Supervised Classification

January 24, 2015

1 Supervised Classification on Higgs and Converters Datasets

1.1 Introduction

The goal of this analysis is to evaluate and constrast five different supervised algorithms by applying them to two classification problems:

- **Higgs Boson**: given outcomes of particle decays detect Higgs boson; separate signal from the noise based on multitude of particle collisions created in the Atlas experiments, where protons of extra-high energy are brought head-on.
- **Converter**: given minimal user and geo information along with the converting concept, learn how to detect users who are more likely to convert; this probability is used for deciding when and how much to bid on the advertisment slot in realtime.

1.2 Data

1.2.1 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 place at CERN in the 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 signals 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;

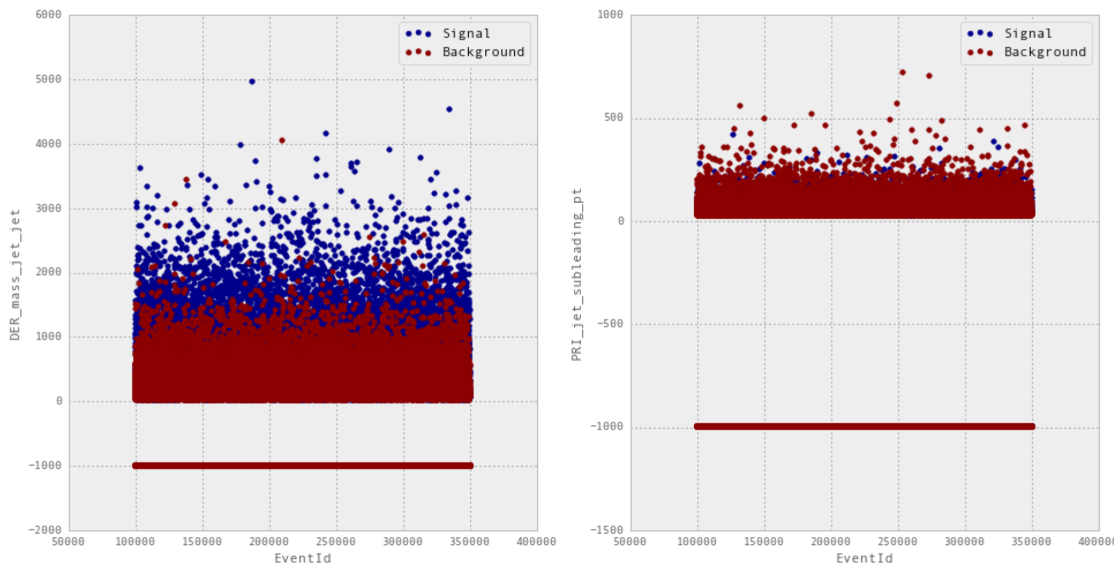
- DER features are better at differentiating the signal as if the signal is being amplified in constrast to PRI features;
- weights columns is not uniformly distributed which means not all events are equially important; so there probabilities will need to be accounted for during accuracy evaluation

Differetiating power of DERived features vs PRImitives is demonstrated below on the example of two features only without too much of the loss of generallity (the rest of the features exhibit similar behavior)

Also note the definite noise around the -999.0 level which most certainly does not the signal

```
In [1]: %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 based on my experiments it actually hurts the accuracy;
- select only DER features for classification since PRI are already indirectly used and the signal is not so easily separable;

Higgs data and metadata after cleanup and prunning (here prunning means selecting a subset of features based on initial experiments):

```
In [3]: higgs_data = load_higgs_train()
        features, weights, labels = higgs_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
```

1.2.2 Converters Dataset

For secondary classification problem, I have selected converters dataset from my work group research. Dataset represents examples for one campaign for 1-4 days (based on the number of impressions shown by campaign per day)

In contrast to real-values features in the Higgs dataset, converters features are mostly categorical, so in order to apply classification algoritms, I did pre-processing which transformed non-numerical features to numerical using sklearn LabelEncoding.

The last column in the dataset is prediction attribute - we want to predict which impressions led to conversion vs not. The class variable is what we are trying to predict and can be one of the following values:

- non-converter : If this impression did not generate lead or conversion
- lead : If this impression made the user to visit lead pixel (for eg homepage, form page, etc)
- converter : If this impression led to conversion

Converters data and metadata after cleanup and transformations:

```
In [4]: bid_data = load_bidding_train()
        bid_features, bid_weights, bid_labels = bid_data
        print 'Size of the dataset:', bid_features.shape[0]
        print 'Number of features:', bid_features.shape[1]
        print 'Number of converters:', bid_labels.value_counts()['converter']
        print 'Number of non-converters:', bid_labels.value_counts()['non-converter']
        print 'Number of leads:', bid_labels.value_counts()['lead']
```

```
Size of the dataset: 57970
Number of features: 15
Number of converters: 399
Number of non-converters: 54615
Number of leads: 2956
```

1.3 Higgs Classification

1.3.1 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 using sklearn collection of algorithms, I've split the dataset, leaving 1/3 out for evaluation purposes.

