Classification and Feature Selection for High Energy Particle Detection

Matthew Resnick
Matthew.Resnick@colorado.edu
SID: 109570948
CSCI 5502

Taylor Egner
Taylor.Egner@colorado.edu
SID: 830353139
CSCI 5502

***Abstract*—Just a few years ago, the Higgs boson became the third confirmed particle theorized by the Standard Model post-inception, a discovery which was largely dominated by data analytics. A major challenge in the study of this and other high energy particles is detecting them among the vast amounts of data that come from the particle colliders in which the experiments take place. In this paper we use simulated collider data to evaluate several methodologies for generating robust classifiers that can be used to detect the Higgs particle from a combination of raw sensor outputs and engineered features. We looked at neural networks, logistic regression, support vector machines, and other classifiers and ultimately determined that neural networks showed the strongest performance. We also applied novel feature selection techniques to cull unnecessary data points, increasing the computational efficiency of our classification models while maintaining a sufficient level of accuracy.**

I. Introduction

In 2012, it was announced that detection of the Higgs boson had been confirmed by multiple international high energy physics projects. The Higgs boson then became one of just three elementary particles to be added to the Standard Model since it's consolidation in the 1970's (the other two being the top quark and the tau neutrino). The confirmation was a necessary step in solidifying the Higgs mechanism as the primary mass generation explanation which is compatible with the Standard Model.

The experiments that yielded the data from which the Higgs was confirmed used particle colliders, which by their nature must yield a mess of information. This "mess" includes both crucial, insightful information about fundamental reality, but also a sea of noise with no discernable patterns or information whatsoever. Thus, a primary problem in confirming the existence of a new elementary particle is extracting a signal of interest from this noise.

Our work is intended to be a survey of both standard and non-standard approaches to classification for high energy particle collision data as well as an exploration of potential feature selection methods which could be implemented to improve the performance of classifiers.

The data set we to used for our analysis is simulated collider data (via Monte Carlo) which emulates the data yielded by the LHC that preceded the Higgs confirmation. A majority of the features are simulated physical measurements that a collider's detectors might record, and the remaining features are functions of these first features, engineered by high energy particle physicists.

Our challenge was to determine the best techniques and models to generate an accurate classifier using such a large and intricate data set. There has been some activity in this space, with several published articles and a data science competition hosted by Kaggle that used a similar data set. However, it is difficult to determine what approaches can be refined and expanded upon, and which have fundamental limitations - and this was a main focus of our research. Perhaps a greater challenge, given our general lack of domain knowledge, was to understand ways in which we could make meaningful modifications to existing methods to improve performance.

There are two areas where we might be able to make novel contributions: an improved model for the signal classification task and/or a reduced amount of features required to achieve significant performance for such a model.

II. Related Work

Following the confirmation of the Higgs discovery, there was significant interest in this topic and countless papers in multiple fields of research have been published on the matter. The dataset we are using in particular was generated by Baldi, et al [1] out of UC Irvine as part of a study of the performance of several machine learning techniques including boosted decision trees as well as shallow and deep neural networks. Significantly, they found that deep neural networks not only showed the best performance, but also suffered no loss in performance when the engineered features were removed, suggesting that the neural network was implicitly incorporating the information that the engineered features were designed to provide.

There was a major machine learning challenge hosted on Kaggle in 2014 using a similar, though considerably smaller dataset. An analysis of the highest performing solution [9], which also happened to use neural networks to build a predictive model, suggested that there were several features which could potentially be removed without harming performance, though the results on this were inconclusive. There were, however, many practical pre-processing techniques that enhanced the performance of the model. Notably, many of the solutions proposed in the challenge were optimized for a much smaller (250,000 records) dataset, so there may be room for optimizations even in the top performing solutions.

Feature selection, widely recognized as an NP-hard problem has long been an active field of study in the literature, and can be categorized into two subsets – the filter model and the wrapper model [6]. Filter models are easier to implement, as they rely on the properties of the dataset to remove features, rather than on a learning algorithm. Wrapper methods, however, promise better results at the cost of computational complexity.

III. Evaluations

The evaluation of a model we have implemented will depend largely on the portion of the project the model is being employed within. For the first major portion, that of binary classification, the evaluation will consist of specific metrics, consistent across all models, and of performance charts unique to each model. For the metrics we will use simple accuracy and area under the Receiver Operating Characteristic curve (AUC). The performance charts will consist of iteration-specific cost function and error values, and occasionally overall training time. For Feature Selection models, we will again use accuracy and AUC, however will will also compare these with the number of features selected and, in some cases, values for tune-able parameters.

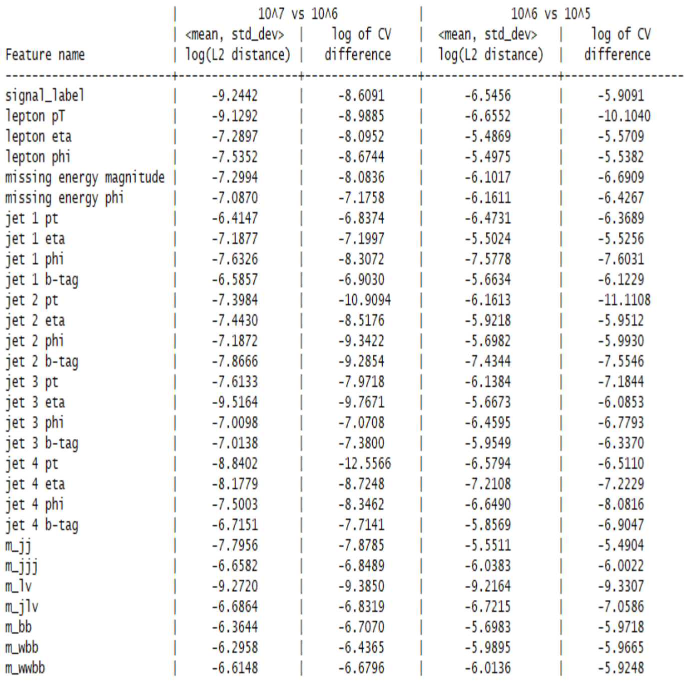
IV. Data Pre-Processing and Pre-Analysis

Since the Higgs data set is both simulated data and has been the subject of numerous papers and data science competitions, it is already relatively clean from the source. Thus, our main concern with pre-processing was largely in the realm of compression and transformation for particular models.

