Perceptron learning algorithm

The multiclass percept

1: **procedure** PERCEPTRON($\mathbf{x}^{1:N}$, $y^{1:N}$)
2: $t \leftarrow 0$
3: $\theta^{(0)} \leftarrow \mathbf{0}$
4:   **repeat**
5: $t \leftarrow$
6:   Sele   ▷ Online training
7: $\hat{y} \leftarrow \text{argmax}_y \, \theta \qquad \cdot \, ($
8:   **if** $\hat{y} \neq y^{(i)}$ **then**
9: $\theta^{(t)} \leftarrow \theta^{(t-1)} + \mathbf{f}(\mathbf{x}^{(i} \qquad\qquad , \hat{y})$
10:   **else**
11: $\theta^{(t)} \leftarrow \theta^{(t-1)}$
12:   **end if**
13:   **until** tired
14: **end procedure**

The averaged preceptron algorithm

1: **procedure** Averaged-Perceptron
2:   $t \leftarrow 0$
3:   $\boldsymbol{\theta}^{(0)} \leftarrow \boldsymbol{0}$
4:   $\boldsymbol{m} \leftarrow \boldsymbol{0}$
5:   **repeat**
6:     $t \leftarrow t + 1$
7:     Sele   ▷ Online training
8:     $\hat{y} \leftarrow$
9:     **if** $\hat{y} \neq y$ **then**
10:       $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} + \boldsymbol{f}(\boldsymbol{x}^{(i}, \hat{y})$
11:     **else**
12:       $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)}$
13:     **end if**
14:     $\boldsymbol{m} \leftarrow \boldsymbol{m} + \boldsymbol{\theta}^{(t)}$
15:   **until** tired
16:   $\bar{\boldsymbol{\theta}} = \frac{1}{t}\boldsymbol{m}$
17:   **return** $\bar{\boldsymbol{\theta}}$
18: **end procedure**

Making predictions

Assuming that you have trained weights g the best scoring
label is straightforward given a data instance

Note that the score cannot be interpreted prob

What is the Loss function of Perceptron?

▶ Hinge Loss:

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

▶ When $\hat{y} =$ early with the gap between the score for the pred 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:

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

- The derivative of the loss function:

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

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

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

$$= \boldsymbol{\theta} - \boldsymbol{f}(\mathbf{x}^{(i)}, \hat{y}) + \boldsymbol{f}(\mathbf{x}^{(i)}, y^{(i)})$$

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 t performance on previously unseen data
- ► One way is to check the difference between the average weight vecto
  difference fal n stop
  training
- ► **Early stopping**. Hold out a proportio
  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

	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

$score(y,x) =$

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

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

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

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"

$$\hat{y} = \underset{y}{\mathrm{argmax}}\ \textbf{\textit{score}}(y, \textbf{\textit{x}}) = \mathrm{POS}$$

$y \neq y'$, so update

Multi-class Perceptron: parameter estimation

Updated θ

	not	funny	painful	people_all	study_games	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

	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

New Training insta

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

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

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

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

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

Perceptron versus Naïve Bayes

- Both ℓ_{NB} and m
  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 exampl
  logarithm of z   e Bayes
  will therefore
  underemph

- The Naïve Bayes classifier assumes t   tures
  are conditionally independent, given t
  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 P
  discriminative models, but the score for Logistic Regression
  can be interpreted probabilistically, while the score for
  Perceptron can not.
- ▶ Parameter Estimation. The parameters fo
  Regression a   y
  gradient des   on
  between Log   om their
  loss functions.
  - ▶ The features for which the weights nee
    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 a score 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 for the correct label and the score for the highest-scoring 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

$$\ell_{\text{MARGIN}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \begin{cases} 0 & \ldots, y^{(i)}) \geq 1 \\ & \text{otherwise} \end{cases}$$

where $(x)_+ = max(0, x)$. The margin loss i   in 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 achiev

$$\hat{y} = \underset{y \in \mathcal{Y}}{\arg\max} \, \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{\quad}, y) + c(y^{\quad}, y)$$

where the cost function that can be defined as

$$c(y^{(i)}, y) =$$

Note:

- ▶ The cost function is only used to compute $\hat{\quad}$ ring training as the true label is unknown for unseen data
  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 fi

$\boldsymbol{\theta}^{(t)}$ $(\mathbf{x}^{(i)}, \hat{y})$

- ▶ The previous weights $\boldsymbol{\theta}^{(t-1)}$ are s , 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**: ~~https://eduassistpro.github.io/~~ required on a singl probabilities to predicted labels; controls overfitting with parameter smoothing; *Cons*: often has poor accuracy, especially with correlated features

- **Perceptron** *Pros*: easy to implement; on learning typi fter averaging. to stop learning; lack of margin can lead to overfit

- **Support vector machine** *Pros*: 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