```
In [5]: 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!

With this split I acquired benchmark of 78% accuracy on the test set. Given the underlying difficulty of detecting higgs signal in general, the initial accuracy of the Decision Trees was not bad, however I wanted to see how much more can be achieved with indirect prunning.

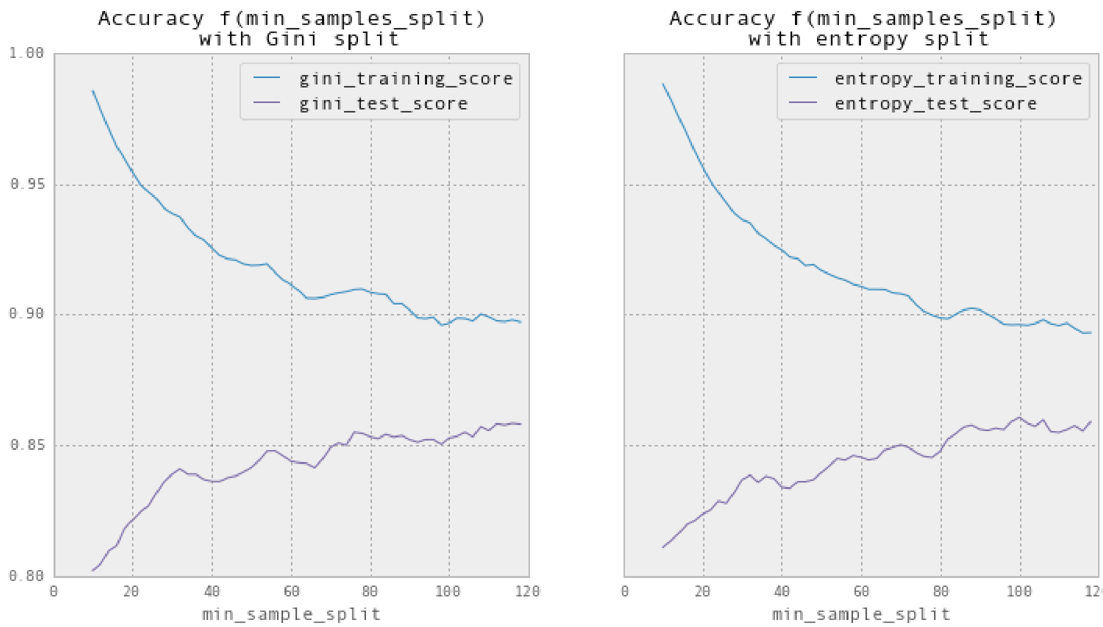
Prunning by tunning minimum number of samples required to split an internal node: Below I plotted two different versions of the accuracy function of decision tree complexity expressed through min samples required to split - one for the Gini splitting criterion and another for Entropy.

Note: Here and for all consequent graphs I have used rolling means (with adjustable smoothing factor) to smooth the plotting of accuracy function.

```
In [6]: from algo_evaluation.algos import decision_tree as dt
```

```
In [7]: df = dt.estimate_best_min_samples_split()
```

```
In [8]: 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 visaully inspecting the graphs, it can be inferred that optimal setting for minimum number of samples required to split is around 60 which reduces the complexity of the decision tree greatly.

Observation on splitting rule:

Generally, choosing decision-tree splitting criteria can affect the accuracy of the classification; for example entropy based criteria favors features with multiple nominals and given we have real-valued data, this criteria could be too sensitive. 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 settings.

Below are the final scores achieved by Decision Tree classifier:

```
In [9]: dt_trn_acc, dt_tst_acc = dt.run_decision_tree(higgs_data, min_samples_split=60)
        print 'Accuracy on training data:', dt_trn_acc
        print 'Accuracy on test data', dt_tst_acc
```

