Learning from Data
Some Real World Considerations

Course lecturer: Amos Storkey
a.storkey@ed.ac.uk
Course page : http://www.anc.ed.ac.uk/∼amos/lfd/

There are a number of issues that arise when working with real data. These fall into two main areas, pre- and post-processing.

- Pre-processing: Types of data and how to code them.
- Pre-processing: Feature selection and feature extraction.
- Post-processing: The reject option and loss.

We also consider what to do if there are missing inputs to a classifier.

1 Coding data

For some machine learning algorithms, these data types need to be coded so that they can be input into the algorithm. For categorical data, the usual encoding method is 1-of-n encoding. That is, if the variable can take on n different values, we make a binary vector of length n and turn the relevant bit on, and leave all the others off. This kind of coding can be used for inputs to neural networks and to the k-nearest neighbours algorithm. Notice that a decision tree does not need to recode categorical data.

For ordinal data there are a couple of alternatives. As there is a natural ranking of the categories, it makes sense to code each category into a single real-valued number. E.g. we might have $cool = 0.1$, $cold = 0.5$, $warm = 0.8$ and $hot = 1.0$. The choice of the values between 0 and 1 might seem arbitrary, but if we have a number of examples drawn from a distribution, then we can work out the relative frequencies of each class and make a percentile coding. For example if we have 10, 40, 30 and 20 examples of cool, cold, warm and hot respectively, then we obtain the numbers given above.

A second method for dealing with ordinal data is to use *thermometer* coding. For n categories we take $n-1$ bits. For the first category we turn on 0 bits, for the second 1, and so on up to the nth category for which we turn on all $n-1$ bits. If we take a simple Euclidean distance between these vectors, we see that a pair of entries that are further apart on the scale will have a larger distance. Compare this to 1-of-n encoding, where the distance between any entries that are not identical is the same.

For numeric data it would be possible to use this directly as an input to a neural network or k-nearest neighbour (k-NN) classifier. However, say that one of our variables is the height of a person. Should it be entered in metres or millimetres? For a k-NN classifier this will clearly make a huge difference unless we have scaling factors on each dimension to compensate. The usual choice is to rescale each variable x to a new variable $\tilde{x}$ having zero mean and unit standard deviation. This is achieved by setting

$$\tilde{x} = \frac{x - \overline{x}}{s}, \tag{1.1}$$

where $\overline{x}$ is the mean of the variable (in the training data available) and s is the corresponding standard deviation. This ensures that all inputs have equal magnitude initially. It is also good practice in numerical algorithms to scale variables so that they are of $O(1)$ when possible.

1.1 Scaling the weights in a neural network

For neural networks it is also useful to choose the scale of the initial values of the weights to avoid saturation of the hidden units. For example consider a hidden unit j which computes the weighted sum of inputs $s_j = \sum_{i=1}^{n} w_{ji} x_i$. If we assume that the x_i's are $O(1)$ then we can calculate the mean and variance of s_j if the weights are chosen from some distribution. If say the w's are chosen from a Gaussian distribution with mean 0 and standard deviation σ_w, then s_j will have mean 0 and variance $E[s_j^2] = \sigma_w^2 \sum_{i=1}^{n} x_i^2 \simeq n\sigma_w^2$. Hence if we choose

$$\sigma_w = \frac{1}{\sqrt{n}} \tag{1.2}$$

then the input to the hidden unit will be $O(1)$ and this will avoid saturation of the hidden unit. This is a good idea as units that are hard on or off give rise to very small training gradients. This scaling is used in the `netlab` software which you will be using for neural network experiments.

2 Dimensionality reduction: Feature selection and feature extraction

Some types of data can give rise to a large number of inputs. Examples include images (with up to millions of pixels), audio data (sampled at high frequency) and the output of spectrometers. Applying this data raw to a machine learning algorithm would lead to many problems. For example with a neural net a million inputs wold give rise to several million weights in the network, and we know that these weights may not be well-determined by limited amounts of training data. Also we do not believe that each entry in these kinds of data is independent; for example for images the structure of the world that we look at which gives rise the scenes that we capture means there are correlations between the nearby pixels.

Faced with this problem we can either carry out

- feature selection, or
- feature extraction.

Feature selection means choosing a subsect of the original features as variables for our learning algorithm. In feature extraction we create some new features based on the original ones.

Examples of feature extraction are (i) the creation of some hand-crafted features (e.g. the number of holes in a binary image, which might be a useful feature for character recognition) or (ii) the use of automatic dimensionality-reduction mechanisms. An example of the latter is Principal Components Analysis (PCA), where k linear combinations of the original n features are created (see the section of the course on unsupervised learning for further details on this).

For feature selection, we could consider using all subsets of size k out of the original d features. However, there are $\binom{d}{k}$ such combinations and it would be very computationally expensive to train and evaluate a machine learner for each combination. Instead, forward or backward stepwise strategies are usually used.

In forward selection, we start with no features, and add them one at a time. At the first step, the feature that gives the best predictive accuracy is used. At subsequent steps, the next feature is added so that the joint predictive performance of all the features available so far is maximized. This is continued until a stopping criterion is satisfied (this may be a maximum number of features, or that overfitting is setting in an predictive performance is declining).

Alternatively in backwards selection, we start with all features and sequentially remove features to maximize predictive performance until a stopping criterion is satisfied.

