Galil Gertner, Sarah Mathew
CSCI 493.71
Professor Xie
21st May 2017

Top-K Student's T-Test for Patients with Alzheimer's Disease vs. Patients with No Cognitive Impairment (NCI)

I. Introduction

The purpose of this project is to determine which gene-clusters exhibit the greatest potential to be used as indicator of Alzheimer's disease. To do this, our program evaluates the gene-expression statistical difference for all human genes between patients with Alzheimer's disease and patients that exhibit no cognitive impairment. The program evaluates the gene values for both groups as they are measured in gene clusters that are known to interact together. The mean of the gene cluster values for each group are then compared using the Student's T-Test. The aim is that these clusters can be further used as predictors for the onset of Alzheimer's disease among new patients. The t-test values can then be used in machine learning to determine if they are, or a combination of them can be used as Alzheimer's predictors.

II. Methods

The program takes two csv files from HDFS, gene_cluster.csv and rosmap.csv file. The gene-cluster file lists the cluster ids and the genes that belong in the cluster, while the rosmap file has the gene expression values for each patient. Our goal is to be able to compare the mean and standard deviation for each cluster in each control group. Using the MapReduce paradigm we implement Spark to produce a series of state transformations to regroup the data in order to calculate those values across a distributed database (HDFS). To achieve these transformations, we used a series of "map", "reduce", "filter", "join" and "sort" transformations.

Algorithm Design Idea:

Using the header of the ROSMAP file, we create a reference of (Entrez ID, Column Position) that tells the column position that a gene value has in the file. We also make a flat map of (Entrez ID, Cluster Number) from the gene_cluster file, so that we can join the two on the Entrez_ID and to yield the result: (position, cluster_number).

We filter out only NCI and AD patients into separate groups. For each one, we map a column position number to each gene-expression value for each patient so that each patient's gene value is paired with its location (patient_id, (1, gene_value_1), (2, gene_value_2), …, (n, gene_value_n)). We save the size of this RDD variable because it provides us the number of patient records for that particular patient type. We flatmap these values, and switch the order of each entry so that the key is the position, and the patient id and gene_values are value-pairs: (position, (patient_id, gene_value)). We can then join the (position, cluster_number) to (position, (patient_id, gene_value)) on the position key. We get a long list of positions as the key: (Position, ((Patient_id, gene_value), cluster_number)). The following transformation is what the algorithm hinges on: we map the values such that we drop the position, the cluster_number and patient_id become a key-pair, and the gene value becomes the value: ((cluster, patient_id), gene_value). Now we can reduce by key summing the values because each key will be a (cluster, patient_id), so each summation will only be between the values of one patient on one cluster. We now have a long collection of clusters as keys and cluster_values as values. This gets saved to be used later to calculate deviation from the mean.

The next step is to get the mean values for each cluster score and the standard deviation. First we will reduceByKey with the cluster as key, summing the values. It remains to calculate the mean values for each cluster score and the standard deviations, which are straight forward transitions. With our (cluster, sum) pairs, we find average using the count. Then, the (cluster, avg) values are joined to another rdd to create (cluster, (avg, [a list of patient cluster values])). We then find the standard deviation using this information. We filter for just one line from the rosmap data, the header line.

With information from both AD and NCI clusters, we join the groups and then perform a t-test and create the final (cluster, (statistics)) pairing. Finally we locate and return the top 10 of these values.

Part 1: File Transformation

A. gene_cluster.csv: We create map entrez ids to their column position in the file

	Action	Resulting Pair in RDD
1	Filter for human records	
2	Map cluster id to list of entrez ids	(cluster, [entrez, entrez,...,])
3	Flatmap cluster id to each entrez id	(cluster, entrez)
4	Switch key-value	(entrez, cluster)

B. rosmap.csv: we map entrez id to its location on the column

	Action	Resulting Pair in RDD
1	Filter for the header row	(‘PATIENT_ID’, ‘DIAGNOSIS’, ‘1’, …)
2	Create pairs with each header value and its position. ‘PATIENT_ID’ will be the key for this list of pairs	(‘PATIENT_ID’, [(entrez, pos), (entrez, pos),...])

3	Flatmap ‘PATIENT_ID’ to each pair	('PATIENT_ID", (entrez, pos))
4	Remove ‘PATIENT_ID’ as the key	(entrez, pos)

C. gene_cluster & rosmap info: map locations to each cluster

	Action	Resulting Pair in RDD
1	Join our rosmap pairs to cluster data	(entrez, (cluster, pos))
2	Remove entrez id, switch cluster and position	(pos, cluster)

Part 2: Statistics for Groups

A. Determine the gene sums for each cluster

	Action	Resulting Pair in RDD
1	Filter rosmap.csv for particular records (NCI or AD)	
2	Map each value in the patient record to its position	(patient_id, [(pos, gene_value), (pos, gene_value),...])
3	Flatmap patient id to each gene_value pair	(patient_id, (pos, gene_value))
4	Switch key-value	(pos, (patient_id, gene_value)
5	Join rosmap gene pairs and cluster pairs	(pos, (cluster, (patient_id, gene_value)))
6	Remove position and pair cluster and patient_id as the key	((cluster, patient_id), gene_value)
7	Reduce by summing gene_values	((cluster, patient_id), cluster_sum)
8	Remove patient_id info	(cluster, cluster_sum)

