CISC 271 Class 30

Supervised Learning – Perceptron Rule

Text Correspondence: Hastie *et al.*, 2009 [6], pp 130–132

Main Concepts:

- *Perceptron – simple binary neuron cell*
- *Learning – false positives and false negatives*
- *Perceptron Rule for machine learning*

Sample Problem, Machine Inference: How can we learn a hyperplane from data?

30.1 The Perceptron Rule

A basic algorithm for finding the weight vector $\vec{w}$ and the bias value b is the Perceptron Rule. Historically, a Perceptron was one version of an artificial neuron that was developed by Frank Rosenblatt [11] who worked independently from McCulloch and Pitts [9]. His algorithm is the basis of a great deal of current work in machine learning.

Rosenblatt's idea can be expressed as simple actions:

- If we are right, and the class matches the label, we do not change the hyperplane
- If we are wrong, and the class does not match the label, we adjust the hyperplane

The Perceptron Rule works by iteratively estimating the weight vector $\hat{w}$ that is used to determine the hyperplane of separation. The concept is basic: if the correspondence is correct then we do not change $\hat{w}$, but if there is a mis-classification then $\hat{w}$ must be adjusted. Updates to the weight vector, in the Perceptron Rule, occur if and only *if there is a false negative or a false positive produced by the current weight vector*.

We can reason about the Perceptron Rule by enumerating the cases. Each observation's label has one of two values, because $y_i \in \{0, 1\}$. Likewise, the quantization for each observation has one of two values, $q_i \in \{0, 1\}$. Together these constitute four cases: true positive, true negative, false negative, and false positive. These are summarized in Table 30.1.

We can use a common method of machine learning, which is to interpret the logical states of $\{0, 1\}$ as numerical values. A key observation that we can draw from Table 30.1 is that the difference between the label and the classification can be used to succinctly state the update to $\hat{w}$.

Table 30.1: Logic for the Perceptron Rule for observation $\#i$, which is in the "vector" $\hat{x}_i$. The observation's label is $y_i \in \{0, 1\}$ and it classification, by quantizing the linear weighted sum of the observation and the augmented weight vector $\hat{w}$, is $q_i \in \{0, 1\}$. The action is the change to $\hat{w}$ according to the Perceptron Rule.

Condition	Case	Action
$(y_i = 1) \wedge (q_i = 1)$	True Positive	$\hat{w}$ unchanged
$(y_i = 0) \wedge (q_i = 0)$	True Negative	$\hat{w}$ unchanged
$(y_i = 1) \wedge (q_i = 0)$	False Negative	$\hat{w}$ becomes closer to $\hat{x}_i$
$(y_i = 0) \wedge (q_i = 1)$	False Positive	$\hat{w}$ becomes farther from $\hat{x}_i$

Consider the residual error between the label y_i and the quantization q_i, which we can write as

$$e_i \stackrel{\text{def}}{=} y_i - q_i \tag{30.1}$$

We can write the contents of Table 30.1 using the residual error term e_i of Equation 30.1, which is given in Table 30.2.

Table 30.2: Residual error terms for the Perceptron Rule for observation $\#i$, which is in the "vector" $\hat{x}_i$. The observation's label is $y_i \in \{0, 1\}$ and it classification, by quantizing the linear weighted sum of the observation and the augmented weight vector $\hat{w}$, is $q_i \in \{0, 1\}$. The residual error is $e_i \stackrel{\text{def}}{=} y_i - q_i$. The action is the change to $\hat{w}$ according to the Perceptron Rule.

Condition	Error	Case	Action
$(y_i = 1) \wedge (q_i = 1)$	$e_i = 0$	True Positive	$\hat{w} \leftarrow \hat{w}$
$(y_i = 0) \wedge (q_i = 0)$	$e_i = 0$	True Negative	$\hat{w} \leftarrow \hat{w}$
$(y_i = 1) \wedge (q_i = 0)$	$e_i = +1$	False Negative	$\hat{w} \leftarrow \hat{w} + e_i \hat{x}_i$
$(y_i = 0) \wedge (q_i = 1)$	$e_i = -1$	False Positive	$\hat{w} \leftarrow \hat{w} + e_i \hat{x}_i$

The actions in Table 30.2 can be summarized as a single update for the Perceptron Rule:

$$\hat{w} \leftarrow \hat{w} + e_i \hat{x}_i \tag{30.2}$$

There are many theoretical and practical results associated with the Perceptron Rule. For us, a very important result is

If the training data vectors $\hat{x}_j$ can be separated by a hyperplane, then the Perceptron Rule converges to a correct weight vector

