Multitask Learning, Overview of Some Other Topics, Conclusion and Take-aways

Piyush Rai

Introduction to Machine Learning (CS771A)

November 15, 2018

Announcement

- Final exam: Nov 29, 4pm-8pm (L18, L19, L20)

Announcement

- Final exam: Nov 29, 4pm-8pm (L18, L19, L20)
- Final exam review Session: Nov 21 (Wed). Timing/venue: TBD

Announcement

- Final exam: Nov 29, 4pm-8pm (L18, L19, L20)
- Final exam review Session: Nov 21 (Wed). Timing/venue: TBD
- Project presentations scheduled between Nov 24-27. Fill-in your preferences ASAP

Announcement

- Final exam: Nov 29, 4pm-8pm (L18, L19, L20)
- Final exam review Session: Nov 21 (Wed). Timing/venue: TBD
- Project presentations scheduled between Nov 24-27. Fill-in your preferences ASAP
- Project final report due on Nov 30

Plan for today

- Very quick walk-through (not a review) of what we have seen in this course

Plan for today

- Very quick walk-through (not a review) of what we have seen in this course
- Multitask Learning

Plan for today

- Very quick walk-through (not a review) of what we have seen in this course
- Multitask Learning
- Overview of some other topics
  - Learning Theory
  - Online Learning
  - Learning from time-series data
  - One-shot/few-shot learning
  - Zero-shot learning
  - Bias and Fairness
  - Interpretability of ML models
  - Model Compression

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
    - Discriminative models: probabilistic linear regression, logistic and softmax classification

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
  - Discriminative models: probabilistic linear regression, logistic and softmax classification
  - Generative models: Generative classification

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
  - Discriminative models: probabilistic linear regression, logistic and softmax classification
  - Generative models: Generative classification
  - Connections to non-probabilistic approaches

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
  - Discriminative models: probabilistic linear regression, logistic and softmax classification
  - Generative models: Generative classification
  - Connections to non-probabilistic approaches
- Solving optimization problems arising in machine learning models

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
    - Discriminative models: probabilistic linear regression, logistic and softmax classification
    - Generative models: Generative classification
    - Connections to non-probabilistic approaches
- Solving optimization problems arising in machine learning models
- Hyperplane and large-margin classifiers (Perceptron and SVM)

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
  - Discriminative models: probabilistic linear regression, logistic and softmax classification
  - Generative models: Generative classification
  - Connections to non-probabilistic approaches
- Solving optimization problems arising in machine learning models
- Hyperplane and large-margin classifiers (Perceptron and SVM)
- Kernel methods to turn linear models into nonlinear models

Things we saw..

- Distance based methods (prototype based and nearest neighbors). Simple but powerful
- Learning by asking questions (Decision Trees). Simple but powerful (+fast at test time)
- Learning as optimization (loss function and regularizer) and linear models for regression
- Learning as probabilistic modeling (loss = NLL, reg. = prior, MLE, MAP, fully Bayesian)
- Probabilistic models for supervised learning
  - Discriminative models: probabilistic linear regression, logistic and softmax classification
  - Generative models: Generative classification
  - Connections to non-probabilistic approaches
- Solving optimization problems arising in machine learning models
- Hyperplane and large-margin classifiers (Perceptron and SVM)
- Kernel methods to turn linear models into nonlinear models
- Basic clustering algorithms: K-means and extensions (e.g., soft K-means, kernel K-means)

Things we saw..

- Latent Variable Models for unsupervised and supervised learning

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning
- Recommender Systems via Matrix Factorization/Completion

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning
- Recommender Systems via Matrix Factorization/Completion
- Model Selection, Evaluation Metrics, Learning from Imbalanced Data

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning
- Recommender Systems via Matrix Factorization/Completion
- Model Selection, Evaluation Metrics, Learning from Imbalanced Data
- Reinforcement Learning, Markov Decision Process

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning
- Recommender Systems via Matrix Factorization/Completion
- Model Selection, Evaluation Metrics, Learning from Imbalanced Data
- Reinforcement Learning, Markov Decision Process
- Ensemble Methods (Bagging and Boosting)

Things we saw..