B. Find the mean and standard deviation for each

	Action	Resulting Pair in RDD
1	Put each cluster_sum into a list	(cluster, [cluster_sum])
	Put the sums into one list	(cluster, [cluster_sum, cluster_sum, …])
2	Reduce the cluster_sum	(cluster, summed_clusters)
3	Divide the summed_clusters by number of records	(cluster, mean)
4	Join the cluster average with the cluster list	(cluster, (mean, [cluster_sum, cluster_sum,...]))
5	Create a list of differences between the average and cluster_sums	(cluster, (mean, [difference, difference, …]))
6	Sum the differences and divide by number of records	(cluster, (mean, sum_of_difference/number of records))
7	Find the square root of the sum/number	(cluster, (mean, standard deviation))

Part 3: t-test

A. Perform the t-test evaluation

	Action	Resulting Pair in RDD
1	Join our statistical information for AD and NCI groups	(cluster, ((mean$_{AD}$, standard deviation$_{AD}$), (mean$_{NCI}$, standard deviation$_{NCI}$)))
2	Run t-test on information and include it in pair	(cluster, (t-test, ((mean$_{AD}$, standard deviation$_{AD}$), (mean$_{NCI}$, standard deviation$_{NCI}$))))
3	Return top 10 values for t-test	

Running the program locally:

1) Upload your files into HDFS

    a) Hdfs dfs -put <file_1>

    b) Hdfs dfs -put <file_2>

2) run python AD_interface.py

Running Program on Amazon

1) spark-submit --num-executors 3 --executor-cores 3 PY2_Top_k.py

III. Results

A. Top 10 t-test values

	Cluster ID	t-test	AD mean	AD std	NCI mean	NCI std
1	5401	7.423219027751293	148.84174887892374	29.0072360554775	129.31983333333332	23.78491908927447
2	2762	6.354783759211143	37.9514798206278	10.420656919563332	32.45133333333333	6.869427147392518
3	3025	5.658841989217762	13.696322869955154	5.3041291068236305	11.039333333333333	4.119882900169749
4	2378	5.64208981501117	25.270538116591933	5.5093239904281175	22.492722222222223	4.373960553416035
5	5467	5.373468390481473	62.570627802690574	23.3419782086066	52.3986111111111	14.326145436613004
6	2842	5.327669913585766	59.82327354260089	14.550640580464355	52.8748888888889	11.630825254416195
7	5464	5.32136144358799	64.96345291479821	23.96516579298902	54.619388888888885	14.716428014821807
8	3028	5.254801415009784	16.092062780269057	5.749004925058393	13.390277777777776	4.57231590122744

9	5475	4.9102767009177235	81.88295964125561	23.433401599491162	72.46638888888889	14.78994777858292
10	3160	4.740172402701689	10.02542600896861	3.177089464314196	8.6625	2.594869055784254

B. Estimated time complexity vs number of node

<u>Data in 4 Partitions (On local hdfs)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)	4 nodes (secs)
74.56	48.25	43.68	45.03
74.43	47.67	45.03	40.99
73.26	49.09	42.73	40.81
73.62	47.66	41.60	40.81

<u>Data in 6 Partitions (on local hdfs)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)	4 nodes (secs)
68.64	46.00	44.7	44.01
65.71	42.29	40.05	40.79
64.87	42.08	38.83	39.32
64.87	40.82	38.87	38.6

<u>Data in 8 Partitions (on local hdfs)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)	4 nodes (secs)
73.68	49.41	47.90	47.29
68.76	45.04	42.98	43.91

68.37	44.09	47.9	47.24

<u>Data in 4 Partitions (on AWS EC2)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)
65.60	80.36	81.05
71.80	69.72	64.15
65.99	69.81	85.02

<u>Data in 6 Partitions (on AWS EC2)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)
55.50	56.02	66.12
55.50	60.08	59.43
55.71	57.38	53.50

<u>Data in 8 Partitions (on AWS EC2)</u>

1 node (secs)	2 nodes (secs)	3 nodes (secs)
53.28	48.95	56.01
57.18	52.25	50.02
54.77	50.73	59.38

Executor Variance (3 cores)

<u>Data in 4 Partitions (on AWS EC2)</u>

1 (secs)	4 (secs)
75.33	65.12

83.26	63.83
96.86	59.97

<u>Data in 6 Partitions (on AWS EC2)</u>

1 (secs)	4 (secs)
67.86	54.18
67.49	64.20
66.64	62.07

<u>Data in 8 Partitions (on AWS EC2)</u>

1 (secs)	4 (secs)
65.48	50.76
58.60	49.25
64.48	49.32

IV. Machine Learning

In order to use the python MLlib package to run machine learning algorithms on the data, we first needed to clean the data and reshape it so that it would fit into LabeledPoint format. LabeledPoint format looks like a key-value pair where the label is the first values, followed by a python list of the features. Using top 10 cluster information, as determined by the Student's t-test, as the feature values we labeled the alzheimer's disease control group with a feature label of 1.0 and the NCI control group with a label of 0.0. (*Note*: In joining the top 10 t-test cluster values to each patient's data, the rank order of the clusters was not preserved, but they were reordered consistently relative to each other and thus usable as MLlib feature values). Thus the dependent variable in our project is binary -- it is set to 1 when a patient has AD and a 0 otherwise.

