Review of PTYS 595B: Machine Learning for Planetary Sciences

Alexander Prescott University of Arizona Fall 2016

Summary

This paper serves as a review of the topics covered in the first half of the Fall 2016 semester of PTYS 595B, Machine Learning for Planetary Sciences, in the Department of Planetary Sciences at the University of Arizona. Topics covered include learning types, basic matrix algebra, linear regression, logistic regression, support vector machine learning, neural networks, autoencoders, convoluted neural networks, and validation methods. Further documentation for each of these topics can be found at the course webpage (https://leonpalafox.github.io/MLClass) or at the sources referenced at the end of this paper.

Types of Learning

There are two general types of learning in machine learning methods: supervised and unsupervised. Supervised learning methods use predetermined labels to classify data, while unsupervised learning has no predetermined labels for classification, instead creating classifications as part of the learning process.

Linear Regression

A set of predictors is used to predict an outcome using a linear combination of the predictors. Each predictor has a multiplicative weight applied that conveys the relative importance of each predictor. Weights are iteratively chosen to minimize the error between predicted outcomes and the observed outcomes in a training set of data. There are two methods employed in selecting the weights that result in the smallest error: matrix algebra and gradient descent.

Matrix Algebra

A set of *n* outcomes $\boldsymbol{y} = \{y_1, \dots, y_n\}$ that are determined by a corresponding set of *n* * *m* predictors $\boldsymbol{X} = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\}$ weighted individually by weights $\boldsymbol{\beta} = \{\beta_1, \dots, \beta_n\}$, where each element of $\boldsymbol{X}$ is a vector of length *m*, can be written as the following system of equations:

(1)
$$\begin{aligned} y_1 &= \beta_1 x_{1,1} + \cdots + \beta_m x_{1,m} \\ &\vdots \\ y_i &= \beta_1 x_{i,1} + \cdots + \beta_m x_{i,m} \\ &\vdots \\ y_n &= \beta_1 x_{n,1} + \cdots + \beta_m x_{n,m} \end{aligned}$$

Using matrix notation, this system of equations can be written as:

(2)
$$\boldsymbol{y} = \mathbf{X\beta}$$

The measurement of errors is implementer through a least-squares cost function:

(3)
$$J(\boldsymbol{\beta}) = \frac{1}{2}(\boldsymbol{X\beta} - \boldsymbol{y})^2$$

To minimize the errors, the minimum of the cost function is desired. This corresponds to the derivative of the cost function being equal to zero. Without showing the derivation, the ideal set of weights is found by:

(4)
$$\mathbf{B} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

The inversion of the predictor matrix is what renders matrix algebra useless for very large datasets, as the complexity of the inversion procedure increases exponentially with increasing matrix size. For large datasets, gradient descent is the way to go.

Gradient Descent

In gradient descent, an initial guess is chosen for the weights. Then, the weights are iteratively updated based on the derivative of the cost function (equation 3). Formulaically, this is:

(5) $$\boldsymbol{\beta}_{j+1} = \boldsymbol{\beta}_j - \alpha \frac{\partial}{\partial \boldsymbol{\beta}_j} J(\boldsymbol{\beta})$$

(6) $$\boldsymbol{\beta}_{j+1} = \boldsymbol{\beta}_j - \alpha(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) * \boldsymbol{X}^T$$

Here, α is considered the learning rate or step size. This procedure is continued either for a preset number of iterations or until the change in the cost function between consecutive iterations drops below a preset threshold.

Regularization

A common problem with regressions of any sort to be aware of is overfitting of the data. To avoid this, a penalty term is commonly added to the cost function:

(7) $$J(\boldsymbol{\beta}) = \frac{1}{2}(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^2 + \lambda \|\boldsymbol{\beta}\|$$

Here, λ is termed the regularization parameter or the weight decay parameter.

Logistic Regression

Logistic regression uses many of the ideas outlined in linear regression, but they are applied in service of classifying input data into two or more classes. In the case of two classes, this is done by mapping the linear equations into a bounded range (e.g. [0, 1]), selecting a threshold value, and classifying based on whether the mapped value falls above or below the threshold value. The function used in this process is the sigmoid function, which maps real numbers into (0, 1):

(8) $$f(z) = sig(z) = \frac{1}{1+e^{-z}}$$

A pleasant aspect of this function is that the derivative is a simple function of the original function:

(9) $$f'(z) = f(z) * \left(1 - f(z)\right)$$

The input variable to the functions is $z = \boldsymbol{\beta}^T \boldsymbol{X}$.

