ISSN (online): 2349-784X

A Review of Perspectives on Establishing the Cost Function, Regularization and Optimization for Machine Learning Techniques Along with Applications

Advait Pravin Savant
Sardar Patel Institute of Technology, India

Abstract

Philosophers across ages have wondered if a machine could perform tasks that require human cognitive abilities. With the formalization of computational theory and the advent of electronic computers, scientists have worked on developing computational models for intelligence and learning. Machine learning is the field of AI that provides computer systems the ability to improve their efficiency at performing a certain task as measured by a performance measure based on the experience that is given. With the increase in processing power of modern computers along with telecommunication technology, there is large amount of data that is available which enables us to build models of complex systems and extract information from the system, create representations of knowledge, recognize patterns from data and improve our ability to solve complex problems. From astronomical data analysis to credit card fraud detection, from medical data analysis to stock market predictions, today machine learning is everywhere. In this paper I attempt to review, describe and compare certain machine learning paradigms.
Keywords: Machine Learning, Cost Function, Regularization, Entropy, Gradient Descent

I. INTRODUCTION

Supervised machine learning which is what we are going to discuss in this paper is all about creating a hypothetical description of a system and then tuning that framework based on the training samples that we have. We can predict the values of a function(regression) or classify patterns(classification) using our attained model. The performance of our model will be tested on test data which it has not seen before so that we see to it that the model acquires the ability to generalize and understand the underlying dependencies in the data. We will cover multivariate linear regression for the extrapolation of a continuous function and logistic regression, kernel based Support Vector machines for classification problems. We will look at Bayesian and frequentist paradigms for machine learning, derive the error functions and derive the methods for optimization and look at concepts from linear algebra, probability theory and information theory as applied to machine learning.

II. MULTIVARIATE LINEAR REGRESSION

Consider an input vector x that contains the variables x(1), x(2) up to x(n) which is producing an output variable d. We do not know the functional dependence between x and y and in this model we propose a parameter vector w such that $d=\sum_{i=1}^{N} w(i)x(i) + b$

Here b is the bias term and we can denote it by w(0) taking x(0)=1 for convenience. For m such training examples of the input vector we have the corresponding outputs. We also assume a stationary environment wherein the nature of the relationship between the input vector and the output as captured by the parameter vector does not change across the training examples that we have sampled.

In the frequentist paradigm, w is considered to be a fixed vector whose value is estimated using some error function across the distribution of possible datasets. In the Bayesian approach we have one dataset which is the one that is observed and to quantify the uncertainty in the parameters we establish a probability distribution over the parameter vector w.

Let p(w/D) be the probability density that w is the parameter vector given the dataset D(posterior probability). Let p(D/w) be the probability density function for observing the dataset D given some parameter vector w (known as the likelihood function). p(w) is the prior probability density of w and p(D) is the probability density function for observing the dataset D.
From Bayes theorem we have

$$p(w/D) = (p(D/w)*p(w))/p(D)$$

We can express this as:
Posterior α likelihood * prior

If we solve for w based on maximizing the likelihood function we get the Maximum likelihood estimate and if we solve for w using the posterior function and maximize the posterior probability we get the maximum a posteriori estimate.

We will now formulate the cost functions for a maximum likelihood estimate and a MAP(maximum a posteriori estimate) of the parameter vector.

For a Gaussian environment, we have M independent and identically distributed samples. The Gaussianity is a reasonable assumption considering the central limit theorem and the widespread application of the Gaussian distribution. These samples of input and output pairs are drawn from the same underlying distribution. We have the expectational error as given by a Gaussian of 0 mean and σ^2 variance.

$$p(E) = \frac{1}{\sqrt{2 * \pi} * \sigma} e^{-\frac{E^2}{2*\sigma^2}}$$

In the prior information for the weights we what we have is that the weights themselves coming from a Gaussian of mean 0. This means we are preferring weights of smaller magnitude. We will use the same concept when we do regularization.
For each weight k in the parameter vector,

$$p(w_k) = \frac{1}{\sqrt{2 * \pi} * \sigma_w)} e^{-\frac{{w_k}^2}{2*{\sigma_w}^2}}$$

