Naive Bayes

- Strong Assumption: All words are independent given the class
- We have a model of label and text: $p(y, x) = q(y) \prod_{i=1}^{|x|} q(w_i|y)$
- To assign a label to new text: $\text{argmax}_{y \in C} q(y) \prod_{i=1}^{|x|} q(w_i|y)$
- Maximum Likelihood Estimation (MLE)
- Parameter to estimate:
  - $q(y) = \theta_y$ for each topic y
  - $q(x|y) = \theta_{xy}$ for each topic y and text x
- Data:
  - $\{(x_1, y_1)\}_{j=1}^{N}$
- Maximum Likelihood Estimation (MLE)
  - $q(y)$ $\frac{\text{count}(y)}{N}$
  - $q(x|y)$ $\prod_{i=1}^{|x|} q(w_i|y)$ $\prod_{i=1}^{|x|} \frac{\text{count}(w_i, y) + \alpha}{\sum_{w \in V} \text{count}(w, y) + \alpha N}$
  - Time/Memory Complexity: $O(|V||C|)$

Logistic Regression

- Given input feature vector: $\phi(x) \in R^d$
- Output: $\sigma(p(y = 1|x, w))$
- Learn weight vector: $w \in R^d$
  - $z = w \cdot \phi(x)$, $z = w \cdot \phi(x) + b$
- Binary Classification (σ = sigmoid)
  - Feature vector: $\phi(x)$
  - $p(y = 1|x, w) = \frac{1}{1+e^{-(w \cdot \phi(x))}}$
  - $y^* = \text{argmax}_{y \in 0,1} p(y|x, w)$
- Multi Class Classification (σ = softmax)
  - Feature vector: $\phi(x, y)$
  - $p(y|x, w) = \frac{e^{(w \cdot \phi(x,y))}}{\sum_{y'} e^{(w \cdot \phi(x,y'))}}$
  - $y^* = \text{argmax}_{y \in C}\ w \cdot \phi(x, y)$
- Cross Entropy Loss: $L(w)$
- update: $w^{t+1} = w^t + \eta \frac{d}{dw} L(w)$
  - $w_j^{t+1} = w_j^t + \eta((\sigma(wx + b) - y)x_j)$
  - η: learning rate

Perceptron

- Begin with zero weight vector
- Visit training examples one by one
- Decision rule: $w \cdot \phi(x) > 0$
- If incorrect:
  - If label is positive: $w \leftarrow w + \phi(x)$
  - If label is negative $w \leftarrow w - \phi(x)$

