Notes on Machine Learning (incomplete)

Aswin

Contents

1 Overview

I made this document as a way to learn ML for my Maters' thesis . It's not an original work,but a compilation of scribed notes of the ML course by Dr Sanjoy Dasgupta(UCSD), [SD] which I audited.Some parts are drawn directly/inspired from Andrew NG's Stanford CS229 course notes, [ANG] and Kilian weinberger's Cornell CS4780 course notes[KW].My primary reference was 'The Elements of Statistical Learning' by Trevor Hastie,Robert Tibshirani,Jerome Friedman, [HTF].So there will be considerable overlap with the aforementioned materials.This note might lack mathematical rigor but I have tried to give proofs and supplementary topics wherever necessary.For each algorithm, I have given a url to Github repo containing the implemetation using one of the three datasets viz Fisher's Iris dataset, Wisconsin Breast Cancer Dataset and MNIST handwritten digits dataset. Please write to me if you find any inaccuracies.I hope this proves at least moderately interesting or useful for you.

2 Supervised Learning

In a typical scenario, we have an outcome measurement, usually quantitative (such as a stock price) or categorical (such as heart attack/no heart attack), that we wish to predict based on a set of features (such as diet and clinical measurements). We have a training set of data, in which we observe the outcome and feature measurements for a set of objects (such as people). Using this data we build a prediction model, or learner, which will enable us to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome. The examples above describe what is called the supervised learning problem. It is called "supervised" because of the presence of the outcome variable to guide the learning process.[HTF]

2.1 Variable Types and Terminology

The outcome measurement which we wish to predict denoted as ***outputs*** depend on a set of variebles denoted as ***inputs***. Classicaly,the *inputs* are independent varibales whereas *outputs* are dependent variables. The term *features* will be used interchangeably with inputs.

The *outputs* which we wish to predict can be qualitative or quantitative(as in blood sugar level). When the *outputs* are qualitative (as in spams or not spams),it is referred as categorical or discrete variables and are typically represented numerically by codes,as in -spam or not spam can be coded as -1 or 1. Depending upon the kind of output varible, the prediction task can be of two types: *regression* when we predict quantitative outputs and *classification* when we predict qualitative outputs.

The input variables/features are denoted by $x^{(i)}$ and the space of all such $x^{(i)}$ is X. The output variable that we are trying to predict is denoted as $y^{(i)}$ and the space of all such $y^{(i)}$ is Y. A pair $(x^{(i)}, y^{(i)})$ is called a training example and the dataset, we will be using to learn - a collection of n training examples $\{(x^{(i)}, y^{(i)}); i = 1, 2, ..., n\}$ - is called a training set. The superscript $\backslash(i)$" in the notation is simply an index into the training set.

3 Linear Regression

Given a vector of inputs, $x^T = (x_1, x_2, ..., x_d)$.We need to know functions/hypotheses h that can approximate y as a linear function of x:

$$h_\theta(x) = \theta_0 + \sum_{i=1}^{d} x_j \theta_j$$

θ_is are parameters/weight parametrizing the space of linear functions mapping from X to Y.The term θ_0 is the intercept, also known as the *bias* in machine learning.It is covenient to include θ_0 in the vector of weights θ and add constant variable 1 to the vector x, so that

$$h_\theta(x) = \theta^T x$$

w (weights) and b (bias) can be interchangably used with θ and θ_0 respectively.

For a training set,we have to learn the parameters θ, so that we can predict y. One reasonable method seems to be to make $h(x)$ close to y,for the training examples.To formalize this, we will define a function that measures, for each value of the θ's, how close the $h(x^{(i)})'$ s are to the corrsponding $y^{(i)}$'s.

