Large-scale Machine Learning and Statistical Analysis of Dark Matter Halos Using Apache Spark

W251: Scaling Up! Really Big Data - Spring 2015

Samuel Kahn, Lisa Kirch, Nikhil Gopinath Kurup, Wei Shi, Bovard Tiberi

Summary

As part of the Scaling Up! Really Big Data class, we examined a number of machine learning algorithms in a variety of languages to explore large-scale cosmological data in a manner which has not previously been done. Starting with 2 terabytes of halo catalog data, we built a pipeline in the SoftLayer cloud to preprocess this data to get it ready for statistical and machine learning analysis. The scalable pipeline we built should be capable of streaming data from a new simulation run as it is generated into the preprocessor in order to capture the data time step by time step. In addition, this data can be enhanced with observational data to do further statistical correlations and analysis. In this paper, we present the methodology of how the pipeline was created, the tools used, provide preliminary results and links to our code so others may reproduce our work and to allow for more in-depth future analysis.

Introduction

Background

The Big Bang theory[1] is the most commonly accepted model for the origin and evolution of the universe. Numerous predictions originating from well understood theoretical models, supplanted with observational evidence has led scientists to believe that is how The Universe started.

However, this model is not without its problems: according to this model, quantum fluctuations in the initially expanding universe amplified over the life of the universe lead to clumpy clusters of matter which eventually formed galaxies and other structures. The visible matter that we see in the universe does not account for enough gravity[2] to form the dense structures that we observe, at least not in these time scales. It was almost as if there exists more undetected matter in the universe that does not emit light, heat or radio waves but interacts with electromagnetic waves through gravitational forces - a kind of "dark matter."

Observation of the rotation of galaxies[3] supported this hypothesis. Stars and gases in spiral galaxies rotate around their galactic cores at speeds that should be governed by the mass of the entire galaxy. Estimates of the mass of the galaxies are done by using well understood models that measure the brightness and intensity of the galaxies and the stars within them. However, the observed speed of the rotating stars would require the galaxy to have many times more matter than what was suggested by the observation of mass. This again suggests the existence of dark matter.

---

[1] http://map.gsfc.nasa.gov/universe/bb_theory.html
[2] http://www.physicsoftheuniverse.com/topics_bigbang_darkmatter.html
[3] http://www2.lbl.gov/Science-Articles/Archive/sabl/2006/Jan/Rubin-Dark-Matter.pdf

Dark Matter Halos

One of the prevailing theories explaining dark matter is WIMPs[4] (Weakly Interacting Massive Particles). These are hypothetical particles that do not emit light, heat or electromagnetic radiation, but can interact with regular matter through gravity. As they coalesce, they form an envelope around galaxies, extending well beyond their visible edges. This contributes to them becoming the dominant mass of the galaxy. While these particles cannot be directly observed, they do explain the two failings of the Big Bang model we discussed above. These envelopes of dark matter around galaxies are known as "dark matter halos"[5]

Dark matter halos can be spherical, ellipsoidal, or irregularly shaped. It is interesting to explore the shape of the dark matter halos because they are the nurseries in which galaxies are born. Understanding the shape gives us much greater understanding of how galaxies are formed and why they look the way they do. The shape of the halo also has strong implications of where a galaxy is in the universe, or what cosmologists call the Cosmic Web.

Bolshoi Simulation

The Bolshoi simulation[6] is a cosmological simulation of the large-scale structure of the universe that was run on the Pleiades supercomputer at the NASA Ames Research Center in 2010. The simulation took roughly 2 million processing hours to complete. The simulation uses the current models of the Big Bang, seeded with measurements of the initial conditions of the universe from observations of the cosmic microwave background and best guesses from theoretical models. Dark matter is a key part of the simulation and relies on a model known as the Lambda Cold Dark Matter model. It tracks the evolution of 8.6 billion particles of dark matter, each of which represents about 200 million solar masses, in a cube 1 billion light years on edge. As better observations of the initial conditions of the universe became available through better and more advanced telescopes, subsequent runs of the simulations were done - the Bolshoi Planck simulations.

Data from the simulations were collected at discrete time steps known as the cosmological scale factor and recorded. This gives us the view of the universe during different points in time which can then be validated against observations. This data was found to be be in agreement with observations and predictions and a deeper understanding of the structure of the universe has ensued[7].

The outputs of the simulations are raw particle data, simply documenting the position and velocity of each particle in the simulation. The mass of a particle in this simulation is roughly 100 million times the mass of the sun. The raw particle data, which is roughy a petabyte, is stored on an Network File System (NFS) on the Pleiades

---

[4] https://www.astro.umd.edu/~ssm/darkmatter/WIMPexperiments.html
[5] http://www.ucolick.org/~diemand/vl/publ/dm_dm_minirev.pdf
[6] http://hipacc.ucsc.edu/Bolshoi/images/PrimackBell_S&T_Bolshoi%207-2012.pdf
[7] http://hipacc.ucsc.edu/Bolshoi/

supercomputer. Instead of analyzing the raw particle data, we analyzed what are called halo catalogs. These halo catalogs are derived from the raw particle data and document the more interesting properties of dark matter halos such as shape, angular momentum, and spin.

Halo Analysis

One of the critical outputs captured in every time step in the Bolshoi simulation was dark matter halos as the billions of dark matter particles started coalescing into defined structures. As the universe evolves, the degree of influence of the different features on the shape and structure of the universe changes over time. With the Bolshoi simulation data, we can see how the shape and size of the dark matter halos change over time. The objective of this project is to setup the data and compute the infrastructure that can help in analyzing this data and discern patterns. We will attempt running analysis that generates summarized information from the models. Scientific analysis of these results, however, are outside the scope of this project.

