METHODOLOGY FOR ANALYSIS

I use Knime analytics platform with weka 3.6 and 3.7 extensions to run all the tests.
I begin by evaluating how well the learners perform with different amounts of samples from the data sets by generating learning curves using the default parameters for the learners in Weka.
I then try to optimize the parameters for each learner by evaluating the performance of the learners over different parameter values. For e.g.: for the decision tree learner I gradually increase the confidence factor (pruning factor) and re-train the learner and observe the performance.
After optimizing the parameters, I re-generate the learning curves to see how the optimized parameters affect the performance over varied data samples.

PROCESS FOR GENERATING A LEARNING CURVE

I generate all the learning curves through the following set of steps:

1) Begin by sampling 10% of the data randomly from the training set and call it *train1*.
2) Train the learner using *train1*.
3) Get the accuracy score of the learner on both *train1* and the test set.
4) Sample 10% more data randomly from the training set, add it to *train1* and continue from step 2.
5) Repeat the above process until 100% samples have been added into *train1* and used.

METRIC OF PERFORMANCE

I take accuracy and Cohen's kappa as our metric of performance.
While accuracy helps in comparing the performance of the learners, kappa score gives a general picture of how much better a learner is performing on a data set compared to a random guesser.

ALGORITHMS

The following algorithms are used in the analysis:

- **J48**: This is an implementation of the C4.5 algorithm for decision trees in Weka.
- **AdaBoost M1:** This is available in Weka. I use it with the J48 classifier to create boosted decision tree.
- **Multilayer Perceptron**: This is a type of neural network available in Weka. It is a network of sigmoid nodes and uses back-propagation to adjust weights on each node.
- **K Nearest Neighbor**: I use the implementation of K-NN available in Knime 2.11 as I observe it to be much faster than the **IBK** implementation available in Weka.
- **LIBSVMLearner**: This is an extension of LIBSVM in Knime 2.11. I use the C-SVC type of SVM for the classifications.

DATA SETS

POKER HAND [2]

The data set has 10 attributes which represent 5 cards drawn from a deck and an 11th Class attribute which represents the "Poker Hand". Each card is described by 2 attributes Suite(S) and Rank(R).

CLASS DISTRIBUTION IN TRAINING SET

Label	Count	%
0	12493	49.95202%
1	10599	42.37905%
2	1206	4.82207%
3	513	2.05118%
4	93	0.37185%
5	54	0.21591%
6	36	0.14394%
7	6	0.02399%
8	5	0.01999%
9	5	0.01999%

WHAT MAKES THIS DATASET INTERESTING

This dataset has been found to be challenging for classification algorithms [3], it would be interesting to see how well supervised classification learners actually perform on this dataset and see if the performance can be improved.

CONNECT-4 [2]

This dataset contains all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced.

There are total 43 attributes, the first 42 represent each slot on the game board, their values can be either (x=player 1 has taken, o=player 2 has taken, b=blank), the 43rd represents whether player 1 wins, looses or the game draws.

CLASS DISTRIBUTION IN TRAINING SET

Label	Count	%
win	44473	65.83%
loss	16635	24.62%
draw	6449	9.55%

WHAT MAKES THIS DATASET INTERESTING

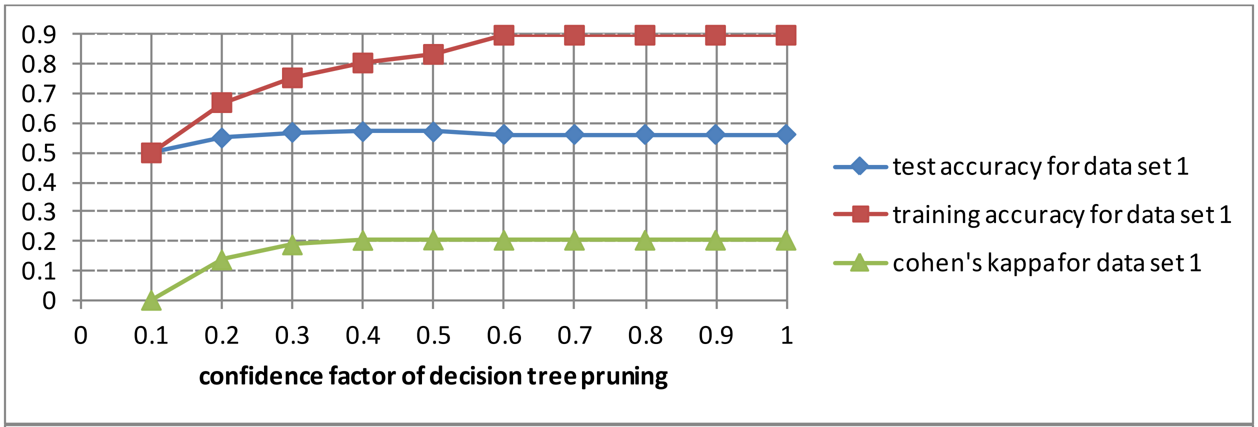
Connect-4 is a solved game, which means that the outcome of the game can be predicted if both the players play optimally, hence just by looking at the first few moves made by the players, I can predict whether player1 would win, lose or draw assuming both players make optimal moves. Each sample in the dataset does not represent all the moves made by the players, instead, only a portion of the moves is shown in the sample and it is assumed that the players will make optimal moves and based on that assumption a class label (win/loose/draw) is given to the sample. Each sample can be thought of a sparse matrix of 6x7 dimensions. Intuition says that this should be an easy classification problem and it would be interesting to see how well the learners perform on it.