We define Cost/Loss function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2,$$

this is the Least Squares cost function.In this approach, we pick the coefficients θ to minimize the cost function J.The LS cost function is quadratic function in weights, θ and hence it's minimum always exist, but may not be unique. Given a set of training examples $(x^{(i)}, y^{(i)})$ i.e training set ,define a matrix X to be the m-by-n matrix that contains the input values of training examples in it's rows:

$$\begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

Let $\mathbf{y}$ be the m-dimensional vector containing the target/output values from the training set:

$$\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Since $h_\theta(x^{(i)}) = (x^{(i)})^T\theta$,we can verify that

$$X\theta - \mathbf{y} = \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}$$

$\frac{1}{2}(X\theta - \mathbf{y})^T(X\theta - \mathbf{y}) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 = J(\theta)$ This is the cost function.To minimize J,we have to find the derivatives with respect to θ

Some matrix derivative results

$$\nabla_A trAB = B^T$$

$$\nabla_A|A| = |A|(A^{-1})^T$$

$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T$$

$$\nabla_A trABA^TC = CAB + C^TAB^T$$

Using the last two results

$$\nabla_{A^T} trABA^TC = B^TA^TC^T + BA^TC$$

The cost function, $J(\theta) = \frac{1}{2}(X\theta - \mathbf{y})^T(X\theta - \mathbf{y})$

$$\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\theta - \mathbf{y})^T(X\theta - \mathbf{y})$$

$$= \frac{1}{2}\nabla_\theta(\theta^TX^TX\theta - \theta^TX^T\mathbf{y} - \mathbf{y}^TX\theta + \mathbf{y}^T\mathbf{y})$$

$$= \frac{1}{2}\nabla_\theta tr(\theta^T X^T X\theta - \theta^T X^T \mathbf{y} - \mathbf{y}^T X\theta + \mathbf{y}^T\mathbf{y})$$

$$= \frac{1}{2}\nabla_\theta (tr\theta^T X^T X\theta - 2tr\mathbf{y}^T X\theta)$$

$$= \frac{1}{2}(X^T X\theta + X^T X\theta - 2X^T\mathbf{y}) = X^T X\theta - X^T\mathbf{y}$$

To minimize J, we set the derivative to zero and obtain:$X^T(X\theta - \mathbf{y}) = 0$. If $X^T X$ is nonsingular then the value of θ that minimizes $J(\theta)$ is given in closed form by the equation,

$$\theta = (X^T X)^{-1} X^T \mathbf{y}$$

Now that we have the parameters,we can predict the output corresponding to the input $x^{(i)}$ as $y^{(i)} = \theta^T x^{(i)}$

4 Logistic Regression

Logistic regression is a proabilistic method,where a linear function of x, $wx + b$ is mapped to $[0,1]$ using our new hypothesis $h_w(x)$ defined as

$$h_w(x) = g(w^T x) = \frac{1}{1 + \exp(-w^T x)} ,$$

Where

$$g(z) = \frac{1}{1 + \exp(-z)}$$

is called the logistic function or the sigmoid function. The learning problem is, given a data set and logistic regression model how will we find the parameter w.We can achieve this using probabilistic assumptions, and then fit the parameters via maximum likelihood.

Let us assume that

$$P(y = 1|x; w) = h_w(x)$$

$$P(y = 0|x; w) = 1 - h_w(x)$$

This together can be expressed as

$$p(y|x;w) = (h_w(x))^y(1-h_w(x))^{1-y}$$

Assuming that we have m training examples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(w) &= P(\mathbf{y}|\mathbf{X};w) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};w) \\ &= \prod_{i=1}^{m} (h_w(x^{(i)}))^{y^{(i)}}(1-h_w(x^{(i)}))^{1-y^{(i)}} \end{aligned} \tag{1}$$

Maximizing the likelhood is equivalent to maximizing any strictly incresing function of likelihood.

$$\begin{aligned} l(w) &= logL(w) \\ &= \sum_{i=1}^{m} y^{(i)}logh(x^{(i)}) + (1-y^{(i)})log(1-h(x^{(i)})) \end{aligned} \tag{2}$$

