The Perceptron

Simon Šuster, University of Groningen

Course *Learning from data*
November 18, 2013

References

- Hal Daumé III: A Course in Machine Learning
  `http://ciml.info`
- Tom M. Mitchell: Machine Learning
- Michael Collins, 2002: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms

Some slides are adapted from Luke Zettlemoyer and Xavier Carreras.

You've seen Naive Bayes

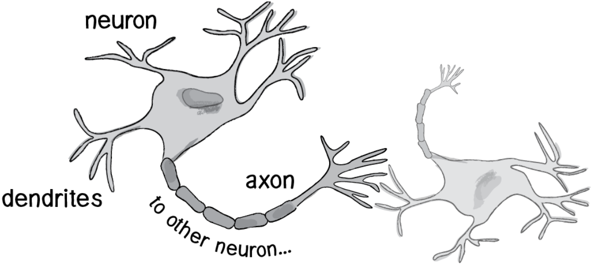
- Model-based
- Generative: joint probability (x,y)
- Assumes independence between features given label
- One pass through data

The Perceptron

is different from the Naive Bayes:

- Mistake-driven
- Often no probabilities
- Discriminative: predicting y directly from x
- Iterative
- Accuracy often comparable to more complex algorithms
- Robust: good accuracy in presence of redundant/irrelevant features

Biological inspiration

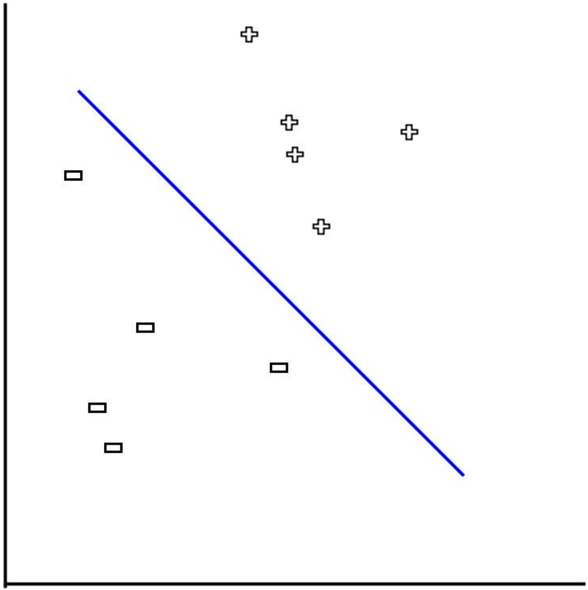


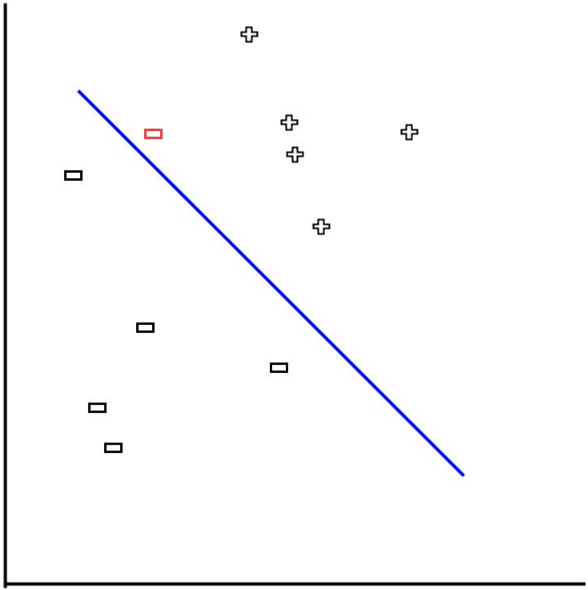
Geometric intuition I

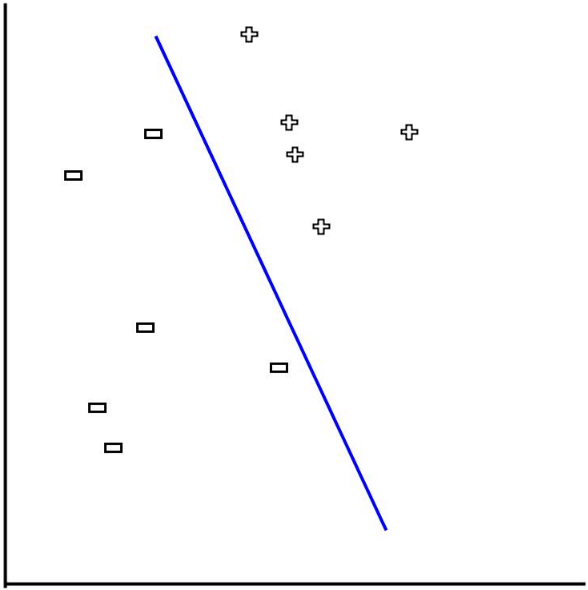
- Want to find a way of separating data points in a hyperspace (with a hyperplane)
- In a low dimensional space (2D, two features), find a line that separates the points

⇒ finding a weight vector that will separate the points

  - start with some random line
  - a data point comes in
  - if it's on the wrong side of the line, move the line







Perceptron algorithm outline

Repeat for a specified number of times:

Prediction step

- For each training instance, make a prediction (compute *activation*) with the current set of weights

  Update step

  - If the prediction is correct, don't change the weight vector
  - If it's incorrect, update the weights

For a wrongly classified instance, the perceptron should do better next time around

Representation

x: vector of *n* features (values) for a single instance
w: vector of *n* weights
y: class label

$$\mathbf{x}_{n,1} = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{n,1} \end{bmatrix} \qquad \mathbf{w}_{n,1} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{n,1} \end{bmatrix} \qquad y = \in \{-1, 1\}$$