```
Accuracy on training data: 0.909422374408
Accuracy on test data 0.844064103952
```

1.3.2 Neural Networks

Since sklearn does not implement Neural network, in this analysis I am using pybrain library.

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

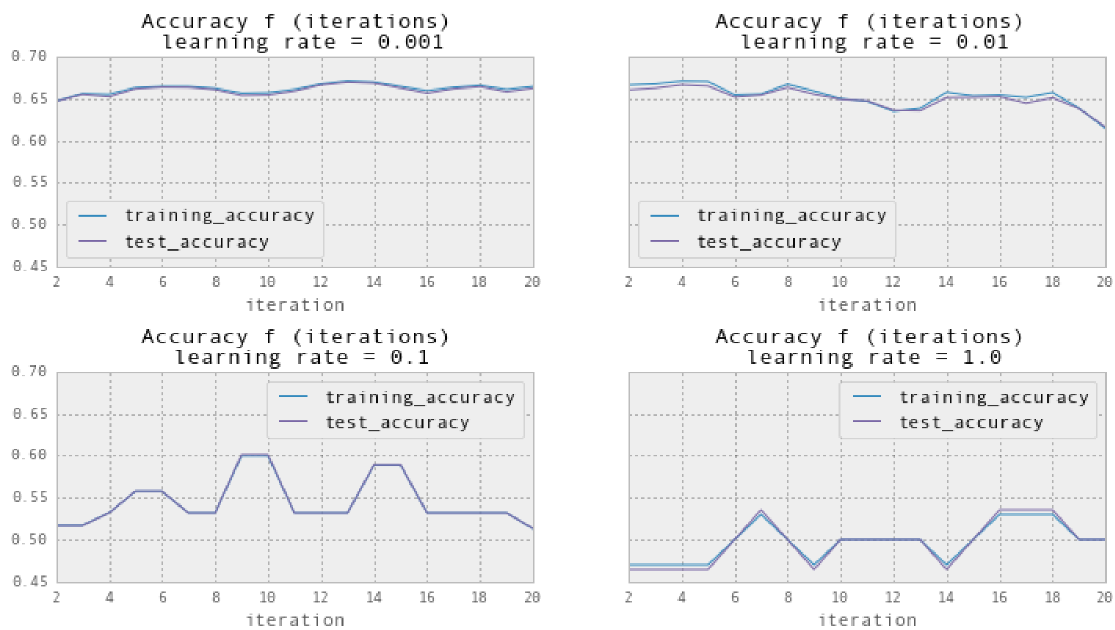
```
In [10]: from algo_evaluation.algos import neural_network as nn
```

```
In [11]: nn_df = nn.estimate_training_iterations(n_iterations=20)
```

I have run the experiment for up to 500 iterations (only 20 iterations shows here due to time constraints) and unfortunately the network is not learning very well.

Neural Net on Higgs is the most stable with learning rate 0.001 as accuracy curve is not jumping very much. Demonstrated below are learning curves for 4 settings of learning rates: 0.001, 0.01, 0.1 and 1.0.

```
In [12]: nn_plot = nn.plot_accuracy_function(nn_df, smooth_factor=2)
```



Final accuracy scores of the Neural Networks classification:

```
In [13]: epochs, nn_trn_err, nn_tst_err = nn.run_neural_net(higgs_data)
         nn_trn_acc = 1 - nn_trn_err/100
         nn_tst_acc = 1 - nn_tst_err/100
         print 'Total epochs (iterations) trained', epochs
         print 'Accuracy on training data:', nn_trn_acc
         print 'Accuracy on test data', nn_tst_acc
```