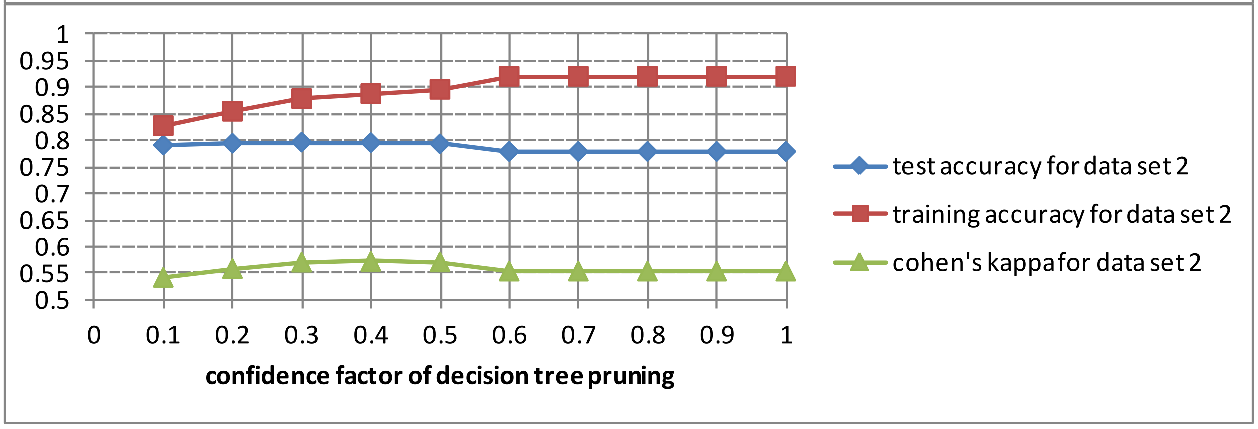
Parameter Optimization

In this section I try to optimize the parameters for each learner.

Decision Tree

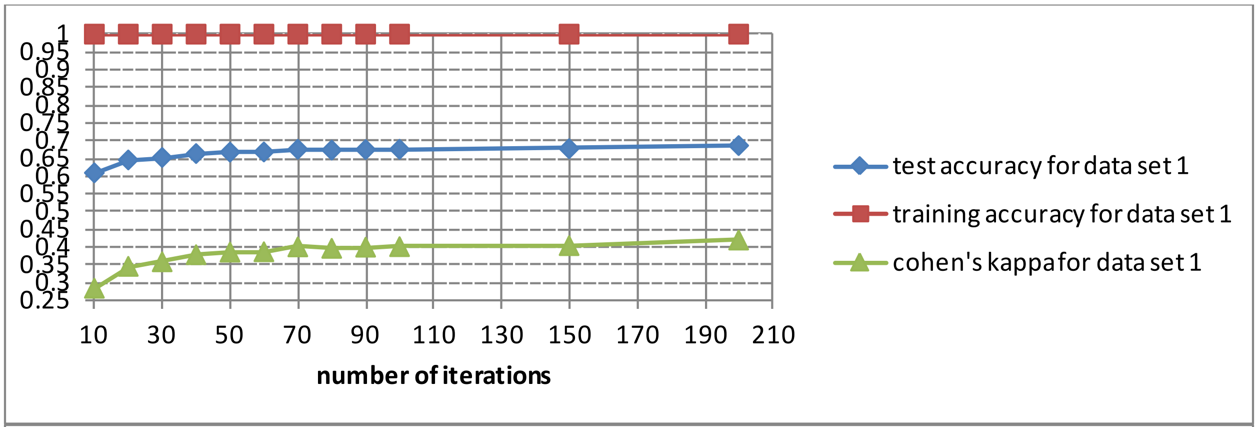
I start with a confidence factor of 0.1 and keep on increasing the confidence factor by 0.1 until I have reached 1.0. I observe that for a very small confidence factor, there is maximum pruning and the training accuracy is very low. I find that for both the data sets the best value observed is **0.3**.

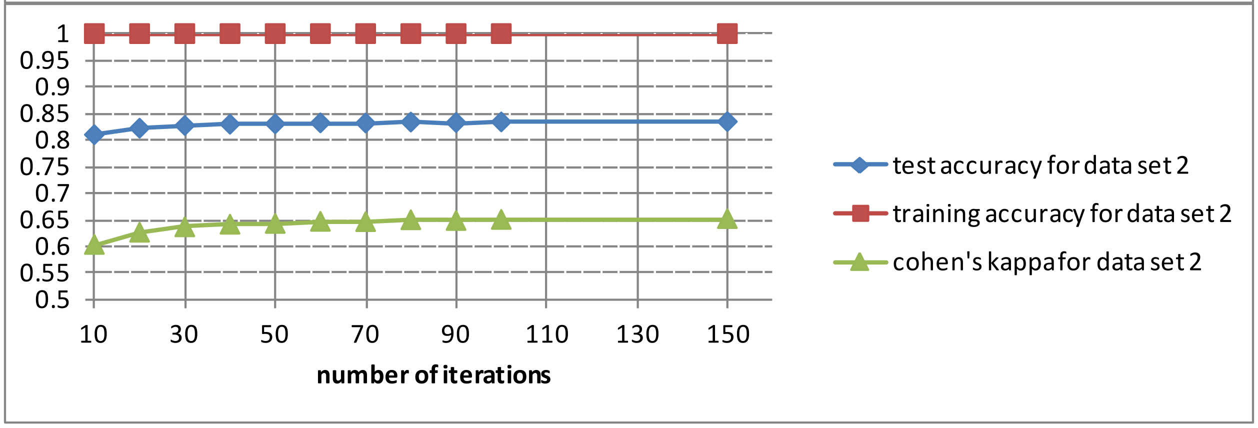




Boosting

I try to find the optimum number of iterations for ADABoost.
I observe that no matter how many training iterations are done, *the accuracy keeps on increasing very slightly each time*. Although, as a downside the overall time of learning also keeps on increasing, for e.g.: *for dataset 2, it took 2 hours (108402437ms) for 150 training iterations*.
In order to keep the learning time and consumed memory manageable, I settled for 200 as the optimum number of iterations on dataset 1 and 150 for data set 2.