We started by building a decision tree which is a highly interpretable model. Besides interpretability, decision trees can handle both numeric and categorical inputs. They require almost no data preparation, are not affected by outliers or missing values and do not assume an underlying relationship between the outputs and inputs.

To evaluate the model's true performance, 3-fold cross validation was used. We started by splitting the data into 3 random samples. To make our code reproducible, we set a seed to get the same random samples every time the model is run. In the first step, we used samples 1 and 2 to train the model and sample 3 to evaluate the model's performance. In the second step, we used samples 1 and 3 for training purposes and sample 2 for validation. In the last iteration, we used samples 2 and 3 as training samples and sample 1 as a testing sample. During each iteration we recorded misclassification errors and averaged them to get the final error. In addition, confusion matrix was used to get the count of True Positive, True Negative, False Positive and False Negative.

	Predicted	
Actual	0	1
0	TN	FP
1	FN	TP

First Iteration: Decision Tree, Gini Splitting Criterion, Max Depth of 5, Top 10 gene clusters

In the first iteration, we built a model that used only top 10 gene clusters as inputs. We selected Gini criterion that splits a node to minimize impurity its node or to make it more homogeneous.

We restricted the depth of the tree to 5 which served as a stopping criterion. Since the number of patients in our data is only 500, we didn't want the decision tree to be too deep.

Second Iteration: Decision Tree, Entropy Splitting Criterion, Max Depth of 5, Top 10 clusters

In the second iteration, we decided to experiment with entropy splitting criterion.

We expected to see results that are comparable to the results of the first iteration.

We found slightly better results using the entropy stopping criterion.

Third Iteration: Random Forest, Gini Splitting, Top 10 Gene Clusters

Unlike decision trees, random forest is not an interpretable model. It is, however, a more sophisticated modeling technique. Random forest usually provides more accurate results due to lower variance and is less likely to overfit the data.

The trees are built independently of each other on randomly selected portions but with replacement of the dataset. For each tree the inputs are also selected at random. This helps to minimize bias in the model. The predictions of all trees are aggregated. Our first random forest consisted of 25 trees. We then increased the number of trees to 50, and tried including age as a feature. The models' misclassification errors are shown below .

Testing Sample	Misclassification Error
Mean Gini Error	0.4036

Mean Entropy Error	0.4011
Mean RF Error with 25 trees	0.340
Mean RF Error with 50 trees	0.336
Mean RF Error with 50 trees and age statistic	0.35

Confusion Matrix for Random Forest with Age factor

True Negative (%)	0.32
True Positive	0.336
False Positive	0.112
False Negative	0.232

V. Discussion

A. Issues

The algorithm utilizes the position of each entrez_id in the file. In the future, if we try to add more rosmap data, we must depend that these gene ids continue to appear in the same positions that they did in the original file. This may be likely, considering that the number of genes and clusters would be constant, in addition to the data being sorted well. The algorithm also spends a few operations on simply switching values within the key-value pair. When we calculate the t-test values, we do numerous operations on the value for each pair. It would probably be better to make these operations singular and thus, simpler.

In our empirical time tests, we noticed a marked time speedup across the board if the data was run on multiple nodes. The greatest increase is achieved in simply switching from one node to two nodes, and then continuing to increase up to four nodes, but at a diminishing rate. We also saw that partitioning the data into separate RDD files also increased speed, but actually decreased once the data was partitioned into more than six parts. Too many partitions result in loss of speed. The optimal settings for this Macbook Pro 2.7 GHz Intel Core i5 being reached four nodes with the data partitioned into six RDDs. We also noticed a minimal, but dependable speedup in the empirical tests as they were repeated on with the same settings, indicating that there is likely some kind of time saved in not repartitioning the data, and also, that some data references or settings may be cached and reused.

B. Benefits

We filter for relevant records early on, for human gene clusters and for NCI and AD patient records. This allows us to avoid many operations on extraneous data. In addition, we have abstracted the method to determine the statistical values for both NCI and AD gene cluster information. All of our data is also kept on HDFS, nothing is kept locally (ie. a list).

V. Conclusion

Form the machine learning it appears that random forest performs better than decision trees, which we anticipated. While the misclassification error appears slightly higher than without age, we feel it is a powerful predictor and should be included in the final model.