UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INFORMATION ENGINEERING DEPARTMENT

COMPUTER ENGINEERING - AI & ROBOTICS

Machine Learning

Francesco Crisci 2076739

Model Notions

Loss

We define the loss over a distribution D and a function f of an hypothesis h as: $L_{D,f}(h) = P_{x\sim D}[h(x) \neq f(x)] = D(\{x : h(x) \neq f(x)\})$.

Training Error

We define the training error over the training set S, with $|S| = m$ and $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ over an hypothesis h as: $L_S(h) = \frac{|\{i:h(x_i)\neq y_i \; 1\leq i\leq n\}|}{m}$

Empirical risk minimization

Let H be the hypothesis class, we define the ERM as: $ERM_H \in argmin_{h\in H}(L_S(h))$. It follows that the realizability assumption is: $\exists h^* \in H \; s.t \; L_{D,f}(h^*) = 0$ and that the samples in S are i.i.d. according to D, that is $S \sim D^m$.

Simplified PAC learning

Let us define two values:

- Accuracy ϵ: we are satisfied with a good h_s s.t. $L_{D,f}(h_s) \leq \epsilon$, with ϵ small;
- confidence δ: we want h_s to be a good hypothesis with probability $1 - \delta$, with δ small.

Corollary 2.3

Given $H < +\infty, \delta \in (0, 1), \epsilon \in (0, 1), |S| = m \in N$s.t. $m \geq \frac{ln(|H|/\delta)}{\epsilon}$, then, for any f and D for which the realizability assumption holds, with probability $1 - \delta$, we have, that, for every ERM hypothesis h_S, it holds that $L_{D,f}(h_S \leq \epsilon)$. In other words: if $m \geq \frac{ln(|H|/\delta)}{\epsilon} \Rightarrow L_{D,f}(h_S) \leq \epsilon$.
If i have enough data, I can always find a good hypothesis with probability $1 - \delta$.

Poof Corollary 2.3

Let $S|_x = \{x_1, ..., x_m\}$ be instances of S. Let $H_B = \{h \in H : L_{D,f} > \epsilon\}$be the bad hypothesis, and M=$\{S|_x : \exists h \in H, L_S(h = 0)\}$ be the misleading samples.
Due to the realizability assumption we have that $L_S(h) = 0 \Rightarrow L_{D,f}(h_S) > \epsilon$ only if $h \in H_B$ has $L_S(h) = 0$. That is, our training data must be in the set M: $\{S|_x : L_{D,f}(h_S) > \epsilon\} \subseteq M$.
Note that M=$\cup_{h\in H_B}\{S|_x : L_S(h) = 0\}$. Therefore: $D^m(\{S|_x : L_{D,f}(h_s) > \epsilon\}) \leq D^m(M) = D^m(\cup_{h\in H_B}\{S|_x : L_S(h) = 0\})$. Which is, due to the union bound $\leq \Sigma_{h\in H_B} D^m(\{S|_x : L_S(h) = 0\})$ $(*)$
Let's fix $h \in H_B : L_S(h) = 0 \Leftrightarrow h(x_i) = f(x_i) \forall I = 1, ..., m$. Therefore: $D^m(\{S|_x : h(x_i) = f(x_i) \forall i = 1, ..., m\})$. Since $x_1, ..., x_m$ are i.i.d. from D, we have that the previous quantity is equal to $\Pi_{i=1}^m D(\{S|_x : h(x_i) = f(x_i) \forall i = 1, ..., m\})(**)$.
Consider some i, $1 \leq i \leq m : D(\{x_i : h(x_i) = f(x_i)\}) = 1 - D(\{x_i : h(x_i) \neq f(x_i)\})$
$= 1 - L_{D,f}(h) \leq 1 - \epsilon \leq e^{-\epsilon}$ (due to Taylor expansion). (We have used the fact that $L_{D,f}(h) = D(\{x_i : h(x_i) \neq f(x_i)\}) = P_{x\sim D}[h(x) \neq f(x)]$).
By combining this last quantity with $(**)$, we obtain that $D^m(\{S|_x : L_S(h) = 0\}) \leq \Pi_{i=1}^m e^{-\epsilon} =$

$e^{-m\epsilon}$.
By combining it with $(*)$, we obtain: $D^m(\{S|_x : L_{D,f}(h_S) > \epsilon\})\Sigma_{h\in H_B} e^{-m\epsilon} = |H_B|e^{-m\epsilon} \leq |H|e^{-m\epsilon}(***)$.
By choosing m, we obtain that $(***) \leq |H|e^{-\epsilon log(|H|/\delta)1/\epsilon} = |H|\frac{\delta}{|H|} = \delta$.

PAC learnability

H is learnable if there exists a function $m_H : (0,1)^2 \to N$ and a learning algorithm s.t. for every $\epsilon, \delta \in (0,1)$, for every D over X, and for every function $f : X \to \{0,1\}$, if the realizability assumption holds with respect to H,D,f, then when running the algorithm on $m \geq m_H(\epsilon,\delta)$ i.i.d samples generated by D and labeled by f, the algorithm returns an hypothesis h s.t., with probability $\geq 1-\delta : L_{D,f} \leq \epsilon$. m_H is the minimal integer which satisfies the requirements.

Corollary

Every finite hypothesis class is PAC learnable with sample complexity $m_H \leq \lceil \frac{log(|H|/\delta)}{\epsilon} \rceil$.
We can define the Bayes optimal predictor as:

$$f_D(x) = \begin{cases} 1 \ if \ P[y=1|x] \geq 0.5 \\ 0 \ otherwise \end{cases}$$

for any classifier g:$X \to \{0,1\}$, it holds that $L_D(f_D) \leq L_D(g)$.

Agnostic PAC learnability

H is agnostic PAC learnable if there exists a function $m_H : (0,1)^2 \to N$ and a learning algorithm s.t. for every $\epsilon, \delta \in (0,1)$, for every D over XxY, when running the algorithm on $m \geq m_H(\epsilon,\delta)$ i.i.d samples generated by D, the algorithm returns an hypothesis h s.t., with probability $\geq 1-\delta$:

$$L_D(h) \leq min_{h' \in H} L_D(h') + \epsilon$$

Generalized loss function

Given an hypothesis H and some domain Z, a loss function is any function $l : HxZ \to R_+$. We can define the generalized error as $L_D(h) = E_{z\sim D}[l(h,z)]$, and the empirical risk as $L_S(h) = \frac{1}{m}\Sigma_{i=1}^m l(h,z_i)$.

Agnostic PAC learnability for general model

As before, but $L_D(h) = E_{z\sim D}[l(h,z)]$, $l : HxZ \to R_+$: $L_D(h) \leq min_{h' \in H} L_D(h') + \epsilon$.

Linear models

Linear affine function

$$L_d = \{h_{\vec{w},b} : \vec{w} \in R^d, b \in R\}$$
$$h_{\vec{w},b} = <\vec{w},\vec{x}> + b = (\Sigma_{i=0}^d w_i, x_i) + b$$

Hypothesis class

$H : \Phi \circ L_d$, where $\Phi : R \to Y$, $h \in H$ is $h : R \to Y$. Φ can be used for binary classification, so $Y = \{-1, 1\} \Rightarrow \Phi(z) = sign(z)$; or for regression, where $Y = R \Rightarrow \Phi(z) = z$.

Equivalent notation

We can rewrite the weight vector and the samples vector as: $\vec{w'} = (b, w_1, ..., w_d)$ and $\vec{x'} = (1, x_1, ..., x_d)$.

Linear classification

We consider half-spaces $HSd = sign \circ L_d\{\vec{x} \to sign(h_{\vec{w},x}(\vec{x})) : h_{\vec{w},x} \in L_d\}$. If the inner product between x and w is positive we assign the class 1 to the point, otherwise we assign the class -1. We have a line that divides the data.
We say that the data are linearly separable if $\exists \vec{w}$ s.t. $y_i < \vec{w}, \vec{x_i} >> 0$.