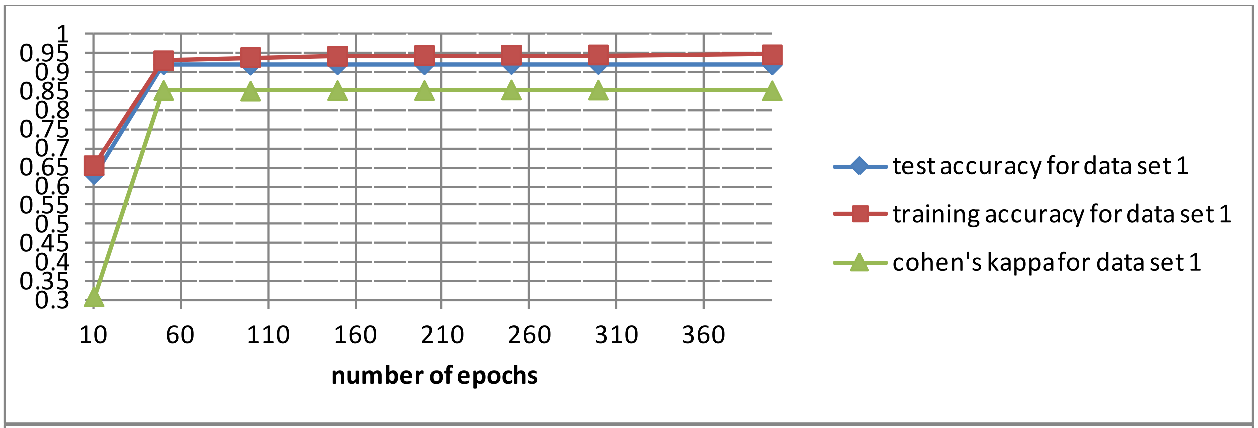


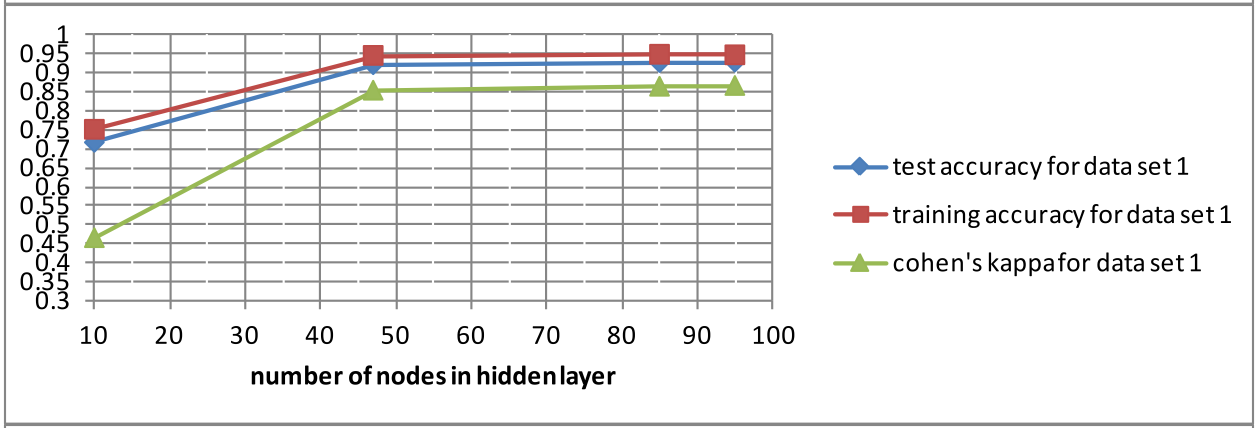
Neural Network

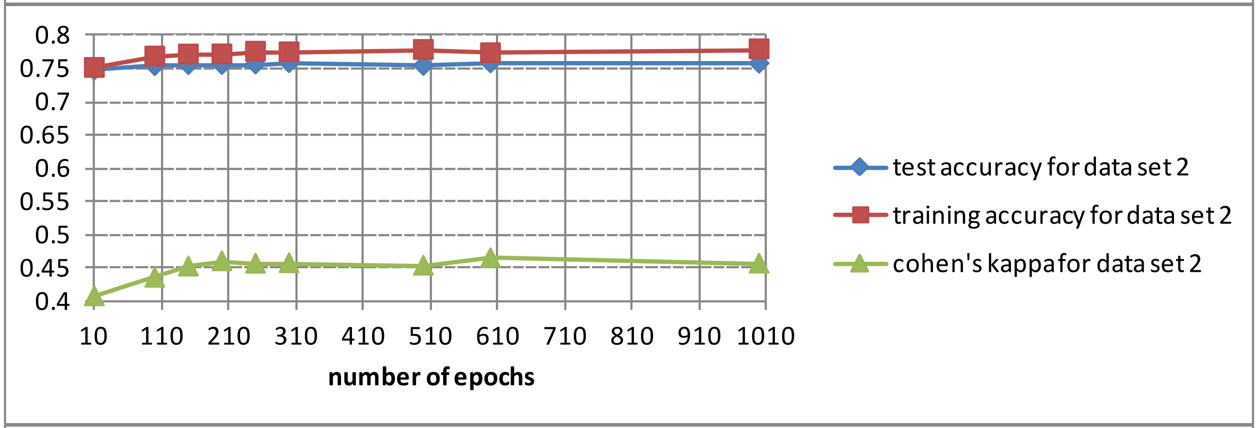
I try to find the optimum number of epochs and the nodes in hidden layer. I start by finding the optimum number of epochs, I set the number of hidden layer nodes to *a*, where '*a*' is the average of the total number of attributes and the number of classes (a_{DS1} is calculated to be 47and a_{DS2} is 55). Once the optimum number of epochs is found, I find the optimum number of hidden layers; I only test for 4 values to see which one generates the best performing learner. The four values are:

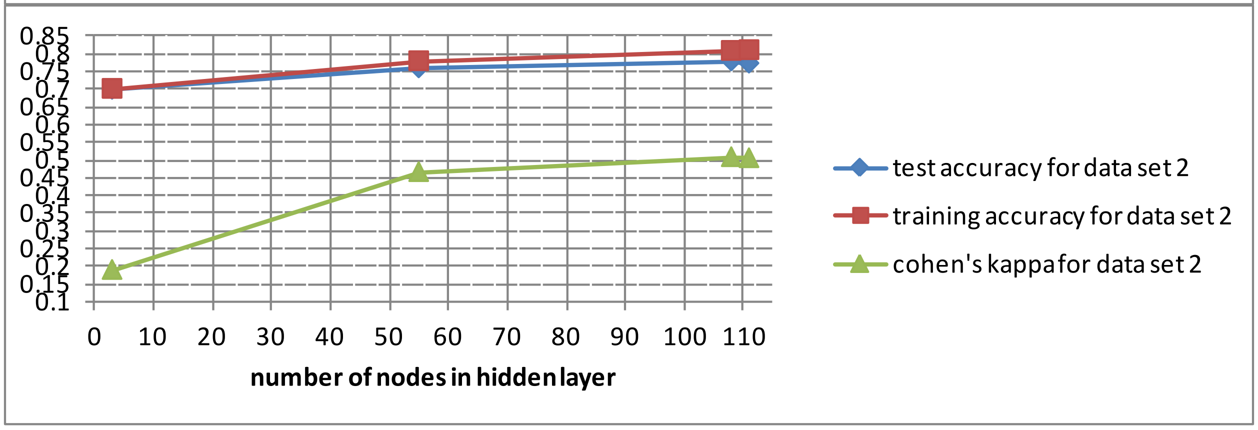
- '*o*': number of classes
- '*i*': number of attributes
- '*a*': ('*i*'+'*o*')/2
- '*t*': '*i*'+'*o*'

For dataset 1, the optimum number of epochs was found to be 250 and the optimum number of nodes in hidden layer was 't' = 95.
For dataset 1, the optimum number of epochs was found to be 300 and the optimum number of nodes in hidden layer was 'i' = 108.