We can use gradient descent to maximize the above equation and the update rule will be $w := w + \alpha\nabla_w l(w)$.For a single training example,

$$\frac{\partial l(w)}{\partial w_j} = (y - h_w(x))x_j$$

So the update rule for stochastic gradient ascent becomes

$$w_j := w_j + \alpha(y^{(i)} - h_w(x^{(i)}))x_j^{(i)}$$

5 The Perceptron

In a binary classification problem , where dataset(D) is $(x,y) \in \mathbb{R}^d \times \{-1,1\}$, the learning problem is to find a hyperplane which separates the data into two classes, assuming the data is linearly classifiable. The hyperplane is parametrized by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $w.x + b = 0$, so the poblem is equivalent to learning the parameters w and b.On point x, we predict the label as **sign(w.x + b)**.

$$(w.x + b) > 0 \implies y = +1 \therefore y(w.x + b) > 0$$

$$(w.x + b) < 0 \implies y = -1 \therefore y(w.x + b) > 0$$

i.e., If the true label of x is y, then $y(w.x + b) > 0$, whereas for a misclassified pointi$y(w.x + b) \leq 0$. The Loss function for the perceptron can be defined as

$$Loss = \begin{cases} 0, & \text{if } y(w.x + b) > 0 \\ -y(w.x + b), & \text{if } y(w.x + b) \leq 0 \end{cases} \tag{3}$$

This loss function is a convex function of $y(w.x + b)$, and we can use stochastic gradient descent to find the value of parameters that minimizes the loss function. The update on the parameter w can be written as $w := w - \eta \nabla L(w)$,where $w = [w, b]$.The derivative of Loss function with respect to the parameters(assuming bias term is not absorbed into the weight vector) are

$$\frac{\partial L}{\partial w} = -yx, \frac{\partial L}{\partial b} = -y$$

.

So the update rule will be $w := w + \eta yx$ and $b := b + \eta y$. For $w = [w, b]$, this is equivalent to $w := w + \eta yx$. If $\eta = 1$, then the update rule will be $w := w + yx$.

Perceptron Algorithm

```
Initialize $\vec{w} = 0$
while TRUE do
    $m = 0$
    for $(x_i, y_i) \in D$ do
        if $y_i(\vec{w}^T.x_i) \leq 0$ then
            $\vec{w} \leftarrow \vec{w} + yx$
            $m \leftarrow m + 1$
        end if
    end for
    if $m = 0$ then
        break
    end if
end while
```

If the taining data is linearly classifiable, Perceptron is guaranteed to converge after finite number of steps and return a seperating hyperplane with zero training error.

Margin γ of a hyperplane w is defined as $\gamma = min_{(x_i,y_i)\in D}\frac{|x_i^T w|}{\|w\|_2}$, i.e it is the distance to the closest data point from the hyperplane paramtrized by w.

Theorem. *If a data set is linearly separable, the Perceptron will find a separating hyperplane in a finite number of updates.*

Proof. Suppose $\exists w^*$ such that $y_i(x^T w^*) > 0\ \forall (x_i, y_i) \in D$.Suppose that we rescale each data point and the w^* such that

$$\|w^*\| = 1 \ and\ \|x_i\| \leq 1 \forall x_i \in D$$

So the margin γ for the hyperplane w^* becomes $\gamma = min_{(x_i,y_i)\in D}|x_i^T w^*|$. After rescaling, all inputs x_i lies in a unit sphere in d-dimensional space.The separating hyperplane is defined by w^* with $\|w\|^* = 1$ i.e is w^* lies exactly on the unit sphere.

We claim that if the above assumptions hold,then the Perceptron algorithm makes atmost $\frac{1}{\gamma^2}$ mistakes.The update on w is only done in the instance of misclassification, i.e when $y(x^T w) \leq 0$ holds.As w^* is a seperating hyper-plane and classifies all points correctly, $y(x^T w) > 0\ \forall x$.