Perceptron

The aim of the perceptron is to modify the line if the point $\vec{x_i}$ has been wrongly classified. The pseudo-code is the following:
Input: $S = \{(x_1, y_1), ..., (x_m, y_m)\}$
Initialize $w^{(1)} = \vec{0}$
For t=1,... do:
    $if \exists i \; s.t. \; y_i < \vec{w}^{(t)}, \vec{x_i} >\leq 0$ then $w^{(t+1)} \leftarrow w^{(t)} + y_i \vec{x_i}$;
    $else \; return \; w^(t)$;
Notice that $y_i < \vec{w}^{(t+1)}, \vec{x_i} >= y_i < \vec{w}^{(t)}, y_i \vec{x_i} \vec{x_i} >= y_i < \vec{w}^{(t)}, \vec{x_i} > + ||\vec{x_i}||^2$.

Proposition

Assume that S is linearly separable and let $B = min\{||w|| : y_i < \vec{w}, \vec{x_i} >\geq 1 \forall i = 1, ..., m\}$, and let $R = max_i ||\vec{x_i}||$, Then the perceptron stops at most $(RB)^2$ iterations, and it will still hold that $y_i < \vec{w}^{(t)}, \vec{x_i} >\leq 0$.

Gradient Descent

The gradient $\nabla f(\vec{w})$ of f at $\vec{w} = (w_1, ..., w_d)$ is $\nabla f(\vec{w}) = (\frac{\delta f(\vec{w_1})}{\delta w_1}, ..., \frac{\delta f(\vec{w_d})}{\delta w_1})$. let $\eta \in R, \eta > 0$ be a parameter, we can define the update of the perceptron as:
$\vec{w}^{(0)} \leftarrow \vec{0}$
$for \; t \leftarrow 0 \; to \; T - 1 \; do$:
    $\vec{w}^{t+1} \leftarrow \vec{w}^{(t)} - \eta \nabla f(\vec{w}^{(t)})$
return $\vec{w} = 1/T \Sigma_{t=1}^{T} \vec{w}^{(t)}$

Stochastic Gradient Descent

Instead of using exactly the gradient, we take a random vector with expected value equal to the gradient direction. The perceptron algorithm, then, becomes:
$\vec{w}^{(0)} \leftarrow \vec{0}$

$for\ t \leftarrow 0\ to\ T\ do$:
chose $\vec{V_t}$ at random form distribution s.t. $\vec{E}[\vec{V_t}|\vec{w}^{(t)}] \in \nabla f(\vec{w}^{(t)})$;
It means that V_t has the expected value equals to the gradient of $f(\vec{w}^{(t)})$
$\vec{w}^{t+1} \leftarrow \vec{w}^{(t)} - \eta \nabla f(\vec{w}^{(t)})$
return $\vec{w} = 1/T\Sigma_{t=1}^{T}\vec{w}^{(t)}$

SGD for linear classification

SGD: take i uniformly at random from $\{1, ..., m\}$. Let $(\vec{x'}, y')$ be the corresponding point in the training set, and consider the vector $\nabla l(\vec{w}, (\vec{x'}, y'))$. Notice that $\nabla L_S(\vec{w}) = \frac{1}{m}\Sigma_{i=1}^{m}\nabla l(\vec{w}, (\vec{x}, y))$.
$E[\nabla l(\vec{w}, (\vec{x'}, y'))] = \Sigma_{i=1}^{m} P[(\vec{x'}, y') = (\vec{x_i}, y_i)]\nabla l(\vec{w}, (\vec{x_i}, y_i)) =$
$= \frac{1}{m}\Sigma_{i=1}^{m}\nabla l(\vec{w}, (\vec{x}, y)) = \nabla L_S(\vec{w})$
The SGD algorithm then becomes:
$\vec{w}^{(0)} \leftarrow \vec{0}$
$for\ t \leftarrow 0\ to\ T-1\ do$:
$\vec{w}^{t+1} \leftarrow \vec{w}^{(t)} - \eta \nabla l(\vec{w}, (\vec{x_i}, y_i))$
return $\vec{w} = 1/T\Sigma_{t=1}^{T}\vec{w}^{(t)}$
Where $\nabla l(\vec{w}, (\vec{x_i}, y_i)) = \begin{cases} \vec{0}\ if\ y_i < \vec{w}, \vec{x_i} >> 0 \\ \nabla(-y_i < \vec{w}, \vec{x_i} >)\ otherwise \end{cases}$
Assume that $y_i < \vec{w}, \vec{x_i} >< 0$, then:
$\nabla(-y_i < \vec{w}, \vec{x_i} >) = [\frac{\delta((-y_i < \vec{w}, \vec{x_i} >))}{\delta w_1}, ..., \frac{\delta((-y_i < \vec{w}, \vec{x_i} >))}{\delta w_d}]^T$.
Let $\vec{x_i} = \begin{bmatrix} x_{i1} \\ ... \\ x_{id} \end{bmatrix}$. Since $-y_i < \vec{w}, \vec{x_i} >= -y_i\Sigma_{j=1}^{d}(w_j x_{ij})$, we then have that

$$\nabla l(\vec{w}, (\vec{x_i}, y_i)) = [-y_i x_{1i}, ..., -y_i x_{id}]^T = -y_i\vec{x_i}$$

Therefore the new SGD algorithm becomes:
$\vec{w}^{(0)} \leftarrow \vec{0}$
$for\ t \leftarrow 0\ to\ T-1\ do$:
$\vec{w}^{t+1} \leftarrow \vec{w}^{(t)} - \eta y_i \vec{x_i}$
return $\vec{w} = 1/T\Sigma_{t=1}^{T}\vec{w}^{(t)}$

Linear Regression

In linear regression we consider: $H_{reg} = L_d = \{\vec{x} \rightarrow < \vec{w}, \vec{x} > +b : \vec{w} \in R^d, b \in R\}$, the squared loss: $l(h, (\vec{x}, \vec{y})) = (h(\vec{x}) - \vec{y})^2$, and the empirical risk as: $L_S(h) = \frac{1}{m}\Sigma_{i=1}^{m}(h(\vec{x_i}) - \vec{y_i})^2$.

ERM for linear regression

$argmin_{\vec{w}} L_S(h_{\vec{w}}) = argmin_{\vec{w}}\Sigma_{i=1}^{m}(h(\vec{x_i}) - \vec{y_i})^2$, which is the residual sum of squares(RSS).

Matrix notation

In matrix form we have that: $\vec{x} = \begin{bmatrix} ... & \vec{x_1} & ... \\ ... & & \\ ... & \vec{x_m} & ... \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} y_1 // ... // y_m \end{bmatrix}$. The RSS becomes then $(\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$, so our aim is to have $argmin_{\vec{w}}(\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$. Then, we can obtain the solution to the argmin through: $\frac{\delta RSS}{\delta w} = -2X^T(\vec{y} - X\vec{w})$, and if we impose it to 0 we obtain $\vec{w} = (X^TX)^{-1}X^T\vec{y}$. The solution can be obtained if and only if (X^TX) is invertible. If it is not, we can use the generalized inverse, defined as $A^+ = VD^+V^T$, where $AA^+A = A$ and $A = VDV^T$.

