COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Linear Models: Naïve Bayes, Perceptron

CMSC 470

Marine Carpuat

Slides credit: Jacob Eisenstein

Linear Models for Multiclass Classification

Feature function representation

$$\hat{y} = \arg\max_{y} \boldsymbol{\theta}^{\top} \mathbf{f}(\mathbf{x}, y)$$

Weights

Naïve Bayes recap

- Define $\mathrm{p}(\boldsymbol{x}, \boldsymbol{y})$ via a *generative model*
- Prediction: $\hat{y} = \arg\max_y \mathrm{p}(\boldsymbol{x}_i, y)$
- Learning:

$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}} \mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

$$\mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \prod_i \mathrm{p}(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) = \prod_i \mathrm{p}(\boldsymbol{x}_i | y_i)\mathrm{p}(y_i)$$

$$\phi_{y,j} = \frac{\sum_{i:Y_i=y} x_{ij}}{\sum_{i:Y_i=y} \sum_j x_{ij}}$$

$$\mu_y = \frac{\text{count}(Y = y)}{N}$$

This gives the maximum likelihood estimator (MLE; same as relative frequency estimator)

Prediction with Naïve Bayes

$$
\begin{aligned}
\text{Score(x,y)} \quad &:= \log P(\mathbf{x}, y; \phi, \mu) \\
&= \log P(\mathbf{x}|y;\phi) P(y;\mu) \qquad \text{Definition of conditional probability} \\
&= \log P(\mathbf{x}|y;\phi) + \log P(y;\mu) \\
&= \log \text{Multinomial}(\mathbf{x};\phi_y) + \log \text{Cat}(y;\mu) \qquad \text{Generative story assumptions} \\
&= \log \frac{(\sum_n x_n)!}{\prod_n x_n!} + \log \prod_n \phi_{y,n}^{x_n} + \log \mu_y \\
&\propto \sum_n x_n \log \phi_{y,n} + \log \mu_y \qquad \text{This is a linear model!} \\
&= \boldsymbol{\theta}^{\mathsf{T}} \mathbf{f}(\mathbf{x}, y)
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{\theta} &= [\log \phi_1^{\mathsf{T}}, \log \mu_1, \log \phi_2^{\mathsf{T}}, \log \mu_2, \ldots]^{\mathsf{T}} \\
\mathbf{f}(\mathbf{x}, y) &= [\mathbf{0}, \ldots, \mathbf{0}, \mathbf{x}^{\mathsf{T}}, 1, \mathbf{0}, \ldots, \mathbf{0}]^{\mathsf{T}}
\end{aligned}
$$

- Naïve Bayes worked example on board

The perceptron

- A linear model for classification

- Prediction rule $$\hat{y} = \arg\max_{y} \boldsymbol{\theta}^{\top}\mathbf{f}(\mathbf{x}, y)$$

- An algorithm to learn feature weights given labeled data
  - online algorithm
  - error-driven

Multiclass perceptron

Algorithm 3 Perceptron learning algorithm

procedure PERCEPTRON($\boldsymbol{x}^{(1:N)}, y^{(1:N)}$)

  $t \leftarrow 0$

  $\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$

  repeat

    $t \leftarrow t + 1$

    Select an instance i

    $\hat{y} \leftarrow \operatorname{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$

    if $\hat{y} \neq y^{(i)}$ **then**

      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$

    else

      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$

  until tired

  return $\boldsymbol{\theta}^{(t)}$

Online vs batch learning algorithms

- In an online algorithm, parameter values are updated after every example
  - E.g., perceptron
- In a batch algorithm, parameter values are set after observing the entire training set
  - E.g., naïve Bayes

Multiclass perceptron: a simple algorithm with some theoretical guarantees

Definition 1 (Linear separability). *The dataset* $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ *is linearly separable iff (if and only if) there exists some weight vector* $\boldsymbol{\theta}$ *and some* **margin** ρ *such that for every instance* $(\boldsymbol{x}^{(i)}, y^{(i)})$*, the inner product of* $\boldsymbol{\theta}$ *and the feature function for the true label,* $\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)})$*, is at least* ρ *greater than inner product of* $\boldsymbol{\theta}$ *and the feature function for every other possible label,* $\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')$.

$$\exists \boldsymbol{\theta}, \rho > 0 : \forall (\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}, \quad \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) \geq \rho + \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y'). \qquad [2.35]$$

Theorem: If the data is linearly separable, then the perceptron algorithm will find a separator (Novikoff, 1962)

Practical considerations

- In which order should we select instances?
  - Shuffling before learning to randomize order helps
- How do we decide when to stop?
  - When the weight values don't change much
    - E.g., norm of the difference between previous and current weight vectors falls below some threshold
  - When the accuracy on held out data starts to decrease
  - Early stopping

ML fundamentals aside: overfitting/underfitting/generalization

Training error is not sufficient

- We care about **generalization** to new examples

- A classifier can classify training data perfectly, yet classify new examples incorrectly
    - Because training examples are only a sample of data distribution
        - a feature might correlate with class by coincidence
    - Because training examples could be noisy
        - e.g., accident in labeling

Overfitting

- Consider a model θ and its:
  - Error rate over training data $error_{train}(\theta)$
  - True error rate over all data $error_{true}(\theta)$

- We say h overfits the training data if

$$error_{train}(\theta) < error_{true}(\theta)$$

Evaluating on test data

- Problem: we don't know $error_{true}(\theta)$!

- Solution:
  - we set aside a test set
    - some examples that will be used for evaluation
  - we don't look at them during training!
  - after learning a classifier θ, we calculate

$$error_{test}(\theta)$$

Overfitting

- Another way of putting it

- A classifier θ is said to overfit the training data, if there is another hypothesis θ', such that
  - θ has a smaller error than θ' on the training data
  - but θ has larger error on the test data than θ'.

Underfitting/Overfitting

- Underfitting
  - Learning algorithm had the opportunity to learn more from training data, but didn’t
- Overfitting
  - Learning algorithm paid too much attention to idiosyncracies of the training data; the resulting classifier doesn’t generalize

Back to the Perceptron

Averaged Perceptron improves generalization

Algorithm 4 Averaged perceptron learning algorithm

procedure AVG-PERCEPTRON($\boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(1:N)}$)
  $t \leftarrow 0$
  $\boldsymbol{\theta}^{(0)} \leftarrow 0$
  repeat
    $t \leftarrow t + 1$
    Select an instance i
    $\hat{y} \leftarrow \text{argmax}_y \, \boldsymbol{\theta}^{(t-1)} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$
    if $\hat{y} \neq y^{(i)}$ **then**
      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$
    else
      $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
    $\boldsymbol{m} \leftarrow \boldsymbol{m} + \boldsymbol{\theta}^{(t)}$
  until tired
  $\overline{\boldsymbol{\theta}} \leftarrow \frac{1}{t}\boldsymbol{m}$
  return $\overline{\boldsymbol{\theta}}$

Properties of Linear Models we’ve seen so far

Naïve Bayes

- Batch learning
- Generative model p(x,y)
- Grounded in probability
- Assumes features are independent given class
- Learning = find parameters that maximize likelihood of training data

Perceptron

- Online learning
- Discriminative model score(y|x), Guaranteed to converge if data is linearly separable
- But might overfit the training set
- Error-driven learning

What you should know about linear models

- Their properties, strengths and weaknesses (see previous slides)

- How to make a prediction given a model

- How to train a model given a dataset