- Latent Variable Models for unsupervised and supervised learning
  - Expectation Maximization (and ALT-OPT) for parameter estimation (MLE/MAP) in LVMs
  - Examples: Gaussian Mixture Model, Probabilistic PCA, Mixture of Experts, Missing Data Problems
- Various dimensionality reduction methods
  - Linear: Classical PCA, SVD; Nonlinear: kernel PCA, LLE, tSNE, etc
  - Other variants: Supervised dim-red, dim-red from pairwise distances (e.g., MDS)
- Deep neural networks for supervised and unsupervised learning
- Recommender Systems via Matrix Factorization/Completion
- Model Selection, Evaluation Metrics, Learning from Imbalanced Data
- Reinforcement Learning, Markov Decision Process
- Ensemble Methods (Bagging and Boosting)
- Bias/Variance Trade-off, Some Practical Issues, Semi-supervised and Active Learning

Machine Learning = Density Estimation ?

To a large extent, YES

Machine Learning = Density Estimation ?

To a large extent, YES

Supervised Learning: Learn $p(y|\boldsymbol{x}, \Theta)$

Machine Learning = Density Estimation ?

To a large extent, YES

Supervised Learning: Learn $p(y|\boldsymbol{x}, \Theta)$

Unsupervised Learning: Learn $p(\boldsymbol{x}|\Theta)$ or $\int p(\boldsymbol{x}, \boldsymbol{z}|\Theta)d\boldsymbol{z}$

Machine Learning = Density Estimation ?

To a large extent, YES

Supervised Learning: Learn $p(y|\boldsymbol{x}, \Theta)$

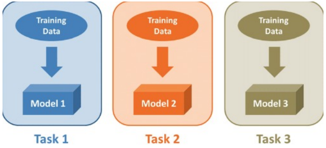
Unsupervised Learning: Learn $p(\boldsymbol{x}|\Theta)$ or $\int p(\boldsymbol{x}, \boldsymbol{z}|\Theta)d\boldsymbol{z}$

That's why the probabilistic viewpoint is important!

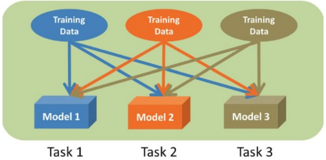
Multitask Learning

Multitask Learning

- In many learning problems, we wish to learn many models, each having its own data



(a)

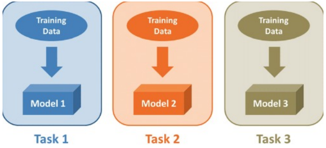


(b)

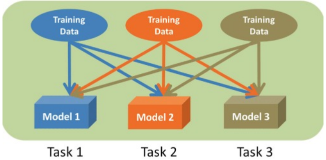
- Example: We wish to learn spam classifiers for M users using each user's training data

Multitask Learning

- In many learning problems, we wish to learn many models, each having its own data



(a)



(b)

- Example: We wish to learn spam classifiers for M users using each user's training data
- Multitask Learning is about designing ways to learn them jointly!

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 \quad + \quad \lambda||\boldsymbol{w}_1||^2$$

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$\begin{aligned} ||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 &\quad + \quad \lambda||\boldsymbol{w}_1||^2 \\ ||\boldsymbol{y}^{(2)} - \mathbf{X}^{(2)}\boldsymbol{w}_2||^2 &\quad + \quad \lambda||\boldsymbol{w}_2||^2 \end{aligned}$$

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$\begin{aligned}
||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 &+ \lambda||\boldsymbol{w}_1||^2 \\
||\boldsymbol{y}^{(2)} - \mathbf{X}^{(2)}\boldsymbol{w}_2||^2 &+ \lambda||\boldsymbol{w}_2||^2 \\
&\vdots \\
||\boldsymbol{y}^{(M)} - \mathbf{X}^{(M)}\boldsymbol{w}_M||^2 &+ \lambda||\boldsymbol{w}_M||^2
\end{aligned}$$

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$\begin{array}{rcl} ||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 & + & \lambda||\boldsymbol{w}_1||^2 \\ ||\boldsymbol{y}^{(2)} - \mathbf{X}^{(2)}\boldsymbol{w}_2||^2 & + & \lambda||\boldsymbol{w}_2||^2 \\ & \vdots & \\ ||\boldsymbol{y}^{(M)} - \mathbf{X}^{(M)}\boldsymbol{w}_M||^2 & + & \lambda||\boldsymbol{w}_M||^2 \end{array}$$