We began the pre-processing with compression. Since the data set is relatively large as previously discussed, it would be cumbersome to load even relatively small sections of it into memory every time we needed to us it. Thus, we used the H5Py package to compress the data into an hdf5 file via chunks of 500,000 instances.

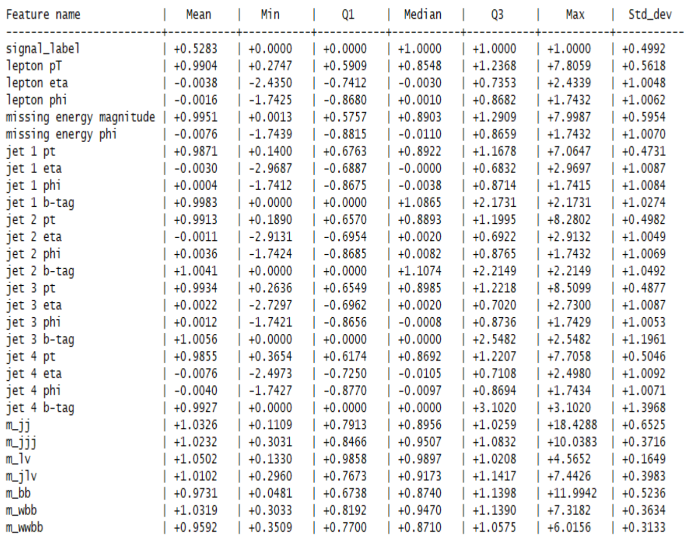
We also applied two types of transformation on the data. The first, z-score normalization, was ultimately only applied to subsets of the data as it was needed for particular models, as some models performed better without normalization. The second transformation, whereby the high-level features were converted to log-space, was cut entirely from our pipeline. Some features had a great deal of skew in their distribution densities, so we considered using a log-transform to allow more information to be gleaned from them. The winners of the 2014 Kaggle competition for classification using this data set mentioned in their paper that they had done a log-transform on features with long tails as well, so we decided it might be a well-informed decision to use it too. However, only the high-level features were viable for this type of transformation, as they all represent some mass and so are not negative or zero. When we compared some models using both the transformed and not-transformed data, they performed either the same or slightly worse (Logistic Regression lost 0.01 in AUC and gradient boosted decision trees had no change). In order to save some computation time, we removed the transformation entirely.

Another attempt to reduce computation time led us to the discovery that using successively larger subsets of the data for model training/analysis yielded next to no improvements. Many of our models appeared to not improve by any metric when the training subsets were made larger, and so we investigated the distribution of the features at various scales and found them to be nearly identical, about 100,000 instances. The table below shows a comparison of $<$ mean, stddev $>$ vectors by log of L2 distance as well as the log of the difference in coefficients of variation between a big subset of the data (10 million instances) and a medium subset (1 million instances) and between the medium subset and a small subset (100,000 instances).



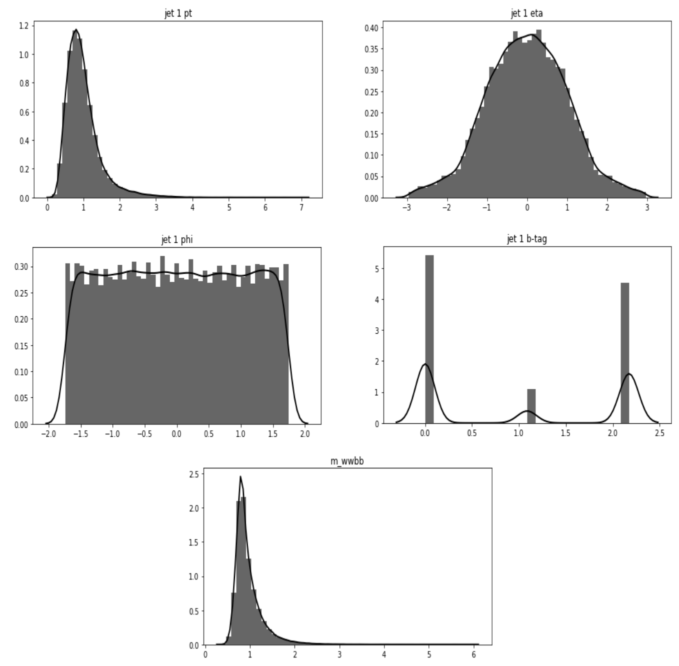
	10^7 vs 10^6		10^6 vs 10^5	
Feature name	<mean, std_dev> log(L2 distance)	log of CV difference	<mean, std_dev> log(L2 distance)	log of CV difference
signal_label	-9.2442	-8.6091	-6.5456	-5.9091
lepton pT	-9.1292	-8.9885	-6.6552	-10.1040
lepton eta	-7.2897	-8.0952	-5.4869	-5.5709
lepton phi	-7.5352	-8.6744	-5.4975	-5.5382
missing energy magnitude	-7.2994	-8.0836	-6.1017	-6.6909
missing energy phi	-7.0870	-7.1758	-6.1611	-6.4267
jet 1 pt	-6.4147	-6.8374	-6.4731	-6.3689
jet 1 eta	-7.1877	-7.1997	-5.5024	-5.5256
jet 1 phi	-7.6326	-8.3072	-7.5778	-7.6031
jet 1 b-tag	-6.5857	-6.9030	-5.6634	-6.1229
jet 2 pt	-7.3984	-10.9094	-6.1613	-11.1108
jet 2 eta	-7.4430	-8.5176	-5.9218	-5.9512
jet 2 phi	-7.1872	-9.3422	-5.6982	-5.9930
jet 2 b-tag	-7.8666	-9.2854	-7.4344	-7.5546
jet 3 pt	-7.6133	-7.9718	-6.1384	-7.1844
jet 3 eta	-9.5164	-9.7671	-5.6673	-6.0853
jet 3 phi	-7.0098	-7.0708	-6.4595	-6.7793
jet 3 b-tag	-7.0138	-7.3800	-5.9549	-6.3370
jet 4 pt	-8.8402	-12.5566	-6.5794	-6.5110
jet 4 eta	-8.1779	-8.7248	-7.2108	-7.2229
jet 4 phi	-7.5003	-8.3462	-6.6490	-8.0816
jet 4 b-tag	-6.7151	-7.7141	-5.8569	-6.9047
m_jj	-7.7956	-7.8785	-5.5511	-5.4904
m_jjj	-6.6582	-6.8489	-6.0383	-6.0022
m_lv	-9.2720	-9.3850	-9.2164	-9.3307
m_jlv	-6.6864	-6.8319	-6.7215	-7.0586
m_bb	-6.3644	-6.7070	-5.6983	-5.9718
m_wbb	-6.2958	-6.4365	-5.9895	-5.9665
m_wwbb	-6.6148	-6.6796	-6.0136	-5.9248