Goals

The object of this project could be summarized as follows:

- Build a scalable pipeline to download the Bolshoi and Bolshoi Planck simulation data. The pipeline should also be capable of streaming data from new simulation runs.
- Make the downloaded data available as a shared resource for different models of scalable computation on commodity hardware or multi-tenant cloud infrastructure
- Set up a repeatable process for spinning up and scaling compute resources for different types of data analysis
- Conduct basic correlation analysis among the various features of the simulation data and generate summarized correlation data to conduct further statistical analysis
- Generate a decision tree model to determine the most important features in determining dark matter halos
- Present preliminary results from the analysis
- Recommend future analysis steps

Approach

The cloud computing resources provided for this project was IBM's Softlayer[8] which is an IBM product for dedicated servers, managed hosting and cloud computing. Each individual member of the team was provided a $1,000 per month allowance on the platform. Initial calculations of data volumes and computing needs hinted that we may have to pool the resources from all members. When this was requested to IBM, they graciously made a grant for $10,000 to one of the accounts towards this project.

The set of tools that was initially planned for executing the project included:

- Data transfer - FTP, rsync, scp, Netcat

[8] http://www.ibm.com/cloud-computing/us/en/softlayer.html

- Data storage - Block storage, HDFS cluster
- Cluster provisioning - Vagrant, Puppet, ansible
- Scalable computing - Hadoop with Yarn, Spark
- Statistical analysis and Machine learning - MLlib, NumPy, SciPy, pandas, PySpark
- Programming languages - Scala, Python
- IDE - Eclipse, IPython, PyCharm
- Build and deployment - sbt, shell scripts
- Monitoring - nmon
- Collaboration - ISVC, Google Drive, Google Hangouts, Speek.com, Email
- Source code hosting - Github

The source code repository for this project is available at:
https://github.com/nikhilgk/halo-ml

Data and Compute Management

Data Storage And Transfer

Our data was obtained from Joel Primack, Distinguished Professor of Physics at the University of California, Santa Cruz and Director of the University of California High Performance Astro Computing Center. Joel and his colleagues were the first team to theorize the currently accepted theory of dark matter and more recently they have been running the largest and most accurate dark matter cosmological simulations. He has kindly given us permission to use the data from these simulations for our project. Although we had access to four different simulations, they do not differ greatly so we chose to analyze the Bolshoi simulation. The Bolshoi simulation is roughly 2 terabytes in size and documents dark matter halos as they evolve from the big bang to today in 180 timesteps.

We transferred the data from the NFS on the Pleiades supercomputer to a data storage node we set up through SoftLayer. Our 'datastore' node had 12GB of RAM, 2.1 TB of disk space, a 1 Gigabit per second connection. It took roughly 6 hours to transfer our data from the NFS at NASA to our 'datastore' node at an average transfer rate of 702.5 Megabits per second.

Since we were most interested in analyzing dark matter halo shape, there were some natural preprocessing steps we made in order to make the data more manageable. The algorithms which determine the shape of the dark matter halos are not accurate for small halos because they do not contain a lot of particles and are thus not well-defined. So limit our analysis to halos which are only larger than 10^10 solar masses. This allows us to analyze those halos that are well-defined and reduces the size of our data. In addition, we removed some features from our dataset which would not be useful. By taking these two preprocessing steps our data went from roughly 2 terabytes to 308 gigabytes. Note that this data is partitioned across 180 timesteps.

We can see the effect of our data preprocessing step in Figure 2 in the Appendix. The x-axis represents the scale factor, a number between 0 and 1, which is analogous to time. A scale factor of zero represents the time described by the Big Bang and a scale

factor of one represents today. If we take the logarithm of the scale factor and plot it against the fraction of halos larger than 10^10 solar masses, we get a linear relationship. This means that the fraction of halos that are larger than 10^10 solar masses increases log-linearly with time. This also implies that the size of our data, reduced by preprocessing, also decreases log-linearly with each timestep.

Compute Management

It was apparent early on that we would need provisioning tools to setup, manage and teardown the clusters used for computation. Initially a combination of Vagrant and Puppet were identified to be a good combination for achieving this. But as we started building the configuration scripts, numerous incompatibilities and version mismatches between existing puppet packages, softlayer provisioning and hadoop/spark versions started showing up. At this point we switched over from puppet to ansible and to the vagrant-cluster[9] setup project that had already addressed many of the issues we were facing with puppet. We tailored[10] this provisioning setup to our specific needs which included:

- Normalize all git sub-module dependencies
- Adding support for a cluster to join named private or public VLANs instead of creating one every time
- Defaulting the cluster communication to go over private IP VLANs instead of public IPs
- Auto mounting the shared data block storage volume to the cluster's master node in read only mode
- Including utility additions like colored bash prompts and default authorized keys
- Installing all necessary python, MLlib, and scikit-learn modules

Correlation Analysis

The Bolshoi simulation data measures numerous basic and derived features of the halos as they evolve over time. The evolution is driven by mergers between different halos and the underlying expansion of the universe. This would mean that the relevance of different features changes over time. As an example, as halos evolve, many halos tend to become more elliptical in shape - this can explain the elliptical nature of most observed galaxies. It will be interesting to see if this shape is influenced by the mass, angular momentum or the number of smaller halos that merged to become bigger halos. It will also be interesting to see how these correlations change over time.

The objective of this analysis is to generate a 62x62 grid for all the filtered features for a given time step. Since there are 180 time steps, we will generate as many correlation grids. Once done, we can then visualize the changes in these correlation parameters over time and make the data available for further statistical processing. These correlations can be further analyzed for two purposes:

[9] https://github.com/irifed/vagrant-cluster
[10] https://github.com/nikhilgk/halo-ml/tree/master/vagrant-cluster

- To get a deeper understanding of the evolution of halos
- To simplify the computing model behind the Bolshoi simulation so that future computations become cheaper/faster/easier.

We will use the Pearson product-moment correlation coefficient as the measure for correlation.

Approach 1

Given the volume of data and the number of correlations, we posited that the data distributed in an HDFS cluster with correlation analysis done through Python's scikit-learn over a Spark cluster would be a good fit. Using PySpark and MLib worked well for files of up to roughly 500 Megabytes, but files of any larger size gave us "Out of Memory Errors." We were unable to resolve this problem. In general, this lead us to an important conclusion, using Spark's Python wrapper was much less efficient and was not really what Spark was designed for. As we later explain, we had much more success using Scala.

Approach 2

After being unable to scale scikit-learn, a second attempt[11] was made to generate the correlation matrix using Scala on Spark with the data again in an HDFS cluster. Once the code was found to run successfully on a local cluster, it was time to run the analysis at scale on a bigger cluster.

