The Story So Far (Mid-Sem Review)

Piyush Rai

Machine Learning (CS771A)

Sept 9, 2016

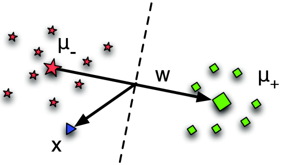
Topics Seen So Far

- **Supervised Learning**
  - Distance based methods
    - Distance from Means
    - Nearest Neighbor methods
  - Learning by asking questions. Decision Trees
  - Learning as Optimization (Loss + Regularizer)
  - Learning via Probabilistic Modeling (Likelihood + Prior)
  - Online Learning via Stochastic Optimization. Perceptron
  - Learning Maximum-Margin Hyperplanes (SVM)
- **Unsupervised Learning**
  - Clustering: K-means
  - Dimensionality Reduction: PCA
- **Nonlinear Supervised and Unsupervised Learning via Kernels**

Supervised Learning

"Distance from Means" for Classification

- Predicts the class of a point based on its closeness to the class means

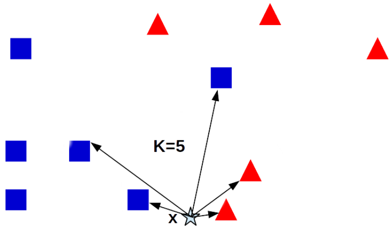


$$\begin{aligned} f(\boldsymbol{x}) &= ||\boldsymbol{\mu}_- - \boldsymbol{x}||^2 - ||\boldsymbol{\mu}_+ - \boldsymbol{x}||^2 \\ &= 2\langle \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \boldsymbol{x}\rangle + ||\boldsymbol{\mu}_-||^2 - ||\boldsymbol{\mu}_+||^2 \end{aligned}$$

- Predicts positive class if $f(x) > 0$, negative class otherwise
- $f(\boldsymbol{x})$ effectively denotes a hyperplane based classification rule (with the vector $\boldsymbol{w} = \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-$ representing the direction normal to the hyperplane)
- Easily extends to multiclass classification
- Other extensions: Replace the means by "distributions", use kernels

Nearest Neighbors Methods

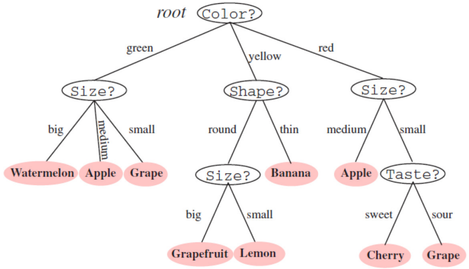
- Can be used for both regression and classification (label can be real/discrete)
- Predict the label by looking at labels of most similar training examples



- Choice of K is important
- The distance function can be crucial (also for distance from means)

Decision Trees

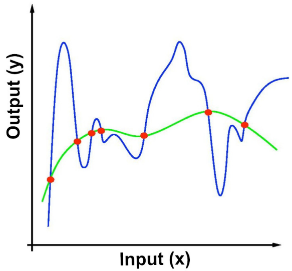
- Can be used for both regression and classification (label can be real/discrete)
- Predict the label by sequentially checking values of a set of features
- Need some criterion to decide the order in which features are tested



- Complexity is determined by the depth of tree or number of internal nodes
- Often we want to prune the tree for better generalization

Overfitting and Generalization

- Care more about performance on test data rather than on training data
- A very complex model can do very well on training data but will not necessarily generalize well on test data



Learning as Optimization

- Learning can be seen as a form of function approximation
- In supervised learning, we want to learn f s.t. $y_n \approx f(\boldsymbol{x}_n), \forall n$
- Can do so by learning f by Empirical Risk Minimization (ERM)

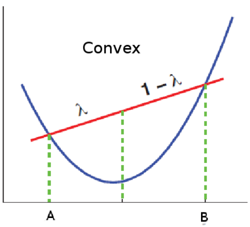
$$\hat{f} = \arg\min_f L_{emp}(f) = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