```
Total epochs (iterations) trained 5
Accuracy on training data: 0.46698950413
Accuracy on test data 0.470792365529
```

Due to poor results, I have concluded that Neural Networks is not the best algorithm for detecting Higgs.

1.3.3 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 which shrinks the contribution of each classifier

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 (final settings diplayed below along with the scores)

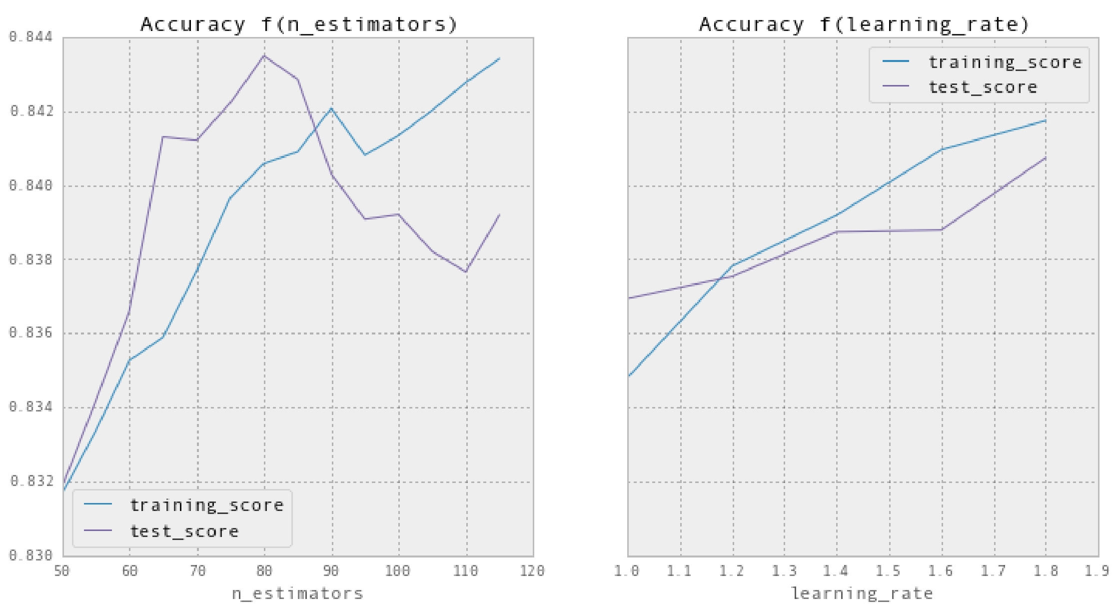
```
In [14]: from algo_evaluation.algos import adaboost as ab
```

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
- ensemble classifier was able to learn better when contributions from from each local classifier were considered equally important (learning_rate ~ 1.0; learning rate below 0.1 leads to very poor results)

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

```
In [40]: ab.plot_accuracy_functions(estimator_df, learning_rate_df, smoothing_factor=5)
```



The most surprising effect I have observed with the ensemble classifier is the dynamics of accuracy function depending on the number of estimators. At some point, above 80, accuracy drops on the training set while continues to grow on the test set.

Below are the final scores achieved by AdaBoost classifier:

```
In [17]: ab_trn_acc, ab_tst_acc = ab.run_AdaBoost(higgs_data, n_estimators=85, learning_rate=1.0)
         print 'Accuracy on training data:', ab_trn_acc
         print 'Accuracy on test data', ab_tst_acc
```

```
Accuracy on training data: 0.844630908339
Accuracy on test data 0.835518091704
```

1.3.4 Support Vector Machines

Support Vector Machines is very effective in high-dimensional spaces and given that I have selected 13 features so far for the Higgs dataset, SVM is expected to work very well. From my pre-analysis of the data, linear separability was out of question, so it is important to choose a good non-linear kernel.

Default kernel is `rbf` and upon training on the dataset it classified every single example correctly (more on this phenomenon in the conclusion section).

```
In [18]: from algo_evaluation.algos import svm
```

```
In [19]: svm_trn_acc, svm_tst_acc = svm.run_svm(higgs_data, regularization_term=1.0, gamma=0.0)
         print 'Accuracy on training data: {} \nAccuracy on test data: {}'.format(svm_trn_acc, svm_tst_a
```