For the ith example we have : $E_i = d_i - w^T x_i$
From our discussion by substituting for E_i in the corresponding equation, it follows that :

$$p(d_i/w, x_i) = \frac{1}{\sqrt{2 * \pi} * \sigma} e^{-\frac{(d_i - w^T x_i)^2}{2*\sigma^2}}$$

Now we have,
$p(D/w)=\prod_{i=1}^{M} p(d_i/w, x_i)$

Substituting the value of the likelihood probabilities for each training example into the last equation, we will have a summation of terms in the exponent. Notice that these sum of squared terms is preceded by a negative sign. It is computationally more feasible to deal with the logarithm of the likelihood function and since the logarithm is a monotonically increasing function, maximizing the likelihood is equivalent to maximizing the log of the likelihood function and maximizing the negative of a sum of squares term is equivalent to minimizing the squared term. We then get the following error function:

$$L(w) = \frac{1}{2}\sum_{i=1}^{M} (w^T x_i - d_i)^2$$

If we differentiate with respect to w and solve by equating the gradient to zero we get the maximum likelihood estimate of the parameter vector.
A geometric perspective on the Maximum Likelihood solution:
Let X be the M*(N+1) input matrix with M training examples and (N+1) variables in the input vector.
The first value corresponding to each input vector is 1, this is for the bias terms. w is the weight vector.
t is the vector of M outputs. We have,

$$Xw = t$$

Now, as the number of training instances in our matrix is generally larger than the number of variables in the input vector, we wouldn't find an exact solution of w, we have to find the best fit.

Consider the column picture of matrix multiplication. There are N+1 rows in w and N+1 columns in X. The multiplication of matrix X with the vector w is given by a linear combination of the columns of X where each column is multiplied by a scalar which is the corresponding element is the vector w. In other words, the ith column of X is multiplied with the ith element in the column vector w.

Thus we obtain a vector which is a linear combination of vectors in the column space of X. We need to find a vector w such that the point p=Xw is closer to t than any other point in the column space of X. Therefore, the point p is the projection of t on the column space of X and the error vector (t-Xw) is perpendicular to that column space.
For every column z of X we get :

$$z^T(t - Xw) = 0$$

Stacking all such columns z of X we get the transpose of X. Hence:

$$X^T(t - Xw) = 0$$
$$w = (X^T X)^{-1} X^T t$$

This equation is also referred to as the Normal equation.

Now let me introduce the concept of regularization. This section is valid in general for all machine learning models. The behavior of the model we are creating is dependent upon the kind of functions that exist in its hypothesis space and also on the training data that we show the model. The training data itself comes from an underlying distribution which produces the input and output samples. The capacity of a model is its ability to fit a variety of functions. If we have a model of high complexity, it can so happen that we develop a representation that is overly dependent on the training data provided, we get a low error on the training data but a high error on the test data. The model here is said to be overfitting, for different sets of training data if the models are significantly different from each other our system is displaying a high variance. For a complex model, the expectation of these different model parameters is reasonably close to the underlying function, in this context, in general complex models are said to have a low bias. For models which are overly simple, they may not be able to obtain a sufficiently low error on the training set, we say that the

model is underfitting, the function learned by the model is significantly different from the underlying function and the model is said to have a high bias.

Regularization is the modification we make to our learning algorithm that is intended to reduce its generalization error and not its training error. Notice that in the maximum likelihood estimate we did not take into consideration the prior information about the weights, we have with us a reasonable estimation that the weights come from a zero mean Gaussian. We would like to drive the weights closer to the origin and shrink the larger parameter values in order to control overfitting. This is done by adding a weight decay term or the L2 norm to our error function for the unregularized least squares as we discussed earlier. This strategy is known as L2 regularization or Tikhonov regularization.

The error function becomes:

$$L(w) = \frac{1}{2}\sum_{i=1}^{M}(w^T x_i - d_i)^2 + \frac{\alpha}{2}w^T w$$

I would now like to show that this regularized least squares cost function is same as the MAP estimate we discussed earlier. We can draw from our earlier discussions that:

$$p(w) = \prod_{k=1}^{N} p(w_k)$$

Where

$$p(w_k) = \frac{1}{\sqrt{2*\pi}*\sigma_w} e^{-\frac{{w_k}^2}{2*{\sigma_w}^2}}$$

We can find the posterior probability density by multiplying the likelihood function with the probability corresponding to the weight vector.

Recall that for the ith example,

$$p(d_i/w, x_i) = \frac{1}{\sqrt{2*\pi}*\sigma} e^{-\frac{(d_i - w^T x_i)^2}{2*\sigma^2}}$$

And,

$$p(D/w) = \prod_{i=1}^{M} p(d_i/w, x_i)$$

Taking the product of the likelihood probabilities for all examples and also multiplying the probabilities for the weights, we get in the exponential a sum of squared errors for all training example along with a sum of all the weights in the parameter vector.

Posterior probability $\alpha\ e^{\left(-\frac{1}{2*\sigma^2}*\sum_{i=1}^{M}(w^T x_i - d_i)^2 - \frac{1}{2*{\sigma_w}^2}\|w\|^2\right)}$

As we want to maximize the probability with respect to w , this is similar to maximizing the log of the probability which is analogous to minimizing the negative log of that value. Thus we will have a new error function corresponding to the MAP estimate.

$$L(w) = \frac{1}{2}\sum_{i=1}^{M}(w^T x_i - d_i)^2 + \frac{\alpha}{2}w^T w$$

This is the same as regularized least squares error function.

By taking the derivative and equating to zero we get:

$$w = (X^T X + \alpha I)^{-1} X^T t$$

Now we will look at the relationship between the parameter vectors obtained from the MAP estimate and the ML estimate.

Let E'(w) be the regularized cost function and E(w) be the un-regularized cost function. Let w* be the vector corresponding to the optimum value of E(w). Let w' be the vector corresponding to which E'(w) is optimal.

Using the Taylor series expansion of E(w) around w* and substituting w' we get,

$$E(w') = E(w*) + (w' - w*)^T \nabla E(w*) + \frac{1}{2}(w' - w*)^T H(w*)(w - w*)$$

Since the function E(w) is a quadratic function, this approximation is perfect. H is the hessian matrix of E(w) evaluated at w*. Notice that the gradient of E(w) for w=w* is 0.

We have,

$$E'(w') = E(w') + \frac{\alpha}{2}w^T w$$

From the previous two equations, equating the derivative of E'(w) to zero for the parameter vector w' we get,

$$\alpha w' + H(w' - w*) = 0$$
$$(H + \alpha I)w' = Hw*$$
$$w' = (H + \alpha I)^{-1} H w*$$

Due to the symmetry of second order partial derivatives, H here is a symmetric matrix. The eigenvectors of a symmetric matrix are orthogonal. Therefore we can do the eigen-decomposition of H as $H=Q\Lambda Q^T$ wherein the transpose of Q is its inverse.

We now obtain,

$$w' = (Q\Lambda Q^T + \alpha I)^{-1} Q\Lambda Q^T w*$$
$$w' = (Q(\Lambda + \alpha I)Q^T)^{-1} Q\Lambda Q^T w*$$

$$w' = Q(\Lambda + \alpha I)^{-1}\Lambda Q^T w *$$

We see that this is a similarity transformation.

The inverse of the diagonal matrix is the reciprocal of the corresponding diagonal values and the multiplication of the subsequent two diagonal matrices will be a diagonal matrix with the ith element on the diagonal being $\lambda_i/(\lambda_i+\alpha)$ where λ is used to denote the eigenvalues of Q.[2]

From our frame of reference we have the vector w* which we transform into a vector in the basis of the eigenvectors of H. We then apply the scaling operation to that vector in the space spanned by that basis. We then again transform the scaled vector from the basis of the eigenvectors into a vector following the basis corresponding to our frame of reference. We see that when λ is greater than α ,the effect of regularization is relatively small. When the value of λ is lesser as compared to α , there is a shrinkage in that direction. When movement in one direction does not significantly reduce the cost function, the weights for such directions are decayed away by the use of regularization.

III. LOGISTIC REGRESSION

Logistic regression is an algorithm used for binary classification. Let the two classes be class 1 and class 2. Correspondingly, the output variable y is either 0 or 1. In scenarios where we want to find out whether a tumor is malignant or benign, we can use logistic regression. We calculate the weighted sum for all the parameters, inputs and then pass this into the sigmoid function.

$$z=w^T x + b$$

$$a=\sigma(z)=1/(1+\exp(-z))$$

The logistic function varies between zero and 1. As z tends to infinity the value of the sigmoid tends to 1 and as z tends to negative infinity the value of the sigmoid function tends to 0.The derivative of the sigmoid can be expresses in terms of the sigmoid as $\sigma(z)(1-\sigma(z))$.

We can interpret the value 'a' as the probability that y=1 for a given w and x. Hence (1-a) will be the probability that y=0 for that x and w.

We can see that the output variable y follows a Bernoulli distribution with the following equation:

$$p(y/(x,w)) = a^y(1-a)^{1-y}$$

This is the maximum likelihood probability function which we would like to maximize. We take the logarithm of the function and minimize the negative of it to get the following error function across m examples.

$$L(w) = -\frac{1}{m}\sum_{i=1}^{m}(y^i \log(a^i) + (1-y^i)\log(1-a^i))$$

We will now see the analogy of this cost function with cross entropy -a concept from information theory. Information theory was developed by Claude Shannon to find fundamental limits on signal processing and communication operations such as data compression etc. In this field of study we are interested in quantifying the amount of information in an event. Information is the resolution of uncertainty. The self-information for an event x=X is given by I(X)= –log(p(X)). This definition helps us to ensure that the lesser the likelihood of an event the higher its information content. Notice that independent events have additive information content for their joint probability given by the product rule.

The expected amount of information in an event drawn from a distribution P is given by

$$H(x) = E_P[I(x)]$$

$$H(x) = -E_P[\log(p(x))]$$

This is known as the entropy of the distribution of the random variable. Let p(x) be the underlying distribution from which the symbols of a message are drawn and suppose that we have modelled this distribution using q(x). In case of discrete random variables the extra amount of information needed to send the message would be given by Kullback-Leiber divergence between distributions the p(x) and q(x).

$$KL(p||q) = -E_p(\log(q(x)) - (-E_p\log(p(x)))$$

$$KL(p||q) = E_p[\log(P(x)/Q(x))]$$

We now define the cross entropy of the distributions p and q, which is related to the Kullback Leiber divergence as:

$$H(p,q) = H(p) + KL(p||q)$$

$$H(p,q) = -E_p(\log(q(x))$$

Notice that minimizing the KL divergence with respect to q is the same as minimizing the cross entropy. Now with regards to logistic regression if we consider q to be the values of a which is our predicted distribution and p to be the values of y which is the actual data generating distribution, notice that we get the same cost function which we got from the maximum likelihood estimate as when we use cross entropy as the loss function.

IV. OPTIMIZATION

In practice, finding the inverse in the matrix equations we previously described to find the parameter vector is computationally expensive, Gaussian elimination for finding the inverse takes $O(n^3)$ time. We use iterative optimization algorithms like gradient

descent. We conceptualize a multivariate error function as dependent on the weights. At any point on the error surface corresponding to a vector w, we will have a value of the error function. Consider the Taylor series expansion of E(w):

$$E(w + \alpha u) = E(w) + au^T \nabla E(w) \ldots.$$

We approximate the function for $\alpha << 0$ as we make a small change to w in the direction of u. We want the error function to decrease and therefore want $E(w+\alpha w) - E(w)$ to be as small as possible. We know that the gradient is the direction of steepest ascent and hence we will get the steepest descent if we move opposite the gradient. We can see from the equation that the dot product between the gradient at w and vector u must be a negative number with a large magnitude. Using the properties of the dot product we infer that the direction u is linearly opposite the gradient. We can write the update equation as:

$$w_{t+1} = w_t - \alpha \nabla E(w)$$

Here α is a small value which is the learning rate. We initialize the weights randomly (preferably not equal and not zero) and repeat the gradient descent algorithm iteratively until convergence wherein we obtain a sufficiently low error value or after a certain number of epochs.

In batch gradient descent, we consider all the examples in the training data when we calculate the gradient for one epoch and then we make the update in that epoch. This is because the error function is defined with respect to all the possible data points. This is computationally expensive. A better strategy which experimentally works better for convex optimization problems is Stochastic gradient descent in which for every epoch, we calculate the gradient with respect to one training example and then next we make the update to the weights. We repeat this procedure for all training examples.

Other improvements of gradient descent such as momentum based gradient descent, Nesterov accelerated gradient descent etc have been developed.

V. SUPPORT VECTOR MACHINES

If we have a complex non linearly separable problem in an input space, we can transform the input vectors to a high dimensional hidden feature space using a non linear mapping between the elements of an input vector to the elements of the corresponding transformed vector in the hidden feature space, we can significantly increase the likelihood that the pattern classification problem becomes linearly separable in that space. This is the work of Cover (1965). Let x be a input vector in m_0 dimensional space. We define a vector $\varphi(x)$ which is made up of real valued functions $\phi_i(x)$ where (i) goes from 1 to m_1. A dichotomy(C1,C2) is said to be linearly separable if there exists an m_1 dimensional vector w such that,

If x belongs to C1,

$w^T \phi(x) > 0$

If x belongs to C2,

$w^T \phi(x) < 0$

Where $\varphi(x)=[\phi_1(x),\phi_2(x)\ldots..\phi_{m1}(x)]$

We now define a kernel function as an inner product between the vectors in the hidden space.

$$k(x, x') = \phi(x)^T \phi(x')$$

Consider a binary linearly separable problem in the hidden space with the equation of the decision surface in the form of a hyperplane being,

$$w_0^T \phi(x) + b = 0$$

We write this as $g(\varphi(x))=0$

For elements which lie in C1 the value of the function will be greater than zero and for elements which lie in C2 the value of the function will be less than zero. We can rescale w_0 and b such that the elements which lie in class C1 give a value greater than equal to 1 and the elements which lie in class C2 give a value less than equal to -1 for that function. There are vectors in the hidden space which lie closest to the hyperplane such that the value of the function as represented by the hyperplane for these elements is 1 or -1.We call them the support vectors.

The distance between the support vectors $\varphi(s)$ and the hyperplane is from the property of hyperplanes given by $\varphi(s)/||w_0||$ which becomes,

$1/||w_0||$ for class C1 and $-1/||w_0||$ for class C2.

The margin of separation between the two classes is thus given by $2/||w_0||$

The support vector machine finds an optimal hyperplane which maximizes the margin of separation between binary classes , from the formula we see that this is equivalent to minimizing the Euclidean norm. For each training example $\{\varphi(x_i),d_i\}$ and d_i equals one if it belongs to class C1 and equals -1 if the input belongs to class C2.We have the following constraint for all N examples,

$$d_i(w^T \phi(x_i) + b) \geq 1$$

This is a constrained optimization problem in which we minimize the norm of w. Using the Lagrange multipliers we get the following cost function.

$$J(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i [d_i(w^T \phi(x_i) + b) - 1] \text{ -}$$

The alphas are non negative multipliers ,the function has to be minimized with respect to w,b and maximized with respect to α.If the constraint is not satisfied then the then the cost function grows unbounded and gives an infeasible solution while when the

constraint is satisfied the cost function comes down to the norm of the weight vector. These properties enable us to find the optimal feasible solution. This approach is known as the Karush-Kuhn-Tucker approach.
Differentiating J with respect to w and equating to zero we get,

$$w = \sum_{i=1}^{N} \alpha_i d_i \, \phi(x_i)$$

Differentiating J with respect to b and equating to zero we get,

$$\sum_{i=1}^{N} \alpha_i d_i = 0$$

Looking at the cost function one more thing to note is that whenever the constraint is not satisfied as an equality, we get the value of the corresponding multiplier to be zero. In other words, only for the support vectors where the constraint is satisfied as an equality we get the value of the multiplier to be a non zero value. This property comes from the Karush Kuhn Tucker conditions.
If we expand the cost function and substitute the values we obtained from the earlier two equations, we can express the cost function in terms of α and the training data. This is known as the dual representation of the cost function.

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \phi(x_i)^T \phi(x_j)$$

Subject to the constraints that α values are greater than zero and the summation of alpha and d values across all N examples is 0.
The advantage of the dual formulation is that we can use the substitution in the cost function,

$$k(x, x') = \phi(x)^T \phi(x')$$

This substitution is known as the kernel trick. We can express the function entirely in terms of the kernel which enables us to work directly with the kernel and avoid bringing in the feature vector φ(x). For a hidden space where the dimensionality of φ(x) is very high, we can do our calculations in a more feasible manner using the kernel trick. Notice that α values are non zero only for the support vectors, our solution vector is sparse. Having determined the optimum Lagrange multipliers, we compute the value of the weight vector w using the equation we described earlier. Here we will vary the iterator over the S support vectors.

$$w_0 = \sum_{i=1}^{S} \alpha_i d_i \, \phi(x_i)$$

We can choose any support vector from one of the two classes and use the following equation (let us say we select the class where the d value is 1),

$$w_0^T \phi(x) + b = 1$$

We will find a more numerically stable solution if we take the average over all the support vectors.

VI. IMPLEMENTATION

For an ecommerce website we analyze the parameters of customers in order to predict the amount of money he or she will spend on goods using the website. We see that regularization helps in giving better results on the training data. We also analyze the data for an advertising website and classify customers based on whether or not they click on an advertisement. We use logistic regression and a support vector machine for this purpose. We draw the confusion matrix for both cases and find the precision, recall. We have thus implemented the three paradigms we discussed.

VII. CONCLUSION

We have seen various perspectives of analysis for three machine learning models, we saw the geometric meaning of the normal equation, we reviewed the concept of regularization and we saw the similarity transformation based relation between the regularized and unregularized solution for the parameter vector of a least squares problem. We saw two angles of establishing the cost function of a logistic regression problem, we discussed techniques for optimization and we also discussed Support Vector Machines and the kernel trick for nonlinear pattern classification problems. We then applied these techniques for real world problems.

REFERENCES

[1] Christopher Bishop,Pattern Recognition and Machine learning,Springer,2006
[2] Goodfellow, Bengio and Courville, Deep Learning, MIT press,2016
[3] Evgeniou and Pontil, Support Vector Machines :Theory and Application,ACAI,2001
[4] Marijana Zekić-Sušac, Nataša Šarlija , Adela Has and Ana Bilandžić .Predicting company growth using logistic regression and neural networks. Croatian Operational Research Review,2016
[5] Saimon Haykin, Neural Networks and Learning machines,Pearson,2009
[6] Gilbert Strang,Linear Algebra and its applications,MIT press,2006.