- Let's call each learning problem a "task". Here we are learning each task independently

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$\begin{array}{rcl} ||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 & + & \lambda||\boldsymbol{w}_1||^2 \\ ||\boldsymbol{y}^{(2)} - \mathbf{X}^{(2)}\boldsymbol{w}_2||^2 & + & \lambda||\boldsymbol{w}_2||^2 \\ & \vdots & \\ ||\boldsymbol{y}^{(M)} - \mathbf{X}^{(M)}\boldsymbol{w}_M||^2 & + & \lambda||\boldsymbol{w}_M||^2 \end{array}$$

- Let's call each learning problem a "task". Here we are learning each task independently
- Usually okay to learn independently if we have plenty of training data for each learning task

Multitask Learning: Formally

- Suppose we are given M datasets $(\mathbf{X}^{(1)}, \boldsymbol{y}^{(1)}), (\mathbf{X}^{(2)}, \boldsymbol{y}^{(2)}), \ldots, (\mathbf{X}^{(M)}, \boldsymbol{y}^{(M)})$
- Assume a linear model for each dataset, with weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$
- Naïve way: Learn $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$ by minimizing individual loss functions for each dataset

$$
\begin{array}{rcl}
||\boldsymbol{y}^{(1)} - \mathbf{X}^{(1)}\boldsymbol{w}_1||^2 & + & \lambda||\boldsymbol{w}_1||^2 \\
||\boldsymbol{y}^{(2)} - \mathbf{X}^{(2)}\boldsymbol{w}_2||^2 & + & \lambda||\boldsymbol{w}_2||^2 \\
 & \vdots & \\
||\boldsymbol{y}^{(M)} - \mathbf{X}^{(M)}\boldsymbol{w}_M||^2 & + & \lambda||\boldsymbol{w}_M||^2
\end{array}
$$

- Let's call each learning problem a "task". Here we are learning each task independently
- Usually okay to learn independently if we have plenty of training data for each learning task
- If training data per-task is very little and if tasks are related, it may not be the most ideal approach

Multitask Learning

- A better alternative will be to learn all the tasks jointly by minimizing the following loss function

$$\sum_{m=1}^{M} ||\boldsymbol{y}^{(m)} - \mathbf{X}^{(m)}\boldsymbol{w}_m||^2 + R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M)$$

Multitask Learning

- A better alternative will be to learn all the tasks jointly by minimizing the following loss function

$$\sum_{m=1}^{M} ||\boldsymbol{y}^{(m)} - \mathbf{X}^{(m)}\boldsymbol{w}_m||^2 + R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M)$$

- $R(.)$ is a regularizer that encourages these weight vectors to be close to each other

Multitask Learning

- A better alternative will be to learn all the tasks jointly by minimizing the following loss function

$$\sum_{m=1}^{M} ||\boldsymbol{y}^{(m)} - \mathbf{X}^{(m)}\boldsymbol{w}_m||^2 + R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M)$$

- $R(.)$ is a regularizer that encourages these weight vectors to be close to each other
- Example 1: Assume all weight vectors to be close to some "global" weight vector $\boldsymbol{\mu}_0$

$$R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} ||\boldsymbol{w}_m - \boldsymbol{\mu}_0||^2$$

Multitask Learning

- A better alternative will be to learn all the tasks jointly by minimizing the following loss function

$$\sum_{m=1}^{M} ||\boldsymbol{y}^{(m)} - \mathbf{X}^{(m)}\boldsymbol{w}_m||^2 + R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M)$$

- $R(.)$ is a regularizer that encourages these weight vectors to be close to each other
- Example 1: Assume all weight vectors to be close to some "global" weight vector $\boldsymbol{\mu}_0$

$$R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} ||\boldsymbol{w}_m - \boldsymbol{\mu}_0||^2$$

- Example 2: Assume K groups with means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and each $\boldsymbol{w}_m$ to belong to one of the groups

$$R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} ||\boldsymbol{w}_m - \boldsymbol{\mu}_{z_m}||^2$$

Multitask Learning

- Example 3: Assume we have an $M \times M$ task similarity graph G ($G_{mm'}$ large if tasks highly related)

$$R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} \sum_{m' \neq m} G_{mm'} ||\boldsymbol{w}_m - \boldsymbol{w}_{m'}||^2$$

