Supervised Classification

January 18, 2015

1 Supervised Learning on Higgs and Bidding Datasets

1.1 Abstract

1.2 Introduction

The goal of this analysis is to use Supervised methods of Machine Learning to detect Higgs boson particle from the noise of various particle collisions created in the Atlas experiment

1.3 Data

1.3.1 Higgs Dataset

On July,4 2012 physicists of the Large Hadron Collider announced the discovery of the long-saught Higgs boson particle. Experiment was taking at CERN by ATLAS group where billions of head-on colisions were recorded in the hope that elusive particle will eventually show itself. The method of observing a Higgs particle is through it's decay into another two tau particles. The challenge lies in the fact that these decays are small signal in the large background noise, which makes the problem very interesting for Machine Learning classification.

Dataset Description ATLAS provided dataset with 250000 events: mixture of signal and background. The dataset is characterised by 30 predictor variables (features) prefixed with either:

- PRI (for PRImitives) - "raw" quantities from the bunch collision as measured by the detector
- DER (for DERived) - quantities computed from the primitive features, which were selected by the physicists of ATLAS

Additionally this training dataset includes weight column for each event as well as label ("s" for signal and "b" for background)

Data Wrangling As part of pre-analysis of the data, I have plotted all 30 features to understand their predictive power to distinguish between signal and background.

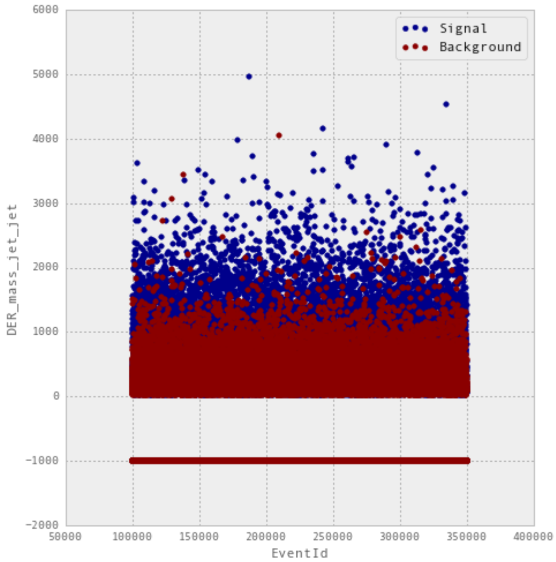
What I have found is:

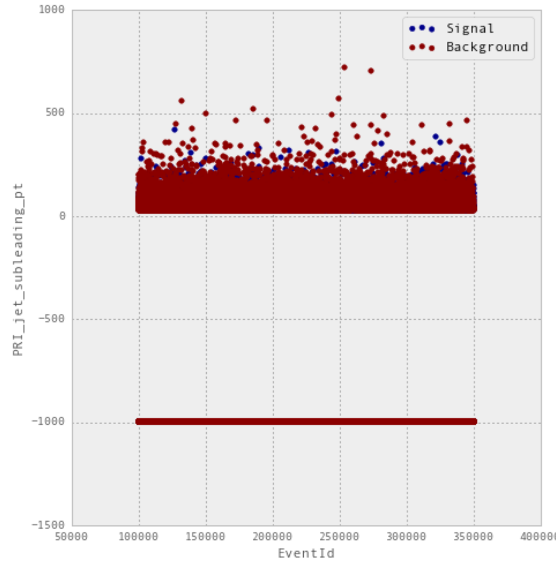
- there is a lot of missing data in the both DER and PRI features (value = -999.0) which is considered just a noise and upon consulting ATLAS data description I have confirmed that this data values are outside of normal range;
- DER features are better at differentiating the signal as if the signal is being amplified in constrast to PRI features;
- weights columns is not uniformly distributed which means not all events are equially important; so there probabilities will need to be accounted for when calculated accuracy of the classifiers

Both these phenomena are demonstrated below on the example of two features without too much of the loss of generallity.

```
In [4]: %matplotlib inline
        from algo_evaluation.datasets import *
```

```
In [2]: df = describe_higgs_raw()
```





As a result of the visual analysis, I made following adjustments to the data prior to classification:

- drop data values (-999.0) as they do not contribute to the accuracy; sometimes such data is considered a missing values and is being replaced with mean, however in this case this might actually hurt the accuracy;
- select only DER features for classification since PRI are already indirectly used and the signal is not so easily separable;

Final data after cleanup and prunning:

```
In [5]: raw_data = load_higgs_train()
        features, weights, labels = raw_data
        print 'Size of the dataset:', features.shape[0]
        print 'Number of features:', features.shape[1]
        print 'Number of positives (signal):', labels.value_counts()['s']
        print 'Number of negatives (background):', labels.value_counts()['b']
```

```
Size of the dataset: 68114
Number of features: 13
Number of positives (signal): 31894
Number of negatives (background): 36220
```

1.3.2 Bidding Dataset

1.4 Classification

1.4.1 Decision Trees

The goal here is to create a model which predicts signal by learning simple decision rules inferred from derived features.

