Data Preprocessing and Machine Learning

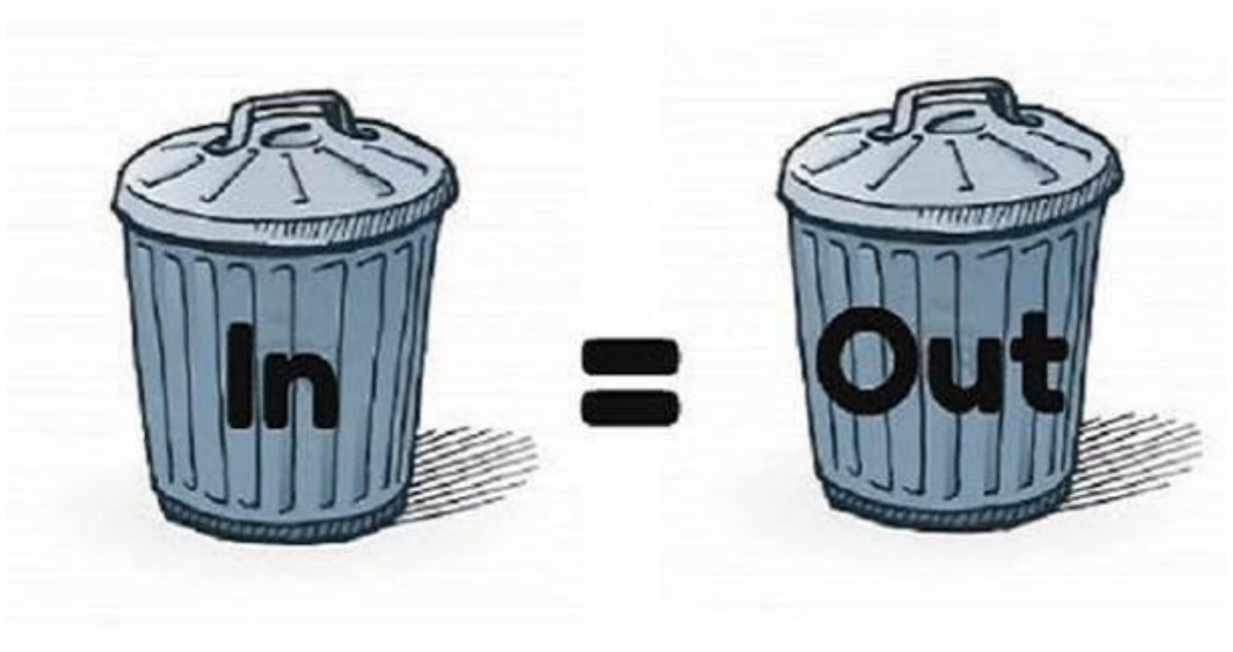
Dr. Jinjun Xiong
(jx2308@columbia.edu)

Outline

- Importance of data preprocessing
- Missing data, scaling data, encoding categorical data
- General paradigm of machine learning
- Supervised vs unsupervised ML
- Data for ML
- Metrics for ML
- Models for ML
- The curse of dimensionality & dimensionality reduction
- Conclusion

Importance of data preprocessing

- Data preprocessing is to make sure we have sensible data for ML



[https://www.linkedin.com/pulse/supply-chain-planning-garbage-out-cheryl-wiebe]

Outline

Some data issues need to be addressed before applying ML algorithms

- Missing values
  - Observation we intended to collect but did not get them
    - Data entry issues, equipment errors, incorrect measurement etc
      - An individual may only responded certain questions in a survey, but not all (“sounds familiar”)
  - Problems of missing data
    - Reduce representativeness of the sample
    - Complicating data handling and analysis
    - Bias resulting from differences between missing and complete data

- Data in different scales
  - Weight of a person (Pounds) vs weight of an elephant (US ton)
    - 1 US ton = 2000 Pounds
  - For predicting weights of them, the error of elephant weights will significantly bias the prediction accuracy than the error of persons weights

Missing data handling

- Reducing the data set
  - Elimination of samples with missing values
  - Elimination of features (columns) with missing values

- Imputing missing values
  - Replace the missing value with the mean/median (numerical) or most common (categorical) value of that feature

- Treating missing attribute values as a special value
  - Treat missing value itself as a new value and be part of the data analysis

“Applied Missing Data Analysis” C. Enders, 2010

Data in different scale

- Approaches to bring different values onto the same scale
  - Normalization: rescale the feature to a range of [0,1]
  - Standardization: re-center the feature to the mean and scaled by variance

$$x_{norm}^{(j)} = \frac{x^{(j)} - x_{min}}{x_{max} - x_{min}}$$

x_{min} and x_{max} are the min/max values of feature column $x^{(j)}$

$$x_{std}^{(j)} = \frac{x^{(j)} - \mu_x}{\sigma_x}$$

μ_x and σ_x are the mean and standard deviation of feature column $x^{(j)}$

- Data scaling should be one of the first steps of data preprocessing for many machine learning algorithms
  - Some machine learning algorithms can handle data indifferent scales (e.g., decision trees and random forests)

Categorical data handling

- for ordinal data, convert the strings into comparable integer values
  - E.g., XL > L > M > S → 5 (XL) > 4 (L) > 3 (M) > 2 (S)
  - Note that the value of integer itself has no special meaning besides for ordering
  - Mapping needs to be unique: 1 to 1 mapping for going back and forth
- For nominal data, convert the strings into integers
  - E.g., Red (0), Blue (1), Green (2)
  - A common practice to avoid software glitches in handling strings
  - Note that the value of integer itself has no special meaning (non-comparable)
  - Mapping needs to be unique: 1 to 1 mapping for going back and forth
- To avoid mistakenly compare encoded integers for nominal data, one-hot encoding can be used
  - Each unique value becomes a separate dummy feature

