CAP 6610 Spring 2014

Machine Learning Project

Ishadutta Yadav
iyadav@cise.ufl.edu
UFID : 54931916

Harsh Tarang Yadava
htarang@cise.ufl.edu
UFID : 03610017

Ximing Wang
x.wang@ufl.edu
UFID : 23869171

Sanchit Gupta
sanchit@cise.ufl.edu
UFID : 59339993

1. Data Sets

Data Set	Number of Classes	Number of Features	Size of Dataset	Size of Dataset After Pruning (if any)	Different Training Percents trials
Breast Cancer Wisconsin (Original)	2	9	699	683	10%, 30%, 50%
Optical recognition	10	64	5620	5620	10%, 30%, 50%
HIGGS	2	28	11,000,000	100,000 (randomly picked)	10%, 30%, 50%

Dataset Analysis

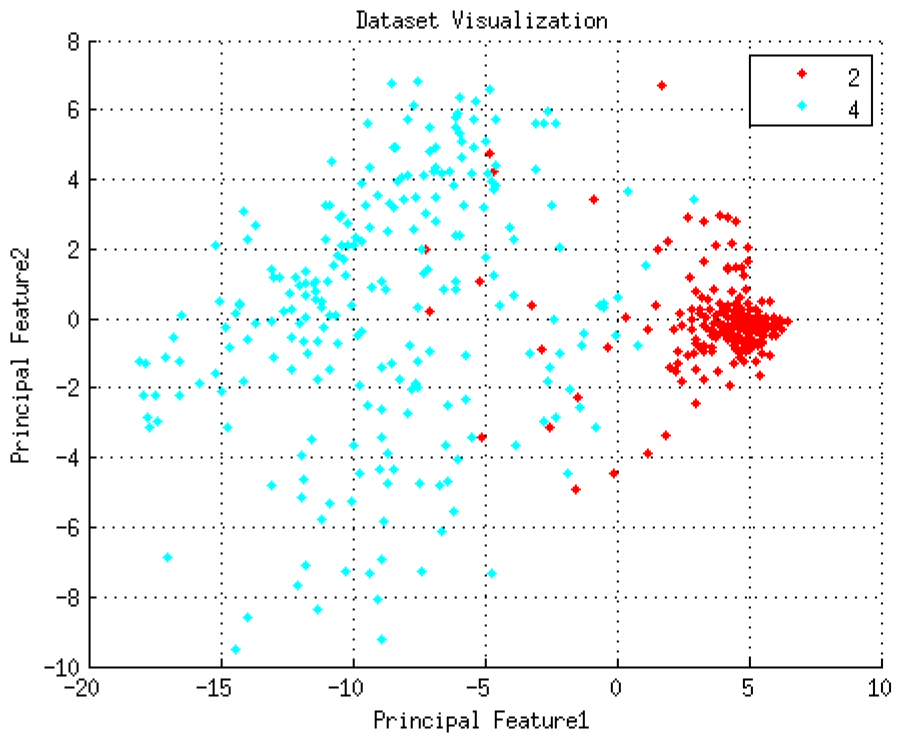
Principal Component Analysis (PCA)

Principal component analysis is a quantitatively rigorous method for achieving dimensionality reduction. The method generates a new set of variables, called *principal components*. Each principal component is a linear combination of the original variables. All the principal components are orthogonal to each other, so there is no redundant information. The principal components as a whole form an orthogonal basis for the space of the data**.**

We ran PCA for each of the datasets and took two principal features from it.
Below is the depiction of the principal features for each data set.

A) Breast Cancer Wisconsin (Original) -

From the following visualization we can infer that the data is mostly linearly separable



Dataset Details

Number of Instances: 699 (as of 15 July 1992)

Number of Attributes: 10 plus the class attribute (1st is sample id, not feature)

Attribute Information: (class attribute has been moved to last column)

#	Attribute	Domain
1.	Sample code number	id number
2.	Clump Thickness	1 - 10
3.	Uniformity of Cell Size	1 - 10
4.	Uniformity of Cell Shape	1 - 10
5.	Marginal Adhesion	1 - 10
6.	Single Epithelial Cell Size	1 - 10
7.	Bare Nuclei	1 - 10
8.	Bland Chromatin	1 - 10
9.	Normal Nucleoli	1 - 10
10.	Mitoses	1 - 10
11.	Class:	(2 for benign, 4 for malignant)

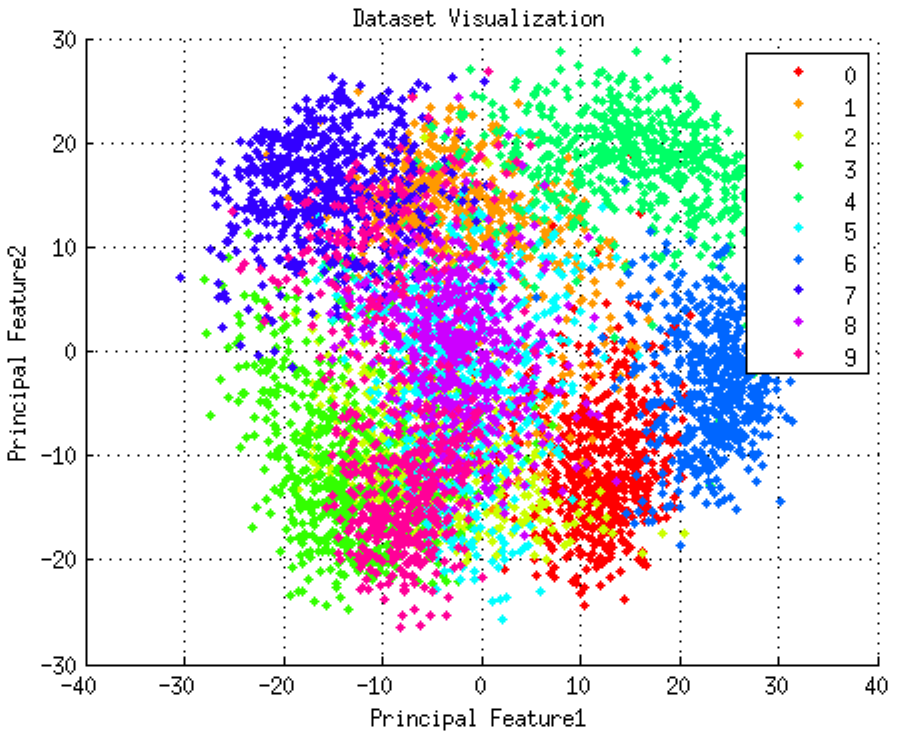
Missing attribute values: 16

There are 16 instances in Groups 1 to 6 that contain a single missing (i.e., unavailable) attribute value, now denoted by "?".

Class distribution:
Benign: 458 (65.5%)
Malignant: 241 (34.5%)

2) Optical recognition

From the following visualization we can infer that the data is more or less clustered for each class.



Data details

Number of Instances : 5620
Number of Attributes : 64
For Each Attribute:

All input attributes are integers in the range 0..16.
The last attribute is the class code 0..9

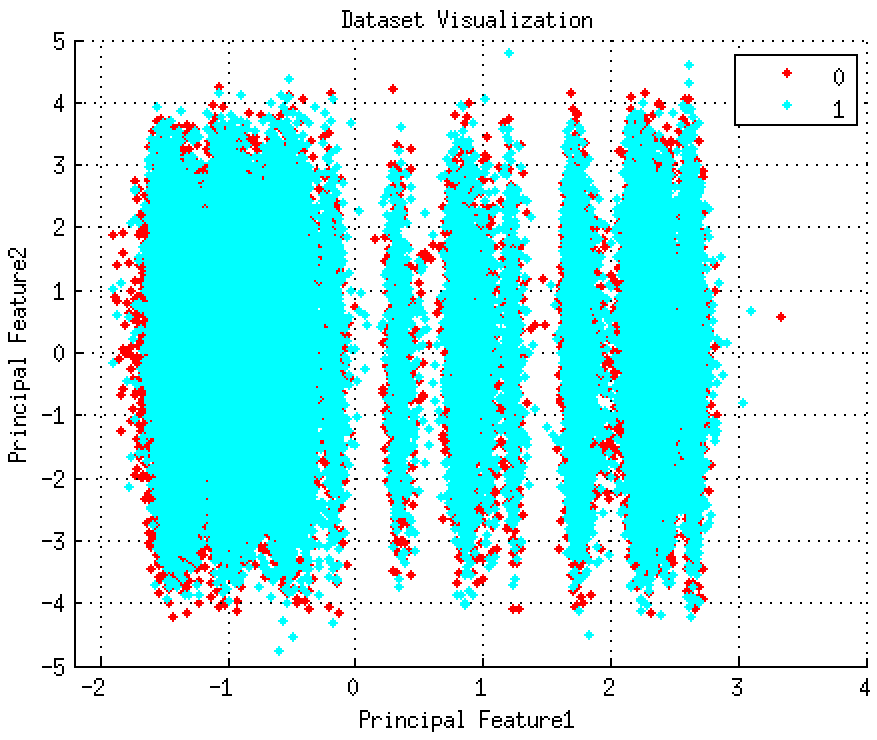
Class Distribution

Class:	No of examples in training set
0:	376
1:	389
2:	380
3:	389
4:	387

5: 376
6: 377
7: 387
8: 380
9: 382

3) Higgs Dataset Visualization

From the following dataset we can see that data is not linearly separable and one of the classes resides on the boundary of the other class.



Number of Instances : 1100000
Number of Attributes : 21
Class distribution : 50% each

Note : Since the data set was huge and MATLAB was unable to process the whole data(throwing out of memory error) we created 10 random samples of the data set, each containing 100,000 entries. We ran all the algorithms over these 10 data sets and computed the average accuracy over them.