Splitting data Prior to running and tunning the classifier from sklearn library, I've split the dataset, leaving 1/3 out for evaluation purposes. This gave me the initial benchmark of 78% accuracy on the test set. Given the underlying difficulty of detecting higgs signal in general, the accuracy "out-of-the-box" was not bad, however I wanted to see how much more can be achieved with indirect prunning.

```
In [101]: dataset = split_dataset(features, weights, labels)
          for category in dataset:
              data = dataset[category]
              print '{}: {}'.format(category, {d: len(data[d]) for d in data})
```

```
test: {'labels': 22478, 'weights': 22478, 'features': 22478}
training: {'labels': 45636, 'weights': 45636, 'features': 45636}
```

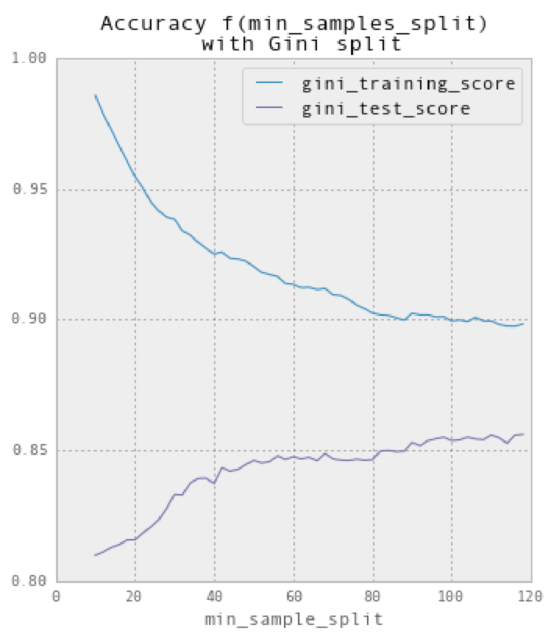
Note: Same splitting applies to the rest of algorithm evaluation as well

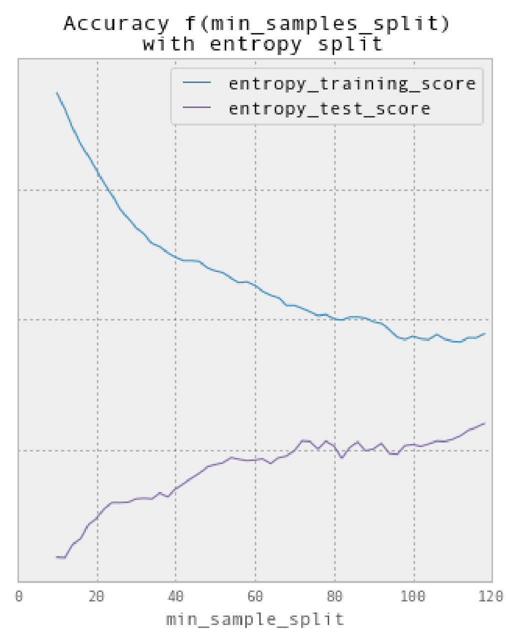
Prunning by tunning minimum number of samples required to split an internal node

```
In [22]: from algo_evaluation.algos import decision_tree as dt
```

```
In [ ]: df = dt.estimate_best_min_samples_split()
```

```
In [48]: dt.plot_accuracy_function(df, smoothing_factor=5)
```





Observation on min_sample_split:

Given that default setting for minimum number of samples required to split is 2, the classifier was clearly overfitting the data. By tunning the parameter, I was able to increase the test accuracy to above 85% while decreasing accuracy on training dataset. By visaul inspection, it can be inferred that optimal setting for minimum number of samples required to split is around 60.

Observation on splitting rule:

Generally, the decision-tree splitting criteria matters as entropy based criteria favors multinominal features? However, it does not appear to make a big difference on the higgs dataset. Both 'gini' and 'entropy' produce similar accuracy trends with hardly detectable slower ramp-up of the entropy for smaller values of min_samples_split.