Optimization

An advantage of the sigmoid function is that it's outputs can be interpreted as probabilities for whether an input should be classified as a 1 or a 0. In equation form:

(12) $$p(y_i = 1 \,|\boldsymbol{x}_i, \boldsymbol{\beta}) = f(\boldsymbol{\beta}^T \boldsymbol{x}_i)$$

(13) $$p(y_i = 0 \,|\boldsymbol{x}_i, \boldsymbol{\beta}) = 1 - f(\boldsymbol{\beta}^T \boldsymbol{x}_i)$$

(14)

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}) = f(\boldsymbol{\beta}^T \boldsymbol{x}_i)^{1-y_i}\left(1 - f(\boldsymbol{\beta}^T \boldsymbol{x}_i)\right)^{y_i}$$

The goal in optimization is to maximize equation 14 across all inputs. That is, maximize the maximum likelihood estimator:

(15) $$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} (p(y_i|\boldsymbol{x}_i, \boldsymbol{\beta})$$

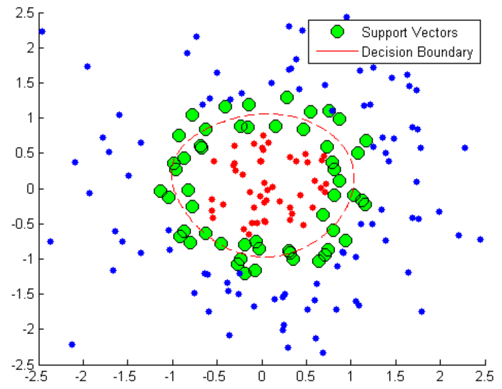
Maximizing the log of equation 15 is logically equivalent and very common. In addition, a penalty term can be added to avoid overfitting, as was done in linear regression. Incorporating both of these, the goal is to maximize:

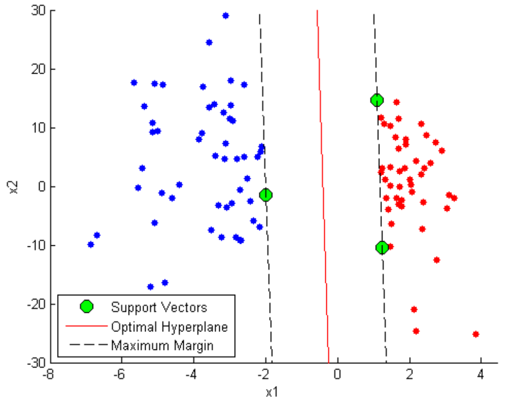
(16) $$\sum_{i=1}^{n} \left[\log\left(p(y_i|\boldsymbol{x}_i, \boldsymbol{\beta})\right)\right] - \lambda \|\boldsymbol{\beta}\|^2$$

Equation 16 does not have a closed-form solution and therefore must be solved numerically. This same cost function (or a very similar version of it) is used in the following machine learning methods as well.

Support Vector Machines

Support vector machines (SVM's) are by far the most commonly implemented machine learning tool today. They are relatively fast, have a low learning curve, and work well even for very large datasets. As a form of supervised learning, they can be used to classify data into 2 or more classes. SVM's identify the optimal hyperplane between adjacent classes, which serves to separate the classes as much as possible. If the data is completely separable, the hyperplane is chosen to maintain the maximum distance between classes (i.e. the maximal margin). Different kernels can be used to quantify the distance between classes – examples include the linear, polynomial, and Gaussian (RBF) kernels. Each kernel has its optimal application, and selection of a kernel is dependent on the distribution of the data in space. For example, a RBF kernel works well for classes distributed radially about an origin, while the linear kernel works well for data with a distinct linear split.





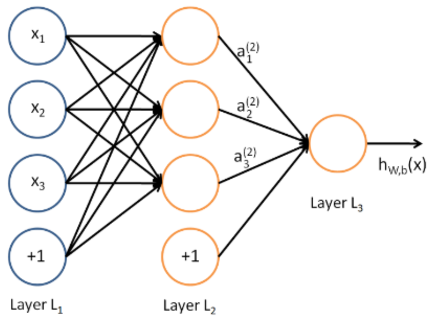
SVM's trained using an RBF kernel (top) and a linear kernel (bottom). Figures made using MATLAB.

Neural Networks

Neural networks (NN's) are so named because they are composed of individual units called perceptrons that mimic neurons. Each perceptron takes in all of the predictors, applies a unique set of weights, and passes their linear combination to an activation function. The output of the activation function can then be passed as an input to another unit. Perceptrons can be arranged in layers so that the outputs of the perceptrons in one layer can be passed as inputs to the next layer.

NN's are calibrated through a forward-back-propagation process, wherein:

i. all weights are initialized as small, random, different values
ii. the NN is run forward to its end
iii. gradient descent is used backwards to update all of the weights
iv. the model is run forward again.
v. this process repeats for a set number of iterations or until a minimum of the cost function is found.



Single-layer neural network with output $h_{w,b}(x)$. Figure from Palafox class slides.