Learning model I

- Activation a is the outcome score, used in *both* training and testing.
- It's about making prediction for a single instance (*online*) with the current set of weights.

$$
\begin{aligned}
a &= \sum_{n=1}^{N} w_n x_n \\
&= \mathbf{w}^T \mathbf{x}
\end{aligned}
$$

- Detail: shift the decision point by b (*bias*):

$$a = \mathbf{w}^T \mathbf{x} + b$$

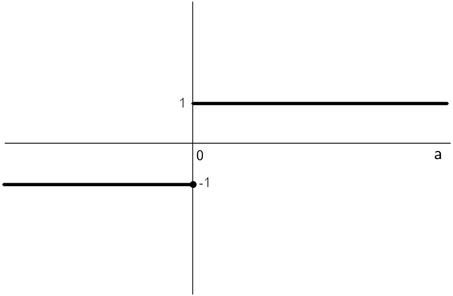
Learning model II

Testing

Assume we have already figured out $\mathbf{w}$ and b, then the output of the classifier is simply:

- computing activation a for the current test instance $\hat{\mathbf{x}}$ (see previous slide)
- applying the *SIGN* function

$$\hat{y} = SIGN(a)$$

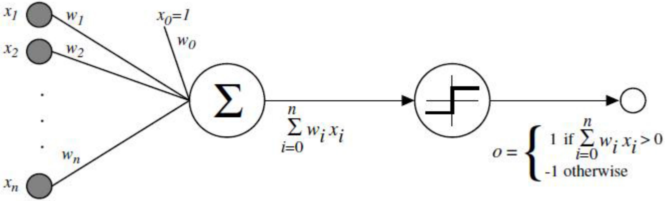


Learning model III

Training How do we learn $\mathbf{w}$ and b?
Perceptron is mistake driven:

- Start with some initial $\mathbf{w}$
- For each training instance, do prediction (activation)
- If $ya > 0$, do nothing
- If $ya \leq 0$, update the weights:

$$\mathbf{w} = \mathbf{w} + y\mathbf{x}$$
$$b = b + y$$



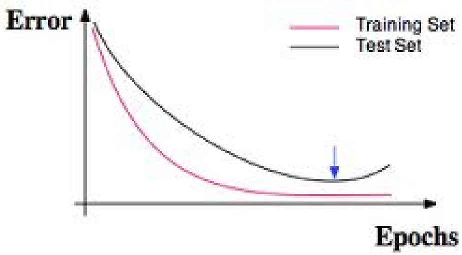
x_1
w_1
x_2
w_2
$x_0=1$
w_0
w_n
x_n
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Practical notes I

Hyperparameter

- The perceptron has one hyperparameter, *MaxIter*: number of passes through the training data
- 1 is usually not enough
- Too many iterations also not desirable
    - Overfitting the training data

Practical notes II



When to stop?

- "early stopping"
  - use a held-out set
  - measure performance with a current set of weights
  - stop when performance plateaus

Practical notes III

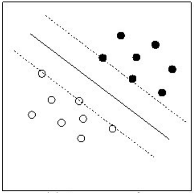
Separability

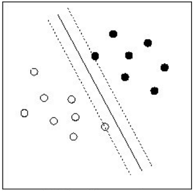
- We want to find a *separating* hyperplane, but that's possible only when data is linearly separable
- Often, that might not be the case: linguistic problems?
- In that case, find a best-fit approximation
  - find the *optimal* separating plane by gradient descent

Practical notes IV

Convergence

- It always converges if the data is linearly separable
- After how many iterations?
  - Depends on the learning problem
  - Harder problems have smaller margins


(a) Larger margin


(b) Smaller margin

Practical notes V

Presentation of training instances

- If we first present all positive instances, and then all negative instances?
- A bad classifier because it "remembered" mostly negative instances

- Permute the training data before starting
- Can permute before *each* iteration, influencing convergence rate as well

Extensions: multiclass perceptron

- Every class has its own weight vector, $\mathbf{w}_y$
- Predict the class whose weight vector produces the highest activation
- If correct, do nothing
- If wrong, update the weights:
  - downweight score of wrong answer:
    $\mathbf{w}_y = \mathbf{w}_y - \mathbf{x}$
    $b_y = b_y - 1$
  - increase score of right answer:
    $\mathbf{w}_{y^*} = \mathbf{w}_{y^*} + \mathbf{x}$
    $b_{y^*} = b_{y^*} + 1$

Extensions: voted and averaged perceptrons I

- Differ in the weight update step
- Often perform better (improved generalization)

- Fixed (ordered) data presentation can be harmful for ordinary perceptron
- It puts too much emphasis to later instances
- Solution: make it harder to override weights that survived a long time

Extensions: voted and averaged perceptrons II

Voting

- in training, remember how long weight vectors survive
- when testing, use counts for weighted majority vote
- likely to work better than ordinary perceptron, but requires storing all weight vectors

Averaging

- similarly to voting, maintain all weight vectors
- compute the average weight vector
- when testing, more efficient than voting

Extensions: structured perceptron

- Used often in NLP (tagging, NER, parsing)
- Given a sentence, predict its POS-tag sequence
- Ordinary perceptron can deal with atomic outputs but not sequences
- How do we make predictions for sequences?
  - Use factored representations (indicator features), e.g. look at bigrams of output labels
    - example: "previous/JJ 20/CD years/NNS"
    - if word at position 3 is "years", its tag is NNS, and previous tag is CD ⇒ a feature scores 1
  - Then sum these feature vectors

⇒ Best sequence found with Viterbi algorithm given current weights

- Weight update step similar to multiclass perceptron:
  - Incorrect features in a sequence are downweighted
  - Correct features are increased