Multitask Learning

- Example 3: Assume we have an $M \times M$ task similarity graph G ($G_{mm'}$ large if tasks highly related)

$$R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} \sum_{m' \neq m} G_{mm'} ||\boldsymbol{w}_m - \boldsymbol{w}_{m'}||^2$$

- Example 4: Assume each $\boldsymbol{w}_m$ to be a linear combination of K shared "basis" weight vectors

$$\boldsymbol{w}_m = \sum_{k=1}^{K} z_{mk} \boldsymbol{\mu}_k$$

.. or an alternative $R(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M) = \sum_{m=1}^{M} ||\boldsymbol{w}_m - \sum_{k=1}^{K} z_{nk} \boldsymbol{\mu}_k||^2$

Multitask Learning

- Example 3: Assume we have an $M \times M$ task similarity graph G ($G_{mm'}$ large if tasks highly related)

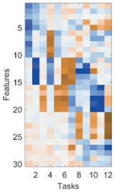
$$R(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M) = \sum_{m=1}^{M} \sum_{m' \neq m} G_{mm'} ||\mathbf{w}_m - \mathbf{w}_{m'}||^2$$

- Example 4: Assume each $\mathbf{w}_m$ to be a linear combination of K shared "basis" weight vectors

$$\mathbf{w}_m = \sum_{k=1}^{K} z_{mk} \boldsymbol{\mu}_k$$

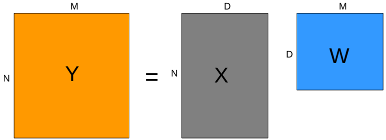
.. or an alternative $R(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M) = \sum_{m=1}^{M} ||\mathbf{w}_m - \sum_{k=1}^{K} z_{nk} \boldsymbol{\mu}_k||^2$

- Example 5: Assume all weight vectors to have same/similar sparsity pattern (relevant features)



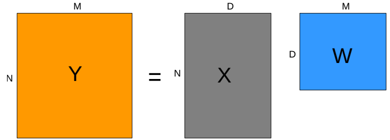
Multitask Learning vs Multi-output/Multilabel Learning

- Multi-output and multi-label learning problems can also be thought of as multitask learning
- Same inputs and multiple outputs/labels to be predicted
- Here too, we need to learn a weight vector for each output/label
- The stadard approach is to simply model these as $\mathbf{Y} \approx \mathbf{XW}$ and solve for $\mathbf{W}$



Multitask Learning vs Multi-output/Multilabel Learning

- Multi-output and multi-label learning problems can also be thought of as multitask learning
- Same inputs and multiple outputs/labels to be predicted
- Here too, we need to learn a weight vector for each output/label
- The stadard approach is to simply model these as $\mathbf{Y} \approx \mathbf{X}\mathbf{W}$ and solve for $\mathbf{W}$

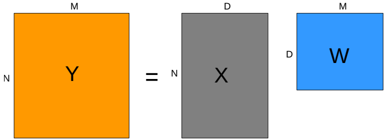


- We have seen that $\mathbf{W}$ has closed form solution if $\mathbf{Y}$ is real-valued

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y} \qquad \text{(same as } \boldsymbol{w}_m = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}_m, \text{ for each m)}$$

Multitask Learning vs Multi-output/Multilabel Learning

- Multi-output and multi-label learning problems can also be thought of as multitask learning
- Same inputs and multiple outputs/labels to be predicted
- Here too, we need to learn a weight vector for each output/label
- The stadard approach is to simply model these as $\mathbf{Y} \approx \mathbf{XW}$ and solve for $\mathbf{W}$



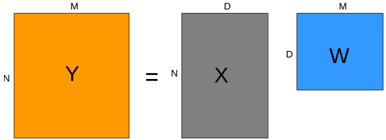
- We have seen that $\mathbf{W}$ has closed form solution if $\mathbf{Y}$ is real-valued

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y} \qquad \text{(same as } \boldsymbol{w}_m = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}_m, \text{ for each m)}$$

- However, the above apprpach is equivalent to treating each outputs/labels independently

Multitask Learning vs Multi-output/Multilabel Learning

- Multi-output and multi-label learning problems can also be thought of as multitask learning
- Same inputs and multiple outputs/labels to be predicted
- Here too, we need to learn a weight vector for each output/label
- The stadard approach is to simply model these as $\mathbf{Y} \approx \mathbf{XW}$ and solve for $\mathbf{W}$



