Perceptron Classifiers

Charles Elkan
elkan@cs.ucsd.edu

January 15, 2010

Suppose that we have n training examples. The training data are a matrix with n rows and p columns, where each example is represented by values for p different features. Assume that each feature value is a real number. Let feature value j for example number i be written x_{ij}. The label of example i is y_i. For example, $y_i = 1$ if message i is spam and $y_i = 0$ if it is not spam.

We have separate training and test sets of examples. Each test example is also represented as a row vector of length p. The label y for a test example is unknown. The output of a classifier is a guess at y.

The simplest way to distinguish between two classes in p-dimensional Euclidean space $\mathbb{R}^p$ is a hyperplane, that is a linear subspace of dimension $p-1$. The parameters defining a hyperplane are a vector w in $\mathbb{R}^p$ and a scalar b. The former gives the orientation of the hyperplane, which is at right angles (also called perpendicular, also called orthogonal) to w. Only the direction of w is important, not its length, so the true number of parameters of w is $p - 1$. The scalar b specifies the distance from the origin to the hyperplane along the direction specified by w. In all, it takes a total of p scalars to specify a hyperplane in $\mathbb{R}^p$.

The projection of a vector (i.e. a point) u onto a unit vector v is the scalar $u \cdot v = \sum_{j=1}^{p} u_j v_j$, where $\cdot$ is the symbol for scalar product. The vector v is a unit vector if its length is 1, that is $||v|| = \sqrt{v \cdot v} = (\sum_{j=1}^{p} v_j^2)^{1/2} = 1$. Think of $u \cdot v$ as the length of the shadow of u falling onto the direction of v. If w is a unit vector, then the hyperplane consists of all points x whose projection onto w equals b, i.e. $x \cdot w = w \cdot x = b$. The points on one side have projection greater than b, while the points on the other side have projection less than b.

The hyperplane classifies a training example $\langle x, y \rangle$ correctly if and only if $y(w \cdot x - b) > 0$, that is if and only if y and $w \cdot x - b$ have the same sign. A

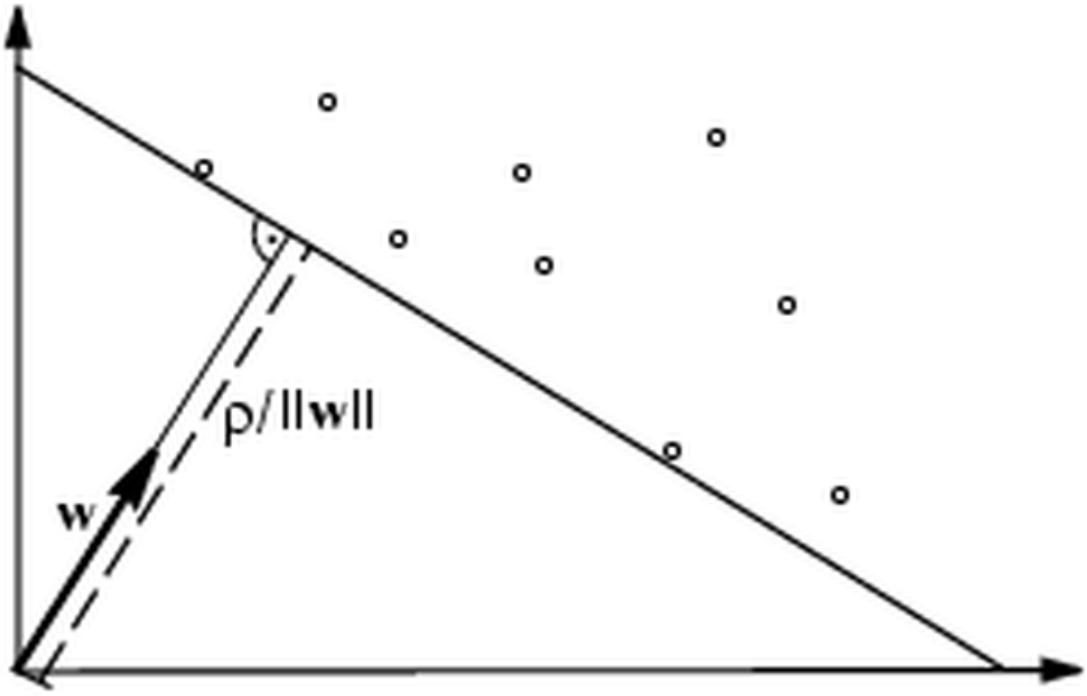

Figure 1: A hyperplane defined by a vector w and a scalar ρ. All points x indicated by circles have $x \cdot w > \rho$. Picture from an unknown web source.

hyperplane is a convenient classifier because it is fast to apply to a test example z, since it only takes $O(p)$ time to compute $w \cdot z$.