Definitions

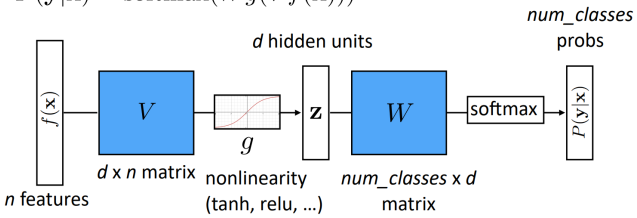
- Generative Models:
  - Modeling joint distribution of data: $p(x, y)$ (Often assume functional form for $p(x|y)$, $p(y)$)
  - Estimating probabilities from the data
  - Pros: Easy to estimate model parameters, Learning weights is easy
  - EX: Naive Bayes, HMM
- Discriminatory Models:
  - Modeling conditional probability: $p(y|x)$, Directly compute: $p(y|x)$, doesn't model x
  - Estimate parameters from the data
  - Pros: Flexible feature representation, You do not model $p(x)$
  - Cons: Requires numerical optimization methods
  - EX: Logistic Regression, Perception, Most Neural Networks
- Features:
  - Mapping an input to a fixed dimensional vector $f : x \rightarrow \phi(x)$
  - Can incorporate linguistic intuitions/domain expertise
  - Can use complex rules
- Sigmoid Function σ: $\sigma(z) = \frac{1}{1+e^{-z}}$, $f(z) \times (1 - f(z))$
- Softmax Function σ: $\sigma(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$
- Log Conditional Likelihood:
  - How much the predicted label distribution differs from the true label distribution
  - $L(w) = \log \prod_{i=1}^{N} p(y^i|x^i, w) = \sum_{i=1}^{N} \log(p(y^i|x^i, w))$
  - $w^* = \text{argmax}_w L(w)$
  - Gradient: $\frac{dL(w)}{dw_j} = \sum_{i=1}^{N} \left[y^i - \sigma(w \cdot \phi(x^i))\right] \phi(x^i)_j$
    - $\sum_{i=1}^{N}$: Summation over whole training set
    - y^i: Correct Label
    - $\sigma(w \cdot \phi(x^i))$: Predicted label
    - $\phi(x^i)_j$: $J-$th input feature value
- Cross Entropy Loss:
  - Minimize discrepancy between true distribution $p(y|x)$ and the predicted distribution $p(y^*|x)$
  - $H(p, q) = -E_p(\log q)$
  - $H(p(y|x), p(y^*|x)) = -\sum_y p(y|x) \log p(y^*|x)$
  - $L(w) = \log \prod_{i=1}^{N} p(y^i|x^i, w) = \sum_{i=1}^{N} \log(p(y^i|x^i, w))$
  - $w^* = \text{argmax}_w L(w)$
- L2 Regularization:
  - Add L2 Regularization term to the likelihood, to push weights towards 0
  - $L(w) = \log \prod_{i=1}^{N} p(y^i|x^i, w) - \frac{\lambda}{2}||w||^2$
  - $\frac{dL(w)}{dw_j} = \sum_{i=1}^{N} \left[y^i - \sigma(w \cdot \phi(x^i))\right] \phi(x^i)_j - \lambda w_j$

Multi-class Perceptron

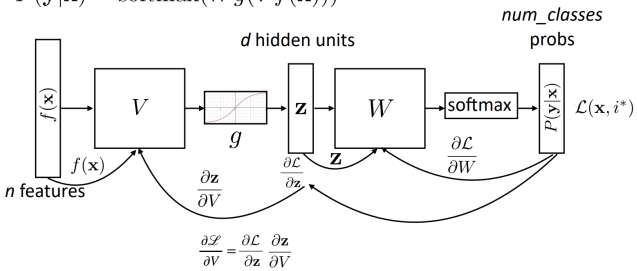
- A weight vector for each class: w_y
- Make prediction: $y^* = \text{argmax}_{y \in C}\ w_y \cdot \phi(x_i)$
- If incorrect: $w = w + \phi(x^i, y^i) - \phi(x^i, y^*)$

Feed-Forward Network (FNN)

$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$

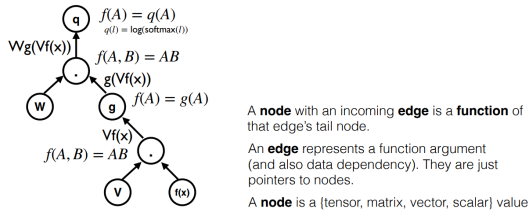


$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$

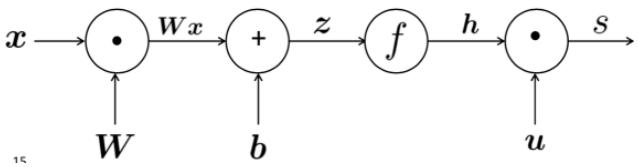


Computational Graphs

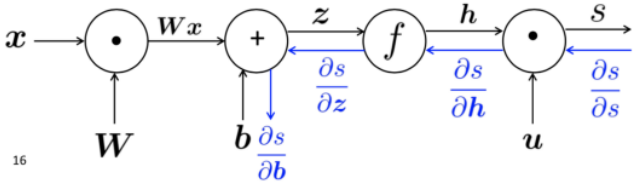
$\log P(\mathbf{y}|\mathbf{x}) = \log(\text{softmax}(Wg(Vf(\mathbf{x}))))$