It can be seen that these values are sufficiently small, and thus it can be reasonably assumed that, likely due to the simulated nature of the data, distributions of features will look nearly identical at every scale above 100,000. Comparing some classification models even showed that 10,000 instances was often similar, however we could usually gain 0.01 – 0.05 AUC at 100,000 and so made that the standard subset for "fast" evaluation. After processing the data as much as was necessary for performance and efficiency, we turned to gaining an understanding of the make-up of the features themselves. The following table shows statistical summaries for all 28 features:



```
Feature name              |  Mean    |  Min     |  Q1      |  Median  |  Q3      |  Max      |  Std_dev
--------------------------+----------+----------+----------+----------+----------+-----------+----------
signal_label              | +0.5283  | +0.0000  | +0.0000  | +1.0000  | +1.0000  | +1.0000   | +0.4992
lepton pT                 | +0.9904  | +0.2747  | +0.5909  | +0.8548  | +1.2368  | +7.8059   | +0.5618
lepton eta                | -0.0038  | -2.4350  | -0.7412  | -0.0030  | +0.7353  | +2.4339   | +1.0048
lepton phi                | -0.0016  | -1.7425  | -0.8680  | +0.0010  | +0.8682  | +1.7432   | +1.0062
missing energy magnitude  | +0.9951  | +0.0013  | +0.5757  | +0.8903  | +1.2909  | +7.9987   | +0.5954
missing energy phi        | -0.0076  | -1.7439  | -0.8815  | -0.0110  | +0.8659  | +1.7432   | +1.0070
jet 1 pt                  | +0.9871  | +0.1400  | +0.6763  | +0.8922  | +1.1678  | +7.0647   | +0.4731
jet 1 eta                 | -0.0030  | -2.9687  | -0.6887  | -0.0000  | +0.6832  | +2.9697   | +1.0087
jet 1 phi                 | +0.0004  | -1.7412  | -0.8675  | -0.0038  | +0.8714  | +1.7415   | +1.0084
jet 1 b-tag               | +0.9983  | +0.0000  | +0.0000  | +1.0865  | +2.1731  | +2.1731   | +1.0274
jet 2 pt                  | +0.9913  | +0.1890  | +0.6570  | +0.8893  | +1.1995  | +8.2802   | +0.4982
jet 2 eta                 | -0.0011  | -2.9131  | -0.6954  | +0.0020  | +0.6922  | +2.9132   | +1.0049
jet 2 phi                 | +0.0036  | -1.7424  | -0.8685  | +0.0082  | +0.8765  | +1.7432   | +1.0069
jet 2 b-tag               | +1.0041  | +0.0000  | +0.0000  | +1.1074  | +2.2149  | +2.2149   | +1.0492
jet 3 pt                  | +0.9934  | +0.2636  | +0.6549  | +0.8985  | +1.2218  | +8.5099   | +0.4877
jet 3 eta                 | +0.0022  | -2.7297  | -0.6962  | +0.0020  | +0.7020  | +2.7300   | +1.0087
jet 3 phi                 | +0.0012  | -1.7421  | -0.8656  | -0.0008  | +0.8736  | +1.7429   | +1.0053
jet 3 b-tag               | +1.0056  | +0.0000  | +0.0000  | +0.0000  | +2.5482  | +2.5482   | +1.1961
jet 4 pt                  | +0.9855  | +0.3654  | +0.6174  | +0.8692  | +1.2207  | +7.7058   | +0.5046
jet 4 eta                 | -0.0076  | -2.4973  | -0.7250  | -0.0105  | +0.7108  | +2.4980   | +1.0092
jet 4 phi                 | -0.0040  | -1.7427  | -0.8770  | -0.0097  | +0.8694  | +1.7434   | +1.0071
jet 4 b-tag               | +0.9927  | +0.0000  | +0.0000  | +0.0000  | +3.1020  | +3.1020   | +1.3968
m_jj                      | +1.0326  | +0.1109  | +0.7913  | +0.8956  | +1.0259  | +18.4288  | +0.6525
m_jjj                     | +1.0232  | +0.3031  | +0.8466  | +0.9507  | +1.0832  | +10.0383  | +0.3716
m_lv                      | +1.0502  | +0.1330  | +0.9858  | +0.9897  | +1.0208  | +4.5652   | +0.1649
m_jlv                     | +1.0102  | +0.2960  | +0.7673  | +0.9173  | +1.1417  | +7.4426   | +0.3983
m_bb                      | +0.9731  | +0.0481  | +0.6738  | +0.8740  | +1.1398  | +11.9942  | +0.5236
m_wbb                     | +1.0319  | +0.3033  | +0.8192  | +0.9470  | +1.1390  | +7.3182   | +0.3634
m_wwbb                    | +0.9592  | +0.3509  | +0.7700  | +0.8710  | +1.0575  | +6.0156   | +0.3133
```

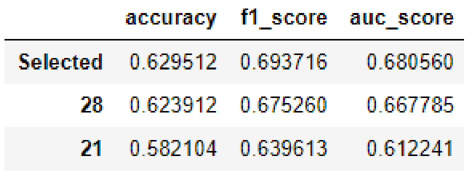
Looking closely at the low-level features reveals some -perhaps predictable- trends. All phi features tended to have a similarly uniform distribution, which makes sense because of the fact that they are all associated with relatively small angles which, at the speeds these events are occurring at, are ultimately irrelevant. The winners of the 2014 Kaggle competition even said that they began their pre-processing by directly removing these features from the data. All eta features were also quite similar and not terribly useful because of their nearly perfectly standard-normal distribution. pT features and b-tag values tended to also be similar across all jets, however they had slightly more interesting distributions. Most of the high-level features had interesting distributions, but generally were all skewed right heavily. We show the distribution densities for some common feature distributions below:



Since these data are based on real physical values, we need not check for dependence or correlation. We know the low-level features are all independent of one another, and we know that since the high-level features are functions of the low-level features, they are all dependent on the low-level features. This makes pre-training on the different feature sets viable, as well as the possibility of applying Bayesian models to the low-level features. However, the original paper which produced this data set demonstrated that some combination of high- and low-level features would likely be necessary for best predictive performance. We decided to leave such questions to the feature selection portion of the project.

