CS 217: Artificial Intelligence and Machine Learning **Jan-Apr 2024**

Lecture 9: Perceptron Algorithm

Lecturer: Swaprava Nath *Scribe(s): Tejas Sharma*

9.1 Perceptron: Find Separator of Regions of Data Points

We try to optimize linear deterministic separation where all positive examples are on one side and negative examples on the other. This is on two dimensions, although it can be extended to multiple dimensions with hyperplane or plane separators.

Let that plane in 2D be $y_i = f(x_i) \in \{+1, -1\}$. Our goal is to learn a weight vector $\vec{w}$ that linearly separates the training points such that: $\vec{w}^T \vec{x_i} \; + \; w_0 \quad \begin{cases} \geq 0 & \forall y_i = 1 \\ \leq 0 & \forall y_i = -1 \end{cases}$

Note that for the simple 1D case, the parameter $\vec{w}$ becomes a 1D scalar. We increase the dimensions of $\vec{x}$ and $\vec{w}$ for multiple dimensions, like the plane, for instance.

How does $f_{\vec{w}}(\vec{x}) - sgn(\vec{w}^T \vec{x})$ find out the optimum value of $\vec{w}$ or more appropriately how does the perception find $\vec{w}$ to separate the training examples linearly?

We iterate through the training examples $(\vec{x_i}, y_i)$ one by one. Firstly, we add w_0 to $\vec{w}$ and 1 to each of the training examples $\vec{x_i}$

At each step, we check if the current classifier $\vec{w_t}$.

If the sign of $y_i = sgn(\vec{w}^T \vec{x_i})$ is correct, no update is required.

If not: we write $\vec{w}_{t+1} = \vec{w}_t + y_i \vec{x_i}$. This is how we update it.

We stop if there is no update for a certain number of iterations.

9.1.0.1 An Interesting Observation

This algorithm differs from those in other algorithms like regression, where we minimize some loss functions directly.

Those algorithms are called **dashed algorithms** while this one goes through the dataset one by one. Algorithms like this perception algorithm are known as **online algorithms**

9.1.0.2 A Formal Description of Perception Algorithm

- Initialize $\vec{w}_0$. Note that this is the $d + 1$ dimensional vector, NOT the scalar constant of our answer.
- $\forall t \in range(0, maxRounds)$: Choose a training sample randomly. $(\vec{x_i}, y_i)$ say.
  If $y_i \left(\vec{w_t}^T \vec{x_i} \right) < 0$: update $\vec{w_{t+1}} = \vec{w_t} + y_i \vec{x_i}$. OR
- Stop as before.

9.1.1 Correctness

Why do we add $\vec{x_i}$ when y_i is positive and subtract $\vec{x_i}$ when y_i is negative?

Well, this rule kicks in whenever there is a misclassification but still, why this way?

We reorient the decision boundary so that x_i and other points fall on the same side. It is possible that the new decision point is still misclassified, but not by that large a magnitude.

9.1.1.1 Why is Perceptron doing a Meaningful Update?

It will always kick in during a misclassified example, so consider $(\vec{x_i}, y_i)$. Basically $sgn\left(\vec{w_{old}}^T\vec{x_i}\right) \neq y_i$. I want to make it as positive as possible after multiplying by y_i on both sides.

$$y_i(\vec{w_{new}}^T\vec{x_i}) \;=\; y_i(\vec{w_{old}} + y_i\vec{x_i})^T\vec{x_i} \;=\; y_i\vec{w_{old}^T}\vec{x_i} + \|\vec{x_i}\|^2 > y_i(w_{old}^T x_i)$$

What if correcting this changes the boundary so much that it misclassifies previous points? Will this ever converge?

9.1.1.2 Answering the Doubt

- Yes, it is a **mistake-driven** online learning algorithm.
- It is guaranteed to converge if the dataset is linearly separable i.e. there does exist a line separating the positive points and the negative points.