Additional tunning of the classfier, such as maximum depth of the tree or minimum number of samples required to be at a leaf node did not contribute to the accuracy of the predictions, so they were left to default.

Below are the final scores achieved by Decision Tree classifier:

```
In [23]: dt_training_accuracy, dt_test_accuracy = dt.run_decision_tree(raw_data, min_samples_split=60)
         print 'Accuracy on training data:', dt_training_accuracy
         print 'Accuracy on test data:', dt_test_accuracy
```

```
Accuracy on training data: 0.912119515125
Accuracy on test data: 0.859211327755
```

1.4.2 Neural Networks

Since sklearn does not implement Neural network, in this analysis I am using pybrain library. Same goal as in the decision trees evaluation: classify Higgs boson from the background.

Fully connected Neural Network is constructed with the following specifications:

- input layer - 13 sigmoid neurons
- hidden layer - 19 sigmoid neurons
- output layer - 1 softmax neuron (since output should be binarized)
- training algorithm - backpropagation

```
In [1]: from algo_evaluation.algos import neural_network as nn
        reload(nn)
```

```
Out[1]: <module 'algo_evaluation.algos.neural_network' from '/Users/maestro/schoolspace/bag-of-algorithm
```

```
In []: net = nn.run_neural_net(raw_data)
```

```
INFO:algo_evaluation:epoch:    5
INFO:algo_evaluation:train error: 46.9377917041
INFO:algo_evaluation:test error: 46.5942963919
```

```
In []: #df = nn.estimate_training_iterations()
```

1.4.3 AdaBoost

AdaBoost is an example of the ensemble classifier, where a collection of weak learners are combined to produce a meta estimator.

Sklearn python library is using DecisionTrees as the base estimator, so I should be able to get at least same accuracy as found during DecisionTrees evaluation.

Striving to increase the performance above my benchmark, I tunned two parameters:

- maximum number of estimators at which boosting is terminated
- learning rate

There is an obvious tradeoff between these two parameters and using grid search I was able find the most suitable combination for my Higgs dataset.

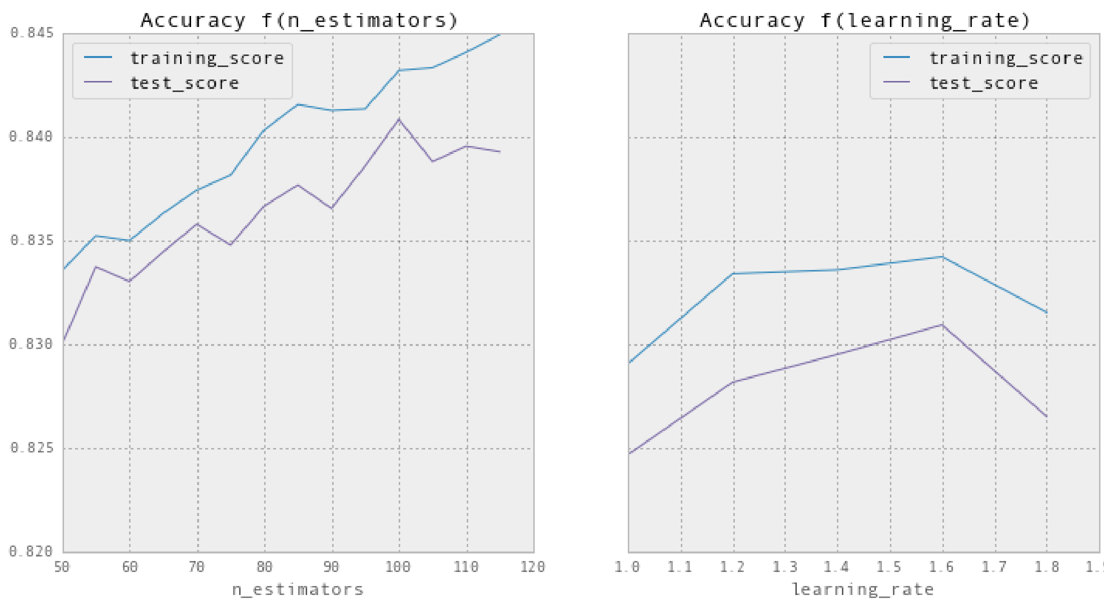
```
In [24]: from algo_evaluation.algos import adaboost as ab
```

Accuracy functions plotted below showed the expected behavior of the classifier.