```
Accuracy on training data: 1.0
Accuracy on test data: 0.996046097547
```

Even though the accuracy is perfect, I performed multiple experiments tunning:

- regularization parameters ‘C’: [1, 10, 100, 1000]
- gamma ‘gamma’: [1e-3, 1e-4]
- kernel: rbf

Using grid search with python library, none of the parameters combinations gave better results, so there isn’t anything interesting to plot here.

In addition to tunning SVM parameters, I also iteratively changed the size of the dataset (each iteration size = 10X of the previous) and suprisingly enougth even on the 1/10 of the original dataset size, SVM still performed as good as on full dataset.

1.3.5 K-Nearest Neighbours

K-Nearest Neighbours is an example of the instance based classification where instead of the learning the predictive function, all examples are stored and considered and when the new event is encountered, as set of similar events is used for classification.

Given this intuition behind the classifier, KNN is the best suited for classifying Higgs particle. To detect signal from background, we do not need to look at the whole data, but rather similar decaying events.

The main question to answer here, how many of such events should we look for.

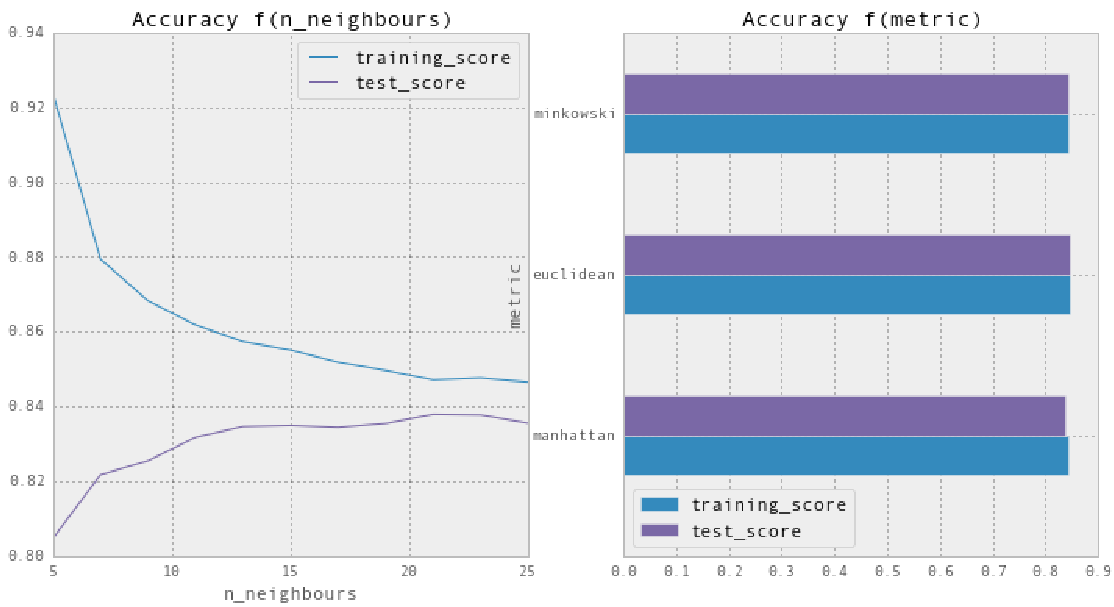
The accuracy curve below given as a function of the number of neigbours suggests 10 to be an optimal number which generalizes the dataset pretty well.

```
In [20]: from algo_evaluation.algos import knn
```

```
In [21]: knn_df = knn.estimate_best_n_neighbours()
```

```
In [22]: p_df = knn.estimate_best_power()
```

```
In [23]: knn_plot = knn.plot_accuracy_function(knn_df, p_df, smoothing_factor=3)
```



Additionally, KNN in theory classifies examples very different depending on the distance metric used. However in the current dataset, metric did not alter accuracy very much and so default can be used. Metrics attempted in the evaluation:

- euclidian
- manhattan
- minkowski

Final estimation was using euclidian distance metric and produced following results:

```
In [24]: knn_trn_acc, knn_tst_acc = knn.run_knn(higgs_data, n_neighbours=10)
         print 'Accuracy on training data:', knn_trn_acc
         print 'Accuracy on test data', knn_tst_acc
```

```
Accuracy on training data: 0.8846735875
Accuracy on test data 0.860250244311
```

Aggregate all accuracy scores observed on the Higgs dataset for further constrast with Converters dataset:

```
In [25]: higgs_scores = [[dt_trn_acc, nn_trn_acc, ab_trn_acc, knn_trn_acc, svm_trn_acc],
                         [dt_tst_acc, nn_tst_acc, ab_tst_acc, knn_tst_acc, svm_tst_acc]]
```

1.4 Bidding Classification

This classification problem couldn't be more different than Higgs classification and there is huge constrast in the field (physics vs programmatic marketing), data sctructure (categorical versus real-values features), sample and concept complexity (correlation between concept and features is stronger in the bidding dataset)

In the interest of space, only final scores are recorded for bidding dataset accros all algorithms.

```
In [26]: dt_trn_acc, dt_tst_acc = dt.run_decision_tree(bid_data)
         print 'Accuracy on training data:', dt_trn_acc
         print 'Accuracy on test data', dt_tst_acc
```

```
Accuracy on training data: 0.969515178043
Accuracy on test data 0.96246929068
```

```
In [27]: iterations, nn_trn_err, nn_tst_err = nn.run_neural_net(bid_data)
         nn_trn_acc = 1 - nn_trn_err/100
         nn_tst_acc = 1 - nn_tst_err/100
         print 'Accuracy on training data:', nn_trn_acc
         print 'Accuracy on test data', nn_tst_acc
```

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

```
In [28]: ab_trn_acc, ab_tst_acc = ab.run_AdaBoost(bid_data)
         print 'Accuracy on training data:', ab_trn_acc
         print 'Accuracy on test data', ab_tst_acc
```

```
Accuracy on training data: 0.941785318881
Accuracy on test data 0.942815325911
```

