CS 434 - Spring 2016 - Final Project Report

Garrett Smith, Cody Malick

smithgar@oregonstate.edu — malickc@oregonstate.edu

Abstract

In this report, we examine four different algorithms for machine learning: linear regression, logistic regression, support vector machine, and decision tree. We apply these four algorithms to the same problem, spam classification, and examine how they perform, how they perform in different conditions, and how they compare to each other.

Contents

I. Introduction

Classification is a common problem in machine learning. There are many approaches, some simple, some more complex, that all involve sorting a given set of items into two or more classifications. The goal, of course, being able to quickly and efficiently classify large sets of items into their correct classes. In our final project, we have examined four seperate classification algorithms, and applied them all to the same problem of classifying spam. The algorithms we cover are linear regression, logistic regression, support vector machine, and decision tree. Before we jump into examining the algorithms, let us examine the data we want to use.

II. The Data

Our data set is a pre-processed set of spam from University of California Irvine, the `Spambase Data Set`.[1] The data set is comprised of the following:

Number of Instances: 4601

Number of Attributes: 58

- 48 continous real [0,100] attributes of type word_freq_WORD percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A “word” in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
- 6 continuous real [0,100] attributes of type char_freq_CHAR] = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
- 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
- 1 nominal 0,1 class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.

Class Distribution

- Spam: 1813 (39.4%)
- Non-Spam: 2788 (60.6%)

This data was great for a comparison set because we could look at the exact distribution of data, what kind of data it was, what the different attributes meant, etc. Because we did not have a separate training set, we built our training set out of about 750 entries from the original training set of 4000. The training set contained both spam and non-spam classified entries.

Now that we have the layout of the data, next we will examine our first algorithm, linear regression.

III. Linear Regression

Linear regression is a very simple algorithm, designed to handle data in very simple dimensions. We ran into a lot of problems trying to implement this algorithm over a data set with over fifty features. After some trial and error, we decided that applying this algorithm to a feature set this complex did not make a lot of sense. In order to do so, we would have to transform the data into something more simple, therefore changing the fundamental problem. We do not have one variable corresponding to a dependent variable.

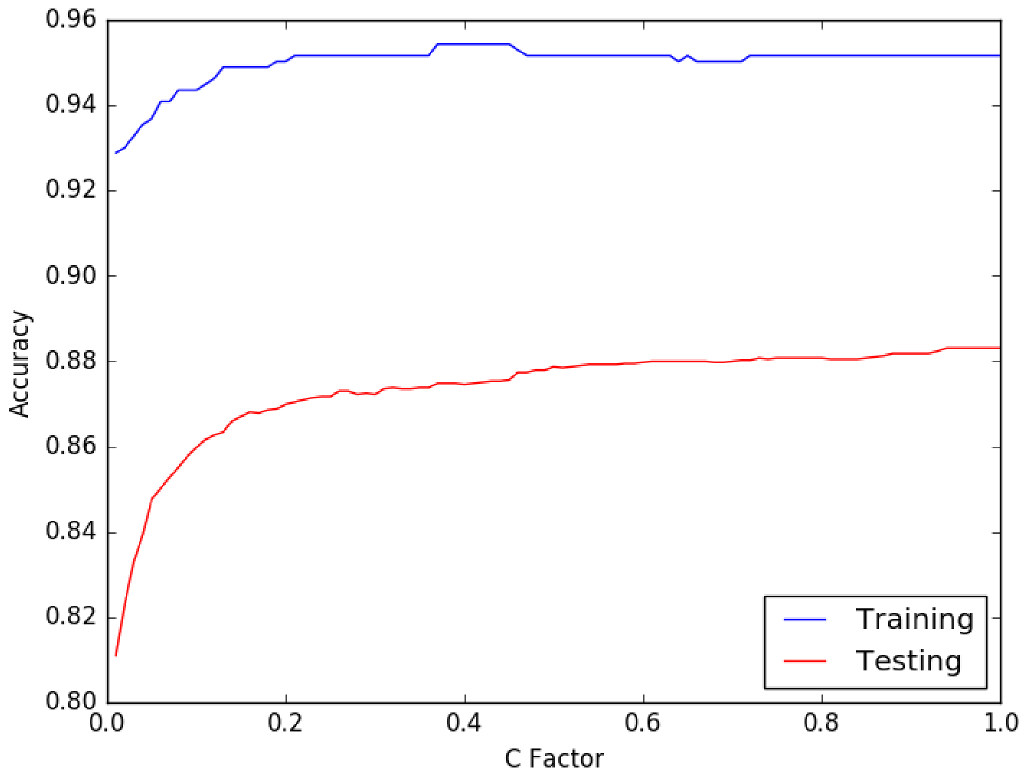
After doing a quick implementation, we were hitting a few percentage points in accuracy, and we believe that it was purely luck that we got that many. Instead of thoroughly investigating linear regression, we added support vector machine to our lineup of algorithms to examine. Before jumping into a support vector machine, we will now examine the next closest thing to linear regression, logistic regression.

