Tanay Gavankar
11-441 HW3

Logistic Regression and SVM

Gradient likelihood derivation

$$z = w^T x = w_0 + \sum_{j=1}^{m} w_j x_j$$

$$l(w) = y\, ln(\sigma(z)) + (1-y) ln(1-\sigma(z)) - \frac{1}{2} C||W||^2$$

$$\frac{\partial}{\partial w_j} l(w) = \frac{dl(w)}{d\sigma} \frac{d\sigma}{dz} \frac{\partial z}{\partial w_j}$$

$$= (y \frac{1}{\sigma(z)} - (1-y) \frac{1}{1-\sigma(z)}) \sigma(z)(1-\sigma(z)) \frac{\partial}{\partial w_j} z - \frac{\partial}{\partial w_j} \frac{1}{2} C(\sum_i w_i^2)$$

$$= (y(1-\sigma(z)) - (1-y)\sigma(z)) x_j - C w_j$$

$$= (y - \sigma(z)) x_j - C w_j$$

Gradient descent outline

The gradient derived above is the update rule for the weight vector. Initialize a weight vector with all 0's and let each X^i represent the ith training data, and let the first element of the vector be 1, and the rest are the actual data point's vector elements. (X[0] is set to 1 so that W[0] is included in the calculation.) Then, we perform stochastic gradient ascent. To do this, we loop until convergence. Convergence is defined as the Euclidean distance between the current weight vector and the previous weight vector is less than a specified threshold (0.0001 in this implementation). Within each iteration of the convergence loop, we iterate over all the training data points. For each training data point, we perform the update rule on the weight vector by, for all weight vector elements, calculating the value of the derivative (derived above) for each vector element, multiplying it by a learning rate (0.01 in this implementation), and adding it to the current weight vector's element.

LR vs SVM

The logistic regression classifier performs better in accuracy than the support vector machine on testing data. Furthermore, as C gets smaller, logistic regression becomes slightly more accurate (up until the optimal C), but the training time significantly increases (by the factor of how much C changes - i.e. C/10 -> 10x training time). At extremely small values of C (i.e. 0.00001) and a reasonable learning rate (0.01), the training time was in the range of hours, and so it was prohibitively small. The learning rate can be increased to balance C, resulting in reasonable training times, and so for the small C values, the learning rate was increased to 0.1. Classification time for LR is unrelated to C, and it is quite fast. However, for SVM, the training time is negligible (0.00 CPU seconds) for all values of C, as is the classification time. Because of this, it is difficult to determine if there is a correlation between training time and C for SVM.

The tradeoff for the better runtime is a lower accuracy. For small values of C (i.e. C < 0.001), the experimental data shows that SVM has a lower accuracy on test data than LR given the same training and test data sets. Both LR and SVM seem to have a peak accuracy with a C value of around 0.000075. For SVM, as the C gets larger, the F1 value seems to be higher, which is strange. The experimental data table can be found at the end of the report, as is the plot of C vs F1 score for the various types of classifiers.

Design Decisions

In this implementation of logistic regression, the LogisticRegressionRunner is the class that holds the main method. In this class, a DataParser object is first constructed and used to parse the config file data.txt. This file includes the path to the training and test data, as well as the value of C to use for the regularization factor. The training data is parsed into a list of AbstractEntry's, where each AbstractEntry has fields for things like abstract ID, actual label, predicted label, and a SparseTermList. The SparseTermList is just a wrapper implementation of the input vector for the logistic regression, as so to allow for easy changes of the implementation for the actual vector. This decision was extremely useful because, initially, I had thought that the best way to implement the vector was to use a hashmap as the backing as the vector would be sparse, there was no need to keep a contiguous array that had many zeros. However, it turned out that this was extremely slow for training, as the get() and put() methods were slow (even though they were constant time). As an improvement, the hashmap was replaced with a primitive double array. Accessing and modifying values in this array became much, much faster, and so that is the chosen implementation. However, this comes at a tradeoff of using more memory, and so the max heap size has to be increased to 512MB. Each data point would need one SparseTermList object, and so with n data points, there would have to be n+1 primitive double arrays of length k, where k is the number of terms in the bag of words representation.

Since this implementation for multiclass logistic regression was to train many binary classifiers, there were 17 classifiers constructed (one for each class), and then trained on the given training data with the given class as positive examples and all other classes as negative examples. Within each LogisticRegression class (i.e. a classifier), the train method iterates until convergence and performs a stochastic gradient descent to find the weight vector. For classification, the sigmoid function is calculated and returned using the trained weight vector for that classifier. The LogisticRegression class implements Serializable so that (since training is often times slow), the classifier can be trained once and then tested on various data sets without needing to retrain across various runs of the code. When classifying a data point to get what label it is, LogisticRegressionRunner gets the sigmoid value for each classifier and chooses the label for which the sigmoid value is the highest.

The SVM experiment was performed as instructed (as a set of binary classifiers) via an external Python script to automate the process.