2. Algorithms

Algorithm	**Platform**	**Library**	**Function/Algorithm**
SVM	Matlab	LIBSVM	svmtrain
Deep Learning	Matlab	DeepLearnToolbox https://github.com/rasmusbergpalm/ DeepLearnToolbox	neural network stack- autoencoder
Random Forest	Matlab	https://code.google.com/p/randomfo rest-matlab/	train_rf
Ensemble Methods	Matlab	Statistical Toolbox	fitensemble
K Nearest Neighbour	Matlab	Statistical Toolbox	ClassificationKNN
Naive Bayes	Matlab	Statistical Toolbox	NaiveBayes

2.1 SVM

Introduction

The soft margin SVM allows for mislabeled samples. It introduces non-negative slack variables ξ_i to measure the distance of within-margine or misclassified data x_i to the hyperplane with the correct label. The objective function is then adding a term that penalizes these slack variables, and the optimization is a trade off between a large margin and a small error penalty. The soft margin SVM with linear kernel is:

$$\arg\min_{\mathbf{w},\xi,b} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \right\}$$

$$\text{s.t.} \quad y_i(\mathbf{w}\cdot\mathbf{x_i} - b) \geq 1 - \xi_i, \qquad \xi_i \geq 0$$

Besides linear kernel $k(x_i,x_j) = x_i \cdot x_j$, nonlinear kernels are also introduced into SVM to create nonlinear classifiers. The maximum-margin hyperplane is constructed in a transformed fearture space with a possible nonlinear transformation and high-dimensional transformed space, therefore, it could be nonlinear in the original feature space. A widely used nonliear kernel is Gaussian radial basis function $k(x_i,x_j) = exp(-\gamma\|x_i - x_j\|^2)$. It corresponds to a Hilbert space of infinite dimensions.

Experiment with SVM

MATLAB has its own code for SVM, but only allows for binary classification. LIBSVM provides the matlab code for multi-class support vector classification. As the second data set, optical recognition of handwritten digits data has 10 classes, for the consideration of consistance, LIBSVM is used in our project.

Division of Data

10% Training, 90% Testing
30% Training, 70% Testing
50% Training, 50% Testing

Kernels

Linear kernel $k(x_i, x_j) = x_i \cdot x_j$
Gaussian kernel $k(x_i, x_j) = exp(-\gamma\|x_i - x_j\|^2)$

Cross Validation

5-fold (80/20 percentage division) cross validation is used to get the optimal parameter values. The training data set is divided equally into 5 folds, then with each previously given parameter value, a classifier is constructed based on 4 folds and the remaining 1 fold is the validation set to test the performance of the classifier with this parameter value. This test keeps 5 times to get the average performance. Then compare the average performance with difference parameter values and pick the best one as the parameter to train the whole training data set and test on the testing data set to get the performance of the algorithm.

Linear kernel, the only parameter is the multiplier C of penalty on slack variables:

C **varies from** 2^{-10} **to** 2^{10}

For Gaussian kernel, besides C, the parameters also include in the Gaussian kernel:

C **varies from** 2^{-10} **to** 2^{10}
γ **varies from** 2^{-10} **to** 2^{10}

Multi-Class SVM

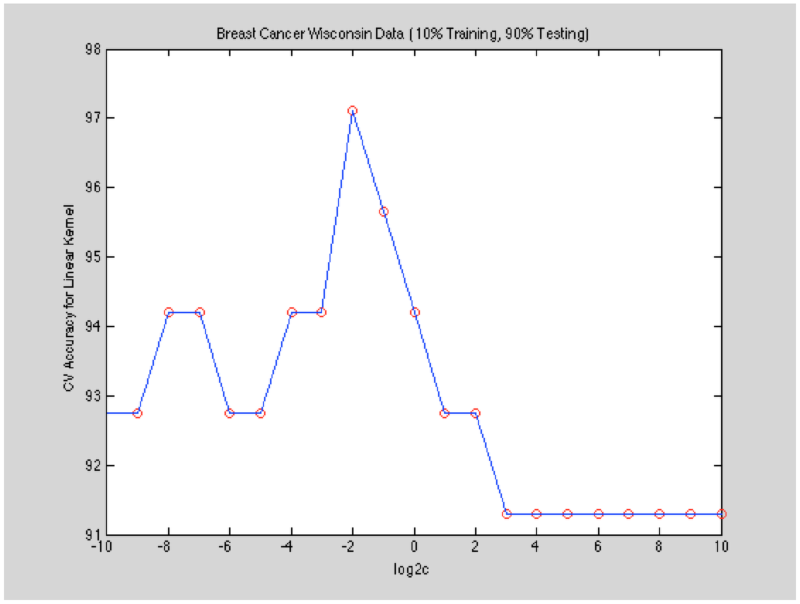
LIBSVM uses "one-against-one" for multi-class SVM, so when there are n classes, a total of n(n-1)/2 binary classifiers are constructed. In our project, the optical recognition of handwritten digits data has 10 classes, therefore in each run, there would be 45 classifiers. For the parameter selection, to avoid over-fitting, the same parameters are used for all n(n-1)/2 binary classification problems.

Experiment Results

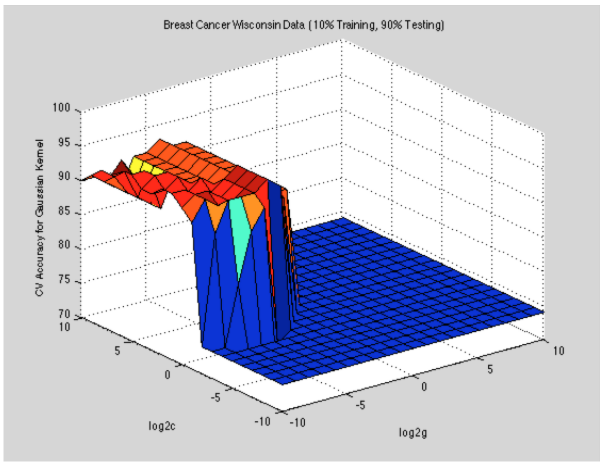
<u>Breast Cancer Wisconsin Data (2 classes)</u>

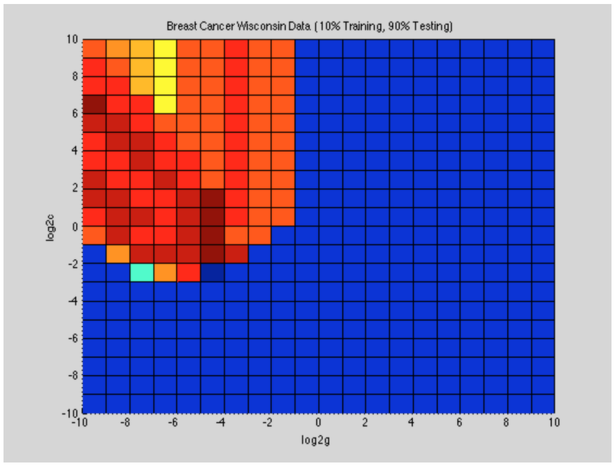
10% Training, 90% Testing

Linear Kernel: best log2c=-2, c=0.25, cv acc rate=97.1014
Accuracy on testing set: 95.1140



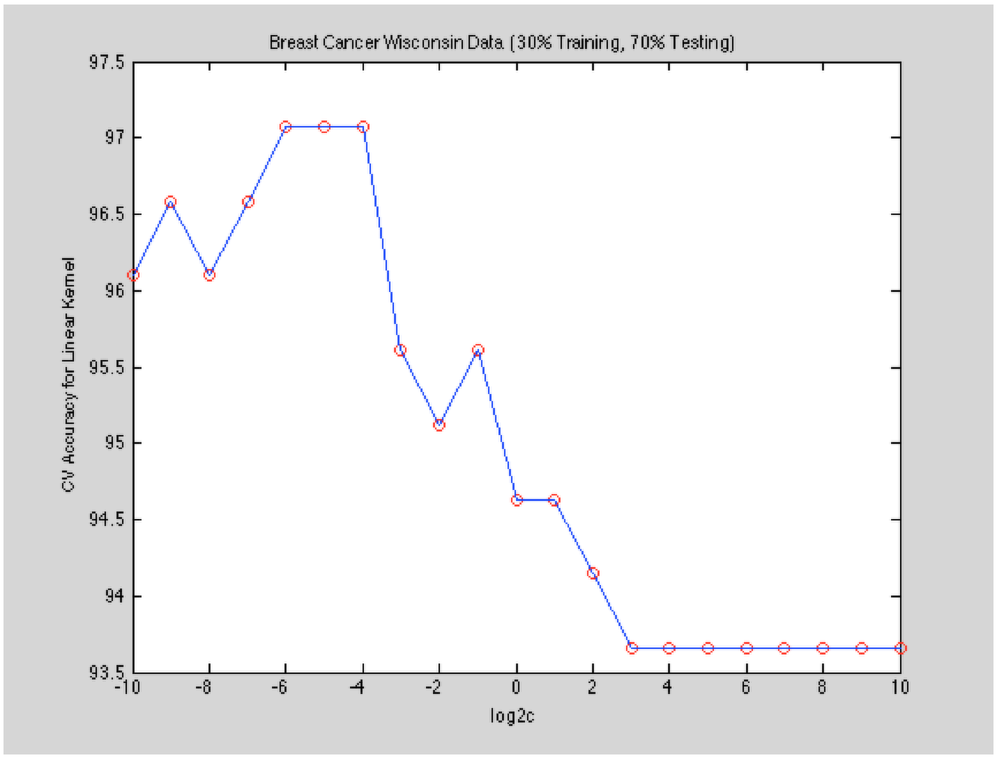
Gaussian kernel: best log2c=6, c=64, log2g=-10, g=0.000976562, cv acc rate=95.6522
Accuracy on testing set: 95.4397



