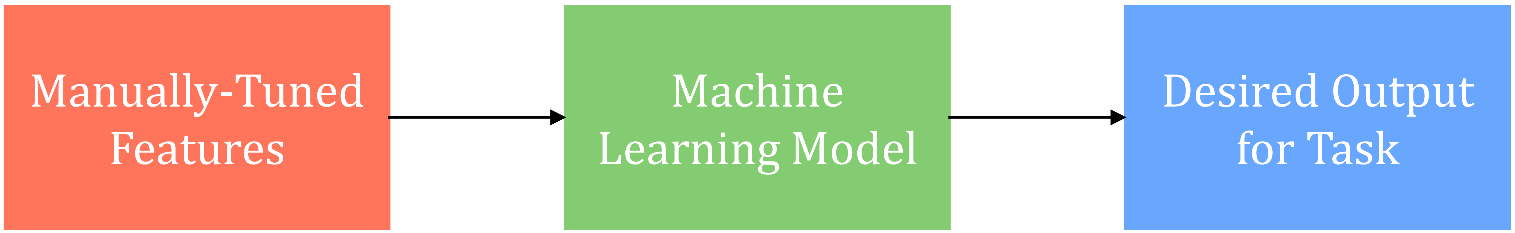
Learning Models of Similarity: Metric and Kernel Learning

