STAT 775 Midterm - April 12, 2016

Timothy Sweet

This report describes the method and results of three classifiers used to classify spam data in the spam dataset (http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/spam.info.txt). A projection with the fisher discriminant, a projection with a random discriminant, and logistic regression each in a binary decision tree is used to classify the data.

The rest of this report is structured as follows. Section 1 describes the tree building approach. Sections 2, 3, and 4 describe the approach, results, and evaluation of the Fisher Classifier, Random Classifier, and Logistic Regression, respectively. Section 5 lists the source code and instructions for executing the code.

1 - Tree Building Approach

The decision tree is computed and used online, minimizing the time needed to traverse the tree and avoiding unneeded traversals. To do this, a recursive function is implemented to evaluate both the training and test data in each recursion, and if certain criteria are met (described in Section 2) it recurses on one or two children, forming a binary tree.

A maximum traversal depth is implemented to avoid infinite recursion. The maximum reasonable depth is the number of samples in the training dataset, thus this value is set as the maximum. Note this criteria is slightly different for the random discriminant as noted in section 3.

2 - Fisher Classifier Approach, Results, and Evaluation

The Fisher Discriminant has been described in a previous assignment (titled “STAT 775 Homework #3 - March 10, 2016”). The description from that assignment is included below for reference:

> The Fisher Discriminant projects a multidimensional dataset onto a single, one-dimensional space. This greatly simplifies the the tasks of classifying data since there is only a single feature in the data. The Fisher Discriminant applies the Fisher Criteria to minimize the projected variance and maximize the separation between two classes of data.

In the decision tree, each node computes the Fisher Discriminant for the training data, projects the training and test data (separately), and assigns a label to each sample in the datasets. These computed labels may or may not match the truth labels, thus a decision must be made based on the following criteria:
If all of the following criteria are true:

1. The set of training samples which are actually part of Class A are not all classified as Class A
2. The set of training samples which are actually part of Class A are not all classified as Class B
3. The set of test samples classified as Class A are not all actually part of Class A
4. There exists at least one test sample classified as Class A which belongs to Class A
5. The set of training samples classified as Class A are not all actually part of Class A
6. There exists at least one training sample classified as class A which belongs to Class A

Then a recursion is performed on this branch (described next). Note that item 2 is a programming check to verify the correctness of the code. Item 5 is the “purity” clause described in the assignment. The other items handle conditions in which there is no data left to classify in either the test or training set[1]. An identical procedure is followed for Class B.

To perform a recursion, the following operation is performed:

```
new_training_data = old_training_data where computed training
classification is Class A
new_testing_data = old_testing_data where computed testing
classification is Class A
```

Then the process is repeated with the new test and training data. An identical procedure is followed for Class B.

The spam dataset contains 3064 samples in the training dataset and 1536 samples in the testing dataset. Of the 3064 training samples, 1218 are spam and 1846 are not spam. Of the 1536 testing samples, 595 are spam and 941 are not spam.

The classifier generates 30 nodes, with a maximum tree depth of 5. This is likely shorter and fewer nodes than a typical solution which builds the tree offline and then tests it online since this tree is allowed to halt early.

The error rate for this classifier at the first node (that is, the non-branched error) is provided in Table 1:

Table 1: Node #1 Error for Fisher Classifier

	Training Dataset Error	Testing Dataset Error
Class A (Not spam)	7.151%	8.289%
Class B (spam)	11.659%	11.597%

The error rate for this classifier at the completion of the tree for the testing set is provided in Table 2. Note that the error of the training set at the bottom of the tree is 0% because it is trained until that threshold is reached.

Table 2: Final Node Error for Fisher Classifier

	Testing Dataset Error
Class A (Not spam)	11.690%
Class B (spam)	9.748%

As seen in Tables 1 and 2, the results of the classifier did not particularly improve with the addition of the decision tree. The intuitive understanding here is that each stage of the classifier

[1] In a real application the training data would be evaluated offline and several sets of testing data would be run through the classifier later. In this case, the classifier would need to be trained to completion. However, since there is no model storage requirement for this assignment, an optimization is performed to limit the depth of the tree be halting evaluation early if there is no more testing data to classify.

has progressively less information to make a decision on, and with such as relatively small dataset there is just not enough nodes in the tree to keep information flowing through the tree. It is often seen that nodes will have less than 30 testing samples in them during computation, often with only a few samples from one class, thus at that point the classifier has practically no information about the data and is just as good as randomly guessing at the data. A larger dataset (if available) may have provided more interesting and useful results.

3 - Random Classifier Approach, Results, and Evaluation

The random discriminant is similar to the Fisher Discriminant in that it is projecting data, however the line to project onto is chosen at random. Although the assignment implied to select the projection completely randomly, a computation optimization is implemented such that if a particular set of projections does produce any split in the data (i.e. it is all projected to one class) then instead of recursing another layer down, a new projection is generated until at least one sample is projected to a different class. This does not change the result of the program, but does allow an intuitive maximum depth to be specified.

The logic for branching is unchanged from the Fisher section (Section 2) thus will not be repeated here.

