Perceptron learning algorithm

The multiclass preceptron learning algorithm

procedure $\textsc{Perceptron}(\boldsymbol{x}^{1:N}, y^{1:N})$
    $t \leftarrow 0$
    $\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$
    repeat
        $t \leftarrow t + 1$
        Select an instance i ▷ Online training
        $\hat{y} \leftarrow \text{argmax}_y\, \boldsymbol{\theta}^{(t-1)} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$
        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
            $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
        end if
    until tired
end procedure

The averaged preceptron algorithm

procedure $\textsc{Ave\_Perceptron}(\boldsymbol{x}^{1:N}, y^{1:N})$
    $t \leftarrow 0$
    $\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$
    $\boldsymbol{m} \leftarrow \mathbf{0}$
    repeat
        $t \leftarrow t + 1$
        Select an instance i $\triangleright$ Online training
        $\hat{y} \leftarrow \operatorname{argmax}_y \boldsymbol{\theta}^{(t-1)} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$
        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
            $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
        end if
        $\boldsymbol{m} \leftarrow \boldsymbol{m} + \boldsymbol{\theta}^{(t)}$
    until tired
    $\bar{\boldsymbol{\theta}} = \frac{1}{t}\boldsymbol{m}$
    return $\bar{\boldsymbol{\theta}}$
end procedure

Making predictions

Assuming that you have trained weights $\boldsymbol{\theta}$, finding the best scoring label is straightforward given a data instance $\boldsymbol{x}^{(i)}$:

$$\hat{y} = \operatorname*{argmax}_{y} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$$

Note that the score cannot be interpreted probabilistically.

What is the Loss function of Perceptron?

- Hinge Loss:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \max_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y) - \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)})$$

- When $\hat{y} = y^{(i)}$, the loss is zero, otherwise, it increases linearly with the gap between the score for the predicted label $\hat{y}$ and the score for the true label $y^{(i)}$.

- Plot the loss against the input gives a Hinge shape, giving its name **Hinge loss**

What is the Loss function of Perceptron?

- Hinge Loss:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \max_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y) - \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)})$$

- The derivative of the loss function:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)})$$

- Updating: At each instance, the perceptron algorithm takes a step of magnitude one in the opposite direction of this gradient

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{\theta} - \nabla_{\boldsymbol{\theta}} \ell_{\text{perceptron}} = \boldsymbol{\theta} - \frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) \\ &= \boldsymbol{\theta} - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}) + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) \end{aligned}$$

When do you stop the training?

How many iterations do you have to train, and when do you stop?

- Ideally you want to stop where the model have the best performance on previously unseen data

- One way is to check the difference between the average weight vectors after each pass of the data. If the norm of the difference falls below a certain predefined threshold, then stop training

- **Early stopping**. Hold out a proportion of the training data, and when the accuracy on this held out data set starts to decrease, the model has begun to **overfit** the training set. It's time to stop training.

Multiclass Perceptron: Parameter estimation

θ

	not	funny	painful	ok	overall	story	good	jokes	*bias*
POS	0	0	0	0	0	0	0	0	1
NEG	0	0	0	0	0	0	0	0	0
NEU	0	0	0	0	0	0	0	0	0

Training instance: y = NEG, x = "not funny at all"

$score(y, x) = \boldsymbol{\theta} \cdot \boldsymbol{f}$

$score(y = POS, x) = 1 \times 0 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 = 1$

$score(y = NEG, x) = 1 \times 0 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 0 = 0$

$score(y = NEU, x) = 1 \times 0 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 0 = 0$

Multiclass Perceptron: Parameter estimation

θ

	not	funny	painful	ok	overall	story	good	jokes	*bias*
POS	0	0	0	0	0	0	0	0	1
NEG	0	0	0	0	0	0	0	0	0
NEU	0	0	0	0	0	0	0	0	0

Training instance: y = NEG, x = "not funny at all"

$$\hat{y} = \operatorname*{argmax}_{y} \boldsymbol{score}(y, \boldsymbol{x}) = \text{POS}$$

$y \neq y'$, so update

Multi-class Perceptron: parameter estimation

Updated $\boldsymbol{\theta}$

	not	funny	painful	ok	overall	story	good	jokes	*bias*
POS	-1	-1	0	0	0	0	0	0	0
NEG	1	1	0	0	0	0	0	0	1
NEU	0	0	0	0	0	0	0	0	0

Updating:

- Add the feature vector of the instance $x^{(i)}$ to weights for the *NEG* class and subtract from the weights for the *POS* class.
- The weights for *NEU* are not updated. Why not?