Eric Heim, University of Pittsburgh

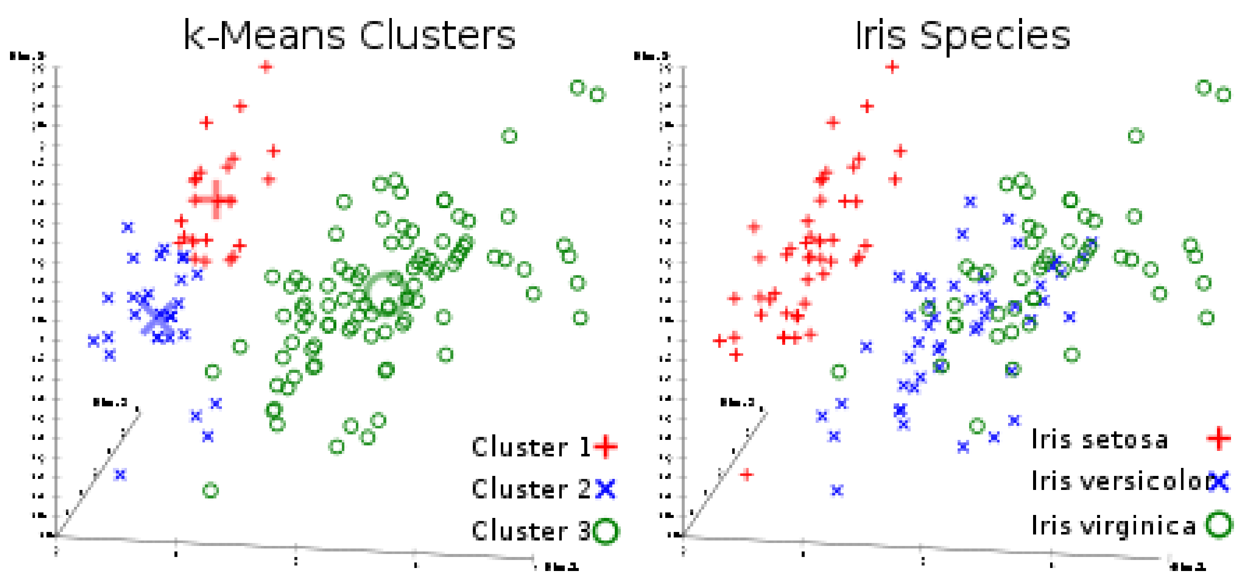
Standard Machine Learning Pipeline



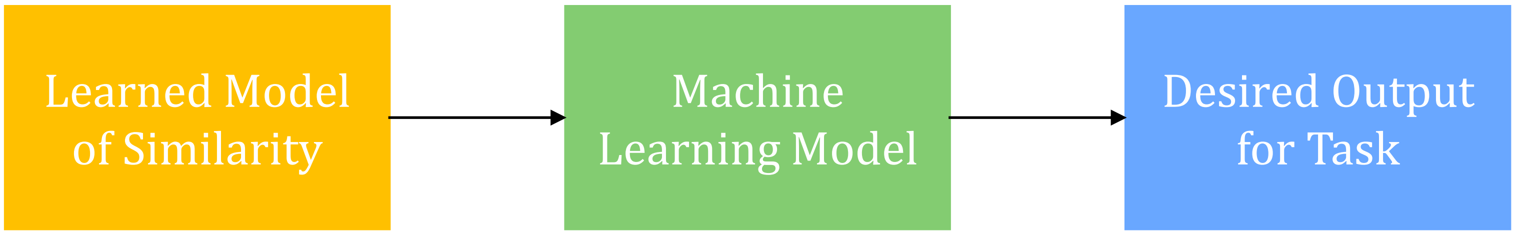
Features MUST be “good” for a model to perform a task!

Standard Machine Learning Pipeline

Manually-Tuned Features → Machine Learning Model → Desired Output for Task



Standard Machine Learning Pipeline



How to Learn a Similarity Model?

- Inputs:
  - Objects as Features
    - $\mathbf{x}_i \in \mathbf{X} \subset \mathbb{R}^d$
  - Constraints
    - Similarity/Dissimilarity
      - $x_i, x_j \in S, x_i, x_k$
    - Set/class membership
      - $x_i \in A, x_j \in B$
    - Relative
      - x_i is more similar to x_j than x_k
  - Tasks
    - Classification, regression, clustering, ranking, etc.
- Methods:
  - What we will focus on throughout the talk.

Outline

- Methods
  - Mahalanobis Distance Metric Learning
  - Kernel Learning
  - Multiple Kernel Learning
- Current Trends
  - Representation Learning

Mahalanobis Distance Metrics

- Mahalanobis Distance:
$$d_{\boldsymbol{\Sigma}}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})$$
- <u>Generalized</u> Mahalanobis Distance Metric:
$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T \mathbf{M}\, (\mathbf{x} - \mathbf{y})$$
- $d_{\mathbf{M}}$ defines the squared Euclidean distance after a linear transformation.
$$\begin{aligned} d_{\mathbf{M}}(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} - \mathbf{y})^T \mathbf{M}\, (\mathbf{x} - \mathbf{y}) \\ &= (\mathbf{x} - \mathbf{y})^T \mathbf{L}^T \mathbf{L}\, (\mathbf{x} - \mathbf{y}) \\ &= (\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{y})^T (\mathbf{L}\mathbf{x} - \mathbf{L}\mathbf{y}) \\ &= d^2(\mathbf{L}\mathbf{x}, \mathbf{L}\mathbf{y}) \end{aligned}$$
- If we learn **M** so that the distances between observed points are "good", then the same distance metric can be applied to unobserved points.
- Note: **M** must be positive semidefinite (PSD) ($\mathbf{M} \in S_+^{dxd}$)

MMC (Xing et al., 2003)

- Main idea: If initial features are bad for clustering, provide an easy way to refine space given feedback.
- Input:

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in \mathbb{R}^d$$
$$S = \{(x_i, x_j) | x_i \text{ and } x_j \text{ are similar}\}$$
$$D = \{(x_k, x_l) | x_k \text{ and } x_l \text{ are dissimilar}\}$$

- Output:

$$\mathbf{M} \in S_+^{d\text{x}d}$$

MMC (Xing et al., 2003)

$$\max_{\mathbf{M}} \sum_{(x_k, x_l) \in D} d_{\mathbf{M}}(\mathbf{x}_k, \mathbf{x}_l)$$

$$\text{s.t} \sum_{(x_i, x_j) \in S} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq 1, \mathbf{M} \in S_{+}^{d\text{x}d}$$

Algorithm:

1. Take objective gradient step w.r.t. **M**
2. Iterate until **M** converges
    1. Project **M** onto feasible region of similarity constraints
    2. Project **M** onto PSD cone
3. Iterate 1-2 until convergence

MMC (Xing et al., 2003)

$$\max_{\mathbf{M}} \sum_{(x_k, x_l) \in D} d_{\mathbf{M}}(\mathbf{x}_k, \mathbf{x}_l)$$

$$\text{s.t} \sum_{(x_i, x_j) \in S} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \le 1, \mathbf{M} \in S_+^{d\text{x}d}$$

Algorithm:

1. Take objective gradient step w.r.t. **M**
2. Iterate until **M** converges
    1. Project **M** onto feasible region of similarity constraints
    2. Project **M** onto PSD cone (O(d^3) operation)
3. Iterate 1-2 until convergence

LMNN (Weinberger et al., 2005)

- Main idea: Learn a metric for k nearest neighbor classification, but without having constraints over every pair of points.
  - Instead, ensure local neighborhoods contain only objects of the same class.
- Input:

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in \mathbb{R}^d$$

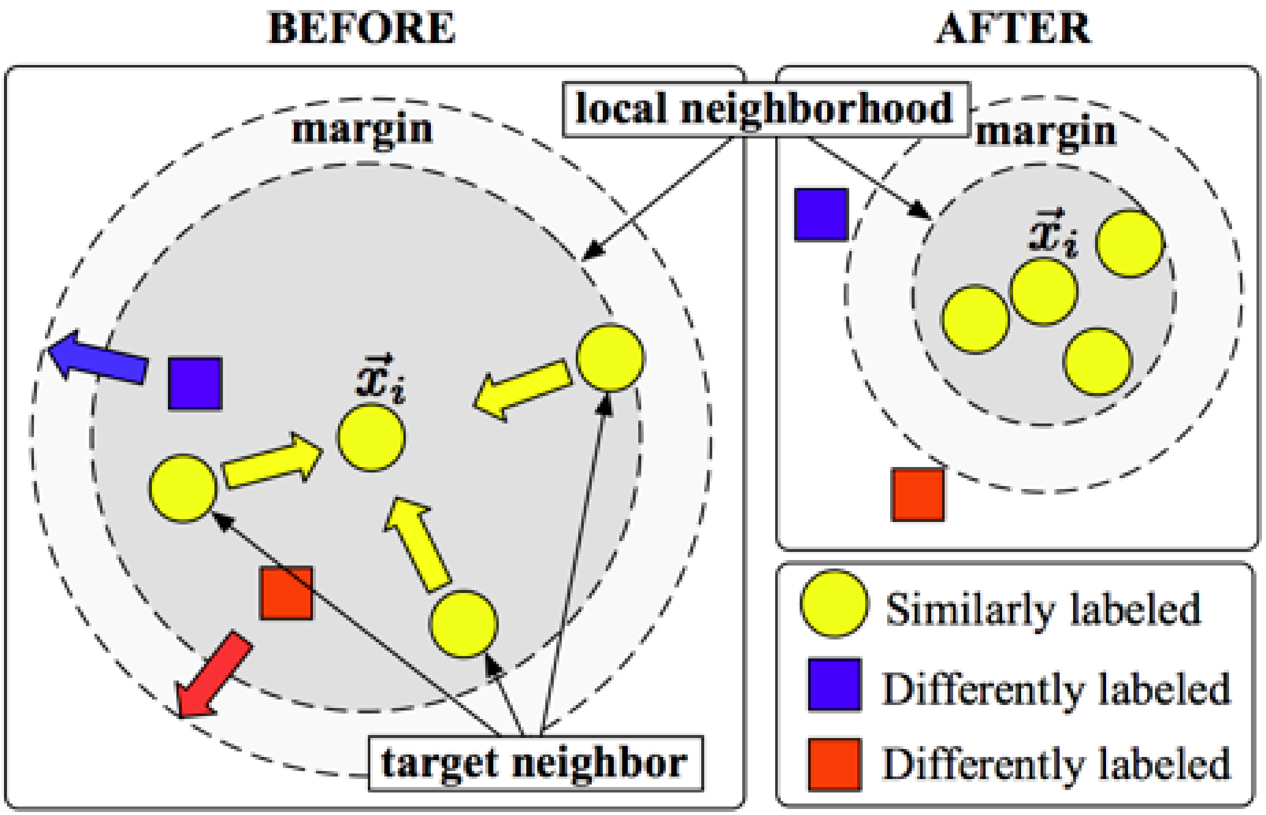
$$T = \{\forall_{x_i}(x_i, x_j) | x_j \text{ is a "target neighbor"}\}$$

$$I = \{\forall_{x_i}(x_i, x_j, x_l) | x_j \text{ is a "target neighbor" } and\ x_l \text{ is an "impostor"}\}$$

