ICMLA 2010 Tutorial on

Sequence Labeling: Generative and Discriminative Approaches

Hidden Markov Models, Conditional Random Fields and Structured SVMs

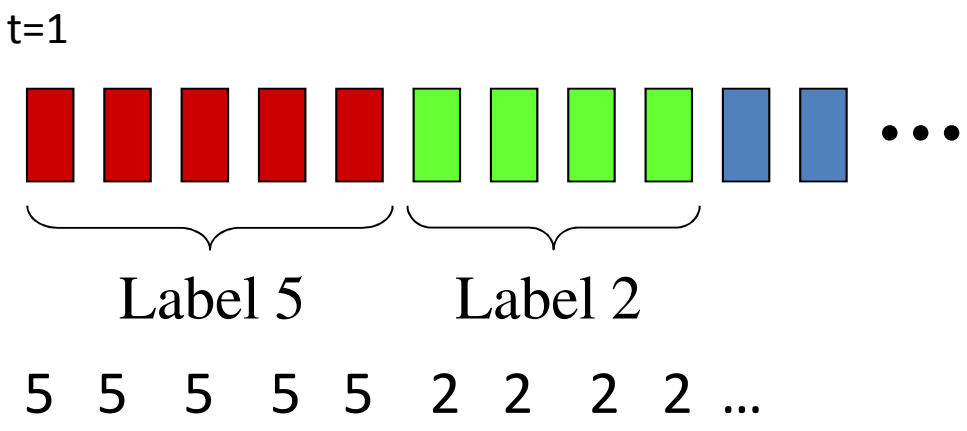
Hakan Erdogan
Sabanci University
Istanbul, Turkey

December 2010
ICMLA 2010, Bethesda, MD

Outline

Sequence labeling problem definition I

- Given a sequence of observations/feature vectors, determine an appropriate label/state for each observation
- We will assume the observations can be discrete or continuous, scalar or vector
- We assume the labels/states are discrete and from a finite set
- Try to reduce errors by considering the relations between the observations and states (observation-state) and the relation between neighboring states (state-state)



Sequence labeling applications

- Speech recognition
- Part-of-speech tagging
- Shallow parsing
- Handwriting recognition
- Protein secondary structure prediction
- Video analysis
- Facial expression dynamic modeling

Urns and balls example

- Assume there are two urns with black and white balls [Rabiner, 1989]
- One urn has more black than white (90% vs 10%) and vice versa
- Someone pulls out one ball at a time and shows us without revealing which urn he uses and puts it back into the urn
- He is more likely to use the same urn (90% chance) once he starts using one
- We are looking only at the sequence of balls and recording them

Questions about the urns and balls example

- Questions of interest:
  1. Can we predict which urn is used at a given time?
  2. What is the probability of observing the sequence of balls shown to us?
  3. Can we estimate/learn the ratio of balls in each urn by looking at a long sequence of balls if we did not know the ratios beforehand?

Jason Eisner's ice-cream example

- Try to guess whether the weather was hot or cold by observing only how many ice-creams (0, 1, 2 or 3+) Jason ate each day in a sequence of 30 days
- Two states and observations with 4 distinct values (discrete observations)
- Question: Can we determine if a day was hot or cold given the sequence of ice-creams consumed by Jason?
- Example excel sheet online (illustrates forward backward algorithm)
- Example also adopted in [Jurafsky and Martin, 2008]

Human activity labeling in an exercise video

- Assume we are given an exercise video of a single person and we are interested in labeling actions of the person as either "standing", "squatting" or "lying down" (assume for now that no other action is present)
- We track the subject and have a bounding box around her/him at each frame of the video
- We consider as features $\mathbf{x}_t = [h_t, w_t]^T$ where h_t is the height of the bounding box and w_t is the width of the bounding box
- So, we have continuous (real) observations and three labels
- Question: Given the height and width of the bounding boxes in all frames, can we determine the action type in each frame?

Approach, notation and variables

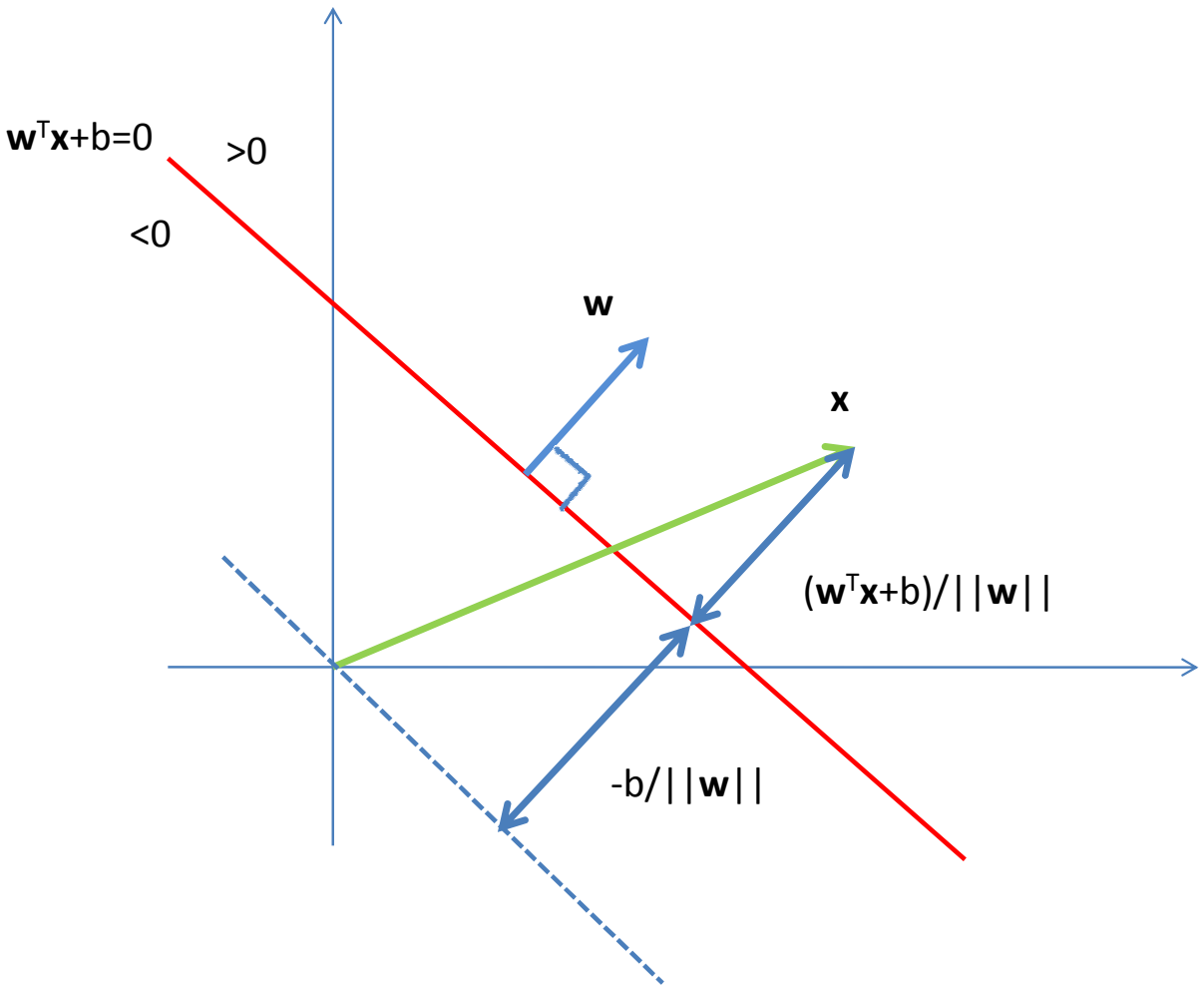
- We will first analyze binary and multi-class classification with linear models
- Multi-class classification will be the basis for understanding the sequence labeling problem
- Then, we will introduce HMM, CRF, and structured SVM approaches for sequence labeling
- Notation:
  - $\mathbf{x}$ is an observed feature vector, $\mathbf{x}_t$ a feature vector at sequence position t, $\mathbf{x}_{1:T}$ a sequence of feature vectors
  - y is a discrete label (or state), $y \in \mathcal{Y}$ where $\mathcal{Y} = \{-1, +1\}$ for binary classification, $\mathcal{Y} = [M] = \{1, 2, \ldots, M\}$ for multi-class classification
  - y_t is the label/state at sequence position t, $y_{1:T}$ is a sequence of labels/states
  - $\mathbf{w}$ and $\tilde{\mathbf{w}}$ are parameter vectors, w_j is the jth component
  - $\mathbf{F}(\mathbf{x}_{1:T}, y_{1:T})$ is a feature vector for CRF and structured SVM

Outline

Linear binary classification I

- Given training data $\{(\mathbf{x}_i, y_i) : i \in [N]\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y} = \{-1, +1\}$
- $[N] = \{1, \ldots, N\}$, $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ are feature vectors, y_i are class identities
- Find a weight vector $\tilde{\mathbf{w}} \in \mathbb{R}^{d+1}$ such that for test data $\mathbf{x}$, we obtain a score $\tilde{\mathbf{w}}^T \tilde{\mathbf{x}} = \mathbf{w}^T \mathbf{x} + b$ where:
  - $\tilde{\mathbf{x}} = [\mathbf{x}^T, 1]^T$ is the augmented feature vector
  - $\tilde{\mathbf{w}} = [\mathbf{w}^T, b]^T$ is the augmented weight vector, b is called the bias term
- $\tilde{\mathbf{w}}$ represents a hyperplane in $\mathbb{R}^{d+1}$, it is the normal vector to the hyperplane $\{\tilde{\mathbf{x}} : \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} = 0\}$ that passes through the origin

Linear binary classification II



Linear binary classification

- The score can be used to classify a new sample $\mathbf{x}$ into one of two classes by thresholding:

$$\hat{y} = \begin{cases} +1 & \text{if } \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \geq T \\ -1 & \text{if } \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} < T \end{cases}$$

- We can obtain an ROC curve (or DET curve) by changing the threshold T
- By default, we may assume $T = 0$ since the bias term is included in the model
- One can also obtain posterior probabilities $p(y|\mathbf{x})$ from the score $\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$
- The problem: how to obtain the "best" weight vector $\tilde{\mathbf{w}}$?
- We consider three different classifiers: Fisher's linear discriminant, logistic regression and support vector machines

Fisher's linear discriminant (FLD) I

- Assume each class' data are multi-variate Gaussian distributed with a common covariance matrix (homoscedastic)
- $p(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \mu_y, \Sigma)$ where Σ is the common covariance matrix, μ_y are class means

$$\mathcal{N}(\mathbf{x}; \mu_y, \Sigma) = \frac{\exp\left\{-\frac{1}{2}(x-\mu_y)^T\Sigma^{-1}(x-\mu_y)\right\}}{((2\pi)^d|\Sigma|)^{1/2}}$$

- Using Bayes' therorem, we can show that

$$p(y=1|\mathbf{x}) = \frac{p(\mathbf{x}|y=1)p(y=1)}{\sum_{y'\in\mathcal{Y}} p(\mathbf{x}|y')p(y')} = \sigma(\mathbf{w}^T\mathbf{x}+b)$$

where

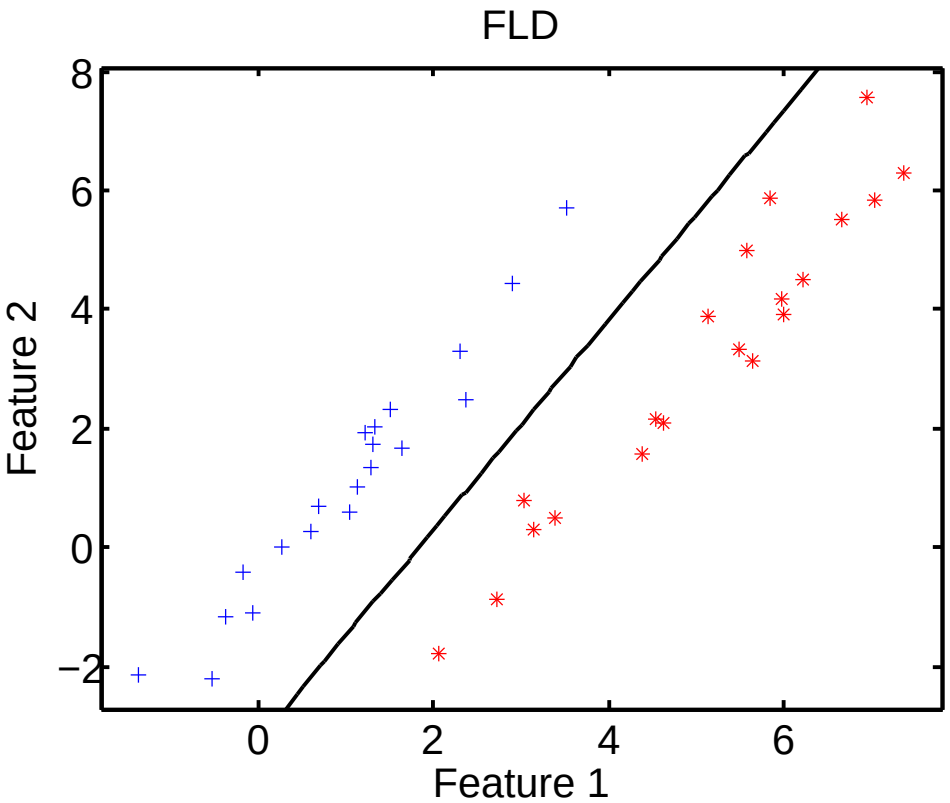
$$\sigma(t) = (1+\exp(-t))^{-1}$$

Fisher's linear discriminant (FLD) II

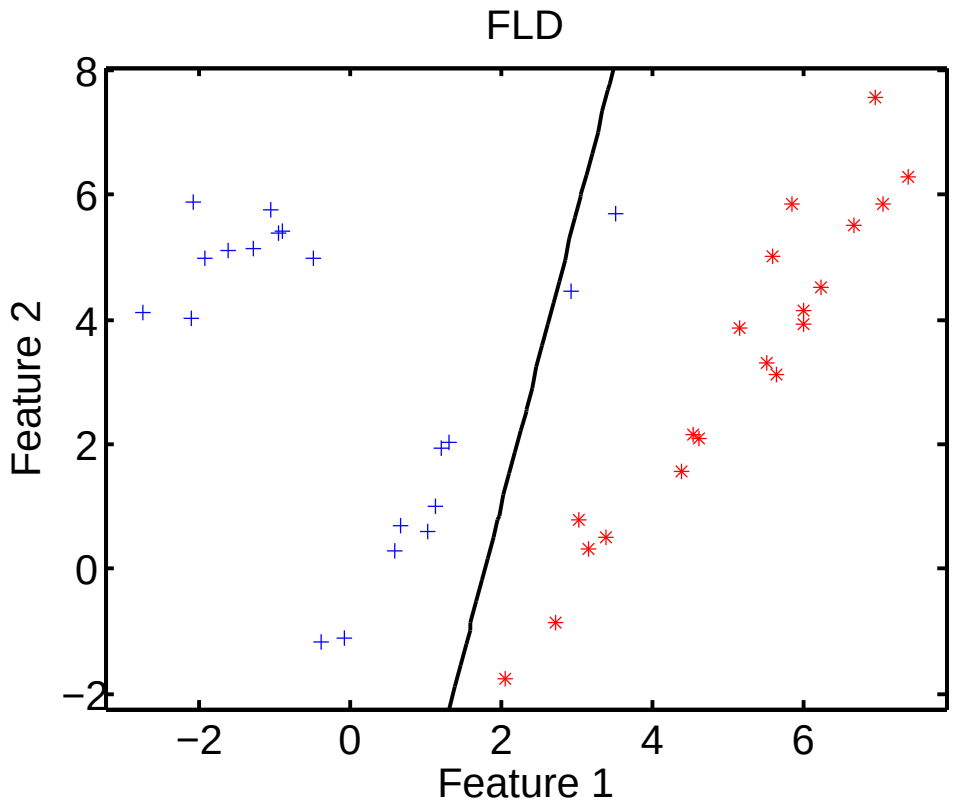
is the logistic sigmoid function, and

$$\begin{aligned} \mathbf{w} &= \Sigma^{-1}(\mu_1 - \mu_{-1}) \\ b &= -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_{-1}^T\Sigma^{-1}\mu_{-1} + \log(\frac{p(y=1)}{p(y=-1)}) \end{aligned} \quad (1)$$