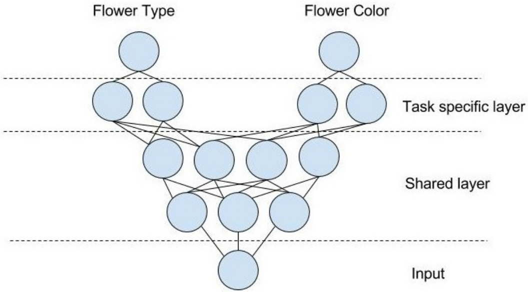
- We have seen that $\mathbf{W}$ has closed form solution if $\mathbf{Y}$ is real-valued

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y} \qquad \text{(same as } \boldsymbol{w}_m = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \boldsymbol{y}_m, \text{ for each m)}$$

- However, the above apprpach is equivalent to treating each outputs/labels independently
- The ideas we saw today can be used to improve multi-output/multi-label learning

Multi-output/Multilabel Learning using Deep Neural Networks

- Deep neural networks are also popular these days for solving multi-output learning problems
- Basic idea: Have shared hidden layers to learn features that are good for predicting each output



- Such neural networks are called multitask neural networks

Multitask Learning: Some Comments

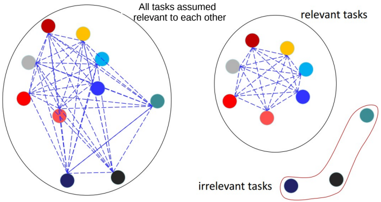
- Very useful and widely used in many applications
- In some contexts, also referred to as “Transfer Learning”

Multitask Learning: Some Comments

- Very useful and widely used in many applications
- In some contexts, also referred to as "Transfer Learning"
  - Note: Usually TL refers to the setting when we learn some task leveraging knowledge acquired from previous tasks whereas Multitask Learning typically assumes all tasks are being learned simultaneously

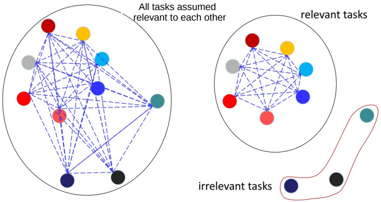
Multitask Learning: Some Comments

- Very useful and widely used in many applications
- In some contexts, also referred to as "Transfer Learning"
  - Note: Usually TL refers to the setting when we learn some task leveraging knowledge acquired from previous tasks whereas Multitask Learning typically assumes all tasks are being learned simultaneously
- Inappropriate sharing assumption can also hurt performance (e.g., assuming all weight vectors to be related with each other may not be correct if not all tasks are related)



Multitask Learning: Some Comments

- Very useful and widely used in many applications
- In some contexts, also referred to as "Transfer Learning"
  - Note: Usually TL refers to the setting when we learn some task leveraging knowledge acquired from previous tasks whereas Multitask Learning typically assumes all tasks are being learned simultaneously
- Inappropriate sharing assumption can also hurt performance (e.g., assuming all weight vectors to be related with each other may not be correct if not all tasks are related)



- Automatically learning how the tasks are related can help. There has been work on this too

Overview of Some Other Topics

Learning Theory

- Study of theoretical properties of learning models/algorithms, e.g.,
  - What is the generalization error (difference of test and training error) of some model?
  - What is the minimum number of training examples needed to get a certain accuracy?
  - What is learnable, what is not
- Some typical results from learning theory might look like this..

"size" or "complexity" of the class h belongs to

Test error of h   Training error of h

$$L_P(h) \leq L_{\mathcal{D}}(h) + \sqrt{\frac{\log |\mathcal{H}| + \log \frac{1}{\delta}}{2N}}$$

Number of training examples required to <= epsilon error

$$N \geq \frac{1}{2\epsilon^2}(\log |\mathcal{H}| + log \frac{1}{\delta})$$

Learning Theory

- Study of theoretical properties of learning models/algorithms, e.g.,
  - What is the generalization error (difference of test and training error) of some model?
  - What is the minimum number of training examples needed to get a certain accuracy?
  - What is learnable, what is not
- Some typical results from learning theory might look like this..