- Output:

$$\mathbf{M} \in S_+^{dxd}$$

LMNN (Weinberger et al., 2005)



LMNN (Weinberger et al., 2005)

$$\varepsilon_{\text{pull}}(\mathbf{M}) = \sum_{(x_i, x_j) \in T} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_{\boldsymbol{j}})$$

$$\varepsilon_{\text{push}}(\mathbf{M}) = \sum_{(x_i, x_j, x_l) \in I} [1 + d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_l)]$$

$$\min_{\mathbf{M}} (1 - \mu)\, \varepsilon_{\text{pull}}(\mathbf{M}) + \mu \varepsilon_{\text{push}}(\mathbf{M})$$
$$\text{s.t. } \mathbf{M} \in S_{+}^{d\text{x}d}$$

Algorithm (Works with **L** not **M**):

1. Take objective gradient step w.r.t. **L**
2. Update impostor set
3. Iterate 1-2 until convergence

LMNN (Weinberger et al., 2005)

$$\varepsilon_{\text{pull}}(\mathbf{M}) = \sum_{(x_i,x_j)\in T} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_{\boldsymbol{j}})$$

$$\varepsilon_{\text{push}}(\mathbf{M}) = \sum\nolimits_{(x_i,x_j,x_l)\in I} [1 + d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_l)]$$

$$\min_{\mathbf{M}} (1-\mu)\, \varepsilon_{\text{pull}}(\mathbf{M}) + \mu \varepsilon_{\text{push}}(\mathbf{M})$$
$$\text{s.t. } \mathbf{M} \in S_{+}^{d\text{x}d}$$

Algorithm (Works with **L** not **M**):

1. Take objective gradient step w.r.t. **L**
2. Update impostor set every *p* iterations
3. Iterate 1-2 until convergence

Other Considerations

- Regularization?
  - Frobenius Norm
  - Trace (= trace/nuclear-norm)
- Can we learn $\mathbf{M}$ directly without having to perform expensive projections onto PSD cone?
  - Yes!
    - Information-theoretic Metric Learning (ITML, Davis et al. 2007)
      - Uses Log-Determinant divergence measure as an objective and performs bregman-like projections to satisfy constraints
        - Maintains, low-rank and PSD without explicitly projecting.
  - Kind of!
    - Linear Similarity Learning (Qamar, 2008; Chechik et al., 2009; Bellet et al., 2012; Cheng 2013)
    - Learn a generalized cosine similarity:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{M} \mathbf{x}_j}{N(\mathbf{x}_i, \mathbf{x}_j)}$$

More Recent Topics in Metric Learning

- Non-linear metrics (Chopra, 2005; Salakhutdinov and Hinton, 2007; Xu et al., 2012; Kedem et al., 2012)
- Local Metric Learning (Weinberger and Saul, 2008; Noh et al., 2010; Wang et al., 2012; Xiong et al. 2012)
- Extensions (Parameswaran and Weinberger, 2010; Zhang and Yeung, 2010; McFee and Lankreit 2011)
- Few theoretical guarantees…
- http://arxiv.org/pdf/1306.6709v4.pdf

Kernels

$$k(x_i, x_j) = \langle x_i, x_j \rangle_k = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

$$\mathbf{K} \in \mathbb{R}^{nxn}, \mathbf{K}^{ij} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \mathbf{K} \in S_{+}^{nxn}$$

- Common Kernel Types:
  - Linear: $k(x_i, x_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - d-Degree Polynomial: $k(x_i, x_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d$
  - Gaussian (RBF): $k(x_i, x_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2})$
- Kernel Trick: Easy non-linear transformation
  - Even for Mahalanobis Distance Metrics!

$$k(x_i, x_j) = \exp(-\frac{d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j)}{2\sigma^2})$$

Learning a Kernel Directly

- Can we learn a kernel directly from information that cannot be directly modeled by features?
- Examples:
  - Survey data
  - Feedback through mouse clicks

	Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree
Scale Week is a worthwhile feature on The Research Bunker Blog.	○	○	○	●	○
I would like to read more posts about survey rating scales.	○	○	○	○	●
Vance Marriner is, without a doubt, the most insightful contributor to The Research Bunker Blog.	●	○	○	○	○

- Yes!