- Fisher's linear discriminant yields a linear boundary for classification
- Fisher's linear discriminant is a **generative model** for **x** conditioned on y
- It seems a waste to find two class means (of size $2d$) and a common covariance matrix (size $d(d+1)/2$) where in the end what matters for classification is the weight vector **w** and the bias b (size $d+1$ only)



Modeling assumption valid



Modeling assumption invalid

Logistic regression I

- Discriminative linear model as opposed to FLD which is a generative model
- Assume $p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}})$ where $\sigma(.)$ is the logistic sigmoid [Bishop, 2006]
- Find parameters $\tilde{\mathbf{w}}$ by maximizing the conditional log-likelihood (CLL) of the class identities y, $p(y|\mathbf{x})$ (instead of $p(\mathbf{x}|y)$ in FLD, a generative model) using the training data
- Define $\pi_i(\tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_i)$

$$\hat{\tilde{\mathbf{w}}} = \arg\max_{\tilde{\mathbf{w}}} \sum_{y_i=1} \log(\pi_i(\tilde{\mathbf{w}})) + \sum_{y_i=-1} \log(1-\pi_i(\tilde{\mathbf{w}}))$$

Logistic regression II

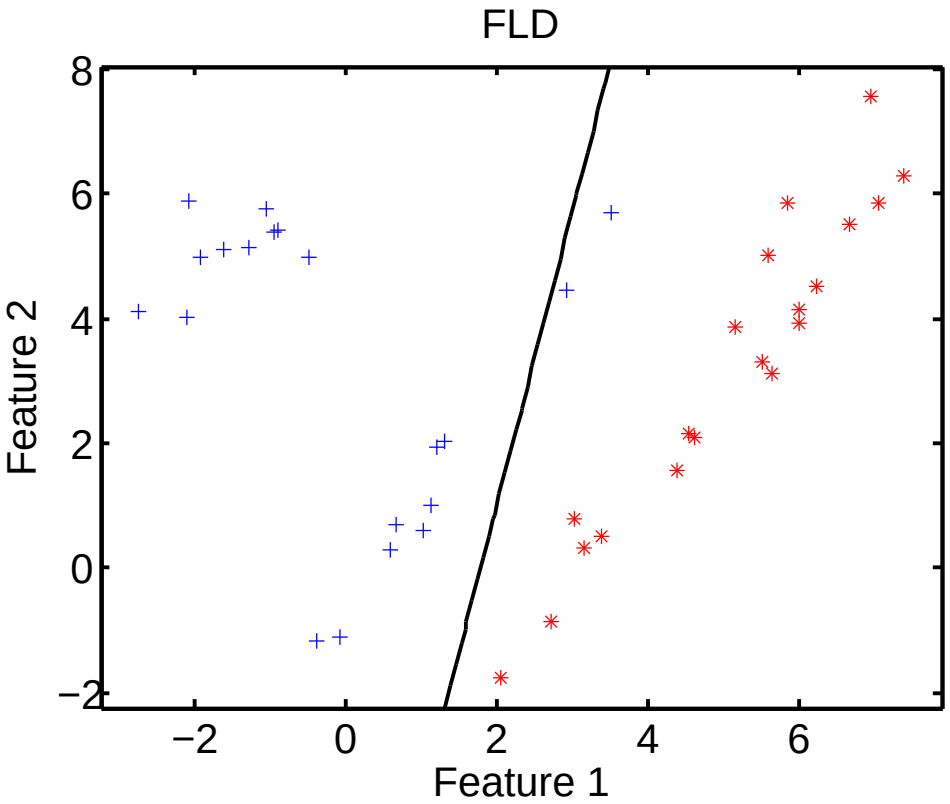
- Gradient of the log-likelihood wrt to the parameters is [Bishop, 2006]

$$\nabla L(\tilde{\mathbf{w}}) = \sum_{y_i=1} x_i(\pi_i(\tilde{\mathbf{w}}) - 1) + \sum_{y_i=-1} x_i \pi_i(\tilde{\mathbf{w}})$$

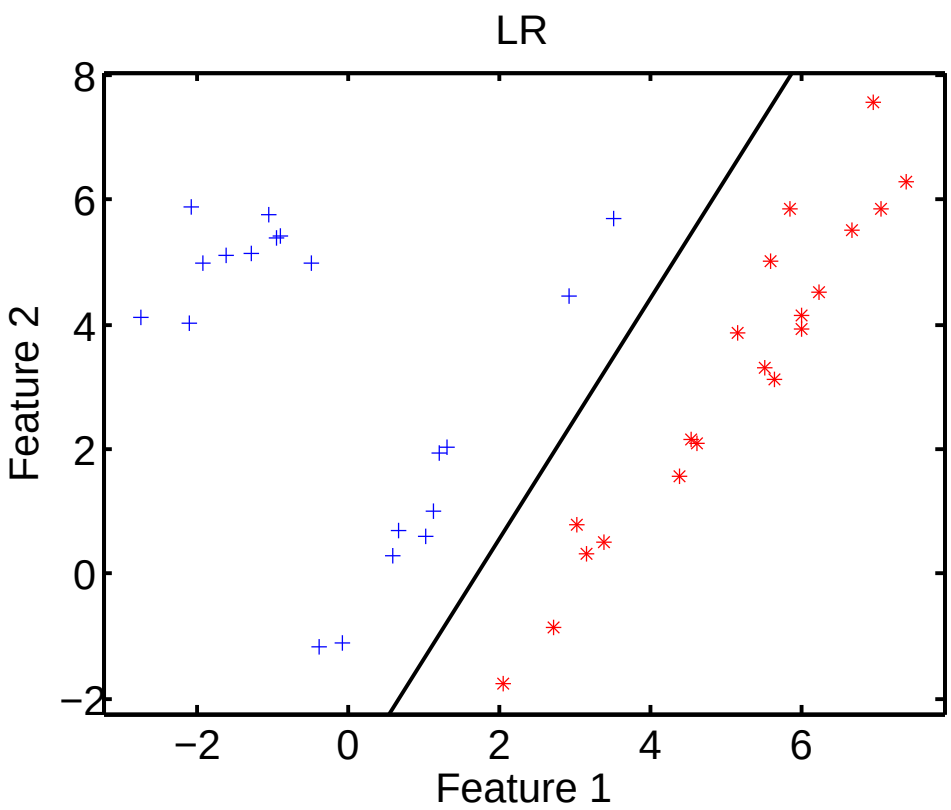
- There is an iterative reweighted least squares (IRLS) algorithm to solve for $\tilde{\mathbf{w}}$ given in [Bishop, 2006]
- We only estimate $d + 1$ parameters from data directly
- There is no need to assume a distribution $p(\mathbf{x}|y)$ since $\mathbf{x}$'s are given to us in a training scenario
- So, we only model $p(y|\mathbf{x})$ which becomes "discriminative modeling"
- The assumed parametric form of $p(y|\mathbf{x})$ comes from the generative FLD model though!
- Question: Can I get back a discriminatively trained conditional Gaussian distributions from logistic regression result $\tilde{\mathbf{w}}$?

Logistic regression III

- Answer is yes, choosing μ_y and Σ consistent with $\tilde{\mathbf{w}}$ obtained through logistic regression using equation 2 will give discriminatively trained parameters for Gaussians (note that they will not be ML trained parameters)



FLD when assumption invalid



LR with the same data

Support vector machines I

- A discriminative classifier that tries to maximize the soft margin between classes to increase generalizibility

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i$$

subject to:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad (2)$$

$$\xi_i \geq 0 \quad (3)$$

Support vector machines II

- The constraint optimization is simply equivalent to minimizing the following objective function wrt $\mathbf{w}$ and b without constraints (called the primal formulation)

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}(1 - y_i(\mathbf{w}^T\mathbf{x} + b))_+$$

where $(x)_+ = \max(x, 0)$ and C is a fixed parameter to be determined

Support vector machines III

- Usually, the dual formulation is used to solve the SVM problem since it appears to be easier, results in sparsity due to support vectors and enables using kernel functions for nonlinear separability

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

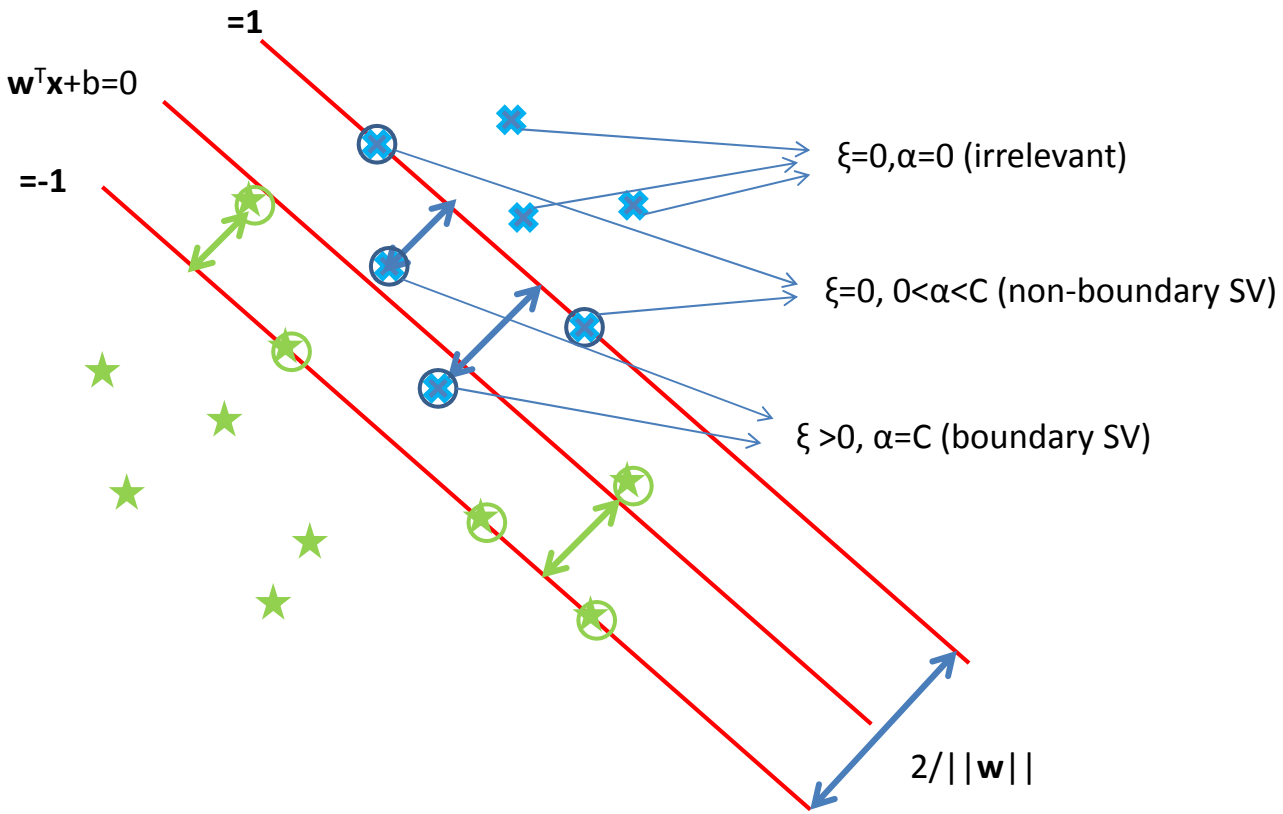
subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad (4)$$

$$0 \geq \alpha_i \geq C \quad (5)$$

where α_i are the dual variables (Lagrange multipliers for each constraint, that is training sample). Optimal weight vector can be found by $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$

Support vector machines IV



- There is sparsity in α_i and most α_i turn out to be zero
- The $\mathbf{x}_i$ that correspond to nonzero α_i are called support vectors
- If $\alpha_i = C$, then $\mathbf{x}_i$ are boundary support vectors

Support vector machines V

- The support vectors are the only training samples that matter in determining the optimal weights (hence the separating hyperplane)
- To find optimal b, take any nonzero α_i and use the equation $\alpha_i \left(y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1\right) = 0$ (comes from KKT conditions) or average values from all nonzero α_i's.

Regularized empirical risk (RER) minimization I

- Unifying framework for different linear classifiers
- Consider a predictor function $f_\theta(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^m$ that can be used to predict output labels y from features $\mathbf{x}$, m is the number of prediction values (discriminants)
  - For the linear binary classification problem, $m = 1$, $\theta = \tilde{\mathbf{w}}$ and $f_{\tilde{\mathbf{w}}}(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$
- Define a "loss function" $\mathcal{L} : \mathbb{R}^m \times \mathcal{Y} \rightarrow \mathbb{R}^+$ that measures the cost of prediction
- We define the expected risk as follows:

$$\mathcal{R}(\theta) = \int \mathcal{L}\left(f_\theta(\mathbf{x}), y\right) p(\mathbf{x}, y) d\mathbf{x} dy$$

Regularized empirical risk (RER) minimization II

- Since the joint distribution $p(\mathbf{x}, y)$ is usually unknown, we can find an estimate of the risk from the training data called the "empirical risk" which needs to be minimized wrt the parameters θ

$$\hat{\mathcal{R}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f_\theta(\mathbf{x}_i), y_i\right)$$

- This can be interpreted as the training set error rate
- Usually we add a penalty term $\Omega(\theta)$ for the parameters which penalizes complex (or large) predictor functions to arrive at what is called the "regularized empirical risk"

$$\hat{\mathcal{R}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f_\theta(\mathbf{x}_i), y_i\right) + \Omega(\theta)$$

Regularized empirical risk (RER) minimization III

- Note that "structured risk" minimization is similar where the regularization function is replaced by the Vapnik-Chervonenkis (VC) confidence of the predictor function (which is also a measure of complexity of the predictor)
- All previous methods (FLD, LR and SVM) can be seen as variants of regularized empirical risk minimization by utilization of a different loss function as we elaborate in the following slides

Loss functions I

- RER for linear binary classification is as follows:

$$\hat{\mathcal{R}}(\tilde{\mathbf{w}}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i, y_i\right) + \Omega(\tilde{\mathbf{w}})$$

- Using different loss functions yield different methods for linear binary classification

LS loss

$$\mathcal{L}(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}, y) = (\tilde{\mathbf{w}}^T\tilde{\mathbf{x}} - t_y)^2$$

- Here t_y are regression targets for each class in least squares regression. Using LS loss with no regularization is equivalent to FLD when $t_y = y\frac{N}{N_y}$ where N_y is the number of samples in class y [Bishop, 2006]
- LS-SVM uses $t_y = y$ and a quadratic regularizer [Suykens and Vandewalle, 1999]. Without regularization, LS-SVM is similar to FLD
- Regularized LDA [Friedman, 1989] is equivalent to using a regularization function in the RER framework

BNLL loss

$$\mathcal{L}(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}, y) = \log\left(1 + \exp\left\{-y\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}\right\}\right)$$

- Using binomial negative log-likelihood (BNLL) loss function with no regularization is equivalent to performing a logistic regression classification