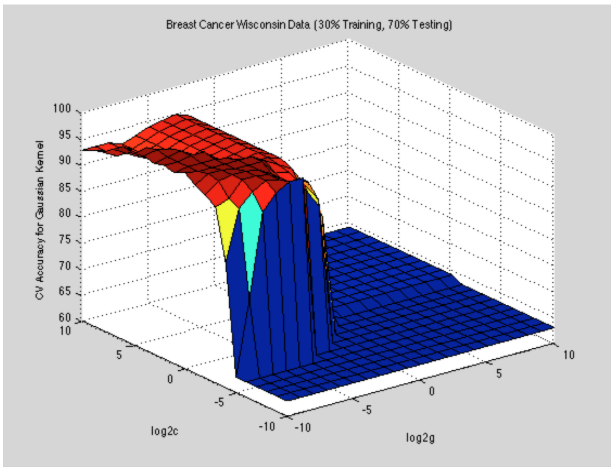


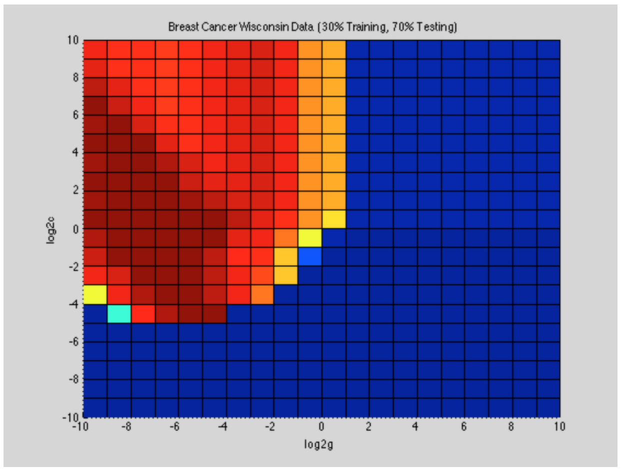
30% Training, 70% Testing

Linear Kernel: best log2c=-4, c=0.0625, cv acc rate=97.0732
Accuracy on testing set: 96.8619



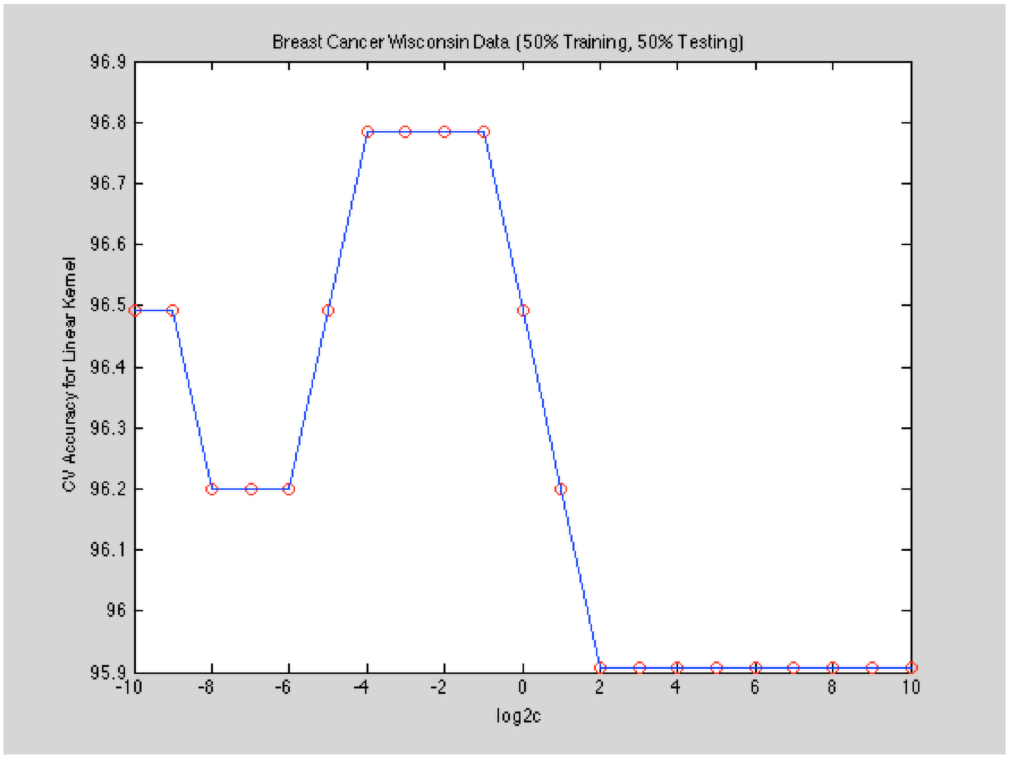
Gaussian kernel: best log2c=4, c=16, log2g=-9, g=0.00195312, cv acc rate=97.561
Accuracy on testing set: 97.0711



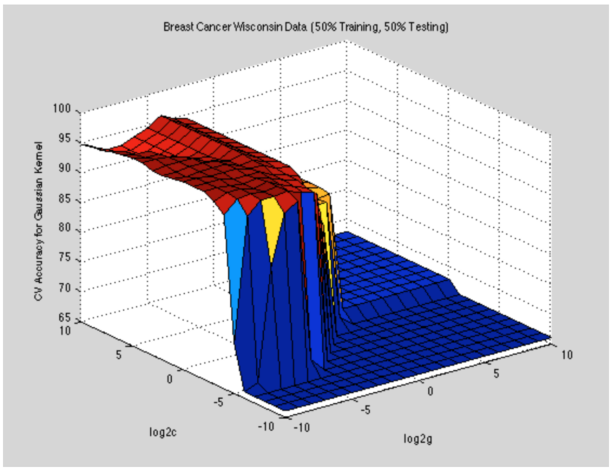


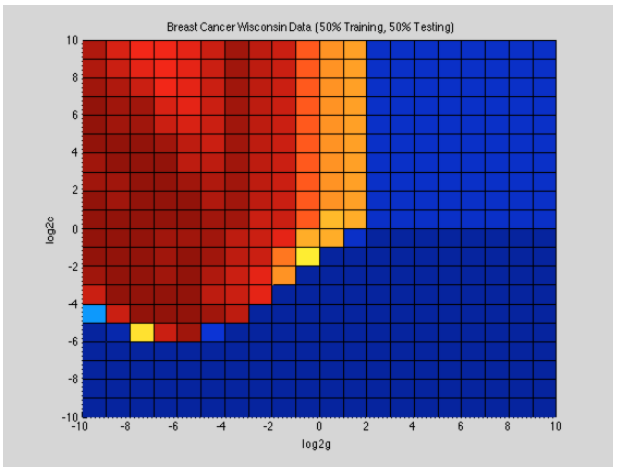
50% Training, 50% Testing

Linear Kernel: best log2c=-1, c=0.5, cv acc rate=96.7836
Accuracy on testing set: 96.7742



Gaussian kernel: best log2c=5, c=32, log2g=-10, g=0.000976562, cv acc rate=96.4912
Accuracy on testing set: 97.3607

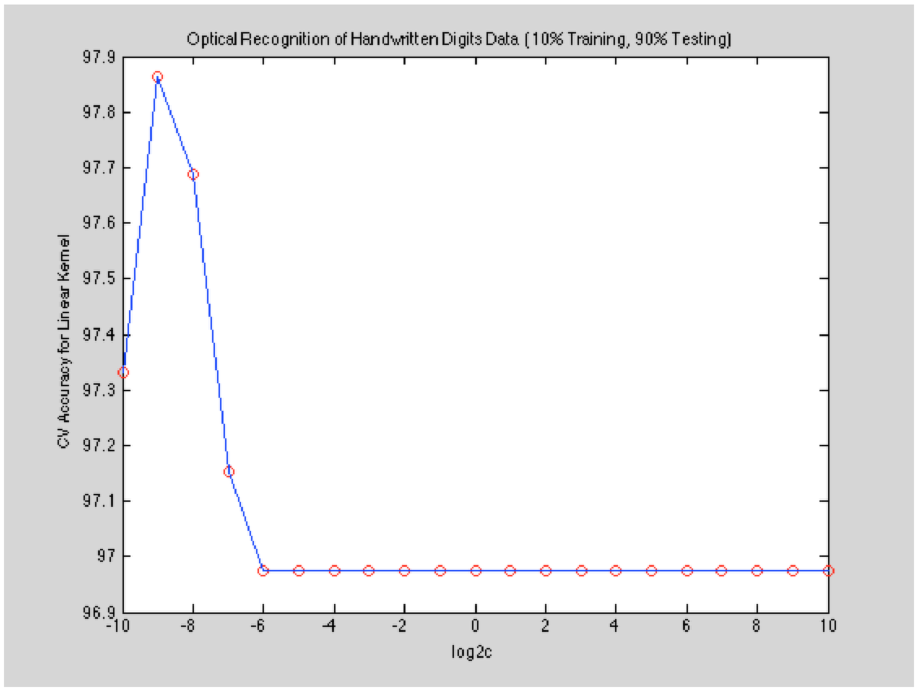




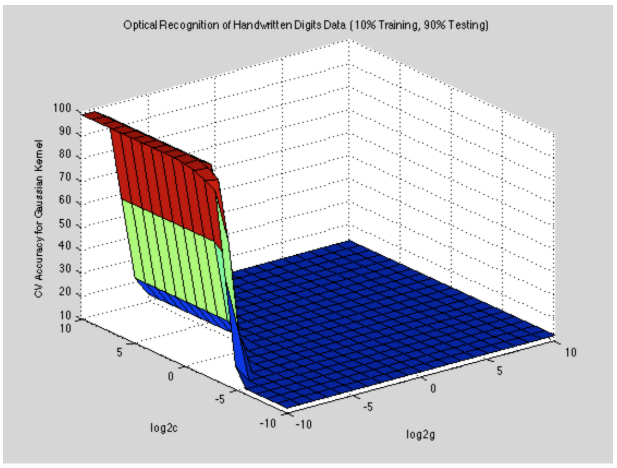
<u>Optical Recognition of Handwritten Digits Data (10 classes)</u>