Without loss of generality, we will only look at learning hyperplanes that go through the origin, that is with $b = 0$. "Without loss of generality" means that if we have an algorithm for learning hyperplanes through the origin, we can use it to learn hyperplanes with shift b by simply rewriting the training examples. Specifically, we extend each original example x with an extra coefficient $x_{p+1} = 1$. Let x' be the extended version of x and suppose $y(w' \cdot x') > 0$ for every training example $\langle x, y \rangle$. Then $w' \cdot x' = w \cdot x - b$ where $b = -w'_{p+1}$ so w' includes a vector w and a scalar b defining a hyperplane shifted away from the origin. (Illustration of changing 2D data on a circle to data on a circle on a sphere.)

The task of learning a hyperplane through the origin is:

Task: Given training data $\{\langle x_i, y_i \rangle\}$ find w such that $y_i(w \cdot x_i) > 0$ for all i.

Geometrically, the constraint for each training example specifies a half-space that w must lie inside: $w \cdot x_i > 0$ if $y_i > 0$ and $w \cdot x_i < 0$ if $y_i < 0$. Some obvious questions arise: (1) What if there is no solution? (2) If more than one solution w exists, then which one should we choose? That is, what is the objective function of w to maximize?

If we fix a linear objective function, then the learning task becomes what is called a "linear programming" problem: maximizing an objective function that is linear subject to constraints that are also linear. This raises a third issue: (3) Even if a solution exists, algorithms for linear programming are complicated and slow and/or biologically not plausible.

In 1958 Frank Rosenblatt created much enthusiasm when he published a method called the "perceptron algorithm" that answers the three concerns above.[1] For convenience, let the training labels y_i be -1 or $+1$. The basic algorithm is very simple:

initialize $w = 0$
while any training example $\langle x, y \rangle$ is not classified correctly
    set $w := w + yx$

Consider an example of the algorithm in operation. Suppose that all training examples are points on a circle, with positive points in one semicircle, and negative points in the opposite semicircle. Remember that the hyperplane goes through the origin and is perpendicular to the vector w.

[1] For interesting and important historical background, see the Wikipedia biographies of Rosenblatt and of Marvin Minsky.

Notice that as the algorithm proceeds the coefficients of the vector w may keep growing in magnitude, and/or in digits of precision.

The perceptron algorithm has an "online" version also. Given an infinite stream of training examples $\langle x_t, y_t \rangle$ where time is indexed $t = 1, 2, 3, \ldots$, this version is as follows:

initialize $w = 0$
for $t = 1, 2, 3, \ldots$
    if $y_t(w \cdot x_t) \leq 0$ then set $w := w + y_t x_t$

This online learning algorithm is biologically plausible, because (i) it uses only simple arithmetic, (ii) the learner does not need to memorize examples, (iii) examples can arrive in any order selected by nature, and (iv) the learner can run the algorithm throughout its life.

The algorithm also has aspects that are not biologically plausible. The operation $w := w + y_t x_t$ needs more and more digits of precision to be performed correctly. It is plausible that neurons can do basic arithmetic easily, and that they may have evolved to implement an algorithm like the online perceptron. However, it is not plausible that they can do high-precision arithmetic.

The perceptron algorithm is mathematically important because of a theorem about its convergence, due to Novikoff in 1962.

Assumptions: Let the training set be finite or infinite. Let $R = \max_t ||x_t||$. If the training data are all on the unit sphere, then $R = 1$ for example. Suppose that the learning task is solvable, i.e. there exists some vector w^* of unit length and some $\delta > 0$ such that $y_t(w^* \cdot x_t) \geq \delta$ for all t.

Theorem: Under these assumptions, the perceptron algorithm converges after at most $(R/\delta)^2$ updates.

Proof: Let w_n be the w vector after n updates and let $w_0 = 0$. We will argue that whenever w is updated it becomes closer to w^*.

