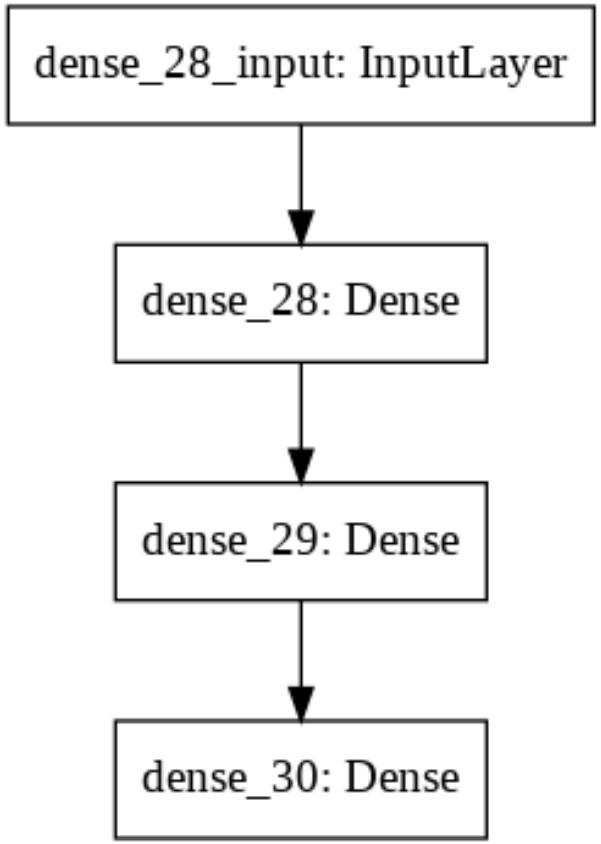
Test Data (102 samples)



https://colab.research.google.com/drive/1lgITg2vEmKfgqp7yDtrOCbWmtYuzRwIm

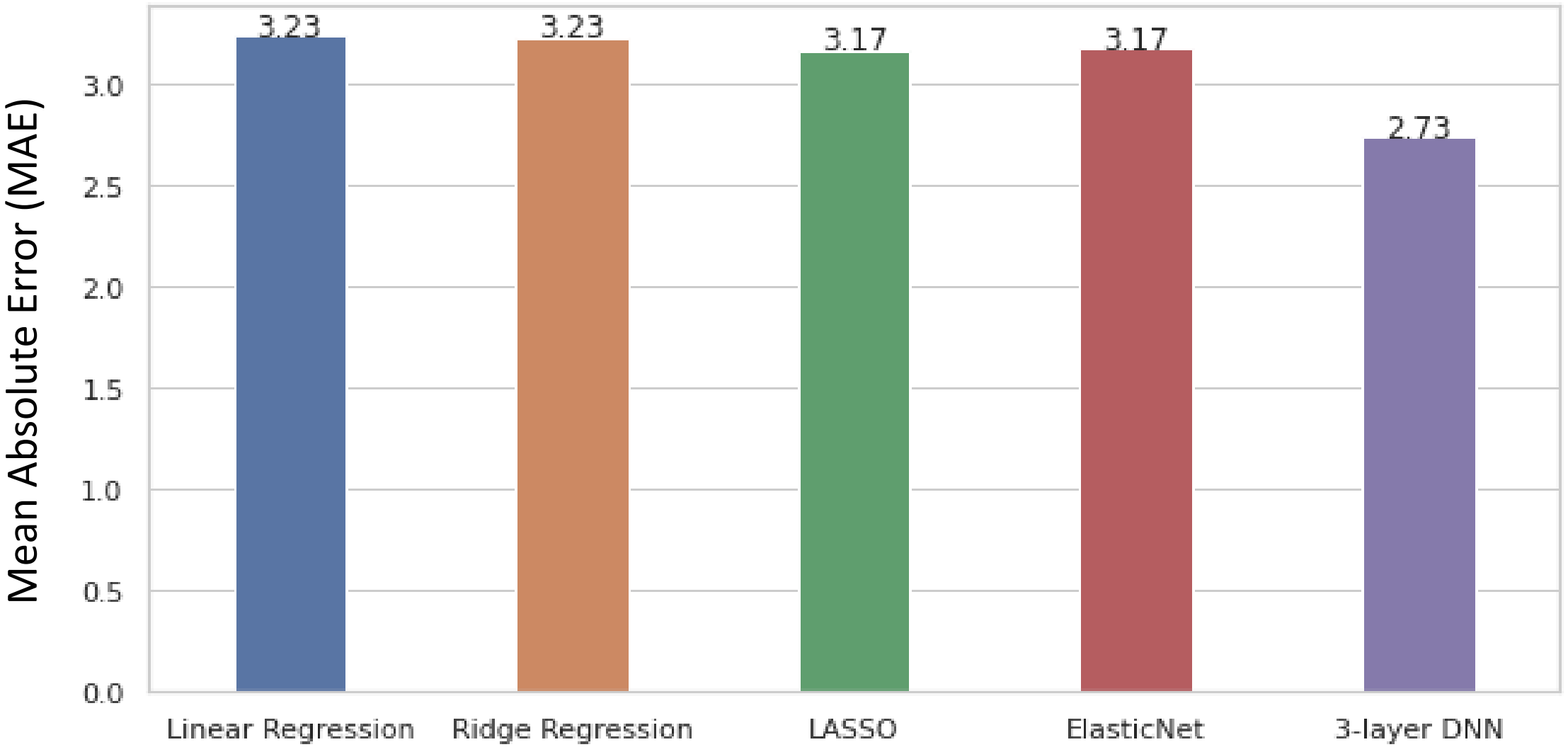
Predicting House Price using DNN

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 64)                896
_________________________________________________________________
dense_29 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 65
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```



https://colab.research.google.com/drive/1tJztaaOIxbk_VuPKm8NpN7Cp_XABqyPQ

Final Results



Logistic Regression

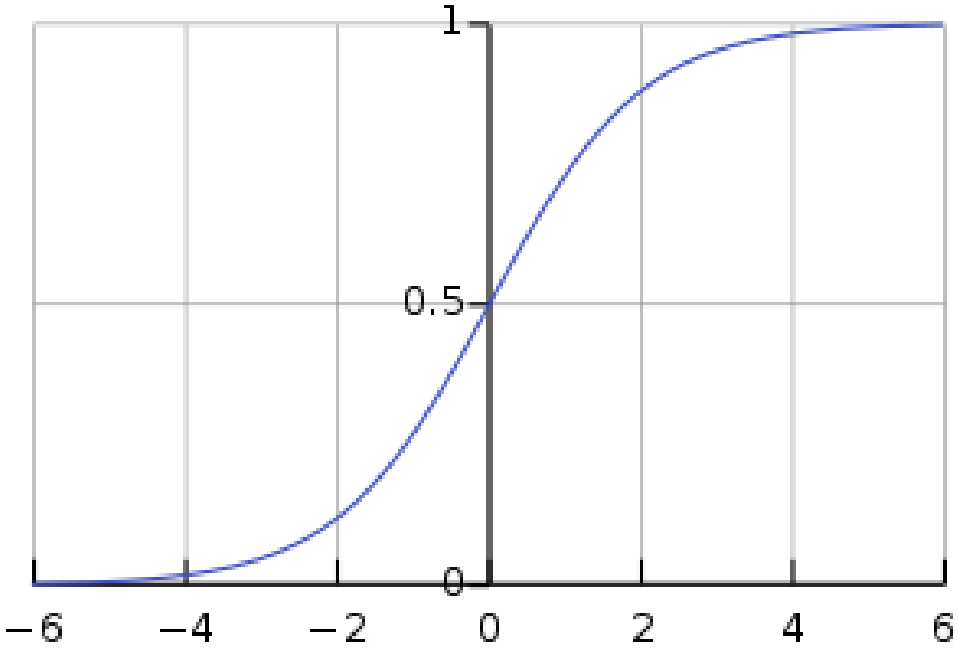
- Sigmoid function

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

- Derivative of Sigmoid

$$S(x) = S(x)(1 - S(x))$$

S-shaped curve



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

Decision Boundary

- Binary classification with decision boundary t

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

$$y' = \begin{cases} 0, & P_\theta(x) < t \\ 1, & P_\theta(x) \geq t \end{cases}$$