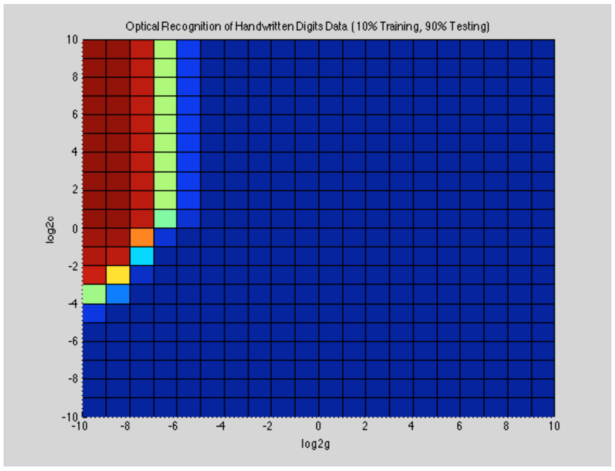
10% Training, 90% Testing

Linear Kernel: best log2c=-8, c=0.00390625, cv acc rate=95.0178
Accuracy on testing set: 95.8679



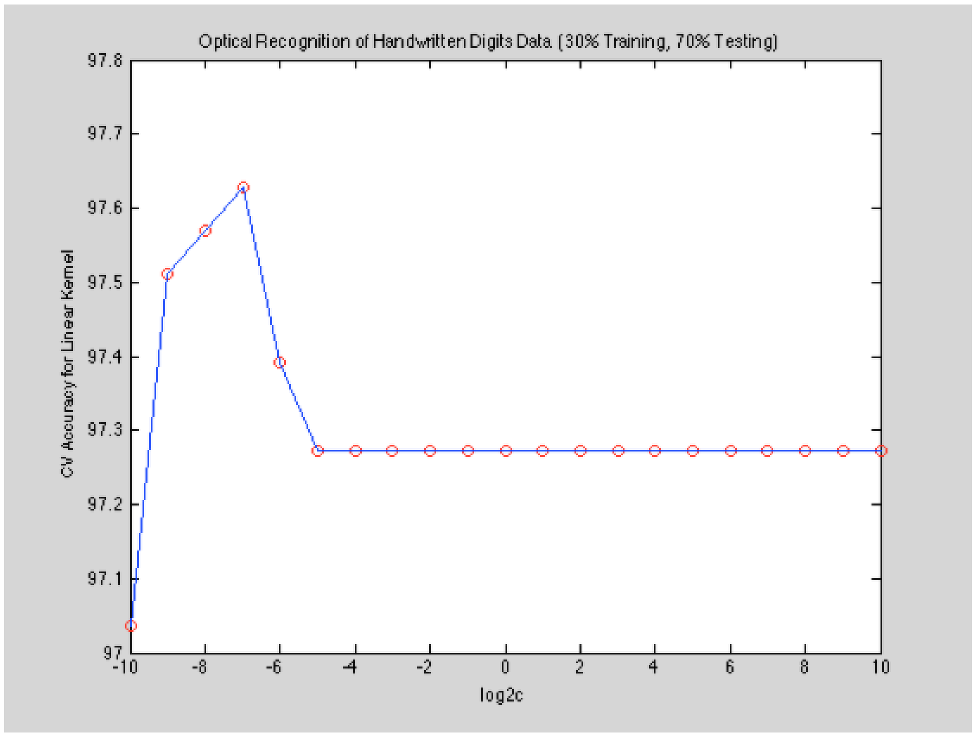
Gaussian kernel: best log2c=10, c=1024, log2g=-10, g=0.000976562, cv acc rate=95.9075
Accuracy on testing set: 97.5484



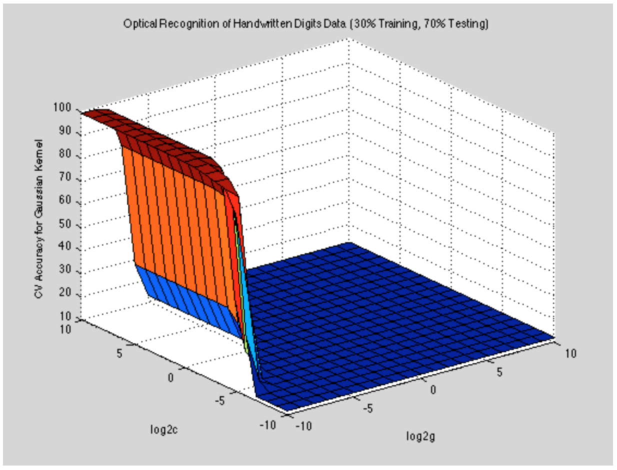


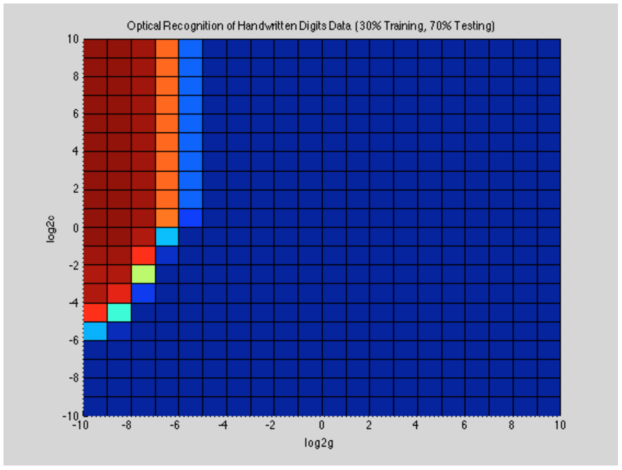
30% Training, 70% Testing

Linear Kernel: best log2c=-7, c=0.0078125, cv acc rate=97.6289
Accuracy on testing set: 97.5591



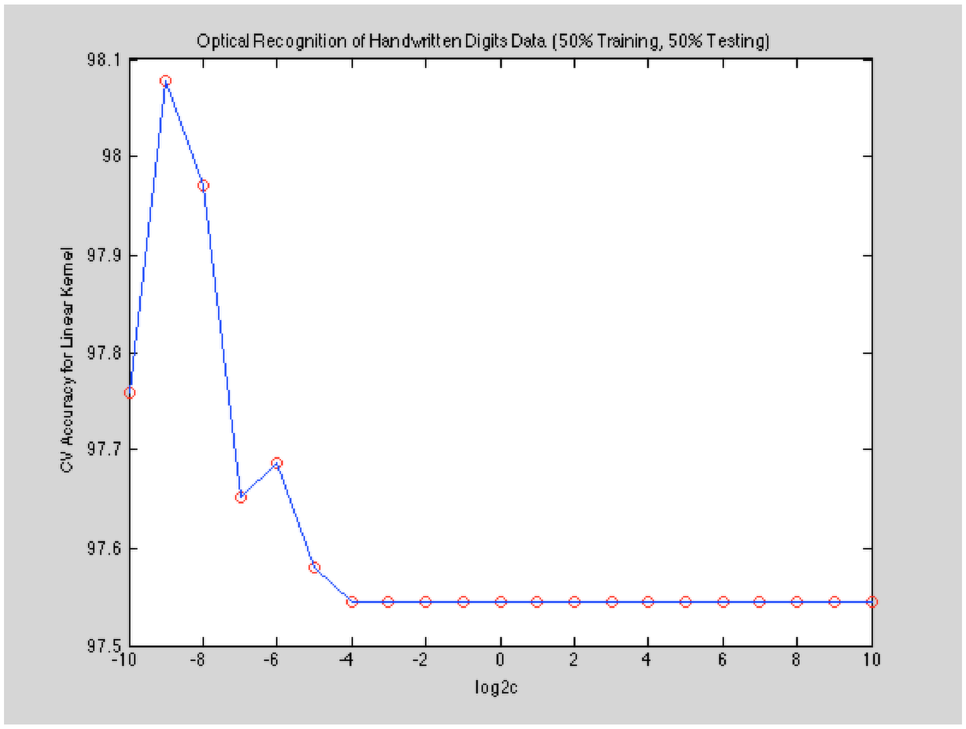
Gaussian kernel: best log2c=10, c=1024, log2g=-9, g=0.00195312, cv acc rate=98.8737
Accuracy on testing set: 98.7541



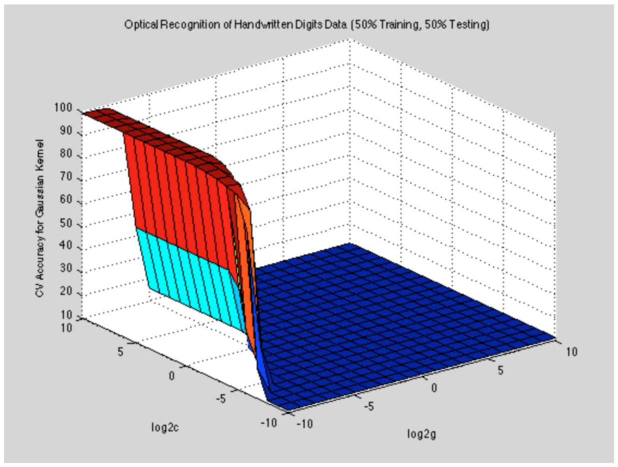


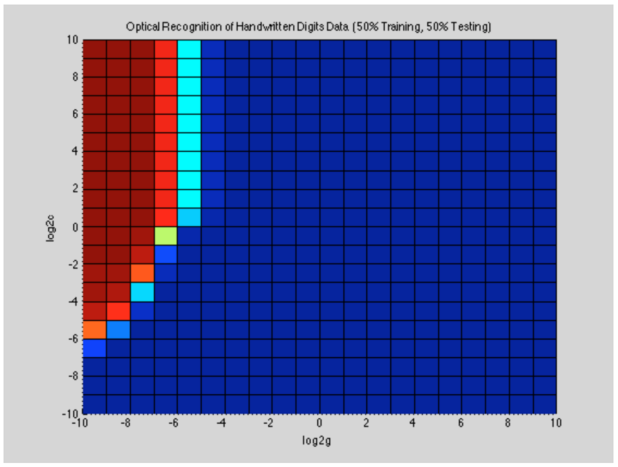
50% Training, 50% Testing

Linear Kernel: best log2c=-9, c=0.00195312, cv acc rate=98.0783
Accuracy on testing set: 97.7936