V. Classification: Neural Networks

There has been much excitement in the literature about the power of neural networks to build strong classifiers for this data set and similar data sets generating using Monte Carlo simulation. The winners of the 2014 Kaggle competition made use of an ensemble learning technique using deep neural networks [9], an approach from which our study was derived. Furthermore, the team that generated the data set we used for our experiments were able to show that deep neural networks, in particular, could perhaps have the very interesting property of being able to 'infer' the information contained in the engineered features even when they were excluded from the data [1]. We were unfortunately unable to replicate this finding in our study.
Neural networks fall into two major categories: single-layer networks, and multi-layer or "deep" networks. The difference between the two, as may be inferred from the nomenclature, is in the number of hidden layers included. For our study, we wanted to take a comparative look at these methods. Our single-layer perceptron (SLP) had a single layer of 600 neurons. For our study of deep networks, we designed our MLPs with 3 fully-connected layers of 300 neurons each. A comparison of the performance of these networks is given in Figure V-B. For our final analysis, we used an 80/20 split of the data (consisting of 11 million rows) for our training and test sets. The results of our study of single networks broken down by feature inclusion are summarized in Figure V.



	accuracy	f1_score	auc_score
Selected	0.629512	0.693716	0.680560
28	0.623912	0.675260	0.667785
21	0.582104	0.639613	0.612241

Fig. 1. MLP Network Performance by Features Included

A. Activation Functions

The most common activation functions for neural networks are the Sigmoid function and the Rectified Linear Unit (ReLU) function. However, the winners of the Kaggle competition to build a classifier for a similar dataset [9] used a function inspired by biological neural networks called the Local-Winner-Take-All (LWTA) activation function as described originally in Srivastava, et al [13]. Figure V-A gives a visual comparison of different activation functions.

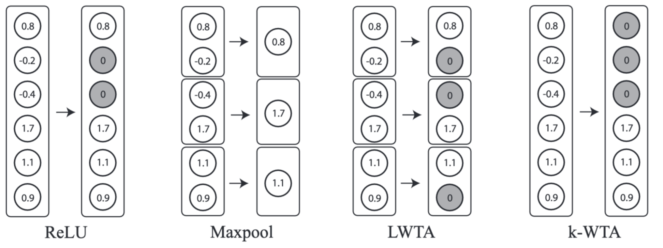


Fig. 2. A comparison of various activation functions. Reprinted from [14]

We decided to implement this LWTA function in our model to evaluate its effectiveness.

B. Performance

Our results show that deep networks do perform better than single layer networks for this classification task, however, they are much more computationally expensive. Figure V-B shows the results of a time study of the training time of the three networks on a sample set of 10,000 rows. It illustrates the large difference in expense between the network types, and the variability in computation time for the LWTA activation function. The network that made use of novel Local-Winner-Take-All [13] activation function did not show increased performance over the network using the more standard ReLu activation.

	LWTA	SLP	MLP
count	30.000000	30.000000	30.000000
mean	0.390269	0.033860	0.117612
std	0.016567	0.001225	0.007809
min	0.357843	0.031514	0.105658
25%	0.380833	0.033063	0.113466
50%	0.389166	0.033955	0.115991
75%	0.400943	0.034605	0.117552
max	0.423169	0.036892	0.139268

Fig. 3. Results of a time study of the three architecture types

Model	Accuracy	AUC	F1
LWTA	0.7336	0.8123	0.7502
MLP	0.7382	0.8184	0.7534
SLP	0.7155	0.7888	0.7391

Fig. 4. A comparison of performance by network architecture

C. Bagging

A common enhancement technique in machine learning is ensemble learning, which is a class of techniques that make use of several trained models to enhance the accuracy of the classifier. For each of our model archetypes, we performed ensemble learning using the Bagging [3] method in which m models are trained using a subset n' of the data set that is drawn uniformly at random with replacement.
Figure 5 illustrates how the performance of the classifier is enhanced by the bagging method. We built our ensemble of models using 25 networks each trained on a subset equal to the size of the training set, but this appears to have been overkill.

Score	MLP (Single)	MLP (Bagged)
Accuracy	0.7382	0.7407
AUC	0.8184	0.8214
F1 Score	0.7534	0.7557

Fig. 5. A comparison of Bagged vs Single Network Performance

As Figure 6 illustrates, most of the performance increase of the bagging method is achieved after only a few networks are included.

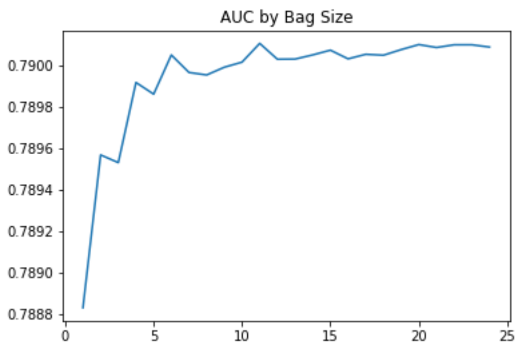


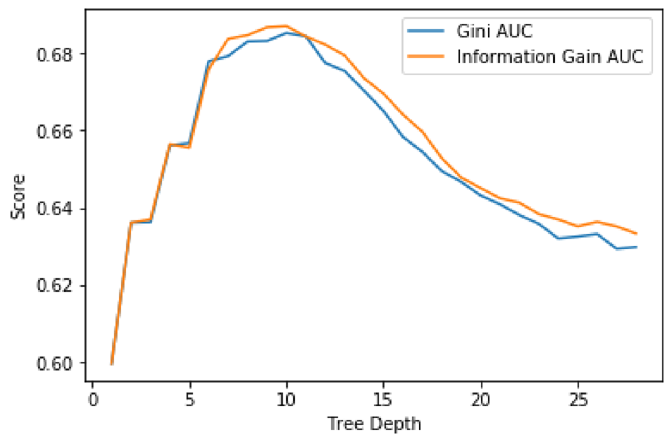
Fig. 6. AUC Score of Bagged SLP by Bag Size. Average over 30 Trials.

VI. Classification: Non-Standard Approaches

Since many binary class prediction methods in high energy physics as a whole, let alone simulated Higgs decay events, are founded primarily on a bedrock of fully connected neural networks and/or boosted decision trees, we consider several alternatives which either depart partially or wholly from these standard approaches. Borrowing from the conventional wisdom of the preceding work in this field, we generally expect ensemble methods to be the most powerful predictors, due to the size of the data set and the intricate nature of the physical measurements which form the features. However, in order to gauge how these ensembles will be formulated, we begin by comparing a few single-model approaches which will ultimately serve as the underlying weak classifiers. Ensemble methods are not proven to improve performance, so in any case we are interested to see how these classifiers perform for their own merit.