In order to scale the cluster sizes, we ran a series of trial runs to estimate the computing requirements. A sample subset of data was used to run these trials.

Trial 1

2 nodes with 4 cores and 8GB RAM in each node
- Average time per correlation = 180 seconds
- CPU usage - maxed out
- Memory usage - maxed out
- Network usage - minimal
- Disk usage - minimal

Total compute time needed per correlation = 180 seconds * 4 core * 2 nodes = 1440 seconds

Trial 2

4 nodes with 12 cores and 32GB RAM in each node
- Average time per correlation = 37 seconds
- CPU usage - maxed out (see attachment)
- Memory usage - does not go above 12G per machine
- Network usage - minimal
- Disk usage - minimal

[11] https://github.com/nikhilgk/halo-ml/tree/master/nikhil/scala-corelation

Total compute time needed per correlation = 37 seconds * 12 core * 4 nodes = 1776 seconds

From these two trial runs, we can see that the processing is CPU intensive and that we are below the threshold for disk access, RAM and network. With this fact secure, we planned to create a 20 node cluster with 16 cores each and 16GB RAM on the public cloud infrastructure. Since we had approximately 2TB of unprocessed data and with a default replication of 3, we need total of 6TB on the cluster, which leads to 6000GB/20 = 300 GB per node of local block storage. Adding 1GB/s NIC for the best network performance, the total cost of each instance adds up to around 60 cents per hour including tax and bandwidth charges. This setup adds up to 20 * 16 = 320 Compute cores. From the trial runs, processing took 1440 and 1776 seconds per each correlation; this can be extrapolated to 2000 compute seconds per correlation in a bigger cluster.

This amounts to 2000 seconds / 320 cores = 6.25 parallel seconds per correlation. We will round this to 8 seconds to allow for a buffer. Now choosing just the important features leads to (40 * 39) / 2 = 780 correlations per file (since we do not want to do the diagonals and eliminating the duplicates). Since we have 180 files, this adds up to a total of = 780*180 = 140400 correlations.

Total runtime for calculating all these correlations = 140400 correlations * 8 seconds / correlation = 1123200 seconds = 312 Hours = 13 days.
Also this will end us costing 312 Hours * 60 cents * 20 nodes = $3744

Before we launched this processing on a cluster, we decided to try alternative approaches and optimization to cut down the computing requirements.

Approach 3

After analyzing the bottlenecks with the Scala on Spark approach, it became apparent that a more columnar approach of data storage and distribution would be a better fit to the problem. This is because of the manner in which the correlations are calculated. It requires that all the data for the two columns be available locally for calculation. In the row-wise data structures that we used the data gets chunked row-wise and gets distributed across the nodes. From a rough estimate, it became apparent that for a typical file with about 4 million rows, calculations will span about three quarters of the cluster nodes. Two solutions were considered at this point:

- Rebuild the Spark cluster using Cassandra or any other columnar database as the underlying data storage medium
- Attempt to distribute the data by file boundaries onto different machines and use a simpler linear approach to calculate the correlations per file

The second approach proved to be more effective after simple tests using Python and NumPy showed that NumPy was far more performant in calculating a correlation matrix as long as it can load all the data into memory. With that as a potential approach, we created a 6 node cluster with 32GB RAM and 8 cores. Further, we shipped the 180 step files equally to each of these clusters. With more robust error handling Python code[12] and shell scripts to manage task executions, we were able to calculate the correlation matrix for all input files on this cluster in under 15 hours.

The summarized data from the correlations were exported as pickled Python data set files; the entire summary data set is just a 8MB in size. This can be either analyzed further using iPython notebooks or exported to other formats and tools. Some sample correlations are shown in Appendix 3, a more detailed analysis and interpretation would require deeper understanding of cosmology and is beyond the scope of this project

Feature Importance

Given the massive amount of data that we had processed, we could begin asking some more interesting questions such as "what features are most important in determining halo-shape?" and "do these features change over time?" Usually these questions would be very difficult (if not impossible) to answer when looking at this much data. However by using machine learning techniques at scale we hoped to be able to answer them.

Feature importance was of interest to our analysis because up until this point anyone studying these models only had their intuition to guess what features of the model were most important to predicting a halo's shape. Even if we are able to extract some sort of feature importance it will be the first time something other than intuition was used.

Approach 1

To begin with, we thought principal component analysis (PCA) would be a good choice to do some of this analysis. During our first approach we attempted to get PCA running over each time series in the data set. However, after delving more into the inner workings of PCA it became evident that it would not be a very good fit for finding feature importance. To train the PCA model we had to reduce the dimensionality of the data, which made it very difficult to get feature importance in a meaningful way.

We decided that Decision Trees would be a better choice as they naturally lend themselves to finding feature importance. That would be our next approach.

Approach 2

We decided to try out Python on Spark to train the decision trees. The first thing we did was to use Vagrant to provision a 3-node cluster. Our plan was to use the scikit-learn decision tree[13] to train the model. However, we soon ran into problems. Input files of under approximately 500 MB passed successfully through but anything above that would throw an "OutOfMemory" exception. This was a very large problem for us because most of our files

[12] https://github.com/nikhilgk/halo-ml/tree/master/nikhil/ipython
[13] http://scikit-learn.org/stable/modules/tree.html

were around 2GB in size. There were several different approaches we tried to solve this problem, but none were successful. Ultimately we abandoned this approach, hoping that switching to Scala would solve the issue.

Approach 3

As the semester deadline raced toward us we placed our last hope of getting feature importance to work in the Scala MLlib decision tree model[14]. After coding up something, it finally worked! And it worked on the full 2GB file. It was time to scale up!

We planned to put the full dataset into HDFS on the Spark cluster. For this we spun up an 11 machine (100 GB HDD, 4 GB RAM, 2 cores) Spark cluster. We loaded the full dataset into HDFS and prepared to run the job.

At this point we had to actually figure out how to extract the feature importance out of the resulting model. To do this we had to inspect the resulting model. The decision tree was represented as a binary tree. At each node, there was a split on a single feature based on a threshold. The node also recorded its information gain. By traversing these trees and recording the total gain from all nodes in the tree for each feature we were able to extract the relative feature importance.