Hinge loss

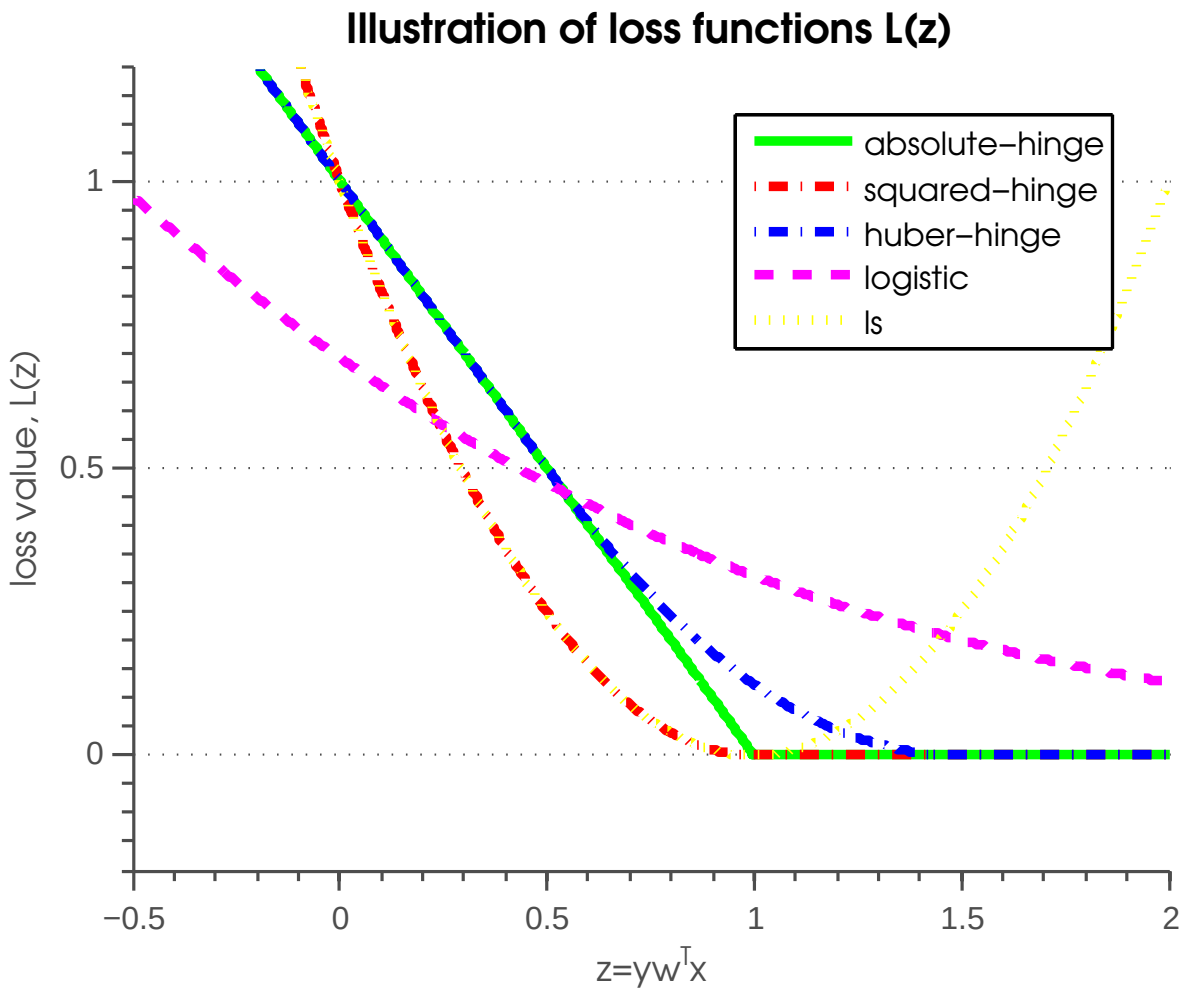
$$\mathcal{L}(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}, y) = (1 - y\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})_+$$

- Using hinge loss function and a regularization function $\Omega(\tilde{\mathbf{w}}) = \lambda\tilde{\mathbf{w}}^T\mathbf{D}\tilde{\mathbf{w}}$ (where $\mathbf{D}$ is a diagonal matrix with all ones in the diagonal except for the last entry which is zero) is equivalent to performing an SVM classification
- Note that $\Omega(\tilde{\mathbf{w}}) = \lambda\tilde{\mathbf{w}}^T\tilde{\mathbf{w}}$ is also used which is slightly different (in liblinear [Fan et al., 2008])

Other loss functions

- There are many other loss functions defined in the literature [LeCun et al., 2006]
- Huber-hinge loss is one which smooths the edge of the hinge loss so that we end up with a differentiable loss function
- Squared-hinge loss is the square of the hinge loss which is also differentiable (called L_2-SVM)

Loss function plots



Regularization functions I

- Although originally FLD and LR do not use regularization, it was shown to be beneficial in all cases
- Typically an L_2-norm regularization $\Omega(\tilde{\mathbf{w}}) = \lambda||\tilde{\mathbf{w}}||^2$ is used
- λ is a hyper-parameter that needs to be determined (next slide)
- Regularization helps in all classifiers arguably due to
  - improving test accuracy due to penalizing the complexity of the classifier
  - avoiding overtraining/overfitting
  - avoiding numerical problems in implementation.
- L_1-norm is used when sparse parameter vectors are desired
  - For example L_1-regularized L_1-SVM is considered in [Mangasarian, 2006].

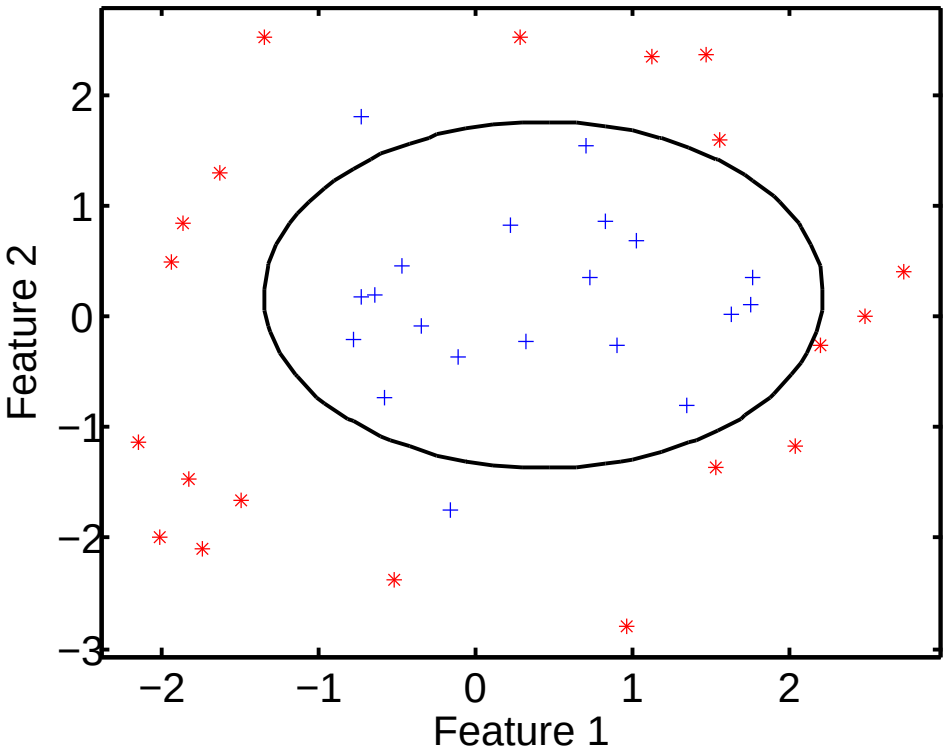
Estimating the hyperparameters I

- The hyperparameter λ can be found by grid search on validation data, that is by evaluating the performance of the trained classifier on the validation data
- Multiple fold cross-validation on the training data can also be performed for this purpose
- Bayesian "evidence framework" may be used with some losses to find the hyperparameters, this requires re-interpreting the empirical risk optimization as a maximum aposteriori probability (MAP) estimation problem [Hoffmann, 2007] - Bayesian LDA
- Perturbation analysis may be used [Zheng et al., 2009]
- Other ad-hoc methods exist as well

RER optimization methods

- LS and logistic losses are differentiable, so primal algorithms such as Newton's method work very well
- LS loss has closed form solution
- For logistic loss, Newton's method results in iterative re-weighted least squares (IRLS) formulation
- For hinge loss, one needs to go to the dual QP problem and solve it in the dual
- For squared-hinge or huber-hinge losses, one can solve it in the primal domain as well
- libsvm and liblinear are good libraries for solving RER formulations

How to obtain nonlinear classifiers? I



- Sometimes, class data are not separable by a line!
- Replace $\mathbf{x}$ with $\phi(\mathbf{x})$ which is a nonlinear mapping into a higher dimensional space
- If $\phi(\mathbf{x})$ is known explicity, you may use it instead of $\mathbf{x}$ to solve the problem

- Use the kernel trick to replace inner products with the kernel function: $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$

How to obtain nonlinear classifiers? II

- No need to know what $\phi(.)$ is, if we use the kernel trick (ϕ can be infinite dimensional) $\rightarrow$ just replace inner products with the kernel function
- Kernel versions usually solved in the dual, but it was shown it is possible to solve in the primal as well [Chapelle, 2007]
- We will not focus on kernel versions of these classifiers in this talk!
- linear classifiers are as good as or better than kernel versions for large scale learning [Fan et al., 2008]

Outline

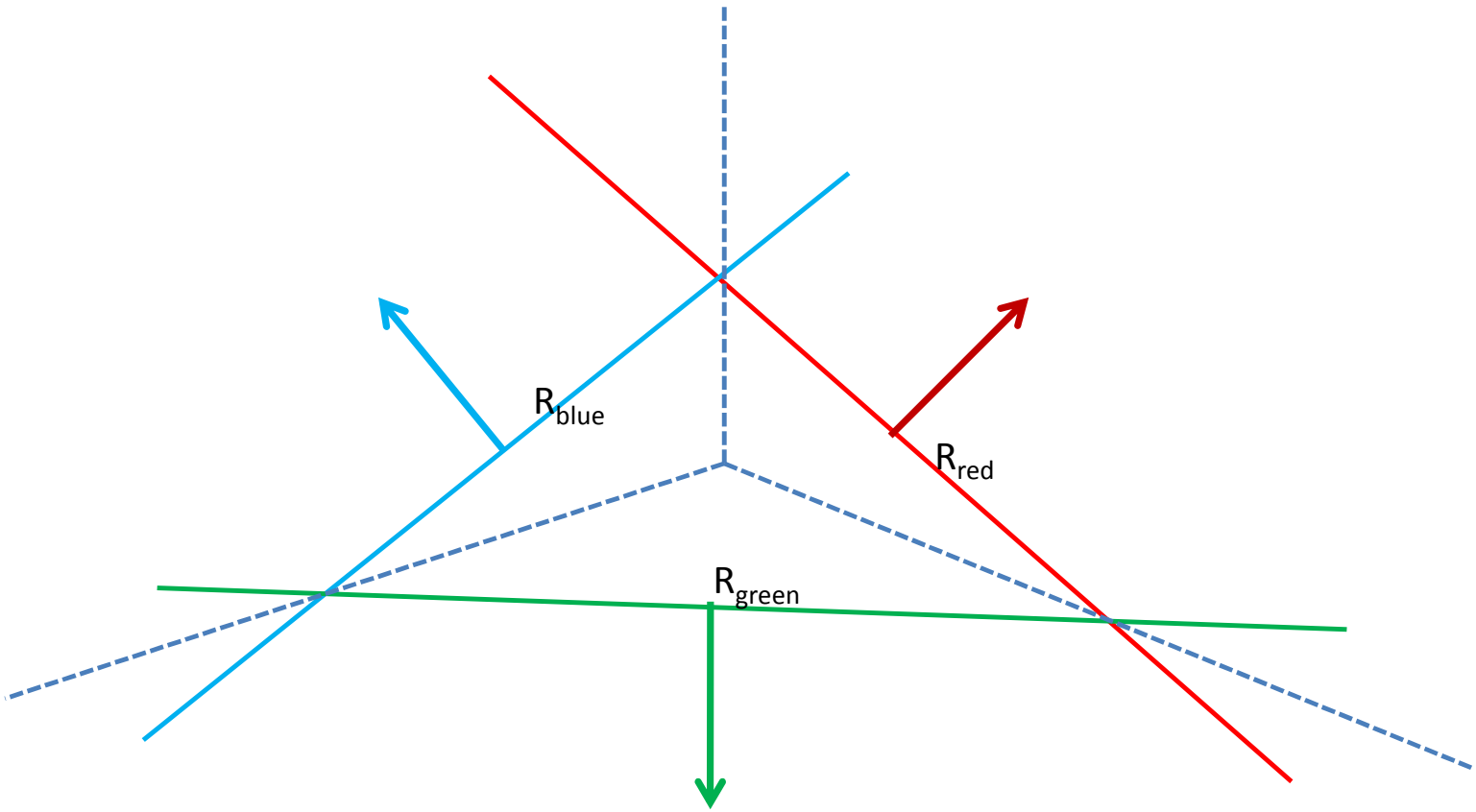
Multi-class classification

- Using multiple binary classifiers and combining them (multiple machine)
  - One-vs-all
  - One-vs-one
  - Error correcting output codes (ECOC)
- Direct multi-class classification (single machine) (explanation next slide)
- A paper compared these approaches and found that one-vs-one gave the best result (used in libsvm) [Hsu and Lin, 2002]
- For structured classification, we need direct multi-class classification since almost impossible to enumerate all possible output labels (topic of future lectures)

Linear multi-class classification

- Given training data $\{(\mathbf{x}_i, y_i) : i \in [N]\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y} = [M]$
- $\mathbf{x}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ are feature vectors, y_i are class identities
- Find a set of weight vectors $\tilde{\mathbf{w}}_y \in \mathbb{R}^{d+1}$ such that for test data $\mathbf{x}$, we obtain a score for class y as $\tilde{\mathbf{w}}_y^T \tilde{\mathbf{x}} = \mathbf{w}_y^T \mathbf{x} + b_y$
- Classification is done by $\hat{y} = \arg\max_y \tilde{\mathbf{w}}_y^T \tilde{\mathbf{x}}$
- Each $\tilde{\mathbf{w}}_y$ represents a hyperplane in $\mathbb{R}^{d+1}$

Geometry of linear multi-class discriminants



Generative multi-class classification I

- Given class conditional probability distributions $p(\mathbf{x}|y)$
- Obtain class posteriors as:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{\sum_{y'} p(\mathbf{x}|y')p(y')}$$

- If class conditional probability distributions are from the exponential family

$$p(\mathbf{x}|y;\boldsymbol{\theta}_y) = h(\mathbf{x}) \exp\left\{\boldsymbol{\theta}_y^T \mathbf{t}(\mathbf{x}) - A(\boldsymbol{\theta}_y)\right\}$$

where $\boldsymbol{\theta}$ are the parameters (or transformed parameters) of the pdf, $\mathbf{t}(\mathbf{x})$ is the sufficient statistics vector, $A(\boldsymbol{\theta})$ is the log-partition function and $h(\mathbf{x})$ is a base measure independent of the parameters.