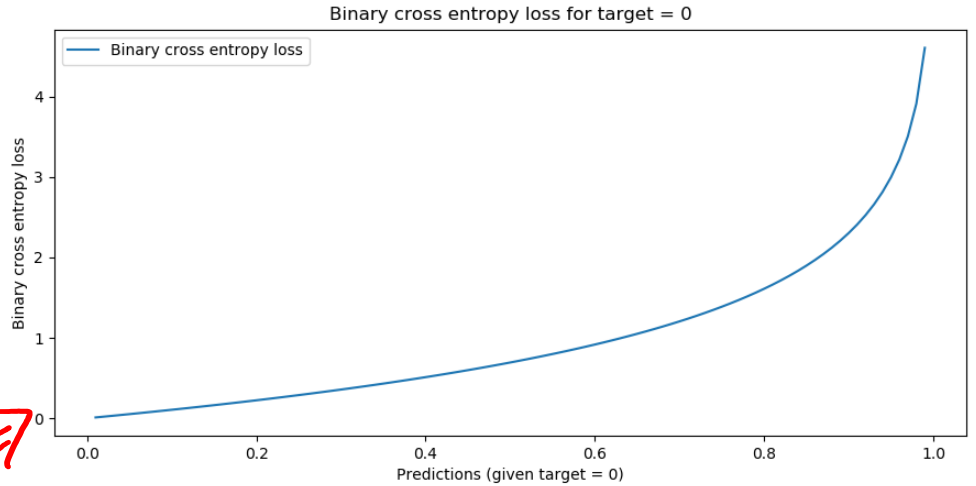
Cross Entropy Loss

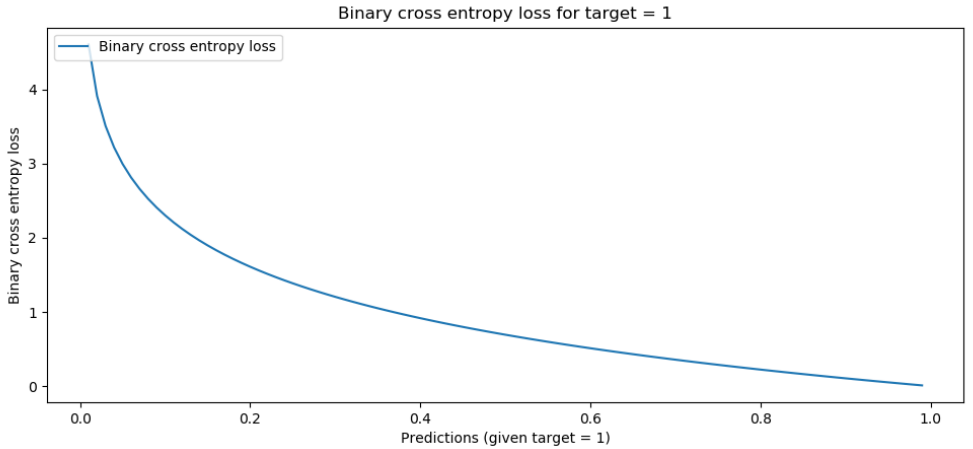
- Loss function: cross entropy

$\log \frac{1}{P(x)} = -\log P(x)$ information

$-P(x) \log P(x)$ entropy

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





https://towardsdatascience.com/a-guide-to-neural-network-loss-functions-with-applications-in-keras-3a3baa9f71c5

Cross Entropy Loss

- Loss function: cross entropy

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

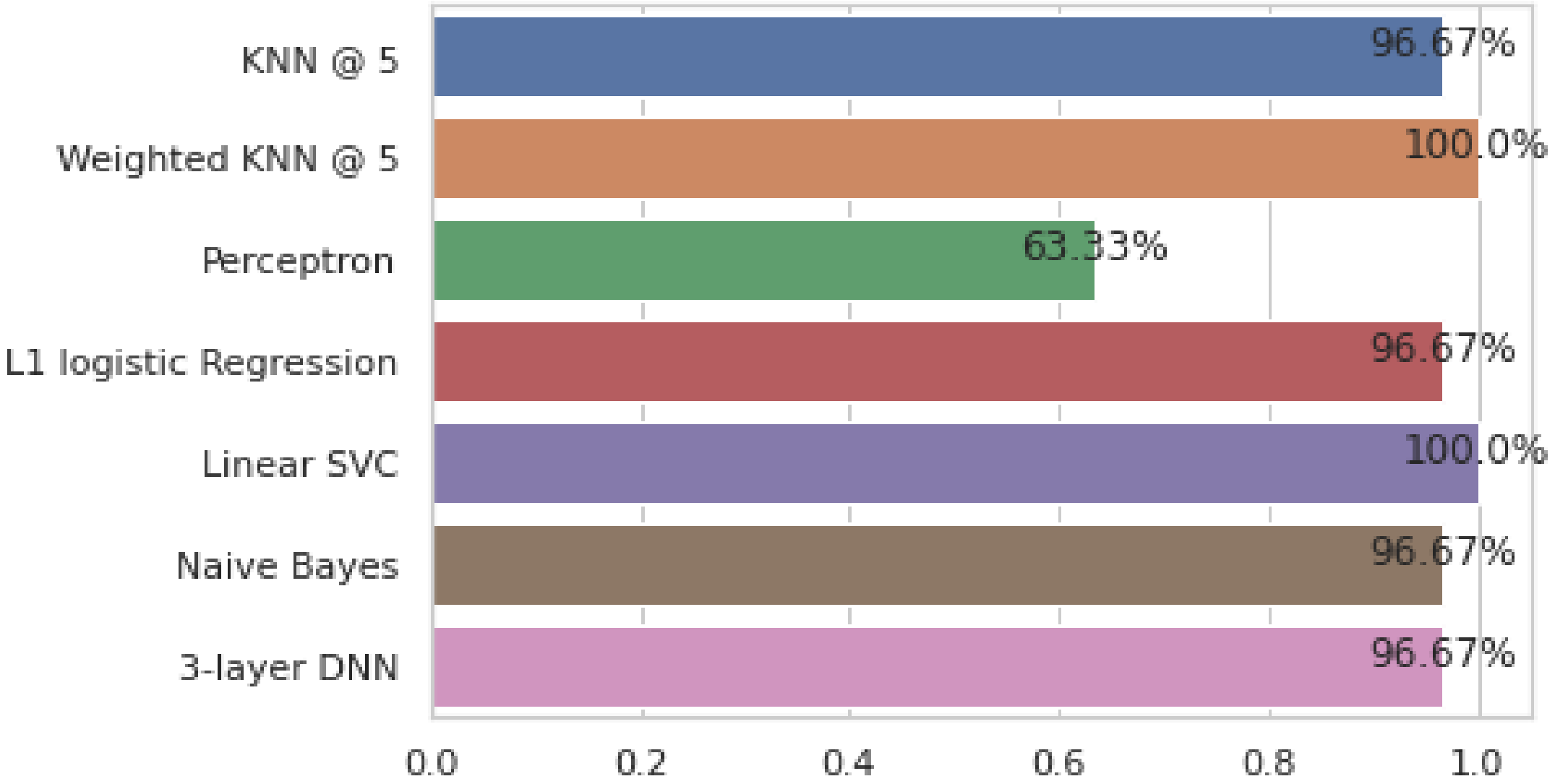
$$\Rightarrow L_\theta(\mathrm{x}) = -y\log\big(P_\theta(x)\big) + -(1-\mathrm{y})\log\big(1 - P_\theta(x)\big)$$