Multi-class Perceptron: parameter estimation

Updated $\boldsymbol{\theta}$

	not	funny	painful	ok	overall	story	good	jokes	*bias*
POS	-1	-1	0	0	0	0	0	0	0
NEG	1	1	0	0	0	0	0	0	1
NEU	0	0	0	0	0	0	0	0	0

Assignment Project Exam Help

New Training instance: y=NEG x="painful, not funny"

https://powcoder.com

$score(y, x) = \boldsymbol{\theta} \cdot \boldsymbol{f}$

$score(y = POS, x) = 1 \times -1 + 1 \times -1 + 1 \times 0 + 0 \times 0 + 0 \times 0 +$
$0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 0 = -2$

Add WeChat powcoder

$score(y = NEG, x) = 1 \times 1 + 1 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 0 +$
$0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 = 3$

$score(y = NEU, x) = 1 \times 0 + 1 \times 0 + 1 \times 0 + 0 \times 0 + 0 \times 0 +$
$0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 0 = 0$

$\hat{y}$ = NEG, correct, so no update

Perceptron versus Naïve Bayes

- Both ℓ_{NB} and $\ell_{PERCEPTRON}$ are convex, making them relatively easy to optimize. ℓ_{NB} can be optimized in closed form, while $\ell_{PERCEPTRON}$ requires iterating over the training set multiple times.

- ℓ_{NB} can suffer infinite loss on a single example, since the logarithm of zero probability is negative infinity. Naïve Bayes will therefore overemphasize some examples and underemphasize others.

- The Naïve Bayes classifier assumes that the observed features are conditionally independent, given the label, and the performance of the classifier depends on the extent to which this assumption holds. The perceptron requires no such assumption.
- $\ell_{PERCEPTRON}$ treats all correct answers equally. Even if $\boldsymbol{\theta}$ only gives the correct answer by a tiny margin, the loss is still zero.

Perceptron vs Logistic regression

- Inference: Both Perceptron and Logistic Regression are discriminative models, but the score for Logistic Regression can be interpreted probabilistically, while the score for Perceptron can not.
- Parameter Estimation: The parameters for both Logistic Regression and Perceptron are estimated iteratively by gradient descent. The difference in parameter estimation between Logistic Regression and Perceptron follows from their loss functions.
  - The features for which the weights need to be updated for Perceptron are features for the correct label and the label that receives the highest score if they are different. For Logistic Regression, the features "fire" for all labels are updated.
  - In perceptron, incorrect features penalized by 1, while for Logistic Regression incorrect features penalized proportionally by how much off is its prediction is.

Online large margin classification

- For the perceptron, if the classifier gets the correct answer on the training example by a small margin, it may give a different answer on a nearby test instance.
- The address this, we need the answer to be correct by a large margin. **Margin** can be formalized as:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y)$$

- The margin represents the difference between the score of the correct label and the score for the highest-scoring incorrect label.
- The intuition is that it's not enough to label the training data correctly — the correct label should be separated from other labels by a comfortable margin.

Margin Loss

Assignment Project Exam Help

$$
\begin{aligned}
\ell_{\text{MARGIN}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) &= \begin{cases} 0, & \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) \geq 1 \\ 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}), & \text{otherwise} \end{cases} \\
&= \left(1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})\right)_+
\end{aligned}
$$

https://powcoder.com

Add WeChat powcoder

where $(x)_+ = max(0, x)$. The margin loss is zero when the margin between the score for the true label and the best alternative $\hat{y}$ is at least 1.

Minimizing the margin loss

The margin can be achieved by adding a cost to the score

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y) + c(y^{(i)}, y)$$

where the cost function that can be defined as

$$c(y^{(i)}, y) = \begin{cases} 1, & y^{(i)} \neq \hat{y} \\ 0, & \text{otherwise} \end{cases}$$

Note:

- The cost function is only used to compute $\hat{y}$ during training as the true label is unknown for unseen data. During test time the best label is just the label with the best score.
- For purposes of training, don't simply use the label that has the highest score. Instead, choose the one that has the highest score and cost.
- When the label that has the highest score, it may still be chosen when it has a score that is higher than an (incorrect) alternative label by at least 1, in which case the loss will be 0

The update rule for SVM

Finding the gradient of the loss function involves constrained optimization with Lagrangian multipliers. The final update rule is:

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

- The previous weights $\boldsymbol{\theta}^{(t-1)}$ are scaled by $1-\lambda$, where $\lambda \in \{0, 1\}$.

- It pulls the weights back towards zero, and in this sense it serves as a form of regularization that prevents overfitting.
- How is this different from $L2$ regularization?

Linear algorithms: a summary

- **Naïve Bayes**: *Pros*: easy to implement, estimation is fast, required on a single pass on the training data; assigns probabilities to predicted labels; controls overfitting with parameter smoothing; *Cons*: often has poor accuracy, especially with correlated features
- **Perceptron**: *Pros*: easy to implement; online; error-driven learning typically produces good accuracy, especially after averaging. *Cons*: not probabilistic; hard to know when to stop learning; lack of margin can lead to overfitting.
- **Support vector machine** *Pros*: optimizes an error-based metric, usually resulting in good accuracy; overfitting is controlled by a regulariation parameter. *Cons*: not probabilistic.
- **Logistic regression** *Pros*: error-driven and probabilistic; overfitting is controlled by a regularization parameter. *Cons*: batch learning requires black-box optimization; Logistic loss can “overtrain” on correctly labeled samples