The last thing to do was to run across all of the timesteps. We ran three runs, each of which took about 13 hours to complete. We discuss the interesting results from these three successful runs in the Analysis and Conclusion section.

Analysis and Conclusion

The nature of our experiment allows us to draw thousands of different conclusions, particularly from the correlational analysis. Appendix 4 shows the two most interesting correlations we were able to extract from our analysis. We find that the halo shape starts out negatively correlated with the radius of a halo and gradually becomes positively correlated as The Universe evolves. We also see that shape of the correlation between the shape and speed at which the particles inside the halo are moving starts out positively correlated, becomes negatively correlated, and then gradually becomes positively correlated as The Universe evolves.

In Appendix 5, we see the plots showing the feature importance for the simulation. The most interesting plot is the average feature importance for each feature. Surprisingly, the halo shape seems to be highly dependent on the x, y and z coordinates of the halo in the simulation. This likely means that halo shape is highly dependent on the position of the halo in the Cosmic Web and the way matter is flowing into the halo. We also find that halo shape is highly dependent on how long ago a halo merged with another halo, its velocity in space, its mass, its angular momentum and its spin. The plots of the feature importance over time are unfortunately very noisy and harder to compare. They do provide insight into how the importance of features change over time.

[14] https://spark.apache.org/docs/1.2.0/mllib-decision-tree.html

Over the course of project we touched a lot of the tools used in Big Data in order to analyze large cosmological simulations. We used provisioning tools such as Puppet and Vagrant, languages including Python and Scala, data transfer tools like scp and ftp and worked on a large distributed Spark cluster running statistical analysis and machine learning.

Using all these tools, we were able to successfully test our hypotheses on the 2 terabyte Bolshoi simulation. A very important takeaway from this analysis is that Spark is best suited to be used with Scala, its native language. We encountered many errors and difficulties using Spark's Python wrapper PySpark, spending nearly a month trying to fix our code. When we finally switched to Scala, we were able to complete our analysis.

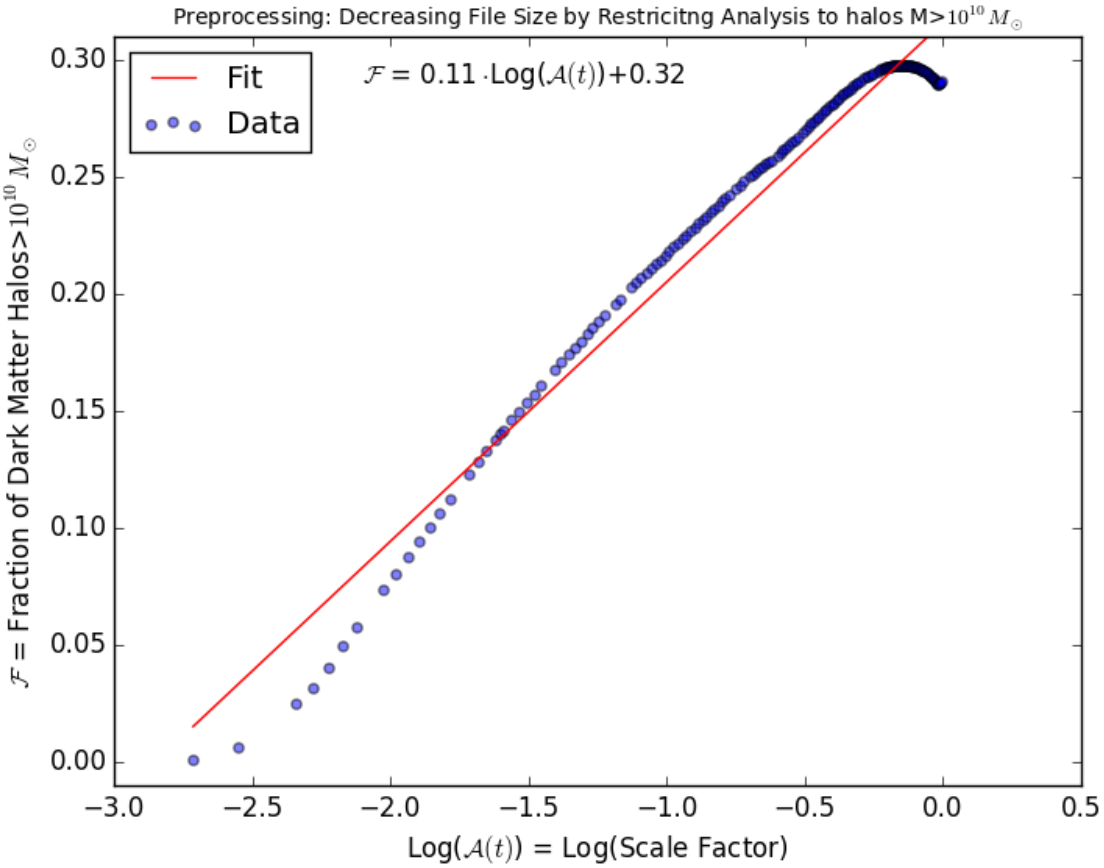
Though we had limited time for analysis, much of what we have produced has been quite interesting. We are publishing our results up to github to allow for future analysis and reproduction of our work. Large scale machine learning analysis has never been done on any of this data and with more analysis the results will quite likely be publishable. With this being said, in order to publish these results we will need to work with Joel Primack to better understand the physical meanings and implications of our findings.