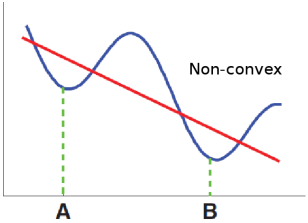
- Loss function $\ell()$ measures how well f predicts the true output
- We also want f to be "simple". To do so, we add a "regularizer" $R(f)$

$$\boxed{\hat{f} = \arg\min_f \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n)) + \lambda R(f)}$$

- The regularizer $R(f)$ is a measure of complexity of our model f
- This is called **Regularized** (Empirical) Risk Minimization

Convex Functions



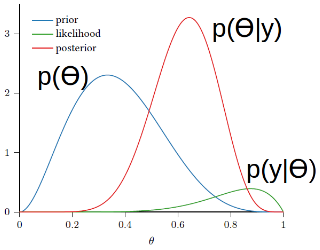


- Every local minima is also a global minima
- Thus it's good to have loss functions and regularizers that are convex
- Note: There are other subtle technicalities (e.g., convex vs strictly convex)

Learning via Probabilistic Modeling

- Model the data distribution using a likelihood function $p(\boldsymbol{y}|\theta)$
- Assume a prior distribution on the model parameters $p(\theta)$
- The posterior distribution over θ is defined using Bayes Rule as follows

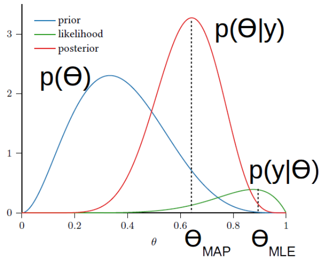
$$p(\theta|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\theta)p(\theta)}{p(\boldsymbol{y})}$$



Learning via Probabilistic Modeling

- Model the data distribution using a likelihood function $p(\boldsymbol{y}|\theta)$
- Assume a prior distribution on the model parameters $p(\theta)$
- The posterior distribution over θ is defined using Bayes Rule as follows

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



- Can either find the θ that maximizes the likelihood (MLE) or maximizes the posterior distribution (MAP estimation)

Maximum Likelihood Estimation (MLE)

- Maximum Likelihood parameter estimation

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \sum_{n=1}^{N} \log p(y_n \mid \theta)$$

- We can also think of it as minimizing the **negative** log-likelihood (NLL)

$$\boxed{\hat{\theta}_{MLE} = \arg\min_{\theta} NLL(\theta)}$$

  where $NLL(\theta) = -\sum_{n=1}^{N} \log p(y_n \mid \theta)$

- We can think of the negative log-likelihood as a loss function
- Thus MLE is equivalent to doing empirical risk (loss) minimization
- This view relates the optimization and probabilistic modeling approaches

Maximum-a-Posteriori (MAP) Estimation

- Same as MLE with an extra log-prior-distribution term (acts as a regularizer)

$$\boxed{\hat{\theta}_{MAP} = \arg\max_{\theta} \sum_{n=1}^{N} \log p(y_n|\theta) + \log p(\theta)}$$

- Can also write the same as the following (equivalent) minimization problem

$$\boxed{\hat{\theta}_{MAP} = \arg\min_{\theta} NLL(\theta) - \log p(\theta)}$$

- When $p(\theta)$ is a uniform prior, MAP reduces to MLE

Example: Linear and Ridge Regression

- Assumes a linear model for input-output relationship, i.e., $y_n \approx \boldsymbol{w}^\top \boldsymbol{x}_n$
- The following loss function is minimized in ridge regression

$$L(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2 + \lambda ||\boldsymbol{w}||^2$$

- The closed form solution is now

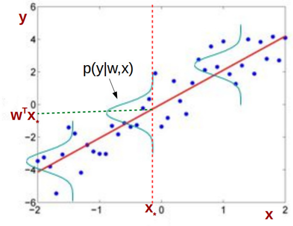
$$\boldsymbol{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

- For $\lambda = 0$, reduces to simple linear regression
- Regularizer promotes simple solutions (good generalization). Also has many other benefits (e.g., faster convergence of iterative methods such as gradient descent, helps find a unique global minima)

Example: Probabilistic Linear/Ridge Regression

- Assuming Gaussian distributed responses y_n, we have

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_n, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2}{2\sigma^2}\right\}$$