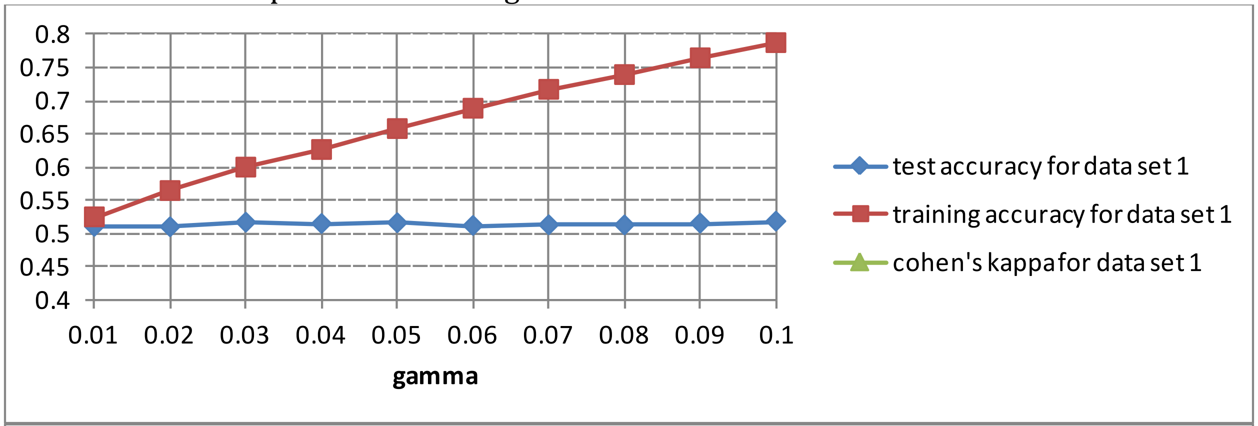
SVM

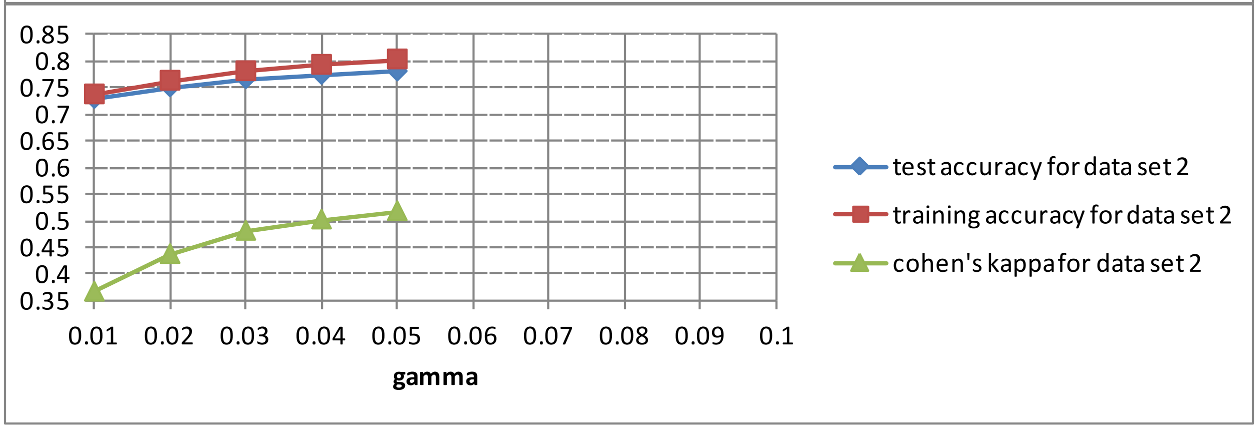
I first test the linear kernel for various parameter values on both the datasets and observe that for dataset1 the accuracy stays below 0.49 and does not improve for any value of C and epsilon, the kappa score also remains 0. For dataset 2, linear kernel performs just as well as the RBF kernel. The RBF kernel's performance on dataset1 seems to vary when different values of gamma, C and epsilon are tried out, therefore it is decided that only RBF kernel will be used in parameter optimization tests since it works better compared to Linear on both the datasets.

I keep the C as 1.0 and try to find the optimized value for gamma; although a "grid-search" on both the parameters would have been a much better method to find the optimized values [1], due to time constraints I leave that as a future step.

For dataset 1, the values for Cohen's kappa are smaller than 0.1 therefore they don't show up on the graph, this means that a random guesser could have performed just as better as SVM for dataset 1. I observe that as the value of gamma increases the training accuracy keeps on increasing, which means that the SVM is somehow correctly classifying more than two classes, I check the *confusion matrix* and confirm that at gamma 0.1, it is able to classify 4 out of the 10 classes with some accuracy (classes 0, 1, 5 and 9). I learn that libSVM implements the "one-against-one" [5] approach which internally splits multiclass problems into multiple datasets with two classes in each dataset and then trains over those datasets. The better value for gamma was found to be 0.03.

In dataset 2, SVM seems to be classifying all three labels with some accuracy, therefore at gamma 0.05, I see Cohen's kappa to be 0.517 and the test accuracy to be 0.78. For gamma 0.06, it takes more than 5 hours to train the learner therefore I stop at 0.05 and consider it as the optimal value for gamma.