Generative multi-class classification II

- Then, the conditional likelihood has the form:

$$p(y|\mathbf{x}) = \frac{\exp\{\tilde{\mathbf{w}}_y^T \phi(\mathbf{x})\}}{\sum_{y'} \exp\{\tilde{\mathbf{w}}_{y'}^T \phi(\mathbf{x})\}}$$

  where $\tilde{\mathbf{w}}_y = \left[\boldsymbol{\theta}_y^T, -A(\boldsymbol{\theta}_y) + log(p(y))\right]^T$ and $\phi(\mathbf{x}) = \left[\mathbf{t}(\mathbf{x})^T, 1\right]^T$
- This form of the conditional likelihood is called normalized exponential or *softmax* function.
- Hence, if we map the features $\mathbf{x}$ using the sufficient statistics and add a constant feature, conditional likelihood will have a log-linear form

Generative multi-class classification III

- For example, for the Gaussian setup with means $\boldsymbol{\mu}_y$ and equal covariances (known Σ) (multi-class FLD), we get the following

$$\begin{aligned} \boldsymbol{\theta}_y &= \boldsymbol{\mu}_y \\ \mathbf{t}(\mathbf{x}) &= \mathbf{x} \\ \mathbf{w}_y &= \Sigma^{-1}\mu_y \\ b_y &= -\frac{1}{2}\mu_y^T\Sigma^{-1}\mu_y + \log(p(y)) \end{aligned} \tag{6}$$

- Exercise: Perform the same analysis when both $\boldsymbol{\mu}_y$ and $\boldsymbol{\Sigma}_y$ are different for each class. What are the sufficient statistics and parameters then?

Multi-class (multinomial) logistic regression I

- Assuming $p(y|\mathbf{x}) = \frac{\exp\{\tilde{\mathbf{w}}_y^T \phi(\mathbf{x})\}}{\sum_{y'} \exp\{\tilde{\mathbf{w}}_{y'}^T \phi(\mathbf{x})\}}$ we can maximize the conditional log-likelihood of the training data

$$\log p(y_1, \ldots, y_N | \mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{i=1}^{N} \log p(y_i | \mathbf{x}_i)$$

- This yields the following negative log likelihood (NLL) objective function to minimize

$$\sum_{i=1}^{N} -\tilde{\mathbf{w}}_{y_i}^T \phi(\mathbf{x}_i) + \log \left(\sum_{y'} \exp\{\tilde{\mathbf{w}}_{y'}^T \phi(\mathbf{x}_i)\} \right)$$

- This objective can be minimized using gradient-based techniques.

Multiclass SVM I

- Multiclass SVM was formulated in [Crammer and Singer, 2001] as follows
- First define $\mathbf{w} = [\mathbf{w}_1^T, \mathbf{w}_2^T, \ldots, \mathbf{w}_M^T]^T$ as concatenation of weight vectors.

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i$$

subject to:

$$\begin{aligned} \tilde{\mathbf{w}}_{y_i}^T\tilde{\mathbf{x}}_i - \tilde{\mathbf{w}}_{y'}^T\tilde{\mathbf{x}}_i &\geq 1-\xi_i, \quad \forall i, y' \neq y_i \\ \xi_i &\geq 0 \end{aligned} \tag{7}$$

- Let $\bar{y}_i = \arg\max_{y' \neq y_i} \tilde{\mathbf{w}}_{y'}^T\tilde{\mathbf{x}}_i$

Multiclass SVM II

- The first set of inequalities can also be written as:

$$\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i - \tilde{\mathbf{w}}_{\bar{y}_i}^T \tilde{\mathbf{x}}_i \geq 1 - \xi_i, \quad \forall i$$

- The problem can be formulated as an unconstrained minimization problem as follows:

$$\min \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} (1 - \tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \max_{y' \neq y_i} \tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i)_+$$

- In multi-class classification, some class pairs may be less costly to be confused, so we can define a *label-loss* function $\Delta(y, y')$ which gives a cost to replacing y with y' during classification, which can be incorporated in the SVM constraints formulation [Tsochantaridis et al., 2005]

Multiclass SVM III

- "Margin rescaling" yields the set of constraints

$$\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i - \tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i \geq \Delta(y_i, y') - \xi_i, \quad \forall i, y' \neq y_i$$

- whereas "slack rescaling" yields the following set of constraints

$$\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i - \tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i \geq 1 - \frac{\xi_i}{\Delta(y_i, y')}, \quad \forall i, y' \neq y_i$$

- There is a cutting-plane algorithm in [Tsochantaridis et al., 2005] which solves a series of constrained optimization algorithms to solve these problems
- Furthermore, [Joachims et al., 2009] introduced 1-slack constraints, 1-slack formulation with margin rescaling uses the constraints

$$\frac{1}{N} \sum_{i=1}^{N} (\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i - \tilde{\mathbf{w}}_{y_i'}^T \tilde{\mathbf{x}}_i) \geq \frac{1}{N} \sum_{i=1}^{N} \Delta(y_i, y_i') - \xi, \quad \forall (y_i')_{i=1}^N$$

Multiclass SVM IV

- 1-slack formulation with slack rescaling is also provided in the same paper
- There are efficient cutting-plane algorithms in [Joachims et al., 2009] for solving 1-slack problems in the dual space
- These solvers are provided in the svm-struct package

Regularized empirical risk for multi-class I

- Remembering RER

$$\hat{\mathcal{R}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}\left(f_\theta(\mathbf{x}_i), y_i\right) + \Omega(\theta)$$

- For multi-class, the parameters are $\tilde{\mathbf{w}} = [\tilde{\mathbf{w}}_1^T, \ldots, \tilde{\mathbf{w}}_M^T]^T$ and the loss functions for each type are as follows:
- multi-class FLD:

$$\mathcal{L}\left(f_{\tilde{\mathbf{w}}}(\mathbf{x}_i), y_i\right) = \sum_{y=1}^{M}(\tilde{\mathbf{w}}_y^T\tilde{\mathbf{x}}_i - t(y, y_i))^2$$

where $t(y, y_i)$ are appropriate targets for class y for least squares regression when the true class is y_i

Regularized empirical risk for multi-class II

- [Ye, 2007] shows that using $t(y, y_i) = \sqrt{N/N_y} - \sqrt{N_y/N}$ when $y = y_i$ and $t(y, y_i) = -\sqrt{N_{y_i}/N}$ otherwise, is equivalent to linear discriminant analysis
- FLD is equivalent to using a least squares loss function
- multi-class LR:

$$\mathcal{L}\left(f_{\tilde{\mathbf{W}}}(\mathbf{x}_i), y_i\right) = -\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \log\left(\sum_{y'} \exp\{\tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i\}\right)$$

- multi-class SVM:

$$\mathcal{L}\left(f_{\tilde{\mathbf{W}}}(\mathbf{x}_i), y_i\right) = (1 - \tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \max_{y' \neq y_i} \tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i)_+$$

Regularized empirical risk for multi-class III

- Re-writing the SVM loss function is possible as follows:

$$\mathcal{L}\left(f_{\tilde{\mathbf{W}}}(\mathbf{x}_i), y_i\right) = -\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \max_{y'}(\tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i + 1 - \delta_{y,y_i})$$

- This form is equivalent to the one before since this is guaranteed to be nonnegative
- We can generalize using label-loss $\Delta(y, y')$ and "margin rescaling" introduced before to get

$$\mathcal{L}\left(f_{\tilde{\mathbf{W}}}(\mathbf{x}_i), y_i\right) = -\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \max_{y'}(\tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i + \Delta(y_i, y'))$$

- We can replace the max with a softmax (log-sum-exp) to get an "approximation" SVM-softmax

$$\mathcal{L}\left(f_{\tilde{\mathbf{W}}}(\mathbf{x}_i), y_i\right) = -\tilde{\mathbf{w}}_{y_i}^T \tilde{\mathbf{x}}_i + \log\left(\sum_{y'} \exp\{\tilde{\mathbf{w}}_{y'}^T \tilde{\mathbf{x}}_i + \Delta(y_i, y')\}\right)$$

Regularized empirical risk for multi-class IV

- Comparing the LR loss function with the SVM-softmax one, we see that they are similar except for the addition of the label-loss function to provide additional margin for confusable classes in the SVM-softmax
- It is possible to get the RER expression for "slack rescaling" and 1-slack versions of margin and slack rescaling as well
- For other possible loss functions, for example see [LeCun et al., 2006]

Optimization algorithms I

- Multi-class FLD has closed form solution
- Multi-class LR can be solved using Newton's method in the primal yielding an iterative re-weighted least squares algorithm
- Multi-class SVM can be directly solved from the dual problem [Crammer and Singer, 2001, Fan et al., 2008] especially if number of training samples N is small
- Cutting-plane algorithms are attractive alternatives [Tsochantaridis et al., 2005, Joachims et al., 2009]

Optimization algorithms II

- Multi-class SVM-softmax can be solved using Newton's method in the primal
- If Newton algorithm's Hessian is too big to be computed efficiently, L-BFGS can be used
- Stochastic gradient algorithms are fast and applicable, but need to be careful with convergence and choosing step sizes [Bottou, 2004]
- The optimal choice of the algorithm depends on the values of feature dimension d, number of classes M and number of training samples N

Log-linear models

- A general log-linear conditional model is given by:

$$p(y|\mathbf{x};\mathbf{w}) = \frac{1}{Z(x,\mathbf{w})} \exp\left\{\sum_j w_j F_j(\mathbf{x},y)\right\}$$

where

$$Z(x,\mathbf{w}) = \sum_{y'} \exp\left\{\sum_j w_j F_j(\mathbf{x},y')\right\}$$

is called the partition function.

- Note that multi-class LR is a log-linear model where we define a concatenated weight vector $\mathbf{w} = [\tilde{\mathbf{w}}_1^T, \ldots, \tilde{\mathbf{w}}_M^T]^T$ and where the feature vector $F(\mathbf{x},y) = \tilde{\mathbf{x}} \otimes \mathbf{e}_y$ where $\mathbf{e}_y$ is the unit vector with one in position y and zeros elsewhere and $\otimes$ denotes the Kronecker product.

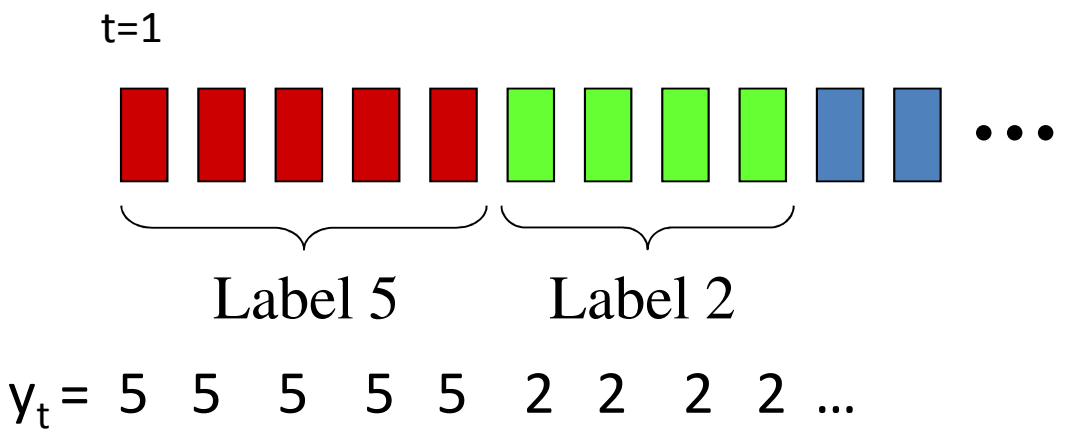
Outline

Sequence labeling problem definition I

- Given a sequence of features $\mathbf{x}_{1:T}$, find appropriate labels $y_{1:T}$ where each $y_t \in \mathcal{Y}$, we can assume wlog that $\mathcal{Y} = [M]$, a finite set
- This is a hard problem and the number of possible $y_{1:T}$ is too high, namely M^T
- We need additional assumptions on output labels y_t, such as being Markov
- Supervised learning problem: given training data sequences $\left\{(x_{1:T}^{(i)}, y_{1:T}^{(i)}) : i = 1, \ldots, N\right\}$, find a model that will predict $y_{1:T}$ given testing data $x_{1:T}$
- Note that, training (as well as test) sequences can be of different length T, but we do not explicitly indicate it to avoid clutter in our representation

Sequence labeling problem definition II

- Partially supervised learning: We do not know the label sequence $y_{1:T}^{(i)}$, but we know a sequence-specific grammar that the label sequence should obey (common case in speech recognition)



What is a hidden Markov model? I

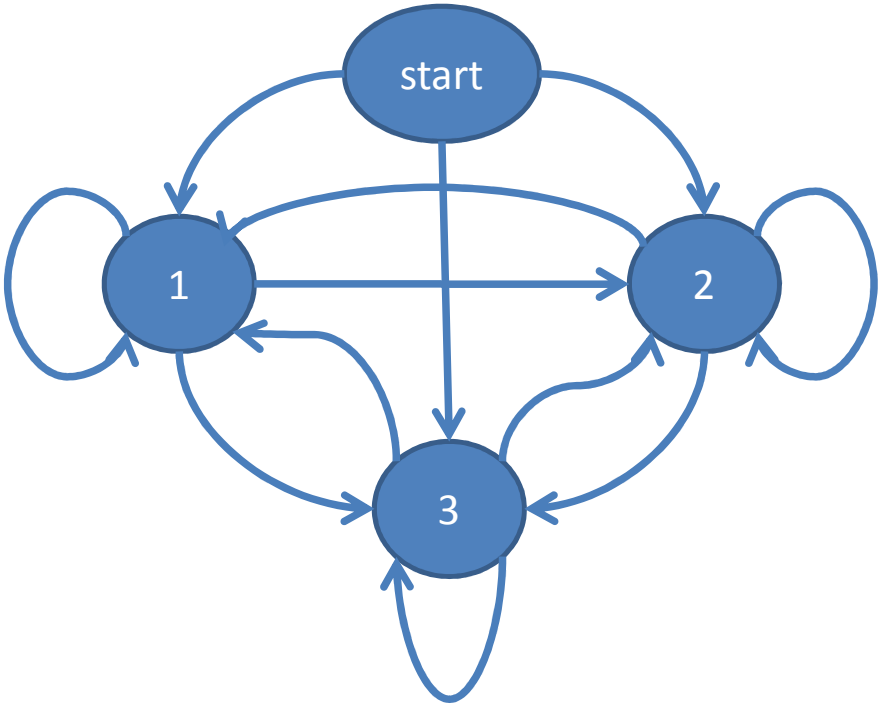
- A tool that helps us solve sequence labeling problems
- Observations $\mathbf{x}_{1:T}$ are modeled by a state machine (that is hidden) that generates them (generative model)
- States y_t correspond to labels, state sequence is $y_{1:T}$
- A finite set of labels is possible, $y_t \in \mathcal{Y}$ where $|\mathcal{Y}|$ is finite
- Markov assumption $p(y_t|y_{t-1}, y_{t-2}, \ldots, y_1) = p(y_t|y_{t-1})$
- Transition from one state (y_{t-1}) to another (y_t) occurs at each time instant
- Meanwhile an observation $(\mathbf{x}_t)$ is emitted after the transition
- Parameters of the model:
  - Probabilities of transitions among states
  - Probabilities of emission of observations from states
  - Probabilities of starting at states

Three views of HMMs

An HMM can be viewed in three different ways

- State transition diagram
- Graphical model
- Trellis / lattice diagram

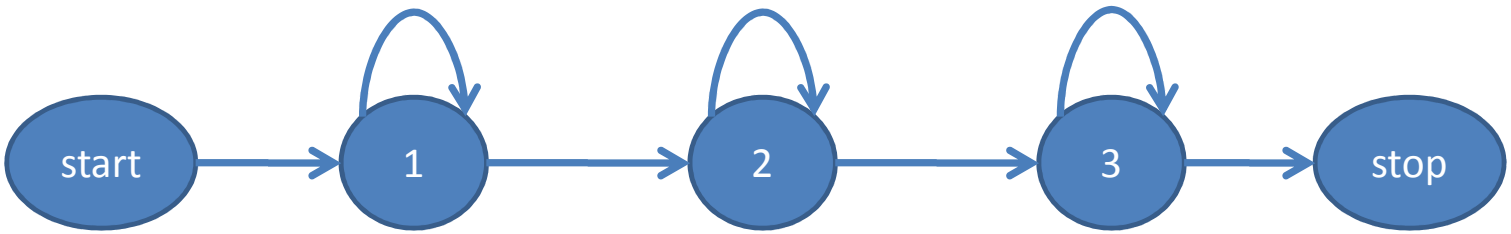
State transition diagram - fully connected