$$\underbrace{L_P(h)}_{\text{Test error of h}} \leq \underbrace{L_{\mathcal{D}}(h)}_{\text{Training error of h}} + \sqrt{\frac{\log \overbrace{|\mathcal{H}|}^{\text{"size" or "complexity" of the class h belongs to}} + \log \frac{1}{\delta}}{2N}}$$

$$\underbrace{N}_{\text{Number of training examples required to <= epsilon error}} \geq \frac{1}{2\epsilon^2}(\log|\mathcal{H}| + log\frac{1}{\delta})$$

- The field is too deep than what the above two equations convey :-)

Online Learning

- Standard ML: Learn a model on some training data, apply it on a test data
- In many problem, there is no distinction b/w training and test data (everything is test data)

Online Learning

- Standard ML: Learn a model on some training data, apply it on a test data
- In many problem, there is no distinction b/w training and test data (everything is test data)
- Learner receives one input at a time, and predicts the output (no training phase)

Online Learning

- Standard ML: Learn a model on some training data, apply it on a test data
- In many problem, there is no distinction b/w training and test data (everything is test data)
- Learner receives one input at a time, and predicts the output (no training phase)
- Online Learning is precisely this setting!

Online Learning

- Standard ML: Learn a model on some training data, apply it on a test data
- In many problem, there is no distinction b/w training and test data (everything is test data)
- Learner receives one input at a time, and predicts the output (no training phase)
- Online Learning is precisely this setting!
- Online Learning algos not evaluated by generalization error (diff. b/w test and training error)

$$R_T = \underbrace{\sum_{t=1}^{T} L(\widehat{y}_t, y_t)}_{\text{Total error of the learner}} - \underbrace{\min_{i=1}^{N} \sum_{t=1}^{T} L(\widehat{y}_{t,i}, y_t)}_{\text{Total error of the best experts}}$$

- Evaluated in terms of how bad they are as compared to the best expert at each step in hindsight

Online Learning

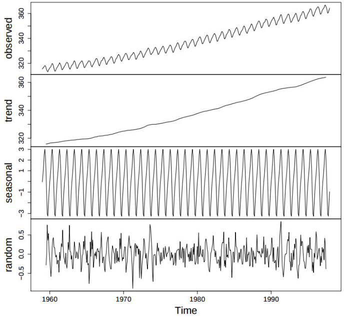
- Standard ML: Learn a model on some training data, apply it on a test data
- In many problem, there is no distinction b/w training and test data (everything is test data)
- Learner receives one input at a time, and predicts the output (no training phase)
- Online Learning is precisely this setting!
- Online Learning algos not evaluated by generalization error (diff. b/w test and training error)

$$R_T = \underbrace{\sum_{t=1}^{T} L(\widehat{y}_t, y_t)}_{\text{Total error of the learner}} - \underbrace{\min_{i=1}^{N} \sum_{t=1}^{T} L(\widehat{y}_{t,i}, y_t)}_{\text{Total error of the best experts}}$$

- Evaluated in terms of how bad they are as compared to the best expert at each step in hindsight
- This difference is known as "Regret" of the online learner

Modeling Time-Series Data

- The input is a sequence of (non-i.i.d.) examples $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T$
- The problem may be supervised or unsupervised, e.g.,
  - Forecasting: Predict $\boldsymbol{y}_{T+1}$, given $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T$
  - Cluster the examples or perform dimensionality reduction
- Evolution of time-series data can be attributed to several factors

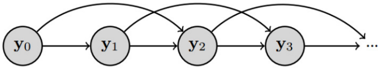


- Teasing apart these factors of variation is also an important problem

Modeling Time-Series Data (Contd)

- Auto-regressive (AR): Regress each example on p previous examples

$$\boldsymbol{y}_t = c + \sum_{i=1}^{p} w_i \boldsymbol{y}_{t-i} + \epsilon_t \qquad : \text{ An AR}(p) \text{ model}$$

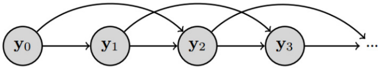


Auto-regressive Model (shown above: 2nd order AR)

Modeling Time-Series Data (Contd)

- Auto-regressive (AR): Regress each example on p previous examples

$$\boldsymbol{y}_t = c + \sum_{i=1}^{p} w_i \boldsymbol{y}_{t-i} + \epsilon_t \quad : \text{ An AR}(p) \text{ model}$$