Common Activation Functions

One of the most popular activation functions used in this process is the sigmoid function, which was visited in the logistic regression section. Another popular function is the hyperbolic tangent, which maps into (-1, 1):

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{9}$$

$$g'(z) = 1 - g(z)^2 \tag{11}$$

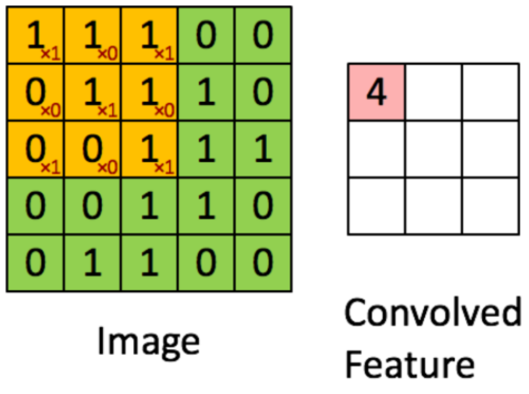
Autoencoders

The use of autoencoders in NN's is a relatively recent technique which has revitalized the usefulness of NN's by increasing optimization speed. An autoencoder is a single NN that aims to duplicate the identity function through a NN architecture. That is, it aims to duplicate:

$$f(x) = x, \forall x \text{ in the input dataset}$$
One key piece of the autoencoder is that it has fewer perceptron units than there are inputs, therefore it acts as a sort of compression of the information contained in the input dataset. Autoencoders can be layered one after another to repeat this process – these are termed stacked autoencoders.

Convoluted Neural Networks

A convoluted neural network (CNN) is only slightly different from a NN and is used primarily for the classification of images. A convolution of the input dataset is performed using weights as the convolution window values. These weights are optimized through the forward-back-propogation process.



*An example of convolution. In a CNN, the "*1" and "*0" would be replaced with "w$_1$, ..., w$_9$". These weights are updated with every iteration. Figure from Palafox class slides.*

Validation Techniques

Cross Validation

Cross validation (CV) is a method wherein the classifier is run with a certain fraction of the input dataset held out of training for testing. This can be useful for determining the optimal parameter set for the classifier model being used. To use this method, a large dataset is desirable and probably required

K-Fold Cross Validation

K-fold CV is a method wherein the entire dataset is split into *k* disjoint subsets and the classifier is run *k* times. For each run, the classifier uses *k-1* of the subsets for training and the other subset is used for testing. In this way, each subset serves as the testing dataset in one of the model runs. The overall error of the classifier is the average of the *k* errors from each run.

The optimal value of *k* can be tricky to determine, but there are a few general guidelines. *k*=3 does a decent job, and for simple models like SVM's or logistic regression, *k=10* is good. *k*=*n-1*, where *n* is the number of data points, should never be used. In general, we want enough data points so that each fold subset is well populated.

Validation Metrics

First, some abbreviations:

- true positives, TR
- false positives, FP
- true negative, TN
- false negatives, FN

Now for validation metrics:

Accuracy is a measure of the statistical bias, given by: $acc = \frac{tp+tn}{tp+tn+fp+fn}$.

Precision measures the fraction of true positives out of all positive classifications: $prec = \frac{tp}{tp+fp}$.

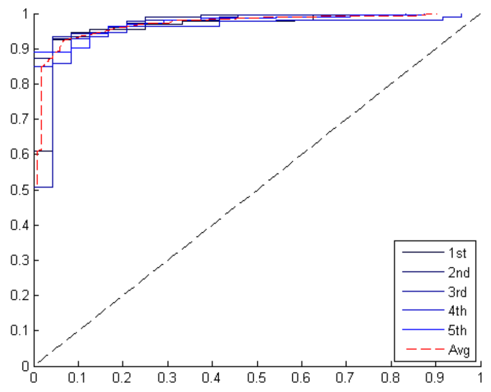
Recall measures the fraction of true positive classifications out of all of the actual positive classifications: $recall = \frac{tp}{tp+fn}$.

F1-Score gives a measure of classifier skill in a single number: $F1 = 2 * \frac{prec*recall}{prec+recall}$

Receiver Operating Characteristic Plots

A ROC plot shows the true positive rate (TPR) vs. the false positive rate (FPR) for a classifier, where the classification threshold is varied over multiple runs. If random classifications are chosen, one would expect a linear 1:1 line in the ROC plot. If a classifier is perfect, the TPR would be 1 across all threshold values and would exhibit a step function shape in the ROC plot.

In evaluating a classifier using a ROC plot, the area under the curve (AUC) is often used to quantify the performance of the classifier. In the random example above, the AUC would equal 0.5, whereas a perfect classifier has an AUC of 1. In general, the closer the AUC is to 1, the better the classifier.



An example of an ROC plot for multiple good classifiers, with FPR on the x-axis and TPR on the y-axis, with an average AUC = 0.967. Figure made using MATLAB.

Sources

Palafox, Leon, 2016, PTYS 595b class slides, Department of Planetary Sciences, University of Arizona, Tucson, https://leonpalafox.github.io/MLClass/

Ng, Andrew, 2011, Lecture Notes on Sparse Autoencoders, Stanford University, https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf

Zhu, Xiaojin, 2010, Logistic Regression, University of Wisconsin, http://pages.cs.wisc.edu/~jerryzhu/cs769/lr.pdf