Recap – Non-parametric estimation

- In non-parametric estimation we do not assume any density model.
- These methods essentially generalize the histogram-based approximation of a density.
- Kernel density estimates approximate the unknown density as mixture of densties centered on data points.
- nearest neighbour estimates are closely related to nearest neightbour classifier.

Linear Classifier for a 2-class problem

- Let $W = (w_0,\ w_1,\ \cdots\ ,\ w_d)^T$ be the parameter vector and let $X = (x_1,\ \cdots\ , x_d)^T$ be the feature vector. Then

$$
\begin{aligned}
h(X) &= 1 \text{ if } g(W,X) = \sum_{i=1}^{d} w_i x_i + w_0 > 0 \\
&= 0 \text{ Otherwise}
\end{aligned}
$$

  is called a linear classifier.
- The $g(W, X)$ is called a linear discriminant function.
- It is linear in parameters, w_i.
- It is also linear in x_i (though it is not important).

- A linear discriminant function can be of the form

$$g(W, X) = \sum_{i=1}^{d'} w_i \phi_i(X) \ + \ w_0$$

- We are essentially using $z_i = \phi_i(X)$ as the features.
- As long as ϕ_i are fixed, this is a 'linear' classifier.
- We will use X as the feature vector but will remember that all the algorithms are valid if we use $\phi_i(X)$ instead of x_i.

- Define $\tilde{X} = (1,\ x_1,\ \cdots,\ x_d)^T$, called the **augumented feature vector**.
- Let $W = (w_0,\ w_1,\ \cdots,\ w_d)^T$ be the parameter vector.
- Now we have

$$g(W,\ X) = w_0\ +\ \sum_{i=1}^{d} w_i x_i\ = W^T \tilde{X}$$

- We assume that the feature vector is augumented (whenever needed) though we write it as X.

- The training set: $\{(X_i, y_i), i = 1, \cdots, n\}$ is said to be **linearly separable** if there exists W^* such that

$$\begin{aligned} X_i^T W^* &> 0 \text{ if } y_i = 1 \\ &< 0 \text{ if } y_i = 0 \end{aligned}$$

- Any W^* that satisfies the above is called a separating hyperplane. (There exist infinitely many separating hyperplanes, if data is linearly separable)

Learning linear classifiers

- The classifier is:

$$h(X) = \text{sgn}\left(\sum_{i=1}^{d} w_i x_i \; + \; w_0\right) = \text{sgn}\left(W^T X\right)$$

- Need to learn 'optimal' W from the training samples.
- Perceptron learning algorithm is one of the earliest algorithms.
- Finds a separating hyperplane, if it exists.
- We start with this algorithm.

Perceptron Learning Algorithm

- The algorithm is an iterative algorithm to learn W corresponding to a separting hyperplane.
- Let $W(k)$ denote the weight vector at k^{th} iteration.
- At each iteration we pick a training sample.
- Let $X(k)$ be the one picked at k and let $y(k)$ denote its class label.
- At k^{th} iteration we classify $X(k)$ using $W(k)$ and based on the correctness or otherwise of the classification, update $W(k)$ to $W(k+1)$.

- We can keep picking feature vectors one-by-one from the training data (and keep repeatedly going over the training set).
- We stop when the current weight vector correctly classifies all the training data.
- For the ‘stopping criterion’, we can remember when we had an incorrect classification.
- We think of this as an online or incremental algorithm

Perceptron Learning Algorithm

Let $\Delta W(k) = W(k+1) - W(k)$. Then

$$
\begin{aligned}
\Delta W(k) \quad &= 0 && \text{if} \quad W(k)^T X(k) > 0 \ \& \ y(k) = 1, \quad \text{or} \\
& && \quad\;\; W(k)^T X(k) < 0 \ \& \ y(k) = 0 \\
&= X(k) && \text{if} \quad W(k)^T X(k) \leq 0 \ \& \ y(k) = 1 \\
&= -X(k) && \text{if} \quad W(k)^T X(k) \geq 0 \ \& \ y(k) = 0
\end{aligned}
$$

- This is a simple ‘error correction’ algorithm.
- Everytime the current sample is incorrectly classified, we ‘locally’ try to correct the error.

- Suppose $W(k)^T X(k) \leq 0$ & $y(k) = 1$. Then

$$\begin{aligned} W(k+1)^T X(k) &= (W(k) + X(k))^T X(k) \\ &= W(k)^T X(k) \; + \; X(k)^T X(k) \\ &\geq W(k)^T X(k) \end{aligned}$$

- Similarly when $W(k)^T X(k) \geq 0$ & $y(k) = 0$,

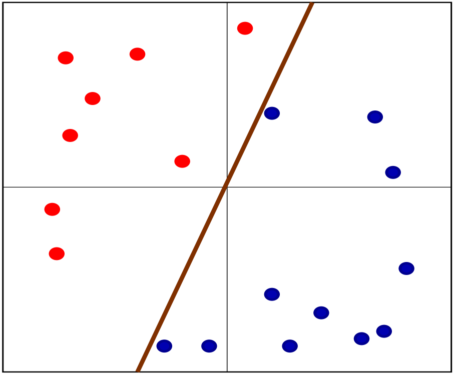
$$W(k+1)^T X(k) \; \leq \; W(k)^T X(k)$$

- Thus the corrections are intuitive.

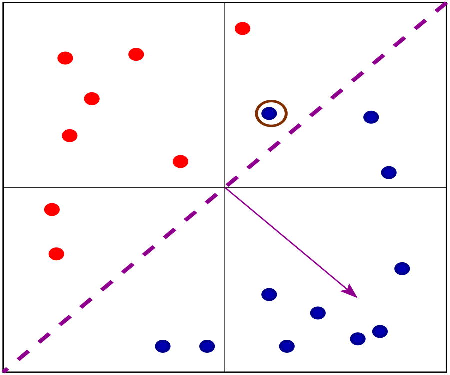
- Thus, the motivation for the algorithm is easy to see.
- However, it is not clear why the algorithm should work.
- Firstly, there is no guarentee that $W(k+1)^T X(k)$ has correct sign. (Note that the 'step size' is arbitrary).
- Secondly, when we correct $W(k)$ to take care of $X(k)$, we may now misclassify some feature vector that $W(k)$ classified correctly.
- Hence, it is remarkable that the algorithm works (as we show later).

Perceptron: Geometric view

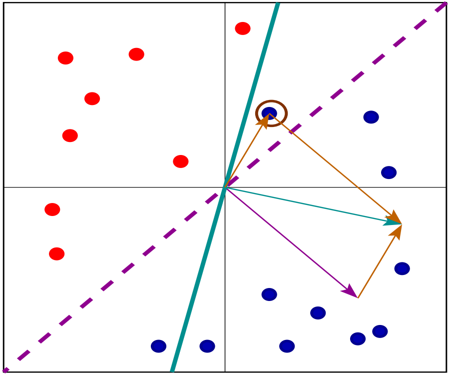
The algorithm has a simple geometric view. Consider the following data set.
(In the 2D feature space but seprable by line through origin)



- Suppose $W(k)$ misclassifies a pattern.



- Suppose $W(k)$ misclassifies a pattern.
- Now the correction made to $W(k)$ can be seen as



Convergence of Perceptron Algorithm

- Now we show that the algorithm learns a separating hyperplane.
- Recall that we assume augumented feature vectors.
- In addition, let us suppose that, in the training set, all X_i with $y_i = 0$ are multiplied by -1.
- Now a weight vector W represents a separating hyperplane if $W^T X_i > 0, \ \forall i$.
- This simplifies our notation.

- Under our notation now, the Perceptron algorithm is as follows.
- Whenever $W(k)^T X(k) \leq 0$, we set $W(k+1) = W(k) + X(k)$.
- Let us count our iterations only when the weight vector is updated.
- Then, under the Perceptron algorithm we have

$$W(k+1) = W(k)+X(k), \;\; W(k)^T X(k) \leq 0, \;\; k = 0, 1, \cdots$$

- The algorithm stops when it finds a separating hyperplane.

- We want to show that the algorithm finds a separating hyperplane if the data is linearly separable.
- We prove this by contradiction. Assume the algorithm fails to find a separating hyperplane.
- If the perceptron algorithm stops, it found a seprating hyperplane.
- Hence, if the algorithm fails to find a seprating hyperplane, then we must have

$$W(k)^T X(k) \leq 0, \ \forall k$$

- Under the perceptron algorithm we have $W(k+1) = W(k) \, + \, X(k), \ \forall k$.
- Hence we have

$$\begin{aligned}
||W(k+1)||^2 &= ||W(k) \, + \, X(k)||^2 \\
&= ||W(k)||^2 \, + \, ||X(k)||^2 \, + \, 2W(k)^T X(k) \\
&\leq ||W(k)||^2 \, + \, ||X(k)||^2
\end{aligned}$$

because we have $W(k)^T X(k) \leq 0, \ \forall k$.

By recursing on this, we get

$$
\begin{aligned}
||W(k)||^2 &\le ||W(k-1)||^2 + ||X(k-1)||^2 \\
&\le ||W(k-2)||^2 + ||X(k-2)||^2 + ||X(k-1)||^2 \\
&. \\
&. \\
&. \\
&\le ||W(0)||^2 + \sum_{i=0}^{k-1} ||X(i)||^2
\end{aligned}
$$

- Without loss of generality, let us take $W(0) = 0$.
- Let $M = \max_i \; ||X_i||^2$
- Then we have

$$
\begin{aligned}
||W(k)||^2 &\le ||W(0)||^2 + \sum_{i=0}^{k-1} ||X(i)||^2 \\
&\le k\, M
\end{aligned}
$$

- The square of the norm of W vector grows linearly with iterations.

- Under the perceptron algorithm we have $W(k+1) = W(k) \; + \; X(k), \; \forall k$.
- Hence we have

$$
\begin{aligned}
W(k) &= W(k-1) \; + \; X(k-1) \\
&= W(k-2) \; + \; X(k-2) \; + \; X(k-2) \\
&\;\;. \\
&\;\;. \\
&\;\;. \\
&= W(0) \; + \; \sum_{i=0}^{k-1} X(i)
\end{aligned}
$$

- This shows that $W(k)$ is always some linear combination of feature vectors.

- Since the data is linearly separable, we have

$$\exists W^*, \;\; \text{such that} \;\; X_i^T W^* > 0, \; \forall i$$

- Let $\gamma = \min_i \; X_i^T W^*$. Note, $\gamma > 0$.
- Now we have

$$W(k)^T W^* = \sum_{i=0}^{k-1} X(i)^T W^* \; \geq \; k\gamma \; > \; 0$$

Putting all this together,

$$\begin{aligned} k^2\,\gamma^2 &\le |W(k)^T W^*|^2 \\ &\le ||W(k)||^2\,||W^*||^2 \\ &\le ||W^*||^2\,kM \end{aligned}$$

because $||W(k)||^2 \le kM$.

- This should be true for all k if the algorithm keeps updating the weight vector.

- If the algorithm keeps updating $W(k)$, we must have

$$k^2 \, \gamma^2 \; \leq \; ||W^*||^2 \, kM$$

- But this can be true only till

$$k \; \leq \; \frac{||W^*||^2 M}{\gamma^2}$$

- Hence algorithm finds a separating hyperplane in finitely many iterations.

- What we showed is that Perceptron algorithm finds a separating hyperplane (if it exists) in finitely many iterations.
- But we do not know the bound on iterations because we do not know W^*.
- But the proof shows that our simple error correcting procedure is effective!
- Possibly, the first provably correct learning algorithm!
- If the data is not linearly separable then, in general, the algorithm will not stop.

- Recall that the algorithm is

$$W(k+1) = W(k) + X(k) \quad \text{if} \;\; W(k)^T X(k) \leq 0$$

- The proof is valid for many generalizations.
- We could use any positive 'step-size' in the algorithm.
- We can pick patterns in any order as long as all patterns are picked repeatedly.
- Many such variations can all be justified by this proof.

- This is an 'online' algorithm.
- We are given a series of X_i and we assume there exists a W to correctly classify all X_i.
- Then what we derived is a **Mistake Bound**.
- The algorithm would not make more than this many mistakes. The bound is

$$\frac{||W^*||^2 \ \max_i\{||X_i||^2\}}{\gamma^2}$$

where $X_i^T W^* \geq \gamma > 0$ (and all are 2-norms).

Batch Vs Incremental

- The algorithm as presented is **incremental** or **online** algorithm.
- We use one example at a time.
- In principle, we can learn with a stream of examples without storing them.
- As opposed to this, we can think of a **batch** version of the algorithm as follows.
- We make one pass over all the examples, with the same $W(k)$, keep track of all wrongly classified examples, and effect all corrections together.

- As earlier, we assume that all X_i with $y_i = 0$ are multiplied by -1.
- Define set of indices, S_k, by

$$S_k = \{j \ : \ W(k)^T X_j \leq 0\}$$

- Then, in the batch version of the algorithm, after the k^{th} pass over the data, we update weight vector as

$$\begin{aligned} W(k+1) &= W(k) + \sum_{j \in S_k} X_j \\ &= W(k) + \sum_{j \,:\, W(k)^T X_j \leq 0} X_j \end{aligned}$$

An Optimization View

- We can look at Perceptron algorithm as minimizing some cost function.
- Define a figure of merit for each W as

$$J(W) \ = \ - \sum_{j \,:\, W^T X_j \leq 0} W^T X_j$$

- If W^* is a separating hyperplane then $J(W^*) = 0$.
- Also, $J(W) \geq 0, \ \forall W$.
- Hence we can learn a separating hyperplane by minimizing $J(\cdot)$.

- We want to minimize

$$J(W) = -\sum_{j\,:\,W^T X_j \leq 0} W^T X_j$$

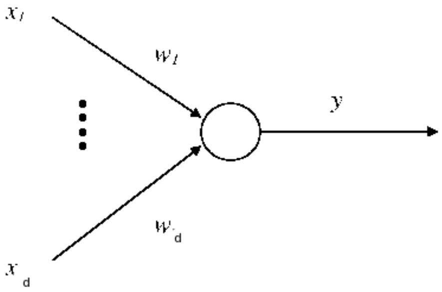
- A gradient descent on this objective function is

$$\begin{aligned} W(k+1) &= W(k) - \eta \nabla J(W(k)) \\ &= W(k) + \eta \sum_{j\,:\,W(k)^T X_j \leq 0} X_j \end{aligned}$$

- This is same as the batch version of Perceptron algorithm.

Perceptron

- A simple 'device': Weighted sum and threshold.
- A simple learning machine. (A neuron model).



- Also, we could use $\phi_i(X)$ for x_i for any fixed functions ϕ_i.
- Originally, Perceptron algorithm was proposed as a model of how we learn visual pattern recognition.
- The ϕ_i can be viewed as 'in-built' feature functions.
- The algorithm also needs only 'local' computations.
- This model is the first one of the so called neural networks models.

- Perceptron is an interesting algorithm to learn linear classifiers.
- Works only when data is linearly separable.
- In general, not possible to know beforehand whether data is linearly separable.
- We next look at other linear methods in classification and regression.

Regression Problems

- In a regression problem, the training set is $\{(X_i, y_i),\ i = 1, \cdots, n\}$ with $X_i \in \Re^d,\ y_i \in \Re,\ \forall i$.
- The goal is to learn a function, $f : \Re^d \to \Re$, that captures the relationship between X and y.
- For linear regression, the model is

$$f(X) = \sum_{j=1}^{d'} w_i \phi_i(X) \ + \ w_0$$

Linear Regression

- For simplicity we take $\phi_i(X) = x_i$ and consider the model

$$f(X) = \sum_{j=1}^{d} w_i x_i \; + \; w_0$$

- As earlier, by using an augumented vector X, we can write this as $f(X) = W^T X$.

Linear Least Squares Regression

- We want to find a W such that $\hat{y}(X) = f(X) = W^T X$ is a good fit for the training data.
- Define a function $J : \Re^{d+1} \to \Re$ by

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} \left(X_i^T W - y_i \right)^2$$

- We take the 'optimal' W to be the minimizer of $J(\cdot)$.
- Known as linear least squares method.

- We want to find W to minimize

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} \left(X_i^T W - y_i \right)^2$$

- If we are learning a classifier we can have $y_i \in \{-1, +1\}$.
- Note that finally we would use sign of $W^T X$ as the classifier output.
- Thus minimizing J is a good way to learn linear classifiers also.

- We want to find minimizer of

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} \left(X_i^T W - y_i\right)^2$$

- This is a quadratic function and we can analytically find the minimizer.
- For this we rewrite $J(W)$ into a more convenient form.

- Recall that we take all vectors to be column vectors.
- Hence each training sample X_i is a $(d+1) \times 1$ matrix.
- Let A be a matrix given by

$$A = [X_1 \; \cdots \; X_n]^T$$

- A is a $n \times (d+1)$ matrix whose i^{th} row is given by X_i^T.
- Hence, AW would be a $n \times 1$ vector whose i^{th} element is $X_i^T W$.
- Let Y be a $n \times 1$ vector whose i^{th} element is y_i.

- Hence $AW - Y$ would be a $n \times 1$ vector whose i^{th} element is $(X_i^T W - y_i)$.
- Hence we have

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} \left(X_i^T W - y_i\right)^2 = \frac{1}{2} (AW - Y)^T (AW - Y)$$

- To find minimizer of $J(\cdot)$ we need to equate its gradient to zero
- The gradient is

$$\nabla J(W) = A^T (AW - Y)$$

- We have
$$\nabla J(W) = A^T(AW - Y)$$
- Equating the gradient to zero, we get
$$(A^T A)W = A^T Y$$
- The optimal W satisfies this system of linear equations. (Called normal equations).

- $A^T A$ is a $(d+1) \times (d+1)$ matrix.
- $A^T A$ is invertible if A has linearly independent columns. (This is because null space of A is same as null space of $A^T A$).
- Rows of A are the training samples X_i.
- Hence j^{th} column of A would give the values of j^{th} feature in all the examples.
- Hence columns of A are linearly independent if no feature can be obtained as a linear combination of other features.
- This is a reasonable assumption.

- The optimal W is a solution of $(A^T A)W = A^T Y$.
- When $A^T A$ is invertible, we get the optimal W as

$$W^* = (A^T A)^{-1} A^T Y = A^\dagger Y$$

  where $A^\dagger = (A^T A)^{-1} A^T$, is called the generalized inverse of A.
- The above W^* is the linear least squares solution for our regression (or classification) problem.

- Even if $A^T A$ is not invertible, we can still find the solution to $(A^T A)W = A^T Y$ as $W^* = A^\dagger Y$ as follows.
- Since $A^T A$ is real symmetric, we have the eigen vector expansion $A^T A = V D V^T$ where V is an orhogonal matrix and D is the diagonal matrix of eigen values.
- Define diagonal matrix $D^\dagger$ by $D^\dagger_{ii} = \frac{1}{D_{ii}}$ if $D_{ii} \neq 0$ and is zero otherwise.
- Now we define $A^\dagger = V D^\dagger V^T A^T$ and $W^* = A^\dagger Y$.
- If $A^T A$ is invertible, then all $D_{ii} \neq 0$ and hence $D^\dagger = D^{-1}$ and hence $A^\dagger = (A^T A)^{-1} A^T$.

- Note that for any vector $\mathbf{q}$,

$$A^T A\mathbf{q} = VDV^T\mathbf{q} = \sum_i D_{ii}\mathbf{v}_i\mathbf{v}_i^T\mathbf{q} = \sum_{i:D_{ii}\neq 0} D_{ii}\mathbf{v}_i\mathbf{v}_i^T\mathbf{q}$$

  Thus, the column space of $A^T A$ is same as the span of those $\mathbf{v_i}$ corresponding to $D_{ii} \neq 0$.
- Denote $\mathbf{b} = A^T Y$.
- Note that $\mathbf{b}$ is in the column space of A^T and hence in the column space of $A^T A$.
- Hence we have

$$\mathbf{b} = \sum_{i:D_{ii}\neq 0} \mathbf{v}_i\mathbf{v}_i^T\mathbf{b}$$

- Recall

$$A^T A = VDV^T, \quad A^\dagger = VD^\dagger V^T A^T, \quad W^* = A^\dagger Y$$

- Now we have

$$A^T A W^* = A^T A A^\dagger Y = VDV^T VD^\dagger V^T \mathbf{b} = \sum_{i:D_{ii}\neq 0} \mathbf{v}_i \mathbf{v}_i^T \mathbf{b}$$

- This shows

$$A^T A W^* = \sum_{i:D_{ii}\neq 0} \mathbf{v}_i \mathbf{v}_i^T \mathbf{b} = \mathbf{b} = A^T Y$$

The generalized Inverse

- Our least squares method seeks to find a W to minimize $||AW - Y||^2$.
- A is a $n \times (d+1)$ matrix and normally $n >> d$.
- Consider the (over-determined) system of linear equations $AW = Y$.
- The system may or may not be consistent. But, we seek to find W^* to minimize squared error.
- As we saw, the solution is $W^* = A^\dagger Y$ and hence the name generalized inverse for $A^\dagger$.

Geometry of Least squares

- The least squares method is trying to find a 'best-fit' W for the systems $AW = Y$.
- If Y is in the column space of A, there is an exact solution.
- Otherwise, we want the projection of Y onto the column space of A.
- That is, we want to find a vector Z in the column space of A that is closest to Y.
- Hence we want to find Z to minimize $||Z - Y||^2$ subject to the constraint that $Z = AW$ for some W.
- That is the least squares solution.

- Let us take the original (and not augumented) data vectors and write our model as

$$\hat{y}(X) = f(X) = W^T X + w_0 \;\; \text{where now} \;\; W \in \Re^d$$

- Now we have

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} \left(W^T X_i + w_0 - y_i\right)^2$$

- For any given W we can find best w_0 by equating the partial derivative to zero.

- Setting the partial derivative to zero

$$
\begin{aligned}
\tfrac{\partial J}{\partial w_0} = \textstyle\sum_{i=1}^{n} \left(W^T X_i + w_0 - y_i\right) &= 0 \\
\Rightarrow \qquad nw_0 \; + \; W^T \textstyle\sum_{i=1}^{n} X_i \qquad &= \sum_{i=1}^{n} y_i
\end{aligned}
$$

- This gives us

$$
w_0 = \frac{1}{n}\sum_{i=1}^{n} y_i \; - \; W^T \left(\frac{1}{n}\sum_{i=1}^{n} X_i\right)
$$

- Thus, w_0 accounts for difference in the average of $W^T X$ and average of y.
- w_0 is often called the bias term.

- We have taken our linear model to be

$$\hat{y}(X) = f(X) = \sum_{j=0}^{d} w_j \, x_j$$

- As mentioned earlier, we could instead choose any fixed set of basis functions ϕ_i.
- Then the model would be

$$\hat{y}(X) = f(X) = \sum_{j=0}^{d'} w_j \, \phi_j(X)$$

- We can learn W using the same method as earlier.
- Thus, we will again have

$$W^* = (A^T A)^{-1} A^T Y$$

- The only difference is that now the i^{th} row of matrix A would be

$$[\phi_0(X_i)\ \phi_1(X_i)\ \cdots\ \phi_{d'}(X_i)]$$

- As an example: Let $d = 1$. (Then $X_i, y_i \in \Re$).
- Take $\phi_j(X) = X^j, \ j = 0, 1, \cdots, m$.
- Now the model is

$$\hat{y}(X) = f(X) = w_0 \ + \ w_1 X \ + \ w_2 X^2 \ + \ \cdots \ + \ w_m X^m$$

- The model is: y is a m^{th} degree polynomial in X.
- All such problems are tackled in a uniform fashion using the least squares method we presented.

LMS algorithm

- We are finding W^* that minimizes

$$J(W) = \frac{1}{2}\sum_{i=1}^{n}\left(X_i^T W - y_i\right)^2$$

- We could have found the minimum through an iterative scheme using gradient descent.
- The gradient of J is given by

$$\nabla J(W) = \sum_{i=1}^{n} X_i \left(X_i^T W - y_i\right)$$

- The iterative gradient descent scheme would be

$$W(k+1) \;=\; W(k) \;-\; \eta \sum_{i=1}^{n} X_i \;\left(X_i^T W(k) - y_i\right)$$

- In analogy with what we saw in Perceptron algorithm, this can be viewd as a 'batch' version.
- We use the current W to find the errors on all training data and then do all the 'corrections' together.
- We can instead have an incremental version of this algorithm.

The LMS Algorithm

- For the incremental version, at each iteration we pick one of the training samples. Call this $X(k)$.
- The error on this sample: $\frac{1}{2}(X(k)^T W(k) - y(k))^2$.
- Using the gradient of only this, we get the incremental version as

$$W(k+1) \;=\; W(k) \;-\; \eta\, X(k)\, (X(k)^T W(k) - y(k))$$

- This is called the LMS algorithm.

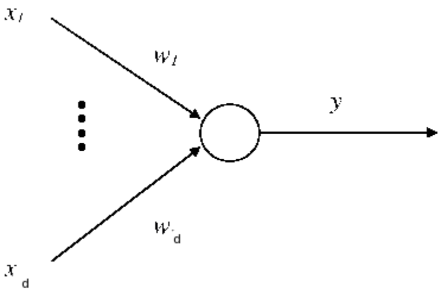
- The LMS algorithm is

$$W(k+1) \;=\; W(k) \;-\; \eta\, X(k)\, \left(X(k)^T W(k) - y(k)\right)$$

- This is very similar to Perceptron algorithm.
- If $y(k) \in \{0,\, 1\}$ and if we use thresholded version of $X(k)^T W(k)$ in the above what we get is exactly the Perceptron algorithm.
- This is also a classical algorithm.

Adaline

- We can view this as a unit similar to Perceptron
- Output is weighted sum of inputs.
- Called Adaline (ADaptine LINear Element); weights are adapted (Widrow 1963).



A simple example

- Let $s(n),\ n = 1, 2, \cdots$ be a signal.
- We want a model: $\hat{s}(n) = \sum_{j=1}^{m} w_j\, s(n-j)$
- Such models are useful in, e.g., speech coding, compression etc.
- We can think of the 'feature vector' at k to be $X(k) = (s(k-1),\ s(k-2),\ \cdots,\ s(k-m))^T$.
- Now we want to find (or adapt) the weights, $w_j, j = 1, \cdots, m$, so that $\hat{s}(n)$ would be a good estimate for $s(n)$.
- We can use linear least squares estimation with LMS algorithm.

- So far we have not used any statistical view point.
- We are finding the best linear model for the given data (according to the criterion of minimizing sum of squares of errors).
- But we can also look at it from a statistical view point.

- The least square error criterion is same as minimizing

$$J(W) \;=\; \frac{1}{2}\,\frac{1}{n}\,\sum_{i=1}^{n}\,\left(X_i^T W - y_i\right)^2$$

- Assuming the training examples to be drawn *iid*, the above is a good approximation of

$$J(W) \;=\; \frac{1}{2}\,E\left[(X^T W - y)^2\right]$$

- That is, the objective is to minimize mean squred error.

- Equating the gradient of $J(W)$ to zero we get

$$E\left[X(X^TW - y)\right] = 0 \Rightarrow E[XX^T]W = E[Xy]$$

- This gives us the optimal W^* as

$$W^* \; = \; (E[XX^T])^{-1}\,(E[Xy])$$

- The earlier expression we have for W^* would be same as this if we approximate expectations by sample averages.

- Since rows of A are X_i, we have

$$A^T A = \sum_{i=1}^{n} X_i X_i^T \approx nEXX^T$$

- Similarly $A^T Y = \sum_{i=1}^{n} X_i \, y_i \approx nE[Xy]$.
- Thus we have

$$(A^T A)^{-1} A^T Y \approx (nE[XX^T])^{-1} \, (nE[Xy])$$

- We are fitting a W to minimize $\frac{1}{2}E\left[(W^TX-y)^2\right]$.
- The gradient descent on this objective would be

$$W(k+1) = W(k) - \eta E[X(X^TW-y)]$$

- However, we cannot calculate the expectation.
- We have *iid* training samples.
- We can evaluate only the 'noisy' gradient at any sample.

LMS and Stochastic Gradient Descent

- The gradient descent on mean square error is

$$W(k+1) = W(k) - \eta E[X(X^T W - y)]$$

- The Stochastic gradient descent on this would be

$$W(k+1) = W(k) - \eta X(k) \left(X(k)^T W(k) - y(k) \right)$$

  where random sample at k^{th} iteration is $(X(k), y(k))$.
- This is the LMS algorithm.
- We use the 'noisy' gradient. (Same Robbins-Munro algorithm).

- Least squares method of fitting a model tries to find a function f to minimize

$$R(f) = E[(f(X) - y)^2]$$

- Since we are learning only linear models here, the minimization is only over all f that are linear (or affine) functions. Hence we minimize

$$J(W) = E[(W^T X - y)^2]$$

- In general, we can find the best f among **all** possible functions.

- This is a problem of approximating a random variable y as a function of another random variable X in the sense of least mean square error:

$$\min_f \; E[(f(X) - y)^2]$$

- If f^* is the optimal function, then $f^*(X)$ is called the regression function of y on X.
- We can show that this f^* is given by

$$f^*(X) \;=\; E[y \,|\, X]$$

  That is, f^* is the conditional expectation of y given X.

Conditional Expectation

- We need some properties of conditional expectation in the proof.
- $E[g(Z) \mid X]$ is a random variable and is a function of X.
- For random variables, X, Z with a joint density

$$E[g(Z) \mid X = x] \; = \; \int g(z) \, f_{Z|X}(z|x) \, dz$$

  where $f_{Z|X}(z|x)$ is the conditional density.
- If Z is discrete random variable

$$E[g(Z) \mid X = x] \; = \sum_j g(z_j) \, P[Z = z_j \mid X = x]$$

Conditional Expectation

- $E[g(Z) \mid X]$ is a random variable and is a function of X.
- It has all the linearity properties of expectation.

Two important special properties of conditional expectation are:

(i) $E\,[\,E[Z \mid X]\,] \;=\; E[Z], \;\; \forall Z, X$

(ii) $E[g(Z)\,h(X) \mid X] \;=\; h(X)\,E[g(Z) \mid X], \; \forall g, h$

- We will need both these properties for our proof.

- We want to show that for all f

$$E\left[(E[y \mid X] - y)^2\right] \;\leq\; E\left[(f(X) - y)^2\right]$$

We have

$$\begin{aligned}
(f(X) - y)^2 &= \left[(f(X) - E[y \mid X]) + (E[y \mid X] - y)\right]^2 \\
&= (f(X) - E[y \mid X])^2 \;+\; (E[y \mid X] - y)^2 \\
&\qquad + 2(f(X) - E[y \mid X])(E[y \mid X] - y)
\end{aligned}$$

Now we can take expectation on both sides.

First consider the last term

$$
\begin{aligned}
& E\left[(f(X) - E[y \mid X])(E[y \mid X] - y)\right] \\
= \; & E\left[\; E\left\{(f(X) - E[y \mid X])(E[y \mid X] - y) \mid X\right\} \;\right] \\
& \qquad\qquad \text{because} \quad E[Z] = E[\,E[Z|X]\,] \\
= \; & E\left[\; (f(X) - E[y \mid X]) \;\; E\left\{(E[y \mid X] - y) \mid X\right\} \;\right] \\
& \qquad\qquad \text{because} \quad E[g(X)h(Z)|X] = g(X)\, E[h(Z)|X] \\
= \; & E\left[\; (f(X) - E[y \mid X]) \;\; \left(E\left\{(E[y \mid X])|X\right\} \;-\; E\{y \mid X\}\right) \;\right] \\
= \; & E\left[\; (f(X) - E[y \mid X]) \;\; (E[y \mid X] \;-\; E[y \mid X]) \;\right] \\
= \; & 0
\end{aligned}
$$

Hence we get

$$\begin{aligned} E\left[\, (f(X) - y)^2 \,\right] &= E\left[\, (f(X) - E[y \mid X])^2 \,\right] \\ &\quad + E\left[\, (E[y \mid X] - y)^2 \,\right] \\ &\geq E\left[\, (E[y \mid X] - y)^2 \,\right] \end{aligned}$$

- Since the above is true for all functions f, we get

$$f^*(X) = E\,[y \mid X]$$

- We showed that if we want to predict y as a function of X to minimize $E[\ (f(X) - y)^2\ ]$, then the optimal function is

$$f^*(X)\ =\ E\,[y\,|\,X]$$

- Suppose $y \in \{0,\ 1\}$. Then

$$f^*(X)\ =\ E\,[y\,|\,X]\ =\ P[y = 1\,|\,X]\ = q_1(X)$$

- It is easy to see that, if $y \in \{-1,\ 1\}$ then $f^*(X) = 2q_1(X) - 1$.

- In a 2-class classification problem, suppose we learnt W to minimize

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} (X_i^T W - y_i)^2$$

- If we had $y \in \{0,\ 1\}$, then we learn a best linear approximation to the posterior probability, $q_1(X)$.
- Linear least squares is learning the conditional distribution of y given X.
- Since $X^T W$ is approximating $q_1(X)$,, by thresholding $X^T W^*$ at 0.5, we get a good classifier.
- If we had $y \in \{-1,\ 1\}$ then we learn a good linear approximation to $2q_1(X) - 1$.
- Hence we can threshold $X^T W^*$ at zero to get a good classifier.

- In most applications, our observations or data would be noisy.
- We can take the X_i to be fixed and the observed y_i to be random.
- Often, we get data by measuring y_i for specific value of X_i. Hence this is a useful scenario.
- Now the W^* obtained through linear least squares regression would also be random.
- Hence we would like to know its variance.

- We assume that noise corrupting different y_i are iid and zero-mean.
- Recall that Y is a vector random variable with components y_i.
- By our assumption,

$$\mathsf{Var}(y_i) = \sigma^2; \quad \mathsf{Cov}(y_i, y_j) = 0, i \neq j$$

- Hence the covariance matrix of Y is

$$\Sigma_Y = \sigma^2 \, I$$

where I is the identity matrix and σ^2 is noise variance.

- For any random vectors, Z, Y,
  if $Z = B\,Y$ for some matrix B then,

$$
\begin{aligned}
\Sigma_Z &= E[(Z - EZ)(Z - EZ)^T] \\
&= BE[(Y - EY)(Y - EY)^T]B^T = B\,\Sigma_Y\,B^T
\end{aligned}
$$

- We have $W^* = (A^T A)^{-1} A^T Y$. Hence

$$
\Sigma_W \;=\; (A^T A)^{-1} A^T\,\sigma^2\,I A (A^T A)^{-1} = \sigma^2\,(A^T A)^{-1}
$$

- This gives us the covariance matrix of the least squares estimate.

- Suppose X, y are related by $y = W^T X + \xi$ where ξ is a zero mean noise.
- Then we expect linear least squares method to easily learn W.
- The final mean sqare error would be variance of ξ.
- Using this idea, we can think of the least squares method as an ML estimation procedure under a reasonable probability model.

- Let y be a random variable, function of X.
- We take the probabilty model for y as

$$f(y \mid X, W, \sigma) \;=\; \frac{1}{\sigma\,\sqrt{2\pi}}\,\exp\left(-\,\frac{1}{2}\,\frac{(y - W^T X)^2}{\sigma^2}\right)$$

  where W and σ are the parameters.
- Let $\mathcal{D} = \{y_1(X_1),\ \cdots,\ y_n(X_n)\}$ be the *iid* data.
- We want to derive the ML estimate for the parameters.
- We are using ML estimation to learn a discriminative model here.

- The data likelihood is

$$\begin{aligned} L(W, \sigma \mid \mathcal{D}) &= \prod_{i=1}^{n} f(y_i \mid X_i, W, \sigma) \\ &= \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \frac{(y_i - X_i^T W)^2}{\sigma^2} \right) \end{aligned}$$

- The log likelihood is

$$l(W, \sigma \mid \mathcal{D}) = n \ln \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - X_i^T W)^2$$

- The log likelihood is given by

$$l(W, \sigma \mid \mathcal{D}) \;=\; n \,\ln \frac{1}{\sigma\sqrt{2\pi}} \;-\; \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - X_i^T W)^2$$

- Maximizing log-likelihood is same as minimizing squares of errors.
- Equating gradient of log likelihood to zero, we get

$$\sum_{i=1}^{n} X_i \,(y_i - X_i^T W) \;=\; 0$$

- This gives us the same W as least squares.

- Supoose we want ML estimate of σ also.

$$\frac{\partial l}{\partial \sigma} = -\frac{n}{\sigma} - \frac{-2}{2\sigma^3} \sum_{i=1}^{n} (y_i - X_i^T W)^2 = 0$$

- This gives us

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - X_i^T W)^2$$

- This is the residual average squared error.
- Thus linear least squares is the ML estimate (for the discriminative model) under the assumption of Gaussian noise.

- The Gaussian noise assumption is alright for a regression problem.
- In a 2-class classification problem, where $y \in \{0,\ 1\}$, Gaussian noise does not make sense.
- So, we will investigate a different model for the classification problem.