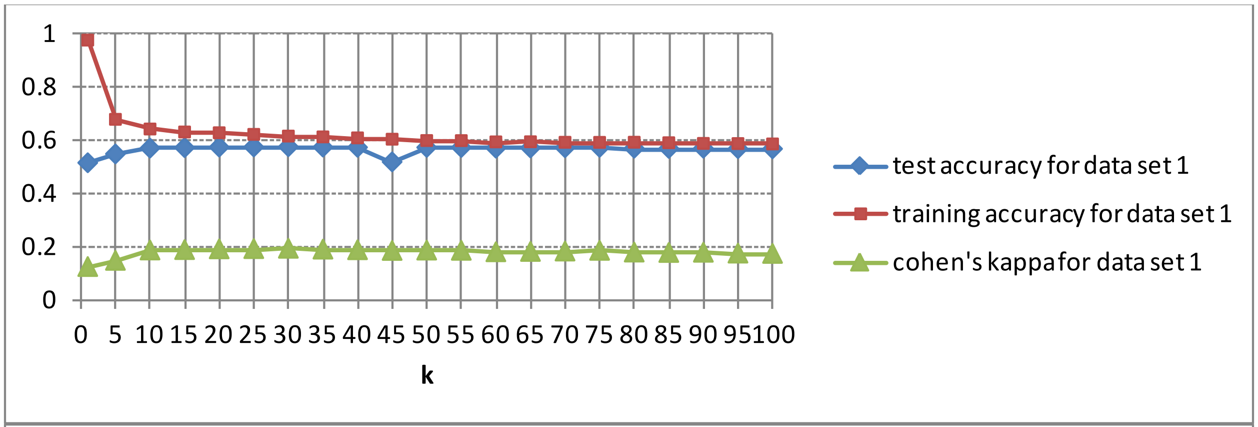


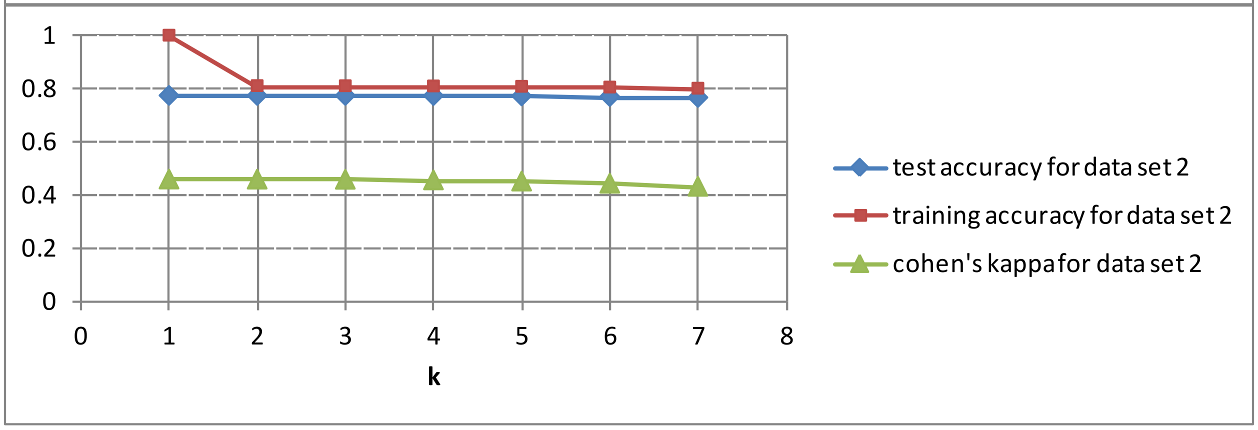


K-NEAREST NEIGHBOR

For dataset 1 the best value of K is found to be 12.

For dataset 2, increasing the k to any number over 1, results in decreased accuracy, therefore k=1 is the only value I can select for optimal performance .

