Umut Akös
21202015
CS464 – HMW2

QUESTION 2)

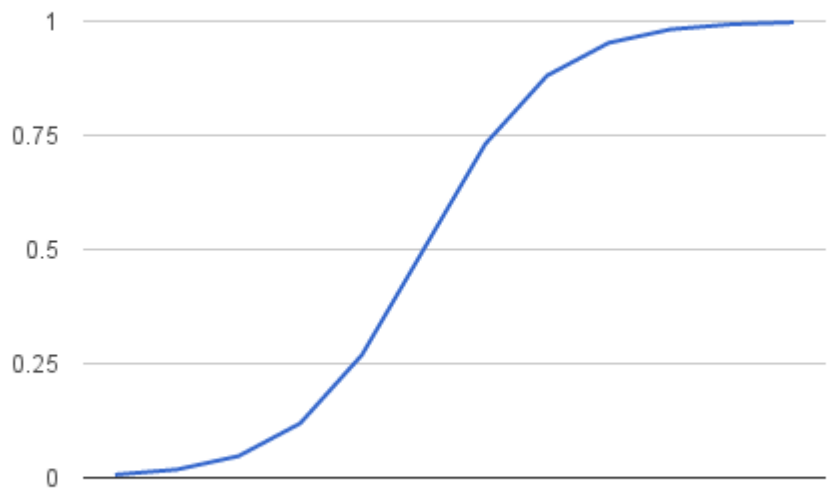
Logistic Regressor:

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$1 / (1 + e^{-value})$$

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function



Logistic Regression by Stochastic Gradient Descent

We can estimate the values of the coefficients using stochastic gradient descent.This is a simple procedure that can be used by many algorithms in machine learning. It works by using the model to calculate a prediction for each instance in the training set and calculating the error for each

prediction. We can apply stochastic gradient descent to the problem of finding the coefficients for the logistic regression model as follows:

Given each training instance:

1. Calculate a prediction using the current values of the coefficients.
2. Calculate new coefficient values based on the error in the prediction.

The process is repeated until the model is accurate enough (e.g. error drops to some desirable level) or for a fixed number iterations. You continue to update the model for training instances and correcting errors until the model is accurate enough orc cannot be made any more accurate. It is often a good idea to randomize the order of the training instances shown to the model to mix up the corrections made.

By updating the model for each training pattern we call this online learning. It is also possible to collect up all of the changes to the model over all training instances and make one large update. This variation is called batch learning.

Calculate Prediction

Let's start off by assigning 0.0 to each coefficient and calculating the probability of the first training instance that belongs to class 0.

B0 = 0.0

B1 = 0.0

B2 = 0.0

The first training instance is: x1=2.7810836, x2=2.550537003, Y=0

Using the above equation we can plug in all of these numbers and calculate a prediction:

prediction = 1 / (1 + e^(-(b0 + b1*x1 + b2*x2)))

prediction = 1 / (1 + e^(-(0.0 + 0.0*2.7810836 + 0.0*2.550537003)))

prediction = 0.5

Calculate New Coefficients

We can calculate the new coefficient values using a simple update equation.

b = b + alpha * (y – prediction) * prediction * (1 – prediction) * x

Where b is the coefficient we are updating and prediction is the output of making a prediction using the model.

Alpha is parameter that you must specify at the beginning of the training run. This is the learning rate and controls how much the coefficients (and therefore the model) changes or learns each time it is updated. Larger learning rates are used in online learning (when we update the model for each training instance). Good values might be in the range 0.1 to 0.3. Let's use a value of 0.3.

You will notice that the last term in the equation is x, this is the input value for the coefficient. You will notice that the B0 does not have an input. This coefficient is often called the bias or the intercept and we can assume it always has an input value of 1.0. This assumption can help when implementing the algorithm using vectors or arrays.

Let's update the coefficients using the prediction (0.5) and coefficient values (0.0) from the previous section.

$$b0 = b0 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 1.0$$

$$b1 = b1 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 2.7810836$$

$$b2 = b2 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 2.550537003$$

or

$$b0 = -0.0375$$

$$b1 = -0.104290635$$

$$b2 = -0.09564513761$$

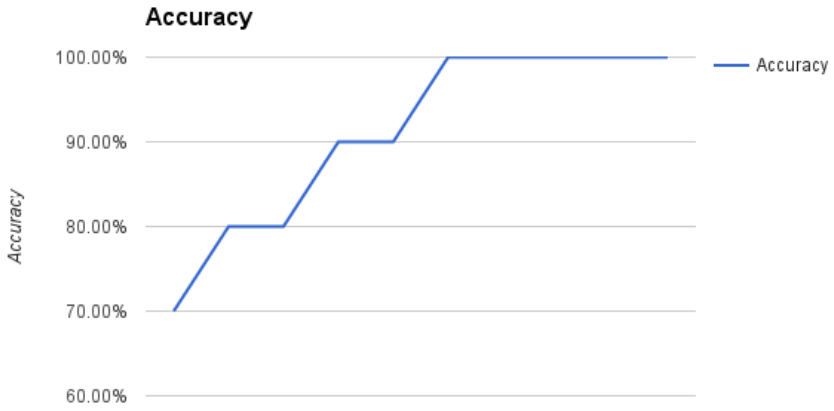
Repeat the Process

We can repeat this process and update the model for each training instance in the dataset.

A single iteration through the training dataset is called an epoch. It is common to repeat the stochastic gradient descent procedure for a fixed number of epochs.