Appendix

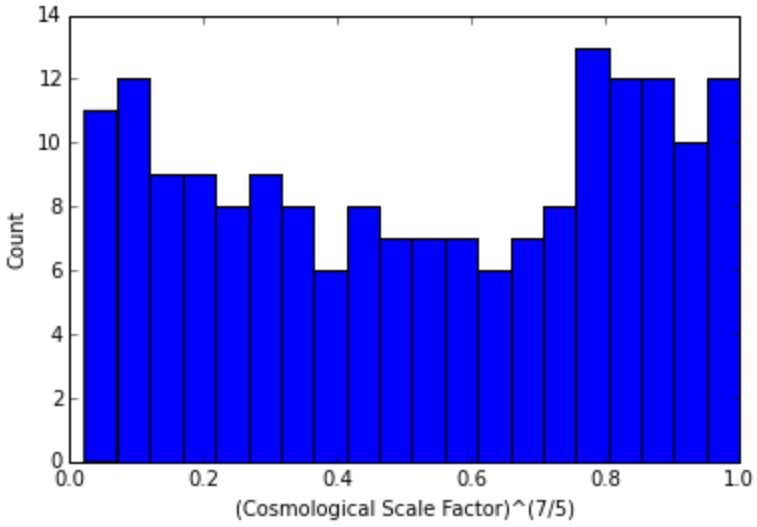
1 - Bolshoi and Bolshoi-Planck Data Fields:

#scale	#Scale: Scale factor of halo.	#Omega_M = 0.307115; Omega_L = 0.692885; h0 = 0.680000
		# Shapes are determined by the method in Allgood et al. (2006).
		#Consistent Trees Version 1.0
		#Includes fix for Rockstar spins & T/|U| (assuming T/|U| = column 53)
0	id	#ID: ID of halo (unique across entire simulation).
1	desc_scale	#Desc_Scale: Scale of descendant halo, if applicable.
2	desc_id	#Descid: ID of descendant halo, if applicable.
3	num_prog	#Num_prog: Number of progenitors.
4	pid	#Pid: ID of least massive host halo (-1 if distinct halo).
5	upid	#Upid: ID of most massive host halo (different from Pid when the halo is within two or more larger halos).
6	desc_pid	#Desc_pid: Pid of descendant halo (if applicable).
7	phantom	#Phantom: Nonzero for halos interpolated across timesteps.
8	sam_mvir	#SAM_Mvir: Halo mass, smoothed across accretion history; always greater than sum of halo masses of contributing progenitors (Msun/h). Only for use with select semi-analytic models.
9	mvir	#Mvir: Halo mass (Msun/h).
10	rvir	#Rvir: Halo radius (kpc/h comoving).
11	rs	#Rs: Scale radius (kpc/h comoving).
12	vrms	#Vrms: Velocity dispersion (km/s physical).
13	mmp?	#mmp?: whether the halo is the most massive progenitor or not.
14	scale_of_last_MM	#scale_of_last_MM: scale factor of the last major merger (Mass ratio > 0.3).
15	vmax	#Vmax: Maxmimum circular velocity (km/s physical).
16	x	#X/Y/Z: Halo position (Mpc/h comoving).
17	y	
18	z	
19	vx	#VX/VY/VZ: Halo velocity (km/s physical).
20	vy	
21	vz	
22	Jx	#JX/JY/JZ: Halo angular momenta ((Msun/h) * (Mpc/h) * km/s (physical)).
23	Jy	
24	Jz	
25	Spin	#Spin: Halo spin parameter.
26	Breadth_first_ID	#Breadth_first_ID: breadth-first ordering of halos within a tree.
27	Depth_first_ID	#Depth_first_ID: depth-first ordering of halos within a tree.
28	Tree_root_ID	#Tree_root_ID: ID of the halo at the last timestep in the tree.
29	Orig_halo_ID	#Orig_halo_ID: Original halo ID from halo finder.
30	Snap_num	#Snap_num: Snapshot number from which halo originated.
31	Next_coprogenitor_depthfirst_ID	#Next_coprogenitor_depthfirst_ID: Depthfirst ID of next coprogenitor.
32	Last_progenitor_depthfirst_ID	#Last_progenitor_depthfirst_ID: Depthfirst ID of last progenitor.
33	Rs_Klypin	#Rs_Klypin: Scale radius determined using Vmax and Mvir (see Rockstar paper)
34	Mvir_all	#Mvir_all: Mass enclosed within the specified overdensity, including unbound particles (Msun/h)
35	M200b	#M200b--M2500c: Mass enclosed within specified overdensities (Msun/h)
36	M200c	
37	M500c	
38	M2500c	
39	Xoff	#Xoff: Offset of density peak from average particle position (kpc/h comoving)
40	Voff	#Voff: Offset of density peak from average particle velocity (km/s physical)
41	Spin_Bullock	#Spin_Bullock: Bullock spin parameter (J/(sqrt(2)*GMVR))
42	b_to_a	#b_to_a, c_to_a: Ratio of second and third largest shape ellipsoid axes (B and C) to largest shape ellipsoid axis (A) (dimensionless).
43	c_to_a	
44	A[x]	#A[x],A[y],A[z]: Largest shape ellipsoid axis (kpc/h comoving)
45	A[y]	
46	A[z]	
47	b_to_a (500c)	# (500c) indicates that only particles within R500c are considered.
48	c_to_a (500c)	
49	A[x] (500c)	
50	A[y] (500c)	
51	A[z] (500c)	
52	T/|U|	#T/|U|: ratio of kinetic to potential energies
53	M_pe_Behroozi	#M_pe_*: Pseudo-evolution corrected masses (very experimental)
54	M_pe_Diemer	
55	Macc	#Macc,Vacc: Mass and Vmax at accretion.
56	Mpeak	#Mpeak_Scale: Scale at which Mpeak was reached.
57	Vacc	
58	Vpeak	#Mpeak,Vpeak: Peak mass and Vmax over mass accretion history.
59	Halfmass_Scale	#Halfmass_Scale: Scale factor at which the MMP reaches 0.5*Mpeak.
60	Acc_Rate_Inst	#Acc_Rate_*: Halo mass accretion rates in Msun/h/yr.
61	Acc_Rate_100Myr	# Inst: instantaneous; 100Myr: averaged over past 100Myr,
62	Acc_Rate_1*Tdyn	# X*Tdyn: averaged over past X*virial dynamical time.
63	Acc_Rate_2*Tdyn	
64	Acc_Rate_Mpeak	
65	Mpeak_Scale	# Mpeak: Growth Rate of Mpeak, averaged from current z to z+0.5
66	Acc_Scale	#Acc_Scale: Scale at which satellites were (last) accreted.
67	First_Acc_Scale	#First_Acc_Scale: Scale at which current and former satellites first passed through a larger halo.
68	First_Acc_Mvir	#First_Acc_(Mvir|Vmax): Mvir and Vmax at First_Acc_Scale.
69	First_Acc_Vmax	
70	Vmax@Mpeak	#Vmax@Mpeak: Halo Vmax at the scale at which Mpeak was reached.