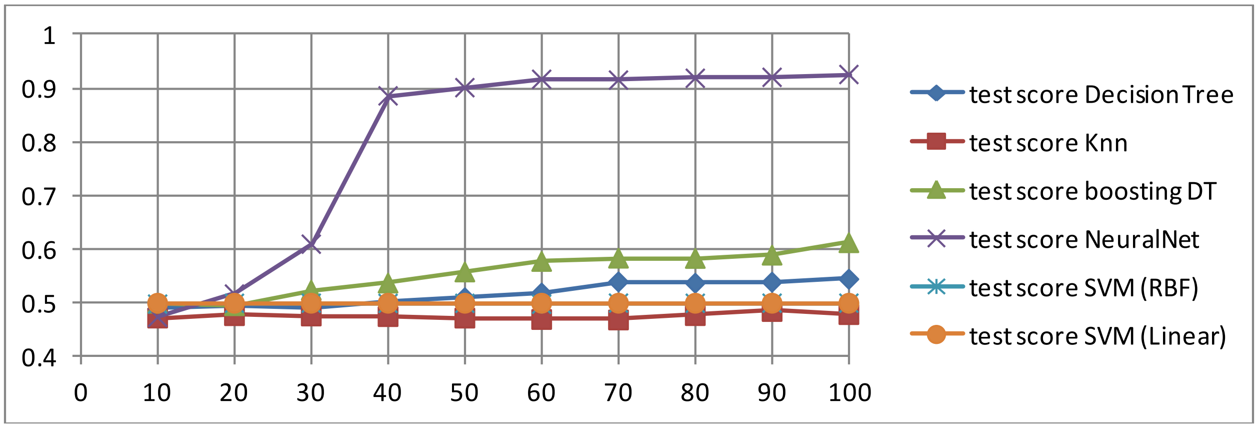


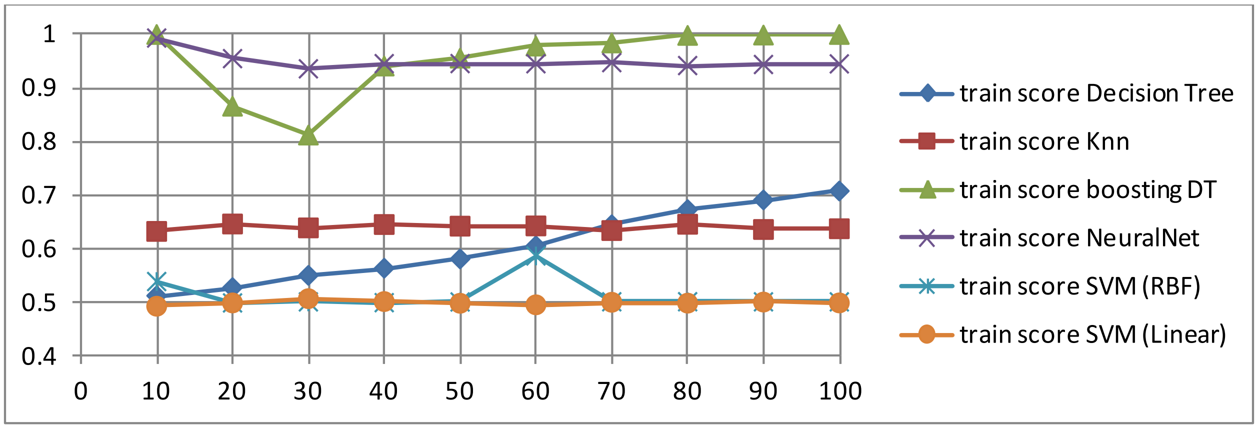


LEARNING CURVE

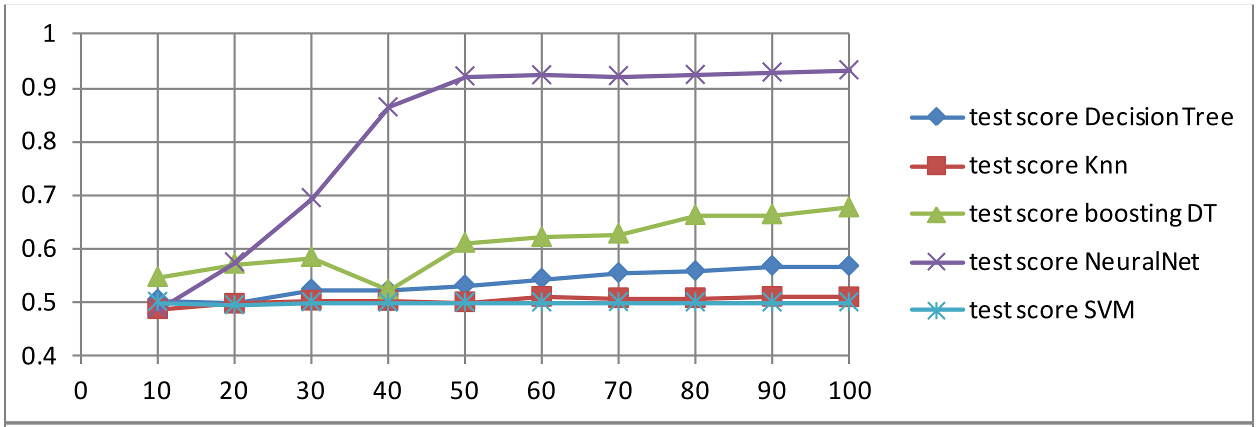
DATA SET 1

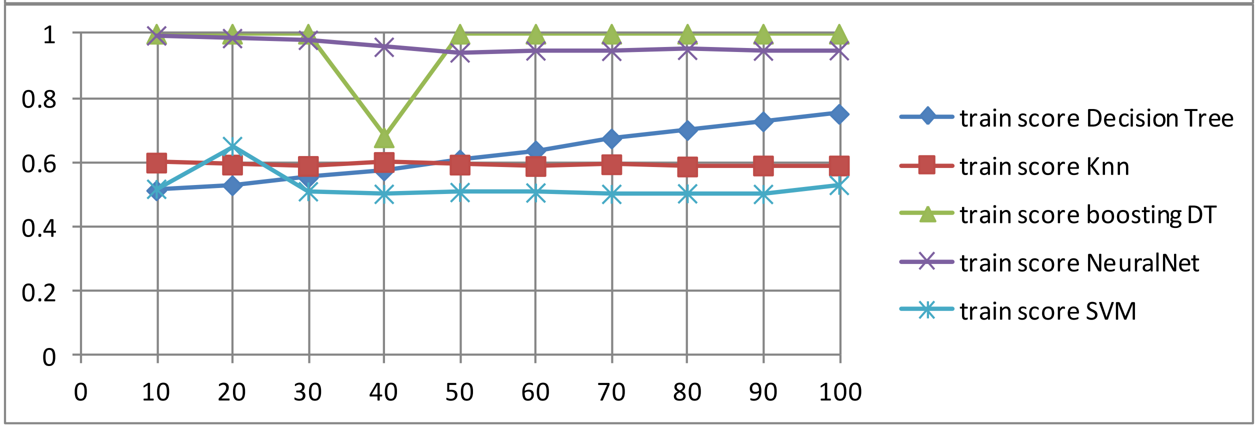
BEFORE PARAMETER OPTIMIZATION





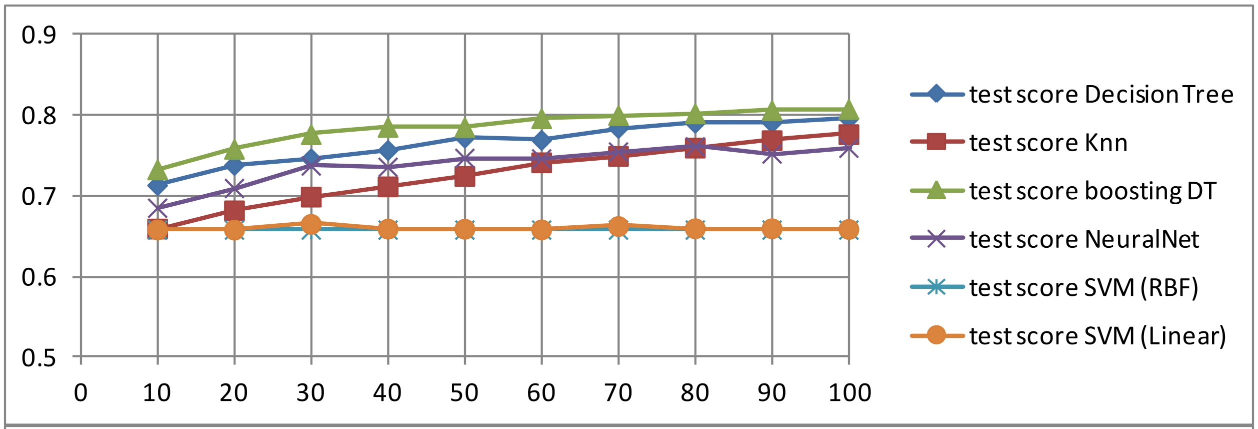
After Parameter Optimization

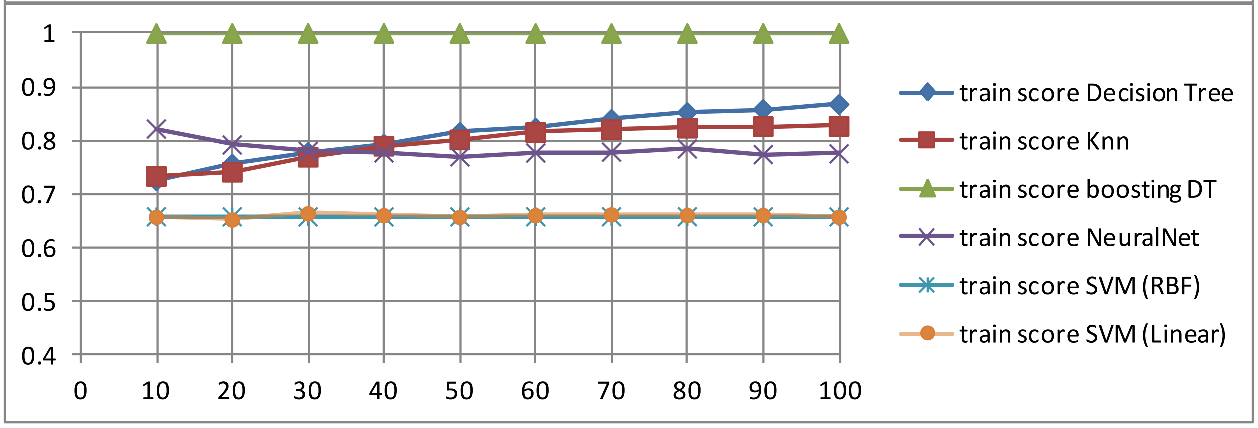




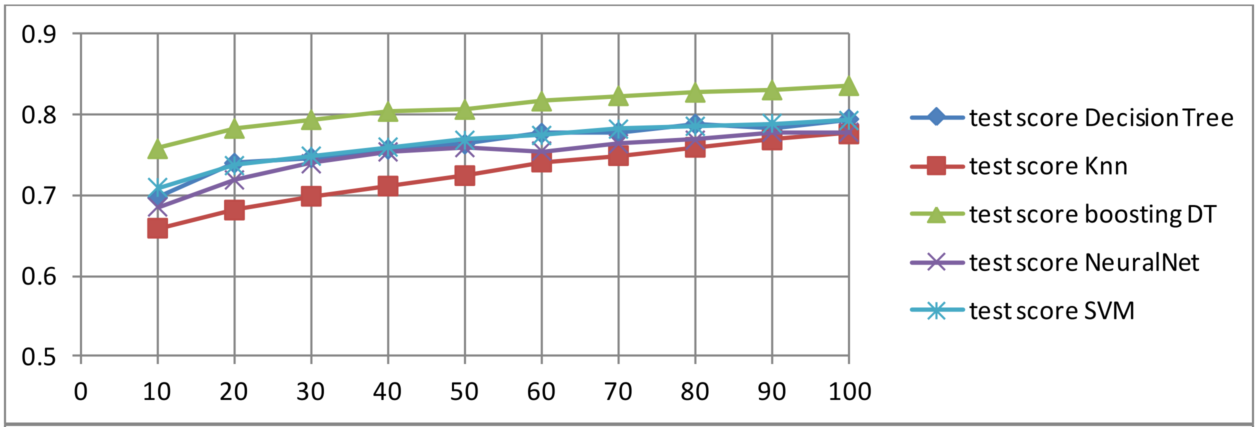
Data Set 2

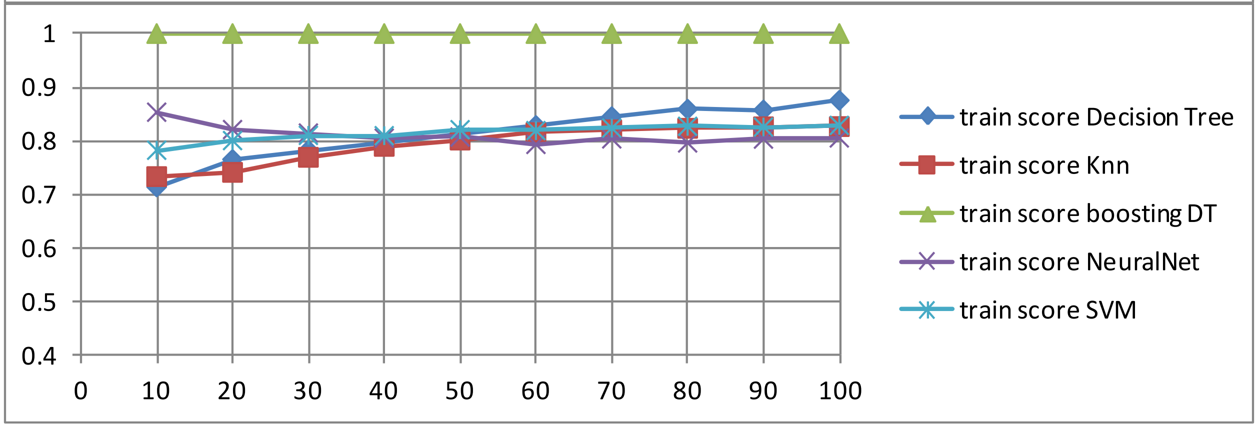
Before Parameter Optimization





After Parameter Optimization





Performance Comparison Table

	Data Set 1 (Poker Hand)				Data Set 2 (Connect-4)			
	Before Optimization		After Optimization		Before Optimization		After Optimization	
	Train score	Testing (accuracy,kappa)	Train score	Testing (accuracy,kappa)	Train score	Testing (accuracy,kappa)	Train score	Testing (accuracy,kappa)
Decision Tree	0.707	0.551, 0.151	0.751	0.567, 0.184	0.867	0.796, 0.566	0.875	0.795, 0.566
Boosting	0.999	0.612, 0.28	1	0.676, 0.401	1	0.807, 0.600	1	0.836, 0.655
Neural Network	0.944	0.925, 0.863	0.948	0.931, 0.874	0.775	0.759, 0.437	0.805	0.778, 0.505
K-NN	0.937	0.478, 0.02	0.591	0.509, 0.067	0.826	0.728, 0.302	0.826	0.728, 0.302
SVM(RBF)	0.601	0.499, 0.0009	0.530	0.499, 0.005	0.658	0.658, 0.0	0.828	0.793, 0.548
SVM(Lin)	0.499	0.499, 0.0	-	-	0.657	0.659, 0.0	-	-