A. Histogram Classifier

Perhaps one of the simplest classifiers we could employ is the histogram classifier. This model is essentially a decision tree with one level, whereby a classification decision is made by comparing a single feature to a threshold. So naturally we did not expect it to outperform full decision trees, let alone boosted trees or forests. Initial tests seemed to fit our expectations, yielding an AUC slightly above 0.6, from both Gini impurity and Shannon information gain as the splitting criterion. While we were at it, we demonstrate performance for a number of decision trees with variable depth and splitting criterion:



B. *Logistic Regression*

Since our data are comprised of features which are functions of other features, we cannot assume class conditional independence, and therefore cannot use a Naïve Bayes classifier to make class predictions. We could, of course, consider a prediction class with these dependent features being separated into groups of independent features, but we will return to this idea in the discussion of our feature selection approaches. So, our first implementation of a generalized linear model will be similar to Naïve Bayes, but without being tethered to an independence assumption: logistic regression.

We use cross entropy for the loss function and an improved Stochastic Average Gradient (SAGA) as the optimization algorithm [7]. While this classifier alone achieves an expected underwhelming value for the performance metric, with an AUC of around 0.64, it is high enough to be promising as a weak classifier in an ensemble, as well as indicating that logistic regression may serve well for minimizing loss in the ensemble itself, even with a different weak classifier.

C. *Support Vector Machines*

Another often powerful classifier is one which utilizes kernel functions to map the data to higher dimensions in order to find a hyperplane which separates the data by class [5]. These Support Vector Machines (SVMs), however, pose a critical challenge: they are very computationally expensive on their own. We were able to obtain results on our data with SVMs, but only if we reduced the training set to around 10,000 samples (just 1/10th of a percent of our entire data set), from which we yielded and AUC of 0.63, which certainly is a promising result given that it is at least significantly above random guessing. A training set of just 1% of our data did not finish training after 3 hours, so more proverbial firepower will be needed for this method.

Dom et. Al. suggest that while boosting may not improve the accuracy of a linear SVM —the reason for which will be discussed in the section of boosting logistic regression— it can potentially allow it to be used with large-scale data sets [10]. They propose that by using boosting and probabilistic SVMs, it may be possible to significantly reduce the computation time for this method. We will revisit our work with this concept later in this section.

D. *Histogram Classifier + LogitBoost*

In order to improve the histogram classifier's performance, the first ensemble method we implement is LogitBoost. This boosting method iteratively updates a stronger learner by optimizing logistic loss over weak learner performance, effectively weighting up weak classifiers which have difficulty in prediction to focus the learning more on these areas [8]. Since decision stumps and trees are considered "unstable" classifiers, and are specifically not linear models, we expected that there could be some improvement in performance from the lone histogram classifier. From a model we built from scratch, we yielded an AUC of about 0.55. Still better than random guessing, but a notable decrease in predictive performance from decision stumps alone.

Implementing a state-of-the-art model, pre-built in a Python package, we yielded an AUC of about 0.69. This not only shows the potential power of this model, but also shows a significant improvement from the base predictor. Interestingly, if we increase the depth of the base trees, the performance does not change. This is an important finding to note because, as previously mentioned, boosted decision trees are a widely used, standard method, and yet a significant amount of computation time could be saved by building stumps instead of full trees.

E. *Logistic Regression + AdaBoost*

Since boosted histogram classifiers didn't exceed our already low expectation for performance, we turn to a weak classifier that performed better on its own. A concern about boosting logistic regression, however, is that since it is a generalized linear model, there is no reason to expect an improvement in performance when boosted with an additive method. This is because adding linear models together simply results in a sum-of-parts linear model. That is, boosting logistic regression may only yield what is equivalent to a lone logistic regression model, and thus there may not be any improvement.

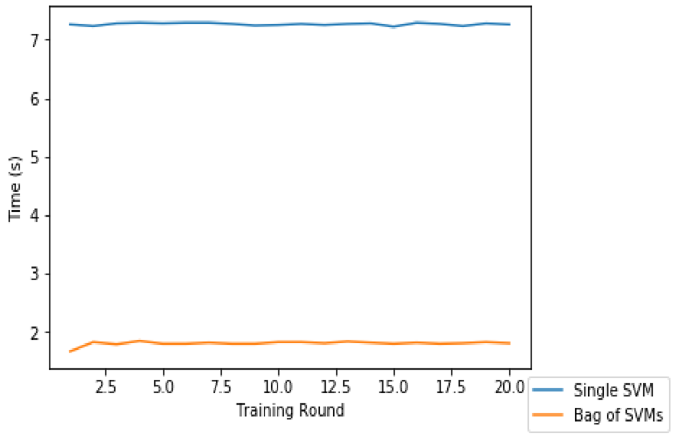
We wanted to test this approach anyway though, for the sake of being exhaustive and to see if an applied test surprises us with better results than would be predicted in theory. To do this we implemented AdaBoost, a similar model to LogitBoost, except that the weak learners must have discrete classification output, rather than being probabilistic predictors, and the function to be optimized is exponential rather than logistic [11].

The AUC from implementing this approach was 0.62, a slight decrease in performance. So, unless we can find a robust, non-additive boosting method, it seems that logistic regression works better for the loss function of a boosting method than as a weak learner.

F. SVMs + Bagging

We now return to the idea that an ensemble of support vector machines could potentially train in significantly faster time. The Boost-SMO method described by Dom et. Al. is constructed by the team meticulously building the ensemble classifier based on optimizing training data size per underlying SVM and monitoring memory usage. Yet, they did not yield better predictive performance, as they had expected initially. Thus, to build a simpler and less complicated model, we employ a bagging ensemble method instead of boosting. This method still trains the weak underlying classifiers on potentially distinct subsets of the data, however the ultimate classification decision is made by weighting the classifiers and taking a consensus [3].

After some testing, it seems that this model does work as it was designed to, and performs just as well as Dom et. Al. indicated it would, based on their boosting model. We yielded an AUC of 0.63 with a model trained on 10,000 instances, which is identical to our tests with a single SVM. However, the training time for a subset of this size for the single SVM was just over 7 seconds, on average, while the bagged model trained in just under 2.