Auto-regressive Model (shown above: 2nd order AR)

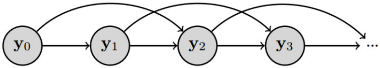
- Moving Average (MA): Regress each example on p previous stochastic errors

$$\boldsymbol{y}_t = c + \epsilon_t + \sum_{i=1}^{p} w_i \epsilon_{t-i} \quad : \text{ An MA}(p) \text{ model}$$

Modeling Time-Series Data (Contd)

- Auto-regressive (AR): Regress each example on p previous examples

$$\boldsymbol{y}_t = c + \sum_{i=1}^{p} w_i \boldsymbol{y}_{t-i} + \epsilon_t \qquad : \text{ An AR}(p) \text{ model}$$



Auto-regressive Model (shown above: 2nd order AR)

- Moving Average (MA): Regress each example on p previous stochastic errors

$$\boldsymbol{y}_t = c + \epsilon_t + \sum_{i=1}^{p} w_i \epsilon_{t-i} \qquad : \text{ An MA}(p) \text{ model}$$

- Auto-regressive Moving Average (ARMA): Regress each example of p previous examples and q previous stochastic errors

$$\boldsymbol{y}_t = c + \epsilon_t + \sum_{i=1}^{p} w_i \boldsymbol{y}_{t-i} + \sum_{i=1}^{q} v_i \epsilon_{t-i} \qquad : \text{ An ARMA}(p,q) \text{ model}$$

One-Shot and Few-Shot Learning

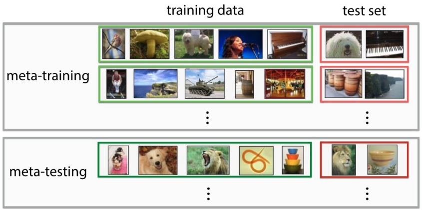
- Humans can learn a concept from as few as one example!
- Example: Can learn to recognize a person even if we have seen then once

One-Shot and Few-Shot Learning

- Humans can learn a concept from as few as one example!
- Example: Can learn to recognize a person even if we have seen then once
- Can ML algorithms be designed to do the same?

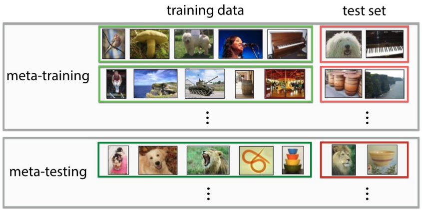
One-Shot and Few-Shot Learning

- Humans can learn a concept from as few as one example!
- Example: Can learn to recognize a person even if we have seen then once
- Can ML algorithms be designed to do the same?
- One-Shot and Few-Shot Learning research tries to address this question



One-Shot and Few-Shot Learning

- Humans can learn a concept from as few as one example!
- Example: Can learn to recognize a person even if we have seen then once
- Can ML algorithms be designed to do the same?
- One-Shot and Few-Shot Learning research tries to address this question



- The basic idea is to training in the same way we are expected to be tested (i.e., training using one example at a time, test, measure error, and repeat to improve)

Zero-Shot Learning

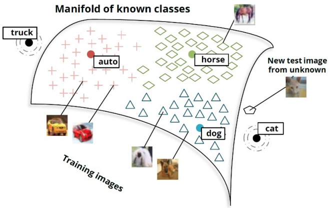
- We have already seen this in the very first homework (programming problem). :-)
- Test data may have examples from classes that were not present at training time
- However, often we have some description of each class (e.g., a class-attribute vector)

Zero-Shot Learning

- We have already seen this in the very first homework (programming problem). :-)
- Test data may have examples from classes that were not present at training time
- However, often we have some description of each class (e.g., a class-attribute vector)
- Can use these class-attribute vectors to extrapolate to the new classes

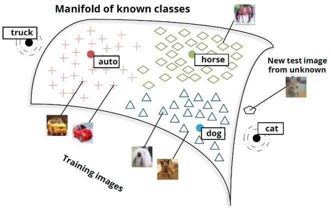
Zero-Shot Learning

- We have already seen this in the very first homework (programming problem). :-)
- Test data may have examples from classes that were not present at training time
- However, often we have some description of each class (e.g., a class-attribute vector)
- Can use these class-attribute vectors to extrapolate to the new classes, e.g.,
    - Can map each test example to the attribute vector space and find the most similar class