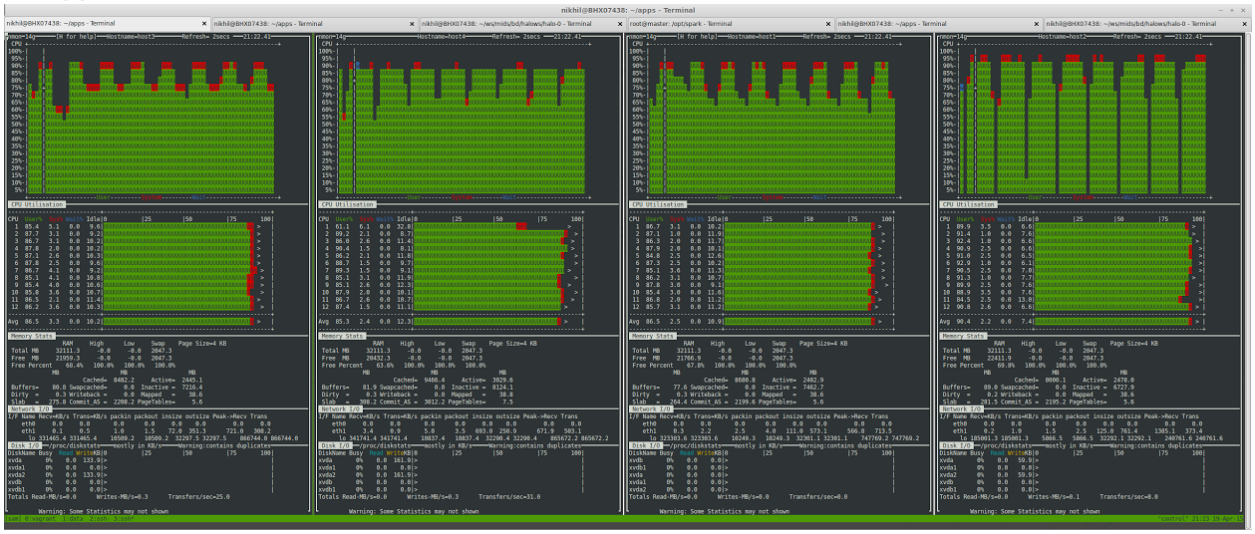
2 - Preprocessing



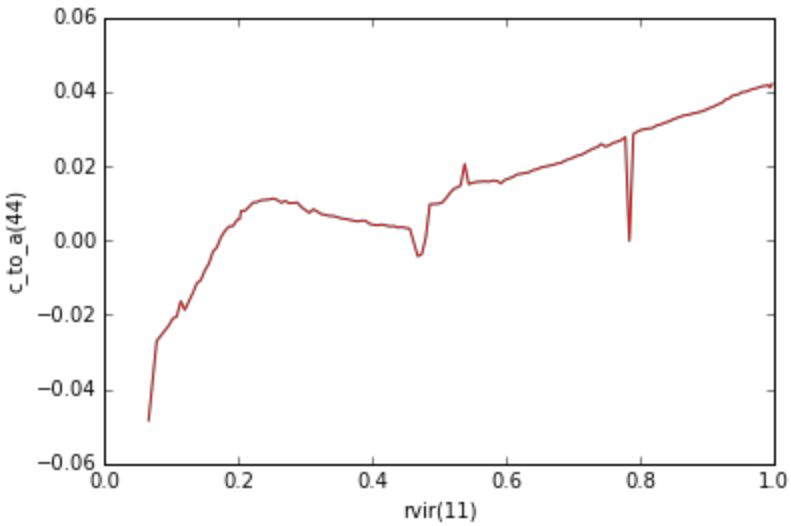
3 - Cosmological Scale Factor



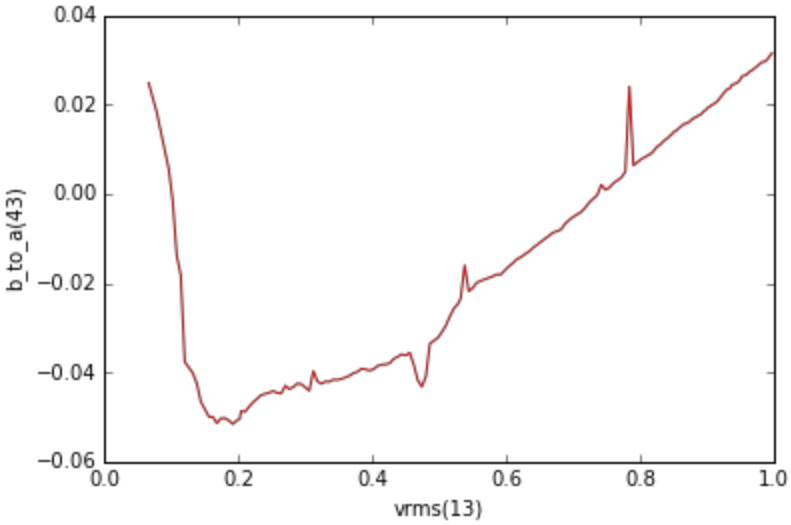
