Introduction to Logistic Regression

Guy Lebanon

1 Binary Classification

Binary classification is the most basic task in machine learning, and yet the most frequent. Binary classifiers often serve as the foundation for many high tech ML applications such as ad placement, feed ranking, spam filtering, and recommendation systems.

Below, we assume that x is a d-dimensional vector $x = (x_1, \ldots, x_d)$ of real numbers, and y is a scalar denoting the label: +1 or -1. We will denote by θ the vector of parameters defining the classifier, whose dimensionality is the same as that of x: $\theta = (\theta_1, \ldots, \theta_d)$. The inner product between θ and x is defined as follows $\langle x, \theta \rangle = \theta_1 x_1 + \cdots + \theta_d x_d$.

The binary classification task is defined as follows: Given a vector of features x, assign a label y of +1 or -1. A binary classifier is a rule that makes such mapping for arbitrary vectors x. The classifier is typicaly learned based on training data composed of n labeled vectors: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$. Note that each $x^{(i)}$ is a vector and $y^{(i)}$ its +1 or -1 label.

For example in the case of the LinkedIn feed, x can contain measurements characterizing the user and item, and y represents whether the user clicks or not on the item. An example for a feature could be $x_5 = 1$ if the user has more than 500 connections and the item is a connection update. There could be thousands or more such features. The training data is a sequence of user-item and click/no-click information, and at serving time the classifier tries to predict whether the user will click on the feed item or not.

2 Linear Classifiers and Hyperplanes

A linear classifer is a classifier that has the following algebraic form: $y = \text{sign}(\langle \theta, x \rangle)$. That is, the predicted label of the vector x is the sign of the inner product of that vector with another vector θ that is called the parameter vector of the classifier. If $\langle x, \theta \rangle$ is positive the predicted class is +1 and if it is negative the predicted class is -1.

Despite their simplicity (and probably because of it), linear classifiers are the most widely used. There are several reasons: (a) they are easy to train, (b) they can predict labels very fast at serve time (if x is sparse i.e., mostly zeros, the inner product computation $\langle \theta, x \rangle$ is extremeley fast), (c) and we know quite well the statistical theory of linear classifiers leading to effective modeling strategies. Arguably, the most widely used linear classifier is logistic regression. Logistic regression, or its variations, power main functions in big companies like Google, LinkedIn, and Amazon, as well as in small startups.

A note on dimensionality: For visualization purposes we should assume that there are only two features i.e., $d = 2$. In reality, d may be quite higher such as $10^4 - 10^6$. Linear classifiers excel at such high-dimensional cases due to their simplicity, their attractive computational load, and the nice statistical properties.

There is another reason why linear classifiers are called linear besides the fact that the classification rule is defined by a linear form $\langle \theta, x \rangle$. That has to do with the fact that if we draw a Cartesian coordinate system and mark the data vectors x as points in that coordinate system, the classification rule of linear classifier boils down to a linear plane decision boundary that is perpendicular to θ. In other words, θ (interpreted as a vector in that coordinate systme) defines a hyperplane that is perpendicular to it, and that hyperplane divides the space of data vectors in half - all vectors on one side are predicted to have positive labels and all vectors on the other side are predicted to have negative labels. This can be seen by recalling that $\langle x, \theta \rangle$ equals projection of the vector x on the vector θ times the norm of θ.

In order to make sure that the decision boundary does not have to pass through the origin, we need to in effect to have a classification rule $x_1\theta_1 + \cdots + x_d\theta_d + c = \langle x, \theta \rangle + c$. The added term c is called a bias term and it enables the decision boundary to not pass through the origin. To simplify notation, however, we assume that all vectors x inculde 1 as one of their dimension, hence the inner product $\langle x, \theta \rangle$ already includes the offset term implicitly.