Zero-Shot Learning

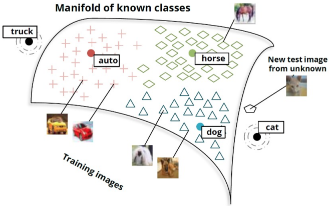
- We have already seen this in the very first homework (programming problem). :-)
- Test data may have examples from classes that were not present at training time
- However, often we have some description of each class (e.g., a class-attribute vector)
- Can use these class-attribute vectors to extrapolate to the new classes, e.g.,
  - Can map each test example to the attribute vector space and find the most similar class



  - Represent each new class as a similarity-based combination of previously seen classes

Zero-Shot Learning

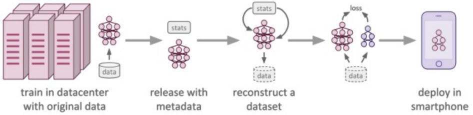
- We have already seen this in the very first homework (programming problem). :-)
- Test data may have examples from classes that were not present at training time
- However, often we have some description of each class (e.g., a class-attribute vector)
- Can use these class-attribute vectors to extrapolate to the new classes, e.g.,
  - Can map each test example to the attribute vector space and find the most similar class



  - Represent each new class as a similarity-based combination of previously seen classes
  - Can learn a mapping from attribute vector to the parameters of the distribution of each class

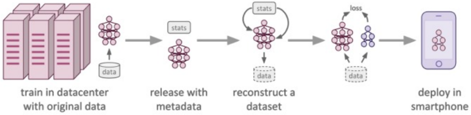
Some Emerging Research Directions in ML

- Model Compression: How to compress and store big models on tiny devices?



Some Emerging Research Directions in ML

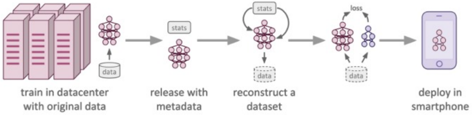
- Model Compression: How to compress and store big models on tiny devices?



- Interpretable and Explainable ML: Can we explain why an ML algo predicts what it predicts?

Some Emerging Research Directions in ML

- Model Compression: How to compress and store big models on tiny devices?

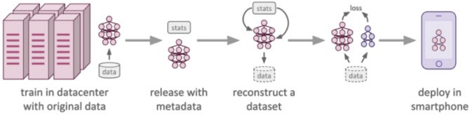


- Interpretable and Explainable ML: Can we explain why an ML algo predicts what it predicts?

- Fairness and Bias in ML

Some Emerging Research Directions in ML

- Model Compression: How to compress and store big models on tiny devices?



- Interpretable and Explainable ML: Can we explain why an ML algo predicts what it predicts?

- Fairness and Bias in ML
- Security and privacy issues

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well
- Think carefully about your features, how you compute similarities, etc.

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well
- Think carefully about your features, how you compute similarities, etc.
- Linear models can be really powerful given a good feature representation/similarities

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well
- Think carefully about your features, how you compute similarities, etc.
- Linear models can be really powerful given a good feature representation/similarities
- Latent variable models are very useful in many problems (and so are algos like EM/ALT-OPT)

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well
- Think carefully about your features, how you compute similarities, etc.
- Linear models can be really powerful given a good feature representation/similarities
- Latent variable models are very useful in many problems (and so are algos like EM/ALT-OPT)
- Helps to learn to first diagnose a learning algorithm rather than trying new ones

Conclusion and Take-aways

- Most learning problems can be cast as optimizing a regularized loss function
- Probabilistic and optimization viewpoints are complementary/equivalent
  - Negative log-likelihood (NLL) = loss function, log-prior = regularizer
- More sophisticated models can be constructed with this basic understanding: Just think of the appropriate loss function/probability model for the data, and the appropriate regularizer/prior
- Always start with simple models that you understand well
- Think carefully about your features, how you compute similarities, etc.
- Linear models can be really powerful given a good feature representation/similarities
- Latent variable models are very useful in many problems (and so are algos like EM/ALT-OPT)
- Helps to learn to first diagnose a learning algorithm rather than trying new ones
- No free lunch. No learning algorithm is "universally" good.

Thank You! Have Fun Learning!