4 - Approach 2 Bottleneck



4 - Sample Correlations

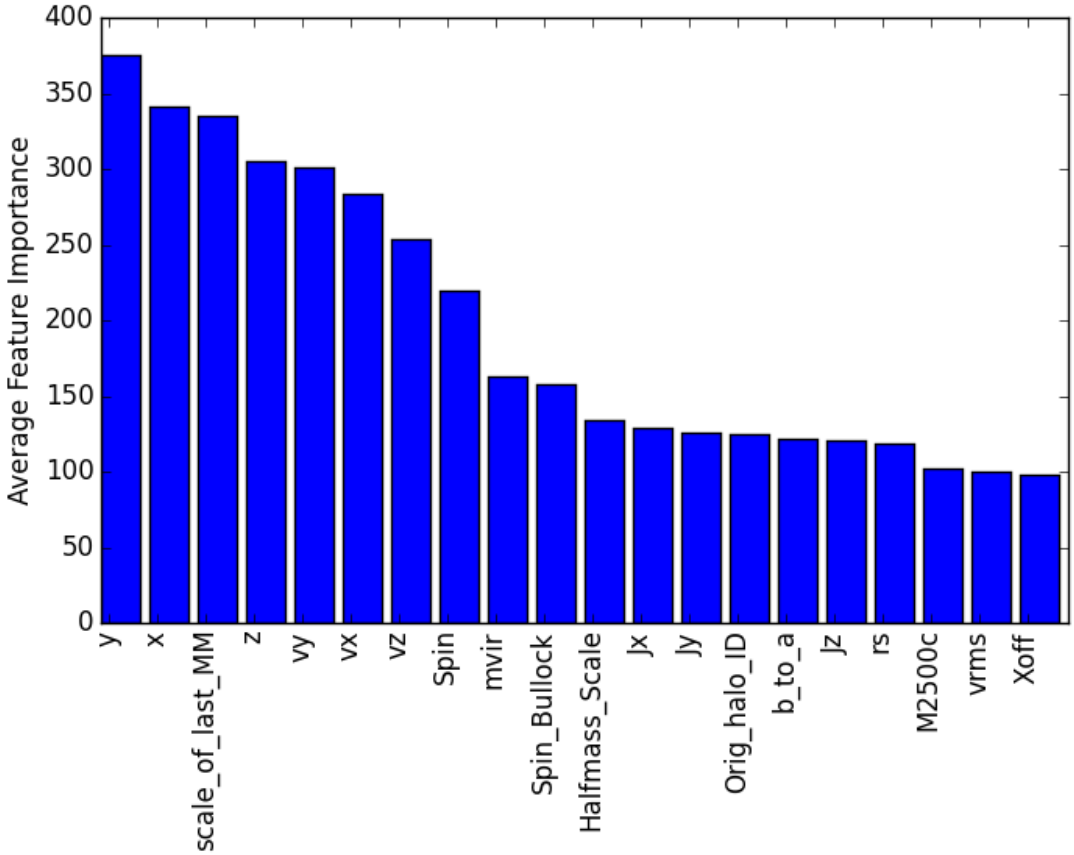


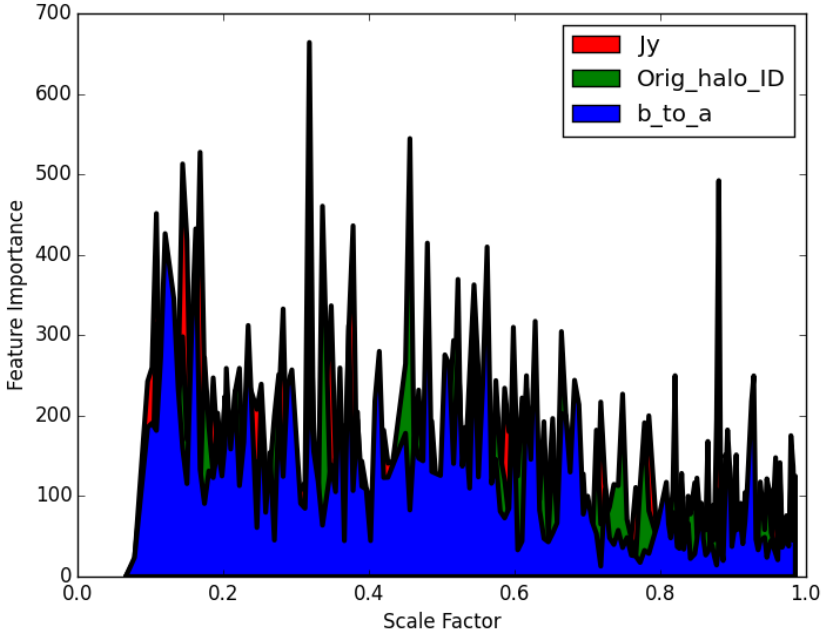
Correlation between Halo radius and Halo shape