G. Random Forests + Gradient Boosting

The last non-standard approach we employed combines two ensemble methods into one. A random forest is essentially a bag of decision trees, whereby the classification decision is made by the mode of the underlying trees [?]. Gradient boosting is a more generalized boosting method than LogitBoost and AdaBoost in that the loss function is arbitrary in the description of the model, rather than being fixed as logistic or exponential [4].

Using the XGBoost Python package, we used random forests as the weak classifier for gradient boosting. We worked the size of the forest up to 50 trees, and to a maximum depth of 10 for each tree (any more depth than this resulted in insurmountable computation time). This yielded a fairly high AUC of 0.72, however using just one tree of depth 10 as the underlying classifier yielded an AUC of 0.7 with significantly less computation time.

Model	AUC
Histogram Classifier/Decision Stumps	0.6
Logistic Regression	0.64
Support Vector Machines	0.63
Histogram Classifier + LogitBoost	0.55 - 0.69
Logistic Regression + AdaBoost	0.63
Support Vector Machines + Bagging	0.63
Random Forests + Gradient Boosting	0.73

VII. Feature Selection

While other data mining work in the area of high energy physics, and specifically in this data set, has employed techniques for feature selection, we were determined to explore some untested and undocumented areas. This especially because none of the main papers or projects associated with this data set mentioned any robust feature selection methods. The UCI Higgs data set itself was initially simulated with 21 features, and some of the very first work on it was to engineer new features which became the final seven "high-level" values which reflect other physical components of a decay event that were not directly simulated. Even with the full set of features, though, 28 is not a staggering amount by any means, but if we could develop a robust way of selecting only the relevant features from the Higgs set, it might be possible to extrapolate to other areas of high energy data mining. The implications of such a successful technique could be numerous: reductions in computation time for prediction and analysis, reduction in over-fitting in prediction models, and perhaps even a reduction in the amount of data which needs to be collected/simulated by physicists in the first place.

Before we began working towards some models which would cull features from the data, we first considered the feasibility and potential effectiveness of creating new features. The primary issue for our group was a lack of domain knowledge, and thus developing features based on an intricate understanding of the physics would prove to be insurmountable in the given time frame. However, such features might be the most information-rich, as they would be based on provable, testable mathematics of the underlying systems, rather than informed guesses. Due to the lack of success found by the winners of the 2014 Kaggle competition with engineering new features —where the team had extensive domain knowledge— we opted to instead use our limited time to only focus on techniques which would reduce the total number of features.

A. Thresholding Based Iterative Selection Procedure (TISP)

Since we intended to cull existing features from our models at all, it would be most concise to say that we assumed that not all input values were necessary to describe the classes we are attempting to predict. Thus, to expand on some of the models we had already implemented, we looked for a selection model which employed sparsity regularization, a concept which is based exactly on our assumption. We first look to TISP, developed by Dr. Yiyuan She of Florida State University [12]. This model allows us to employ the familiar and fairly powerful logistic loss function, except now with the addition of a penalization function which will, in effect, allow for the reduction in the number of input features which influence the prediction of the signal class.

Penalized loss function:

$$L(\mathrm{w}) = \frac{1}{N}\sum_{j=1}^{N} f(\mathrm{w}^T x_j, y_j) + \lambda \sum_{i=1}^{M} \rho(w_i)$$

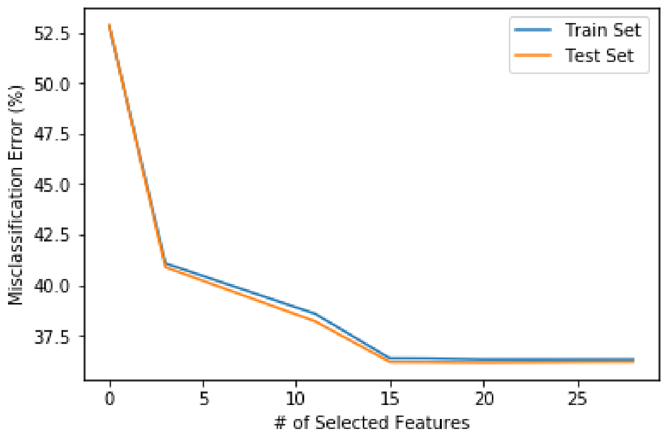
Weight update equation for time step t using logistic regression as the loss function:

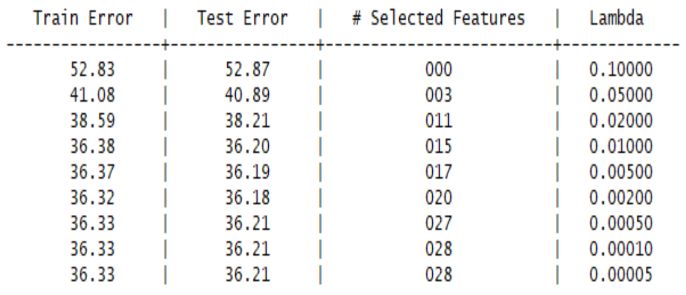
$$\mathrm{w}^{(t+1)} = \Theta\left(\mathrm{w}^{(t)} + \eta X^T \left[y - \frac{1}{1+\exp(-X\mathrm{w}^{(t)})}\right], \lambda\right)$$

Where f is the underlying loss function, $\rho(x)$ is the penalization function, Θ is the thresholding function, λ is the threshold, and η is the learning rate. [12].

The penalization function is rather modular in this model, and allows for the use of penalties such as L1, L2, hard thresholding, and Smoothly Clipped Absolute Deviation (SCAD), among others (however, many penalties besides these will require non-convex optimization). We chose to solely use hard thresholding, with the penalty function $\Theta(t;\lambda) = t/(1+\lambda)$. This function is simple, conceptually and in execution, while still allowing us control enough to fine-tune the parameters which determine the number of features culled.