```
In [29]: knn_trn_acc, knn_tst_acc = knn.run_knn(bid_data)
         print 'Accuracy on training data:', knn_trn_acc
         print 'Accuracy on test data', knn_tst_acc
```

```
Accuracy on training data: 0.943484641726
Accuracy on test data 0.94532434269
```

```
In [30]: svm_trn_acc, svm_tst_acc = svm.run_svm(bid_data)
         print 'Accuracy on training data:', svm_trn_acc
         print 'Accuracy on test data', svm_tst_acc
```

```
Accuracy on training data: 0.994953526095
Accuracy on test data 0.955726308086
```

```
In [31]: bid_scores = [[dt_trn_acc, nn_trn_acc, ab_trn_acc, knn_trn_acc, svm_trn_acc],
                        [dt_tst_acc, nn_tst_acc, ab_tst_acc, knn_tst_acc, svm_tst_acc]]
```

Every single algorithm performed very good on the converters dataset with minimal effort as compared to large number of experiments and tunning for Higgs detection. This supports the conjecture of the strong existing correlation between users profiles and their preference for conversion. This behavior is relatively easy learned for bid and creative placement optimization.

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 three algorithms - Decision Tree, AdaBoost, KNN - is comparable and around 85%.
- SVM algorithm exceeded 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).
- Neural Network on the other side produced very low accuracy (53%) regardless of tuning and number of iterations. Another drawback of this algorithm is that it took very long time to train.

Aggreggate of all scores and their comparison is shown in the table below:

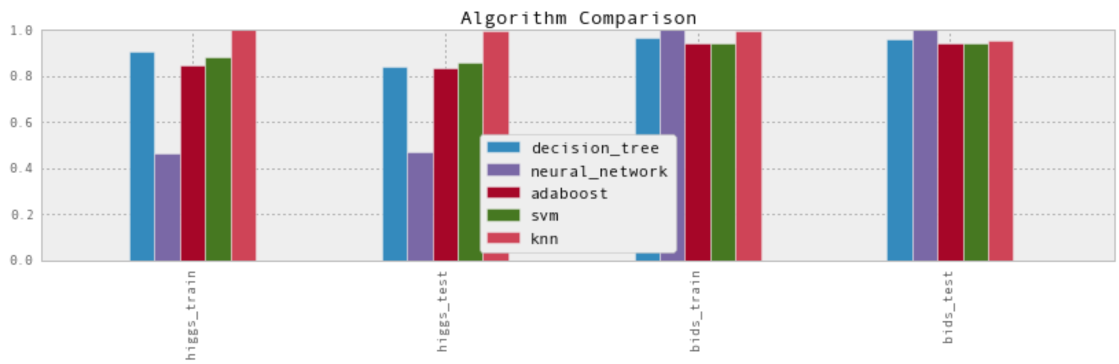
In [32]:
```
scores = higgs_scores + bid_scores
algorithms = ['decision_tree', 'neural_network', 'adaboost', 'svm', 'knn']
scores = pd.DataFrame.from_records(scores, columns=algorithms,
                                   index=['higgs_train', 'higgs_test',