- Doing MLE on this model gives the same solution for $\boldsymbol{w}$ as linear regression
- Can also put a zero mean Gaussian prior on $\boldsymbol{w}$, i.e., $p(\boldsymbol{w}) \propto \exp\left\{-\frac{\boldsymbol{w}^\top \boldsymbol{w}}{2\rho^2}\right\}$, which then makes the model equivalent to ridge regression

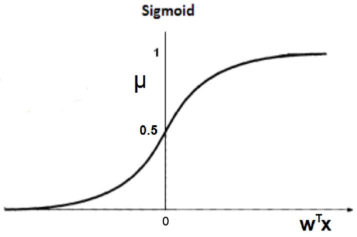
Example: Logistic Regression

- A linear model for binary classification regression
- Also a probabilistic model for binary classification. Predicts *label probabilities* rather than a hard binary label 0/1.

$$\begin{aligned} p(y_n = 1 | \boldsymbol{x}_n, \boldsymbol{w}) &= \mu_n \\ p(y_n = 0 | \boldsymbol{x}_n, \boldsymbol{w}) &= 1 - \mu_n \end{aligned}$$

- Models label probabilities using a sigmoid function, i.e.,

$$\mu_n = \sigma(\mathbf{w}^\top \mathbf{x}_n) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} = \frac{\exp(\mathbf{w}^\top \mathbf{x}_n)}{1 + \exp(\mathbf{w}^\top \mathbf{x}_n)} \qquad \text{(prob. of label = 1)}$$



Parameter Estimation for Logistic Regression

- Can define a loss function (cross entropy) or a direct likelihood model (each label modeled by a Bernoulli distribution with probability μ_n)
- Both approaches results in the same objective function
- For the unregularized case, it will be

$$\boxed{L(\boldsymbol{w}) = -\sum_{n=1}^{N}(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n)))}$$

- For the regularized case (ℓ_2 reg. on $\boldsymbol{w}$ or Gaussian prior on $\boldsymbol{w}$), it will be

$$\boxed{L(\boldsymbol{w}) = -\sum_{n=1}^{N}(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n))) + \lambda||\boldsymbol{w}||^2}$$

- Can't get a closed form solution. Must use some iterative method.

Gradient-based Methods

- When closed form solution can't be obtained (and also for efficiency reasons even it *can* be), we use iterative (e.g., gradient or second order) methods
- Gradient descent in its general form is as follows
    - Start with an initial (maybe randomly initialized) value of $\boldsymbol{w} = \boldsymbol{w}^{(0)}$
    - Update $\boldsymbol{w}$ by moving along the gradient of the loss function L (L_{emp} or L_{reg})

$$\boldsymbol{w}^{(t)} = \boldsymbol{w}^{(t-1)} - \eta \frac{\partial L}{\partial \boldsymbol{w}}\bigg|_{\boldsymbol{w}=\boldsymbol{w}^{(t-1)}}$$

    where η is the learning rate

    - Repeat until converge
- Can also use stochastic iterative methods (e.g., stochastic gradient descent) that make use of one example (or a small number of examples) at a time
- Iterative methods converge to a local optima (unless the function is convex)

Gradient Expressions

- In the case of logistic regression, the gradient is

$$\begin{aligned}
\mathbf{g} = \frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\partial}{\partial \boldsymbol{w}}[-\sum_{n=1}^{N}(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_n)))] \\
&= -\sum_{n=1}^{N}\left(y_n \boldsymbol{x}_n - \frac{\exp(\boldsymbol{w}^\top \boldsymbol{x}_n)}{(1+\exp(\boldsymbol{w}^\top \boldsymbol{x}_n))}\boldsymbol{x}_n\right) \\
&= -\sum_{n=1}^{N}(y_n - \mu_n)\boldsymbol{x}_n = \mathbf{X}^\top(\boldsymbol{\mu} - \boldsymbol{y})
\end{aligned}$$

- Note: Other linear models (e.g., linear regression and generalized linear models) also have the same form for the gradient's expression

$$\mathbf{g} = -\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n))\boldsymbol{x}_n$$