Logistic Regression

We consider $H = \Phi_{sig} \circ L_d$, where $\Phi_{sig} : R \to [0, 1]$ and $\Phi_{sig}(z) = \frac{1}{1+e^{-z}}$ and $Y = \{-1, 1\}$. If the predicted value from h is equal to 1 we have an high confidence that the label is 1; if the predicted value is 0, we have high-confidence that the label is -1. If the predicted value is 0.5, we cannot say anything about the predicted value. We can now define

$$h_{\vec{w}}(\vec{x}) = \frac{1}{1+e^{-<\vec{w},\vec{x}>}}$$

which is large if y=1, small if y=-1. The complementary of the function is:

$$1 - h_{\vec{w}}(\vec{x}) = \frac{1}{1+e^{<\vec{w},\vec{x}>}}$$

We can now define the loss as $l(h_{\vec{w}}, (\vec{x}, y)) = log(1 + e^{<\vec{x},\vec{y}>})$.

Maximum Likelihood estimation (MLE)

We define the likelihood as $P[S|\theta]$, where $L(S, \theta) = log(P[S|\theta])$ is the log likelihood. The maximum likelihood estimator is $\hat{\theta} = argmax_\theta L(S, \theta)$.

Logistic Regression and MLE

If y=1, we have that $h_{\vec{w}}(\vec{x_i}) = \frac{1}{1+e^{-<\vec{w},\vec{x_i}>}}$, or if y=-1 $1 - h_{\vec{w}}(\vec{x_i}) = \frac{1}{1+e^{<\vec{w},\vec{x_i}>}}$.
Therefore the Likelihood of S is $\Pi_{i=1}^m \frac{1}{1+e^{-y_i<\vec{w},\vec{x_i}>}}$. Therefore the log likelihood is
$log(\Pi_{i=1}^m \frac{1}{1+e^{-y_i<\vec{w},\vec{x_i}>}}) = \Sigma_{i=1}^m log(\frac{1}{1+e^{-y_i<\vec{w},\vec{x_i}>}})$.
Which can be rearranged as: $\Sigma_{i=1}^m(log(1) - log(e^{-y_i<\vec{w},\vec{x_i}>}))$, which leads us to
$-\Sigma_{i=1}^m log(1 + e^{-y_i<\vec{w},\vec{x_i}>})$.
Therefore the maximum likelihood estimator is:
$argmax_{\vec{w}\in R^d} - \Sigma_{i=1}^m log(1 + e^{-y_i<\vec{w},\vec{x_i}>}) = argmin_{\vec{w}\in R^d}\Sigma_{i=1}^m log(1 + e^{-y_i<\vec{w},\vec{x_i}>})$
which tells us that the MLE is equivalent to ERM.

Uniform Convergence

The empirical risk of all members of H are good approximations of their true risk. An hypothesis class h has the uniform convergence property if there exists a function $m_H^{UC} : (0.1)^2 \to N$, such that for every $\epsilon, \delta \in (0, 1)$ and for every probability distribution D over Z, if S is a sample of $m \geq m_H^{UC(\epsilon,\delta)}$ i.i.d. samples drawn from D, then we probability $\geq 1 - \delta$, S is ϵ-representative.

Proposition

If a class H has the uniform convergence property with a function m_H^{UC} then the class is agnostically PAC learnable with the sample complexity $m_H(\epsilon, \delta) \leq m_H^{UC}(\epsilon/2, \delta)$. Furthermore, in that case the ERM paradigm is a successful PAC learner for H.

ϵ-representative

A training set S is called ϵ-representative if

$$\forall h \in H, |L_S(h) - L_D(h)| \leq \epsilon \Rightarrow L_S(h) - \epsilon \leq L_D(h) \leq L_S(h) + \epsilon$$

Proposition

Assume S is $\epsilon/2$-representative. Then any output of ERM(S) satisfies:

$$L_D(h_s) \leq min_{h \in H} L_D(h) + \epsilon$$

Proof

For any $h \in H$:
$L_D(h_S) \leq L_S(h_S) + \epsilon/2$ since S is $\epsilon/2$-representative.
$\leq L_S(h) + \epsilon/2$ since h_S is picked by ERM
$\leq L_D(h) + \epsilon/2 + \epsilon/2$ since S is $\epsilon/2$-representative.
$\leq L_D(h) + \epsilon$
That is true for $h = argmin_{h' \in H} L_D(h')$ as well $\Rightarrow L_D(h_S) \leq_{h \in H} L_D(h) + \epsilon$.

Proposition 4.6

Let H be a finite hypothesis class, let Z be domain and let $l : HxZ \rightarrow [0, 1]$ be a loss function. Then:

- H enjoys the uniform convergenze property with sample complexity $m_H^{UC}(\epsilon, \delta) \leq \lceil \frac{log(2|H|/\delta)}{2\epsilon^2} \rceil$
- H is agnostically PAC learnable using the ERM algorithm with sample complexity $m_H(\epsilon, \delta) \leq m_H^{UC}(\epsilon/2, \delta) \leq \lceil \frac{2log(2|H|/\delta)}{\epsilon^2} \rceil$

Proof

To prove it we need Hoeffding's inequality: Let $\theta_1, ..., \theta_m$ be a sequence of i.i.d. random variables and assume that for all i, $E[\theta_i] = \mu$ and $P[a \leq \theta_i \leq b] = 1$. Then for any $\epsilon > 0$:

$$P[|\frac{1}{m}\Sigma_{i=1}^m \theta_i - \mu| > \epsilon] \leq 2e^{-2m\epsilon^2/(b-a)^2}$$

Fix $\epsilon, \delta \in (0, 1)$. We need a sample size m, s.t., for any D, with probability $\geq 1 - \delta$, we have that for all $h \in H : |L_S(h) - L_D(h)| \leq \epsilon$. That is: $D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| \leq \epsilon\}) \geq 1 - \delta$. We need to show that $D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\}) < s\delta$, where the first part of the inequality is $(*)$. We have that $\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\} = \cup_{h \in H}\{S : |L_S(h) - L_D(h)| > \epsilon\}$. Then $(*) \leq \Sigma_{h \in H} D^m(\{S : |L_S(h) - L_D(h)| > \epsilon\})(**)$ by union bound. We want to bound each term in $(**)$. Therefore $E[L_S(h)] = E[\frac{1}{m}\Sigma_{i=1}^m l(h, z_i)] = L_D(h)$.

Let θ_i be the random variable given by $l(h, z_i)$. Since h is fixed, z_i are sampled i.i.d. from D, then all θ_i are i.i.d. random variables. Notice that $L_S(h) = \frac{1}{m}\Sigma_{i=1}^{m}\theta_i$; let's define $\mu = L_D(h)$. Given the assumption that $l : HxZ \to [0,1] \Rightarrow \theta_i \in [0,1] \forall i = 1, ..., m$.
We can apply Hoeffding's inequality with $a_i = 0, b_i = 1 \forall i = 1, ..., m$. Then:

$$D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\} \leq \Sigma_{h \in H} 2e^{-2m\epsilon^2} = 2|H|e^{-2m\epsilon^2}$$

By choosing $m \geq log(2|H|/\delta)\frac{1}{2\epsilon^2}$ then:

$$D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\} \leq 2|H|e^{-2\epsilon^2 log(2|H|/\delta)\frac{1}{2\epsilon^2}} = \delta.$$

No-free lunch theorem

Let A be any learning algorithm for the task of binary classification with respect to the 0-1 loss function over domain X. Let m be any smaller than $|X|/2$, representing a training set size. Then there exists a distribution D over $Xx\{0,1\}$ s.t.:

- there exists a function $f : X \to \{0,1\}$ with $L_D(f) = 0$
- with probability of at least 1/7 over the choice of $S \sim D^m$ we have that $L(A(S)) \geq 1/8$

In other words, H cannot be too large.

Corollary

Let X be an infinite domain set and let H be the set of all functions from X to $\{0,1\}$. Then H is not PAC learnable.

Error Decomposition

Let h_S be an ERM_H hypothesis. Then we can decompose the generalization error as $L_D(h_S) = L_D(h_S) - min_{h \in H} L_D(h) + min_{h \in H} L_D(h)$, where the first two terms are the estimation error, ϵ_{est}, while the last term is the approximation error, ϵ_{app}

Approximation Error

$\epsilon_{app} = min_{h \in H} L_D(h)$ derives from our choice of H; once we have chosen H it is inevitable and it decreases with a larger H.

Estimation Error

$\epsilon_{est} = L_D(h_S) - min_{h \in H} L_D(h)$ derives from our inability to choose the best hypothesis in H and could be avoided if we choose the best hypothesis. To decrease it we need a lower number of hypothesis in H so that the training error is a good estimate of the generalization error for all of them. In other words, we need a small H.
If the estimation error is small and the approximation is large, we have underfitting, low complexity and high inductive bias. if we go in the other direction we have the opposite.

Restrictions

Let H be a class of functions from X to $\{0,1\}$ and let $C = \{c_1, ..., c_m \subset X\}$. Then a restriction H_C of H to C is $H_C = \{h(c_1), ..., h(c_m) : h \in H\}$, where we represent each function from C to $\{0,1\}$ as a vector in $\{0,1\}^{|C|}$. If $|H| \Rightarrow 1 \leq |H_C| \leq 2^m$. And $|H_C|$ is a set of functions.

Shattering

Given $C \subset X$, H shatters C if H_C contains all $2^|C|$ functions from C to $\{0,1\}$.

VC-dimension

The VS-dimension of an hypothesis class H is the maximal size of a set $C \subset X$ that can be shattered by H. If H can shatter sets of arbitrarily large size then we say that VCdim(H)=$+\infty$; if $|H| < +\infty \Rightarrow VCdim(H) \leq log_2(|H|)$.
To show that the VCdim(H)=d we need to show that there exists a set C of size d which is shattered by H ($VCdim(H) \geq d$); and we need to show that every set of size d+1 is not shattered by H (VCdim(H)$\leq d$). The threshold function has a VCdim = 1, while the intervals have a VCdim=2.

Regularized Loss Minimization

Assume h is defined by $\vec{w} = (w_1, ..., w_d)^T \in R^d$. A regularization function $R : R^d \rightarrow R$, the regularization loss minimization is defined as: pick h obtained as $argmin_{\vec{w}}(L_S(\vec{w}) + R(\vec{w}))$, where R is a measure of complexity.

L1-regularization

Lets define $\lambda \in R^+/\{0\}$, which measures the tradeoff between the empirical risk and the complexity, and $||\vec{w}||_1 = \Sigma_{i=1}^d |w_i|$ is the L1 norm and measures the complexity. The learning rule then becomes: pick $A(S) = argmin_{\vec{w}}(L_S(h) + \lambda||\vec{w}||_1)$

LASSO

It is the combination of squared loss and L1 regularization. Pick $\vec{w} = argmin_{\vec{w}}\lambda||\vec{w}||_1 + \Sigma_{i=1}^m(< \vec{w}, \vec{x_i} > -y_i)^2$. It has no closed form solution! but since both L1 and squared loss are convex it can be solved efficiently.

Hikonov Regularization

We define $R(\vec{w} = \lambda||\vec{w}||_2)$, which is the L2 norm of the vector w. $||\vec{w}||_2 = \Sigma_{i=1}^d (w_i)^2$. We pick A(S)=$argmin_{\vec{w}}(L_S(h) + \lambda||\vec{w}||_2)$.

RIDGE regression

It is obtained combining linear regression, squared loss and thikonov regularization. We pick $\vec{w} = argmin_{\vec{w}}\lambda||\vec{w}||_2 + \Sigma_{i=1}^m(< \vec{w}, \vec{x_i} > -y_i)^2$. We can define the RSS($\vec{w}$): $\vec{w} = argmin_{\vec{w}} RSS(\vec{w}) = argmin_{\vec{w}}\Sigma_{i=1}^m(< \vec{w}, \vec{x_i} > -y_i)^2$.

The RIDGE regression pick: $\vec{w} = argmin_{\vec{w}} \lambda ||\vec{w}||_2 + \Sigma_{i=1}^{m} (< \vec{w}, \vec{x_i} > -y_i)^2$, which can be rearranged in matrix form as:

$$\vec{w} = argmin_{\vec{w}} (\lambda ||\vec{w}||_2 + (\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w}))$$

To minimize it we use the gradient of the function, obtaining:

$$2\lambda\vec{w} - 2X^T(\vec{y} - X\vec{w}) \Rightarrow \vec{w} = (\lambda I + X^T X)^{-1} X^T \vec{y}$$

which is the Ridge regression solution.

Fundamental Theorem of statistical learning

Let H be an hypothesis class of function from a domain X to $\{0, 1\}$ and consider the 0-1 loss function. Assume that VCdim(H)=d$< +\infty$. Then there are absolute constants C_1, C_2 s.t. :

- H has the uniform convergence property with sample complexity $C_1 \frac{d+log(1/\delta)}{\epsilon^2} \leq m_H^{UC}(\epsilon, \delta) \leq C_2 \frac{d+log(1/\delta)}{\epsilon^2}$
- H is agnostic PAC learnable with sample complexity $C_1 \frac{d+log(1/\delta)}{\epsilon^2} \leq m_H(\epsilon, \delta) \leq C_2 \frac{d+log(1/\delta)}{\epsilon^2}$

Theorem

Let H be an hypothesis class with VCdim(H)$< +\infty$. Then with probability $\geq 1 - \delta$ we have that: $\forall h \in H, L_D(h) \leq L_S(h) + C\sqrt{\frac{VCdim(H)+log(1/\delta)}{2m}}$, where $|S| = m$.

Theorem

Lt H be a class with $VCdim(H) = +\infty \Rightarrow H$ is not PAC learnable.

Validation

Pick an hypothesis, then use new data to estimate the true error. Let $V = \{(x_1, y_1), ..., (x_v, y_v)\}$ be a set of m_V samples, and let the validation error be $L_V(h) = \frac{1}{m_V} \Sigma_{i=1}^{m_V} l(h, (x_i, y_i))$.

Proposition

For every $\delta \in (0, 1)$, with probability $\geq 1 - \delta$ we have $|L_V(h) - L_D(h)| \leq \sqrt{\frac{log(2/\delta)}{2m_V}}$.

$L_D(h) \leq L_S(h) + \sqrt{C\frac{VCdim(H_d)+log(1/\delta)}{2m}}$.

$L_D(h) \leq L_V(h) + \sqrt{\frac{log(2/\delta)}{2m_V}}$

Where the first term is the measured data and the second term is a measure of uncertainty. With the validation test we pick the best h in ERM_H.

Proposition

with probability $\geq 1 - \delta$ over the choice of V, we have $\forall h \in \{h_1, ..., h_r\} : |L_D(h) - L_V(h)| \leq \sqrt{\frac{log(2r/\delta)}{2m_V}}$. If the training error decreases and validation error increases we are going towards overfitting.

Train-Validation-Test Split

We divide the data in 3 sets: train, validation and test. We use the train to learn the best model, the validation to pick the best hypothesis and the test to estimate the true risk.

k-fold cross validation

Partition the training set into k folds of size m/k. Choose 1 fold at a time: train over the union of the other folds and estimate the error from the chosen fold. Estimate the true error by using the average of the estimated errors. The pseudo-code of the leave-one-out cross validation is the following:
Input: S, set of parameters θ, learning algorithm A, int k.
Partition S in $S_1, ..., S_k$
for each $\theta \in \Theta$:
    for i=1,...,k:
        $h_{i,\theta} = A(S \setminus S_i; \theta)$;
        $error(\theta) = \frac{1}{k}\Sigma_{i=1}^{k} L_{S_i}(h_i, \theta)$;
output $\theta^* = argmin[error(\theta)]; h(\theta^*) = A(S; \theta^*)$;

If learning fails

Tune the parameters if you have them. If $L_S(h)$ is large then enlarge or change H, or change the feature representation. If $L_S(h)$ is small we have two cases: ϵ_{app} is small, so we can either employ more data of reduce the complexity of H; if ϵ_{est} is large, we can either change H or the feature representation.

Subset selection

we want to minimize the empirical error on $\vec{w}$: $min_{\vec{w}}$ subject to $||\vec{w}||_0 \leq k$. How can we find a solution? we can enumerate all subset features S with k elements different from 0, then learn the best model for each of the subset and in the end keep the subset of minimal error.
We define L=$\{1, ..., d\}$, $p = \{i_1, ..., i_k\} \subseteq L : H_p$ = hypothesis or models where only features $w_{i_1}, ..., w_{i_k}$ are used. The algorithm is the following:
$P^{(k)} \leftarrow \{J \subseteq L : |J| = k\}$
for each $p \in P^{(k)}$ do:
    $h_p \leftarrow argmin_{h \in H_P} L_S(h)$;
return $h^{(k)} \leftarrow argmin_{p \in P^{(k)}} L_S(h_P)$;
It is $\Theta(\binom{d}{k}) \in \Theta(d^k)$, which is a a bad solution.

Proposition

The optimization problem of feature selection is NP-hard.

Forward Selection

$Sol \leftarrow \emptyset$
while $|Sol| < k$ do:
    $foreach\ i \in L \setminus Sol$ do:
        $p \leftarrow Sol \cup \{i\}$;
        $h_P \leftarrow argmin_{h \in H_P} L_S(h)$;
    $Sol \leftarrow Sol \cup argmin_{i \in L \setminus Sol} L_S(h_{Sol \cup \{i\}})$;
return Sol;
Which is $\Theta(kd)$.

Backwards selection

Start with a solution containing all the features and then remove 1 feature at a time until $|Sol| = k$. It is $\Theta((d-k)d)$.

Subset selection with validation

for $l \leftarrow 0$ to k do:
    $P^{(l)} \leftarrow \{J \subseteq L : |J| = l\}$
    $foreach\ p \in P^{(l)}$:
        $h_P \leftarrow argmin_{h \in H_P} L_S(h)$;
    $h^{(l)} \leftarrow argmin_{p \in P^{(l)}} L_V(h_p)$;
return $argmin_{h \in \{h^{(0)}, ..., h^{(k)}\}} L_V(h)$;

Forward selection with validation

$Sol \leftarrow \emptyset$
while $|Sol| < k$ do:
    $foreach\ i \in L \setminus Sol$ do:
        $p \leftarrow Sol \cup \{i\}$;
        $h_P \leftarrow argmin_{h \in H_P} L_S(h)$;
    $Sol \leftarrow Sol \cup argmin_{i \in L \setminus Sol} L_V(h_{Sol \cup \{i\}})$;
return Sol;

SVM

S is linearly separable if there exists an halfspace s.t. $y_i = sign(< \vec{w}, \vec{x_i} > +b) \forall i = 1, ..., m$, which is equivalent to $y_i(< \vec{w}, \vec{x_i} > +b) > 0 \forall i = 1, ..., m$. Given an hyperplane defined by $L = \{\vec{v} :< \vec{w}, \vec{v} > +b = 0\}$, and given $\vec{x}$, the distance of $\vec{x}$ to L is: $d(\vec{x}, L) = min\{||\vec{x} - \vec{v}|| : \vec{v} \in L\}$. If $||\vec{w}|| = 1 \Rightarrow d(\vec{x}, L) = | < \vec{w}, \vec{x} > +b|$.

Margin

It is the distance of the closest example in the training set to. If $||\vec{w}|| = 1$, the margin is $min_{i=1,...,m} | < \vec{w}, \vec{x} > +b|$. The closest examples are the support vectors.

Hard-SVM

It works only for linearly separable data. The aim is to seek for the largest margin. The formulation of the problem is $argmax_{(\vec{w},b):||\vec{w}||=1} min_{i=1,...,m} | < \vec{w}, \vec{x} > +b|$, subject to $y_i(< \vec{w}, \vec{x_i} > +b) > 0 \forall i$, or equivalently $argmax_{(\vec{w},b):||\vec{w}||=1} min_{i=1,...,m} y_i(< \vec{w}, \vec{x_i} > +b) > 0 \forall i$.
We can formulate the quadratic programming formulation as:
input: S
Solve: $(\vec{w_0}, b_0) = argmin_{(\vec{w},b)} ||\vec{w}||^2$ subject to $y_i(< \vec{w}, \vec{x_i} > +b) > 1 \forall i$
Output: $\hat{\vec{w}} = \frac{\vec{w_0}}{||\vec{w_0}||}, \hat{b} = \frac{b_0}{||\vec{w_0}||}$.

Equivalent formulation and SVM

Assume that the first component of $\vec{x} \in X$ is 1, then $\vec{w_0} = min_{\vec{w}||\vec{w}||^2}$ subject to $\forall i : y_i(< \vec{w}, \vec{x_i} > +b) \geq 1$. The support vectors are the vectors at minimum distance from $\vec{w_0}$.

Proposition

Let $\vec{w_0}$ be defined as above. Let $\vec{I} = \{i : | < \vec{w_0}, \vec{x_i} > | = 1\}$, then there exists coefficients $\alpha_1, ..., \alpha_m$ such that $\vec{w_0} = \Sigma_{i \in I} \alpha_i \vec{x_i}$. Where the support vectors are $\{\vec{x_i} : i \in \vec{I}\}$. Notice that solving the hard-SVM is equivalent to find $\alpha_i \forall i = 1, ..., m$, and $\alpha_i \neq 0$ only for support vectors.

Soft-SVM

This version of the SVM is used if the data are not separable. The constraints are modified such that:

- We introduce slack variables $\vec{\xi} \geq 0$
- for each i=1,...,m: $y_i(< \vec{w}, \vec{x_i} > +b) \geq 1 - \xi_i$
- ξ_i measures how much the constraint $y_i(< \vec{w}, \vec{x_i} > +b) \geq 1$ is violated.

The problem is formulated as a regularization problem:
Input: S,λ
Solve: $min_{\vec{w},b,\xi}(\lambda||w||^2 + \frac{1}{m}\Sigma_{i=1}^m \xi_i)$, subject to $\forall i : y_i(< \vec{w}, \vec{x_i} > +b) \geq 1 - \xi_i, \xi_i \geq 0$
Output: $\vec{w}, b$

Soft-SVM and hinge

The hinge loss is defined as $l^{hinge}((\vec{w}, b), (\vec{x}, y)) = max\{0, 1 - y_i(< \vec{w}, \vec{x_i} > +b)\}$. Then the Loss function becomes $L_S^{hinge}(\vec{w}, b) = \frac{1}{m}\Sigma_{1=1}^m l^{hinge}((\vec{w}, b), (\vec{x}, y))$, that is:

$$min_{\vec{w},b}(\lambda||\vec{w}||^2 + \frac{1}{m}\Sigma_{1=1}^m l^{hinge}((\vec{w}, b), (\vec{x}, y)))$$

SGD for soft-SVM

How to solve $min_{\vec{w}}(\frac{\lambda}{2}||\vec{w}||^2 + \frac{1}{m}\Sigma_{1=1}^m max\{0, 1 - y_i(< \vec{w}, \vec{x_i} > +b)\}$
Algorithm:
$\vec{\theta^{(0)}} \leftarrow \vec{0}$;

for $t \leftarrow 1$ to T do:
$\eta^{(1)} \leftarrow \frac{1}{\lambda t}$; $\vec{w}^{(t)} \leftarrow \eta^{(t)}\vec{\theta}^{(t)}$;
choose uniformly at random from $\{1, ..., m\}$
$if\ y_i(< \vec{w}^{(t)}, \vec{x_i} >) < 1$ then $\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} + y_i\vec{x_i}$;
$else\ \vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)}$;
return $\bar{\vec{w}} = \frac{1}{T}\Sigma_{t=1}^{T}\vec{w}^{(t)}$.

Duality of SVM

For hard-SVM we want:

$$max_{\vec{\alpha} \in R^m : \vec{\alpha} \geq 0}(\Sigma_{i=1}^{m}\alpha_i - \frac{1}{2}\Sigma_{i=1}^{m}\Sigma_{j=1}^{m}\alpha_i\alpha_j y_i y_j < \vec{x_j}, \vec{x_i} >)$$

The solution gives us the vector α which defines the support vector $\{\vec{x_i} : \alpha_i \neq 0\}$

Kernel trick for SVM

Let $\phi()$ be a non linear transformation:

- Given S, we obtain $S' = \{(\phi(\vec{x_1}), y_1), ..., (\phi(\vec{x_m}), y_m)\}$
- Learn a model with S' and SVM, obtaining h_{SVM}
- Given $\vec{x} \in X$, the prediction is h_{SVM}

Kernel function

$K_\phi(\vec{x}, \vec{x'}) =< \phi(\vec{x}), \phi(\vec{x'}) >$, where $\phi(\vec{x})$ is a transformation of $\vec{x}$. The kernel we have seen are the following:

- Linear $\phi(\vec{x}) = \vec{x}$
- Sigmoid $K(\vec{x}, \vec{x'}) = tanh(\zeta < \vec{x}, \vec{x'} > +\Gamma), for\ \gamma, \zeta > 0$
- Degree Q kernels $K(\vec{x}, \vec{x'}) = (\zeta + \gamma < \vec{x}, \vec{x'} >)^Q$; for Q=2, w have the following $\phi(\vec{x}) = [\zeta, \sqrt{2\zeta\gamma}x_1, ..., \sqrt{2\zeta\gamma}x_d, \gamma x_1 x_1, x_1 x_2, ..., \gamma x_d x_d]^T \in R^{d^2+d+1}$
- Gaussian Radial basis function, or RBF, which is defined as $K(\vec{x}, \vec{x'}) = e^{-\gamma||\vec{x}-\vec{x'}||^2}$, with $\gamma > 0$. For $\gamma = 1, \vec{x} = x \in R$, we have $K(\vec{x}, \vec{x'}) = e^{-||x-x'||^2} = e^{-x^2}e^{2xx'}e^{-(x')^2} = e^{-x^2}(\Sigma_{k=0}^{+\infty}\frac{2^k(x^k)(x'^k)}{k!})e^{-(x')^2} =< \phi(x), \phi(x') > \Rightarrow \phi(\vec{x}) = e^{-x^2}(1, \frac{\sqrt{2}}{1!}x, \frac{\sqrt{2}}{2!}x^2, ...)$, which has an infinite number of dimensions.

Mercer's condition

$K(\vec{x}, \vec{x'})$ is a valid kernel function if and only if the kernel matrix $K = \begin{bmatrix} K(\vec{x_1}, \vec{x_1'}) & ... & K(\vec{x_1}, \vec{x_m'}) \\ ... & ... & ... \\ K(\vec{x_m}, \vec{x_1'}) & ... & K(\vec{x_m}, \vec{x_m'}) \end{bmatrix}$
is always symmetric, positive and semi-definite for any given $\vec{x_1}, ..., \vec{x_m}$

SVM for regression and L2 regularization

The objective function becomes $min \frac{\lambda}{2}||\vec{w}||^2 + \Sigma_{1=1}^{m} V_\epsilon(y_i - < \vec{x_i}, \vec{w} > -b)$, where $V_\epsilon(r) = \begin{cases} 0 \ if \ |r| < \epsilon \\ |r| - \epsilon \ otherwise \end{cases}$
The solution is given by $\vec{w} = \Sigma_{i=1}^{m}(\alpha_i^* - \alpha_i)\vec{x_i}$, with $\alpha_i^*, \alpha_i \geq 0$. the hypothesis is $h(\vec{x}) = \Sigma_{1=1}^{m}(\alpha_i^* - \alpha_i) < \vec{x_i}, \vec{x} > +b$. The support vectors are $\vec{x_i}$ such that $\alpha_i^* - \alpha_i \neq 0$.

Neural Networks

A neuron is a function $\vec{x} \to \sigma(< \vec{v}, \vec{x} >)$, with $\vec{x} \in R^d$. The activation function σ can be the sign function, the threshold function or the sigmoid. The single node point of view can be expressed as $a_{t+1,j}(\vec{x})$, its input when $\vec{x}$ is fed to the neural network, $O_{t+1,j}(\vec{x})$, its output when $\vec{x}$ is fed to the neural network.
We can express $a_{t+1,j}(\vec{x}) = \Sigma_{r:(V_{t,r}, V_{t+1,r}) \in E} W((V_{t,r}, V_{t+1,r}))O_{t,r}(\vec{x})$. If $\delta(a) = a$, then the neuron corresponds to a linear model. We calculate $= O_{t+1,j} = \delta(a_{t+1,j}(\vec{x}))$.

Neural Network's architecture

We define the architecture as $h_{V,E,\sigma,w} : R^{|V_0|-1} \to R^{|V_T|}$, and the hypothesis class as $H_{V,E,\sigma} = \{h_{V,E,\sigma,w} : \text{w is a mapping from E to R}\}$.

General construction for a boolean expression

Let's take an arbitrary function $f : \{1, -1\}^d \to \{1, -1\}$. We want to build a NN thhat corresponds to f such that if the input is $\vec{x}$, then the predictor of such NN is $f(\vec{x})$.

i consider $\vec{x}$ such that $f(\vec{x}) = 1$. For each such $\vec{x}$, there is a neuton in the only hidden layer that corresponds to $\vec{x}$. The neuron implements $g_i(\vec{x}) = sign(< \vec{x}, \vec{x'} > -d + 1)$, where $\vec{x'}$ is the input of the neuron and $\vec{x}$ is the weight on the incoming edges.

ii The output node implements $h(\vec{x'}) = sign(\Sigma_{i=1}^{k} g_i(\vec{x'}) + k - 1)$, where k is the number of inputs $\vec{x}$ such that $f(\vec{x}) = 1$

Expressiveness

It can implement any boolean function. For every d, there exists a graph(V,E) of depth 2, s.t. $H_{V,E,sign}$ contains all functions from $\{1, -1\}^d \to \{1, -1\}$

Proposition

For every d, let s(d) be the minimal integer such that there exists a graph (V,E) with $|V| = s(d)$ such that $H_{V,E,sign}$ contains all function from $\{1, -1\}^d \to \{1, -1\}$. Then s(d) is an exponential function of d. We obtain a similar result for the sigmoid as well.

Universal approximators

For every fixed $\epsilon > 0$ and every Lipschitz function $f : [-1,-1]^d \to [-1,1]$ it is possible to construct a neural network such that for every input $\vec{x} \in [-1,1]^d$ the output of the neural network is in $[f(\vec{x}) - \epsilon, f(\vec{x}) + \epsilon]$.

Proposition

Fix some $\epsilon \in (0,1)$. For every d, let s(d) be the minimal integer such that there exists a graph (V,E) with $|V| = s(d)$ such that $H_{V,E,\sigma}$, with σ=sigmoid, can approximate with precision ϵ, every 1-Lipschitz function $f : [-1,-1]^d \to [-1,1]$. Then s(d) is exponential in d.

Proposition

- VCdim$(H_{V,E,sign}) = O(|E|log(|E|))$;
- VCdim$(H_{V,E,sigmoid}) = \Omega(|E|^2)$;
- VCdim$(H_{V,E,sigmoid}) = O(|V|^2|E|^2)$;

Run-time length

Applying ERM rule with respect to $H_{V,E,sign}$ is computationally difficult even for small NN.

Proposition

Let $k \geq 3$. For every d, let (V,E) be a layered graph with d input nodes, k+1 nodes at the only hidden layer, where one of them is the constant neuron, and a single output node. Then is NP-hard tom implement the ERM rule w.r.t. $H_{V,E,sign}$.

Matrix Notation

Let t denote the number of layers, with $0 < t < T$, $d^{(t)} + 1$ the number of nodes, $V_{t,1}, ..., V_{t,d^{(t)}}$ the values of the nodes, and $w_{ij}^{(t)}$ as the arc from $V_{t-1,i}$ to $V_{t,j}$. Then we can represent all of these quantities in matrix form as follows:

$$\vec{V}^{(t)} = \begin{bmatrix} 1 \\ V_{t,1} \\ ... \\ V_{t,d^{(t)}} \end{bmatrix} = \begin{bmatrix} 1 \\ \sigma(V_{t,1}) \\ ... \\ \sigma(V_{t,d^{(t)}}) \end{bmatrix}$$

$$a_{t,j} =< \vec{w_j}^{(t)}, \vec{v}^{(t-1)} >, \text{ to obtain } \vec{a}^{(t)} = \begin{bmatrix} a_{t,1} \\ ... \\ a_{t,d^{(t)}} \end{bmatrix}$$

$$\sigma(\vec{a}^{(t)}) = \begin{bmatrix} \sigma(a_{t,1}) \\ ... \\ \sigma(a_{t,d^{(t)}}) \end{bmatrix} \Rightarrow \vec{V}^{(t)} = \begin{bmatrix} 1 \\ \sigma(\vec{a}^{(t)}) \end{bmatrix}$$

The weights from layer t-1 to t is:

$$\vec{w}^{(t)} = \begin{bmatrix} w_{01}^{(t)} & ... & w_{0d^{(t)}}^{(t)} \\ ... & ... & ... \\ w_{d^{(t)}1}^{(t)} & ... & w_{d^{(t-1)}d^{(t)}}^{(t)} \end{bmatrix} \Rightarrow \vec{a}^{(t)} = (\vec{w}^{(t)})^T \vec{V}^{(t-1)}$$

Forward propagation

Input: $\vec{x} = (x_1, ..., x_d)^T$
Output: prediction y
$\vec{V}^{(0)} \leftarrow (1, x_1, ..., x_d)^T$;
for $t \leftarrow 1$ to T do:
$\vec{a}^{(t)} \leftarrow (\vec{w}^{(t)})^T \vec{V}^{(t-1)}$;
$\vec{V}^{(t)} \leftarrow (1, \sigma(\vec{a}^{(t)})^T)^T$
$y \leftarrow \sigma(\vec{a}^{(T)})$;
return y;

Learning the parameters

Given a training set $S = \{(\vec{x_1}, y_1), (\vec{x_m}, y_m)\}$, we want to pick $\vec{w_{ij}}^{(t)}$ minimizing $L_S(h) = \frac{1}{m}\Sigma_{i=1}^m l(h, (\vec{x_i}, y_i))$. We use gradient descent by seeing $L_S(h)$ as a function of $\vec{w}^{(t)}, \forall 1 \leq t \leq T$. The updating rule is gonna be the following: $\vec{w}^{(t)} \leftarrow \vec{w}^{(t)} - \eta \nabla L_S(\vec{w}^{(t)})$.
$\frac{\delta L_S}{\delta \vec{w}^{(t)}} = \frac{\delta}{\delta \vec{w}^{(t)}}(\frac{1}{m}\Sigma_{i=1}^m l(h, (\vec{x_i}, y_i))) = \frac{1}{m}\Sigma_{i=1}^m \frac{\delta l(h, (\vec{x_i}, y_i))}{\delta \vec{w}^{(t)}}$

Sensitivity vector for layer t

$$\delta^{(t)} = \frac{\delta l}{\delta \vec{a}^{(t)}} = \begin{bmatrix} \frac{\delta l}{\delta \vec{a}_{t,1}} \\ ... \\ \frac{\delta l}{\delta \vec{a}_{t,d^{(t)}}} \end{bmatrix} = \begin{bmatrix} \delta_1^{(t)} \\ ... \\ \delta_{d^{(t)}}^{(t)} \end{bmatrix}$$

We now calculate $\frac{\delta l}{\delta w_{ij}^{(t)}} = \frac{\delta l}{\delta a_{t,j}} \frac{\delta a_{t,j}}{\delta w_{ij}^{(t)}} =$
$= \delta_j^{(t)} \frac{\delta}{\delta w_{ij}^{(t)}}(\Sigma_{k=0}^{d^{(t+1)}} w_{kj}^{(t)} V_{t-1,k}) =$
$= \delta_j^{(t)} V_{t-1,i}$
$\delta_j^{(t)} = \frac{\delta l}{\delta a_{t,j}} = \frac{\delta l}{\delta V_{t,j}} \frac{\delta V_{t,j}}{\delta a_{t,j}} =$
$= \frac{\delta l}{\delta V_{t,j}} \sigma'(a_{t,j})$
$\frac{\delta l}{\delta V_{t,j}} = \Sigma_{k=1}^{d^{(t+1)}} \frac{\delta a_{t+1,k}}{\delta V_{t,j}} \frac{\delta l}{\delta a_{t+1,k}} = \Sigma_{k=1}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$.
By puttin everything together we obtain that:

$$\delta_j^{(t)} = \sigma'(a_{t,j}) \Sigma_{k=0}^{d^{(t+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)}$$

Which tells us that we need a back-propagation algorithm.

Sensitivity algorithm

Input: data points $(\vec{x_i}, y_i)$ and a NN with weights $w_{ij}^{(t)}, 1 \leq t \leq T$.
Output: $\delta^{(t)}$ for t=1,...,T
Compute $\vec{a}^{(t)}$ and $\vec{V}^{(t)}$ for $t \in \{1, ..., T\}$;

$\delta^{(t)} \leftarrow \frac{\delta l}{\vec{a}^{(T)}}$
for t=T-1 down to 1 do:
$\delta^{(t)} \leftarrow \sigma'(a_{t,j})\Sigma_{t=1}^{d^{(l+1)}} w_{jk}^{(t+1)} \delta_k^{(t+1)} \forall j = 1, ..., d^{(t)}$
return $\delta^{(1)}, ..., \delta^{(T)}$;

Back-propagation

Input δ, NN.
Output NN with weights $w_{ij}^{(t)}$
$for\ s \leftarrow 0, 1, 2, ...\ do$:
$pick(\vec{x_k}, y_k)$ at random from S
forward propagation
compute $V_{t,j}$ for all j,t from $(\vec{x_k}, y_k)$
backwards propagation
compute $\delta_j^{(t)}$ for all j,t from $(\vec{x_k}, y_k)$
$w_{ij}^{(t)} \leftarrow w_{ij}^{(t)} - \eta V_{t-1} \delta_j^{(t)}$ for all i,j,t;
if converged return $w_{ij}^{(t)}$ for all i,j,t.

Regularized NN

The function to be minimized becomes: $L_S(h) + \frac{\lambda}{2}\Sigma_{i,j,t}(w_{i,j}^{(t)})^2$, where λ is the regularization parameter. Ww obtain h with Stochastic gradient descent $\rightarrow (\nabla(L_S(h)) + \lambda w^{(t)})$.

Clustering

We do not have target values or labels. The training set is defined as $S = (\vec{x_1}, ...\vec{x_m})$. The definition is to group together similar object such that similar objects are in the same group and dissimilar objects are in different groups. The difficulty of the task is that similarity is not transitive.