Time is not explicitly shown in this diagram, at each time instant a transition followed by an emission occurs

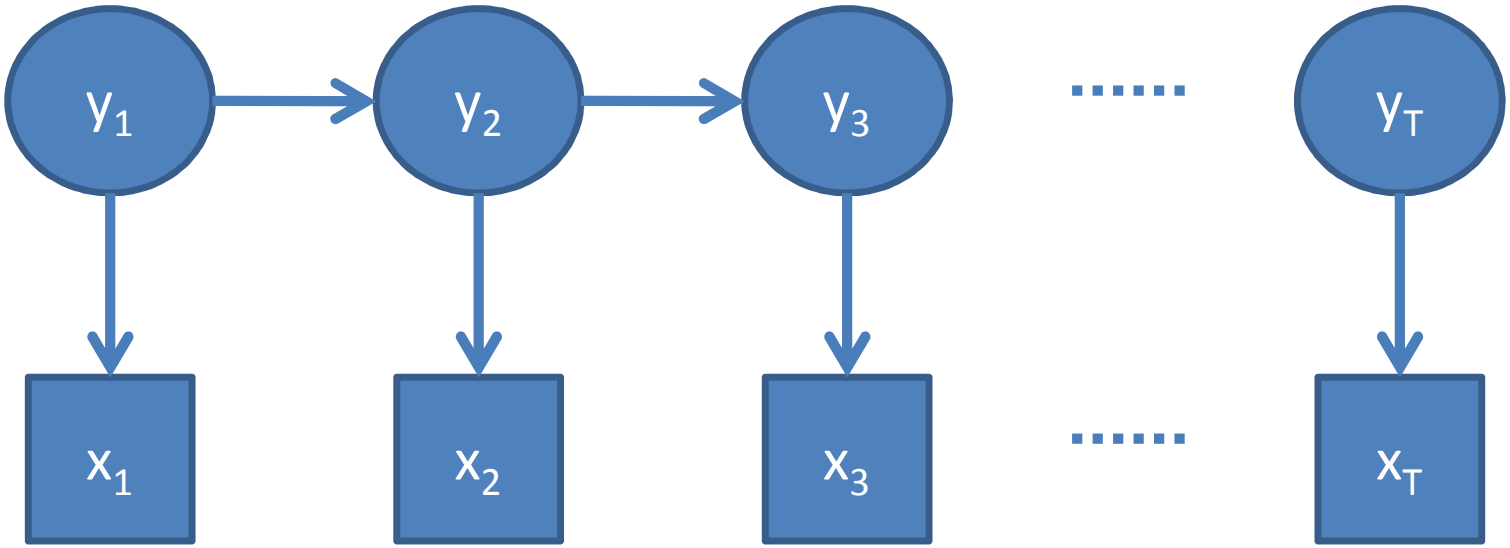
All transitions are possible with a certain probability in this example

State transition diagram - left-to-right

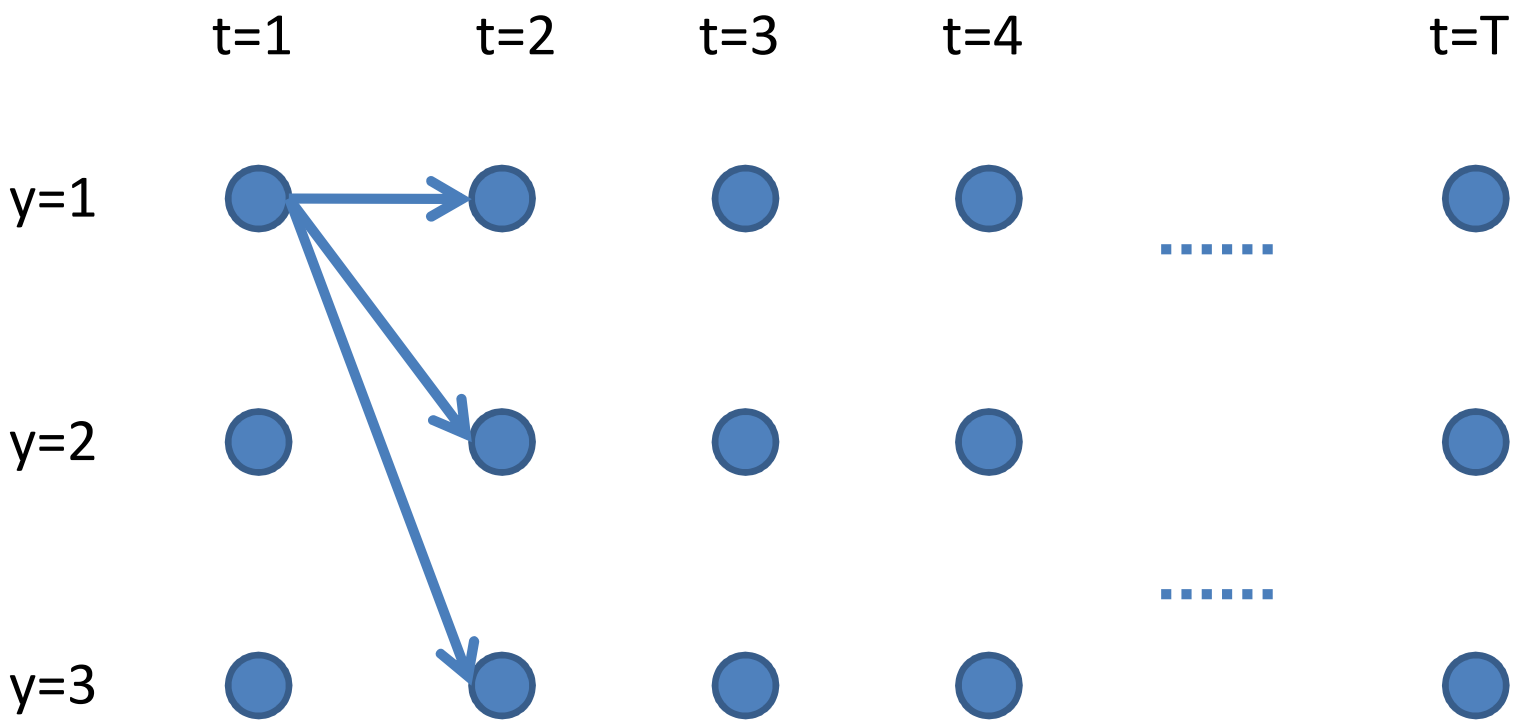


Some transitions are not possible (their probabilities are set to zero)

Graphical model

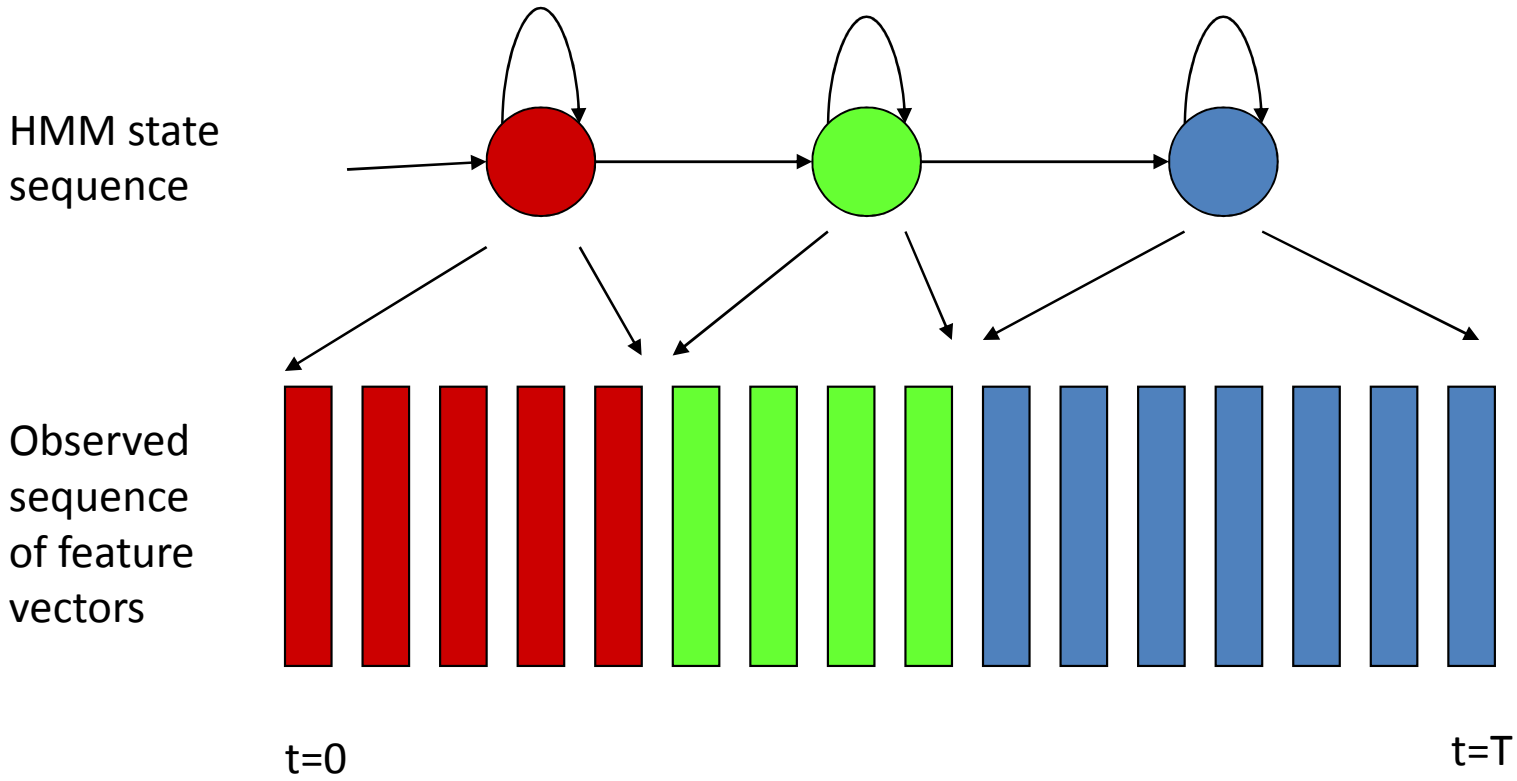


Trellis / lattice



Observations are not shown, the labels (states) are explicitly shown
Graphical model is expanded at each time instant to reveal all possible states

A possible alignment



Depicting a possibility of alignment of observed data to an underlying left-to-right HMM

Variables

- Observations $\mathbf{x}_{1:T}$
  - $\mathbf{x}_t \in \mathbb{R}^d$ for continuous observations HMM
  - $\mathbf{x}_t \in [N_o]$ for discrete observations HMM
- $y_{1:T}$ state sequence, $y_t \in [M]$ is the state at time t
- $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$: model parameters
  - $\mathbf{A}$ where $A_{ij} = p(y_{t+1} = j | y_t = i)$ is the transition matrix
  - For discrete observations $\mathbf{B}$ is a matrix where $B_{ik} = p(x_t = k | y_t = i)$ are emission probabilities
  - For continuous observations with Gaussian emission distributions we have $p(\mathbf{x}_t | y_t = i) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, we may think of $\mathbf{B}$ as the set of mean and (co)variance parameters $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)_{i=1}^M$
  - $\boldsymbol{\pi}$ where $\pi_i = p(y_1 = i)$ initial state probabilities, we can remove $\boldsymbol{\pi}$ if we introduce a "start" state which has initial probability of one

Rabiner's three problems of HMMs

- Problem 1: Probability/likelihood calculation: Given an observation sequence, how can I calculate the probability of observing it given an underlying HMM model $p(\mathbf{x}_{1:T}|\lambda)$
- Problem 2: Alignment/decoding/inference: What is the most likely state sequence given an observation sequence and an HMM model? $y^*_{1:T} = \arg\max_{y_{1:T}} p(y_{1:T}|x_{1:T}, \lambda)$
  - We may also be interested in $y^*_t = \arg\max_{y_t} p(y_t|x_{1:T}, \lambda)$
- Problem 3: Training/learning: How can I train the parameters of an HMM given training data $\mathbf{x}^{(i)}_{1:T}$? How to choose λ to maximize $\prod_i p(\mathbf{x}^{(i)}_{1:T}|\lambda)$?
  - Note that, if we are given $(\mathbf{x}^{(i)}_{1:T}, y^{(i)}_{1:T})$ (aka fully supervised training), maximum-likelihood training becomes just a counting process

Problem 1: Computing $P(\mathbf{x}_{1:T}|\lambda)$

$$\begin{aligned} p(\mathbf{x}_{1:T}|\lambda) &= \sum_{y_{1:T}} p(\mathbf{x}_{1:T}, y_{1:T}|\lambda) \\ &= \sum_{y_{1:T}} p(\mathbf{x}_{1:T}|y_{1:T}, \lambda) p(y_{1:T}|\lambda) \end{aligned}$$

where $p(\mathbf{x}_{1:T}|y_{1:T}, \lambda) = \prod_t p(\mathbf{x}_t|y_t, \lambda)$ is the multiplication of emission probabilities and $p(y_{1:T}|\lambda) = \prod_t p(y_t|y_{t-1}, \lambda)$ is the multiplication of transition probabilities

- Hard to enumerate all state sequences $y_{1:T}$
- Almost impossible to find the result using this way
- Instead, we use an iterative method (dynamic programming) called the forward algorithm

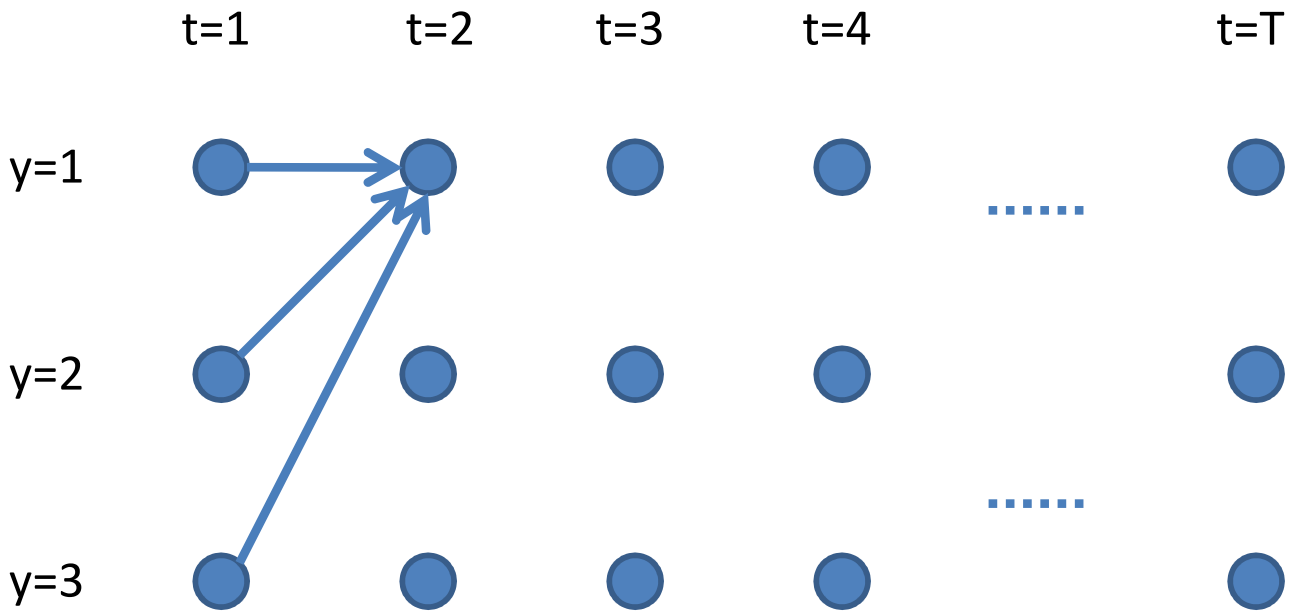
Forward algorithm

- Define partial probabilities $\alpha_t(j) = p(\mathbf{x}_{1:t}, y_t = j|\lambda)$, note that $\sum_j \alpha_T(j)$ is the desired probability of observation $p(\mathbf{x}_{1:T}|\lambda)$
- Iteratively update α's in time $\alpha_t(j) = \sum_{i=1}^{M} \alpha_{t-1}(i) a_{ij} p(\mathbf{x}_t|j)$
- We can visualize this on a trellis