The Perceptron Rule works by updating the weight vector based on the mis-classifications. Where the correct classification of an augmented training data vector $\hat{x}_j$ is y_j, the algorithm updates the augmented weight vector $\hat{w}$ by comparing the computed classification to the known label; depending on the direction of the mis-classification, the incorrectly classified augmented data vector is added to or subtracted from the augmented weight vector. In pseudocode, where we use a "missed" flag to help us to end an enclosing loop early, this could be written as

```
for max_iterations,
   Missed ← F
   for each training sample i,
      q_i ← heaviside(x̂_i^T ŵ)          quantization
      e_i ← (y_i − q_i)
      ŵ ← ŵ + e_i x̂_i
      if (e_i ≠ 0)                      Missed ← T
          need to repeat
      endif
   endfor
   if ¬Missed then BREAK
endfor
```

This simple algorithm, which updates the augmented weight vector according to how the training data vectors are mis-classified, is usually enclosed in another loop. The outer loop typically places a limit on the number of updates to the augmented weight vector, so that an infinite loop does not result from non-separable training data. The "missed" flag can be used to exit the outer loop when all of the training data are correctly classified, that is, when no further computations are necessary.

The above algorithm can be coded in Matlab as a function that iterates from an initial estimate of the augmented weight vector $\hat{w}$ to a final estimate. Because convergence of the Perceptron Rule is guaranteed for linearly separable data, in this course we will not worry about error handling or speed of convergence.

In this course, we will not address the thorny practical problem of what happens when we try to adjust the hyperplane to accommodate one data vector, and the adjustment results in the mis-classification of another data vector. That problem, and related problems, are topics in courses on more advanced data analysis, machine learning, and artificial intelligence.

30.2 Perceptron Rule – Batch Learning

We have often found it useful to gather data into a structure in linear algebra, so let us consider this practice for the Perceptron Rule. We have m observations, which we can gather into an augmented design matrix. Using our circumflex notation, we can write this design matrix as

$$\hat{X} \in \mathbb{R}^{m \times (n+1)} \tag{30.3}$$

We can gather each observation's label, y_i, into a label vector $\vec{y}$. Each quantization q_i can also be gathered into a quantization vector $\vec{q}$. Equation 30.2 can be stated concisely, working on all of the observations simultaneously, at iteration k as

$$\hat{w}_k \leftarrow \hat{w}_{k-1} + \hat{X}^T(\vec{y} - \vec{q}) \tag{30.4}$$

30.3 Example: Performance on the Iris Data Set

The Iris data set, as we saw in previous classes, was not correctly clustered by the k-means algorithm. We understood that the problem was because one data vector was mis-classified. What if we randomly select an augmented weight vector $\hat{w}$ as an initial estimate, and run the Perceptron Rule on the data set?

After converting the original data vectors into augmented training data vectors, the Perceptron Rule correctly separates the data. However, the separating hyperplane is not necessarily located and oriented in a way that a human might separate the data. Examples of convergences from random initial estimates, which have been exaggerated by slowing down the algorithm, are shown in Figure 30.1.

We can immediately appreciate that, in these examples, the separating hyperplane is "close" to one or the other subset of training data. This is a consequence of the Perceptron Rule, which stops updating the augmented weight vector $\hat{w}$ when all of the training data are correctly classified.

This situation seems less than ideal. A great deal of work in machine learning has gone into ways to improve the Perceptron Rule so that a better hyperplane estimate is calculated.

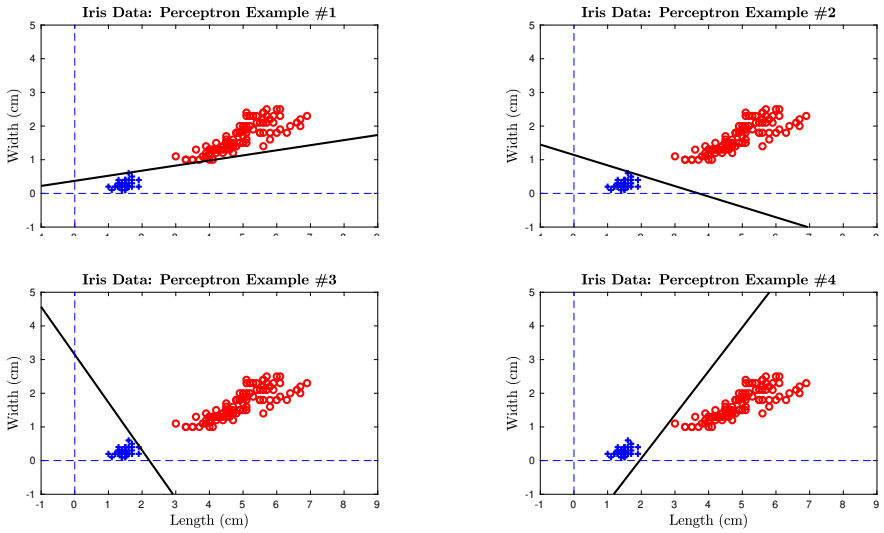

Figure 30.1: Convergence of the Perceptron Rule on the Iris data set. Four randomly selected initial estimates of the separating hyperplane produced four distinctly different results. The effects were exaggerated by slowing the algorithm's convergence with a value η.