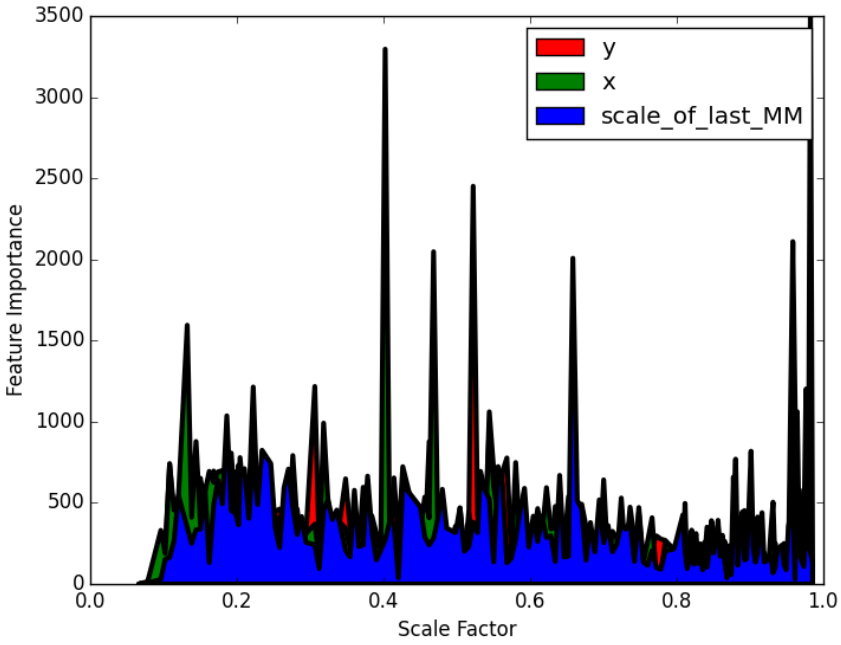


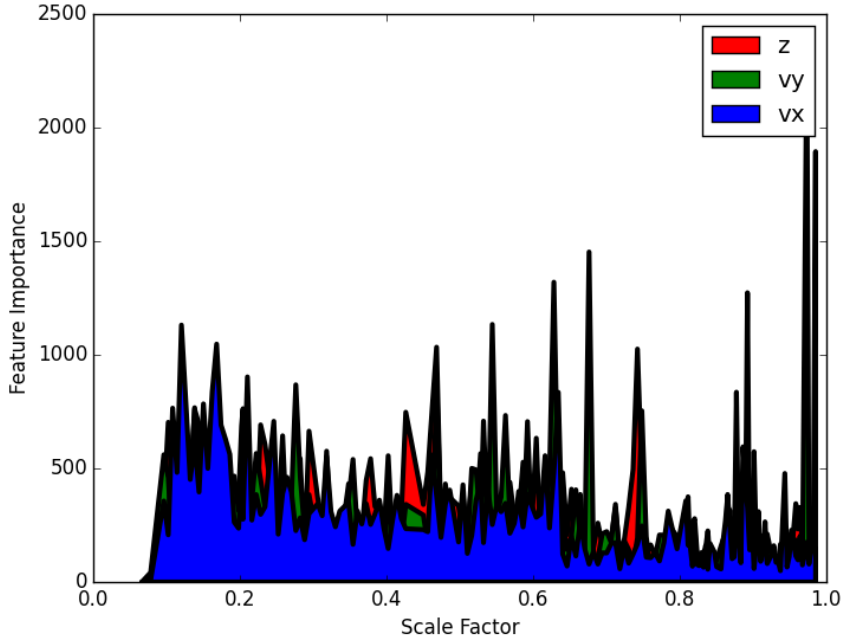
Correlation between velocity of dispersion and Halo shape

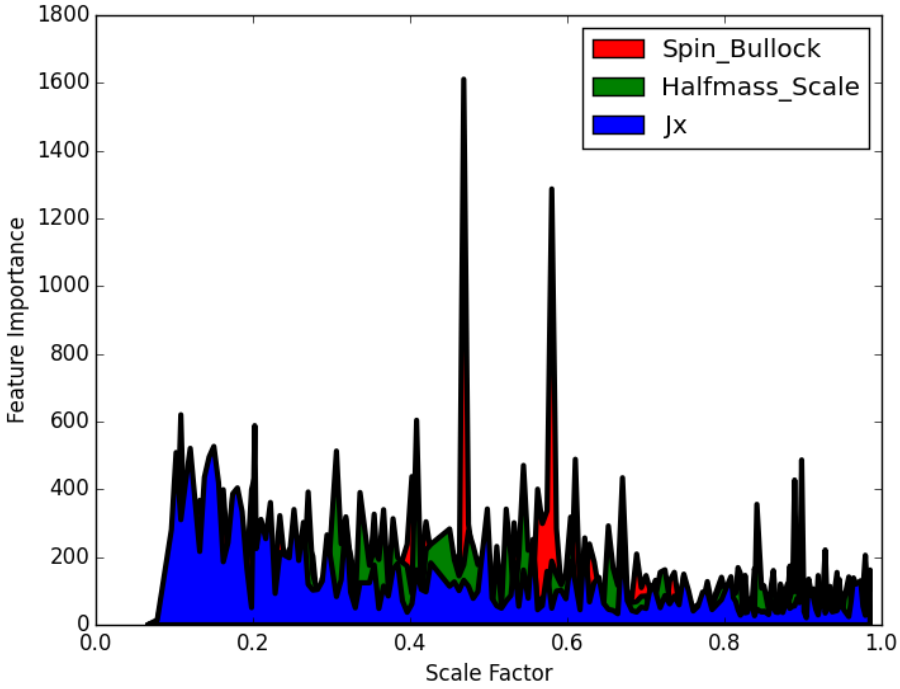
5 - Feature Importance

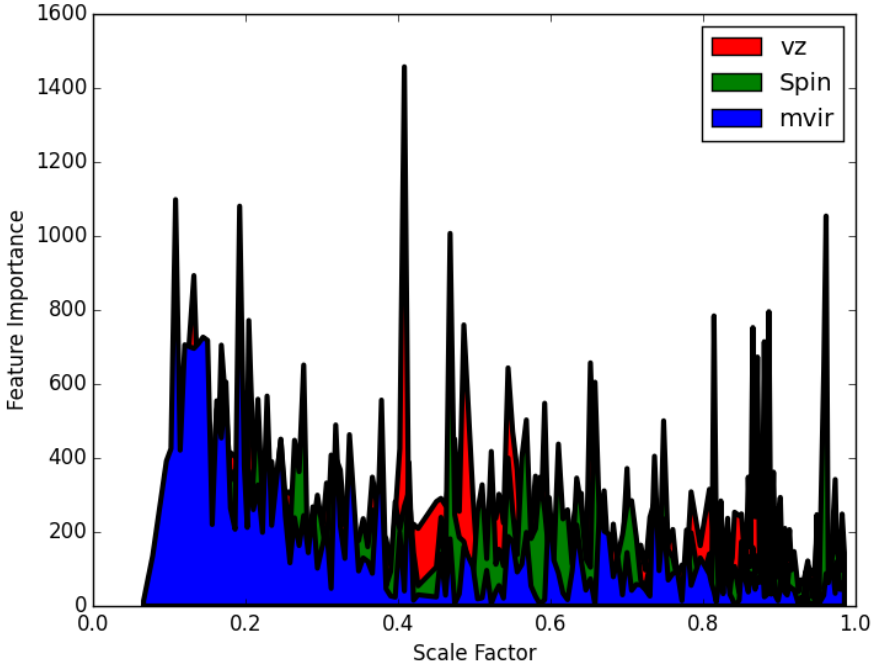



Jy
Orig_halo_ID
b_to_a
Feature Importance
Scale Factor
0
100
200
300
400
500
600
700
0.0
0.2
0.4
0.6
0.8
1.0


y
x
scale_of_last_MM
Feature Importance
Scale Factor
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0


2500
2000
1500
1000
500
0
Feature Importance
0.0
0.2
0.4
0.6
0.8
1.0
Scale Factor
z
vy
vx


Spin_Bullock
Halfmass_Scale
Jx
Feature Importance
Scale Factor
0
200
400
600
800
1000
1200
1400
1600
1800
0.0
0.2
0.4
0.6
0.8
1.0


vz
Spin
mvir
Feature Importance
Scale Factor
0
200
400
600
800
1000
1200
1400
1600
0.0
0.2
0.4
0.6
0.8
1.0