Suppose w_{n+1} is an update, i.e. w_n fails to classify x correctly and hence $w_{n+1} = w_n + yx$. Consider

$$w_{n+1} \cdot w^* = (w_n + yx) \cdot w^* = w_n \cdot w^* + yx \cdot w^* \geq w_n \cdot w^* + \delta.$$

This says that the projection of w_{n+1} onto w^* has increased. We would like this to mean that w_{n+1} is closer to w^*. However, what it really means is that w_{n+1} is closer to w^* and/or w_{n+1} has grown bigger. So, consider the Euclidean length of w_{n+1}:

$$||w_{n+1}||^2 = ||w_n + yx||^2 = ||w_n||^2 + 2y(w_n \cdot x) + ||x||^2 \leq ||w_n||^2 + R^2$$

since $y(w_n \cdot x) \leq 0$. Now, after N actual updates we know two facts: $||w_n||^2 \leq NR^2$ and $w_n \cdot w^* \geq N\delta$. Putting these together gives a contradiction if N is too large: $w_N \cdot w^* \leq ||w_N||||w^*|| = ||w_N||$ so $N\delta \leq ||w_N|| \leq R\sqrt{N}$ so $\sqrt{N} \leq R/\delta$. **End of proof.**

Can the perceptron learn to distinguish between any two classes? If yes, it is a general-purpose biologically plausible learning algorithm! The answer is obvious in retrospect: No. The reason is simple. "You cannot learn what you cannot represent." Many concepts (i.e. distinctions between classes) cannot be represented by a hyperplane. The simplest example of a non-representable, and hence non-learnable, concept is exclusive-or in $\mathbb{R}^1$: u xor v is true if and only if $u = v = 0$ or $u = v = 1$.

The insight that the perceptron algorithm is incapable of learning many very simple distinctions killed most interest in it for many years. However, there are at least three compelling reasons to investigate perceptron methods further.

One, as mentioned above, the online perceptron is a lifelong learning method for an intelligent agent such as an animal or robot. Two, the convergence theorem says that the algorithm can generalize from a *finite* training set to a concept that is valid for an *infinite* set, because the final classifier is a $+1/-1$ weighted sum of a finite number of training points, even if the input is an infinite stream of data.

Of course, this guarantee is only true under some conditions. First, some such valid concept must actually exist. Second, a valid concept must exist without needing too much arithmetic precision. Specifically, learning converges after at most $(R/\delta)^2$ updates, where $R = \max ||x_t||$ and δ is a level of precision: we need $y_t(w^* x_t) \geq \delta$ for all t.

Three, a modern observation is that the number of updates until convergence does not depend on the dimensionality of the data. This suggests that perceptron methods will be useful for very high-dimensional data such as images. Indeed, this is true.

The resurgence of interest in perceptron-type classifiers came in the 1980s because backpropagation was invented as a training algorithm for multiple perceptrons connected in layers. Layers means that the outputs of lower-level perceptrons are the inputs of a higher-level perceptron. Rather confusingly, the lower layer of a two-layer network of perceptrons of often called a hidden layer, and such a network is often said to have three layers. The reason for this terminology is that there are three levels of nodes. The middle level is the outputs of the lower-layer perceptrons. These nodes are hidden because they are not visible input values x and they are not visible training labels y.

It turns out that a multilayer perceptron with a single hidden layer is a universal

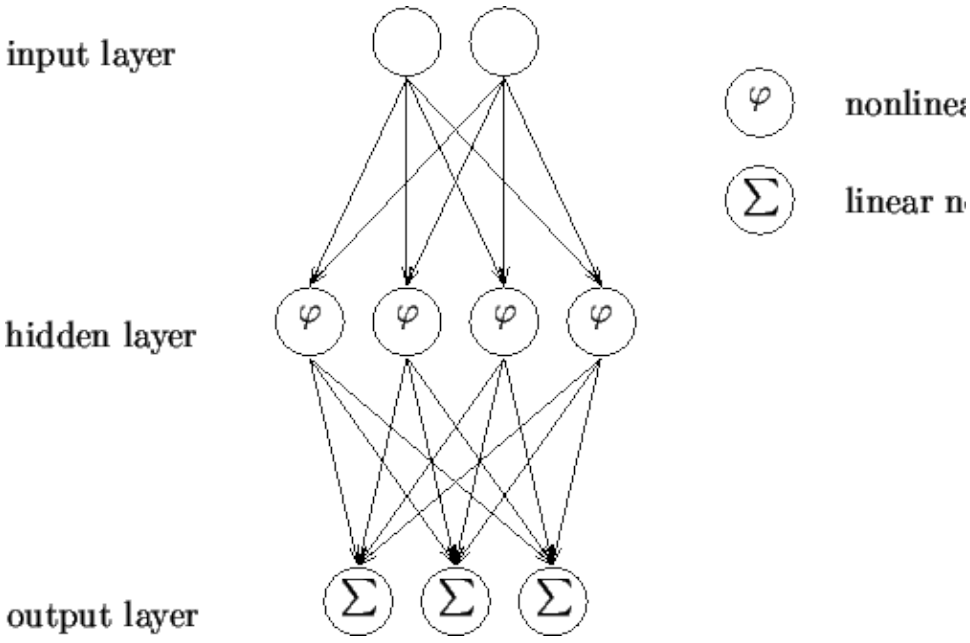

Figure 2: A multilayer perceptron with a single hidden layer. Picture from `www.cis.hut.fi/ahonkela/dippa/`.

approximator. This means that the network can mimic any continuous function $\mathbb{R}^n \to \mathbb{R}^m$ with any specified level of accuracy. However, as the desired accuracy increases, and as the function to be represented becomes less smooth, the number of nodes needed increases, and/or the number of digits of numerical precision needed. One of the first proofs of this fact was given by Hal White, professor of Economics at UCSD [Hornik et al., 1989].