At the end of epoch you can calculate error values for the model. Because this is a classification problem, it would be nice to get an idea of how accurate the model is at each iteration.

The graph below show a plot of accuracy of the model over 10 epochs.



Q2 .1 : In given problem we have repeat the process for iteration count (500,1000,10000,100000). We have found the best value of learning rate is at 0.02 and found the accuracy close to 100%.

Q2.2 : Worst subset we have found is :

- Example from row 43 to row 86 having very leass accuracy
- rest if example having better accuracy
- During Forward elimination and Backward elimination we have found the same feature subset.

Q2.3 : We have also calculated the Precision and Recall for each fold:

Results are as follows :

Fold	Precision	Recall
1st	0.5	0.5
2nd	0.222	1.00
3rd	0.5	0.5
4th	0.5	0.5
5th	0.5	0.5

QUESTION 3)

Support vector machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

Maximal-Margin Classifier

The Maximal-Margin Classifier is a hypothetical classifier that best explains how SVM works in practice.

The numeric input variables (x) in your data (the columns) form an n-dimensional space. For example, if you had two input variables, this would form a two-dimensional space.

A hyperplane is a line that splits the input variable space. In SVM, a hyperplane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. In two-dimensions you can visualize this as a line and let's assume that all of our input points can be completely separated by this line. For example:

$$B0 + (B1 * X1) + (B2 * X2) = 0$$

Where the coefficients (B1 and B2) that determine the slope of the line and the intercept (B0) are found by the learning algorithm, and X1 and X2 are the two input variables.

You can make classifications using this line. By plugging in input values into the line equation, you can calculate whether a new point is above or below the line.

- Above the line, the equation returns a value greater than 0 and the point belongs to the first class (class 0).
- Below the line, the equation returns a value less than 0 and the point belongs to the second class (class 1).
- A value close to the line returns a value close to zero and the point may be difficult to classify.
- If the magnitude of the value is large, the model may have more confidence in the prediction.

The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that as the largest margin. This is called the Maximal-Margin hyperplane.

The margin is calculated as the perpendicular distance from the line to only the closest points. Only these points are relevant in defining the line and in the construction of the classifier. These points are called the support vectors. They support or define the hyperplane.

The hyperplane is learned from training data using an optimization procedure that maximizes the margin.

Soft Margin Classifier

In practice, real data is messy and cannot be separated perfectly with a hyperplane.

The constraint of maximizing the margin of the line that separates the classes must be relaxed. This is often called the soft margin classifier. This change allows some points in the training data to violate the separating line.

An additional set of coefficients are introduced that give the margin wiggle room in each dimension. These coefficients are sometimes called slack variables. This increases the complexity of the model as there are more parameters for the model to fit to the data to provide this complexity.

A tuning parameter is introduced called simply C that defines the magnitude of the wiggle allowed across all dimensions. The C parameters defines the amount of violation of the margin allowed. A C=0 is no violation and we are back to the inflexible Maximal-Margin Classifier described above. The larger the value of C the more violations of the hyperplane are permitted.

During the learning of the hyperplane from data, all training instances that lie within the distance of the margin will affect the placement of the hyperplane and are referred to as support vectors. And as C affects the number of instances that are allowed to fall within the margin, C influences the number of support vectors used by the model.

- The smaller the value of C, the more sensitive the algorithm is to the training data (higher variance and lower bias).
- The larger the value of C, the less sensitive the algorithm is to the training data (lower variance and higher bias).

Support Vector Machines (Kernels)

The SVM algorithm is implemented in practice using a kernel.

The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra, which is out of the scope of this introduction to SVM.

A powerful insight is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. The inner product between two vectors is the sum of the multiplication of each pair of input values.

For example, the inner product of the vectors [2, 3] and [5, 6] is 2*5 + 3*6 or 28.

The equation for making a prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = B0 + sum(ai * (x,xi))$$

This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

Linear Kernel SVM

The dot-product is called the kernel and can be re-written as:

$$K(x, xi) = sum(x * xi)$$

The kernel defines the similarity or a distance measure between new data and the support vectors. The dot product is the similarity measure used for linear SVM or a linear kernel because the distance is a linear combination of the inputs.

Other kernels can be used that transform the input space into higher dimensions such as a Polynomial Kernel and a Radial Kernel. This is called the Kernel Trick.

It is desirable to use more complex kernels as it allows lines to separate the classes that are curved or even more complex. This in turn can lead to more accurate classifiers.

Q3.2 : As value of C tends to infinity then model get overfit and showing same accuracy.

The smaller the value of C, the more sensitive the algorithm is to the training data (higher variance and lower bias).

The larger the value of C, the less sensitive the algorithm is to the training data (lower variance and higher bias).

Q3.3 : For the given problem optimal value of C is 0.9 and Model has best accuracy is 66.66%.

As the value of C is increases then the training accuracy will be going to increase but testing accuracy drop down this phenomeno is called as overfitting. So choosen value of C should be optimal.

Q3.4 : In this we have impliment the Radial Kernel give as:

Radial Kernel SVM

K(x,xi) = exp(-gamma * sum((x – xi^2))

Where gamma is a parameter that must be specified to the learning algorithm. A good default value for gamma is 0.1, where gamma is often 0 < gamma < 1. The radial kernel is very local and can create complex regions within the feature space, like closed polygons in two-dimensional space. In this we have found out best value of gamma is at 0.8.