IV. Logistic Regression

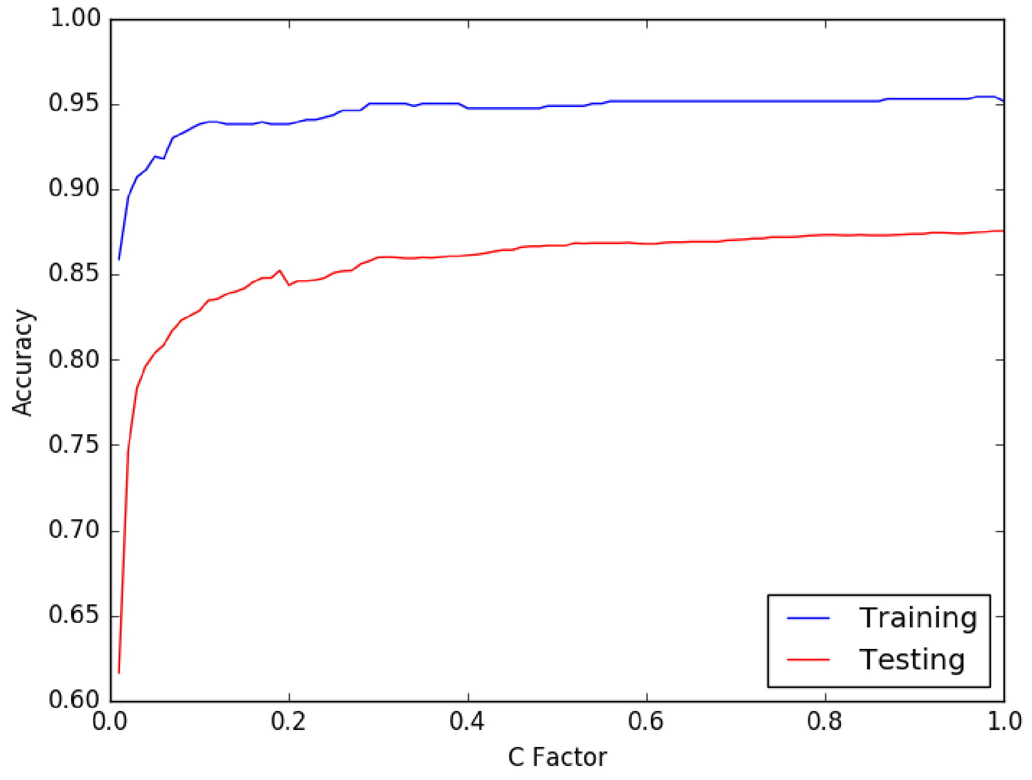
Logistic regression is a form of classification that uses probability. We decided to use logistic regression because it is a form of linear regression that is able to handle large amounts of features with relative accuracy, and often yields good results in a binary classification problem.

A. *Implementation*

We implemented logistic regression by using the `scikit-learn` library in python. `Scikit-learn` is a great library that has several implementations available and an extremely simple setup process. We used the `linear_model.LogisticRegression` module. As far as execution, we first set it up with the appropriate features, and got surprisingly good results right out of the box. With default settings (no regularization, default penalty of l2), we achieved 95% accuracy on the training set, and 88% accuracy on the test set. We experimented with different regularization values from .1, high regularization, to 1.0, no regularization.