Gaussian kernel: best log2c=10, c=1024, log2g=-10, g=0.000976562, cv acc rate=98.9324
Accuracy on testing set: 99.0391

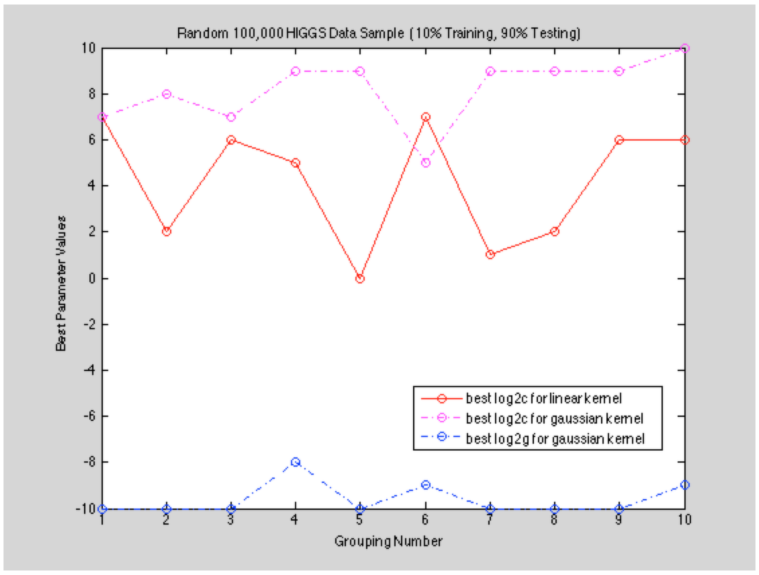


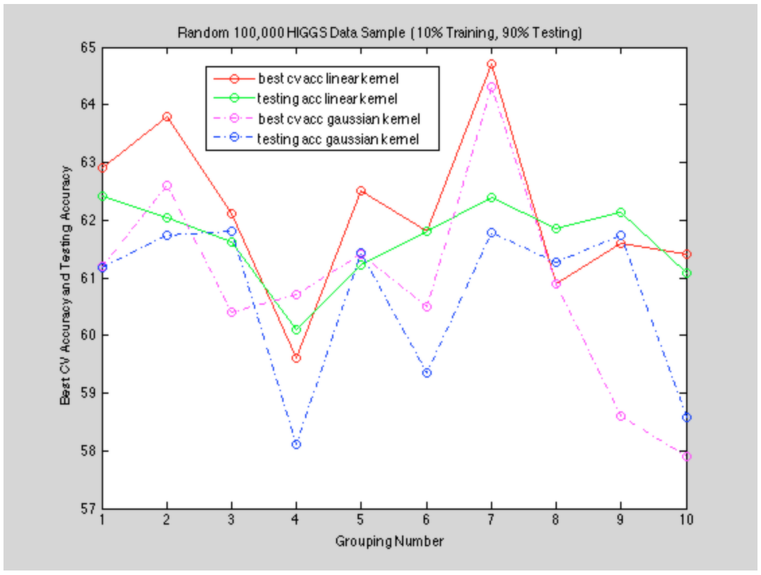


Random 100,000 HIGGS Data Samples (2 classes)

We randomly pick 100,000 samples from the original data file. However, performing SVM on this sample set still cost too much time for even one run in cross validation, therefore too long to run the whole cross validation for different parameter values and for both linear and Gaussian kernels. To achieve a result, these 100,000 samples are randomly divided into 10 groups when each group has 10,000 samples. We can get the performance on the 10 groups and make further analysis.

Firstly, for the 10 groups, each group has 10% training and 90% testing. Through similar procedures as the other two data set, for each group, the best parameter values from cross validation and corresponding cv accuracy and testing accuracy results are shown in the following figures:



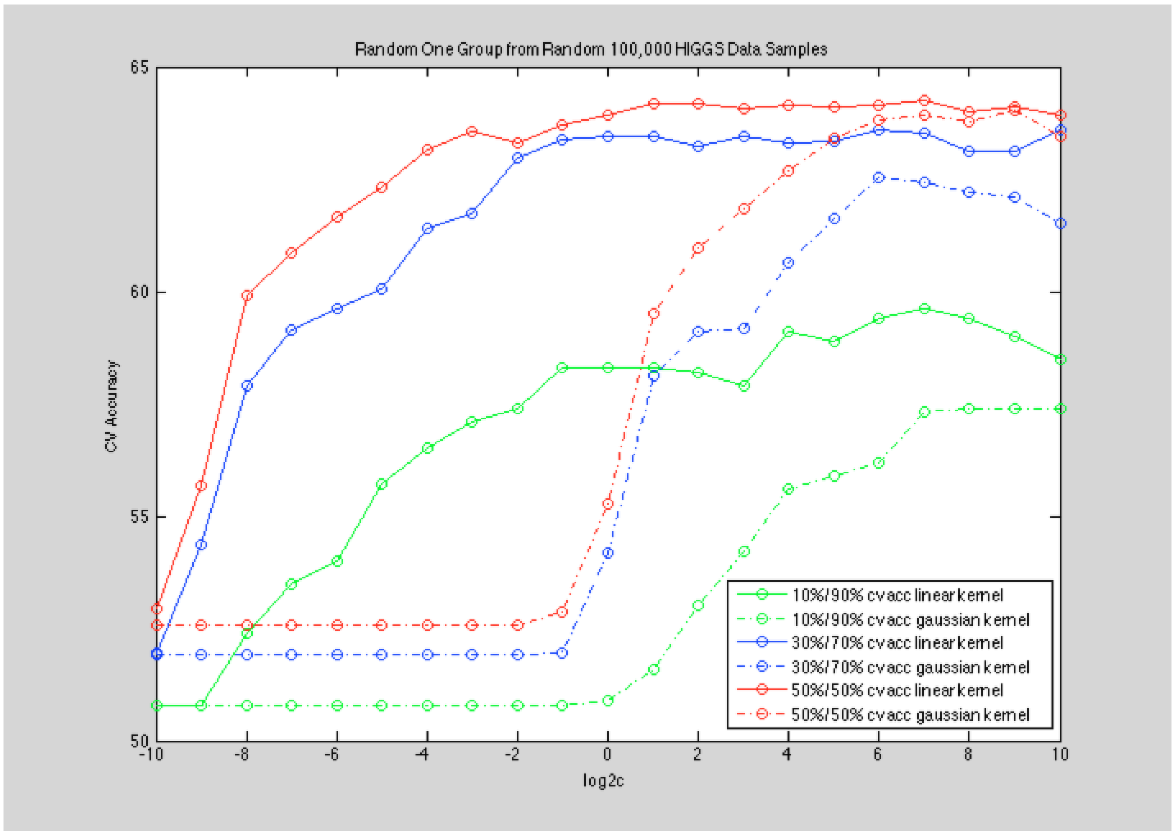


From the result, we could see that the accuracy for the ten groups are all very bad between 57% to 65%. Since 100,000 is already a sampling from the original huge data set, and the running time for 30% training and 50% training is even longer, so only one group, i.e. 10,000 samples, is randomly picked for further experiments. To further reduce the running time, is fixed to be 2-10 according to the previous results. And for the convenience of comparing results, the same data set is used for 10%, 30%, 50% training set.

Summary of results
10% Training, 90% Testing
Linear Kernel: best log2c=7, c=128, cv acc=59.6, testing acc = 61.8556
Gaussian kernel: best log2c=10, c=1024, cv acc=57.4, testing acc = 60.5333

30% Training, 70% Testing
Linear Kernel: log2c=10, c=1024, cv acc=63.6, testing acc = 63.9286
Gaussian kernel: best log2c=6, c=64, cv acc=62.5333, testing acc = 63.2571

50% Training, 50% Testing
Linear Kernel: log2c=7, c=128, cv acc=64.26, testing acc = 64.5000
Gaussian kernel: best log2c=9, c=512, cv acc=64.02, testing acc = 62.9800



Summary of Results

<u>Breast Cancer Wisconsin Data (2 classes)</u>

	10% / 90%	10% / 90%	30% / 70%	30% / 70%	50% / 50%	50% / 50%
Kernel	Linear	Gaussian	Linear	Gaussian	Linear	Gaussian
log2c	-2	6	-4	4	-1	5
log2g	-	-10	-	-9	-	-10
cv acc (%)	97.1014	95.6522	97.0732	97.561	96.7836	96.4912
acc (%)	95.1140	95.4397	96.8619	97.0711	96.7742	97.3607

Optical Recognition of Handwritten Digits Data (10 classes)

	10% / 90%	10% / 90%	30% / 70%	30% / 70%	50% / 50%	50% / 50%
Kernel	Linear	Gaussian	Linear	Gaussian	Linear	Gaussian
log2c	-8	10	-7	10	-9	10
log2g	-	-10	-	-9	-	-10
cv acc (%)	95.0178	95.9075	97.6289	98.8737	98.0783	98.9324
acc (%)	95.8679	97.5484	97.5591	98.7541	97.7936	99.0391

Random One Group from Random 100,000 HIGGS Data Samples (2 classes)

	10% / 90%	10% / 90%	30% / 70%	30% / 70%	50% / 50%	50% / 50%
Kernel	Linear	Gaussian	Linear	Gaussian	Linear	Gaussian
log2c	7	10	10	6	7	9
log2g fixed	-	-10	-	-10	-	-10
cv acc (%)	59.6	57.4	63.6	62.5333	64.26	64.02
acc (%)	61.8556	60.5333	63.9286	63.2571	64.5000	62.9800

Discussions

- The optimal value of C is relatively small for linear kernel, but big for Gaussian kernel. Small C would allow for more samples to be mislabeled. When the data is not linearly separable (which often happens), small C could make SVM even softer for the tolerance of more mislabeled data. The Gaussian kernel constructs a nonlinear classifier based on Hilbert space of infinite dimensions, the property of nonlinearity could make the data more separable and a smaller tolerance of mislabeled data, i.e., bigger C is performing better.