The universal-approximator property depends on having at least one hidden layer, and on the nodes being nonlinear, which are biologically plausible. However it also depends on high-precision arithmetic, which may not be biologically plausible. Moreover, plausibility for the representation does not necessarily imply plausibility for the training algorithm.

Overall, biological plausibility remains an open question. On the one hand, Terry Sejnowski, also a professor at UCSD, has written "No biological significance is claimed for the algorithm (backpropagation) by which the network developed" [Lehky and Sejnowski, 1988, page 454]. On the other hand, current findings in neuroscience suggest that forward and backward waves of activity, until quiescence, may be widespread in the brain. Also, some modern training

algorithms benefit from injecting random noise, so high-precision arithmetic may not be needed.

The basic perceptron algorithm will never converge on nonseparable data. An idea for dealing with nonseparable data is to just iterate through the training data for a fixed number K of epochs. The disadvantage of this procedure is that just before finishing, it might do an update on an outlier, and hence terminate with a very bad concept w. A better idea is, for every w ever considered, to remember for how many training points it was correct.

This is the idea behind a method called the "voted perceptron" due to Yoav Freund, another UCSD professor, and Robert Schapire [Freund and Schapire, 1999]:

initialize $n = 1$, $w_1 := 0$, $c_1 := 0$
repeat for T epochs:
    for $i = 1$ to $i = m$ (this is one epoch)
        if $\langle x_i, y_i \rangle$ is classified correctly then increment c_n
        otherwise:
            increment n
            $w_n := w_{n-1} + y_i x_i$
            $c_n := 1$

When the algorithm terminates we have a set of classifiers w_n each with a weight c_n that is its survival time. The survival times add up to mT. We know that each w_n was correct on either c_n or $c_n - 1$ training examples, so c_n is a reasonable measure of the reliability of w_n.

The final classifier is nonlinear. It is

$$f(x) = \text{sign}(\sum_n c_n \text{sign}(w_n \cdot x)).$$

Computing $f(x)$ requires storing all the intermediate w_n in memory, which is not practical. Making the classifier linear by eliminating the inner sign operator makes it efficient:

$$f'(x) = \text{sign}(\sum_n c_n (w_n \cdot x)) = \text{sign}(x \cdot \sum_n c_n w_n).$$

In experiments this averaging method works slightly better than the voting method.

Why is the voted perceptron important? First, there is a theorem about generalization accuracy with m training examples, k training mistakes, and bounds R and δ. The result of the theorem is an upper bound on the error probability for an iid (independent identically distributed) test example. Second, the algorithm works well in practice on high-dimensional separable as well as nonseparable data.

Quiz 2, January 14, 2010

Write your name:

Let the training set be $\{\langle x_i, y_i \rangle\}$ for $i = 1$ to $i = n$, where $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$. Dr. Rocchio says that the perceptron algorithm is too complicated and instead we should just use the linear classifier $w = u - v$ where u is the average of the positive examples and v is the average of the negative examples:

$$u = \frac{1}{|\{i : y_i = +1\}|} \sum_{\{i:y_i=+1\}} x_i$$

and

$$v = \frac{1}{|\{i : y_i = -1\}|} \sum_{\{i:y_i=-1\}} x_i.$$

[3 points] Draw a simple sketch of a small dataset. Use your sketch to explain in one or two sentences why Dr. Rocchio's suggestion is sensible.

Hint: Draw a few positive and negative examples on the unit circle in two dimensions. Draw the u and v vectors and explain what the hyperplane corresponding to $w = u - v$ looks like.

Note: The idea above is the essence of the Rocchio algorithm that is widely used in information retrieval. If the positive examples are relevant documents and the negative examples are irrelevant documents, then the Rocchio w is a reasonable definition of a corresponding query. However, the Rocchio w sometimes yields a hyperplane that does not separate the positive and negative training examples, even when these are separable.

References

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

[Lehky and Sejnowski, 1988] Lehky, S. R. and Sejnowski, T. J. (1988). Network model of shape-from-shading: neural function arises from both receptive and projective fields. *Nature*, 333(6172):452–454.