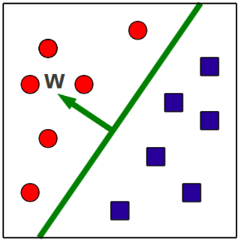
  where $f(\boldsymbol{x}_n)$ denotes the model's prediction

- Note: When doing stochastic gradient descent, the gradient is approximated using a single example or a small number of examples, e.g., for log. reg.

$$\mathbf{g}_n = -(y_n - \mu_n)\boldsymbol{x}_n \qquad \text{(if using a single example)}$$

Hyperplane based Methods for Classification

- Goal: To learn a hyperplane $(\boldsymbol{w}, b)$ that separates the classes



- Decision rule (assuming binary classification with labels = +1/-1)

$$y = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

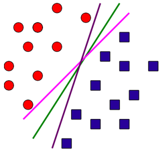
- Saw examples of Perceptron and Support Vector Machines (SVM)
- Many other methods can be seen as learning hyperplanes (in fact any linear model or a generalized linear model, e.g., logistic regression does the same)

Perceptron

- A simple mistake-driven model for learning a hyperplane separator
- If current $(\boldsymbol{w}, b)$ make a mistake on training example $(\boldsymbol{x}_n, y_n)$ then do

$$\boldsymbol{w} = \boldsymbol{w} + y_n\boldsymbol{x}_n \quad \text{and} \quad b = b + y_n$$

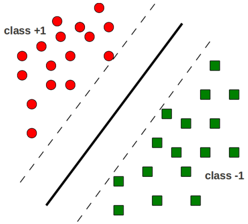
- Converges after a finite number of iterations if data is separable
- Finds one of the possibly many hyperplanes that could separate the classes



- Can be seen as minimizing a loss function $\ell_n(\boldsymbol{w}, b) = \max\{0, -y_n\boldsymbol{w}^\top\boldsymbol{x}_n\}$ w.r.t. the hyperplane parameters $\boldsymbol{w}, b$
- Can also be seen as doing stochastic optimization of logistic regression model with label probabilities replaced by hard labels

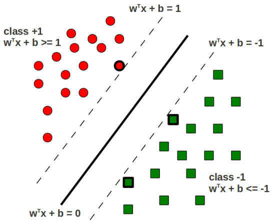
Maximum Margin Classification

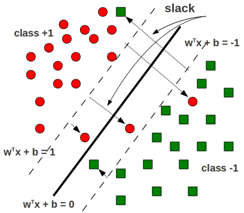
- Hyperplanes with large margins are nice: Generalize well on test data



- For hyperplane based models, margin $\propto 1/||\boldsymbol{w}||$
- SVM incorporates this idea in a proper, principled way
- SVM learns a hyperplane that has the largest possible margin while also keeping points as far away as possible from the hyperplane

Hard-Margin SVM and Soft-Margin SVM





- Objective for the hard-margin SVM (unknowns are $\boldsymbol{w}$ and b)

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w},b) = \frac{||\boldsymbol{w}||^2}{2}$$

$$\text{subject to constraints} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1, \qquad n = 1,\ldots,N$$

- Objective for the soft-margin SVM (unknowns are $\boldsymbol{w}$, b, and $\{\xi_n\}_{n=1}^N$)

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad f(\boldsymbol{w},b,\boldsymbol{\xi}) = \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N}\xi_n$$

$$\text{subject to} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \qquad \xi_n \geq 0 \qquad n = 1,\ldots,N$$

- In either case, we have to solve constrained, convex optimization problem

Hard-Margin SVM and Soft-Margin SVM

- The dual formulation of these problems

$$\boxed{\text{Hard-Margin SVM:} \quad \max_{\boldsymbol{\alpha} \geq 0} \; \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha} \quad \text{s.t.} \quad \sum_{n=1}^{N} \alpha_n y_n = 0}$$

$$\boxed{\text{Soft-Margin SVM:} \quad \max_{\boldsymbol{\alpha} \leq C} \; \mathcal{L}_D(\alpha) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha} \quad \text{s.t.} \quad \sum_{n=1}^{N} \alpha_n y_n = 0}$$