Notice that stepwise strategies may not find the optimal combination of variables. This is easily seen with forward selection. If there are two variables which individually are poorly predictive, but together are very predictive, then a forward selection strategy may not select one of these features, as it cannot anticipate that the other feature could be selected next. Thus this combination of features will not be chosen even though together they are highly predictive. Backwards selection suffers less from this problem but with a large number of variables it is undesirable as overfitting is likely to be a problem from the start (unless steps are taken to control this).

3 Post-processing the output

3.1 The Reject Option

Some classifiers simply output the predicted class label. However, others such as neural networks output an estimate of $P(c_k|\mathbf{x})$, the probability of class k given input $\mathbf{x}$. In this case we might choose to reject some patterns, in the hope they are the difficult ones and that by throwing some out, the performance on the remaining patters is improved. For example the Post Office might require that an automatic postcode reader will deal with 95% of all letters, and that it obtains 99% or better correct classifications on those letters it does not reject.

For each output class k the classifier predicts $P(c_k|\mathbf{x})$, and the optimal classification strategy is to choose the class $i^\star = \text{argmax}_k P(c_k|\mathbf{x})$. However, if $\max_k P(c_k|\mathbf{x})$ is not very close to one, then there is a chance that errors can occur. Hence we can choose a threshold θ so that we carry out the classification iff

$$\max_k P(c_k|\mathbf{x}) > \theta,$$

otherwise we reject the pattern.

As θ moves form 0 to 1, the fraction of rejected patterns increases. We can plot an error-reject curve, as illustrated in Figure 1.

3.2 Loss matrices

The consequences of a misclassification can be more serious in some situations than in others. For example, consider a classifier used to screen patients for cancers. If the machine does not flag a cancer when one is present (a false negative), this is much more serious than creating a false positive (which could then be eliminated in further testing).

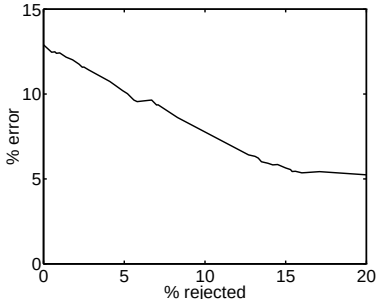

Figure 1: An error-reject curve. Notice the usual trend, that as a larger fraction of examples are rejected, the error rate decreases.

To take into account these consequences we need not only to predict the probabilities of each outcome, but the *loss* associated with the misclassification. Let L_{kj} denote the loss when a pattern from class $\mathcal{C}_k$ is assigned to class $\mathcal{C}_j$. In the cancer example, we might have

$$L = \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix}$$

where state 1 denotes "cancer" and state 2 denotes "normal".

Given a loss matrix, the optimal decision is the one that minimizes the expected loss (or *risk*). This is achieved by computing the risk of each decision (using the probabilities predicted by the net), and then choosing the decision with minimum risk.

4 Missing inputs

Suppose we have trained our predictor and want to make a prediction for a new input $\mathbf{x}$. If all the input attributes are present this is easy. If some are missing, we have to ask if those attributes are *missing at random* (MAR), or if there is a systematic reason why they are not present. For example, a doctor would tend not to order particular tests if (s)he felt that they would not be useful. In this latter case the deliberate non-measurement does convey information.

However, if the observations are MAR, how should we deal with them? Let the input vector $\mathbf{x}$ be partitioned as $\mathbf{x} = (\mathbf{x}_p, \mathbf{x}_m)$ where $\mathbf{x}_p$ denotes those attributes present, and $\mathbf{x}_m$ denotes the missing values. Then if we can model $P(\mathbf{x}_m|\mathbf{x}_p)$ we should average the predictions, weighted by this density. This is formally correct, but difficult to achieve in practice. A cruder version of this is to look for input patterns that have values on the present attributes similar to $\mathbf{x}_p$, and to use then as the sample. An even cruder approach is to replace the missing values by their average values in the dataset.

An alternative strategy (which can cope with attributes missing systematically) is to create a "no value" unit, which indicates that the particular attribute was not observed. For categorical variables this is easily achieved by adding an extra label.

5 And Finally ...

Some hopefully useful advice :

- Try to get as much training data as possible – this will improve both the performance of your model and your confidence in its predictions. If you get too much data that training is very slow, figuring out which datapoints to discard is not too difficult.
- Don't buy the story (that others might tell you) that black box methods are great. Usually, there are much better solutions available if you set your mind to the issue of trying to make a reasonable model for the data.
- Try to avoid methods where you cannot easily figure out if the reason for poor performance is due to a fundamental poor choice of model, or in difficulties in the numerical application of the model. This is really the nightmare scenario (and common to the neural network approach for example) since you don't know if you should just try another numerical approach (new optimisation procedure for example) or start fiddling with the parameters of your model.
- It is crucial you understand the assumptions behind any approach that you take. You are now the expert, and maybe your predictions are used in critical areas – tumour classification for example. If you don't understand well the fundamental model assumptions, there is really no science involved in your work, and you'll be unable to easily defend or justify your position.
- Try to understand what it is that people want you to achieve in a machine learning problem. There are often different issues involved and the choice of method can be dependent on what the user ultimately wants.
- And finally .... use your mind. Machine Intelligence is really just a way to encode your human intelligence. Have fun!