- increasing number of estimator is positively correlated with the accuracy; the optimal number n_estimators ~ 100 did not however improve the accuracy of what I was already getting with Decision Trees
- interestingly, very small values of learning rate were negatively affecting the accuracy which means classifier was overfitting; the graph below showed there is an optimal range of learning rates beyond which accuracy tanks drastically

```
In [15]: estimator_df = ab.estimate_best_n_estimators()
         learning_rate_df = ab.estimate_best_learning_rate()
```

```
In [16]: ab.plot_accuracy_functions(estimator_df, learning_rate_df, smoothing_factor=5)
```

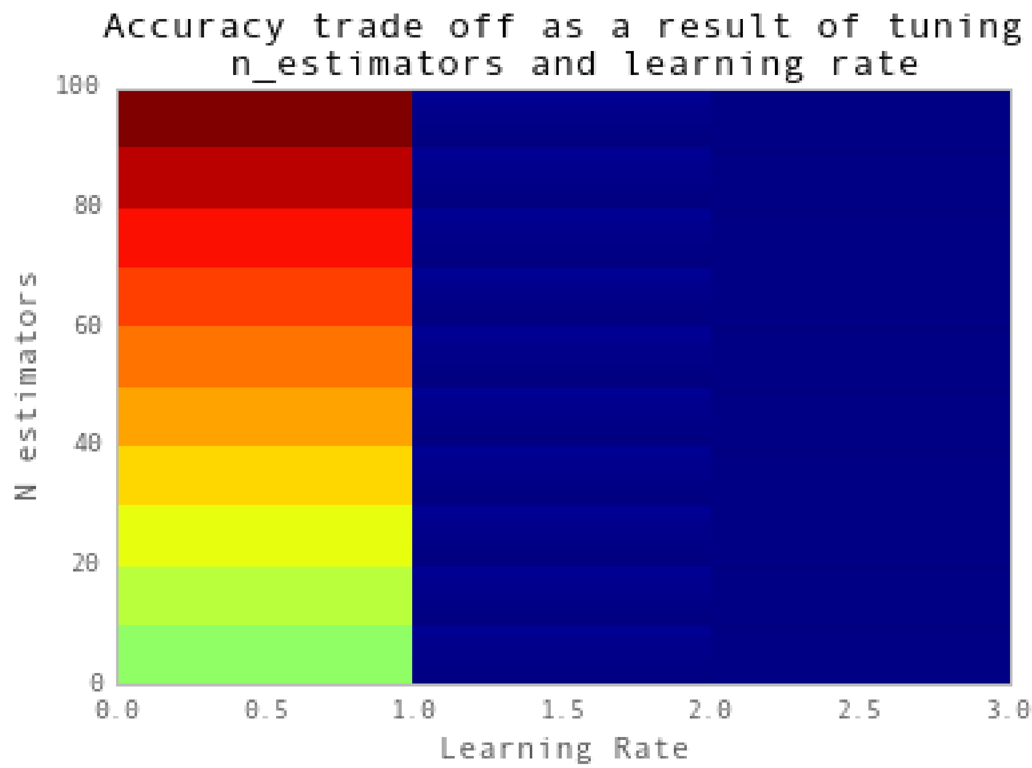


It is easier to observe the accuracy tradeoff with heatmap visualization.
From looking at the plot, following conclusions could be made:

- anything with learning rate higher than 1.0 has low accuracy as indicated by color blue
- as number of estimators is growing, accuracy increases as long as we are in acceptable range of the learning rate

```
In [50]: trades_df = ab.tradeoff_estimators_learning_rate(raw_data)
```

```
In [105]: heatmap = ab.plot_tradeoff(trades_df)
```



Below are the final scores achieved by AdaBoost classifier:

```
In [25]: boost_training_accuracy, boost_test_accuracy = ab.run_AdaBoost(raw_data, n_estimators=85, lear
         print 'Accuracy on training data:', boost_training_accuracy
         print 'Accuracy on test data:', boost_test_accuracy
```

```
Accuracy on training data: 0.840054465166
Accuracy on test data: 0.843939505107
```

1.4.4 Support Vector Machines

```
In [2]: from algo_evaluation.algos import svm
```

```
In []: c_error_df = svm.estimate_best_c()
```

```
In []: gamma_df = svm.estimate_best_gamma()
```

```
In []: svm.plot_accuracy_functions(c_error_df, gamma_df, smoothing_factor=5)
```

```
In [26]: svm_training_accuracy, svm_test_accuracy = svm.run_svm(raw_data, c_error_term=1.0, gamma=0.0)
         print 'Accuracy on training data:', svm_training_accuracy
         print 'Accuracy on test data:', svm_test_accuracy
```