Note that we can increase the dimensionality of the data vectors x by adding powers of x or other transformations. For example, given a two dimensional data vector $x = (x_1, x_2)$ we can create a sixth dimensional corresponding vector $\tilde{x}$, defined by $\tilde{x} = (1, x_1, x_2, x_1^2, x_2^2, x_1x_2)$. We can then train a linear classifier on the transformed data vectors $(\tilde{x}^{(1)}, y^{(1)}), \ldots, (\tilde{x}^{(d)}, y^{(n)})$ rather than on the original data $(x^{(1)}, y^{(1)}), \ldots, (x^{(d)}, y^{(n)})$. The resulting classifier will be linear in the coordinate system of $\tilde{x}$ but *non-linear* in the coordinate system of x. Such "feature engineering" is often used in building non-linear classifiers using linear classifier implementations.

3 Probabilistic Classifiers and Maximum Likelihood

The above definition of classifier defines a map from a vector of features x to a +1 or -1 label. It is useful to have also a measure of confidence of that prediction as well as a way to interpret that confidence in an objective manner. Probabilistic classifiers provide that tool by defining the probabilities of the labels +1 and -1 given the feature vector x. Recall the fact that the two probabilities must be non-negative and sum to one. The probabilities of the two labels given a feature vector are written as $p_\theta(Y = 1|X = x)$ and $p_\theta(Y = -1|X = x)$ where θ is a parameter vector that defines the clasifier.

Statistical theory strongly suggests that prbabilistic classifiers should be learned from the labeled vectors $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ using the maximum likelihood estimator (MLE)[1],

[1]Unless you are a Bayesian in which case you do not belive in a single classifier representing the "truth" but rather in a collection of classifiers, each correct with some probability.

defined below

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \quad p_\theta(Y = y^{(1)}|X = x^{(1)}) \cdots p_\theta(Y = y^{(n)}|X = x^{(n)}) \tag{1}$$

$$= \arg\max_{\theta} \quad \log p_\theta(Y = y^{(1)}|X = x^{(1)}) + \cdots + \log p_\theta(Y = y^{(n)}|X = x^{(n)}). \tag{2}$$

There are several reasons for using the MLE: (a) it converges to the optimal solution in the limit of large data[2], and that convergence occurs at the fastest possible rate of convergence.

4 Logistic Regression

There are many probabilistic classifiers, but the most popular one is logistic regression whose probabilistic definition is given below

$$p(Y = y|X = x) = \frac{1}{1 + \exp(y\langle\theta, x\rangle)},$$

where y is either +1 or -1. We can ascertain that $p(Y = 1|x) + p(Y = -1|x) = 1$:

$$\frac{1}{1 + \exp(\langle\theta, x\rangle)} + \frac{1}{1 + \exp(-\langle\theta, x\rangle)} = \frac{1 + \exp(\langle\theta, x\rangle) + 1 + \exp(-\langle\theta, x\rangle)}{(1 + \exp(\langle\theta, x\rangle))(1 + \exp(-\langle\theta, x\rangle))} \tag{3}$$

$$= \frac{1 + \exp(\langle\theta, x\rangle) + 1 + \exp(-\langle\theta, x\rangle)}{1 + \exp(\langle\theta, x\rangle) + \exp(-\langle\theta, x\rangle) + 1} = 1. \tag{4}$$

Recall that if $p(Y = 1|x) > 0.5$ we have greater confidence of the label 1 than -1 and vice verse if $p(Y = 1|x) < -0.5$. Thus, the decision boundary is precisely the set of vectors x for which $p(Y = 1|x) = p(Y = -1|x) = 0.5$. This occurs when

$$\frac{1}{2} = \frac{1}{1 + \exp(\langle\theta, x\rangle)} \tag{5}$$

$$1 + \exp(\langle\theta, x\rangle) = 2 \tag{6}$$

$$\langle\theta, x\rangle = \log(1) = 0. \tag{7}$$

In other words, the decision boundary or the set of vectors where the logistic regression classifier is not sure how to classify is precisely the set of vectors x for which $\langle\theta, x\rangle = 0$, which implies that logistic regression is a linear classifier whose decision boundary is the hyperplane that is perpendicular to the vector θ.