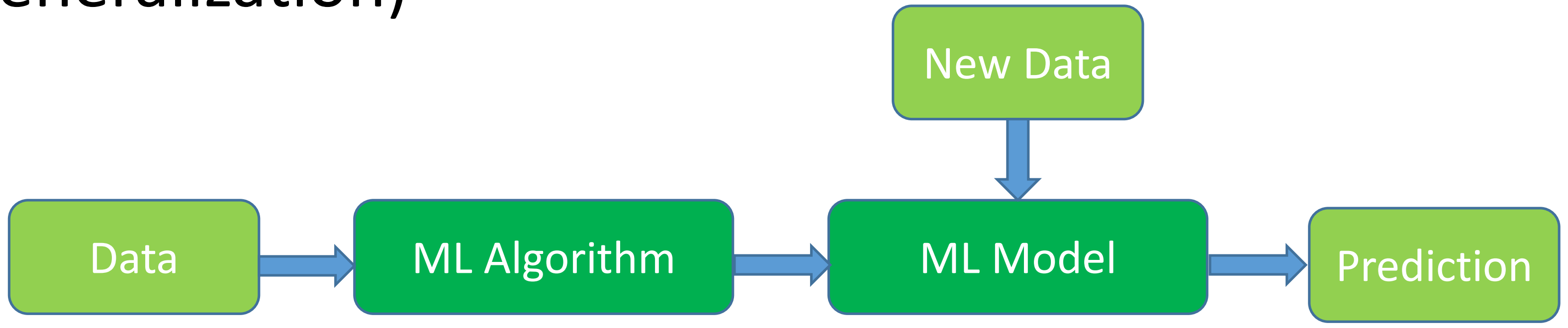
$$Red = [1,0,0]^T, Blue = [0,1,0]^T, Green = [0,0,1]^T$$

Outline

- Importance of data preprocessing
- Missing data, scaling data, encoding categorical data
- General paradigm of machine learning
- Supervised vs unsupervised ML
- Data for ML
- Metrics for ML
- Models for ML
- The curse of dimensionality & dimensionality reduction
- Conclusion

Machine learning, models and data

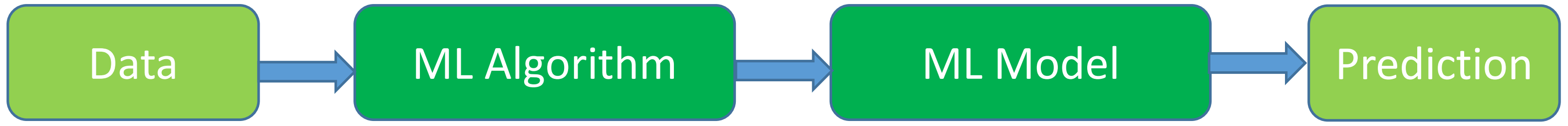
- Machine learning is an algorithm that learns a model from data (training), so that the model can be used to predict certain properties about new data (generalization)



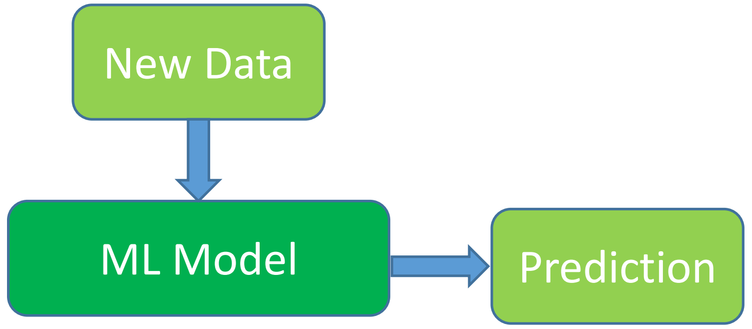
- Different roles
  - ML algorithm developers: who develop new ways of training ML models (and implemented in some software packages, such as R, Python etc)
  - ML model developers: who designs the model and applies appropriate ML algorithms to train the model parameters on data
  - ML application end users: who apply new data to the model to gain insights about the new data (model generalization)

Training vs Inference

- Training is to build the ML model from data



  - Typically, training is a one-time effort, but computationally intensive
  - Speed is a main concern
- Inference is to use the ML model to predict results for new data (generalization – most interesting for applications)



  - Typically, inference is fast but happens more frequently with a lot of more new data (unlabeled)
  - Scalability is a main concern

Outline

What data to use for learning and what to learn? Supervised vs Unsupervised

- If data contains a set of features (factors) that are easy to obtain in practice, and at least one feature that is difficult to obtain
  - The goal of ML becomes clear: can we build a ML model that can help predict that one “difficult” feature based on a set of “easy” features?
  - If the feature of interest is numerical, ML = regression
  - If the feature of interest is categorical (mostly nominal), ML = classification
    - If two categories, ML = binary classification
    - If multiple categories, ML = multiclass classification
  - We’re typically given a set of data with features of interest clearly labeled, and we use these data to train a ML model. This is also called “supervised” learning
- If all features in the data are equally easy to obtain, which is a nice way to say that we can’t precisely define what to look for, ML becomes an exploration problem
  - Discover the hidden structures in data that we don’t know the right answer upfront
  - This is also called “unsupervised” learning

Some common types of ML problems

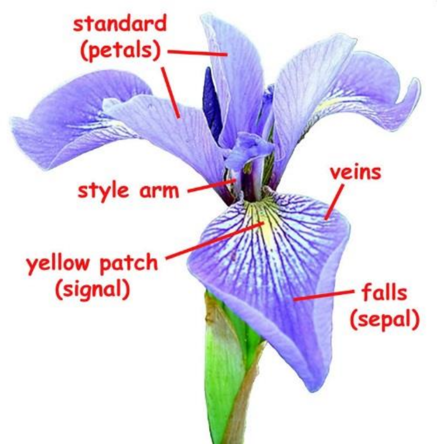
- Binary classification for nominal categorical data
  - Yes or No for e-mail spams
  - Positive or Negative for breast cancers
- Multiclass classification can be easily handled by a binary classification
  - One vs all (OVA) method
    - Treating one class as one label, and all other classes as another label
    - Run binary classifier N times for each class, and pick the one with the highest score
  - We'll discuss some ML models that can handle multiclass classification directly
- Regression
  - Predicting company sales in the future months
- Clustering (without labels)
  - Group users of interests for social studies

How do we tell what ML problems we have?