Learner Configuration				
	Data Set 1 (Poker Hand)		Data Set 2 (Connect-4)	
	Before Optimization	After Optimization	Before Optimization	After Optimization
Decision Tree	binarySplits=false, confidenceFactor=0.25, minNumObj=2, reducedErrorPruning=false, seed=1, subTreeRaising=true, unPruned=false, useLaplace-false	binarySplits=false, confidenceFactor=0.3, minNumObj=2, reducedErrorPruning=false, seed=1, subTreeRaising=true, unPruned=false, useLaplace-false	binarySplits=false, confidenceFactor=0.25, minNumObj=2, reducedErrorPruning=false, seed=1, subTreeRaising=true, unPruned=false, useLaplace-false	binarySplits=false, confidenceFactor=0.3, minNumObj=2, reducedErrorPruning=false, seed=1, subTreeRaising=true, unPruned=false, useLaplace-false
Boosting	Classifier=J48 decision tree with the above	Classifier=J48 decision tree with the above	Classifier=J48 decision tree with the above	Classifier=J48 decision tree with the above

	configuration, numIterations=10, seed=1, useResampling=false, weightThreshold= 100	configuration, numIterations=150, seed=1, useResampling=false, weightThreshold= 100	configuration, numIterations=10, seed=1, useResampling=false, weightThreshold= 100	configuration, numIterations=150, seed=1, useResampling=false, weightThreshold= 100
Neural Network	autoBuild=true, decay=false, hiddenLayers=a, learningRate=0.3, momentum=0.2, nominalToBinaryFilter=true, normalizeAttributes=true, normalizeNumericClass=true, reset=true, trainingTime=500, validationSetSize=0, validationThreshold=20	autoBuild=true, decay=false, hiddenLayers=t, learningRate=0.3, momentum=0.2, nominalToBinaryFilter=true, normalizeAttributes=true, normalizeNumericClass=true, reset=true, trainingTime=250, validationSetSize=0, validationThreshold=20	autoBuild=true, decay=false, hiddenLayers=a, learningRate=0.3, momentum=0.2, nominalToBinaryFilter=true, normalizeAttributes=true, normalizeNumericClass=true, reset=true, trainingTime=500, validationSetSize=0, validationThreshold=20	autoBuild=true, decay=false, hiddenLayers=i, learningRate=0.3, momentum=0.2, nominalToBinaryFilter=true, normalizeAttributes=true, normalizeNumericClass=true, reset=true, trainingTime=300, validationSetSize=0, validationThreshold=20
K-NN	K=3, weight neihbours by distance= false, output class probabilities= false	K=12, weight neihbours by distance= false, output class probabilities= false	K=3, weight neihbours by distance= false, output class probabilities= false	K=1, weight neihbours by distance= false, output class probabilities= false
SVM(RBF)	Type of SVM=C-SVC, Kernel=RBF, gamma=0.0, CacheSize=973, Epsilon=0.001, Shrinking=true, Probability estimates=true	Type of SVM=C-SVC, Kernel=RBF, gamma=0.03, CacheSize=973, Epsilon=0.001, Shrinking=true, Probability estimates=true	Type of SVM=C-SVC, Kernel=RBF, gamma=0.0, CacheSize=973, Epsilon=0.001, Shrinking=true, Probability estimates=true	Type of SVM=C-SVC, Kernel=RBF, gamma=0.05, CacheSize=973, Epsilon=0.001, Shrinking=true, Probability estimates=true
SVM(Linear)	Type of SVM=C-SVC, Kernel=Linear, Cost=0.5, CacheSize=172, Epsilon=0.001, Shrinking=true, Probability estimates=true	-	Type of SVM=C-SVC, Kernel=Linear, Cost=0.5, CacheSize=172, Epsilon=0.001, Shrinking=true, Probability estimates=true	-

CONCLUSION

For dataset 1, only neural network was able to perform well enough; it is observed from the confusion matrix that decision tree is classifying most of the labels as 0, that's because there are 49.95% samples for class 0 in the dataset; this tends to create a bias in the decision tree. Boosting the decision tree improves performance but it still is unable to completely remove the underlying bias. This bias in the Poker Dataset can be overcome by attaching weights/costs to the under-represented classes [4].

For dataset 2, the boosted decision tree performed the best. The kappa score of SVM improved from 0 to 0.548 after optimizing gamma.

SVM performed very slow on dataset 2 as compared to dataset 1 , this could be due to the difference in the number of dimensions of both the sets , dataset1 only has 10 dimensions , where as dataset2 has 42. It took 54.36 minutes (2721619ms) for SVM to learn dataset2 for gamma of 0.05.

K-NN performed the worse for both the datasets. That's because in both the datasets the samples don't relate to each other in the way the algorithm tries to calculate the nearest neighbor. For e.g., in

dataset 1, K-NN might find the following two samples as the nearest neighbors, but in terms of their actual distance according to the rules of poker, they are not nearest neighbors:
1,3,4,7,1,5,2,4,4,2
1,4,1,6,1,3,3,5,3,2

REFERENCES CITED

[1] CW Hsu, CC Chang, CJ Lin. A practical guide to support vector classification

[2] Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] R. Cattral, F. Oppacher, D. Deugo. Evolutionary Data Mining with Automatic Rule Generalization. Recent Advances in Computers, Computing and Communications, pp.296-300, WSEAS Press, 2002.

[4] Ting, K.M.: An instance-weighting method to induce cost-sensitive trees. TKDE 14(3), 659–665 (2002)

[5] LIBSVM: A Library for Support Vector Machines Chih-Chung Chang and Chih-Jen Lin Department of Computer Science National Taiwan University, Taipei, Taiwan Email: cjlin@csie.ntu.edu.tw Initial version: 2001 Last updated: March 4, 2013