Consider the effect of an update $w \leftarrow w + xy$ on the two terms $w^T w^*$ and $w^T w$.

$w^T w^* = (w + xy)^T w^* = w^T w^* + y(x^T w^*) \geq w^T w^* + \gamma$

The inequality follows from the fact that, for w^*, the distance from the hyperplane defined by w^* to x must be at least γ i.e $y(x^T w^*) = |x^T w^*| \geq \gamma$.This implies that with each update $w^T w^*$ grows atleast by γ

$w^T w = (w + xy)^T (w + xy) = w^T w + \underbrace{2y(w^T x)}_{\leq 0} + \underbrace{y^2(x^T x)}_{0 \leq\ \leq 1} \leq w^T w + 1.$

This inequality follows the fact that, $2y(w^T x) \leq 0$, as we had to make an update, meaning x was misclassified. $0 \leq y^2(x^T x) \leq 1$ as $y^2 = 1$ always and all $x^T x \leq 1$ as $\|x\| \leq 1$,(rescaled). This implies that $w^T w$ grows at most by 1.

After M updates, the two inequalities becomes

$$w^T w^* \geq M\gamma$$

$$w^T w \leq M$$

$$M\gamma \leq w^T w^* \leq |w^T w^*| \leq \|w^T\| \underbrace{\|w^*\|}_{1} = \sqrt{w^T w}$$

$w^T w$ can most be M, as w is initialized with 0 and with each update $w^T w$ grows at most by 1.

$$\implies M\gamma \leq \sqrt{M}$$
$$\implies M^2\gamma^2 \leq M$$
$$\implies M \leq \frac{1}{\gamma^2}$$

Hence the number of updates M is bounded from above by a constant.

□

6 Boosting

A Weak Classifier is the one with accuracy marginally better than random guessing. For a binary weak classifier this means, $P(h(x) \neq y) = \frac{1}{2} - \epsilon$. An learning algorithm which consistently generate such weak classifier is called a **weak learner**.
Boosting is a machine learning approach where such weak learners are combined to get better prediction acuracy.

6.1 AdaBoost

AdaBoost Algorithm

```
1. Given $(x^{(i)}, y^{(i)}), ..., (x^{(N)}, y^{(N)})$, where $y^{(i)} \in \{-1, +1\}$
2. Initialize the observation weights $w_i = \frac{1}{N}, i = 1, 2, ..., N$.
3. For m =1 to M:
    (a) Fit a classifier $h_m(x)$ to the trainng data using weights $w_i$.
    (b) Compute
        $$err_m = \frac{\sum_{i=1}^{N} w_i y^{(i)} h_m(x^{(i)})}{\sum_{i=1}^{N} w_i}$$
    (c) compute $\alpha_m = \log((1 - err_m)/err_m)$
    (d) set $w_i \leftarrow w_i.exp[-\alpha_m.y^{(i)} h_m(x^{(i)})],\ i = 1, 2, ..., N$
4. Output $H(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$
```

7 Nearest Neighbour Methods

Nearest-neighbour methods use those observations in the taining set $\mathcal{T}$ closest in input space to x to form $\hat{Y}$. The k-nearest neighbour fit for $\hat{Y}$ is defined as follows:

$$Y(\hat{x}) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i,$$

where N_k is neighbourhood of x defined by k closest points x_i in the training sample.

Closeness is quantified by a metric, which for instance can be assumed as Euclidean distance.We find k observations with x_i closes to x in the input space of training set, and average their responses.

The notion of distance between two vectors is defined by a norm. A norm is a function from a vector space over the real or complex numbers to the nonnegative real numbers that satisfies certain properties pertaining to scalability and additivity, and takes the value zero if only the input vector is zero.