- The optimal value of γ is generally very small, since the Gaussian kernel is $k(x_i,x_j) = exp(-\gamma\|x_i - x_j\|^2)$, small γ means a very flat function with big variance. This would reduce the impact of single sample, but emphasize more on the whole data set when the samples are affecting each other.

- **The performance of Gaussian kernel is slightly better than linear kernel,** but the difference is very small. So by the optimal parameter selection from cross validation, linear kernel could somehow compete with Gaussian kernel in our experiments. **For HIGGS data sample, linear kernel is performing better than Gaussian kernel.** But since HIGGS performance is too bad, it does not provide convincing conclusions.

- **As the percentage of training data increases from 10% to 50%, the performance (accuracy on testing set) is better.** This is reasonable since with more training data, the classifier could learn more information about the data properties.

- LIBSVM is not efficient in processing big nonseparable data. Breast Cancer Wisconsin Data has 683 samples and 2 classes, it is performing well. Optical Recognition of Handwritten Digits Data has 5620 samples and 10 classes, so the number of samples for each class in only around 500~600, also perform well. But for HIGGS Data, the size of the data set 11,000,000 is too huge, even only sampled 10,000 samples from the 100,000 sample (used for other algorithms), for the cross validation it still took too long to get the result. In this case, 10,000 samples for 2 classes is already a big data for LIBSVM to run efficiently for a result with cross validation. And the nonseparability of HIGGS Data made this even worse.

- **The performance on Breast Cancer Wisconsin Data and Optical Recognition of Handwritten Digits Data is pretty good with accuracy over 95%**. But for HIGGS Data, the performance is very bad with accuracy only around 60%, just a little better than the 50% probability of randomly labelling. **Seen from the PCA analysis of HIGGS data, the two classes are overlapping with each other and could hardly distinguish from each other. This might be the reason for the bad performance on HIGGS data.**

2.2 Deep Learning

Introduction

An algorithm is deep if the input is passed through several non-linearities before being output. Most modern learning algorithms (including decision trees and SVMs and naive bayes) are "shallow". A shallow machine learning architecture would involve a lot of duplication of effort to express things that a deep architecture could more compactly. The point being, a deep architecture can more gracefully reuse previous computations.

Back Propagation Algorithm

In a neural network, the inputs are processed by the neurons using certain weights to yield the output. The back propagation step yields new weights for the neurons, through a process of optimization via gradient descent (this is the most basic method). This step is a feedback step.

Feed Forward Neural Network

In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

We have used two main algorithms :**a.) multi-layer perceptron b.) Stacked-autoencoder.**

Activation Function Used: We have used softmax activation function in the output layer of our algorithm. The softmax activation function is useful predominantly in the output layer of a clustering system. Softmax functions convert a raw value into a posterior probability. This provides a measure of certainty. The softmax activation function is given as:

$$y_i = \frac{e^{\zeta_i}}{\sum_{j \in L} e^{\zeta_j}}$$

L is the set of neurons in the output layer.

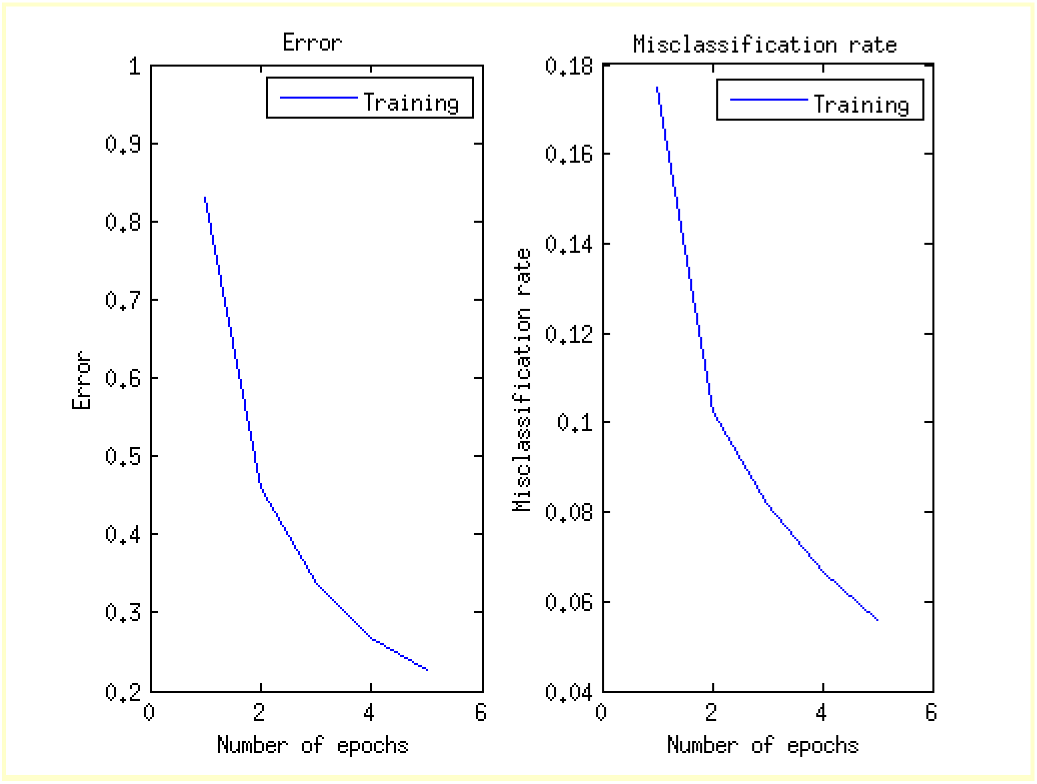
Loss function used: We used cross entropy loss function in our algorithm. This is the loss function defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions.

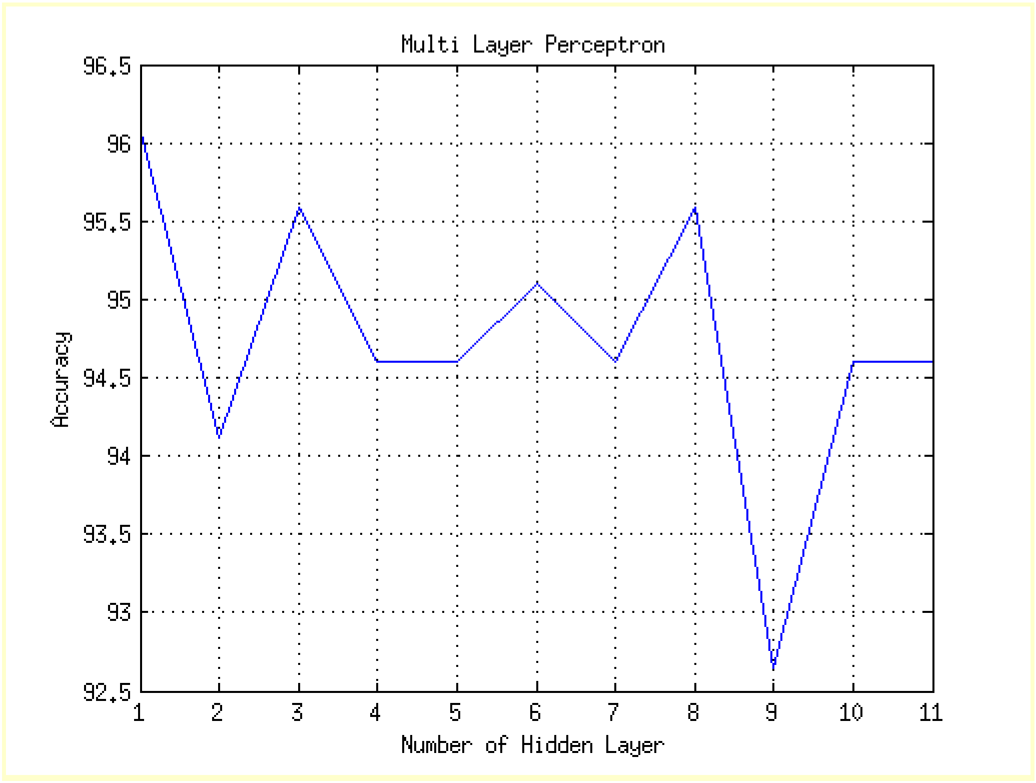
Experiments with Multi Layer Perceptron

Multilayer Perceptron on Breast Cancer Dataset-

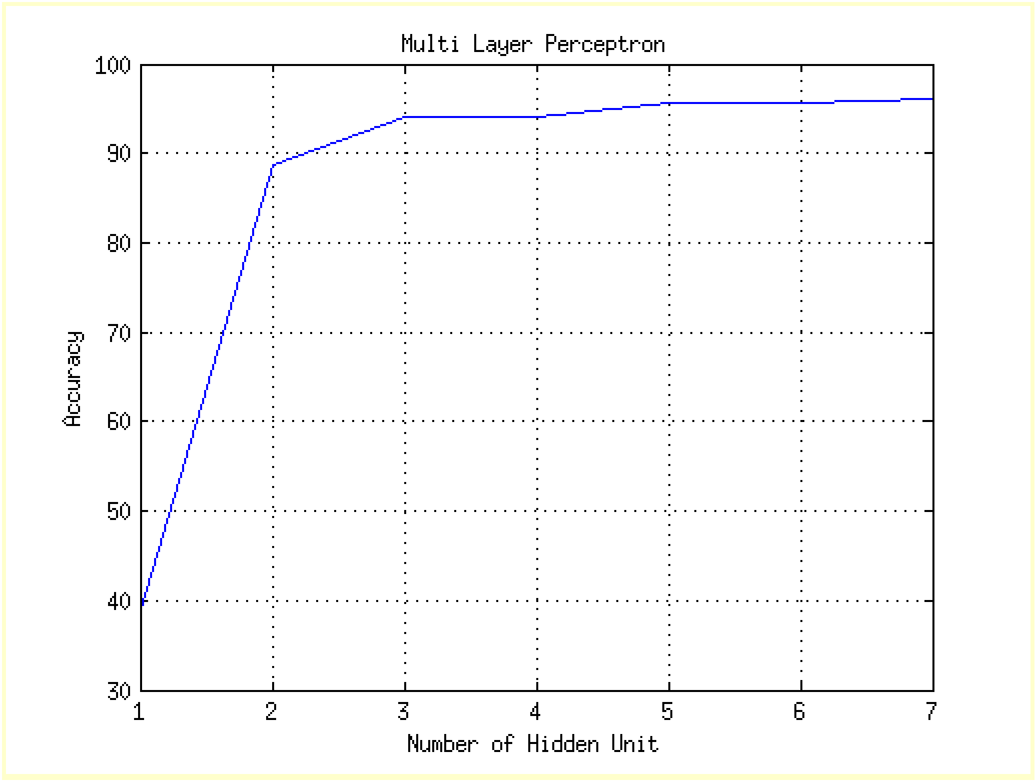
For All test Batchsize=100

	10% Train	**30% Train**	**50% Train**
Accuracy	94.117647	95.588235	96.187683
NumHiddenLayers	4	3	4
NumHiddenunit(neurons)	3	12	6
Num epochs	10	10	10