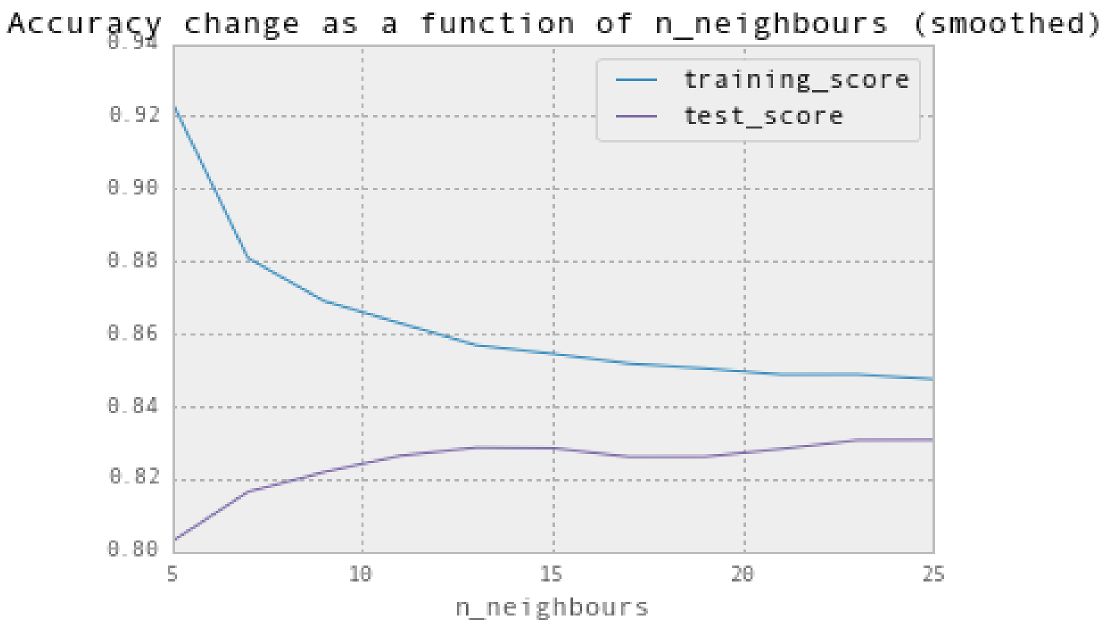
```
Accuracy on training data: 1.0
Accuracy on test data: 0.996152460549
```

1.4.5 K-Nearest Neighbors

```
In [14]: from algo_evaluation.algos import knn
```

```
In [15]: knn_df = knn.estimate_best_n_neighbours()
```

```
In [107]: knn.plot_accuracy_function(knn_df, smoothing_factor=3)
```




```
In [27]: knn_training_accuracy, knn_test_accuracy = knn.run_knn(raw_data, n_neighbours=10)
         print 'Accuracy on training data:', knn_training_accuracy
         print 'Accuracy on test data:', knn_test_accuracy
```

```
Accuracy on training data: 0.888995233367
Accuracy on test data: 0.858143125028
```

1.5 Performance Comparison

After classifying Higgs particle with presented algorithms, it is very interesting to compare they accuracy scores on both training and test data.

Accuracy across all algorithms is comparable with exception if SVM algorithm which exceeding expecations right "out of the box". Having accuracy of 0.99 on test data is extremely high which makes me conjecture that perhaps ATLAS simulated events for training using SVM (reading additional literature on particle detection from CERN boosts this hypothesis).

Aggregate all scores and compare them in the table below:

```
In [35]: training_scores = [dt_training_accuracy,
                            boost_training_accuracy,
                            svm_training_accuracy,
                            knn_training_accuracy]
         test_scores = [dt_test_accuracy,boost_test_accuracy, svm_test_accuracy,knn_test_accuracy]
         algorithms = ['decision_tree', 'adaboost', 'svm', 'knn']
         scores = pd.DataFrame.from_records([training_scores, test_scores],
```