The TISP algorithm is quite fast, and thus usually converged within a few iterations. We found 200 iterations to be fast enough to handle the maximum amount of training instances used for any of our models while not sacrificing performance by any metric. Tuning the lambda threshold proved to be somewhat more difficult, so we instead ran a variety of choices, the results of which can be seen in the below table/graph:

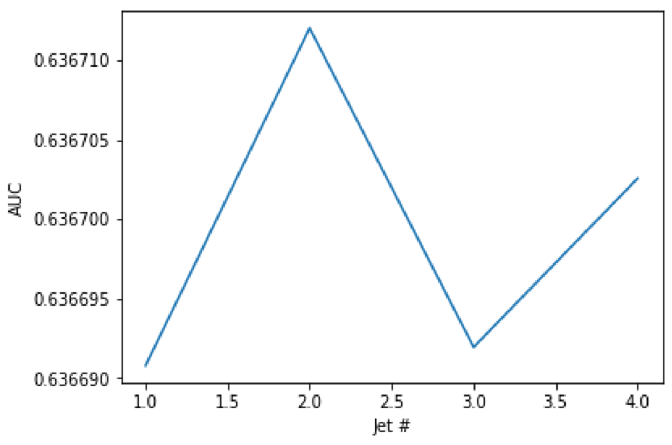




```
 Train Error   |  Test Error   |  # Selected Features   |  Lambda
---------------+---------------+------------------------+----------
    52.83      |    52.87      |          000           |  0.10000
    41.08      |    40.89      |          003           |  0.05000
    38.59      |    38.21      |          011           |  0.02000
    36.38      |    36.20      |          015           |  0.01000
    36.37      |    36.19      |          017           |  0.00500
    36.32      |    36.18      |          020           |  0.00200
    36.33      |    36.21      |          027           |  0.00050
    36.33      |    36.21      |          028           |  0.00010
    36.33      |    36.21      |          028           |  0.00005
```

So, somewhat surprising to us, a reduction in nearly half the number of original features leads to a nearly identical AUC in training validation and testing. What was especially interesting, though, was which features were eliminated (highlighted features were kept): lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pT, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pT, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pT, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pT, jet 4 eta, jet 4 phi, jet 4 b-tag, m jj, m jjj, m iv, m jiv, m bb, m wbb, m wwbb.

All eta and phi features were culled, which makes some sense in looking at how information-poor their distribution densities appeared to be. The surprising features that were cut were the one high-level feature and jet 1's b-tag. The former is surprising because the values are composed of multiple underlying low-level features, and the second is surprising because all of the other b-tag features were kept. We suspected that having any of the other three jet's b-tag information would allow a model to infer the value of the fourth, and some testing reveal that this is probably true. First, we saw that using only the 15 selected features to train the logistic regression, SVM, and gradient boosted tree model yielded very similar performance as with all 28 features. Then we again used logistic regression to show that removing any of the other four b-tag features and adding the first's back in would yield nearly identical performance, as shown below:



TISP allows for the use of many different underlying loss functions in addition to the already modular penalty function component, so not only did it work well for our purposes in this data set, but we also believe it may see more success in other areas of high energy knowledge discovery. To that end, we re-wrote our implementation of the model to be compatible with the SciKit-Learn Python framework, so that it can be easily wrapped up into a larger pipeline, including pre-written ensemble methods. We then tested the newly written TISP class with SciKit-Learn's AdaBoost, BaggingClassifier, and StackingClassifier. Naturally they all performed at the same pace as the base TISP, which itself performed at the same pace as logistic regression, the underlying loss function. We didn't expect any different, since logistic regression is, again, a linear classifier, and all of these ensemble methods are additive in some way. However, we were satisfied to demonstrate that this class could be used in an ensemble pipeline with a non-linear underlying classifier, which TISP does allow for.

B. Feature Selection with Annealing (FSA)

While TISP proved to be fairly successful, we attempted to employ another feature selection technique which was built from TISP and was intended to be less greedy. This method, Feature Selection with Annealing, was developed by Dr. Adrian Barbu, also of Florida State University, largely with computer vision applications in mind, and thus it is thought to be more powerful with data sets with substantially large number of features [2]. We still were keen to implement it and yield results because, had we had more time to work on this project, we would have used this feature selection method in another, perhaps more complicated approach. This approach will be elaborated on more thoroughly in the "Future Work" section. FSA works similarly to TISP in that there is an underlying loss function which learns model parameters to be able to make class predictions after training, however instead of a penalty function, FSA culls features on an inverse schedule during the update step of the algorithm.

Gradient update step:

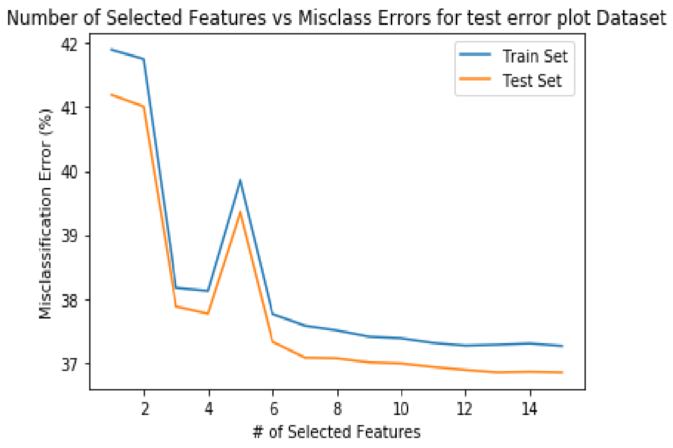
$$\beta \leftarrow \beta - \eta \frac{\delta \mathrm{L}(\beta)}{\delta \beta}$$

Inverse schedule used for feature selection step:

$$\mathrm{M}_i = k + (\mathrm{M} - k)\max\left(0, \frac{\mathrm{N}^{iter} - 2i}{2i\mu + \mathrm{N}^{iter}}\right)$$

Where L is the loss function η is the learning rate, β is the model weights, M is the input's feature dimensionality, M_i is the number of variables to be kept at iteration i, μ is a tune-able parameter, and k is the max number of select-able features. These two steps are iterated over the chosen N^{iter}. (via A. Barbu).

It was again to our surprise that we could cut the original feature space down to around 6, while still achieving similar, if not identical, performance metrics. FSA requires specification of the number of features which are to be remaining at the end, and thus in the following graph/table we show the results for k=1,...,15 (with other tuned meta-parameters being: mu=5, s=0.001, and eta=1):



```
  Train Error    |   Test Error    |   k features
-----------------+-----------------+----------------
     41.90       |     41.19       |       001
     41.75       |     41.01       |       002
     38.17       |     37.88       |       003
     38.12       |     37.76       |       004
     39.86       |     39.36       |       005
     37.76       |     37.33       |       006
     37.58       |     37.08       |       007
     37.50       |     37.07       |       008
     37.40       |     37.00       |       009
     37.38       |     36.98       |       010
     37.31       |     36.93       |       011
     37.26       |     36.88       |       012
     37.28       |     36.84       |       013
     37.30       |     36.85       |       014
     37.26       |     36.84       |       015
```

While FSA's results were as surprising as, if not more than, TISP's, we refrained from building another SciKit-Learn class, as it may be more applicable to build it for a framework such as PyTorch, for reasons that will be more apparent when we expound on our original plans for FSA later in this report.

