1 Curve Fitting

Assume a *hypothesis space* of all polynomial functions

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M$$

Define the *error function* by

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

where (x_i, t_i) is the i^{th} observed sample. To ensure that the dimensionality of the error function is the same as that of the output data, and that the error is not effected by the sample size, we can use the *root-mean-square error* $E_{RMS} = \sqrt{2E(\mathbf{w}^*)/N}$.

To prevent over-fitting for large M, use *regularization*:

$$\tilde{E}(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}||\mathbf{w}||^2.$$

2 The Normal Gaussian

The *likelihood function* of the normal Gaussian for given data points $\mathbf{x} = (x_1, ..., x_D)^T$ is given by:

$$p(\mathbf{x}|\mu, \sigma^2) = \prod_{n=1}^{N}\mathcal{N}(x_n|\mu, \sigma^2)$$

To estimate the Guassian that the data has been taken from, we can maximize the likelihood function. For practicle reasons, we will maximize the *log-likelihood function.*

$$\ln(\mathbf{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi).$$

Solving this we get

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}x_n, \quad \sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu_{ML})^2$$

Later, we will highlight the significant limitations of the maximum likelihood approach. In particular, the maximum likelihood approach systematically underestimates the variance of the underlying distribution. Let's assume that we know that $(x_1, ..., x_N)$ are samples take from the distribution $\mathcal{N}(\mu, \sigma^2)$. Then, we can see that the expected values of maximum likelihood mean and variance are:

$$\mathbb{E}[\mu_{ML}] = \mu, \quad \mathbb{E}[\sigma^2_{ML}] = \left(\frac{N-1}{N}\right)\sigma^2$$

so, we see that the expected calculated maximum likelihood variance will be underestimated. So, an *unbiased* correction would give

$$\tilde{\sigma}^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \mu_{ML})^2$$

As we shall see, the issue of bias in maximum likelihood lies at the root of the over-fitting problem.

2.1 Curve fitting revisited

We want a solution $y(x, \mathbf{w})$, we want to make each of the probabilities $p(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1})$ as large as possible. Specifically, we want to maximize

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N}\mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}).$$

given the training data $\{\mathbf{x}, \mathbf{t}\}$. Translating this to a log-likelihood, we get

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi).$$

Since we are maximizing with respect to $\mathbf{w}$, we see that this is the same as minimazing the error function $E(\mathbf{w})$. Maximizing with respect to β, we get

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{ML}) - t_n\}^2.$$

Now, let us introduce a *prior distribution* (*i.e.*, a probability assigned to each of the guesses for $\mathbf{w}$) $p(\mathbf{w}|\alpha)$. The variable α here is called a *hyperparameter* since it controls the distribution of the model parameters. For simplicity assume

$$p(\mathbf{w}|\alpha) = \mathcal{N}(w|0, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2}\exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

So, we have that the likelihood of our guess for $\mathbf{w}$ is linearly dependent to

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

Maximizing this likelihood is equalent to minimizing

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}.$$

Which is equivalent to minimizing the regularized sum-of-squares error function.

2.2 Bayesian curve fitting

In a fully Bayesian approach, we should consistently apply the sum and product rules of probability, which requires, as we shall see shortly, that we integrate over all values of $\mathbf{w}$. Let us try to fit an x to a x given the training data $(\mathbf{x}, \mathbf{t})$:

$$p(t|x, \mathbf{x}, \mathbf{t}) = \int p(t|x, \mathbf{w})p(w|\mathbf{x}, \mathbf{t})\, \mathtt{d}\mathbf{w}.$$

It can be shown that the solution to this integral will give $p(t|x, \mathbf{x}, \mathbf{t}) = \mathcal{N}(t|m(x), s^2(x))$, where the mean and variance are given by

$$m(x) = \beta\phi(x)^T\mathbf{S}\sum_{n=1}^{N}\phi(x_n)t_n, \qquad s^2(x) = \beta^{-1} + \phi(x)^T\mathbf{S}\phi(x).$$