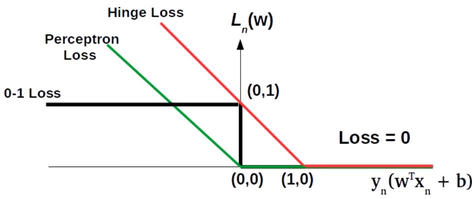
- We can learn $\boldsymbol{\alpha}$ using various methods (e.g., by solving a QP or using gradient based methods)

Loss Function Minimization View of SVM

- Can think of SVM loss as basically the sum of the slacks $\xi_n \geq 0$, which is

$$\boxed{\ell(\boldsymbol{w}, b) = \sum_{n=1}^{N} \ell_n(\boldsymbol{w}, b) = \sum_{n=1}^{N} \xi_n = \sum_{n=1}^{N} \max\{0, 1 - y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)\}}$$

- This is called **"Hinge Loss"**. Can also learn SVMs by minimizing this loss via stochastic sub-gradient descent (can also add a regularizer on $\boldsymbol{w}$, e.g., ℓ_2)



- Perceptron, SVM, Logistic Reg., all minimize convex approximations of the 0-1 loss (optimizing which is NP-hard; moreover it's non-convex/non-smooth)

Unsupervised Learning

Clustering and K-means

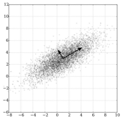
- No ground truth labels. The goal is to discover clusters (homogeneous groups/partitions) by only looking at inputs and their pairwise similarities
- A loss function can be defined in this case as well
- K-means uses a data distortion based objective: How much loss we are suffering if we assign each point to its nearest "center"

$$\boxed{L(\boldsymbol{\mu}, \mathbf{X}, \mathbf{Z}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||\boldsymbol{x}_n - \mu_k||^2}$$

- Non-convex objective. Optimized using the K-means heuristics which learns μ and $\mathbf{Z}$ in an alternating fashion
- The heuristic converges but not necessarily to an optima. Good initialization is important
- Several limitations but can be improved e.g., kernelizing it to handle non-convex cluster shapes

Dimensionality Reduction and PCA

- Learns a new representation ("embedding") using a new set of basis directions $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_D$ in which we represent the data
- Dimensionality reduction (a low dimensional embedding) is achieved if we only keep a few (say $K < D$) of these directions
- These are basically directions along which data has largest "spread"



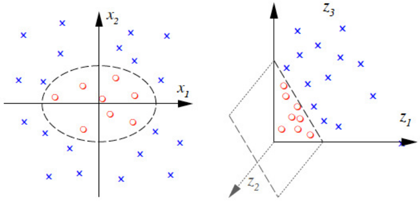
- Eigen-decomposition of the covariance matrix of data gives these direction (these are the top K eigenvectors, also known as the Principal Components)
- PCA to K-dims is also akin to saying that each data point is approximately a linear combination of K basis vectors, i.e., $\boldsymbol{x}_n \approx \sum_{k=1}^{K} z_{nk}\boldsymbol{u}_k$. Thus
- PCA does linear dim-red but can also be kernelized to yield nonlinear dim-red.

Nonlinear Learning via Kernels

Kernel Methods

- Can make a nonlinear learning problem (supervised/unsupervised) easier by mapping data to a higher dimensional space

$$\Phi : R^2 \to R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2) x_1 x_2, x_2^2)$$



- Every kernel function k implicitly defines feature map ϕ on the data, s.t.,

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{x}_m)$$

- Can thus kernelize any algorithm that only consists of inner products

Kernel Methods

Some things that you should know

- Mercer's condition: What makes a similarity function a kernel function?
- How to kernelize a learning algorithm ?
- Representer Theorem: The final solution has the general form

$$\hat{f} = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n) = \sum_{n=1}^{N} \alpha_n k(\boldsymbol{x}_n, .)$$

  .. basically, the solution lies in the span of the training data in the ϕ space

- The prediction on test data in general depends on all the training examples (this kernel methods can usually be expensive storage wise and also in terms of prediction time)