GNMDS (Agarwal et al., 2007)

- <u>Main Idea</u>: Given *relative comparisons* between objects, learn a kernel that reflects these comparisons.
  - <u>Relative Comparison</u>: "Object A is more similar to object B than object C is to object D"
- <u>Input</u>:
  $C = \{(a, b, c, d) \mid a \text{ is more similar to } b \text{ than } c \; is \; to \; d\}$
- <u>Output</u>:

$$\mathbf{K} \in S_{+}^{nxn}$$

No information about the objects other than C

GNMDS (Agarwal et al., 2007)

$$\min_{\mathbf{K},\xi_{abcd}} \sum_{(a,b,c,d)\in C} \xi_{abcd} + \lambda \mathrm{Trace}(\mathbf{K})$$

$$\text{s.t. } d_{\mathbf{K}}(x_c, x_d) - d_{\mathbf{K}}(x_a, x_b) \geq 1 - \xi_{abcd}$$

$$\sum_{ab} \mathbf{K}^{ab} = 0\,, \mathbf{K} \in S_{+}^{n\mathrm{x}n}$$

$$d_{\mathbf{K}}(x_a, x_b) = \mathbf{K}^{aa} + \mathbf{K}^{bb} - 2\mathbf{K}^{ab}$$

- By learning **K** we are implicitly learning ϕ
    - Thus, we are implicitly learning an embedding of the objects in a kernel space.

Metric Learning vs. Direct Kernel Learning

- Metric Learning:
  - Learn a generating function **Lx**
    - Can be used on unobserved objects (inductive)
  - Does not guarantee satisfaction of all constraints
- Direct Kernel Learning
  - Learns a kernel **K** over observed objects
    - Cannot be used on unobserved objects (transductive)
  - Guarantees satisfaction of all constraints (McFee and Lanckreit 2011)
    - Given that constraints are consistent

The burning question of kernel methods

- The true goal of machine learning (in many people's opinion)…

Create methods that can be used without ANY domain knowledge or expertise into the method.

- For kernel methods the big hurdle is which kernel function to choose.
  - Linear? Polynomial? Gaussian? Something else?
- Even with a choice of kernel, what is the best parameter setting?

- Motivates **Multiple Kernel Learning (MKL)**

MKL, a brief history

- Choose kernel and parameterization through some criteria
  - Cristianini and Shawe-Taylor, 2000; Scholkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004
- Transductive Setting (Lanckreit et al., 2004)
  - Learn a kernel directly that minimizes a cost function
    - SVM loss
  - Introduced the idea of learning a linear combination of predefined kernels.
- Goal of MKL:
  - Instead of finding the best single kernel, find the best combination of many different predefined kernels.
- Flood of papers afterward:
  - https://sites.google.com/site/xinxingxu666/mklsurvey

GMKL (Varma and Babu, 2009)

- Input:

$$\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_m \in S_+^{nxn}$$
$$y_1, y_2, \ldots, y_n$$

- Main Idea: Create a framework for MKL for different kernel combinations, regularizers, and error functions.
    - Kernel combinations:
        - Sum: $\mathbf{K} = \sum_{i=1}^{m} d_i \mathbf{K}_i$
        - Product: $\mathbf{K} = \prod_{i=1}^{m} d_i \mathbf{K}_i$
        - More complicated combinations
    - Regularizers:
        - l_1: $\|\mathbf{d}\|_1$
        - l_2: $\|\mathbf{d}\|_2$
    - Error Functions:
        - SVM regression and classification

GMKL (Varma and Babu, 2009)

Algorithm:

1. $i \leftarrow 0$
2. $\mathbf{d}^0 \leftarrow random\ initialization$
3. **repeat**
4. $\quad \mathbf{K} \leftarrow k(\mathbf{d}^i)$
5. $\quad$ Use any SVM solver with **K** to find dual variables
6. $\quad$ Update $\mathbf{d}^{i+1}$ with gradient of objective w.r.t $\mathbf{d}^i$
7. $\quad i \leftarrow i + 1$
8. **until** converged

Conclusion

- Finding a good way to compare objects is vital to many machine learning tasks
- This process can be guided by:
  - Side information (constraints)
  - The task to be accomplished
- Models discussed:
  - Metrics
  - Kernels

- Different take on the problem: Representation Learning:
  - http://arxiv.org/pdf/1206.5538.pdf
  - http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial