1 Perceptron algorithm

1.1 Algorithm

Initialize $\theta^{(0)}$ to some value, and initialize the index k to 0. Iterate: Select the next data point $(\mathbf{x}_t, y_t)$ and check whether $\theta^{(k)}$ classifies it correctly. If it is incorrect (i.e.$y_t(\theta^{(k)})^T\mathbf{x}_t < 0$), set $\theta^{(k+1)} = \theta^{(k)} + y_t\mathbf{x}_t$ and increment $k \leftarrow k+1$.

1.2 Analysis of convergence and correctness

Assumption 1: There exists $R \in (0, \infty)$ such that every input $x_t \in \mathbb{D}$ satisfies $\|\mathbf{x}_t\| \leq R$. Equivalently, the input vectors are bounded.
Assumption 2: There exists a parameter θ^* and positive constant $\gamma > 0$ such that $\min_{t=1,;n} y_t(\theta^*)^T\mathbf{x}_t \geq \gamma$

Theorem 1. Under the initial vector $\theta^0 = \mathbf{0}$, for any data set $\mathbb{D}$ satisfying the above assumptions, the perceptron algorithm produces a vector $\theta^{(k)}$ classifying every example correctly after at most $k_{\max} = \frac{R^2\|\theta^*\|^2}{\gamma^2}$ mistakes (and hence updates steps), where θ^*, γ, R are defined in the two assumptions.

1.3 Margin and geometry

Definition: Let $\gamma = \min_{t=1,\ldots,n} y_t\theta^T\mathbf{x}_t$, we could define $\gamma_{geom} = \frac{\gamma}{\|\theta\|}$, which is the smallest distance from any data point $\mathbf{x_t}$ to the decision boundary.
Remark: We could see $\frac{1}{\gamma_{geom}}$ as a measure of difficulty. Also, we could rewrite $k_{\max} = \frac{R^2}{\gamma_{geom}^2}$, which is not directly dependent on dimension or number of samples.

2 Support vector machine

2.1 Formulation

We propose to find a linear classifier to maximize the margin, as larger margin leads to a more robust classifier. $\max_{\theta,\gamma} \frac{\gamma}{\|\theta\|}$ subject to $y_t\theta^T\mathbf{x}_t \geq \gamma\ \forall t$
By treating $\frac{\gamma}{\theta}$ as one entity, taking norm square and rewrite maximization into minimization, we could formulate as $\min_\theta \frac{1}{2}\|\theta\|^2$ subject to $y_t\theta^T\mathbf{x}_t \geq 1\ \forall t$
The optimal solution is unique.

2.2 Linear classifier with offset

We now add an offset into our linear classifier: $f(x) = \text{sign}(\theta^T\mathbf{x} + \theta_0)$, and consequently the optimization is reformulated as $\min_{\theta,\theta_0} \frac{1}{2}\|\theta\|^2$ subject to $y_t(\theta^T\mathbf{x}_t + \theta_0) \geq 1\ \forall t$ Offsets allow decision boundary not to pass through origin, thus giving greater flexibility. Note that the offset do not appear in the objective.

2.3 Soft margin

Since most datasets are not linearly separable in real life, we consider allowing misclassification, by penalize such mistakes in terms of additional cost in objective. The optimization problem is reformulated as $\min_{\theta,\theta_0,\varsigma} \frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^n \varsigma_t$ subject to $y_t(\theta^T x_t + \theta_0) \geq 1 - \varsigma_t, \varsigma_t \geq 0\ \forall\ t$ In this case, $\varsigma = (\varsigma_1, \ldots, \varsigma_t)$ is a set of slack variables. If $\varsigma_t = 0$, then we recover the original SVM problem. If $\varsigma_t \in (0,1)$, then it means that we still require correct classification, but we allow some points to lie within margin. If $\varsigma_t > 1$, then we allow misclassification. C is the magnitude of penalty, the larger we set C, the greater the price of violation, and thus the slack variables will be small for optimality to avoid large penalty.

2.4 Hinge loss

We define hinge loss as $[z]_+ = \max(0, z)$. We could reformulate soft margin problem as $\min_{\theta,\theta_0,\varsigma} \frac{1}{2}\|\theta\|^2 + C\sum_{t=1}^n [1 - y_t(\theta^T x_t + \theta_0)]_+$
Since if we consider the slack constraint $y_t(\theta^T x_t + \theta_0) \geq 1 - \varsigma_t \implies \varsigma_t \geq 1 - y_t(\theta^T x_t + \theta_0) \implies \varsigma_t \geq \max(0, 1 - y_t(\theta^T x_t + \theta_0)) \implies \varsigma_t \geq [1 - y_t(\theta^T x_t + \theta_0)]_+$. When optimality is reached, the slack variables will converge to hinge loss.

3 Linear regression

3.1 Formulation

We propose to find a linear regression model $y(x) = \theta^T x + \theta_0$ We optimize the model by minimizing mean square loss(visually, square of difference between true values and predicted values), therefore we are dealing with optimization problem $\arg\min_{\theta,\theta_0} \sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2$

3.2 Bayesian perspective

3.2.1 Model and noise distribution

We consider from the perspective that the dataset is generated from a fixed unknown linear relation(assume the true data distribution is somehow linear) and perturbed by random noise $y_t = (\theta^*)^T x_t + \theta_0^* + z_t$ where θ^*, θ_0^* are fixed and unknown. We consider that the random noise is Gaussian noise. $z_t \sim N(0, \sigma^2)$. We see that for a given classifier with parameter θ, θ_0, we have $P(y|x, \theta, \theta_0) = N(y; \theta^T x + \theta_0, \sigma^2)$

3.2.2 Maximum likelihood estimation

By assumption of independence between samples, the likelihood function(intuitively, probability that the dataset is described by the classifier with assumption of Gaussian noise) is the product of likelihood for each data point: $L(\theta, \theta_0, \sigma^2; D) = \prod_{t=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y_t - \theta^T x_t - \theta_0)^2}{2\sigma^2})$
We aim to find a model that is most likely to describe the dataset, which is to maximize L. Note that to maximize L is equivalent to $\max_{\theta,\theta_0,\sigma^2} \log L = K - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2$
We could observe that to optimize (θ, θ_0) is not dependent on σ^2, hence we look at optimization problem $\arg\max_{\theta,\theta_0} -\frac{1}{2\sigma^2}\sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2$
which is equivalent to $\arg\min_{\theta,\theta_0} \sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2$ We recover least square estimator! In other words, to find least square estimator is equivalent to find maximum likelihood estimator assuming Gaussian noise.

3.3 Analytic solution of least square estimator

Switch to matrix notation: $\sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2 = \|y - X\vartheta\|^2$, where $y = [y_1, \ldots, y_n]^T, X = [(x_1, 1), \ldots, (x_n, 1)]^T, \vartheta = [\theta, \theta_0]^T$
Differentiate the loss with respect to ϑ, we have the derivative $-2X^T(y - X\vartheta)$, set derivative to zero and solve for ϑ, and we get $\vartheta_{opt} = (X^T X)^{-1} X^T y$
The matrix $(X^T X)^{-1} X^T$ is known as *pseudo-inverse* of X.

3.4 Bias and variance

The true data y is generated by $y = X\vartheta^* + z, z \sim N(\mathbf{0}, \sigma^2\mathbf{I})$, therefore from result of ϑ_{opt} we have $\vartheta_{opt} = \vartheta^* + (X^T X)^{-1} X^T z$
Intuitively, suppose noise is not significantly high, then ϑ_{opt} is close to true parameter ϑ^*. Since $\mathbb{E}[\mathbf{z}] = \mathbf{0}$, we have $\mathbb{E}[\vartheta_{opt}] = \vartheta^*$, which means we are correct on average.
However, note that $\text{cov}[\vartheta^*, \vartheta_{opt}] = \sigma^2(X^T X)^{-1}$, variance is trace of covariance matrix. Hence if $(X^T X)^{-1}$ has large entries, the corresponding entries of ϑ_{opt} will have high variance.