                                          'bids_train', 'bids_test'])
scores
```

Out[32]:

	decision_tree	neural_network	adaboost	svm	knn
higgs_train	0.909422	0.466990	0.844631	0.884674	1.000000
higgs_test	0.844064	0.470792	0.835518	0.860250	0.996046
bids_train	0.969515	1.000000	0.941785	0.943485	0.994954
bids_test	0.962469	1.000000	0.942815	0.945324	0.955726

Notice that accuracy on the second classification problem (converters) is higher than for Higgs detection which should not come as a surprise.

In [33]:
```
comparison = scores.plot(kind='bar', figsize=(14, 3), title='Algorithm Accuracy Comparison')
```



Higgs Detection Learning Curves. So far the analysis of the accuracy was demonstrated as a function of algorithm parameters. It is also interesting to look at the learning curves as a function of the sample size (number of examples on the dataset)

I have used sklearn utility library for finding the learning curve as a function of sample complexity (number of events recorded in the Higgs dataset and number of impressions shown in the selected campaign).

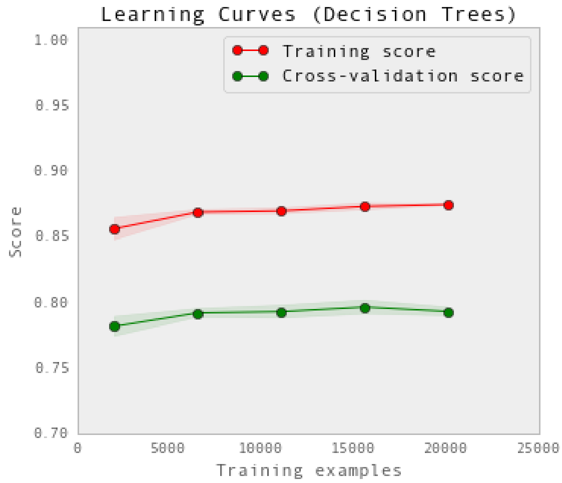
Here, accuracy is computed using cross validation as opposed to splitting dataset.

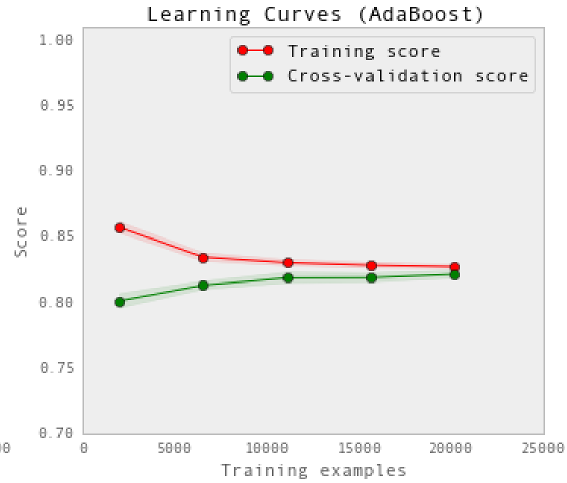
Following reductions made:

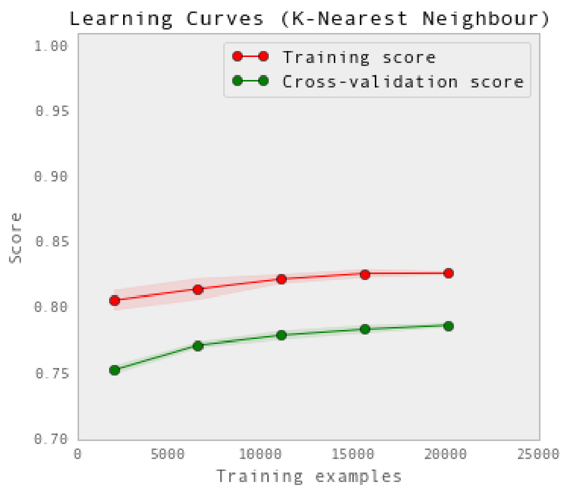
- initial 20 iterations of cross validation were reduced to 5 as learning curve converged to stable state very early;
- initial model evaluations were performed on the full dataset (60k) and eventually reduced to 30k as learning curve as not affected at that point.

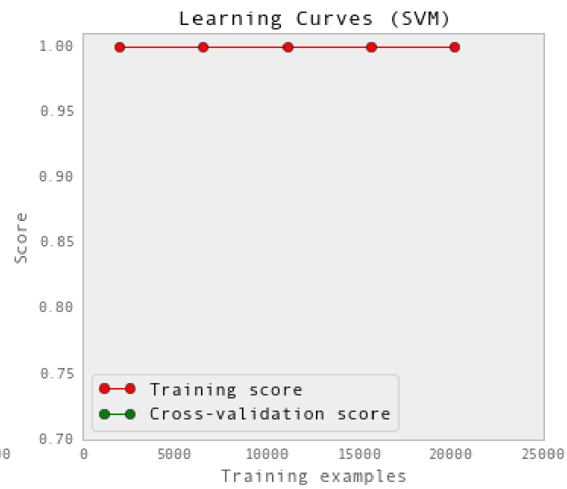
In [34]:
```
from algo_evaluation.plotting.plot_learning_curves import plot_learning_curves
```

In [35]:
```
higgs_learning_curve = plot_learning_curves(higgs_data, limit_size=30000)
```