- If we are just presented the data, we would not know

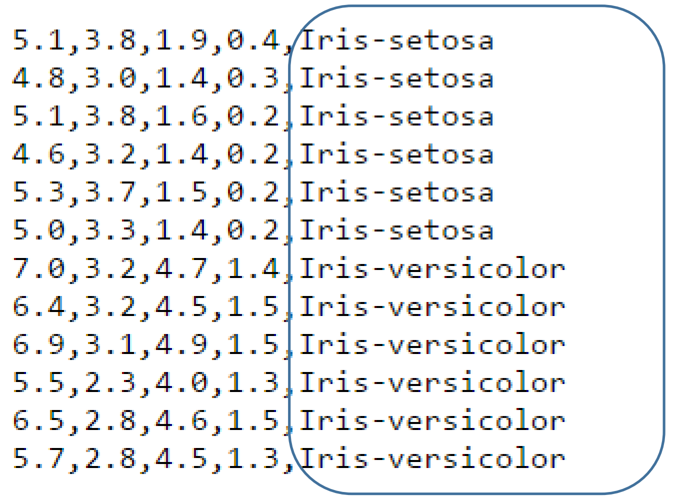
```
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
```

- Even if we know the above data are about Iris Sepal length, Sepal width, Petal length, Petal width, and Species, we still would not know
  - Not necessary all "known" features are presented in the data either



ML problem formulation depends on application specific knowledge

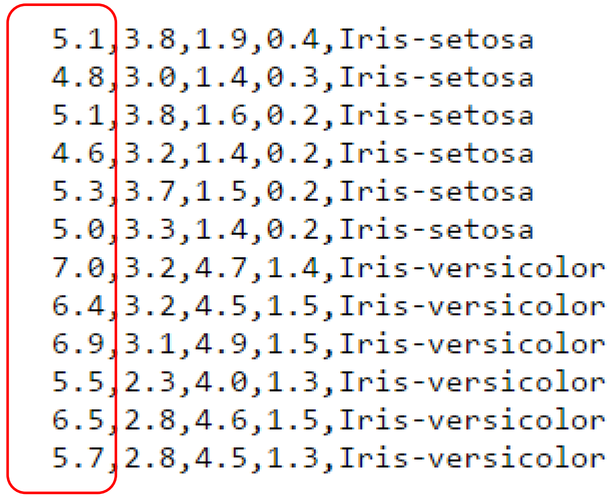
- This is where application specific knowledge kicks in
- If we are interested in identifying Iris species based on “seemingly” easy to measured iris sepal length and width, and petal length and width
- The problem would become a classification problem: Iris species becomes the feature of interests (now called class labels, targets, goals)
- These data would become labeled data for training the ML model



```
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
```

ML problem formulation depends on application specific knowledge

- It would be equally fine if someone for some applications wants to predict the Sepal length based on the known Iris species and sepal width, and petal length and width
- The problem would become a regression problem, and Iris sepal length would become our target variable, and the others are explanatory variables

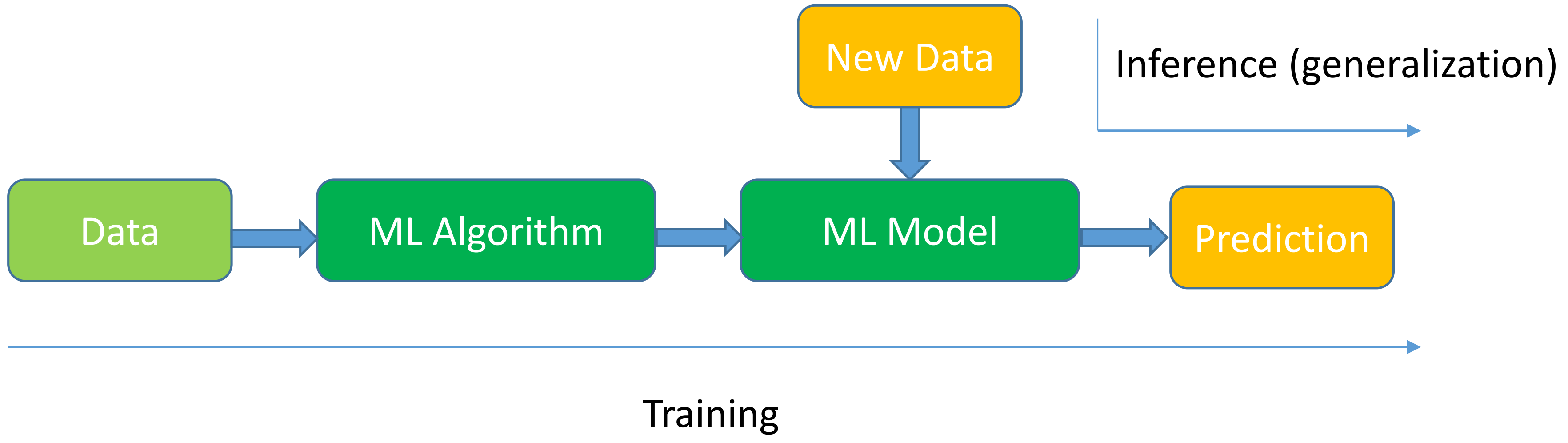


```
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
```

Outline

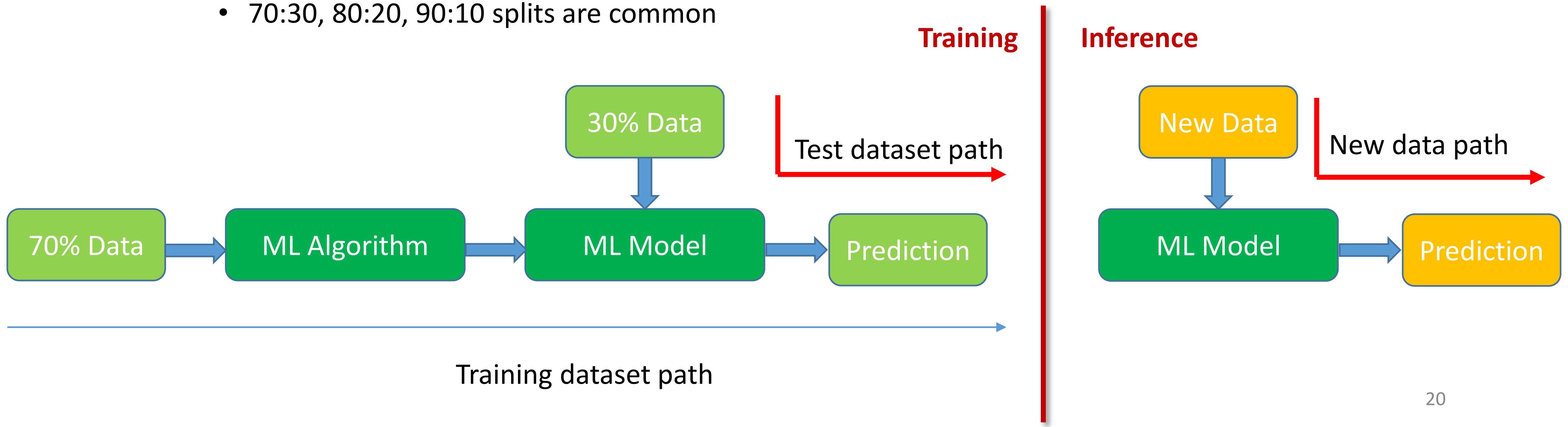
Data for ML: training and test dataset splits and cross-validation

- New data are typically unknown to ML algorithm and model developers, and only be used by end users to judge the quality of ML
- How do we know if the trained ML model would generalize well for the unknown new data?



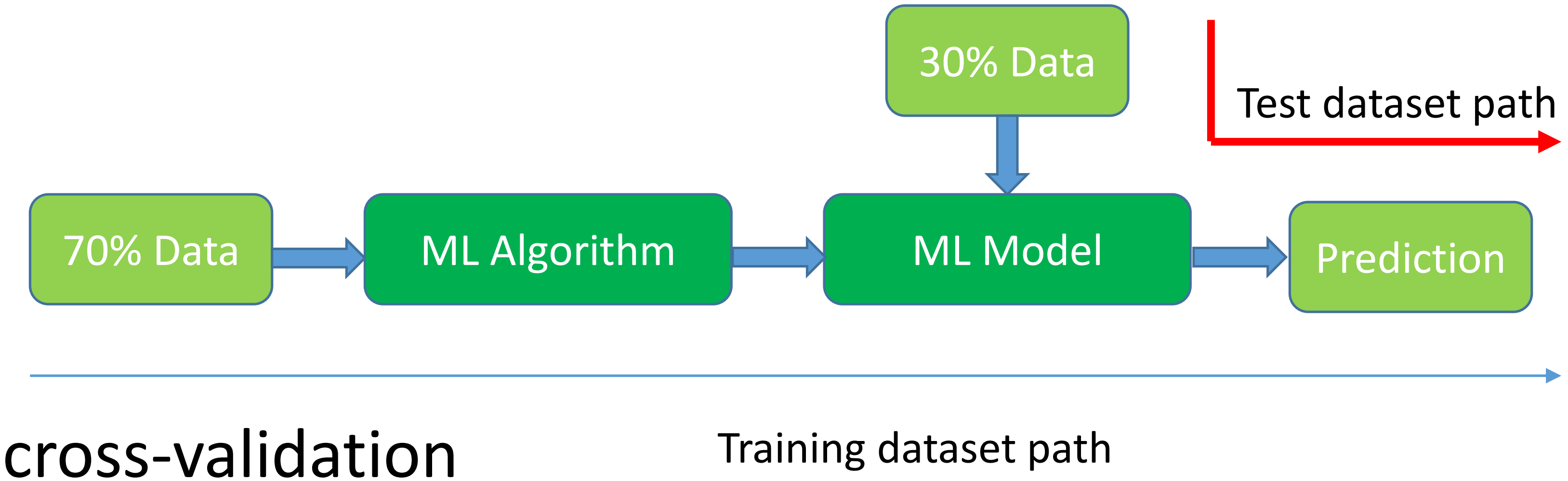
Split known data into training and test datasets

- Data known to ML model developers are split into two sets
  - Training dataset: data used to train the model
  - Test dataset: data used to give an indication on how well the trained model will generalize to new data (unknown at this point)
    - Test dataset is kept till the very end to evaluate the final model
    - Since test dataset withholds valuable information that the learning algorithm could benefit from, we don't want to put too much data into the test dataset either
      - 70:30, 80:20, 90:10 splits are common



Cross-validation: a model tuning process

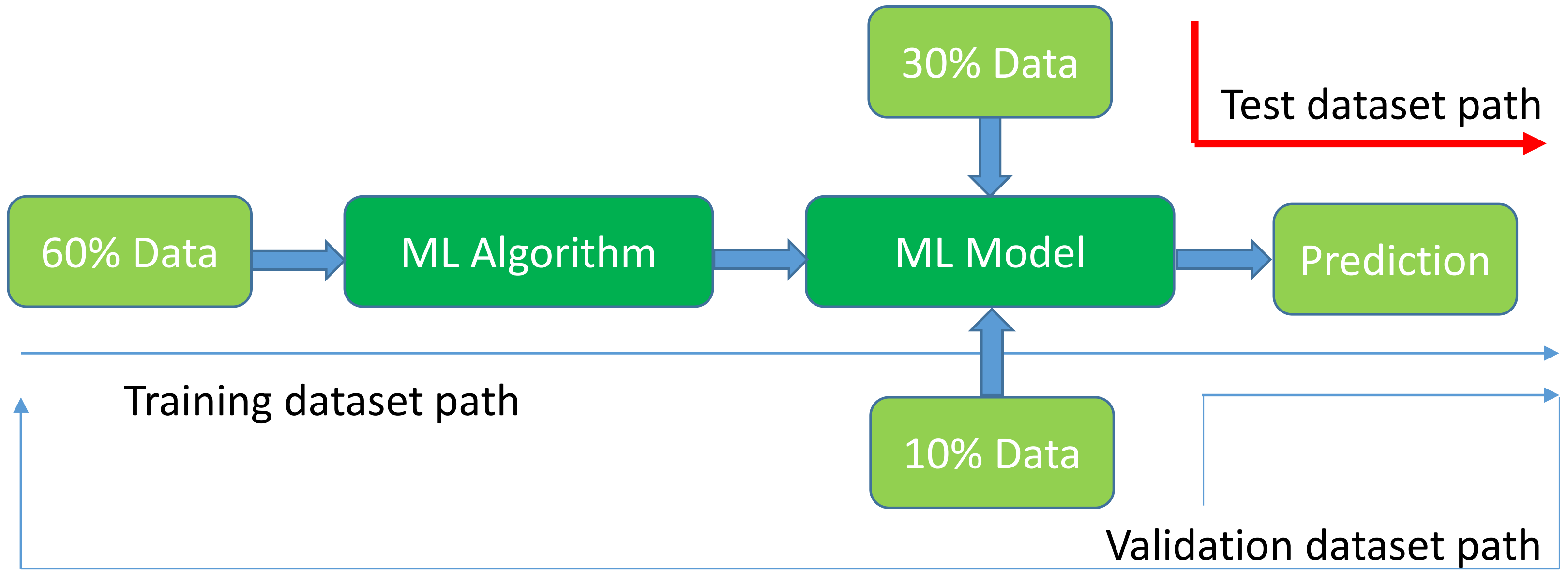
- How can we make the model training process to be aware of the targeted generalization quality so that training can do something about it?
- We need to put the predicated generalization results as part of the training optimization goal
  - We can NOT use the predicated generalization results from the test data, otherwise, the test data would become part of the training process
  - We want to keep the test data still independent of training so that its predication can still be a good indication of generalization quality for future unknown new data



- Solution: cross-validation

Holdout cross-validation

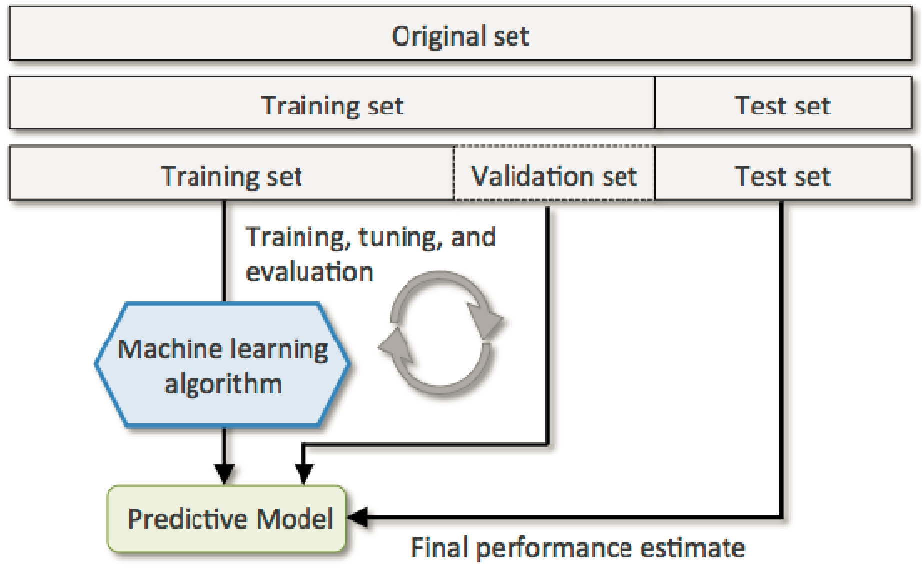
- Holdout cross-validation method
  - Training dataset is further split into two sets: training set + validation set



- Validation results are used to drive the continuation of training process
  - Until we obtain a reasonable validation result
- We still use test data to report the predicated generalization quality

Pros and Cons of holdout cross-validation

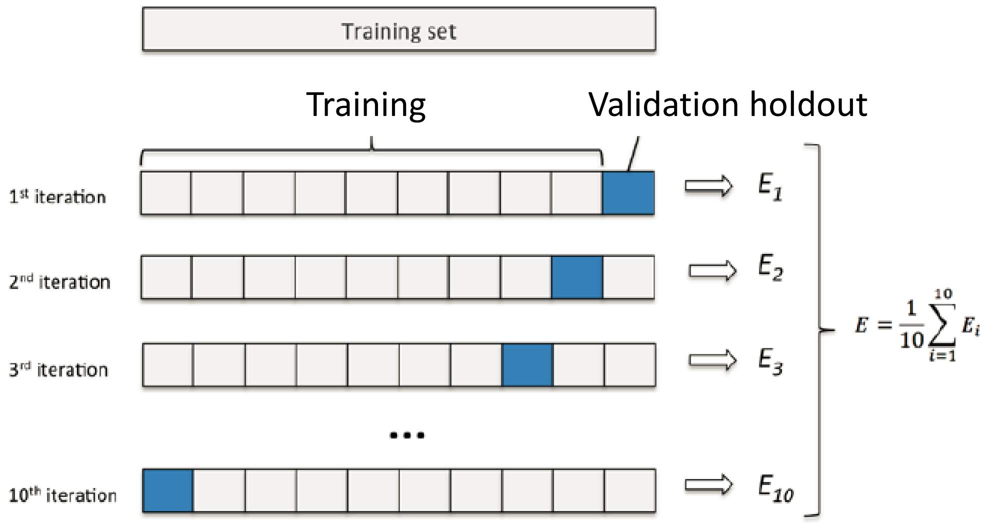
- Another view of the holdout cross-validation


Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

- Pros: validation set is used to tune the model parameters for better generalization
- Cons: final results may be sensitive to how the dataset was split for validation

K-fold cross-validation

- Repeat holdout cross-validation k times on k subsets of the training data
  - Randomly split the training dataset into k folds without replacement
    - K-1 folds are used for training, and one fold used for validation
  - Repeat this k times so that we obtain k models
  - Typically k=10, but larger k for smaller dataset, and smaller k for larger dataset



- Pro: average performance from k models is less sensitive to the split
- Con: more computation time

Outline

Metrics for ML

- So far we have only been hand-waving about the quality metric
  - Training quality
  - Test quality
  - Generalization quality
- What do we really mean by all of these?
- We need to have a more precise definition for the metric of interest
- Different applications and ML techniques may optimize for different metrics
- It's important to understand some of the most commonly used metrics
- We'll introduce these concepts through a binary classification problem
  - Easy to explain the key intuitions behind those metrics

Metrics for binary classification

- Confusion matrix for a binary classification problem

		Predicted class			
		N	**P**		
Actual class	**N**	True Negative (TN)	False Positive (FP)	FPR(False Positive Rate) = FP/(FP+TN)	**Specificity=1-FPR**
	P	False Negative (FN)	True Positive (TP)	TPR(True Positive Rate)=TP/(TP+FN)	**Recall**=TPR **Sensitivity=TPR**
Error			**Precision**=TP/(TP+FP)	**Accuracy**	

$$Error = \frac{FN + FP}{FP + FN + TP + TN}$$

$$Accuracy = 1 - Error = \frac{TP + TN}{FP + FN + TP + TN}$$

$$F1 = 2\frac{Precision \times Recall}{Precision + Recall}$$

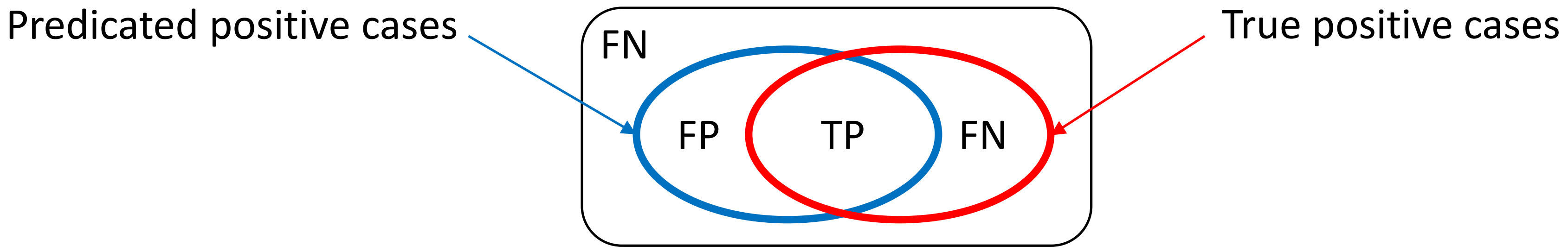
Precision and recall

- Precision: proportion of predicted positive items that are correct

$$Precision = \frac{TP}{TP + FP}$$

- Recall: proportion of true (target) positive items that are predicted positive

$$Recall = \frac{TP}{TP + FN}$$



Metrics for multiclass classification

- Similar concepts can be defined for a multiclass classification problem as well if it is solved via One vs. All (OVA) methods
- For each binary classifier under OVA
  - Calculate the confusion matrix
- Micro-average and macro-average of the metrics
  - Take precision as an example

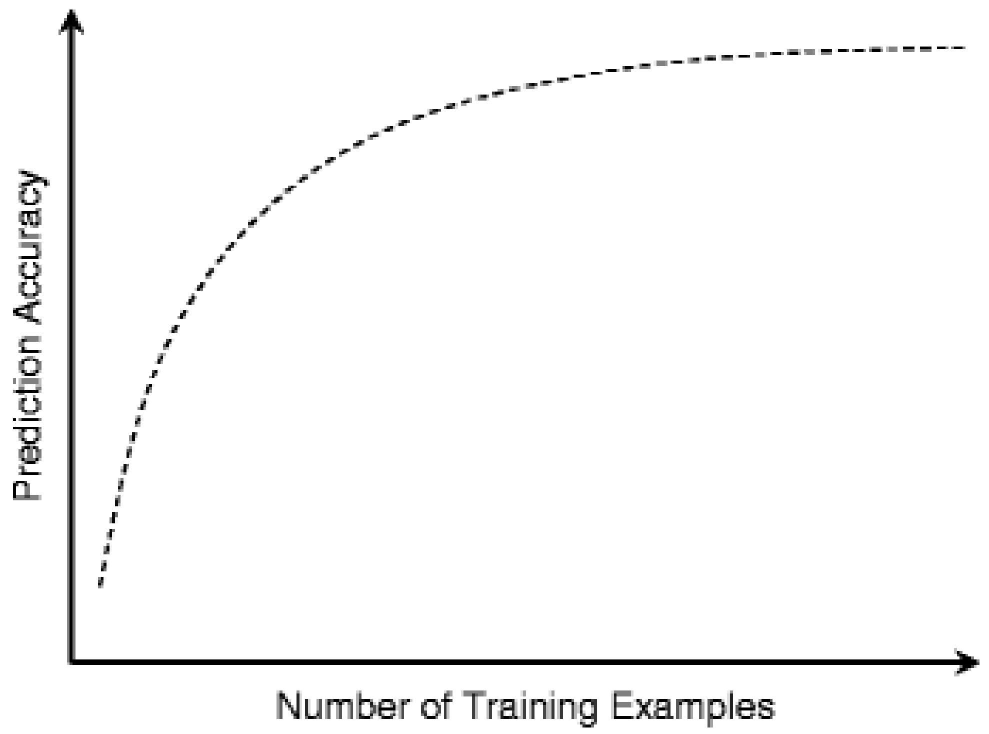
$$Precision_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

$$Precision_{macro} = \frac{Precision_1 + \cdots + Precision_k}{k}$$

  - Micro-averaging weights each instance or prediction equally
  - Macro-averaging weights all classes equally

Learning curve

- Plot the training accuracy (and test accuracy) as a function of sample size
  - As a function of learning effort



Outline

- Importance of data preprocessing
- Missing data, scaling data, encoding categorical data
- General paradigm of machine learning
- Supervised vs unsupervised ML
- Data for ML
- Metrics for ML
- Models for ML
- The curse of dimensionality & dimensionality reduction
- Conclusion

How does a ML model look like?

- A ML model can be abstractly presented as a complex function (not necessarily differentiable)

Machine learning model: $H(w, x, p)$

where:
x is the input feature vector for a data sample
w is the model weight (parameter) vector, typically a large number
p is the model hyper-parameter vector, typically a relative small number
$H()$ gives the predicted feature of interests for a data sample x

- Model weights and model hyper-parameters are typically trained separately
  - Weights obtained via training
  - Hyper-parameters are tuned separately from the ML training process
    - Hyper-parameters are given for training the weights
- Intuitively, the more the parameters, the more powerful (complex) the model

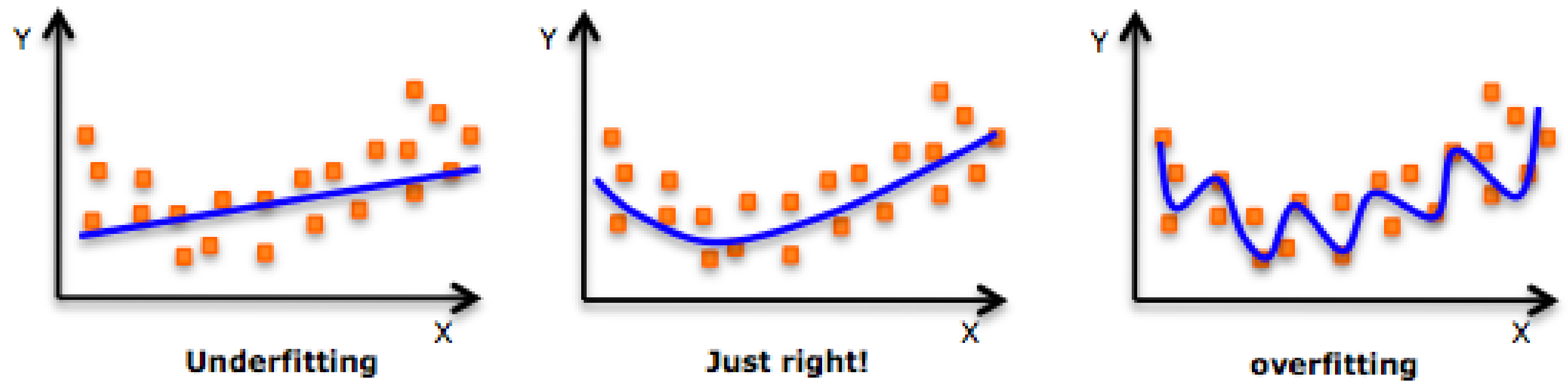
Parametric vs non-parametric models

- Using parametric models, we estimate parameters from the training dataset to learn a function that can classify new data points without requiring the original training dataset anymore

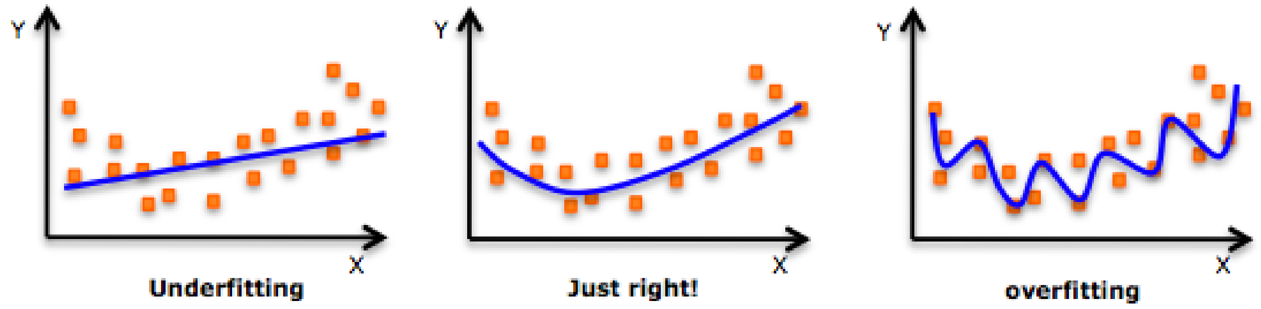
- Nonparametric models can't be characterized by a fixed set of parameters, and the number of parameters grows with the training data
  - Decision tree / random forest
  - Kernel SVM
  - K-nearest neighbor (KNN)
    - A subcategory of nonparametric models (instance-based learning)

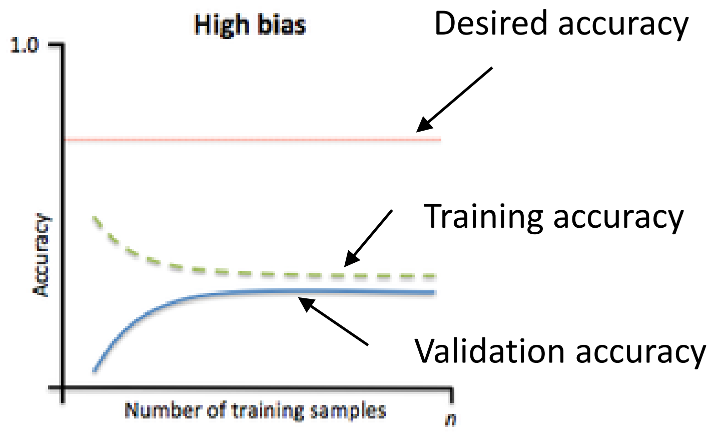
Issues with ML models

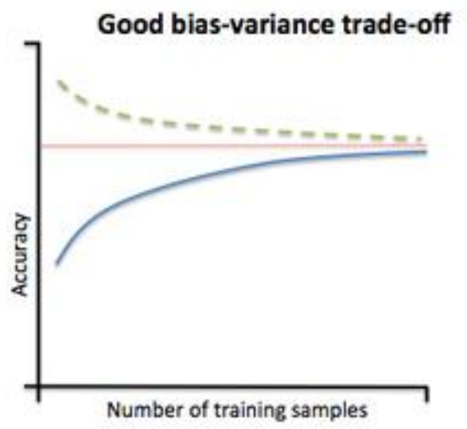
- Underfitting (high bias): model performs poorly even on the training data
  - Model is too simple
- Overfitting (high variance): model performs too well on the training data, but poorly on the unseen data (test data)
  - Model is too complex

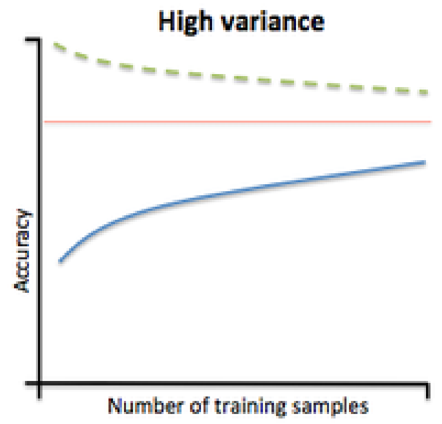


Underfitting (high bias) and overfitting (high variance)









Outline

- Importance of data preprocessing
- Missing data, scaling data, encoding categorical data
- General paradigm of machine learning
- Supervised vs unsupervised ML
- Data for ML
- Metrics for ML
- Models for ML
- The curse of dimensionality & dimensionality reduction
- Conclusion

The curse of dimensionality

- The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces
  - Often with hundreds or thousands of dimensions
- Common theme: when the dimensionality increases, the volume of the space increases so fast that the available data become sparse
- In ML, the feature space becomes increasingly sparse for an increasing number of dimensions of a fixed-size training dataset
  - With a fixed number of training samples, the predictive power reduces as the dimensionality increases
- Use feature selection and dimensionality reduction techniques to help us avoid the curse of dimensionality

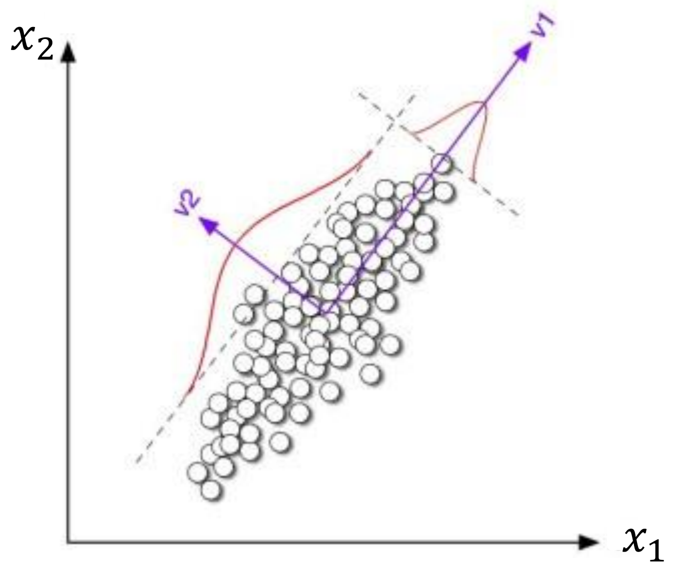
Dimensionality reduction

- Transform features from their original high dimension space into a lower dimension space while keeping as much information about original features as possible
- How could this be potentially possible (or even be helpful)?
  - Some features may be highly correlated, thus including them may not be helpful to train the model
- What information to keep (and ways to measure that)?
  - Different methods have to answer that question first
- Transformation can be linear or nonlinear
  - Linear transformation

$$z = Tx \triangleq \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \text{ with } m \leq n$$