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

https://towardsdatascience.com/a-guide-to-neural-network-loss-functions-with-applications-in-keras-3a3baa9f71c5

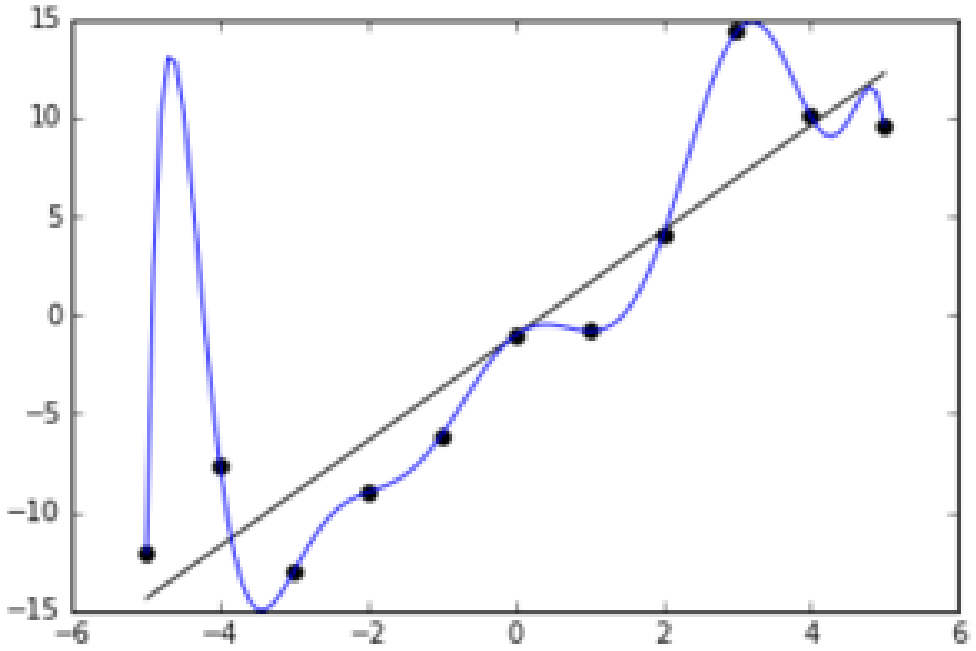
Classifier Evaluation on Iris dataset

https://colab.research.google.com/drive/1CK7NFp6qX0XoGZWqryCDzdHKc3N4nD4J

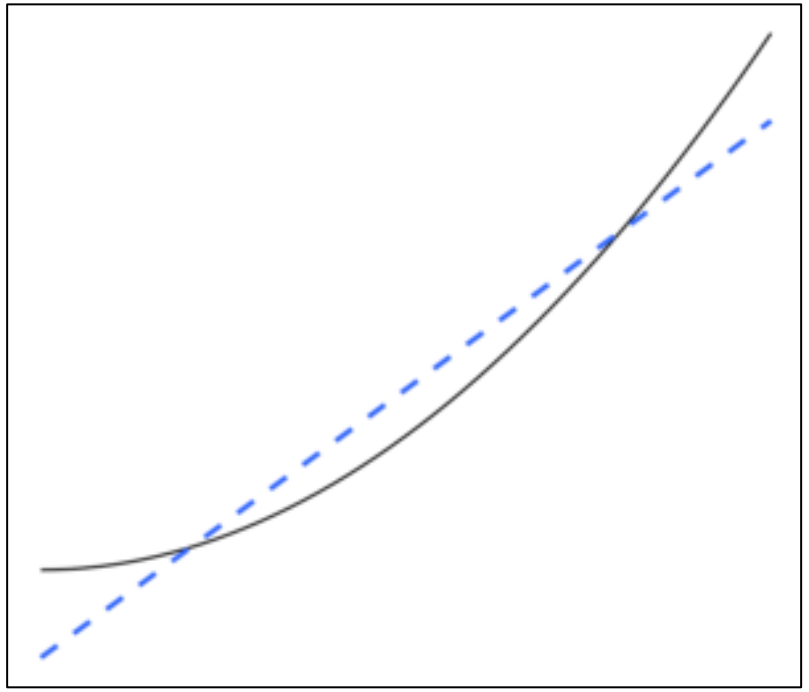


Overfitting & Underfitting

Overfitting



Underfitting



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

Neural Network Urban Legend: Detecting Tanks

- Detector learned the illumination of photos

Prediction Error: Bias and Variance Trade-off

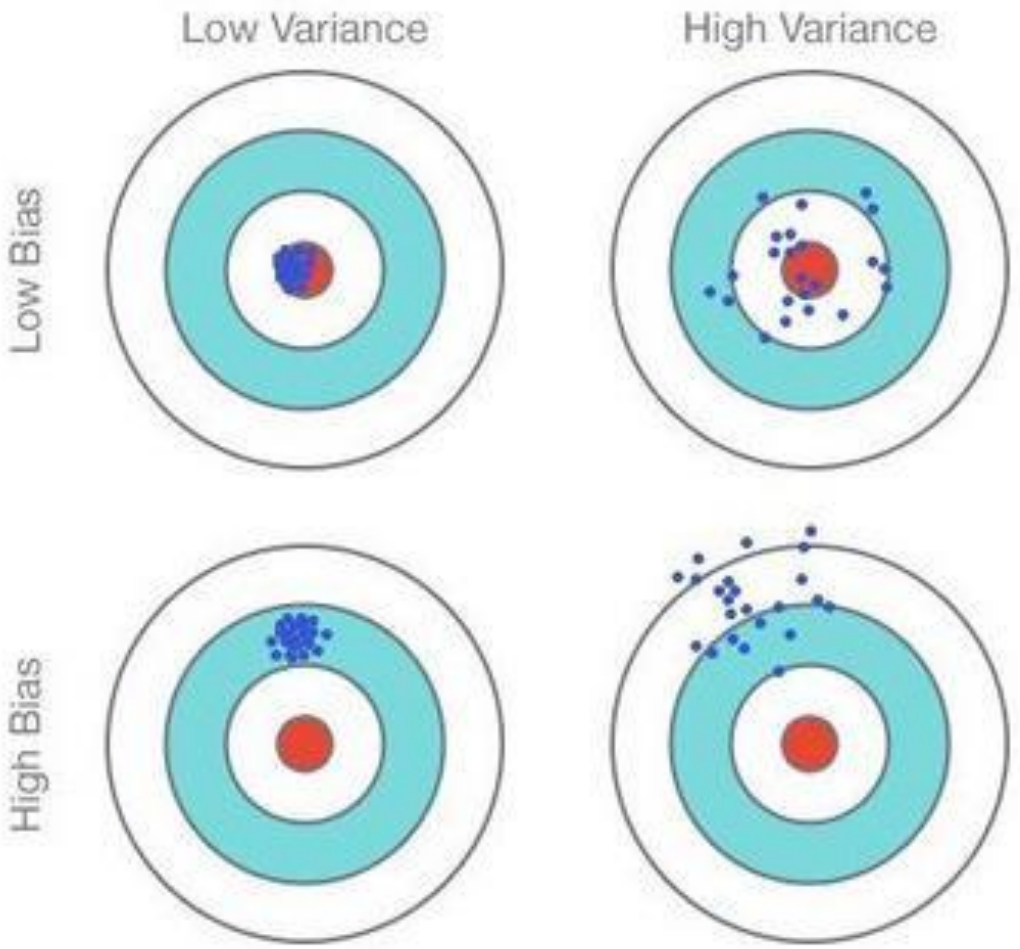
- Model with high variance overfits to training data and does not generalize on unseen test data