Surprisingly, we only saw an increase in accuracy as regularization decreased. From this behavior we can extrapolate that our data has few to no outliers skewing the data in an unhelpful way. Next we tried applying l1 penalty instead of l2:



With l1 penalty we see it takes quite a bit longer for the accuracy to peak, and overall there is a lower overall accuracy of about 3%. This is expected as l1 penalty is designed for data sets with sparse data, while l2 is better for larger, more complete data sets.

V. Support Vector Machine

VI. Decision Tree

The decision tree algorithm deals well with both discrete and continuous features. Aside from the class label all of the features in our data set were continuous. This made decision tree a good choice for spam classification. Without any limits the decision tree will train a tree that perfectly classifies all of the training examples. While this tree perfectly classifies the training set, it leads to over fitting with other data and a loss of accuracy. To combat this we tried limiting two different parts of the decision tree, tree depth and the number of leaf nodes on the tree.

A. *Implementation*

We used the `DecisionTreeClassifier` module of the `scikit-learn` library in python to build our decision trees. The library let us limit the tree depth and number of leaf nodes so we could tune the decision tree to try to get the best accuracy. Before we did any tuning we tested the accuracy of the decision tree when built from the training data when no limits were placed on how big the tree could be. The accuracy on the testing data using a tree that was perfectly fitted to the training data was 85.63%. By limiting the depth of the tree we were able to get some improvement and the best accuracy we obtained on the testing data was 87.87% with a tree where the maximum depth was 135. Next we tried limiting the number of leaf nodes on the tree. A tree with only 24 leaf nodes gave us our best accuracy of 88.09%.

B. Limiting Tree Depth

By limiting the depth of the tree to between 2 and 1000 we limited the number of decisions that it could have, and attempted to reduce over fitting the training data that way. The best accuracy on testing data we were able to obtain by only limiting depth was 87.87% accuracy with a tree that was of depth 135. Below is a plot of accuracy versus tree depth.