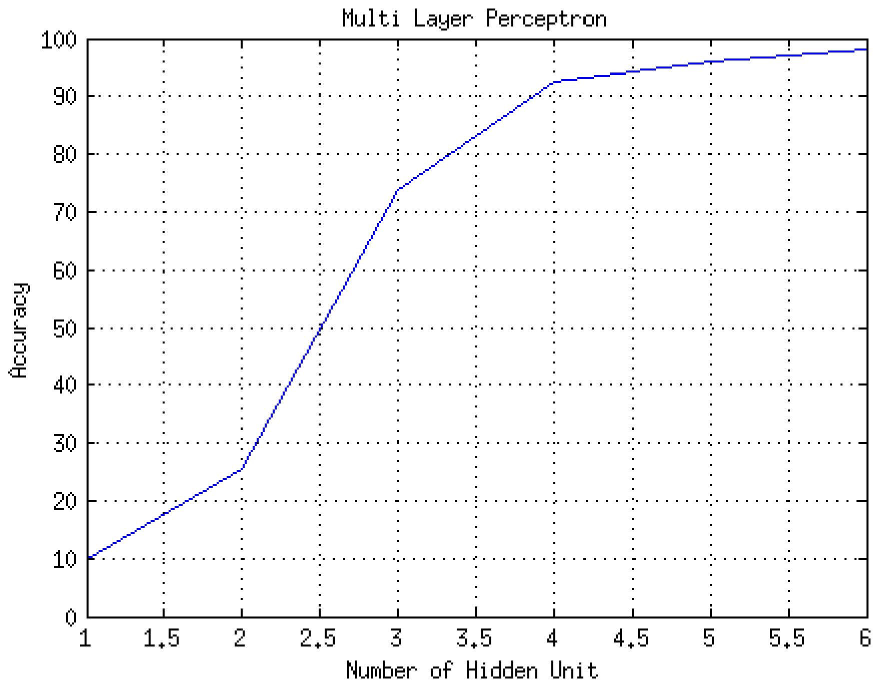
Number of Hidden Unit=10

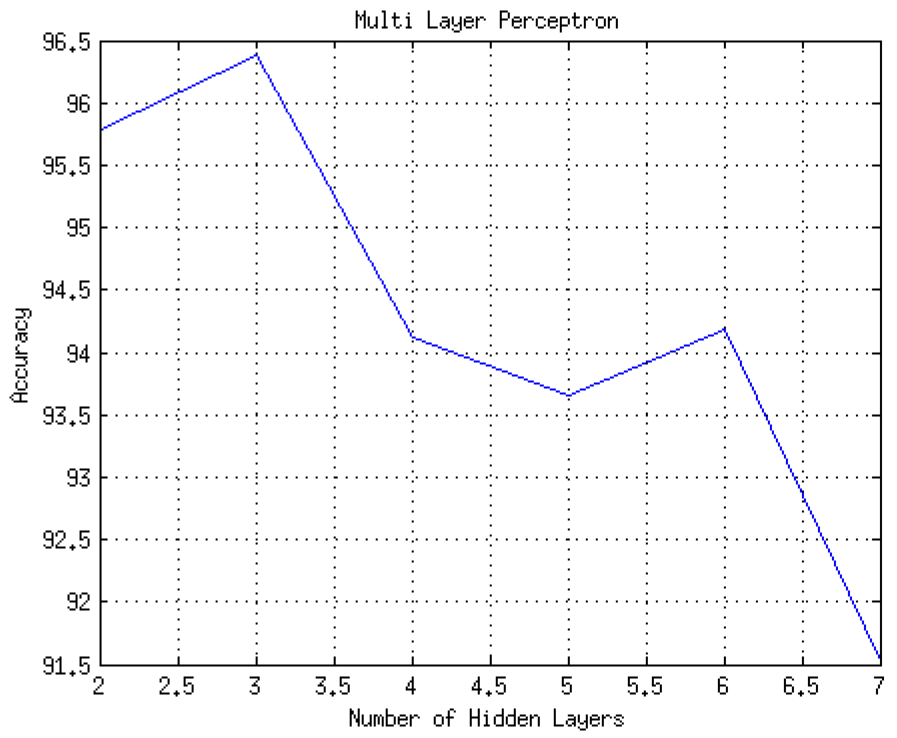


Multi Layer Perceptron OCR Digits-

For all test batchsize = 100.

	10% Train	**30% Train**	**50% Train**
Accuracy	98.398577	98.220641	96.797153
NumHiddenLayers	4	4	4
NumHiddenunit(neurons)	6	12	6
Numepochs	10	10	10

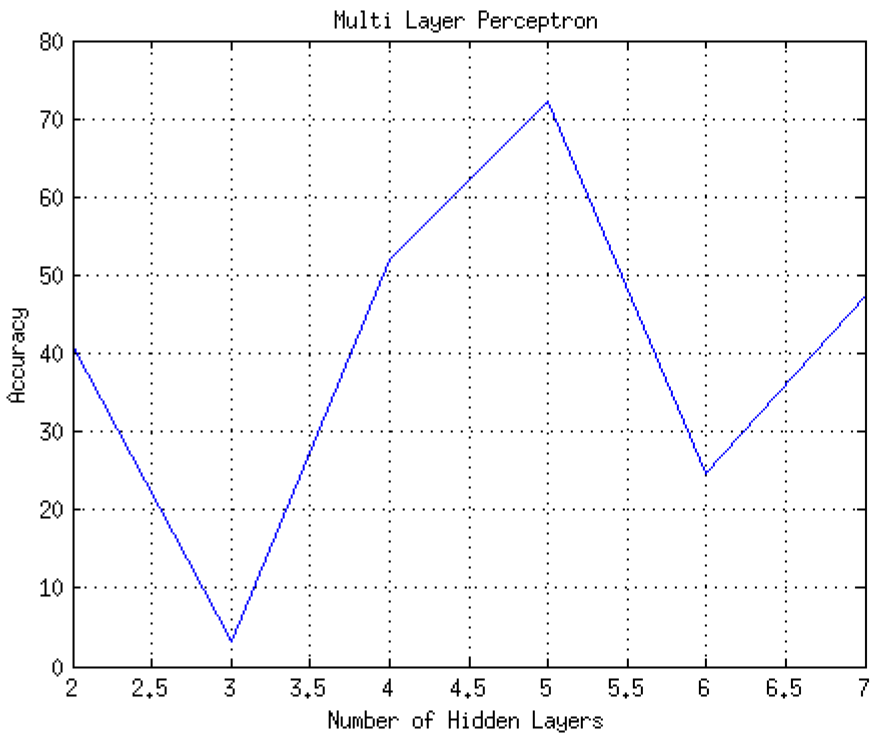




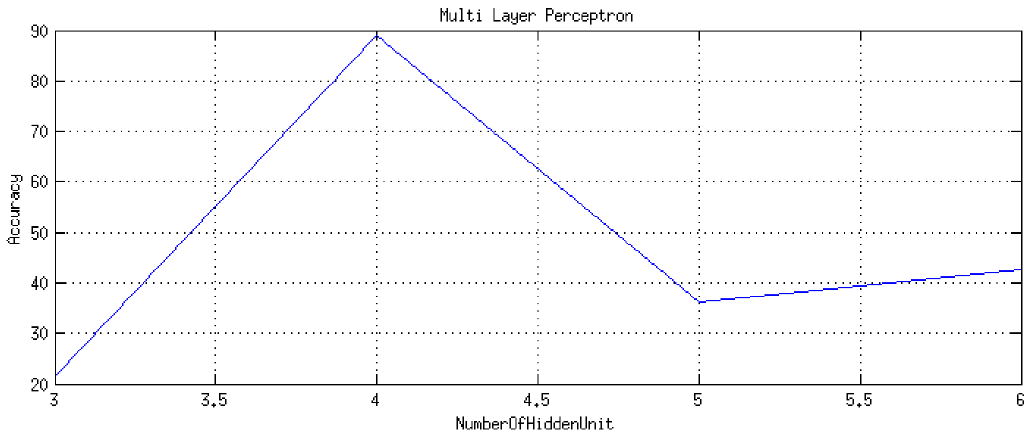
Multi Layer Perceptron on Higgs Dataset

For all test batchsize = 100.