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

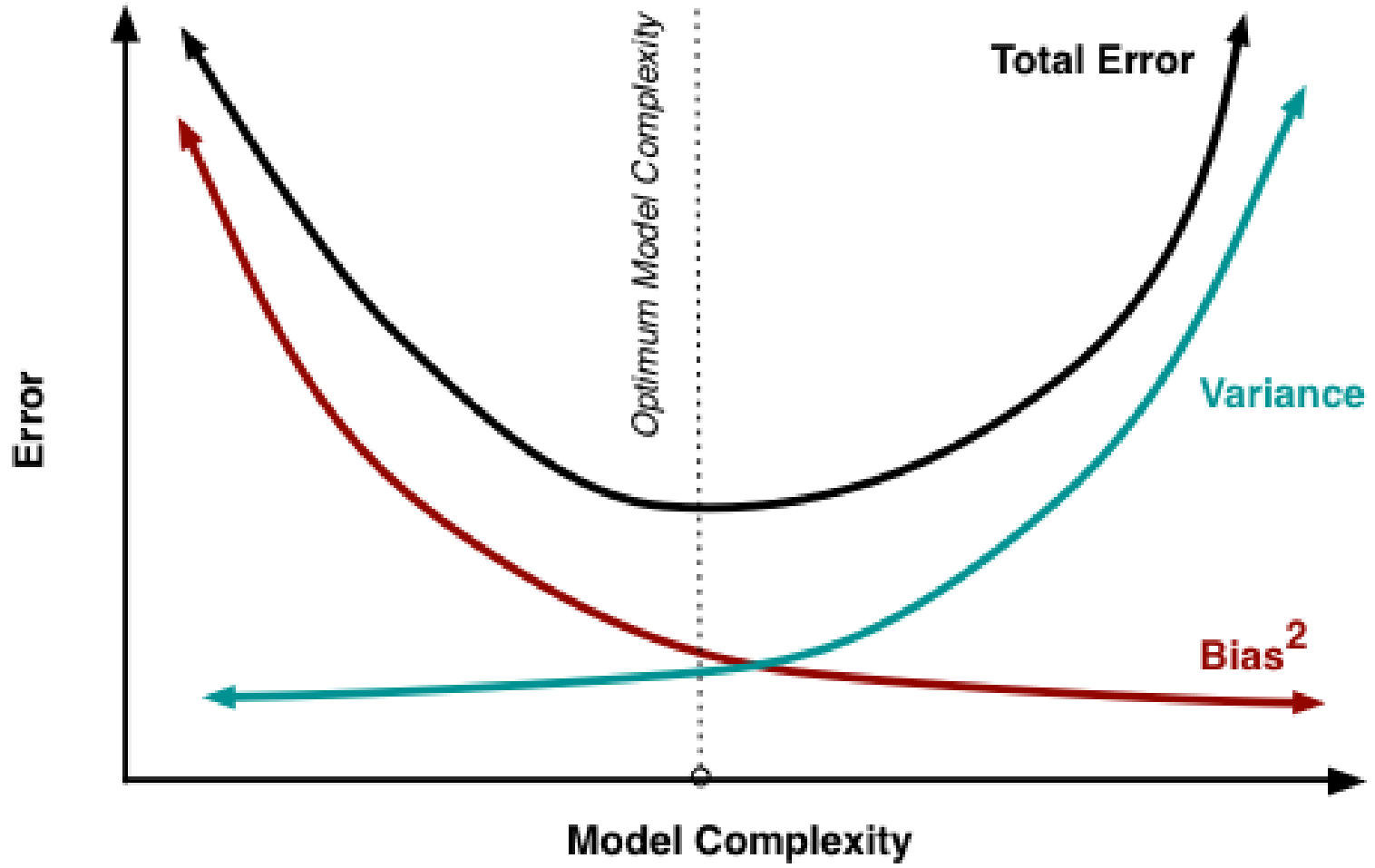
$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229



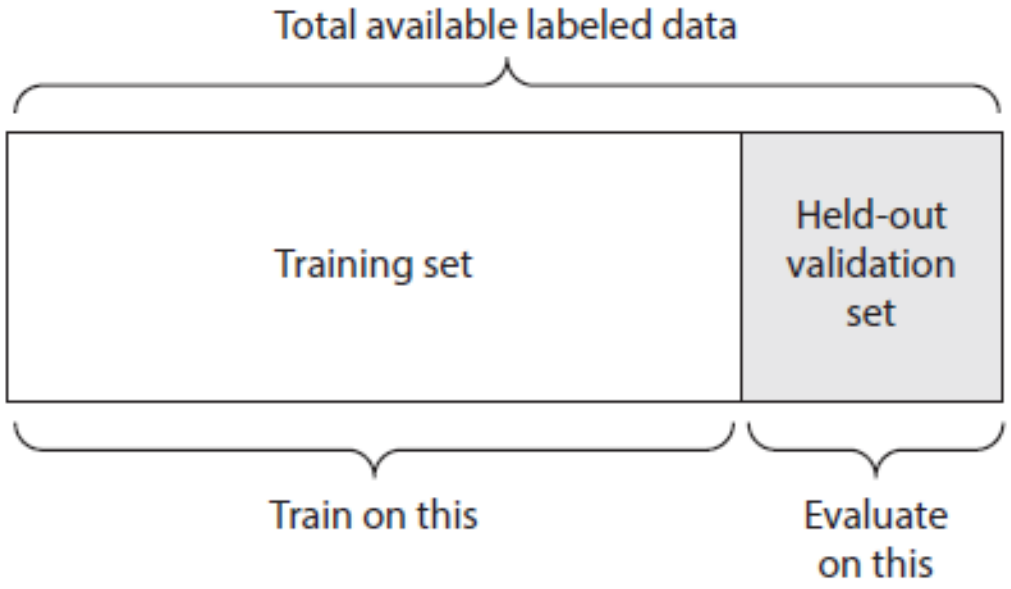
http://scott.fortmann-roe.com/docs/BiasVariance.html