If we want to predict the label associated with a feature vector x we can use the prediction rule $\text{sign}(\theta, x)$, but if we want to have a measure of confidence of that prediction, interpreted as a probability, we can use $p(Y = y|X = x) = 1 + \exp(y\langle\theta, x\rangle)$.

[2]assuming the data is generated based on the logistic regression model family and $n \to \infty$ while d is fixed.

5 Solving the MLE using Iterative Optimization

Substituting the logistic regression formula for $p(Y = y|X = x)$ in (2) we get the following optimization problem

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{n} \log \frac{1}{1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)} = \arg\max_{\theta} \sum_{i=1}^{n} -\log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)$$
$$= \arg\min_{\theta} \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right). \quad (8)$$

To get the θ vector that minimizes the above expression, the method of gradient descent is typically used: (a) initialize the dimensions of θ to random values, (b) for $j = 1, \ldots, d$, update $\theta_j \leftarrow \theta_j - \alpha \frac{\partial \sum_{i=1}^{n} \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$, (c) repeat step (b) until the updates become smaller than a threshold. The parameter α is the step size, and it is customary to let α decay as the gradient descent iterations increase. In cases when the data vectors $x^{(i)}$ are sparse, the above computation of the partial derivative can be made particularly fast. Since (8) is convex in θ, the gradient descent algorithm will reach the single global maximum regardelss of the starting point (though that maximum could be at $\theta_j \to \pm\infty$ for some j).

Gradient descent or its variations work effectively for non-massive data. When the data is very big, the method of stochastic gradient descent is typicallly used: (a) initialize the dimensions of θ vector to random values, (b) pick one labeled data vector $(x^{(i)}, y^{(i)})$ randomly, and update for each $j = 1, \ldots, d$: $\theta_i \leftarrow \theta_j - \alpha \frac{\partial \log\left(1+\exp(y^{(i)}\langle\theta, x^{(i)}\rangle)\right)}{\partial \theta_j}$ (c) repeat step (b) until the updates of the dimensions of θ become too small.

6 High Dimensions, Overfitting, and Regularization

When x is high dimensional the issue of overfitting comes up: there are too many parameters to estimate from the available labeled data, resulting in a trained classifier that fits random noise patterns that exist in the data, rather than the rule governing the likely behavior. For example, consider an extreme case (for illustration purposes) of $d = 10^6$ and $n = 2$. Obviously we cannot properly learn a million parameters and how they define the linear decision boundary in a million dimensional space based on 2 labeled data vectors.

A way to handle this situation is to add a regularization term to the maximum likelihood cost function (2): $\beta\theta_1^2 + \cdots + \beta\theta_d^2$ or $\beta|\theta_1| + \cdots + \beta|\theta_d|$. Such rule penalizes high values of θ and tempers the MLE in order to prevent it from achieving high values of $\theta_1, \ldots, \theta_d$ unless there is strong evidence in the data for it. The selection of β should be done carefully by experimentation, and in fact the best value of β depends on the situation at hand, and in particular (though not exclusively) on the relationship between d and n. It is customary to try different values of β, for example $\beta = 0.01, 0.1, 0.5, 1, 5, 10, 100$, and train the classifier for each of these values and later select the classifier that achives the best performance on a separate held out set of labeled vectors. The grid of possible values should be adjusted to make sure that we find a β close to the optimal value.

Notes

The Shiny App (Web App embedding R code) shows how logistic regression can be used to define non-linear classifiers in the original space https://tonyfischetti.shinyapps.io/InteractiveLogisticRegression. The tutorial
http://nbviewer.ipython.org/gist/vietjtnguyen/6655020 shows Python code implemeting logistic regression and visualizing the results and the training process.