- Forward Computation:



- Backward Computation:



- Chain Rule: $\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$
- Re-using derivatives computed from the later layers in computing derivations for lower layers, allowing efficient computation of gradient

Hidden Markov Model HMM

- Input: $x = (x_1, ..., x_n)$, Output: $y = (y_1, ..., y_n)$, $p(x_1...x_n, y_1...y_n) =$
- Markov Assumption: Future is conditionally independent of the past, given the present: $p(y_i|y_1, y_2, ..., y_{i-1}) = p(y_i|y_{i-1})$
- Independent Assumption: $p(x_i|x, y) = p(x_i|y_i)$
- States $Y = \{\text{DT}, \text{NNP}, \text{NN}, ...\}$ are the POS tags
- Observations $X = V$ are words
- Transition Distribution $q(y_i|y_{i-1})$ models the tag sequences
- Emission Distribution $e(x_i|y_i)$ models words given their POS
- Learning Maximum Likelihood: Transition q and emission e
  - $p(x_1...x_n, y_1...y_n) = q(\text{STOP}|y_n)\prod_{i=1}^{n} q(y_i|y_{i-1})e(x_i|y_i)$
  - $q(s|u, v) = \frac{c(u,v,s)}{c(u,v)}$
  - $e(x|s) = \frac{c(s \to x)}{c(s)}$
- Inference: Viterbi: $y^* = \underset{y_1 \cdots y_n}{\text{argmax}} p(x_1...x_n, y_1...y_n)$
- Given $\mathcal{K}$ possible tags, for an input sentence of length n, there are $|\mathcal{K}|^n$ number of possible tag sequences

Viterbi

- Compute max score of a sequence of length i ending in tag y_i:
- $\pi(i, y_i) = \max_{y_1 \cdots y_{i-1}} p(x_1...x_i, y_1...y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$
- Store bp (argmax) to reconstruct entire sequence

Beam Search

- The Viterbi algorithm except keep the top k number of hypothesis

Maximum Entropy Markov Models (MEMM)

- $p(y_1...y_n|x_1...x_n) = \prod_{i=1}^{n} p(y_i|y_1...y_{i-1}, x_1...x_n) = \prod_{i=1}^{n} p(y_i|y_{i-1}, x_1...x_n)$
- Learning: Train $p(y_i|y_{i-1}, x_1, ..., x_n)$ as discrete log-linear model
- Scoring: $\frac{exp(w \cdot \phi(x_1...x_n, i, y_{i-1}, y_i))}{\sum_{y'} exp(w \cdot \phi(x_1...x_n, i, y_{i-1}, y'))}$
- $\pi(i, y_i) = \max_{y_{i-1}} p(y_i|y_{i-1}, x_1...x_n)\pi(i-1, y_{i-1})$

Conditional Random Fields (CRF)

- Discriminative models with the following globally-normalized form: $P(y|x) = \frac{1}{Z}\prod_k \exp(\phi_k(x, y))$
- Feature-based potential: $\phi_k(x, y) = w^T f_k(x, y)$
- $\pi(i, y_i) = \max_{y_{i-1}} \phi_t(y_{i-1}, y_i) + \phi_e(y_i, i, x) + \pi(i-1, y_{i-1})$

Expectation Maximization (EM)

- Learning Objective: $L(\theta) = \sum_i \log \sum_{y \in Y} p(x_i, y|\theta)$
- Method for finding: θ_{MLE} $\text{argmax} L(\theta) = \text{argmax} \sum_i \log \sum_{y \in Y} P(x_i, y|\theta)$
- Initially guess model parameters θ, Iterate two steps:
- Expectation Step: Use the current parameters (and observation) to reconstruct hidden states
- Maximization step: Use the hidden states (and observations) to reconstruct parameters
- Perceptron Learning:
  - Separability: Some parameters get the train set perfectly correct
  - Convergence: If the training is separable, perceptron will converge
  - Mistake Bound: The maximum number of mistakes (binary case) related to the margin or degree of separability
- Tanh Function σ: $\sigma(z) = \frac{e^{2z}-1}{e^{2z}+1}$, $1 - f(z)^2$
- ReLU Function σ: $\sigma(z) = \max(0, z)$

