# Artificial Intelligence: Homework 4

### ClassifyThis

The purpose of this assignment was to understand the dataset and, given this information, classify a new data set to predict the income of each new entry. We decided to use a logistic function to calculate the output of this new data set. The way that this classifier works is that it uses gradient descent to calculate theta values, and ultimately use these values to conclude whether or not an individual makes more than $50K or less than $50K.

Deep diving into the technical details of this algorithm, first we instantiated this classifier to understand the two output options that are available. In addition, a hashmap was used to create a relationship of the names and their corresponding valid values. Next, we created a vector array of thetas. The length of this vector array is defined by the number of features there are and the number of values that each feature can have. Simply put, the numeric features would relate to one theta value, and the non-numeric discrete features would relate to however many valid values there are for that particular feature. Basically, we cannot just take qualitative data and assign one theta value to each qualitative feature. Thus, each value per value set would have its own theta value, and the value would be '1' if that particular data entry feature had that value, and '0' if not.

While going through the training data set, we adjust the theta values using gradient descent and finely adjusting each theta value per new piece of information. This serves as our cost function that helps the machine learning program understand data and be able to better predict an output for a test data set. The cost function is minimized over time the more information passed in and the more iterations there are, adjusting the thetas to represent a larger amount of past information. Given this training list parsed into theta values, we are able to use a logit function that utilizes a sigmoid function to create weights for evaluating test data.

Thus, the main two parameters we can adjust by hand are the rate at which the thetas can correct over time with gradient descent, and the number of iterations that allow the program to comprehend the training data. After the program runs through the training set and calculates the theta vector, we pass in the test data that the program is supposed to calculate resulting outputs for. The logistic function that we use, enabled by gradient descent, allows us to pass in each new data entry and classify it into either output option (i.e. >50K or <=50K). The logistic function allows us to calculate a value between 0 and 1 given the sigmoid transformation and the previosuly determined thetas, and we take any value greater than or equal to 0.5 as >50K and less than 0.5 as >=50K. This gives us the proper output to classify each new data entry.

The reason we used gradient descent and a logistic function to determine each new data entry's output is due to the fact that we tried both this and the linear regression using normal equation algorithm. Given the comparison of both, gradient descent runs through more iterations and

provides a more fine-tuned approach to gather information for the thetas in our program. The higher the number of iterations, and the adjustable rate aspect of how much the program corrects itself allows the program to be able to understand the training data set slightly better.

## Performance Against a Simple Classifier

Our ClassifyThis classifier handily defeats our SimpleRandomClassifier class. In a run of 1499 data points, the random classifier successfully predicts approximately 50%. The following is data over 5 runs for the random classfier:

1) Correct: 763; Wrong: 737; % Correct: 0.5086666666666667

2) Correct: 753; Wrong: 747; % Correct: 0.502

3) Correct: 740; Wrong: 760; % Correct: 0.49333333333333335

4) Correct: 770; Wrong: 730; % Correct: 0.5133333333333333

5) Correct: 752; Wrong: 748; % Correct: 0.5013333333333333

Our ClassifyThis classifier over 5 runs, however, always shows the following results: 1) Correct: 1222; Wrong: 277; % Correct: 0.81521014

As you can see our classifier's performance is much greater than that of the random classifier

## Versus the BAClassifier

The BAClassifier is a linear regression classifier that uses the normal equation to find theta values that minimize the cost function. The classifier performed well and successfully predicts just over 80% of the training data set when you run the training data (minus the known outcomes) through the classifier's makePredictions method. This classifer depends on the Java Matrix Package JAMA (http://math.nist.gov/javanumerics/jama/) to work and the accompanying .jar file will be included in the submission. In general the classifier works as follows:

1) The classifiers contructor reads in the .names file and preprocesses it to create a) a list of possible outcomes (i.e. >50k and <=50k) b) a list of all value names (i.e. age, workclass, education, etc.) and c) a mapping from value names to potential value types (i.e. { age -> [], workclass -> [Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked]}). This mapping informs the rest of the program as to which type of data we are dealing with. An empty list in this mapping refers to a numeric data type and a list with values means that this is a discrete value type.

2) The classifiers train method reads in the .train file and preprocesses it and adds the value 1 at the beginning of each row. The train method then iterates through this preprocessed data with the goal of turning the training data and outcomes into two matrices X and Y. For each row it does the following:

- First it breaks the splits the row on all whitespace characters creating an array of

individual elements.
- It then begins to iterate on this array of row elements
- If we are looking at the first element (which is the 1 that we added to the row in the preprocessing) then we simply parse the 1 as a double and add it as the first element in the row of the matrix.
- If we are not looking at the first element then we look up the attribute name from the list we made in the contructor and then look up the type of that attribute in the mapping we made in the contructor. If we find that the mapping holds an empty list for that attribute name then we simply parse that value as a double and put it as the next value in the row of the data matrix. If we find that the mapping points to a list of length > 0 then we know that this attribute type is discrete valued and thus we use the index of the element in that list as the value that we place in the data matrix. We do not add the last element in the row to the X (data matrix) because the output is last and belongs in the Y(output matrix)
- For each row we add the last element in the row to the Y (output matrix)
- After we have calculated both the X and Y matrices, the classifier uses the normal equation to calculate the theta values that minimize the cost function.

3) Finally the classifiers checkPredictions method reads in the test data and applies our theta values to test data to come up with a prediction that ranges from just below 0 to just above 1. If our prediction is >= .5 we say that we should predict the second value in the output options and if it falls < .5 we predict the first value in the output options.

Overall this classifier performs well and when we run the training data (minus the outcome) through the makePredictions method we are correctly predicting 1,211 out of 1,499 or 80.78% of the data points. This performance was slightly worse than our ClassifyThis classifier which predicts over 81.5% correctly. Although our ClassifyThis classifer had slightly better prediction performance, the BAClassifier has better run time performance. We are confident in both of these classifiers; however, it seems that the logistic regression classifier fit the dataset slightly better than the linear regression.

---

## Conclusion

This project was an worthwhile and interesting introduction to machine learning in practice. We found that both the logistic and linear regressions worked well for the given the data sets and that a random classifier does not do well enough. The normal equation for the linear regression classifier simplified implementation and gave it improved runtime performance over the logistic regressions gradient descent approach; however, the logistic regression provided slightly better prediction performance.