The algorithm

1. Initialize $\alpha_1(j) = \pi_j p(\mathbf{x}_1|j)$ for $j = 1, \ldots, M$
2. Update $\alpha_t(j) = \sum_{i=1}^{M} \alpha_{t-1}(i) a_{ij} p(\mathbf{x}_t|j)$ for $j = 1, \ldots, M$
3. Terminate: $p(\mathbf{x}_{1:T}|\lambda) = \sum_{j=1}^{M} \alpha_T(j)$

Forward algorithm on a trellis



$$\alpha_t(j) = \sum_{i=1}^{M} \alpha_{t-1}(i) a_{ij} p(x_t \mid j)$$

$$\alpha_2(1) = \left[\alpha_1(1)a_{11} + \alpha_1(2)a_{21} + \alpha_1(2)a_{31}\right] p(x_2 \mid 1)$$

Problem 2: Alignment/decoding/inference

- We would like to find optimal $y^*_{1:T} = \arg\max_{y_{1:T}} p(y_{1:T}|x_{1:T}, \lambda)$
- Use another dynamic programming algorithm called Viterbi algorithm
- Simply replace the sum in the forward algorithm with a max operation
- Also, hold a backpointer at each state to remember the maximum scoring path

Viterbi algorithm

- Define partial maximal probabilities
  $V_t(j) = \max_{y_{1:t-1}} p(\mathbf{x}_{1:t}, y_{1:t-1}, y_t = j|\lambda)$
- Iteratively update V's in time $V_t(j) = \max_{i=1}^{M} V_{t-1}(i) a_{ij} p(\mathbf{x}_t|j)$
- We can visualize this on a trellis (same picture as forward algorithm, replace sum with max)

The algorithm

1. Initialize $V_1(j) = \pi_j p(\mathbf{x}_1|j)$
2. Update
   - $V_t(j) = \max_{i=1}^{M} V_{t-1}(i) a_{ij} p(\mathbf{x}_t|j)$
   - Hold a backpointer $\psi_t(j) = \arg\max_i V_{t-1}(i) a_{ij} p(\mathbf{x}_t|j)$
3. Terminate
   - Perform the update at step T
   - Trace back the path from $\psi_T(y_T^*)$ where y_T^* is the maximum likely end state

Problem 3: Training I

- Given $(\mathbf{x}_{1:T_i}^{(i)})_{i=1}^{N}$, maximum likelihood training requires finding

$$\hat{\lambda} = \arg\max_{\lambda} \sum_{i=1}^{N} \log \left(p(\mathbf{x}_{1:T_i}^{(i)} | \lambda) \right)$$

- For simplicity, assume single sequence $\mathbf{x}_{1:T}$ for training, generalization to multiple sequences is trivial
- Direct maximization is not easy, use Expectation Maximization (EM) algorithm
- Latent data is the label sequence $(y_{1:T})$
  1. Start with an initial λ^{old}
  2. Expectation step (E-step): Compute posterior probability of the latent variables $p(y_{1:T} | \mathbf{x}_{1:T}, \lambda^{old})$

Problem 3: Training II

3. Maximization step (M-step): Find λ that maximizes the auxiliary function which is the expected log-likelihood of the complete data under the posterior found in the E-step

$$Q(\lambda, \lambda^{old}) = \sum_{y'_{1:T}} p(y'_{1:T}|\mathbf{x}_{1:T}, \lambda^{old}) \log p(\mathbf{x}_{1:T}, y'_{1:T}|\lambda)$$

- Initialization is very important and it can be more art than science
- In case of HMMs, EM algorithm is called the forward-backward algorithm
- Need to propagate forward and backward variables for the E-step

Backward Algorithm

Similar to forward algorithm, we need a backward algorithm where we define

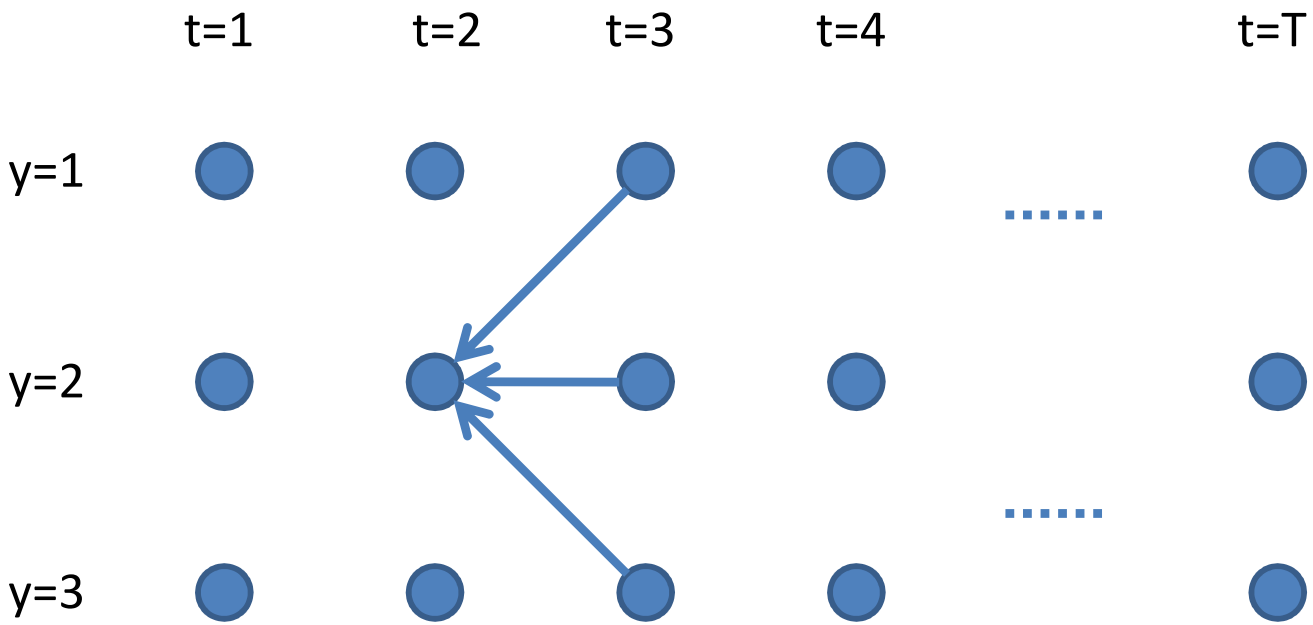
$$\beta_t(i) = p(x_{t+1:T}|y_t = i, \lambda)$$

The update is from final time to the beginning time and the update rule becomes (follows from probabilities and graphical model of HMMs)

$$\beta_t(i) = \sum_{j=1}^{M} a_{ij} p(x_{t+1}|j)\beta_{t+1}(j), \quad \forall i = 1, \ldots, M$$

We can visualize this on a trellis

Backward algorithm on a trellis



$$\beta_t(i) = \sum_{j=1}^{M} \beta_{t+1}(j) a_{ij} p(x_{t+1} \mid j)$$

$$\beta_2(2) = \beta_3(1) a_{21} p(x_3 \mid 1) + \beta_3(2) a_{22} p(x_3 \mid 2) + \beta_3(3) a_{23} p(x_3 \mid 3)$$

Posterior probabilities I

- For the EM algorithm, we need to sum over exponentially many $\sum_{y'_{1:T}} p(y'_{1:T}|\mathbf{x}_{1:T}, \lambda^{old}) \log p(\mathbf{x}_{1:T}, y'_{1:T}|\lambda)$, but both terms in the sum can be factorized due to the graphical model of the HMM
- Using the forward-backward algorithm we obtain local posteriors:

$$\xi_t(i,j) = p(y_{t-1} = i, y_t = j|\mathbf{x}_{1:T}, \lambda^{old})$$

and

$$\gamma_t(j) = p(y_t = j|\mathbf{x}_{1:T}, \lambda^{old})$$

then it is easy to maximize the auxiliary function $Q(\lambda, \lambda^{old})$ which factorizes as follows [Bishop, 2006]

$$\sum_{j=1}^{M} \gamma_1(j) \log \pi_j + \sum_{t=2}^{T} \sum_{i,j=1}^{M} \xi_t(i,j) \log a_{ij} + \sum_{t=1}^{T} \sum_{j=1}^{M} \gamma_t(j) \log p(\mathbf{x}_t|j)$$

Posterior probabilities II

- Once we can obtain the posterior probabilities using previous iteration's parameters (λ^{old}), we can update the emission parameters using $\gamma_t(j)$ and transition parameters using $\xi_t(i,j)$
- We can obtain these two sets of variables using forward-backward probabilities
- After performing one forward and one backward pass, we have all α and β parameters

Then,

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{p(x_{1:T}|\lambda)}$$

and

$$\xi_t(i,j) = \frac{\alpha_{t-1}(i)a_{ij}p(x_t|j)\beta_t(j)}{p(x_{1:T}|\lambda)}$$

Updating the parameters I

- Assume there is only a single training sequence ($x_{1:T}$)
- After $\gamma_t(j)$ and $\xi_t(i,j)$ parameters are found, the parameter estimation becomes like a weighted counting procedure
- For transition parameters $\hat{a}_{ij} = \frac{\sum_{t=2}^{T} \xi_t(i,j)}{\sum_{t=2}^{T} \sum_{j=1}^{M} \xi_t(i,j)}$
- For emission parameters:
  - Discrete case: $p(x|j) := \frac{\sum_{t=1}^{T} \gamma_t(j) \delta_{x_t,x}}{\sum_{t=1}^{T} \gamma_t(j)}$

Updating the parameters II

- Gaussian case: The means and variances are updated using weighted sample averages where weights are $\gamma_t(j)$ for each state j
- So, when there is one training sequence, mean update is as follows

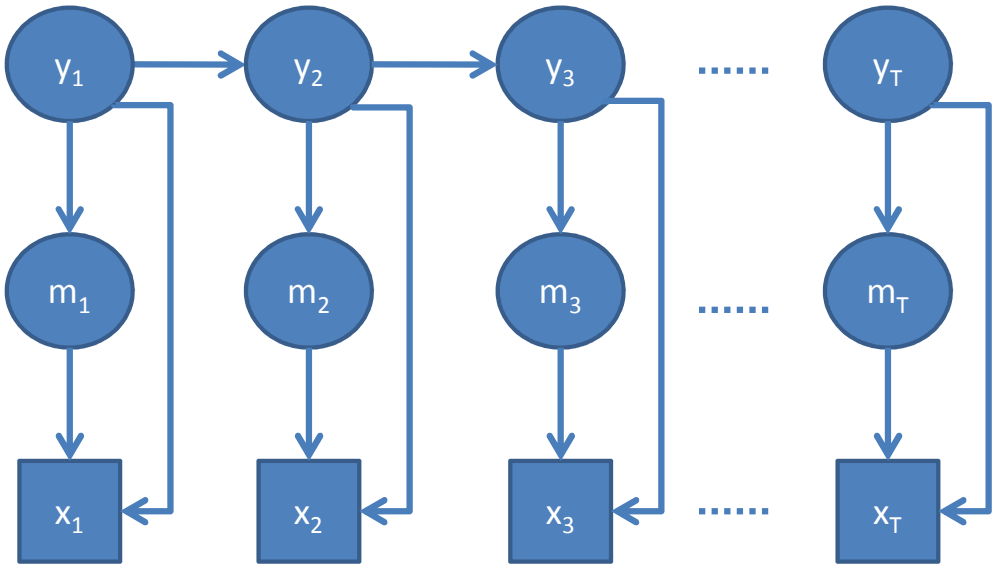
$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)\mathbf{x}_t}{\sum_{t=1}^{T} \gamma_t(j)}$$

- And the covariance update is similarly

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} \gamma_t(j)\mathbf{x}_t\mathbf{x}_t^T}{\sum_{t=1}^{T} \gamma_t(j)} - \hat{\boldsymbol{\mu}}_j\hat{\boldsymbol{\mu}}_j^T$$

Gaussian mixture observations I

- Gaussian mixture model (GMM) distributions are used a lot in HMMs (e.g. for speech recognition)
- The emission probabilities are represented as a GMM



- $p(\mathbf{x}|y) = \sum_m p(\mathbf{x}|m, y)p(m|y) = \sum_m \mathcal{N}(\mathbf{x}; \mu_{y,m}, \Sigma_{y,m})c_{y,m}$

Gaussian mixture observations II

- The emission parameter updates will depend on mixture posteriors

$$\begin{aligned}\gamma_t(j,m) &= p(y_t = j, m_t = m|\mathbf{x}_{1:T}) \\ &= p(y_t = j|\mathbf{x}_{1:T})p(m_t = m|y_t = j, \mathbf{x}_{1:T}) \\ &= \gamma_t(j)\frac{c_{j,m}p(\mathbf{x}_t|j,m)}{\sum_{m'} c_{j,m'}p(\mathbf{x}_t|j,m')}\end{aligned}$$

- Then, when there is a single sequence for training, mean updates will be as follows:

$$\hat{\boldsymbol{\mu}}_{j,m} = \frac{\sum_{t=1}^{T}\gamma_t(j,m)\mathbf{x}_t}{\sum_{t=1}^{T}\gamma_t(j,m)}$$

- (co)variances can be updated in a similar fashion

Conditional likelihood for an HMM I

- The form of the conditional likelihood is

$$\begin{aligned} p(y_{1:T}|\mathbf{x}_{1:T};\lambda) &= \frac{p(\mathbf{x}_{1:T}|y_{1:T})p(y_{1:T})}{\sum_{y'_{1:T}} p(\mathbf{x}_{1:T}|y'_{1:T})p(y_{1:T})} \\ &= \frac{1}{Z(\mathbf{x}_{1:T};\lambda)} \prod_{t=1}^{T} p(\mathbf{x}_t|y_t) \prod_{t=1}^{T} p(y_t|y_{t-1}) \end{aligned} \quad (8)$$

- If the emission distributions $p(\mathbf{x}_t|y_t)$ are from the exponential family

$$p(\mathbf{x}_t|y_t) = h(\mathbf{x}_t) \exp\left\{\boldsymbol{\theta}_{y_t}^T \mathbf{t}(\mathbf{x}_t) - A(\boldsymbol{\theta}_{y_t})\right\}$$

where $\boldsymbol{\theta}_y$ denotes the set of (transformed) parameters of the conditional pdf for state y

Conditional likelihood for an HMM II

- Then, we get (hiding λ dependence, $h(\mathbf{x})$ cancels out)

$$p(y_{1:T}|\mathbf{x}_{1:T}) = \frac{1}{Z(\mathbf{x}_{1:T})} \exp\left\{\sum_{t=1}^{T} \boldsymbol{\theta}_{y_t}^T \mathbf{t}(\mathbf{x}_t) - A(\boldsymbol{\theta}_{y_t}) + \log a_{y_{t-1},y_t}\right\}$$