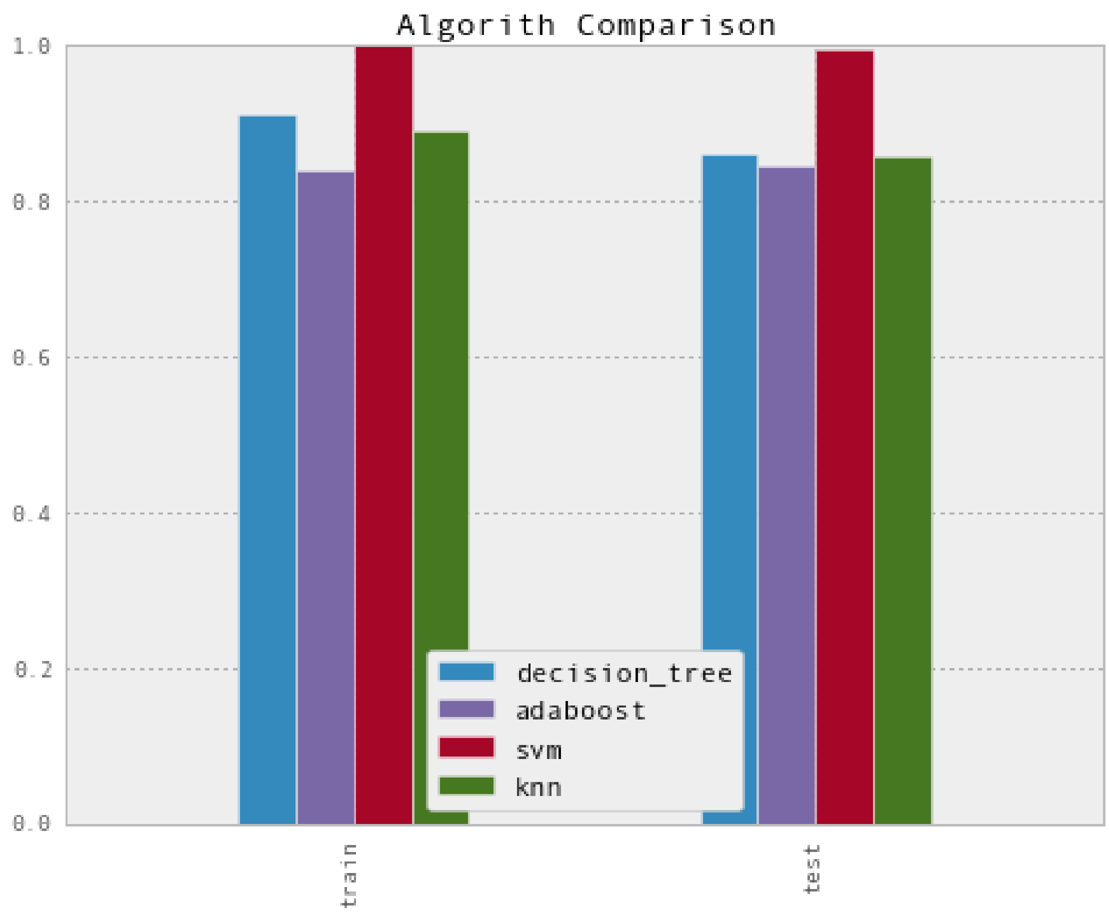
```
                                    columns=algorithms,
                                    index=['train', 'test'])
scores
```

Out[35]:

	decision_tree	adaboost	svm	knn
train	0.912120	0.840054	1.000000	0.888995
test	0.859211	0.843940	0.996152	0.858143

Plot accuracy scores on both training and test data across all algorithms.

In [49]:
```
comparison = scores.plot(kind='bar', figsize=(8, 6), title='Algorith Comparison')
```



1.6 Conclusion

1.7 Acknowledgement

In [5]:
```
# do the matrix between all five algos of how much they agree with each other
# plot accuracy histograms
# compare timing
```

I have used scikit-learn python library for all the algorithms used above, except neural network for which I used pybrain python library. I addition to machine learning tools, I have used data analysis and computational tools such as pandas, numpy and matplotlib for plotting

1.8 References

[1] Higgs Boson Machine Learning Challenge: https://www.kaggle.com/c/higgs-boson

[2] Learning to discover: the Higgs boson machine learning challenge: http://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf

[3] Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC: http://arxiv.org/abs/1207.7214

[4] Support Vector Machines in Analysis of Top Quark Production: http://arxiv.org/abs/hep-ex/0205069

[5] Stephen Marsland. Machine Learning: An Algorithmic Perspective. CRC Press, 2009

[6] Scikit Learn Documentation, Online Available, at http://scikitlearn.org/stable/documentation.html

[7] Pybrain Documentation, Online Available, at http://pybrain.org/docs/index.html