Number of event recorded in the Higgs dataset are mostly noticed only in the ensemble boosting algorithm and testing accuracy eventually converges to training accuracy which shows that AdaBoost in this case does not overfit.

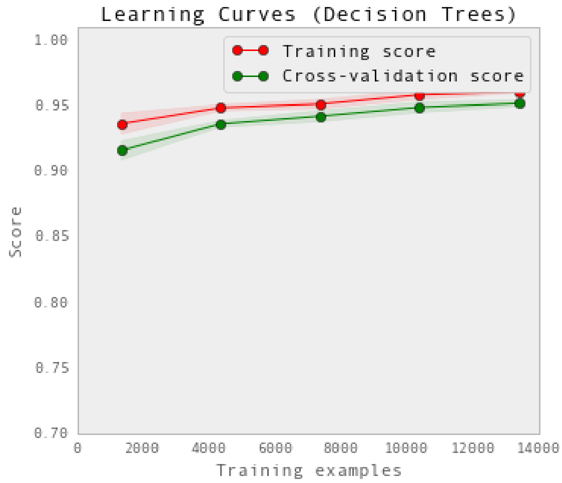
Generally, Higgs particle detection is a very difficult machine learning task, but what the above experimentation showed me that even difficult problems like that could be tackled and provide the sufficient accuracy.

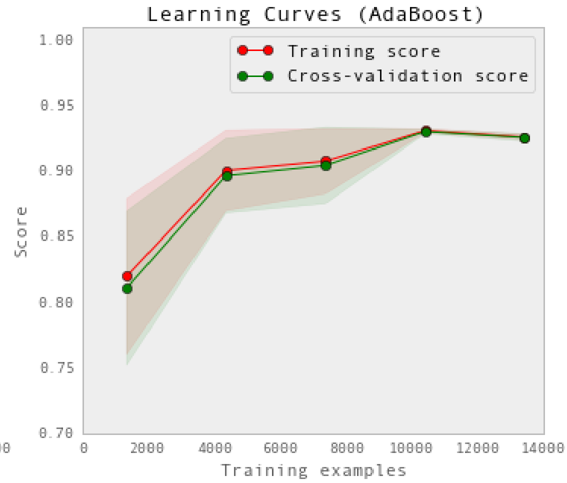
Having more domain knowledge could be helpful in feaure aggregation and fine-tunning of the algorithms (especially the neural networks) to achieve a better score.

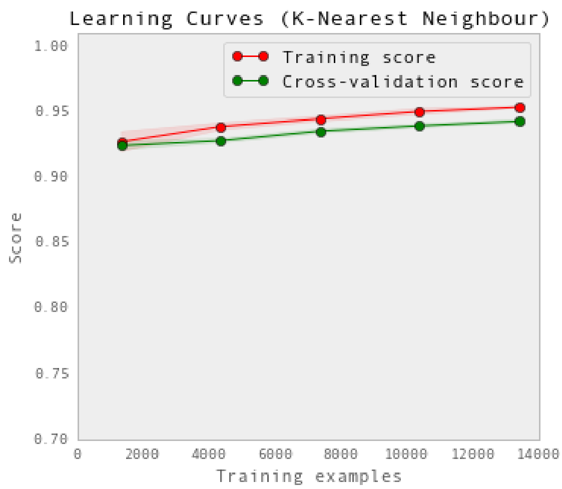
Additionally I would have liked to have more processing power to see if Boosting is capable of doing more than just 85%.

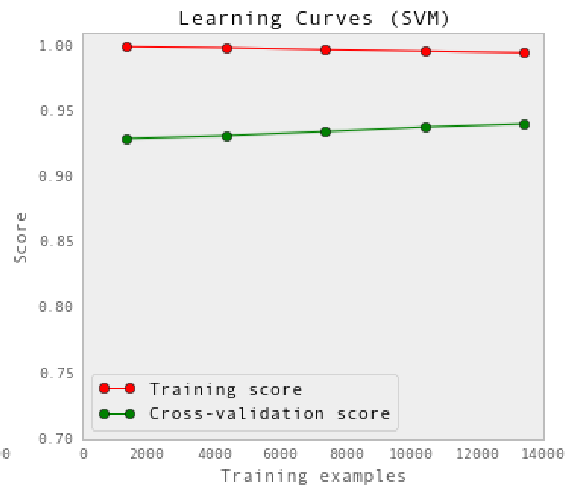
Converters Learning Curves. Same as in building learning curves for Higgs Models, sample size was reduced to 30k since beyond that no change was detected.

```
In [36]: bid_learning_curve = plot_learning_curves(bid_data, limit_size=30000)
```









Learning curves for convertes provided more insight to differences between algorithms and between classification problems:

- Decision Trees: After 4k impressions test accuracy stabilized and no more generalization could be achieved.
- AdaBoost: Accuracy has large variance for up to 10k impressions after which training and testing curves perfectly converged which means we really need a lot of data to make a good prediction of user converting behavior
- KNN and SVM

1.6 Acknowledgement

Following python libraries were used for the evaluation of algorithms:

- scikit-learn (SVM, AdaBoost, KNN, Decision Tree, Accuracy and Error evaluation)
- pybrain (Neural Network)
- pandas (data analysis)

- numpy (data wrangling)
- matplotlib (plotting)

1.7 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