Model Selection



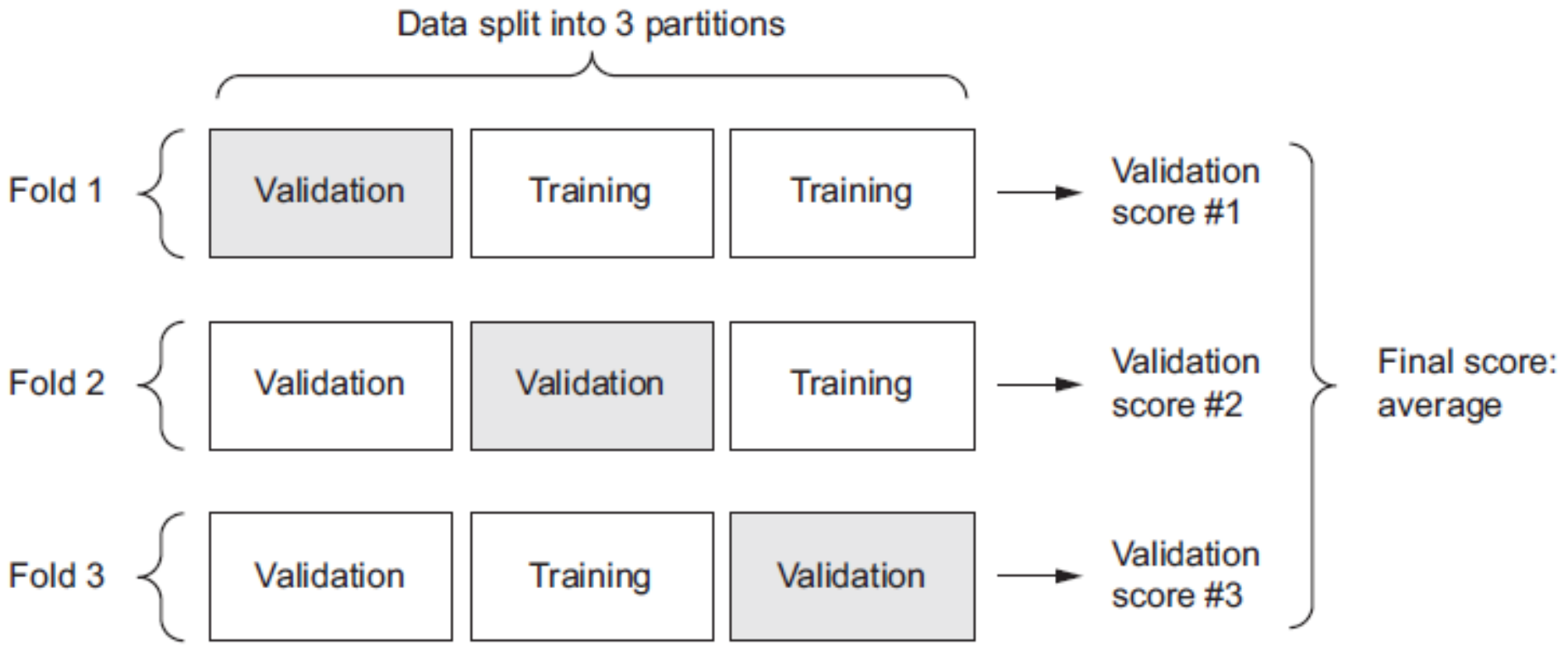
Training, Validation, Testing

- Never leak test data information into our model
- Tuning the *hyperparameters* of our model on validation dataset
- Better generalize the model to future unseen data



K-Fold Cross Validation

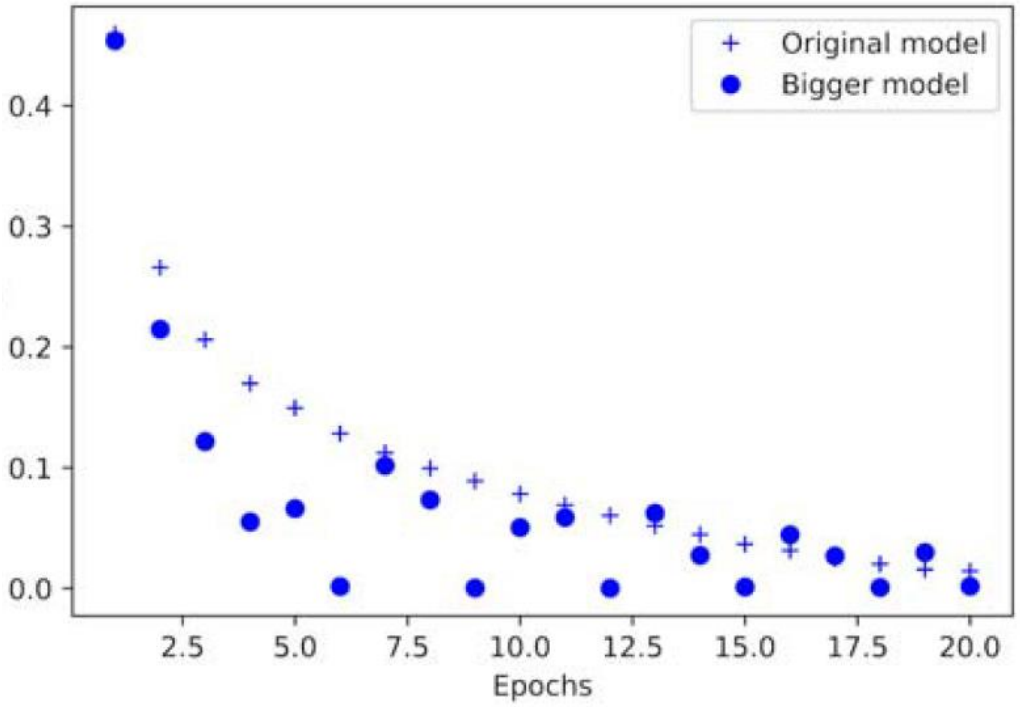
- Lower the variance of validation set



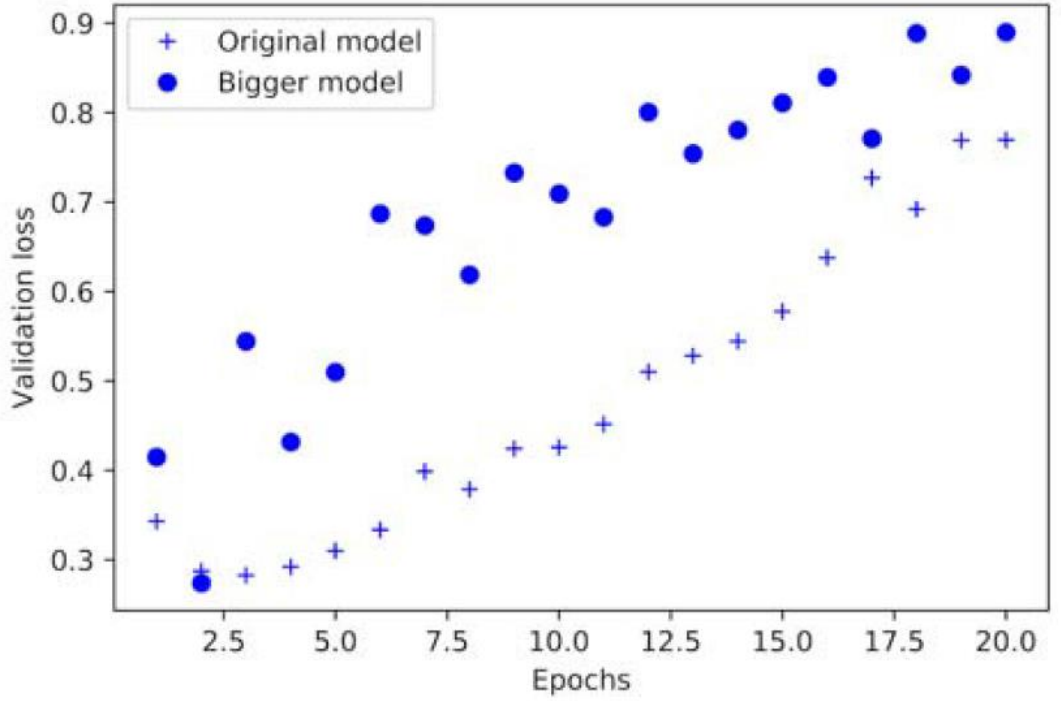
Overfitting with Bigger Model

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

Training Loss



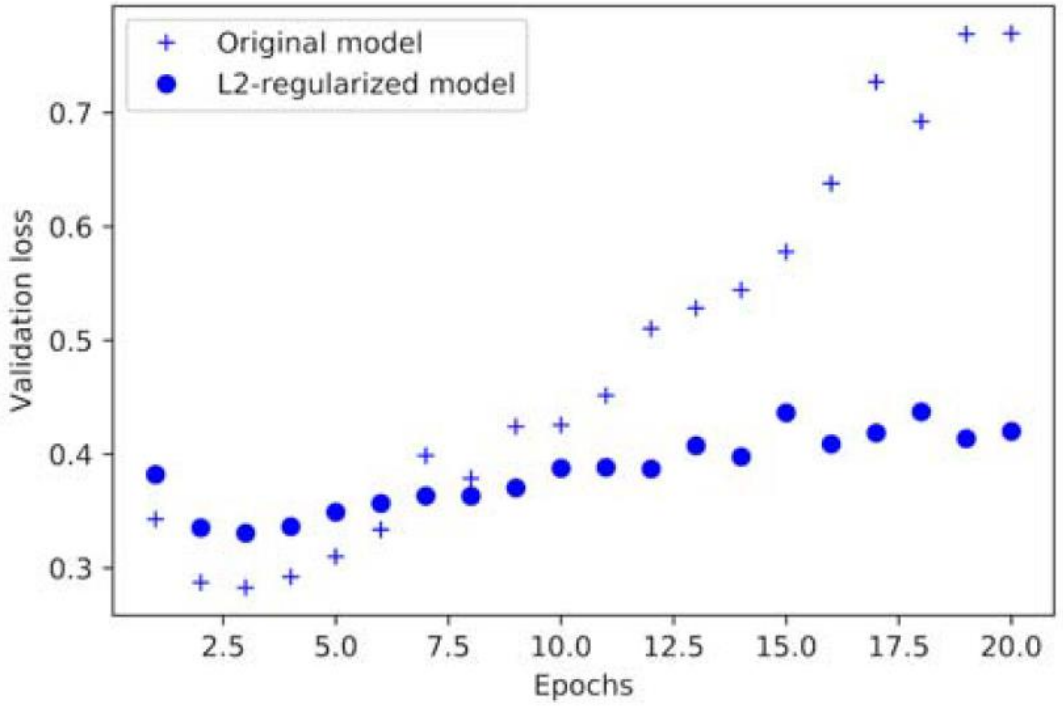
Validation Loss



Regularization

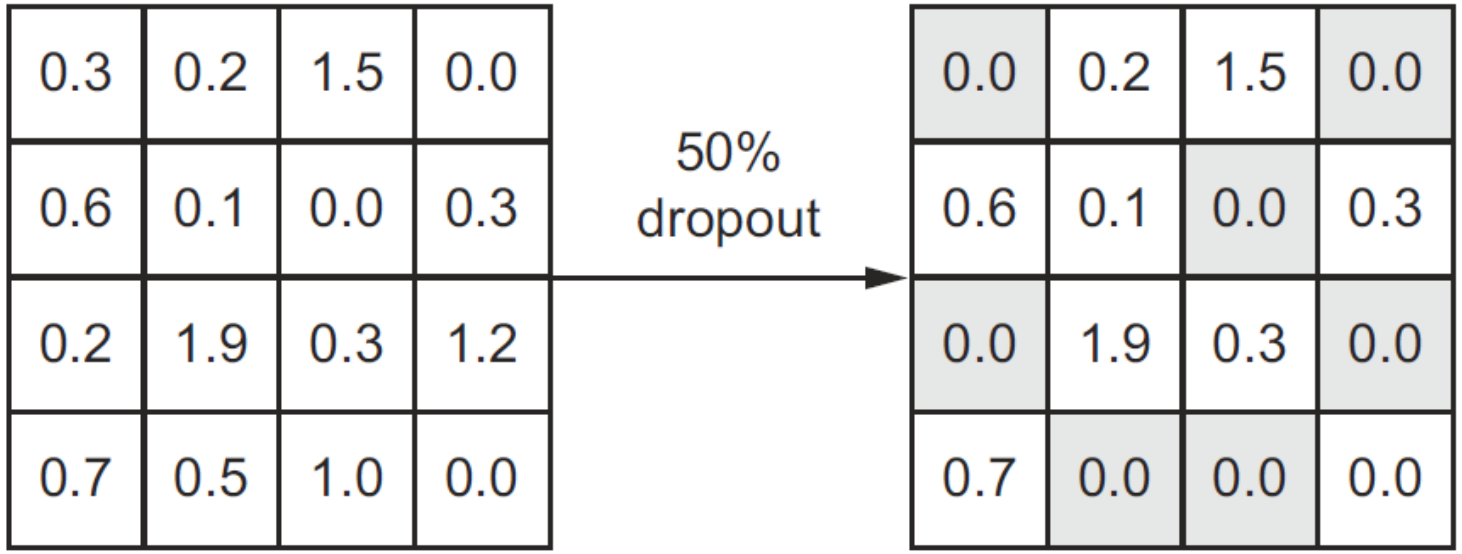
- Weight regularization – L1 and L2 norm

```
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'))
```



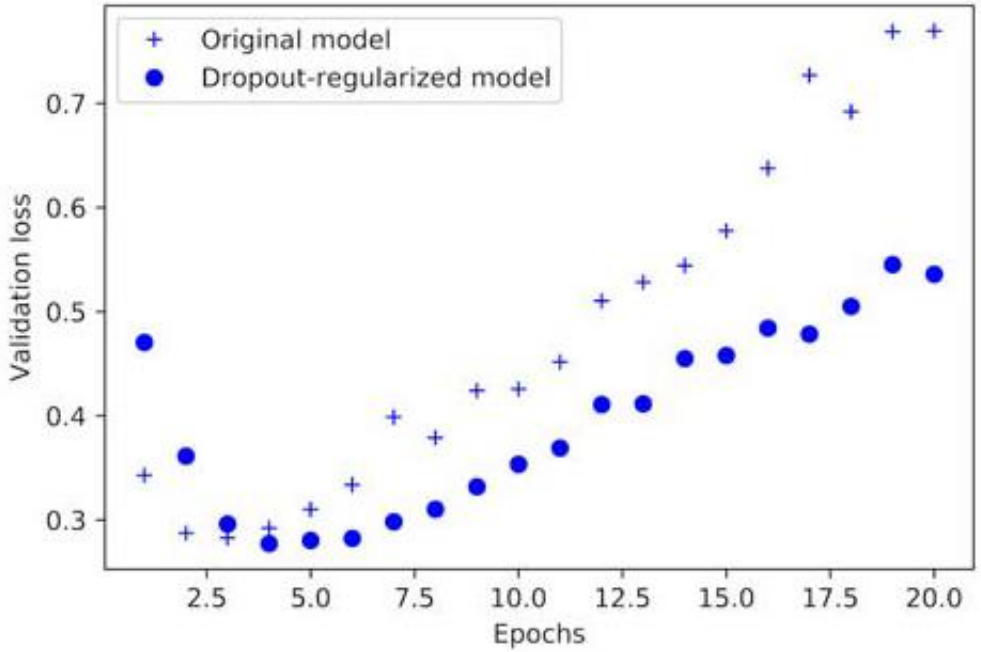
Dropout

- 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 (accuracy, MAE,…)
3. Deciding on an evaluation protocol (squared loss, cross entropy,…)
4. Preparing your data (normalization)
5. Developing a model that does better than a baseline
6. Scaling up: developing a model that overfits
7. Regularizing your model and tuning 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`

Metrics:
Accuracy vs. Precision
in Binary Classification

Confusion Matrix

- Binary classification (positive vs. negative)

		True condition Y (labels)	
	Total population	Condition positive	Condition negative
Predicted condition y'	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

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

Confusion Matrix

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**	**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, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	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 $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		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−) $= \frac{\text{FNR}}{\text{TNR}}$		

Coronavirus Confusion

- Error rate = 80%
- Did the medical staff misuse the test kit?
- Can we just swap positive class and negative class and achieve 80% accuracy?

CZECHIA

80% of Rapid COVID-19 Tests the Czech Republic Bought From China are Wrong

BY PRAGUE MORNING · MARCH 26, 2020 · 1K SHARES · 2 MINUTE READ

Up to 80 percent of the 300,000 rapid coronavirus test kits ordered that the Czech Republic ordered from China are not working properly, according to regional hygienists who have tried the tests.