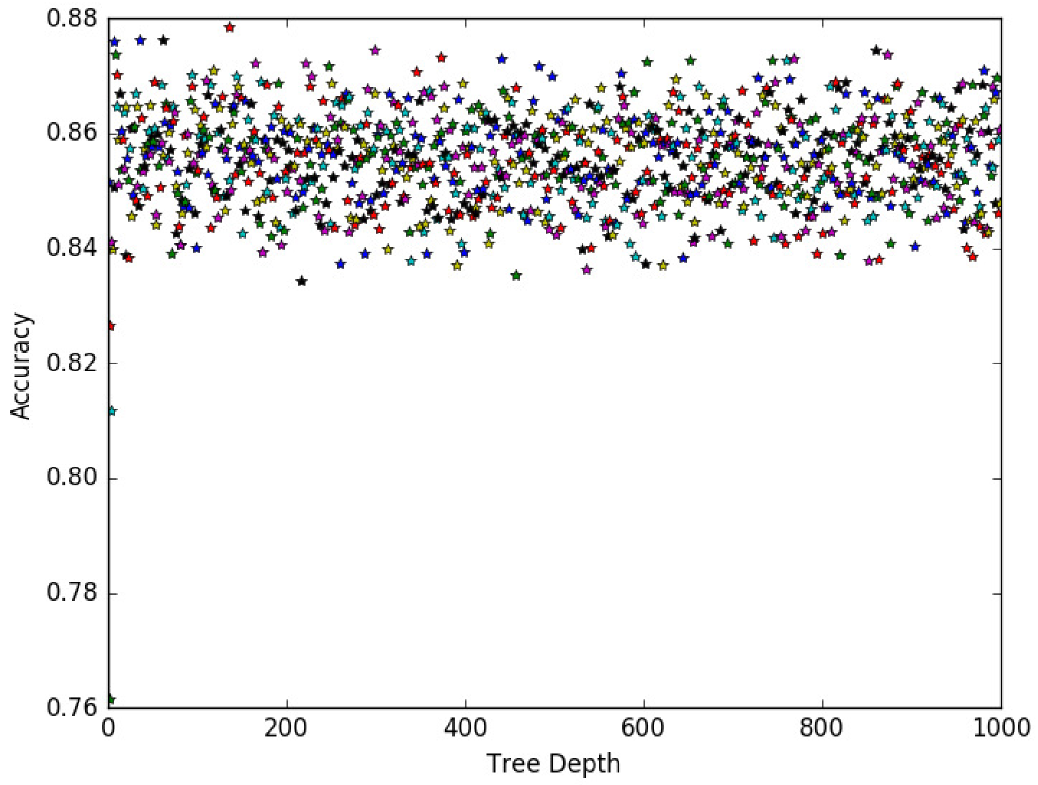


Fig. 1. Accuracy vs. number of leaf nodes

C. Limiting Leaf Nodes

To prevent the tree from growing too large and over fitting we limited the number of leaf nodes to between 2 and 1000. The best accuracy on testing data we were able to obtain by only limiting the number of leaf nodes was 88.09% by limiting the tree to 24 leaf nodes. The trees with leaf node limits of around that of the most accurate tree were also more accurate than many of the other trees. This was an interesting result because it means that with fairly few decisions we were able to correctly classify a large portion of the emails. Below is a plot of accuracy versus number of leaf nodes.

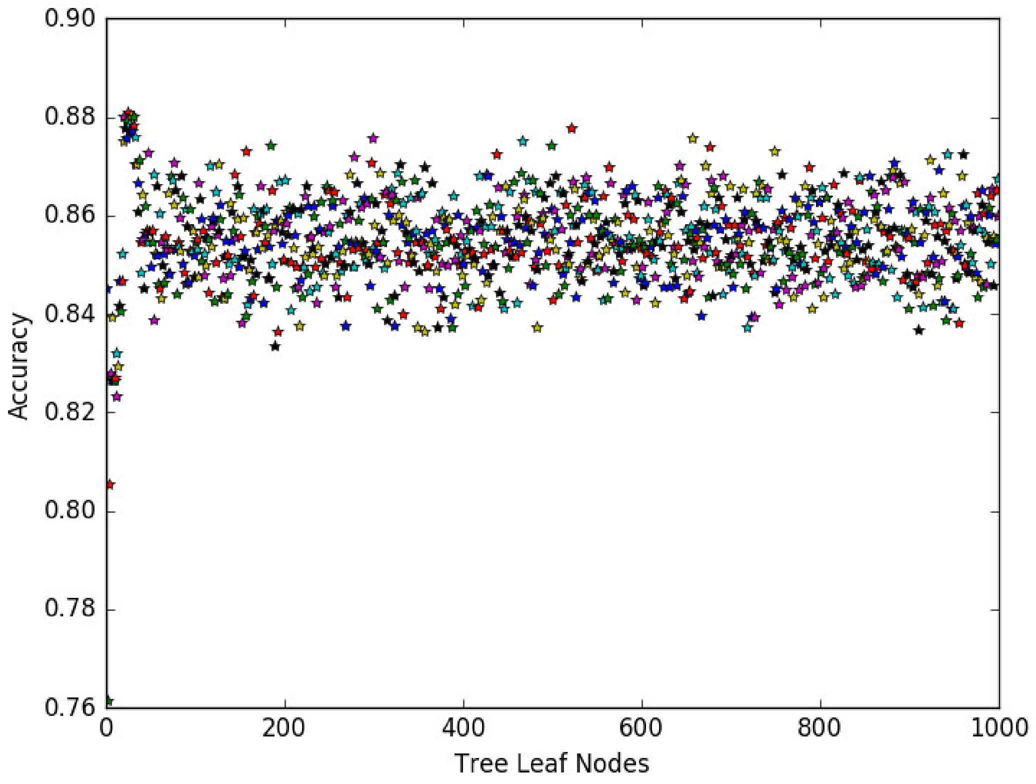


Fig. 2. Accuracy vs. tree depth

VII. Conclusion

VIII. Appendix A - Linux Structs

```
for future code samples
```

IX. Bibliography

References

[1] E. R. Mark Hopkins. (1999, jul) Spambase data set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Spambase