PCA: Principal Component Analysis

- Information: the variation of features contains the most information
    - Keep information = keep the variation as much as possible
- PCA finds a new space with lower dimensions defined by a set of orthogonal axes ("principal components") such that
    - The original data of high dimension will be linearly transformed (mapped or projected) to the low dimension
    - The variation of projected feature along the set of new axes are decreasing
        - The first axis contains the most variation



PCA overall algorithm

- Standardize the n-dimensional original data (features)
- Construct the covariance matrix for features
- Decompose the covariance matrix into its eigenvectors and eigenvalues
  - The eigenvectors define the new axes
  - The ratio of variance explained by each axis corresponding to its eigenvalue
- Select top m largest eigenvalues and their corresponding eigenvectors
  - The linear transformation matrix is defined by the selected m eigenvectors

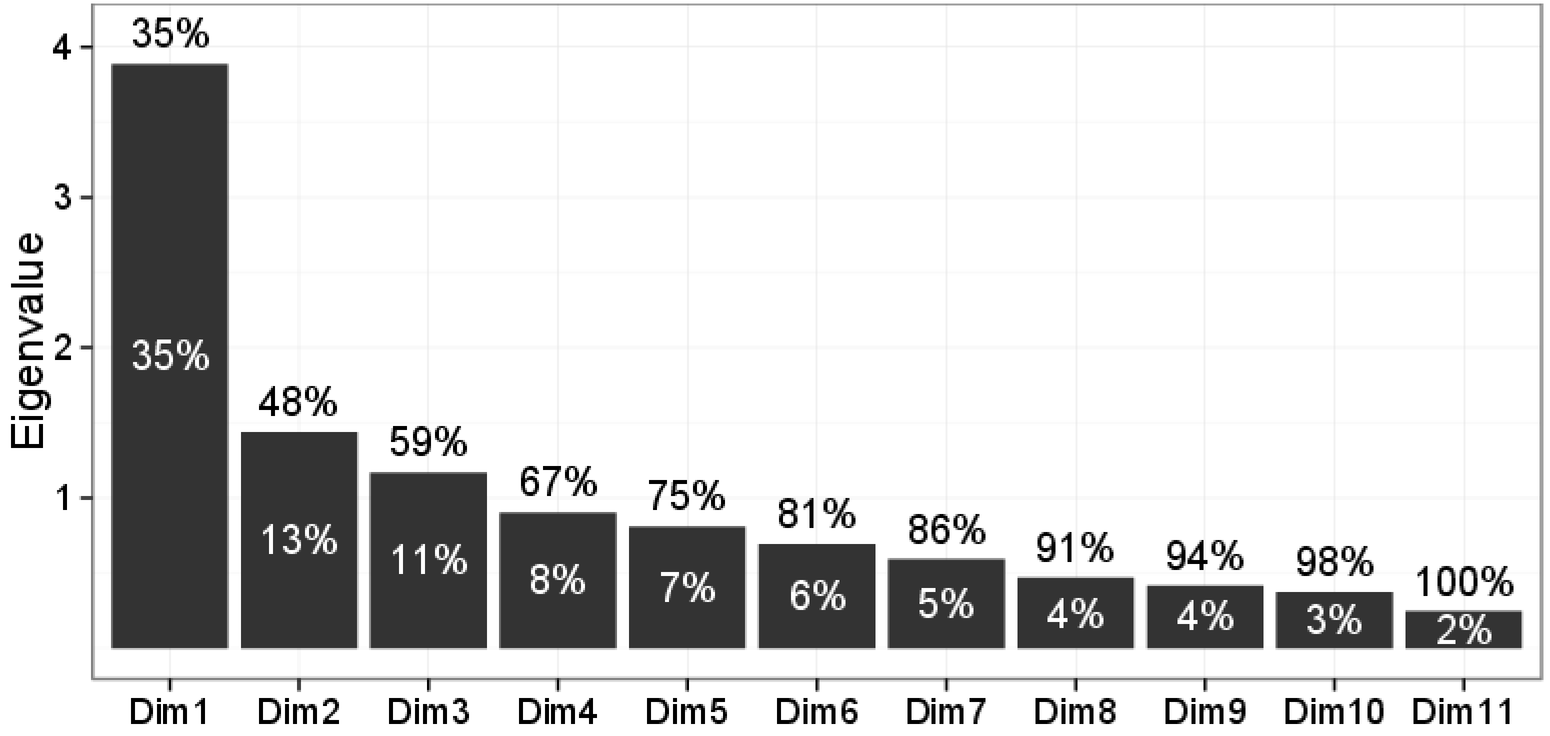
$$T \triangleq \begin{bmatrix} v_1^T \\ \vdots \\ v_m^T \end{bmatrix} = \begin{bmatrix} \alpha_{1,1} & \cdots & \alpha_{1,n} \\ \vdots & \ddots & \vdots \\ \alpha_{m,1} & \cdots & \alpha_{m,n} \end{bmatrix}$$

- Transform the n-dimensional original data into the m-dimensional space

$z^{(j)} = Tx^{(j)}$, for all j from 1 to N

PCA explained variance ratio

- Here is a numerical example, assuming 11-dimensional space



Explained variance ratio for the j^{th} eigenvector $= \frac{\lambda_j}{\sum_{i=1}^{N} \lambda_i}$, for all j from 1 to N with λ_j being the j^{th} eigenvalue

[http://stats.stackexchange.com/questions/133451/is-there-any-required-amount-of-variance-captured-by-pca-in-order-to-do-later-an]

Outline

- Importance of data preprocessing
- Missing data, scaling data, encoding categorical data
- General paradigm of machine learning
- Supervised vs unsupervised ML
- Data for ML
- Metrics for ML
- Models for ML
- The curse of dimensionality & dimensionality reduction
- Conclusion

Conclusions

- Data preparation is important for ML
  - “Garbage in, garbage out”
- Data, model, metrics are closely related to find a good ML solution
  - Training, validation, and test
  - Classification and regression
  - Accuracy, error, precision, recall, etc
- ML typically deal with high dimensional data
  - PCA for dimensionality reduction (and visualization)