3.4.1 Trade-off

The minimum square error could be broken down as $\mathbb{E}[\|\vartheta_{opt} - \vartheta^*\|^2] = \|\mathbb{E}[\vartheta_{opt}] - \vartheta^*\|^2 + \mathbb{E}[\|\vartheta_{opt} - \mathbb{E}[\vartheta_{opt}]\|^2]$ It is the bias-variance decomposition of total loss. Usually, increasing model complexity makes model more flexible and sensitive to data, which reduces bias but increases variance.

3.5 Regularization and ridge regression

We penalize large entries of θ by adding regularization term: $\arg\min_{\theta,\theta_0} \sum_{t=1}^n (y_t - \theta^T x_t - \theta_0)^2 + \lambda\sum_{j=1}^d \theta_j^2$. In matrix form, this gives $\vartheta_{opt} = \arg\min_\vartheta \|y - X\vartheta\|^2 + \lambda\|\vartheta\|^2$. Now closed form solution is $\vartheta_{opt} = (X^T X + \lambda I)^{-1} X^T y$
Similarly, we could consider expectation of estimator, in this case $\mathbb{E}[\vartheta_{opt}] = (I - \lambda(X^T X + \lambda I)^{-1})\vartheta^*$ On average, the estimator now shrinks the ground truth, which is expected given that we penalize large parameters.

4 Kernel methods

4.1 Motivation

We could introduce non-linearity by mapping input data to a designed feature space: $\mathbf{x} \to (\phi_1(\mathbf{x})), \phi_2(\mathbf{x}), \ldots, \phi_n(\mathbf{x}))$.
Inner products capture the geometry of the dataset, so geometrically inspired algorithms depend only on inner products. For example, distance is captured since $|x - x'|^2 = <x, x> + <x', x'> -2 <x, x'>$, and angle is captured since $\theta = \cos^{-1}(\frac{<x, x'>}{|x||x'|})$.
Instead of directly designing ϕ_i to explicitly map each data point to the feature space, we consider a kernel function $k(x_i, x_j)$ as an inner product in a possibly implicit feature space. Kernel trick is to take any algorithm dependent only on inner products and replace each inner product by a kernel value. The kernel function is intuitively a measure of similarity between x_i and x_j.

4.2 Definition

A function $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ is said to be a positive semidefinite kernel if it is symmetric: $k(x, y) = k(y, x)$ and for any integer $m > 0$ and any set of inputs $x_1, \ldots, x_m$ in $\mathbf{R}^d$, the kernel matrix $K = \{k(x_i, x_j)\}$ is positive semidefinite.

Theorem 2. A function $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ is a PSD kernel if and only if it is an inner product $< \phi(x), \phi(y) >$ for some mapping $\phi(x)$.

Theorem 3. If k_1, k_2 are kernels, then so are the following: $k(x, y) = f(x)k_1(x, y)f(y)$ for some function f. $k(x, y) = k_1(x, y) + k_2(x, y)$, $k(x, y) = k_1(x, y)k_2(x, y)$

4.3 Kernel linear regression

We perform some matrix manipulations: $(X^T X + \lambda I)X^T = X^T XX^T + \lambda X^T = X^T(XX^T + \lambda I)$
Multiply $(X^T X + \lambda I)^{-1}$ on the left and $(XX^T + \lambda I)^{-1}$ on the right to give $X^T(XX^T + \lambda I)^{-1} = (X^T X + \lambda I)^{-1} X^T \implies \theta_{opt} = X^T(XX^T + \lambda I)^{-1} y \implies y(x) = \theta_{opt}^T x = x^T\theta_{opt} = x^T X^T(XX^T + \lambda I)^{-1} y$.
Note that the prediction is only dependent on inner product, since $x^T X^T = (< x, x_1 >, < x, x_2 >, \ldots, < x, x_n >)$, and $XX^T = \{< x_i, x_j >\}$
Therefore, we could replace each inner product with kernel value to product a more general prediction function $y(x) = k(x)(K + \lambda I)^{-1} y$, where $k(x) = (k(x, x_1), \ldots, k(x, x_n))$ and $K = \{k(x_i, x_j)\}$
Intuition behind derivation: the estimate y is a weighted sum of the previously observed outputs $y_1, \ldots, y_n$, the more similar x' is to the corresponding x_t, the more weights is given to that y_t.

4.4 Kernel examples

4.4.1 Polynomial kernel

We generalize linear features to quadratic/higher power features by considering the higher order kernel, for example, we map $(x_1, x_2) \to (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$
An example of mapping to cubic feature is that $x \to (1, \sqrt{3}x, \sqrt{3}x^2, x^3)$, note that the inner product of the feature space simplifies nicely as $< \phi(x), \phi(x') >= (1 + xx')^3$.
Generalizing this idea to polynomials with degree p and functions in d dimensions, we have polynomial kernel $k(x, x') = (1 + < x, x' >)^p$. The kernel facilitates computation of inner product in polynomial feature space without explicitly computing ϕ.

4.5 RBF kernel

We consider $k(x, x') = \exp(-\frac{1}{2l^2}|x - x'|^2)$, which the associated feature space is infinite-dimensional. l is a length-scale parameter.

5 Convex analysis

5.1 Dual SVM

Consider primal SVM: hard margin with offset. The program is convex with affine constraints, thus strong duality holds.
The Lagrangian is $L(\theta, \theta_0, \lambda) = \frac{1}{2}|\theta|^2 + \sum_{t=1}^n \lambda_t(1 - y_t(\theta^T x_t + \theta_0))$, set $\frac{\partial L}{\partial\theta} = 0 \implies \theta = \sum_{t=1}^n \lambda_t y_t x_t$, set $\frac{\partial L}{\partial\theta_0} = 0 \implies -\sum_{t=1}^n \lambda_t y_t = 0$, thus $g(\lambda) = \sum_{t=1}^n \lambda_t - \frac{1}{2}\sum_{s=1}^n\sum_{t=1}^n \lambda_s\lambda_t y_s y_t x_s^T x_t$ if $\sum_{t=1}^n \lambda_t y_t = 0$, and $-\infty$ otherwise.

The dual problem is thus to maximize $g(\lambda)$, with constraint $\lambda_t \geq 0$ and $\sum_{t=1}^n \lambda_t y_t = 0$.
Note that $\theta_{opt} = \sum_{t=1}^n \lambda_t y_t x_t$. We just need to find θ_0. If x_t is a support vector, then $y_t(\theta^T x_t + \theta_0) = 1$, and we could compute that $\theta_0 = \frac{1}{y_t} - \theta^T x_t$.

5.2 Soft margin and kernel SVM

Similar analysis to get the dual problem as $\max_\alpha \sum_{t=1}^n \alpha_t - \frac{1}{2}\sum_{s=1}^n \sum_{t=1}^n \alpha_s \alpha_t y_s y_t x_s^T x_t$ subject to $\alpha_t \in [0, C]$ and $\sum_{t=1}^n \alpha_t y_t = 0$
By solving the dual problems and note that $\theta_{opt} = \sum_{t=1}^n \alpha_t y_t x_t$, we find the classifier as $\text{sign}(\theta_{opt}^T x + \theta_0 = \text{sign}(\sum_{t=1}^n \alpha_t y_t x_t^T x + \theta_0)$.
Note that the classifier depends on inner product only, we could apply kernel trick to solve kernel optimization problem: $\max_\alpha \sum_{t=1}^n \alpha_t - \frac{1}{2}\sum_{s=1}^n \sum_{t=1}^n \alpha_s \alpha_t y_s y_t k(x_s, x_t)$ subject to $\alpha_t \in [0, C]$ and $\sum_{t=1}^n \alpha_t y_t = 0$ and the final classifier as $y(x) = \text{sign}(\sum_{t=1}^n \alpha_t y_t y_s y_t k(x_s, x_t) + \theta_0)$, where $\theta_0 = \frac{1}{y_t} - \sum_{s=1}^n \alpha_s y_s k(x_s, x_t)$.

5.3 Gradient descent

Consider the general setting of optimizing parameters to minimize training loss: $\min_\theta l(\theta)$, where $l(\theta) = \frac{1}{n}\sum_{i=1}^n f_i(\theta)$. We assume differentiability, then we compute gradient vector $\frac{\Delta l}{\Delta \theta} = [\frac{\partial}{\partial \theta_1}, \ldots, \frac{\partial l}{\partial \theta_n}]^T$, we know that $-\frac{\Delta l}{\Delta \theta}$ is a descent direction. We update parameters as $\theta = \theta - \eta \frac{\Delta l}{\Delta \theta}$, where η is the learning rate.

5.4 Stochastic gradient descent

Consider the previous setting, the parameter is updated as $\theta = \theta - \eta \frac{1}{n}\sum_{i=1}^n \Delta f_i(\theta)$, which is computationally expensive for large n(large data, complex functions), therefore, we could only select a batch of indices to perform the update: $\theta = \theta - \eta \Delta f_i(x)$, which we randomly select one index i. Alternatively, we could select a mini-batch I and update: $\theta = \theta - \eta \frac{1}{|I|}\sum_{i \in I} \Delta f_i(\theta)$, on average we expect descent in loss.

6 Boosting

6.1 Decision stumps

$h(x, \theta) = h(x, \{s, k, \theta_0\}) = \text{sign}(s(x_k - \theta_0))$, where k is the index of the concerned feature of the stump, $s = \pm 1$, θ_0 is offset. The stump just classify based on whether feature k value is above or below threshold θ_0. The classifiers are correct more than half of time, because otherwise we could flip $s \to -s$.
We consider weighted sum of many simple stumps $f_M(x) = \sum_{m=1}^M \alpha_m h(x; \theta_m)$ where $\theta_m = (s_m, k_m, \theta_{0,m})$, we use AdaBoost algorithm to find good set of parameters $\{(\theta_m, \alpha_m)\}_{m=1}^M$

6.2 Exponential loss

$l(y, f(x)) = \exp(-y f(x))$. It is an upper bound of $0-1$ loss.

6.3 Adaboost algorithm

Input: Dataset $D = \{x_t, y_t\}_{t=1}^n$, number of iterations M. We initialize the weights $w_0(t) = \frac{1}{n}$ for $t = 1, \ldots, n$
Iteration: for $m = 1, \ldots, M$, choose next base learner $h(x, \theta_m)$ as $\theta_m = \arg\min_\theta \sum_{t: y_t \neq h(x_t, \theta)} w_{m-1}(t)$, set $\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$, where $\epsilon_m = \sum_{t: y_t \neq h(x_t, \theta)} w_{m-1}(t)$ is the minimal value attained in previous step. Update the weights: $w_m(t) = \frac{1}{Z_m} w_{m-1}(t) e^{-y_t h(x_t; \theta_m)\alpha_m}$ for each $t = 1, \ldots, n$, where Z_m is defined so that all weights add up to 1: $Z_m = \sum_{t=1}^n w_{m-1}(t) e^{-y_t h(x_t; \theta_m)\alpha_m}$.
Output: $f_M(x) = \sum_{m=1}^M \alpha_m h(x; \theta_m)$, classifier is $y(x) = \text{sign}(f_M(x))$

6.4 Analysis of AdaBoost

Overall intuition: for each m iteration, the next base learner chosen is the best stump when certain data points are treated more important than others. (importance measured by w_{m-1}) Update the weights by computing weighted sum of exponential loss of each data point incurred by the stump chosen and normalize the sum.
The quantity ϵ_m: the weighted training error. We choose a base learning that classifies best when different samples have different importance.
Update of w_m: If $y_t \neq h(x_t; \theta_m)$, then $w_m = \frac{1}{Z_m} w_{m-1}(t) e^{\alpha_m}$. If $y_t = h(x_t; \theta_m)$, then $w_m = \frac{1}{Z_m} w_{m-1}(t) e^{-\alpha_m}$. If the chosen classifier classifies a point x_t correctly, then decrease the associated weight. If the chosen classifier classifies a point x_t incorrectly, then increase the associated weight. Intuitively, future decisions put more importance on points that are classified wrongly by previous learner.
Definition of α_m: It is a decreasing function of ϵ_m (the lower the training error the base learner gives, the higher vote it is given). If $\epsilon_m = 0$, then $\alpha_m = +\infty$(if the classifier is perfect, the put all weights). If $\epsilon_m = \frac{1}{2}$, then $\alpha_m = 0$(if the classifier is only random guess, put no weight). $\epsilon_m > \frac{1}{2}$ is not relevant, since a classifier cannot be worse than random guessing(because we can reverse the sign).