The random projection is compute by the following pseudocode:

```
Start
	Loop
		NumSamplesProjectedToA = 0
		NumSamplesProjectedToB = 0
		Generate a random projection for each feature in the dataset
		For each sample in the training dataset
			Project the sample into a class
			If sample projected into class A
				NumSamplesProjectedToA++
			If sample projected into class B
				NumSamplesProjectedToB++
		If NumSamplesProjectedToA != 0 and NumSamplesProjectedToB != 0
			Break
		Else
			Continue
End
```

The error rate for this classifier (for some random seed) at the first node (that is, the non-branched error) is provided in Table 3:

Table 3: Node #1 Error for Random Classifier

	Training Dataset Error	Testing Dataset Error
Class A (Not spam)	4.821%	4.888%
Class B (spam)	78.900%	76.975%

The error rate for this classifier at the completion of the tree for the testing set is provided in Table 4. Note that the error of the training set at the bottom of the tree is 0% because it is trained until that threshold is reached.

Table 4: Final Node Error for Random Classifier

	Testing Dataset Error
Class A (Not spam)	51.328%
Class B (spam)	58.487%

First an analysis of Table 3 is provided. At first glance, it may appear that the classifier got extremely lucky and happened to select weights which classified Class A quite well (around 4.8% error). However this is a logical fallacy: this belief relies on the assumption that the data is randomly distributed throughout the dependent variable's space. It is not, however: over 60% of the dataset is not spam, thus by randomly guessing at the data it came close to determining the distribution of spam in the dataset, but did not come very close to identifying the correct labels.

Moving on the Table 4, the results are worse than expected. It appears that the classifier somewhat approached the correct samples for Class B, but performed much worse for Class A. The intuitive understanding behind this is that the classifier happened to find many random projections which were not actually rooted in the structure of the data. This could be equated to randomly drawing a line through a plot and by chance cutting at least one point off the plot.

The small steps are especially evident in a log of the classifier's output: it is common for a node to only cut off a small number of samples (often less than 10). It also ran much longer (around 1000 nodes) and deeper (to level 11) compared to the Fisher Classifier. Interestingly, this also lead the classifier to produce a nearly full binary tree, further suggesting that its cuts were not rooted in anything significant in the data.

This method may be improved by changing how much of the data is projected to each class in any node. If the distribution of classes in the data (here we have about 60% of the samples classified as "not spam") then the classifier could take that knowledge into account at the first level and generate a projection such that 60% of the data went to one class and 40% to the other. This wouldn't work on child nodes (since the distribution is no longer guaranteed) however with random restarting the performance may still improve.

4 - Logistic Regression Approach, Results, and Evaluation

The logistic regression classifier is extremely simple and efficient at classifying the data. Logistic regressions has been described in a previous assignment (titled "STAT 775 Homework #1 - February 25, 2016").

Intuitively, logistic regression is finding a best-fit N-dimensional line for the data, where N is the number of features in the data (in this case 57 plus a homogeneous coordinate).

The tree traversal method is identical to the method described in Section 2 and thus is not described again here.

The error rate for this classifier at the first node (that is, the non-branched error) is provided in Table 5:

Table 5: Node #1 Error for Logistic Regression Classifier

	Training Dataset Error	Testing Dataset Error

Class A (Not spam)	4.823%	5.632%
Class B (spam)	20.279%	23.193%

The error rate for this classifier at the completion of the tree for the testing set is provided in Table 6. Note that the error of the training set at the bottom of the tree is 0% because it is trained until that threshold is reached.

Table 6: Final Node Error for Logistic Regression Classifier

	Testing Dataset Error
Class A (Not spam)	13.709%
Class B (spam)	16.471%

This classifier generally performed worse than the Fisher Classifier but better than the random classifier. Interestingly, its Class A performance decreased with further training, while its Class B performance decreased. A possible explanation for this behavior has already been discussed in Section 2 (with regards to child nodes being uninformed of the state of their parent).

Notably, this classifier runs much faster (approximately two orders of magnitude) than either of the previous classifiers (all three are written with very similar optimizations and coding style and are thus somewhat directly comparable). Thus although its quantitative performance was inferior to the Fisher Classifier, if speed was critical it may be a viable alternative.

5 - Software Code Listing for the Classifiers

A single python program implements all three classifiers, with command line parameters and code macros varying the behavior of the program. The following parameters are implemented:

```
usage: decisionTree.py [-h] [-t TRAINING_FILE] [-d TRAINTEST_FILE]

Process input

optional arguments:
  -h, --help            show this help message and exit
  -t TRAINING_FILE, --training_file TRAINING_FILE
                        submit data to train against
  -d TRAINTEST_FILE, --traintest_file TRAINTEST_FILE
                        List indicating which data is training vs. test
```

Where TRAINING_FILE is the spam dataset or similarly formatted data, and TRAINTEST_FILE is the spam traintest file indicating which samples are training versus testing or a similar file.

The classifier selection is performed via lines 16 and 17 of the program. The following table specifies the behavior of the program for these flags:

FISHER	**RANDOM_CLASSIFIER**	**Classifier Selected**
0	0	Logistic Regression

0	1	Random Classifier
1	0	Fisher Classifier
1	1	Not supported

The program prints to the terminal for each recursion:

- Current tree depth
- Class A training dataset error rate for this node
- Class B training dataset error rate for this node
- Class A testing dataset error rate for this node
- Class B testing dataset error rate for this node
- The direction being branched next (“left” or “right”)
- The number of training and test samples being propagated along the next edge.

After completion, the program prints a summary of the error rate on the testing dataset for Class A and Class B

(code continued on next page)