9.1.1.3 Prove: If Data is Linearly Separable, then the Method Works

There exists some $\vec{w}^*$ such that $y_i\left(\vec{w^*}^T\vec{x_i}\right) \geq 0 \quad \forall i \in range(1,n)$

Let us assume that all points $\|\vec{x_i}\| \leq r$, where $r = \left\|\vec{w^*}\right\|$.

We define **margin of separation** as $\gamma = min_i \left\|\vec{w^*}^T\vec{x_i}\right\| = r\,\|x_i\|\,cos(\theta)$.

Theorem If there exists a unit vector, then after updates, the maximum error of separation is $\frac{1}{\gamma^2 r^2}$.

9.1.1.4 The Main Proof of Convergence

We track two quantities: $\vec{w_t}^T\vec{w^*}$ and $\|\vec{w_t}\|^2$.

Claim: $\vec{w_t}^T\vec{w^*}$ increases on every update at least by γ.
Proof: $\vec{w_{t+1}}^T\vec{w^*} \;=\; \vec{w_t}^T\vec{w^*} + y_i\left(\vec{w^*}^T\vec{x_i}\right) \;\geq\; \vec{w_t}^T\vec{w^*} + \gamma$

Claim: $\|\vec{w_t}\|$ increases almost by r^2.
Proof: $\|w_{t+1}\|^2 = \|w_t\|^2 + 2y_i\vec{w_t}^T\vec{x_i} < \vec{w_t}^2 + r^2$.

At the end of k iterations, we have $\vec{w_k}^T \cdot \vec{w^*} \geq k\gamma$ and $\|\vec{w_k}\|^2 < kr^2$.

Do not forget that $\vec{w_k} \cdot \vec{w^*} \leq \|\vec{w_k}\|\left\|\vec{w^*}\right\| = r\,\|\vec{w_k}\|$.

We thus have $\sqrt{k} \geq \|\vec{w_k}\| \geq \frac{\vec{w_k}^T \cdot \vec{w^*}}{r} \geq \frac{k\gamma}{r}$ and thus $k < \frac{1}{\gamma^2 r^2}$.

Summary: The finite number of mistakes, if the data is linearly separable, is true, although, at times, the time taken to converge can be quite long.

9.1.1.5 Limitations

This does not give the rate of convergence, although it gives proof of convergence and the maximum number of rounds needed to converge. It may be very fast or may use all the rounds (very slow).

The number of rounds needed can be very large if γ is very small.

It may not converge at all if points are not linearly separable.

HOMEWORK: Find one example of non-convergence.

9.2 Perceptron but from Loss Minimization Perspective

We define the loss function of a perceptron as a function that we want to maximize.

We are okay with large positives and hate large negatives of the signed value of $y_i \vec{w}^T \vec{x_i} \quad \forall i$.

Thus, let us define the loss function as $-\sum_{i=1}^{n} y_i \vec{w}^T \vec{x_i} = -\vec{w}^T \mathbf{X} \vec{y}$.
This is to be **minimized** or made as largely negative as possible.

Let us define this loss function as **hinge loss**. The net hinge loss is therefore $-\vec{w}^T \cdot \mathbf{X} \cdot \vec{y}$.

Now, we apply $stochastic gradient descent$. Basically, pick a random i and compute the gradient of the loss function $L_i(\vec{w}, \mathbf{X})\ \vec{w}$.

That is, we have $\vec{w_{t+1}} = \vec{w_t} + y_i \vec{x_i}$. This is precisely the same as our perceptron algorithm, except the order is different. However, this is not a very nice gradient. We should take a random value of the derivative at non-differentiable points.

9.3 Decision Tree Algorithm

Based on data, no. of cylinder displacement, origin, year, etc., we categorize them into "good fuel efficiency" and "bad fuel efficiency".

We first look at the cylinders, perhaps. Then, some nodes are leaf nodes, which guarantee that the car is fuel efficient or not. Then, some are intermediate nodes, like disposability or origin. Based on their values, we can classify them as good or bad.

How do we build a decision tree based on data points? We will look into this the next class.