6.5 Analysis of training error

Theorem 4. After M iterations, the training error of AdaBoost satisfies $\frac{1}{n}\sum_{t=1}^n \mathbf{1}(y_t f_M(t) \leq 0) \leq \exp(-2\sum_{m=1}^M (\frac{1}{2} - \epsilon_m)^2)$. In particular, if $\epsilon_m \leq \frac{1}{2} - \gamma$ for all m and some $\gamma > 0$, then $\frac{1}{n}\sum_{t=1}^n \mathbf{1}(y_t f_M(t) \leq 0) \leq e^{-2M\gamma^2}$, which implies that for $M > \frac{\log(n)}{2\gamma^2}$, the training error is zero.

Theorem 5. The updated weights are such that $\sum_{t=1}^n w_m(t)\mathbf{1}(y_t \neq h(x_t; \theta_m)) = \frac{1}{2}$. Intuitively, the weight is updated such that the base learner chosen at this iteration now produces error $\frac{1}{2}$(which makes it the worst possible learner with respect to updated weight), so same learner will not be chosen on two consecutive rounds.

7 Concentration inequalities

Theorem 6 (Hoeffding's inequality)**.** Let $Z = \sum_{i=1}^n X_i$, where X_i are independent and supported on $[a_i, b_i]$, then $\mathbf{P}(\frac{1}{n}|Z - \mathbb{E}[Z]| > \epsilon) \leq 2\exp(-\frac{2n\epsilon^2}{\frac{1}{n}\sum_{i=1}^n (b_i - a_i)^2})$

8 Statistical learning

8.1 Abstract setup

The learner has access to a dataset $D = \{x_i, y_i\}_{i=1}^n$, drawn from an unknown distribution: $(x_i, y_i)\ P_{XY}$. The learner uses data to decide on a classification/regression function. The set of all possible functions under consideration is called the function class F: for example, set of all linear classifiers; set of polynomial functions up to degree d; set of all classifiers generated with 1000 decision stumps.
The performance is measured according to a loss function that we hope to be small.
For $f \in F$, the expected loss over the distribution is given by $R(f) = \mathbb{E}[l(y, f(x))]$, with $(x, y)\ P_{XY}$, which is named as true risk. This is the ideal (and impossible) case of computing expectation over all data points over the distribution.
If the distribution is known, then we could find the Bayes-optimal classifier $f* = \arg\min_{f \in F} \mathbb{E}[l(y, f(x))]$. Since the distribution is unknown, we replace the expectation by the empirical average over the training set: $f_{erm} = \arg\min_{f \in F} R_n(f), R_n(f) = \frac{1}{n}\sum_{i=1}^n l(y_i, f(x_i))$, which is called the empirical risk minimization rule.
We could decomposition the test error(true risk) as follows: $R(f) = R_n(f) + (R(f) - R_n(f))$, first term is training error, second term is generalization error.

8.2 PAC learning

Motivation: We seek to attain a true risk within a small value ϵ of that achieved by the best f in the function class being considered. Since data is random, we cannot guarantee to achieve this, so only insist on probability $1 - \delta$. The number of samples required depend on both δ and ϵ, and also on the size of F.
Definition: Given a loss function $l(.,.)$, a function class F is said to be PAC-learnable if there exists an algorithm $A(D_n)$(with D_n containing n independent samples) and a function $n(\epsilon, \delta)$ such that the following holds: For any distribution P_{XY} used to generated D, and any $\epsilon, \delta \in (0, 1)$, if $n > n(\epsilon, \delta)$, then the following holds with probability $1 - \delta$: $R(\tilde{f}) \leq \min_{f \in F} R(f) + \epsilon$. Intuitively, we could train a learner and control its performance on unseen data.

8.3 PAC learnability for finite F

Theorem 7. For any bounded loss function in $[0, 1]$, any finite function class F is PAC-learnable with sample complexity $n(\epsilon, \delta) = \frac{2}{\epsilon^2}\log\frac{2|F|}{\delta}$. We use ERM rule as our choice of $A(D_n)$.

Proof idea: Hoeffding inequality, union bound.

8.4 Infinite F and VC dimension

Setting: Infinite F for binary classification with $0-1$ loss.
Definition: For an integer n, the growth function $S_n(F) = \sup_{x_1, \ldots, x_n} |\{(f(x_1, \ldots, f(x_n))) : f \in F\}|$, this is the number of different assignments that hypotheses from F can make to $x_1, \ldots, x_n$. It is an integer from 1 to 2^n. The VC dimension $d_{VC} = d_{VC}(F)$ of F is the largest k such that $S_k(F) = 2^k$. If this holds for all k, we define $d_{VC} = \infty$.(intuitively, k is the largest choice such that we can produce k data points to be shattered)

Theorem 8 (Sauer's lemma)**.** $S_n(F) \leq \sum_{i=0}^{d_{VC}} \binom{n}{i}$

Theorem 9. $S_n(F) = 2^n$ if $n \leq d_{VC}$, and $S_n(F) \leq (\frac{d_{VC} e}{n})^{d_{VC}}$ if $n > d_{VC}$. Weaker bound: $S_n(F) \leq (n+1)^{d_{VC}}$

Theorem 10. If the VC dimension $d_{VC} = d_{VC}(F)$ is finite, then the function class F is PAC-learnable under $0-1$ loss with sample complexity $n(\epsilon, \delta) = C\frac{d_{VC} + \log\frac{1}{\delta}}{\epsilon^2}$ for some constant C. Conversely, if $d_{VC} = \infty$, then F is not PAC-learnable.

Intuition to find d_{VC}: If we claim $d_{VC} = k$, we need to show one example (set of points) such that $S_k(F) = 2^k \implies d_{VC} \geq k$, and show that for any set of $k+1$ points, we have a labelling scheme not possible for any classifier to correctly classify all points, then $d_{VC} < k+1$.

9 Unsupervised learning

9.1 Clustering

Given number of clusters K, we wish to learn a partition of the dataset $D = D_1 \cup \cdots \cup D_K$, as well as an associated set of cluster centers $\{\mu_1, \ldots, \mu_K\}$, such that the sum of square distances is minimized: $J(\{D_j\}, \{\mu_j\}) = \sum_{j=1}^K \sum_{x \in D_j} |x - \mu_j|^2$

9.2 K-means algorithm

Motivation: perform two easy steps iteratively: Minimize distance with respect to partition for fixed center. Minimize distance with respect to center for fixed partition. Iterate until convergence: no change in assignment of partition or centers.
Steps: (1)For each data point x_i, assign to cluster j which the distance between x_i and μ_j is smallest as compared to all other centers. (2)For each cluster D_k, recompute μ_k as the average of all data points in the cluster: $\mu_k = \frac{1}{|D_k|}\sum_{x \in D_k} x$.
It is guaranteed by this approach of alternating optimization, J decreases after each iteration.

9.3 Distribution learning

Motivation: Given an unlabelled dataset D, estimate a distribution $p(x)$ that models the data well.
Method: parametric methods, consider classes of distributions $p(x; \theta)$ with some associated parameters collected into θ. The learning algorithm finds some $\tilde{\theta}$ and the estimate of the distribution is $\tilde{p}(x) = p(x; \tilde{\theta})$
Learning objective: maximum likelihood estimation, which selects $\tilde{\theta} = \arg\max_\theta \prod_{t=1}^n p(x_t; \theta)$, which is the parameter that maximizes the likelihood of associated distribution to describe the dataset.
Examples: MLE for mean and covariance matrix for Gaussian$(\mu, \sum)$ distribution are the sample mean and sample covariance: $\tilde{\mu} = \frac{1}{n}\sum_{t=1}^n x_t, \tilde{\sum} = \frac{1}{n}\sum_{i=1}^n (x_i - \tilde{\mu})(x_i - \tilde{\mu})^T$
MLE for Bernouli(θ) distribution is sample mean: $\tilde{\theta} = \sum_{i=1}^n x_i$
MLE for uniform$([0, \theta])$ distribution is $\tilde{\theta} = \max_t x_t$, this is an example where the estimate is biased: $\mathbb{E}[\tilde{\theta}] < \theta*$.