- Clearly this has a log-linear form:

$$p(y_{1:T}|\mathbf{x}_{1:T}) = \frac{1}{Z(\mathbf{x}_{1:T})} \exp\left\{\sum_{j=1}^{N_f} w_j \sum_{t=1}^{T} f_j(y_{t-1}, y_t, \mathbf{x}_t, t)\right\}$$

where

$$\mathbf{f}(y_{t-1}, y_t, \mathbf{x}_t, t) = \begin{pmatrix} \phi(\mathbf{x}_t) \otimes \mathbf{e}_{y_t} \\ \mathbf{e}_{y_{t-1}} \otimes \mathbf{e}_{y_t} \end{pmatrix}$$

and $\phi(\mathbf{x}) = \left[\mathbf{t}(\mathbf{x})^T, 1\right]^T$ as before and $\otimes$ denotes the Kronecker product

Other related models

- Hidden Semi-Markov models: assigns a single label to a segment instead of labeling each observation separately, enables explicit duration model
- Factorial HMM: multiple states explain the observation at the same time
- Multi-stream HMM: the observations are handled in separate streams each of which are independently modeled by a different emission model
- Coupled HMM: two state sequences generate two streams, they interact through their states

Outline

Generative vs Discriminative - HMM vs MEMM I

- Generative models (e.g. FLD) and logistic regression are generative-discriminative pairs
- MEMM is an attempt to get a discriminative version of HMM
- Depends on writing the conditional likelihood as

$$p(y_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p(y_t|y_{t-1}, \mathbf{x}_t)$$

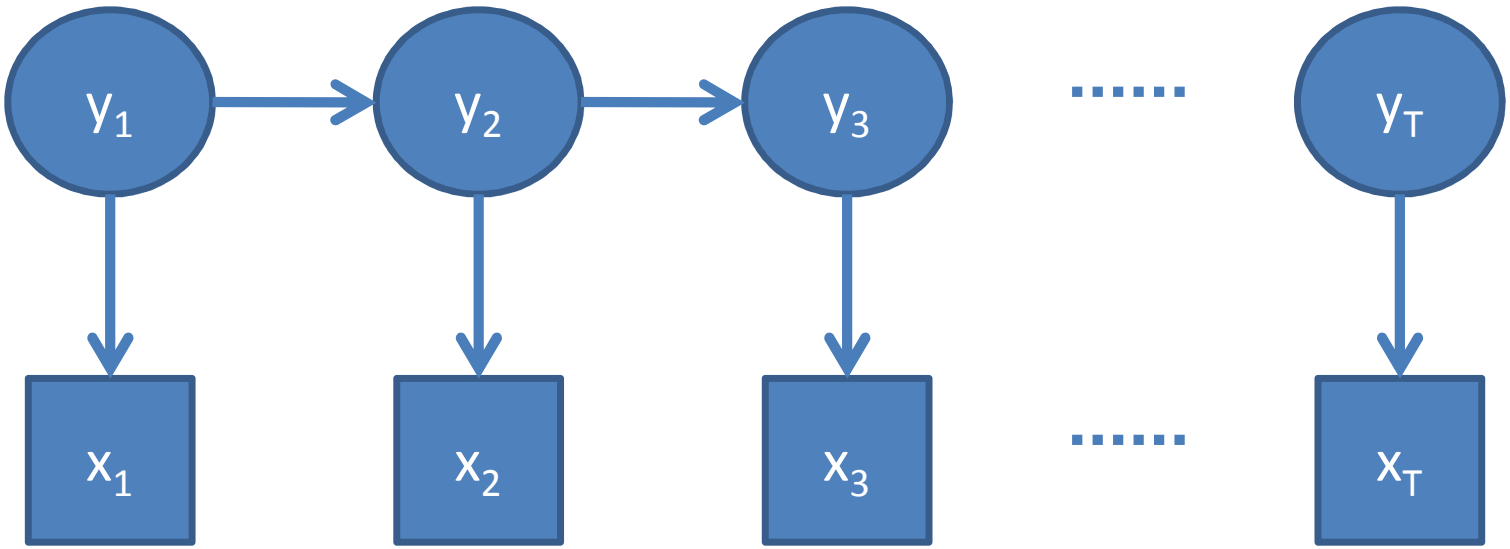
- This may not be a good assumption
- This turns out not to be a good way to obtain a discriminative model from HMM
- Leads to a problem called "label bias"

Generative vs Discriminative - HMM vs MEMM II

- Solution: Directly model $p(y_{1:T}|\mathbf{x}_{1:T})$ (conditional random fields)
  - without assuming probabilistic dependencies among y_t, y_{t-1} and $\mathbf{x}_t$
  - Directly use a log-linear model
  - But use only local features in the log-linear model that depend on y_t and y_{t-1} only! (to enable dynamic programming)

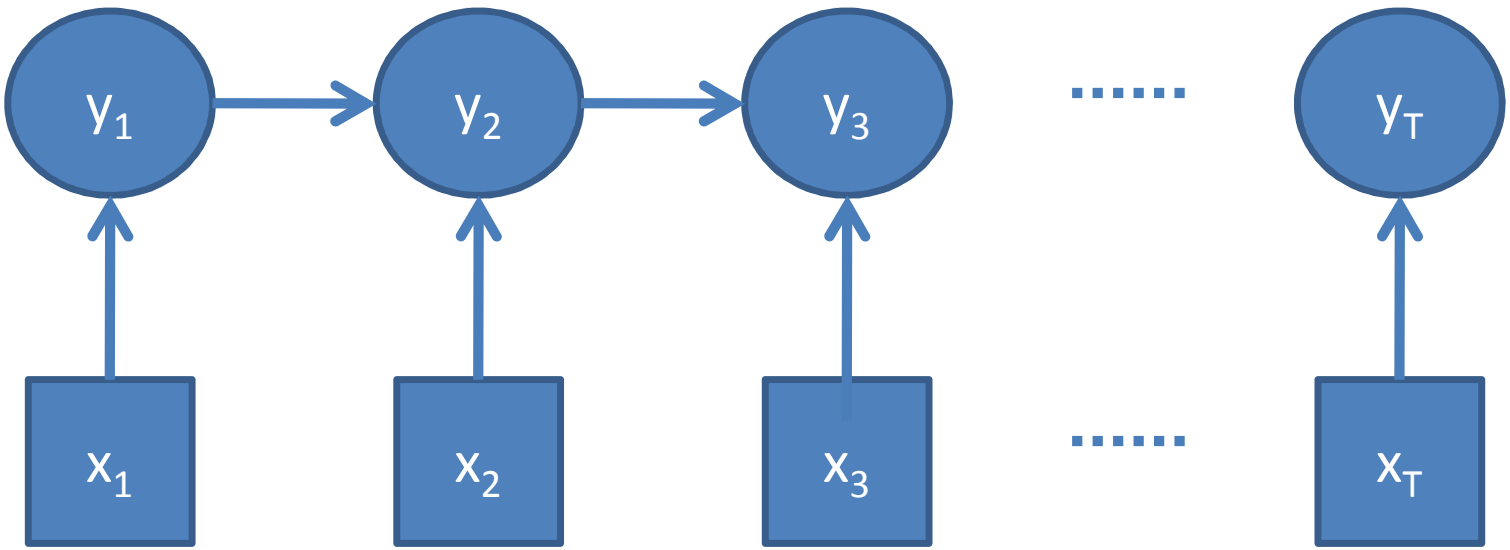
Graphical models I

HMM graphical model:



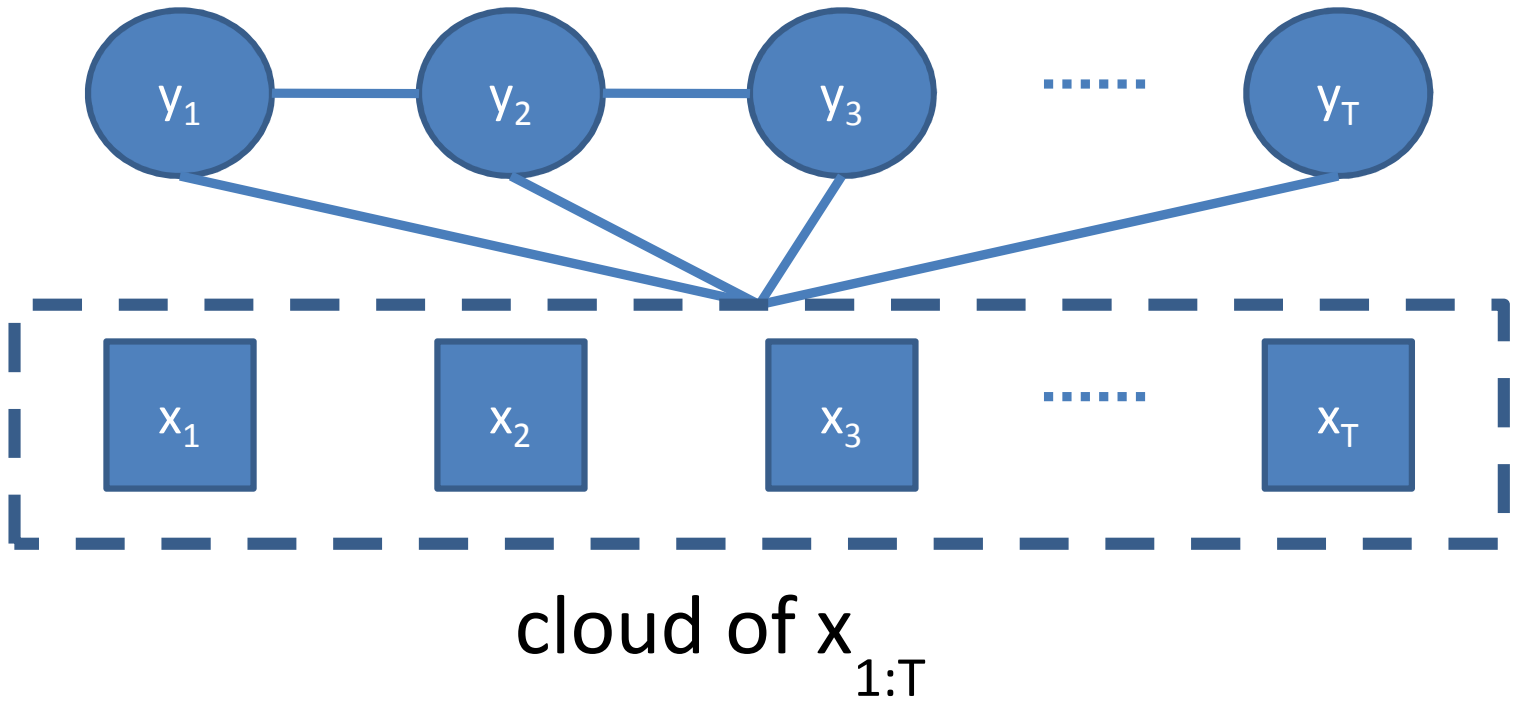
Graphical models II

MEMM graphical model:



Graphical models III

CRF graphical model:



Remembering problem setup I

- Given a sequence of features $\mathbf{x}_{1:T}$, find appropriate labels $y_{1:T}$ where each $y_t \in \mathcal{Y}$, we can assume wlog that $\mathcal{Y} = [M]$, a finite set
- This is a hard problem and the number of possible $y_{1:T}$ is too high, namely M^T
- We may need additional assumptions on output labels y_t, such as being Markov

Outline

Conditional Random Fields I

In CRF, we model the conditional probability of labels wrt observations as follows:

$$p(y_{1:T}|\mathbf{x}_{1:T}) = \frac{1}{Z(\mathbf{x}_{1:T}, \mathbf{w})} \exp\left\{\sum_{j=1}^{N_f} w_j F_j(\mathbf{x}_{1:T}, y_{1:T})\right\}$$

Key thing is to assume that the global feature functions $F_j(y_{1:T}, \mathbf{x}_{1:T})$ should be able to be written as a sum of local features

$$F_j(\mathbf{x}_{1:T}, y_{1:T}) = \sum_{t=1}^{T} f_j(y_{t-1}, y_t, \mathbf{x}_{1:T}, t)$$

- This assumption is necessary to be able to use dynamic programming algorithms in calculations

Conditional Random Fields II

- Each local feature may depend on all $\mathbf{x}_{1:T}$ since they are given to us
- This assumption yields a Markovian label sequence
- Local feature functions can specialize in depending on any combination of their inputs, they do not have to depend on all of their arguments
- For example, a transition feature will depend only on y_t and y_{t-1}

Problems of interest

To be able to solve inference problems in CRFs, we need to be able to compute the most likely label sequence:

$$y^*_{1:T} = \arg\max_{y'_{1:T}} p(y'_{1:T}|\mathbf{x}_{1:T};\mathbf{w})$$

and for the learning problem, we need to calculate the partition function

$$Z(\mathbf{x}_{1:T},\mathbf{w}) = \sum_{y'_{1:T}} \exp\left\{\sum_{j=1}^{N_f} w_j F_j(\mathbf{x}_{1:T}, y'_{1:T})\right\}$$

Note that direct calculation of these two quantities is highly expensive due to exponential amount of all possible $y_{1:T}$ that is needed to be considered

Finding the most likely labeling - Viterbi algorithm I

It is easy to show that:

$$y^*_{1:T} = \arg\max_{y'_{1:T}} \sum_j w_j F_j(\mathbf{x}_{1:T}, y'_{1:T})$$

and after expanding the features

$$y^*_{1:T} = \arg\max_{y'_{1:T}} \sum_j w_j \sum_{t=1}^{T} f_j(y'_{t-1}, y'_t, \mathbf{x}_{1:T}, t)$$

Let $g_t(y_{t-1}, y_t) = \sum_j w_j f_j(y_{t-1}, y_t, \mathbf{x}_{1:T}, t)$ to simplify notation. Define partial maximums:

$$V(y, t) = \max_{y'_{1:t-1}} \left(\sum_{\tau=1}^{t-1} g_\tau(y'_{\tau-1}, y'_\tau) + g_t(y'_{t-1}, y) \right)$$

Finding the most likely labeling - Viterbi algorithm II

- Clearly, this leads to a recursion:

$$V(y,t) = \max_{y'} \left(V(y', t-1) + g_t(y', y) \right)$$

- Similar to HMMs, we need to hold a backpointer to the maximizer label (state) after each time step
- We can view this procedure in a trellis
- In the end we can trace back from $V(y_T^*, T)$ to obtain the most likely label sequence $y_{1:T}^*$.