The test kits, worth 54 million korunas (€1.83 million), show false positive as well as negative results. This is because rapid tests cannot reliably detect infection in its initial phase.

https://praguemorning.cz/80-of-rapid-covid-19-tests-the-czech-republic-bought-from-china-are-wrong/

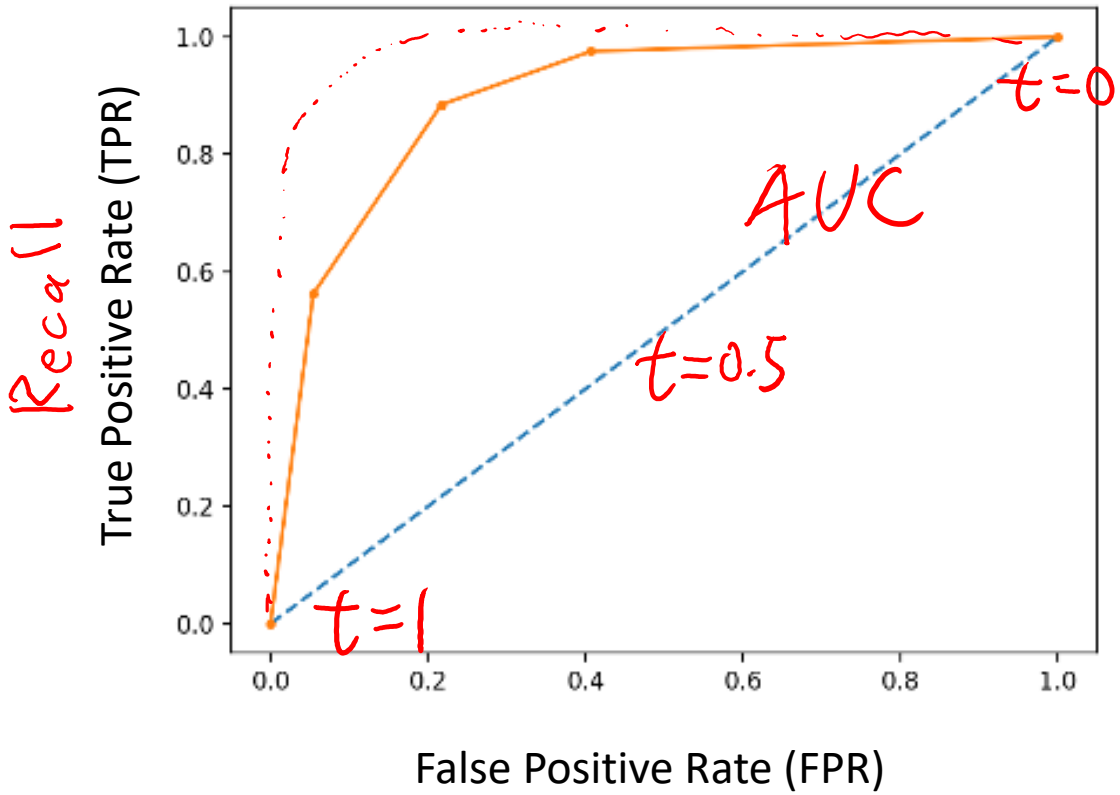
#數感生活 | 一種試劑，不只一種錯誤率

Popular Metrics

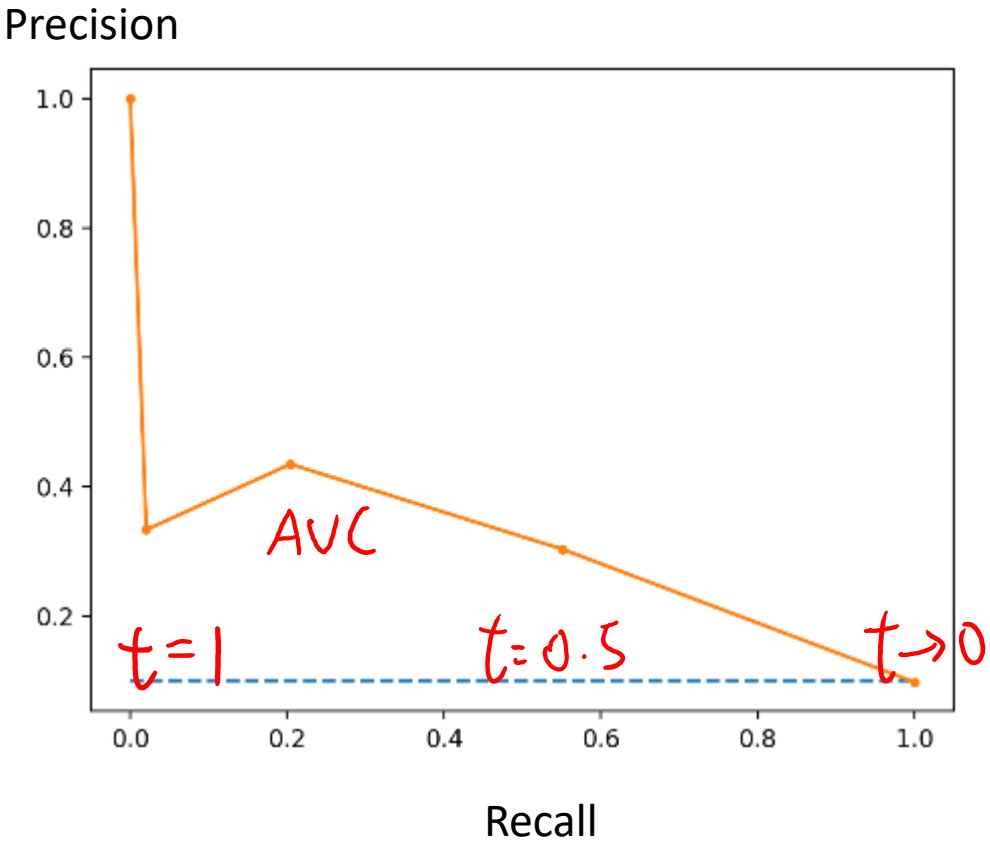
- Notations
  - P: positive samples, N: negative samples, P’: predicted positive samples, TP: true positives, TN: true negatives
- Recall = $\frac{\mathrm{TP}}{\mathrm{P}}$
- Precision = $\frac{\mathrm{TP}}{\mathrm{P}'}$
- Accuracy = $\frac{\mathrm{TP}+\mathrm{TN}}{P+\mathrm{N}}$
- F1 score = $\frac{2}{\frac{1}{recall}+\frac{1}{precision}}$
- Miss rate = false negative rate = 1 – recall