Dependency Parsing

- Dependency Parsing: Which words depend on (modify or are arguments of) which other words
- Graph-based parsing: Score edges independently, find the maximum spanning tree
- Pros: Can handle non-projective trees, guaranteed best-scoring parse (best global solution)
- Cons: Relatively slow $O(n^3R)$, n : length of input, R : number of edges, individual arc scoring (local independence assumption)
- Transition-based parsing: Contains stack and buffer; 3 Ops (Shift: Move word buffer $\to$ stack, Left Arc: Top of stack is head of the second word on stack, Right Arc: Second word on stack is head of top word), Greedy algorithm
- Pros: $O(2n)$, n : length of sentence, Can condition on longer tree context
- Cons: Cannot model non-projective trees
- Unlabeled Attachment Score (UAS): Percentage of words that have been assigned the correct head
- Labeled Attachment Score (LAS): Percentage of words that have been assigned correct head & edge label

Constituency Parsing

- Constituent: a unit that can appear in different places (EX: noun verb, etc)
- Phrase structure organizes word into nested constituents

Context-Free Grammer (CFG)

- N : The set of non-terminal symbols; Phrasal Categories: $S, NP, VP, ADJP$, etc; POS;
- Σ : The set of terminal symbols (the words)
- R : The set of rules; Of the form $X \to Y_1Y_2...Y_n$
- S : The start symbol
- Probabilistic (PCFG):
  - Maximum-Likelihood Parameter Estimates: $q_{ML}(\alpha \to \beta) = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)}$

Word Embeddings

- So far, we think of words as "one-hot" vectors
- 'good' and 'great' seem as dissimilar as 'good' and 'the'
- NN learn sophisticated non linear functions of continuous input; our inputs are sparse and discrete
- Ideally we want a vector space where similar words have similar embeddings (great $\approx$ good)

Deep Averaging Network

- Feed-forward NN on average of word embeddings from input for sentiment classification
- "Predator is a masterpiece" $\to c1, c2, c3, c4$, Average: $\sum_{i=1}^{4} \frac{c_i}{4}$

Language Model

- N-Gram Language Model: Distribution of next word is multinomial conditioned on previous $n-1$ words:
  - $p(x_i|x_1...x_{i-1}) = p(x_i|x_{i-n+1}...x_{i-1})$
  - Unigram: $p(x) = \prod_{i=1}^{n} p(x_i)$
  - Bigram: $p(x) = \prod_{i=1}^{n} p(x_i|x_{i-1})$
- Model Quality:
  - Evaluate log likelihood of held-out data $l = \frac{1}{n}\sum_{i=1}^{n} \log p(x_i|x_1, x_2, ..., x_{i-1})$
  - Perplexity: Higher perplexity $\to$ language with high branching factor
  - different between models perplexity and true perplexity of language estimates quality of model
  - Pros: Easy to compute, standardized, nice theoretical interpretations
  - cons: Domain match between train and test, limited to sequence models, might not correspond to end task performance, log 0 undefined, can be cheated by predicting common tokens

RNN

- Maps from input sequence to hidden state: $x_1, ..., x_n \to h_1, ..., h_n$
- $h_i = R(h_{i-1}, x_i) = \tanh(Wx_i + Vh_{i-1} + b)$
- $\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial V}$
- Has Vanishing/Exploding gradient problem, to solve: LSTM
- can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

Attention

- At each decoder state, compute a distribution over source inputs based on current decoder state, use that in output layer
- $p(y_i|x, y_i, ..., y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$
- Weighted sum of input hidden states: $c_i = \sum_j \alpha_{ij} h_j$
- Attention weights for input x_j at decoding y_i: $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{j'} exp(e_{ij'})}$
- Score function: $e_{ij} = f(\bar{h}_i, h_j)$