Forward-backward algorithm for CRFs I

- Remember that:

$$Z(\mathbf{x}_{1:T}, \mathbf{w}) = \sum_{y'_{1:T}} \exp \left\{ \sum_{j=1}^{N_f} w_j F_j(\mathbf{x}_{1:T}, y'_{1:T}) \right\}$$

- We need to sum over exponentially many sequence labelings which is impractical
- Similar to forward-backward algorithm in HMMs, we can perform a dynamic programming algorithm like that to compute Z

Forward-backward algorithm for CRFs II

- We need to use the local features summed over time to do that

$$Z(\mathbf{x}_{1:T}, \mathbf{w}) = \sum_{y'_{1:T}} \exp\left\{ \sum_{\tau=1}^{T} \sum_{j=1}^{N_f} w_j f_j(y'_{\tau-1}, y'_{\tau}, \mathbf{x}_{1:T}, \tau) \right\}$$

$$Z(\mathbf{x}_{1:T}, \mathbf{w}) = \sum_{y'_{1:T}} \prod_{\tau=1}^{T} G_\tau(y'_{\tau-1}, y'_{\tau})$$

where we define $G_t(y_1, y_2) = \exp g_t(y_1, y_2)$, and $g_t(y_1, y_2) = \sum_j w_j f_j(y_1, y_2, \mathbf{x}_{1:T}, t)$ as defined earlier

- and define partial sums up to time t

$$\alpha(y, t) = \sum_{y'_{1:t-1}} \left(\prod_{\tau=1}^{t-1} G_\tau(y'_{\tau-1}, y'_{\tau}) G_t(y'_{t-1}, y) \right)$$

Forward-backward algorithm for CRFs III

- We can update $\alpha(y,t)$ by the following forward recursion

$$\alpha(y,t) = \sum_{y'} \alpha(y',t-1) G_t(y',y)$$

- Similarly we define backward partial sums

$$\beta(y,t) = \sum_{y'_{t+1:T}} \left(G_{t+1}(y,y'_{t+1}) \prod_{\tau=t+1}^{T} G_{\tau+1}(y'_\tau, y'_{\tau+1}) \right)$$

- which can be updated with the backward recursion

$$\beta(y,t) = \sum_{y'} \beta(y',t+1) G_{t+1}(y,y')$$

Forward-backward algorithm for CRFs IV

- Clearly

$$Z(\mathbf{x}_{1:T}, \mathbf{w}) = \sum_{y'} \alpha(y', T) = \sum_{y'} \beta(y', 1)$$

- Note that if we use start and stop labels/states, we do not need the sums above and we get

$$Z(\mathbf{x}_{1:T}, \mathbf{w}) = \alpha(stop, T+1) = \beta(start, 0)$$

- Besides, we can calculate the following marginal posterior probabilities

$$p(y_t|\mathbf{x}_{1:T}) = \frac{\alpha(y_t, t)\beta(y_t, t)}{Z(\mathbf{x}_{1:T}, \mathbf{w})}$$

$$p(y_{t-1}, y_t|\mathbf{x}_{1:T}) = \frac{\alpha(y_{t-1}, t-1)G_t(y_{t-1}, y_t)\beta(y_t, t)}{Z(\mathbf{x}_{1:T}, \mathbf{w})}$$

Outline

CRF Training I

- We have seen that when $(\mathbf{x}_{1:T}, y_{1:T})$ were given, ML training for HMMs turned into simple counting
- For CRFs, even in that scenario, training is not that simple
- Consider conditional log-likelihood (CLL) for a single training sequence

$$\log p(y_{1:T}|\mathbf{x}_{1:T}; \mathbf{w}) = \mathbf{w}^T \mathbf{F}(\mathbf{x}_{1:T}, y_{1:T}) - \log Z(\mathbf{x}_{1:T}, \mathbf{w})$$

  where $\mathbf{F}$ denotes the vector of all N_f features
- For multiple training sequences, we need to sum the individual CLL's up

CRF Training II

- Gradient of the CLL for a single sequence is

$$\mathbf{F}(\mathbf{x}_{1:T}, y_{1:T}) - \sum_{y'_{1:T}} \mathbf{F}(\mathbf{x}_{1:T}, y'_{1:T}) p(y'_{1:T} | \mathbf{x}_{1:T})$$

$$\mathbf{F}(\mathbf{x}_{1:T}, y_{1:T}) - E_{y'_{1:T} \sim p(y'_{1:T} | \mathbf{x}_{1:T})} \left[\mathbf{F}(\mathbf{x}_{1:T}, y'_{1:T}) \right]$$

- When we obtain the maximizing $\mathbf{w}$, the gradient must be zero which corresponds to making the training data feature values to be the same as the expected values under the trained model

CRF Training III

- The expectation of a feature can be computed using the forward and backward variables as follows:

$$
\begin{aligned}
& E_{y'_{1:T} \sim p(y'_{1:T}|\mathbf{x}_{1:T})} \left[F_j(\mathbf{x}_{1:T}, y'_{1:T}) \right] \\
= \quad & E_{y'_{1:T} \sim p(y'_{1:T}|\mathbf{x}_{1:T})} \left[\sum_{t=1}^{T} f_j(y'_{t-1}, y'_t, \mathbf{x}_{1:T}, t) \right] \\
= \quad & \sum_{t=1}^{T} E_{y'_{t-1}, y'_t} [f_j(y'_{t-1}, y'_t, \mathbf{x}_{1:T}, t)] \\
= \quad & \frac{1}{Z} \sum_{t=1}^{T} \sum_{y_1, y_2} \alpha(t-1, y_1) f_j(y_1, y_2, \mathbf{x}_{1:T}, t) G_t(y_1, y_2) \beta(t, y_2)
\end{aligned}
$$

where $G_t(y_1, y_2) = \exp\{\sum_{j'} w_{j'} f_{j'}(y_1, y_2, \mathbf{x}_{1:T}, t)\}$

CRF Training IV

- The exact calculation of the expected value can be somewhat computationally complex (requiring forward-backward iterations)
- It is possible to approximate the gradient calculation by
  - Considering only the best competitor's feature function instead of considering the average over all (expected value)
  - Performing Gibbs sampling to evaluate the expected value

Relation to belief propagation

- The Viterbi (max-product) and forward-backward (sum-product) algorithms are special cases of belief propagation in graphical models
- Belief propagation generalizes these algorithms to tree-structured graphs
- For loopy graphs, belief propagation does not converge however it can be used in practice (called loopy BP)
- In graphs with a few clusters of loopy parts, junction-tree algorithm can be used
- However, in this talk, we will not focus on BP or general graphical models

RER formulation

- Regularized empirical risk (conditional log-likelihood plus a regularizing penalty term) is optimized in the case of CRFs as well
- Original formulation does not include any regularization term, however it can be added to the objective function for better results
- Dropping dependence on $1:T$, given training data (x^i, y^i) for $i \in [N]$ where each (x^i, y^i) is a training sequence, we get the following RER function to minimize

$$\sum_{i=1}^{N} \left(-\mathbf{w}^T \mathbf{F}(x^i, y^i) + \log Z(x^i, \mathbf{w}) \right) + \Omega(\mathbf{w})$$

- and each entry of the gradient vector is

$$\sum_{i=1}^{N} \left(-F_j(x^i, y^i) + \sum_{y'} p(y'|x^i; \mathbf{w}) F_j(x^i, y') \right) + \frac{\partial \Omega}{\partial w_j}$$

Optimization methods for training/learning

- Almost all methods require computation of the gradient whose exact computation requires forward-backward iterations, but this can be approximated through methods discussed above
- The list of possible optimization methods are:
  1. Iterative scaling (old one, slow)
  2. Conjugate gradient method
  3. L-BFGS (a Quasi Newton method)
  4. Stochastic gradient method: update parameters by moving in the direction of the gradient of one sequence at a time (easy and fast converging)

Stochastic Gradient Updates

- Stochastic gradient update for a single weight w_j is as follows:

$$w_j^{(n+1)} := w_j^{(n)} + \eta^{(n)} \left(F_j(\mathbf{x}^i, y^i) - \sum_{y'} p(y'|\mathbf{x}^i, \mathbf{w}^{(n)}) F_j(\mathbf{x}^i, y') - \frac{\partial \Omega}{\partial w_j} \right)$$

  where $\eta^{(n)}$ is a iteration-dependent learning rate parameter (step size in the gradient direction)
- Only one training sample $(\mathbf{x}^i, y^i)$ is used at a time and multiple epochs through the data are performed
- Usually $\eta^{(n)}$ is chosen to decrease inversely proportional to the iteration number n

For more information on CRFs

- I have used the following papers/documents for CRF's, it is beneficial to explore them all
- Papers and technical reports [Lafferty et al., 2001, Sutton and McCallum, 2006, Elkan, 2008, Gupta, 2005, Memisevic, 2006]
- See video lecture by Prof. Charles Elkan on videolectures.net

Outline

Structured SVM for sequence labeling I

- The inference in CRF's (finding the most likely labeling) is given by

$$y^*_{1:T} = \arg\max_{y'_{1:T}} \mathbf{w}^T \mathbf{F}(\mathbf{x}_{1:T}, y'_{1:T})$$

- Note that there is no need for the normalizing partition function ($Z(\mathbf{x}_{1:T}, \mathbf{w})$) for inference (but it is required for training)
- Idea: we can use large-margin criterion to learn the weight vector $\mathbf{w}$
- Note that the sequence labeling problem is just a multi-class classification problem with exponentially many classes
- So, formulation is very similar to multi-class SVM formulation
- Structured SVM first proposed in [Altun et al., 2003, Taskar et al., 2003]

Structured SVM training I

- Given training data $\left(\mathbf{x}^i_{1:T}, y^i_{1:T}\right)_{i=1}^N$
- We drop dependence on $\{1:T\}$ and denote the training data by $\left(\mathbf{x}^i, y^i\right)_{i=1}^N$
- The primal problem with margin scaling is

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i$$

subject to

$$\mathbf{w}^T\mathbf{F}(\mathbf{x}^i, y^i) - \mathbf{w}^T\mathbf{F}(\mathbf{x}^i, y') > \Delta(y^i, y') - \xi_i, \quad \forall y' \in \mathcal{Y}, \forall i \in [N]$$

Structured SVM training II

- The RER formulation of the same problem is:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\left(-\mathbf{w}^T\mathbf{F}(\mathbf{x}^i, y^i) + \max_{y'}\left(\mathbf{w}^T\mathbf{F}(\mathbf{x}^i, y') + \Delta(y^i, y')\right)\right)$$

- If the label-loss $\Delta(y^i_{1:T}, y'_{1:T})$ factorizes over time (for example Hamming loss over the labels) and the features are locally decomposable (as in CRF), then we can run Viterbi algorithm for finding the most offending label sequence.

Different varieties of structured SVMs

- max(.) operator can be replaced with softmax to obtain a differentiable objective (as discussed in the multi-class section). This would enable using primal optimizers, also will require forward-backward iterations (the problem becomes very similar to CRF training as in multi-class case)
- Different label-loss functions can be used. However, Hamming loss is the only one that would enable fast inference (since it decomposes over the individual terms)
- Different slack scalings can be used (in the constrained version of the problem): namely (a) margin scaling and (b) slack rescaling (as discussed in the multi-class case)
- mn-slack, n-slack, 1-slack versions can be proposed (similar to multi-class case). Typically n-slack formulation is used

Training the structured SVM for sequence labeling

- These methods can be used for training
  1. Taskar's method (decomposition of the structured problem into parts) [Taskar et al., 2003]
  2. Cutting plane method (n-slack, dual) [Tsochantaridis et al., 2005]
  3. Cutting plane method (1-slack, faster) [Joachims et al., 2009]
  4. Stochastic gradient with SMO-like updates (dual) [Bordes et al., 2007]
  5. Stochastic gradient for CRF/SVM-softmax (primal) [Vishvanathan et al., 2006]
  6. Stochastic subgradient method (primal) [Ratliff et al., 2007]

Cutting-plane method

- Directly solving structured SVM requires exponentially many constraints or number of dual variables, namely $\prod_{i=1}^{N} |\mathcal{Y}|^{T_i}$
- Cutting plane idea adds most violated constraints (one for each sample) at each step and solves the problem at each iteration with only those constraints
- Constraint set grows after each iteration, initialize solution from the earlier step
- It is proven that the algorithm converges in time proportional to $1/\epsilon^2$ where ϵ is the desired accuracy of the solution [Tsochantaridis et al., 2005]
- Typically the dual QP problem is solved at each step
- The 1-slack formulation requires much less number of constraints (or dual variables) [Joachims et al., 2009]

Stochastic gradient method for structured SVM I

- Similar to CRF training, SVM-softmax can be trained easily with stochastic gradient leading to the following update for the jth parameter after considering training instance i in iteration n

$$\begin{aligned} w_j^{(n+1)} &= w_j^{(n)} + \eta^{(n)} g_j^{(n)} \\ g_j^{(n)} &= F_j(\mathbf{x}^i, y^i) - \sum_{y'} \gamma_i(\mathbf{w}^{(n)}, y') F_j(\mathbf{x}^i, y') - \lambda w_j \\ \gamma_i(\mathbf{w}, y') &= \frac{exp\{\mathbf{w}^T F(\mathbf{x}^i, y') + \Delta(y^i, y')\}}{\sum_{y''} \exp\{\mathbf{w}^T F(\mathbf{x}^i, y'') + \Delta(y^i, y'')\}} \end{aligned}$$

- Note that, this update requires forward-backward iterations to compute the denominator of the γ_i term

Stochastic gradient method for structured SVM II

- The subgradient version (hard-max) of the stochastic gradient update replaces the negative gradient term with the following easier computed one:

$$g_j^{(n)} = F_j(\mathbf{x}^i, y^i) - F_j(\mathbf{x}^i, \bar{y}^i) - \lambda w_j$$

where $\bar{y}^i$ is the most offending labeling for instance i, that is

$$\bar{y}^i = \arg\max_{y'} \left(\mathbf{w}^T F(\mathbf{x}^i, y') + \Delta(y^i, y') \right)$$

- Note that the stochastic subgradient update (in general) may not converge to the true solution, but can be shown to get close to the solution [Ratliff et al., 2007]

Toolkits

- Hidden markov models
  - HTK by Cambridge (Young et.al.)
  - Sphinx, Julius
  - Matlab statistics toolbox implements discrete HMMs
- Condition random fields
  - CRF++ (for NL problems)
  - CRFSGD
  - Mallet
- Structured SVM
  - Svm-struct (includes SVM for sequence labeling) by Joachims

THANK YOU!

THANK YOU!
Questions?

References I

Y Altun, I Tsochantaridis, and T Hoffman. Hidden Markov support vector machines. In *International Conference on Machine Learning*, 2003.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

A Bordes, L Bottou, P Gallinari, and J Weston. Solving multiclass support vector machines with LARANK. In *International Conference on Machine Learning*, 2007.

Leon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*. LNAI 3176, 2004.

Olivier Chapelle. Training a Support Vector Machine in the Primal. *Neural Computation*, 19(5):1155–1178, 2007.

References II

K Crammer and Y Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–92, 2001.

Charles Elkan. Log-linear models and conditional random fields, notes for a tutorial at CIKM'08, 2008.

R E Fan, K W Chang, C J Hsieh, X R Wang, and C J Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–74, 2008.

J H Friedman. Regularized discriminant analysis. *J. Amer. Statistical Assoc.*, 84:165–175, 1989.

R Gupta. Conditional random fields. Technical report, IIT Bombay, 2005.

Ulrich Hoffmann. *Bayesian machine learning applied in a brain-computer interface for disabled users*. PhD thesis, EPFL, 2007.

References III

C W Hsu and C J Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13:415–25, 2002.

T Joachims, T Finley, and C N J Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, October 2009.

Daniel Jurafsky and James H Martin. *Speech and language processing (second edition)*. Prentice Hall, New Jersey, 2008.

J Lafferty, A McCallum, and F Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, 2001.

Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on Energy based learning. In G. Bakir, T. Hofman, B. Schlkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.

References IV

Olvi L Mangasarian. Exact 1-Norm Support Vector Machines Via Unconstrained Convex Differentiable Minimization. *Journal of Machine Learning Research*, 7:1517–30, 2006.

R Memisevic. An introduction to structured discriminative learning. Technical report, University of Toronto, 2006.

Lawrence R Rabiner. A tutorial on hidden Markov models and selected application in speech recognition. *Proc. IEEE*, 77(2):257–285, February 1989.

ND Ratliff, JA Bagnell, and MA Zinkevich. (Online) subgradient methods for structured prediction. In *AISTATS*, 2007.

C Sutton and A McCallum. An Introduction to Conditional Random Fields for Relational Learning. In Lise Getoor and Ben Taskar, editors, *Introduction to statistical relational learning*. MIT Press, 2006.

J A K Suykens and J Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, June 1999.

References V

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov Networks. In *NIPS*, 2003.

I Tsochantaridis, T Joachims, T Hoffman, and Y Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–84, December 2005.

SVN Vishvanathan, NN Schraudolph, MW Schmidt, and KP Murphy. accelerated training of conditional random fields with stochastic gradient methods. In *International Conference on Machine Learning*, 2006.

Jieping Ye. Least squares linear discriminant analysis. In *International Conference on Machine Learning*, volume 227, pages 1083–97, 2007.

W S Zheng, J H Lai, and P C Yuen. Perturbation LDA: learning the difference between the class empirical mean and its expectation. *Pattern Recognition*, 42(5):764–779, 2009.