VIII. Conclusions

The main take-aways from this project were related to efficiency and information within the features of the data set. A crucial key finding in the classification section was that boosting decision stumps essentially performed just as well as boosted decision trees. This is particularly important because boosted decision trees are one of the primary standard approaches, and building stumps is a lot less computationally expensive than building full trees. Boosted random forests were also found to perform similarly, with any depth of tree, however this method can quickly become even more expensive than boosted trees. Another important finding related directly to efficiency was that while bagged support vector machines perform similarly to single support vector machine models, the bagged approach is significantly faster. The average training time for bagged SVMs was around 2 seconds for 10,000 instances, but the average training time for lone SVMs was around 7.5 seconds for 10,000 instances.

The first surprise we had during our work with feature selection was that TISP could cull the number of influential features down to 15, and further that other classification models performed nearly identically with these as with all 28 features. However, even more surprising was that the Feature Selection with Annealing model could cull the number of features down to 6 without increasing classification error at all, and if fact it only added about 2% error when the features were culled down to 3. These results are also significant with regard to efficiency because, with this data set or any similar data sets, being able to remove even just one feature can greatly reduce computational cost, as there are millions of total instances. In fact, further feature selection research with other high energy data sets may allow experimental physicists to cut down on the data collected and/or stored for analysis.

IX. Future Work

As mentioned in the section on Feature Selection with annealing, we had somewhat broader plans for our feature selection models that didn't have the time to come to fruition, however we intend to continue on building these models after the project has concluded.

Specifically, we had initially intended to build a model similar to that of the winning Kaggle competitor, except with a slightly modified way of dynamically determining model structure. The Kaggle winner's model is of course an enticing approach because of the obvious fact that it outperformed 1,700 other models, but also because of the fact that it seems —from the observations made during this project— that ensembles of neural networks are perhaps the most powerful approach in terms of predictive performance in high energy data. However, it seems that a primary reason that the Kaggle winner's model was so powerful was the choice of activation function.

This function, max-channel activation, seems to still suffer from issues pertaining to the arbitrary nature of the neuron selection. It works by essentially determining the highest value of activation in groups of three neurons, and then zeroing the lower two. We ask: what if the other two were important to the output as well? What if all three were unnecessary? This method is certainly less arbitrary than random dropout, but we believe it can be improved even more. So we have plans of extending the Feature Selection with Annealing method to work as an activation function on each neural layer, whereby neuron selection is performed in a similar manner. We were excited to see such interesting performance of the FSA model on our data, because it indicates that such a modified model may be successful in zeroing neurons in a more "informed" way.

X. Discussion

When this project began, we had two primary concerns: the massive size of the data set and the fact that none in our group had a thorough knowledge of the domain of high energy physics. Furthermore, this domain is not one that is particularly suited for quick self-study. However, we were determined that it would be possible to center the project around data-driven approaches rather than being directed by the underlying physics. As for the first concern, it was quickly apparent that with a bit of compression and slicing of the data, we wouldn't have to worry about the full set of 11 million decay instances.

Initially, one of the biggest obstacles to quick movement with the project was the fact that our group was split between two undergraduate students and two graduate students, so creating a plan of action which was suitable to everyone's skillsets was rather difficult. Unfortunately, at some point the undergraduate students were not producing enough of a contribution to the project, despite delegations of work well within their capabilities and copious amounts of opportunities for assistance. Especially since some later portions of the project (and, sometimes, deadlines) relied on the work the undergrads should have already completed, we decided it would be fairer to everyone to split the group in two; such that the undergrads could complete the project at their own pace, and the graduate students could complete the project at our own pace. Fortunately, with just two members in the group, we were able to greatly streamline the structure and workflow of the project.

At some point during our work on classification, I decided it might be better to switch our initial section on feature selection to simply be an expanded classification where we greatly improved one or two of the models. It seemed to me that feature selection might not make sense to do, but that the feature selection models could be used to improve classification models, as discussed in the Future Work section. However, in testing a crude version of one of the feature selection models on the data, I found that it was actually quite successful in culling features and making decent predictions, so we ended up keeping things the way they were.

In fact, the section on feature selection proved to be one of the most insightful of the project, as the models were able to cut more and more information out of what was necessary to train a decent predictive model. We weren't necessarily beating the top papers or Kaggle competitors in prediction metrics, but we were reducing information requirements and improving training time, which was always our intention. Perhaps one approach or another in our classification testing works better than the others for this data set, but being able to improve models by other metrics means we can be immune to drastic structural changes if we were to switch to a different data set. After all, we intentionally included “Particle” in the title, rather than “Higgs Boson” specifically.

REFERENCES

[1] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1), Jul 2014.

[2] Adrian Barbu, Yiyuan She, Liangjing Ding, and Gary Gramajo. Feature selection with annealing for computer vision and big data learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:272–286, 02 2017.

[3] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996.

[4] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16*, 2016.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[6] Sanmay Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML ’01, pages 74–81, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, 2014.

[8] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28:337–407, 04 2000.

[9] Gábor Melis. Dissecting the winning solution of the higgsml challenge. In Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau, editors, *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *Proceedings of Machine Learning Research*, pages 57–67, Montreal, Canada, 13 Dec 2015. PMLR.

[10] D. Pavlov, J. Mao, and B. Dom. Scaling-up support vector machines using boosting algorithm. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 219–222 vol.2, Sep. 2000.

[11] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI’99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[12] Yiyuan She. Thresholding-based iterative selection procedures for model selection and shrinkage. *Electronic Journal of Statistics*, 3, 12 2008.

[13] Rupesh Kumar Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino J. Gomez, and Jürgen Schmidhuber. Compete to compute. In *NIPS*, 2013.

[14] Chang Xiao, Peilin Zhong, and Changxi Zheng. Resisting adversarial attacks by k-winners-take-all. *arXiv preprint arXiv:1905.10510*, 2019.

XI. APPENDIX

A. Honor Code Pledge

On my honor, as a University of Colorado Boulder student, I have neither given nor received unauthorized assistance.

B. Individual Contributions

Matthew Resnick: Compression, parts of Pre-Analysis, parts of Classification (Non-Standard Approaches), and parts of Feature Selection (TISP and FSA).

Taylor Egner: Parts of Pre-Analysis, parts of Classification (Neural Networks)