Evaluate Decision Boundary *t*

- ROC (Receiver Operating Characteristic) Curve



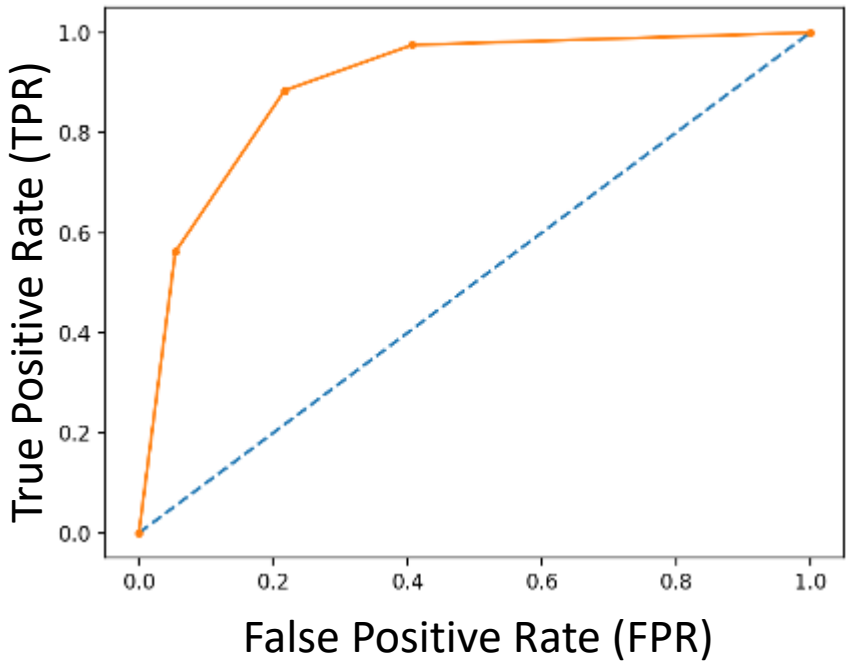
- Precision-Recall (PR) Curve



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

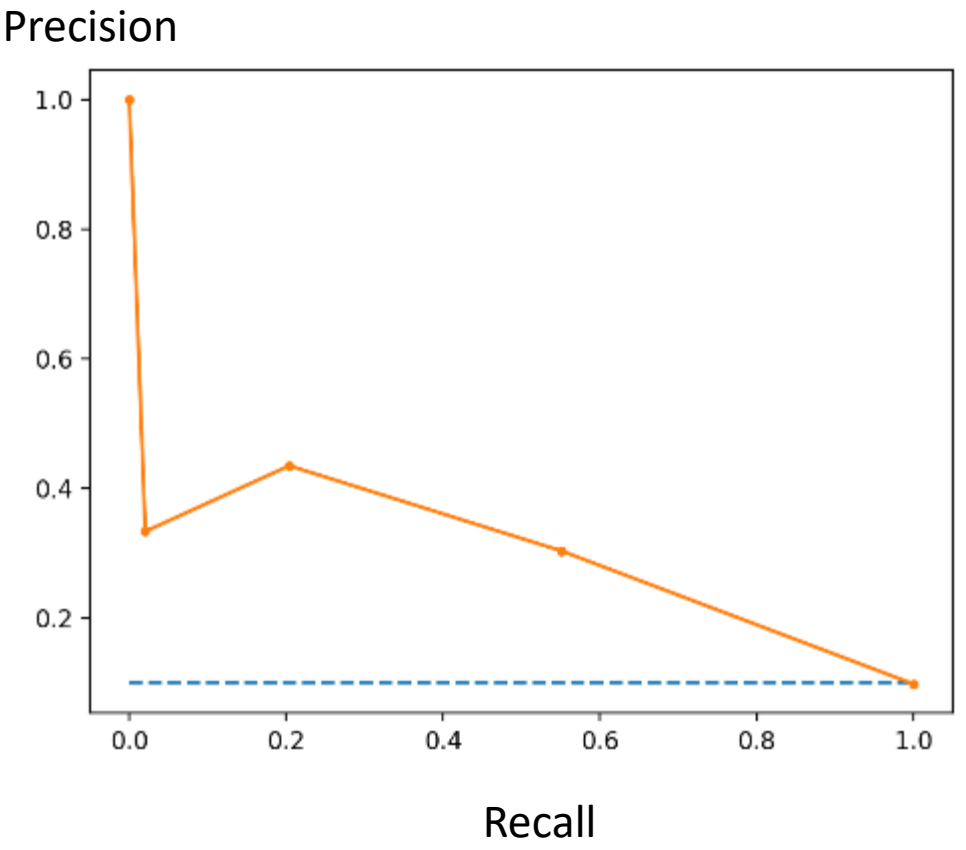
```
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, 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
- 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

Accuracy

- Example: Classifying tumors as malignant or benign (Google Machine Learning Crash Course)

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

https://developers.google.com/machine-learning/crash-course/classification/accuracy

Accuracy

- Example: Classifying tumors as malignant or benign (Google Machine Learning Crash Course)

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

https://developers.google.com/machine-learning/crash-course/classification/accuracy

Precision and Recall

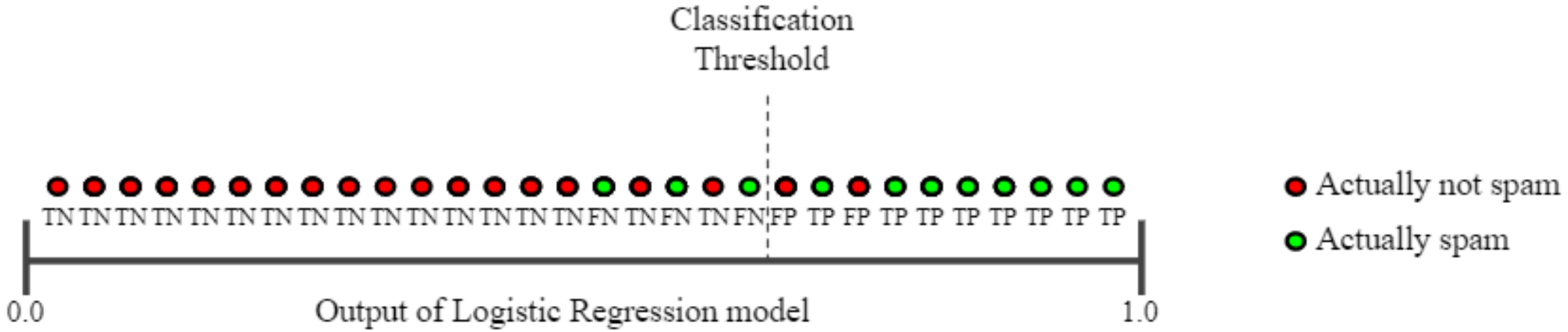
- Classifying tumors as malignant or benign

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1+1} = 0.5$$

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1+8} = 0.11$$



https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Precision and Recall

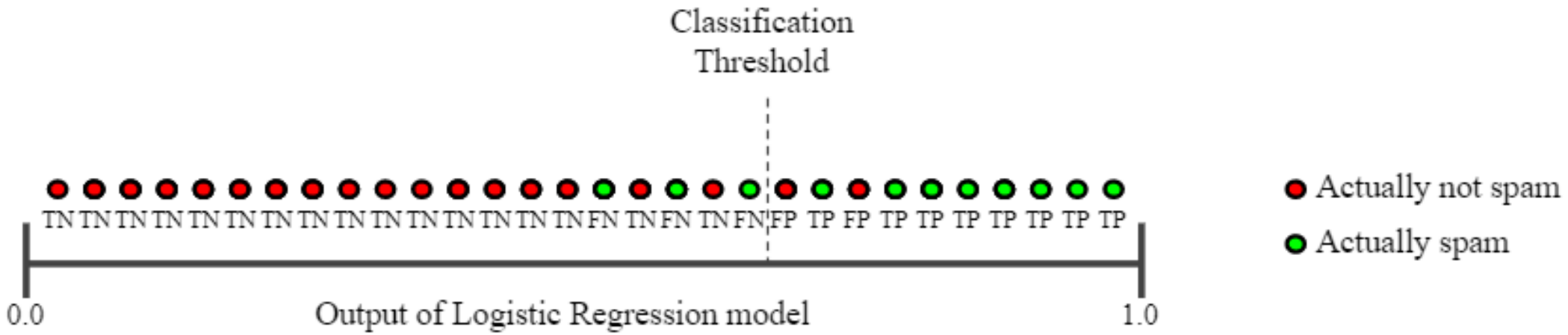
- Classifying tumors as malignant or benign

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{1}{1+1} = 0.5$$

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1+8} = 0.11$$



https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

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

Key Takeaways

- Classification
  - KNN, Linear Classification, SVM, Naïve Bayes, Logistic regression
- Regression
  - Linear regression, Ridge Regression, Lasso Regression, Elastic Net
- Avoid Overfitting
  - Regularization, k-Fold Cross validation
- Confusion Matrix
  - Accuracy vs. Precision, Recall
  - ROC curve, PR curve and AUC

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
- https://en.wikipedia.org/wiki/Confusion_matrix
- https://developers.google.com/machine-learning/crash-course/classification/accuracy