And the matrix $\mathbf{S}$ is given by

$$\mathbf{S}^{-1} = \alpha\mathbf{I}\beta\sum_{n=1}^{N}\phi(x_n)\phi(x_n)^T, \qquad \phi(x) = \begin{pmatrix}1 & x & \cdots & x^M\end{pmatrix}^T.$$

Note that both the variance and the mean of the predictive distribution are dependant on x. Note also that the first term in $s^2(x)$ represents the uncertainty in the predicted value of t due to the noise on the target variables and was expressed already in the maximum likelihood predictive distribution. However, the second term arises from the uncertainty in the parameters $\mathbf{w}$ and is a consequence of the Bayesian treatment.

2.3 Model Selection

If data is plentiful, then one approach is simply to use some of the available data to train a range of models, or given model with a range of values for its complexity parameters, and then to compare them on independent data, sometimes called the *validation set*, and select the one having the best predictive performance. If the model design is iterated many times using a limited size data set, then some over-fitting to the validation data can occur and so it may be necessary to keep aside a third *test set* on which the performance of the selected model is finally evaluated.

When data is not in abundance, we can use *cross-validation*. In cross-validation, the training data is divided into S groups. The model is then trained S times on data from groups $(2, ..., S), (1, 3, ..., S), ...$ respectively. The resulting pattern matchers are then tested on their respective remaining set. This method allows us to sort-of train the model on data as if it was S times larger than the training data we have.

2.4 Decision Theory

Consider a classification problem where we need to decide based on some input vector $\mathbf{x}$ which class out of $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n$ it belongs to. In machine learning, we will often make use of Bayes' law:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k}{p(\mathbf{x})}$$

2.5 Minimizing the misclassification rate

Let's aim to minimize the misclassification rate. Assume that our model assigns points to the set $\mathcal{C}_i$ if they belong to the set $\mathcal{R}_i$. Assume for simplicity that this is a binary classification problem.

$$\begin{aligned} p(\mathbf{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2)\, d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1)\, d\mathbf{x} \end{aligned}$$

2.6 Minimizing the expected loss

Usually the penalty of misclassifications depends on which class was misclassified by which other class, and not all misclassifications bare the same penalty. Therefore, a *loss function* is introduced. So, assume that the cost of misclassifying a point belonging to $\mathcal{C}_k$ as belonging to $\mathcal{C}_j$. Note that we do not know when a misclassification happens, and we also do not know which class the point truly belongs to if a misclassification occurs. So, we can only minimize the expected loss:

$$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, \mathcal{C}_k) \, d\mathbf{x}.$$

A *reject option* may be included to reject uncertain points as unclassified in order to prevent misclassification. Usually, this is done with a *threshold* which rejects all points $\mathbf{x}$ for which the maximum probability $p(C_k|\mathbf{x})$ over all the classes is less than some threshold θ.

We can extend the reject criterion to minimize the expected loss, when a loss matrix is given, taking account of the loss incurred when a reject decision is made.

We can identify three distinct ways of solving decision problems:

(a) First solve the inference problem of determining the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ for each class $\mathcal{C}_k$. Also infer probabilities $p(\mathcal{C}_k)$. Then use Bayes' theorem to find the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$. Note that

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k).$$

These are called *generative models.*

(b) First solve the inference problem of determining the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$, and then subsequently use decision theory to assign each new $\mathbf{x}$ to one of the classes. These are called *discriminative models.*

(c) Find a function $f(x)$, called a discrimination function, which maps points from the input space into the appropriate classes.

The first type of decision models might be advantageous because we can infer $p(\mathbf{x})$ from it. This can be useful for detecting new data points that have low probability under the model and for which the predictions may be of low accuracy, known as *outlier detection.* However, since $\mathbf{x}$ often has high dimensionality, determining class-conditional distributions might require very large training data.

The second approach is a compromise between the other two and can still be used for

(a) **Minimizing risk.** Consider a problem in which the elements of the loss matrix are subjected to revision from time to time. By knowing the posterior probabilities, we can trivially revise the minimum risk decision criterion. For discriminant funcitons, there is not enough data to do this directly.

(b) **Reject option.**

(c) **Compensating for class priors.** In order to have good results from the training data, it would be wise to balance the points artificially so that each class contains about the same number of test points. However, doing this will dissrupt the probabilities $p(\mathcal{C}_k|\mathbf{x})$ since they are linearly proportional to $\mathcal{C}_k$). So, if we have a posterior probability from our artificial data, we will need to normalize the probabilities by dividing by the ratio of members of each class to the size of the training data and multiplying it by the actual amount by which members of this class actually occur in reality. Note that we cannot do this without knowing the posterior distribution.

(d) **Combining models.** If aside from X-rays, we used blood results to build a diagnostics model and assuming we had access to the posterior probabilities for both the X-ray and blood test models, we could simply apply probability laws to combine the two results into a single probability distribution. If we only knew the discriminant function, we would have to build up a whole new model that contains both X-ray and blood test results in the input space.

We can extend the use of loss functions onto non-discrete problems where we try to minimize the loss $L(t, y(\mathbf{x}))$ given training data $(\mathbf{x}, \mathbf{t})$

$$\mathbb{E}[L] = \iint L(t, y(\mathbf{x}))p(\mathbf{x}, t)\, d\mathbf{x}\, dt.$$

3 Information Theory

The information transmited through an observation of a data point $x$is defined by $h(x) = -\log_2 p(x)$. The entropy of a random variable x is defined by

$$H[x] = -\sum_x p(x)\log_2 p(x).$$

Note the relationship between entropy and a shortest code length to transmit a piece of data. Entropy is the lower bound on the number of bits needed to transmit the state of a random variable.

For a density defined over multiple continuous variables, denoted collectively by the vector $\mathbf{x}$, the differential entropy is given by

$$H[x] = -\int p(x) \ln p(x)\, d\mathbf{x}.$$

For the descrete case, maximum entropy is achieved when the data is equally distributed among the possible buckets.

Assuming we fix the mean and variance of a probability distribution, maximum enthropy is achieved for the Gaussian distribution. The entropy in that case is

$$H[x] = \frac{1}{2}\{1 + \ln(2\pi\sigma^2)\}.$$

We also introduce *conditional enthropy* as

$$H[\mathbf{y}|\mathbf{x}] = -\iint p(\mathbf{x}, \mathbf{y}) \ln p(\mathbf{y}|\mathbf{x})\, d\mathbf{y}\, d\mathbf{x}$$

From here we see that $H[x, y] = H[y|x] = H[x]$ as expected.

The information gain of an attribute A is the expected reduction of entropy caused by partitioning on this attribute

$$Gain(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v).$$

4 Linear Models for Classification

Goal of classification is to assign inputs to class labels $\mathcal{C}_k$ where $k = 1, ..., K$. The boundaries between these classes in the feature space are defined by *decision boundaries.* Linear models implies that we the hypothesis space is all linear function of the features. If the classes are such that they can be exactly divided with a linear model we call them *linearly separable.*

To make linear models more robust, we can assign an *activation function* to the fitted linear model: $y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + w_0)$. These are called *generalized linear models.*

4.1 Discriminant Functions

Linear discriminant functions are usually described as a linear function of the input vector $y(x) = \mathbf{w}^T\mathbf{x} + w_0$, where $\mathbf{w}$ is called a *weight vector* and w_0 is called the *bias*.

Since least squares classification method assumes Gaussian conditional distribution it is far from ideal for linear classification problems.

4.1.1 Fisher's linear discriminant

One way to view a linear classification model is in terms of dimensionality reduction. Consider first the case of two classes, and suppose we take the D-dimensional input vector $\mathbf{x}$ and project it down to one dimension using $y = \mathbf{w}^T\mathbf{x}$. If we add a threshold to y between two classes, we get binary classification. In general, the projection onto one dimension leads to a considerable loss of information, and classes that are well separated in the original D-dimensional space may become strongly overlapping in one dimension. All we can do with this is choose a weight vector that provides maximum class separation.

One approach to solving the problem is try and maximize the projected distance of the means of the two classes. The issue with this approach is that the covariances of the class distributions are strongly non-diagonal. That is, aside from focusing just on providing maximum separation of class means, we must also make the within-class variances of the projected class points as small as possible.

The Fisher criterion is defined to be the raio of the between-class variance to the within-class variance and is given by

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

where m_1, m_2 are the respective means of the projections of each class.

4.1.2 The perceptron algorithm

The Perceptron focuses on a two-class model in which the input vector $\mathbf{x}$ is first transformed using a fixed nonlinear transformation to give a feature vector $\phi(\mathbf{x})$, and this is then used to construct a generalized linear model of the form $y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x}))$ where the nonlinear activation function is a simple step function.

Perceptron focuses on finding the vector $\mathbf{w}$ such that patterns $\mathbf{x}_n$ in class $\mathcal{C}_1$ will have $\mathbf{w}^T\phi(\mathbf{x}_n) > 0$, whereas patterns $\mathbf{x}_n$ in class $\mathcal{C}_2$ have $\mathbf{w}^T\phi(\mathbf{x}_n) <$

0. The *perception criterion* is given by

$$E_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\phi_n t_n$$

where $\phi_n = \phi(\mathbf{x}_n)$, $\mathcal{M}$ denotes the set of all misclassified patterns, and t_n is the correct classification of input data x_n.

The reach optimal solution, we apply the stochastic gradient descent algorithm to this error function. The change in the weight vector is given by:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta\phi_n t_n$$

where η is the learning rate, and τ is an integer that indexes the current step of the algorithm.

The algorithm runs the current model consecutively on all traning data. If the current data point is correctly classified, the weights are not changed. If the data point is not correctly classified, we add/subract the vector $\phi(x_n)$ depending on which class it actually belongs to.

We can see that each correction will reduce the error contribution from that specific missclassified point. However, this does not imply that the total error will be reduced.

It can be proven that if there exists an exact solution (i.e. the data is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps.

In practice it is hard to determine if two data classes are linearly separable or not, until we run the perceptron on it. For classes that are not linearly separable, the perceptron will never converge.

4.2 Probabilistic Generative Models

Goal is to model class-conditional probabilities $p(\mathbf{x}|\mathcal{C}_k)$. Use sigmoid as an active function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

4.2.1 Logistic Regression

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma(\mathbf{w}^T\phi)$$

For a large number of features, logistic regression works better than the perceptron, since the complexity of the algorithm grows linearly with the number of features in the logistic regression case.

Using maximum likelihood, we can determine the parameters of the logistic regression model. Note that

$$\frac{d\sigma}{dx} = \sigma(1-\sigma).$$

Therefore the likelihood function can be written as

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}\{1-y_n\}^{1-t_n}$$

where $\mathbf{t} = (t_1, ..., t_N)^T$ is the set of all training results and $y_n = p(\mathcal{C}_1|\phi(x_n))$. Also note that $t_i \in \{0,1\}$ We can define the error function by taking the negative logarithm of the likelihood, which gives the *cross-entropy* error function in the form

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$

where $y_n = \sigma(\mathbf{w}^T\phi(x_n))$. Taking the gradient of the error function we get

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi(x_n)$$

This error gradient turns out to be identical to the gradient of the sum-of-squares error function for the linear regression model.

Note that maximum likelihood can exhibit sever over-fitting for data sets that are linearly separable.