Model

It takes as input elements X and a distant function $d : XxX \rightarrow R_+$, where d is the distance function which has the following properties:

- it is symmetric: $d(\vec{x}, \vec{x'}) = d(\vec{x'}, \vec{x})$ for all $\vec{x}, \vec{x'} \in X$
- it satisfies the triangle inequality: $d(\vec{x}, \vec{x'}) \leq d(\vec{x}, \vec{z}) + d(\vec{z}, \vec{x'})$
- $d(\vec{x}, \vec{x}) = 0$

The output are the clusters $C = \{C_1, ..., C_k\}$ with $\cup_{i=1}^k C_i = X$ and for all $i \neq j : C_i \cap C_j = \emptyset$. Sometimes we use a similarity function $s : XxX \rightarrow R_+$ instead of a distance function , which has the following properties:

- it is symmetric: $s(\vec{x}, \vec{x'}) = s(\vec{x'}, \vec{x})$ for all $\vec{x}, \vec{x'} \in X$
- $s(\vec{x}, \vec{x}) = 1$

Cost minimization for clustering

We need to define a cost function over possible partitions of the objects. The aim is to find the partition of minimal cost. We mathematically define the distance as $d(\vec{x}, \vec{x'}) = ||\vec{x} - \vec{x'}||$. We give as input $X \subseteq X'$, where X' is a bigger space than X.

K-means clustering

Input: $(\vec{x_1}, ...\vec{x_m}), k \in N^+$
Goal: find $C = \{C_1, ..., C_k\}$ of $\vec{x_1}, ...\vec{x_m}$, centers $\mu_1, ...\mu_k$, with $\mu_i \in X'$ for $C_i, 1 \leq i \leq k$, that minimizes the k-means objective function $min_{\mu_1,...,\mu_k \in X'} \Sigma_{i=1}^{k} \Sigma_{\vec{x} \in C_i} d(\vec{x}, \mu_i)^2$.

Other objective functions

K-menoids: $min_{\mu_1,...,\mu_k \in X} \Sigma_{i=1}^{k} \Sigma_{\vec{x} \in C_i} d(\vec{x}, \mu_i)^2$, or k-median $min_{\mu_1,...,\mu_k \in X} \Sigma_{i=1}^{k} \Sigma_{\vec{x} \in C_i} d(\vec{x}, \mu_i)$. Notice that these last two functions have the centroids in X, while the k-means has them in X'.

k-means proposition

Given a cluster C_i, the center μ_i that minimizes $\Sigma_{\vec{x} \in C_i} d(\vec{x}, \mu_i)^2$ is $\mu_i = \frac{1}{|C_i|} \Sigma_{\vec{x} \in C_i} \vec{x}$.

Lloyd's algorithm

Input: X=$\{\vec{x_1}, ..., \vec{x_m}\}, k \in N^+$
Output: Clustering $C = (C_1, ..., C_k)$ of x;centers $\mu_1, ...\mu_k$, with μ_i center for $C_i, 1 \leq i \leq k$.
Algorithm:
Randomly choose $\mu_1^{(0)}, ..., \mu_k^{(0)}$;
for $t \leftarrow 0, ...,$do:
    for i=1,...,k: $C_1 \leftarrow \{\vec{x} \in X : i = argmin_j d(\vec{x}, \mu_j^{(t)})\}$;
    for i=1,...,k: $\mu_j^{(t)+1} \leftarrow \frac{1}{|C_i|} \Sigma_{\vec{x} \in C_i} \vec{x}$
    if convergence reached then
        return $C = (C_1, ..., C_k)$ and $\mu_1^{(t+1)}, ..., \mu_k^{(t+1)}$
To always reach the convergence we need one of these 3 conditions:

- The k-means objective function for the cluster at iteration t is not lower than at iteration t-1
- $\Sigma_{i=1}^{k} d(\mu_i^{(t+1)}, \mu_i^{(t)}) \leq \epsilon$
- $max_{1 \leq i \leq k} d(\mu_i^{(t+1)}, \mu_i^{(t)}) \leq \epsilon$

If we leave the algorithm be, it will reach convergence in a number of iteration $\in O(m^{kd})$ and $\in 2^{\Omega(\sqrt{m})}$. If convergence is reached in t iterations, then the algorithm is $O(tmkd)$, where t is the number of iterations, k the number of clusters, m is the size of the input.

k-means++

Input: X=$\{\vec{x_1}, ..., \vec{x_m}\}, k \in N^+$, a set F where $d(\vec{x}, F) = min_{\vec{f} \in F} d(\vec{x}, \vec{f})$.
The algorithm to compute the initial set F of centers:
μ_1 random point from X chosen uniformly at random;
$F \leftarrow \{\mu_1\}$
for $i \leftarrow 2$ to k do:
    $\mu_i \leftarrow$random point from $X \setminus F$, choosing point $\vec{x}$ with probability $\frac{(d(\vec{x},F))^2}{\Sigma_{\vec{x} \in X \setminus F}(d(\vec{x},F))^2}$
    $F \leftarrow F \cup \{\mu_i\}$
return F;

Theorem

Let $\Phi^*_{k-means}(X, k)$ be the cost of the optimal k-means clustering of X, and let $\Phi_{k-means}(X, F_{k-means++})$ be the cost of the clustering of X obtained using the points i $F_{k-means++}$ returned as centers and assign each poit of X to its cluster center. Then:

$$E[\Phi_{k-means}(X, F_{k-means++})] \leq 8(ln(k) + 2)\Phi^*_{k-means}(X, k)$$

Linkage based clustering

Different distances D(A,B) between clusters A and B can be used:

- single linkage: D(A,B)=min$\{d(\vec{x}, \vec{x'}) : \vec{x} \in A, \vec{x'} \in B\}$
- Average linkage: D(A,B)=$\frac{1}{|A||B|}\Sigma_{\vec{x} \in A, \vec{x'} \in B} d(\vec{x}, \vec{x'})$
- max linkage: D(A,B)=$max\{d(\vec{x}, \vec{x'}) : \vec{x} \in A, \vec{x'} \in B\}$

Termination condition

The common approach is to run the clustering algorithm many times changing k, obtaining $C^{(k)} = \{C^{(k)}_1, ..., C^{(k)}_k\}$. Then use a score S to evaluate $C^{(k)}$, getting $S(C^{(k)})$ scores. In the end, pick the maximum score $C = argmax_{C^{(k)}}\{S(C^{(k)}\}$.

Silhouette

It is a score based on the distance. Given $C = (C_1, ..., C_k)$ clusters of X, and $\vec{x} \in X$, let $C(\vec{x})$ be the cluster to which $\vec{x}$ is assigned to. Assume that $|C_i| \geq 2 \ \forall 1 \leq i \leq k$.
Define $A(\vec{x}) = \frac{\Sigma_{\vec{x'} \neq \vec{x}, \vec{x'} \in C(\vec{x})} d(\vec{x}, \vec{x'})}{|C(\vec{x})|-1}$. Given a cluster $C_i \neq C(\vec{x})$, let $d(\vec{x}, C_i) = \frac{\Sigma_{\vec{x'} \in C_i} d(\vec{x}, \vec{x'})}{|C_i|}$.
We can define $B(\vec{x}) = min_{C_i \neq C(\vec{x})} d(\vec{x}, C_i)$.
We can now define the silhouette score $s(\vec{x})$ of $\vec{x}$ as $s(\vec{x}) = \frac{B(\vec{x})-A(\vec{x})}{max\{A(\vec{x}),b(\vec{x})\}}$, which ranges from $[-1, 1]$.
Then, the silhouette score of the whole clustering is:

$$S(C) = \frac{\Sigma_{\vec{x} \in X} s(\vec{x})}{|X|}$$

Where the higher the score is, the better the clustering. $s(\vec{x})$ measures if $\vec{x}$ is closer to points in its cluster than to the cluster he is in.