As mentioned above we will use the Euclidean norm for all practical purposes.The Euclidean norm is a specific norm on a vector space, that is strongly related with the Euclidean distance, and equals the square root of the inner product of a vector with itself.On an n-dimensional Euclidean space R^n, the intuitive notion of length of the vector $x = (x1, x2, ..., xn)$ is captured by the formula

$$\|x\|_2 = \sqrt{x_1^2 + \dots x_n^2}$$

This definition of norm gives distance between two vectors as the euclidean norm of the component wise difference i.e if $x, y \in R^n$ then $\|(x-y)\|_2 = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

Find an implementation of NN method here

8 Support Vector Machines [HTF][KW]

The Perceptron algorithm assures to return a linear seperating hyperplane if one exists.Existence of one such hyperplane implies that there is infinite such hyperplanes, and the perceptron doesn't guarantee to return the optimal seperating hyperplane i.e, the one with maximum margin. Support vector machines(SVM) finds maximum margin hyperplane.

The training dataset is $(x^{(i)}, y^{(i)}) \in \mathbb{R}^d\{-1, +1\}, i = 1, ..., N$ We define a hyperplane by $\mathcal{H} = \{x | w^T x + b = 0\}$ paramtrized by w and b.Let the margin γ be defined as the distance from the hyperplane to the closest point across both the classes. A classifiction rule induced by the hyperplane is $h(x) = sign(w^T x + b)$

Consider some point x. Let d be the vector from $\mathcal{H}$ to x of minimum length. Let x^P be the projection of x onto $\mathcal{H}$. It follows then that:

$x^P = x - d$. w is perpendicular to $\mathcal{H}$. d is parallel to w, so $d = \alpha w$ for some $\alpha \in \mathbb{R}$. $x^P \in \mathcal{H}$ which implies $w^T x^P + b = 0$ therefore $w^T x^p + b = w^T(x-d) + b = w^T(x-d) + b = w^T(x - \alpha w) + b = 0$, which implies $\alpha = \frac{w^T x + b}{w^T w}$

The length of d is the euclidean norm $||d||_2 = \sqrt{d^T d} = \sqrt{\alpha^2 w^T w} = \frac{|w^T x + b|}{\sqrt{w^T w}} = \frac{|w^T x + b|}{||w||_2}$

Margin $\mathcal{H}$ with respect to $D : \gamma(w, b) = \min x \in D \frac{|w^T x+b|}{||w||_2}$

By definition, the margin and hyperplane are scale invariant : $\gamma(\beta w, \beta b) = \gamma(w, b) \forall \beta \neq 0$

Note that if the hyperplane is such that γ is maximized, it must lie right in the middle of the two classes. In other words, γ must be the distance to the closest point within both classes. (If not, you could move the hyperplane towards data points of the class that is further away and increase γ, which contradicts that γ is maximized.)

Our aim is to find a hyperplane with maximum margin and that all the points should be correctly classified. This is a constrained optimization problem where the objective is the maximization of margin subject to the constraint that all the points must be rightly classified.

$\underbrace{\max_{w,b} \gamma(w, b)}_{\text{maximize margin}}$ such that $\underbrace{\forall i,\ y_i(w^T x_i + b) \geq 0}_{\text{seperating hyperplane, such that all points are correctly classified}}$

By plugging in the definition of γ:

$\underbrace{max_{w,b} \frac{1}{||w||_2} \ min_{x_i \in D} \ |w^T x_i + b|}_{\text{maximize margin}} \ s.t. \ \underbrace{\forall i,\ y_i(w^T x_i + b) \geq 0}_{\text{separating hyperplane}}$

Because the hyperplane is scale invariant, we can fix the scale of w, b anyway we want. Let's choose it such that

$$min_{x \in D} \ |w^T x + b| = 1$$

Now our objective becomes $max_{w,b} \frac{1}{||w||_2}.1 = min_{w,b} \ ||w||_2 = min_{w,b} \ w^T w$

Thus the optimization problem becomes,

$$min_{w,b} \ w^T w$$

$$\forall i,\ y_i(w^T x_i + b) \geq 0$$

$$min_i \ |w^T x_i + b| = 1$$

The two stated constraints for this optimization problem is equivalent.

$min_i \ |w^T x_i + b| = 1$ is same as $min_i \ y_i(w^T x_i + b) = 1$

Now $\forall i,\ y_i(w^T x_i + b) \geq 0$ and $min_i \ \ y_i(w^T x_i + b) = 1$, together this implies $y_i(w^T x_i + b) \geq 1$

$$min_{w,b} \ w^T w$$

$$\forall i,\ y_i(w^T x_i + b) \geq 1$$

Now, this is a convex optimization problem with quadratic objective and linear inequality constraints.

For the optimal w, b pair some training points will have tight constraints i.e $y_i(w^T x_i + b) = 1$.These training points are called **support vectors.** Support vectors are the training points that define the maximum margin of the hyperplane to the data set and they therefore determine the shape of the hyperplane.

8.1 Soft-Margin SVM

If the classes overlap in the feature space i.e the linear inequality constraints are violated, there exist no solution to the optimization problem, this is usually the case with low dimensional data, but nevertheless we would like to have a hyperplane which can get most of the points correctly claassified with few violaion of constraints.

This is achieved by allowing the constraints to be violated ever so slight with the introduction of slack variables $\xi = (\xi_1, \xi_2, ..., \xi_N)$

$$min_{w,b}\ w^T w + C\sum_{i=1}^{n} \xi_i$$

$$s.t.\ \forall i,\ y_i(w^T x_i + b) \geq 1 - \xi_i$$

$$\forall i\ \xi_i \geq 0$$

The constant C decides how much slack we can use.The optimization problem is a tradeoff between slack ξ and margin γ.We need to minimize $||w||$(for maximum margin) and $\sum \xi_i$(total slack) as we dont want too many constraints to be violated.

The slack variable ξ_i allows the input x_i to be closer to the hyperplane (or even be on the wrong side), but there is a penalty in the objective function for such "slack". If C is very large, the SVM becomes very strict and tries to get all points to be on the right side of the hyperplane. If C is very small, the SVM becomes very loose and may "sacrifice" some points to obtain a simpler (i.e. lower$||w||_2^2$) solution.

The value ξ_i is proportioal amount by which the prediction is on wrong side of the margin.Hence for points well inside the boundary the slack is zero and do not influence shape of the hyperplane.

If $C = 0$,total slack is zero and that means slack is free and it's possible to violate as many as constraints, this implies $w = 0$.If $C = \infty$,slack is infinitely costly.So we donts use slack, we fall back to hard margin-SVM.

9 Principal Component Analysis

PCA is a dimensionality reduction algorithm, which aims to find those directions in which most of the variance of data lies i.e is it identifies the subspace in which the data approximately lies.Inorder to run, PCA the data needs to be preprocessed to normalize it's mean and variance.The preprocessing procedure is as follows:

1. Let $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i$
2. Replace each $x^{(i)}$ with $x^{(i)} - \mu$ (zero out mean of the data)
3. Let $\sigma_j^2 = \frac{1}{m}\sum_i (x_j^{(i)})^2$
4. Replace each $x_j^{(i)}$ with $x_j^{(i)}/\sigma_j$ (rescale each coordinate to have unit varaince,which ensures that different attributes are all treated on the same "scale.")

The dimensionality reduction is achieved by projecting the data into directions which capturre most of the variance of the data.

Let $x^{(i)} \in \mathbb{R}^n$ be a point in our dataset and u be a unit vector. $x^{(i)}$'s projection onto u is $x^{(i)^T}u$.Hence, to maximize the variance of the projections, we would like to choose a unit-length u so as to maximize:

$$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)^T}u)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u = u^T\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right)u$$

Maximizing this subject to $\|u\|_2 = 1$ gives the principal eigenvector of $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}$, which is the emperical covariance matrix of the data,assuming it has zero mean. read

The direction of maximum variance turns out to be eigenvectors of the covariance matrix.If we wish to project our data into a k-dimensional subspace $(k < n)$, we should choose $u_1, ..., u_k$ to be the top k eigenvectors of Σ. The u_is now form a new, orthogonal basis for the data.

A datapoint $x^{(i)}$ projected into a low(say $k < n$)-dimensional subspace spanned by top k eigenvectors of the covariance matrix is given by:

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathbb{R}^k$$

The vectors $u_1, ..., u_k$ are called the first principal components of the data.

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times k}$$

$$X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_k \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times k}$$

U is an orthogonal matrix, each column vector is an eigenvector of a covariance matrix.

$$\underbrace{Y}_{\text{low-dimesional data}} = \underbrace{X}_{\text{original data}} \times \underbrace{U}_{\text{k-principal components}}$$

$$\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix} \times \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \dots & u_k \\ | & | & & | \end{bmatrix}$$

We can reconstrut the data by projecting back into the original canonical basis. $Y = XU$, $YU^T = XUU^T \implies YU^T = X$. PCA can also be thought as an algorithm to choose the basis that minimizes the approximation(reconstruction also) error arising from projecting the data onto the k-dimensional subspace spanned by them.

Find an implementation of PCA with MNIST data here.

10 Decision Trees

Most data that is interesting has some inherent structure. In the k-NN case we make the assumption that similar inputs have similar neighbors.. This would imply that data points of various classes are not randomly sprinkled across the space, but instead appear in clusters of more or less homogeneous class assignments. Although there are efficient data structures enable faster nearest neighbor search, it is important to remember that the ultimate goal of the classifier is simply to give an accurate prediction. Imagine a binary classification problem with positive and negative class labels. If you knew that a test point falls into a cluster of 1 million points with all positive label, you would know that its neighbors will be positive even before you compute the distances to each one of these million distances. It is therefore sufficient to simply know that the test point is an area where all neighbors are positive, its exact identity is irrelevant.

Decision trees are exploiting exactly that. Here, we do not store the training data, instead we use the training data to build a tree structure that recursively divides the space into regions with similar labels. The root node of the tree represents the entire data set. This set is then split roughly in half along one dimension by a simple threshold t All points that have a feature value fall into the right child node, all the others into the left child node. The threshold t

and the dimension are chosen so that the resulting child nodes are purer in terms of class membership. Ideally all positive points fall into one child node and all negative points in the other. If this is the case, the tree is done. If not, the leaf nodes are again split until eventually all leaves are pure (i.e. all its data points contain the same label) or cannot be split any further (in the rare case with two identical points of different labels).

Decision trees have several nice advantages over nearest neighbor algorithms:

1. once the tree is constructed, the training data does not need to be stored. Instead, we can simply store how many points of each label ended up in each leaf - typically these are pure so we just have to store the label of all points;
2. decision trees are very fast during test time, as test inputs simply need to traverse down the tree to a leaf - the prediction is the majority label of the leaf;
3. decision trees require no metric because the splits are based on feature thresholds and not distances.

What we try to achieve is a maximally compact tree which has only pure leaves.It's always possible to make trees with pure leaves unless there are two different input vectors having identical features but different label.But it turns out that finding such minimum size tree is NP-hard(comlexity).We can approximate it very effectively with a greedy, top-down, Recursive partitioning approach. We keep splitting the data to minimize an impurity function that measures label purity amongst the children.There are different measures of impurity.

10.1 Impurity Functions

Data : $S = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$, $y^{(i)} \in \{1, 2, ..., c\}$, where c is the number of classes. Let $S_k = \{(x, y) \in S : y = k\}$, $S = S_1 \cup ... \cup S_c$. We define $P_k = \frac{|S_k|}{|S|}$ (fraction of inputs in S with label k)

Gini impurity of a leaf is defined as $G(S) = \sum_{k=1}^{c} p_k(1 - p_k)$. Gini Impurity of a tree : $G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R)$ Where $S = S_L \cup S_R$; $S_L \cap S_R = \phi$;$\frac{|S_L|}{|S|}$ is fraction of inputs in the left sub tree.$\frac{|S_R|}{|S|}$ is fraction of inputs in the left subtree

Entropy Entropy of a leaf $H(s) = -\sum_k p_k log(p_k)$. Entropy over tree $H(s) = P^L H(S^L) + p^R H(S^R)$.Entropy of child nodes formed is weighted average of the nodes.The optimal split is the one in which entropy of the children is less than the parent.

- How to find the tree with minimum entropy?
- How to find the optimal split? NP-Hard Problem
- How to split? Try all splits.Take the one with lowest entropy
- How many possible splits? For N data points, $x \in \mathbb{R}^d$ there are $(N-1)D$ possible splits.

11 Bagging

Bagging is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting of any classifier.
Bias/Varience decomposition :

$$\underbrace{\mathbb{E}[(h_D(x)-y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x)-\bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x)-\bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x)-y(x))^2]}_{\text{Noise}}$$

We would like to reduce the variance: $\mathbb{E}[(h_D(x)-\bar{h}(x))^2]$, for this to happen $h_D] \mapsto \bar{h}$.The weak law of large numbers says that for independent and identically distributed x_i with mean $\bar{x}$,

$$\frac{1}{m}\sum_{i=1}^{m} x_i \to \bar{x} \; as \; m \; \to \infty$$

Assume that we have m taining sets $D_1, D_2, \dots D_m$ drawn from P^n.Train a classifier on each of the datset to obtain h_{D_i}s and extrapolating above idea to classifiers, we obtain average of all such h_{D_i}s:

$$\hat{h} = \frac{1}{m}\sum_{i=1}^{m} h_{D_i} \to \bar{h} \; as \; m \to \infty$$

.We refer to such an average of multiple classifiers as an ensemble of classifiers.If $\hat{h} \to \bar{h} \implies \mathbb{E}[(h_D(x)-\bar{h}(x))^2] \to 0$. But, as we saw earlier this needs m datsets, whereas we have only one dataset D.We can overcome this problem using **Bagging**(Bootstrap Aggregating)

Simulate drawing uniformly with replacement from the set D. i.e let $Q(X,Y|D)$ be a probability distribution that picks a training sample (x_i, y_i) from D uniformly at random. More formally, $Q((x_i, y_i)|D) = \frac{1}{n}\ \forall(x_i, y_i) \in D$ with $n = |D|$.We sample the set $D_i \tilde{Q}^n$, i.e $|D_i| =$ and D_i is picked with replacement from $Q|D$.The bagged classifier is $\hat{h_D} = \frac{1}{m}\sum_{i=1}^{m} h_{D_i}$.Bagging doesnt imply $\hat{h} \to \bar{h}$ as Weak Law of Large Numbers doesnt apply here(W.L.L.N only works for i.i.d. samples).However, in practice bagging still reduces variance very effectively.

unfinished

12 Random Forest

Decision trees as such are not great classifiers because of bias-variance problem.The high variance problem of Decision trees can be reuced by using Bagging. One of the most famous and useful bagged algorithms is the Random Forest! A Random Forest is essentially nothing else but bagged decision trees, with a slightly modified splitting criteria.

1. Sample m data sets $D_1, ..., D_m$ from D with replacement.

2. For each D_j train a full decision tree $h_j()$ $(max - depth\ = \infty)$ with one small modification: before each split randomly subsample $k \geq d$ features (without replacement) and only consider these for your split. (This further increases the variance of the trees.)

3. The Final Classifier is $h(x) = \frac{1}{m}\sum_{i=1}^{m} h_j(x)$

The hyperparameters involved in a random forest are m and k A good choice for k is $k = \sqrt{d}$ where d denotes the number of features.We can set m as large as you can afford.