	10% Train	**30% Train**	**50% Train**
Accuracy	87.3	88.8	90.01
NumHiddenLayers	4	5	4
NumHiddenunit(neurons)	5	8	4
Numepochs	100	120	100



Multi Layer Perceptron
Accuracy
Number of Hidden Layers



Multi Layer Perceptron
Accuracy
NumberOfHiddenUnit

Multiplayer Perceptron Conclusion-

- We get best result for Multilayer perceptron if Number of neuron(Number of Hidden) is close to Number of feature . If we take too many neurons dataset performs very badly.

- Increasing Number of Hidden layer on a simple dataset like Breast cancer from a too large value decreases accuracy.

- By increasing the number of Neurons and hidden layers such that the number of features are less than the number of neurons and hidden layers then the accuracy rate is not improved and for breast cancer dataset, the accuracy rate decreases.

- We used tanh,softmax and sigmoid activation functions and they give almost same result.

Stacked Auto Encoder -

A stacked autoencoder is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer is wired to the inputs of the successive layer. Formally, consider a stacked autoencoder with n layers. Using notation from the autoencoder section, let $W^{(k,1)}, W^{(k,2)}, b^{(k,1)}, b^{(k,2)}$ denote the parameters $W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}$ for kth autoencoder. Then the encoding step for the stacked autoencoder is given by running the encoding step of each layer in forward order:

$$a^{(l)} = f(z^{(l)})$$
$$z^{(l+1)} = W^{(l,1)} a^{(l)} + b^{(l,1)}$$

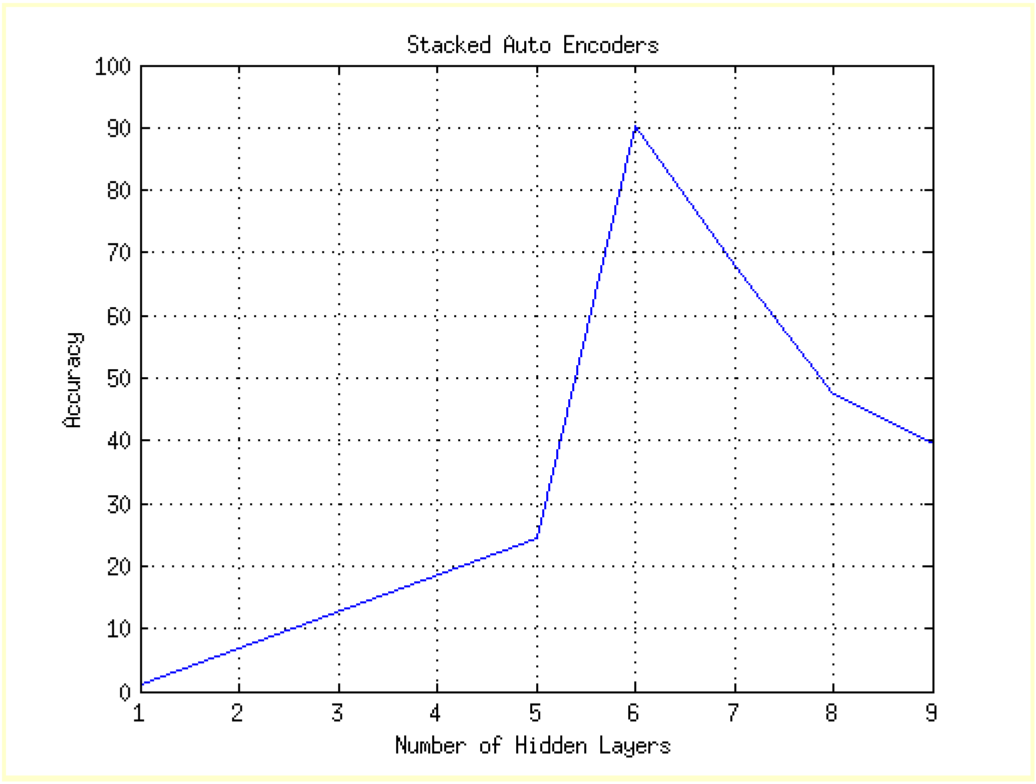
The decoding step is given by running the decoding stack of each autoencoder in reverse order:

$$a^{(n+l)} = f(z^{(n+l)})$$
$$z^{(n+l+1)} = W^{(n-l,2)} a^{(n+l)} + b^{(n-l,2)}$$

The information of interest is contained within $a^{(n)}$, which is the activation of the deepest layer of hidden units. This vector gives us a representation of the input in terms of higher-order features.

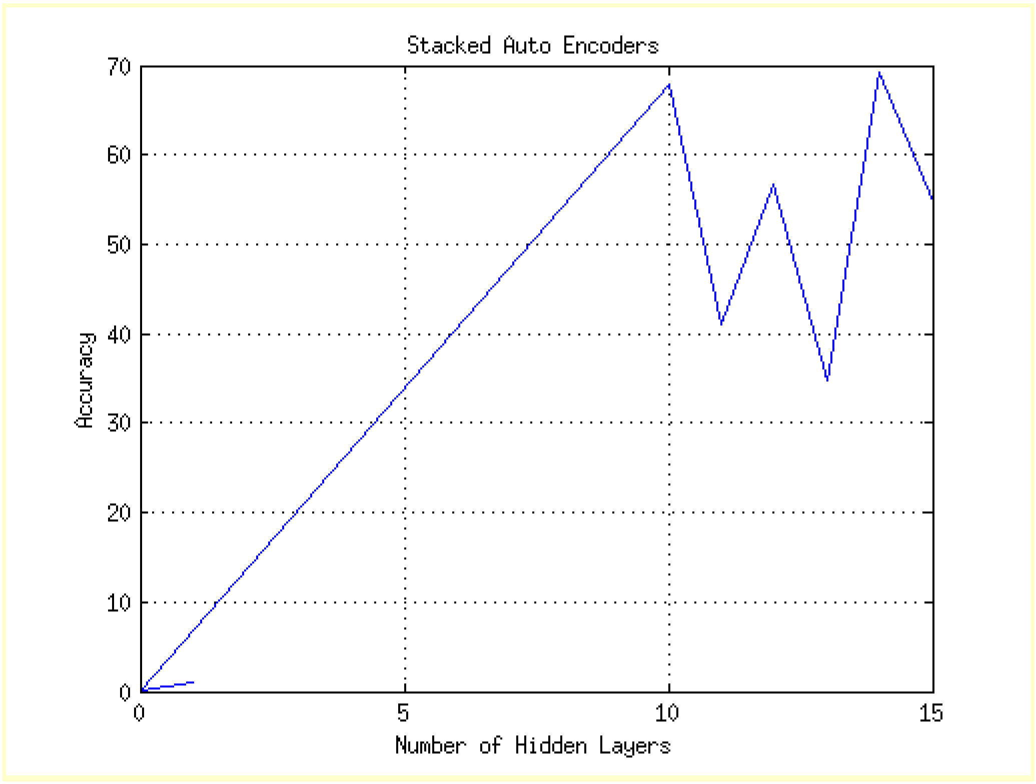
1) Stacked AutoEncoder on Breast Cancer Dataset

	10% Train	**30% Train**	**50% Train**
Accuracy	80.88	90.19	96.797153
NumHiddenLayers	4	5	4
NumHiddenunit(neurons)	6	10	6
Numepochs	10	10	10



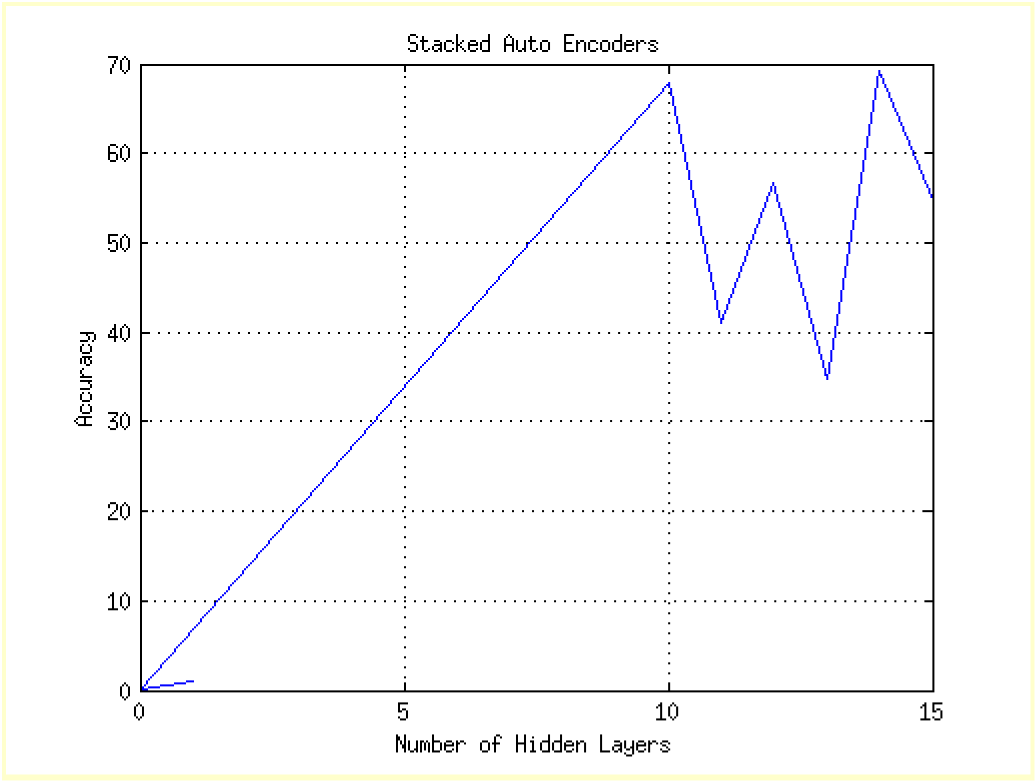
2) Stacked AutoEncoder on OCR Digit Dataset

	10	**30**	**50**
Accuracy	84.88	91.19	94.797153
NumHiddenLayers	4	6	4
NumHiddenunit(neurons)	6	10	6
Numepochs	20	20	20



3) Stacked AutoEncoder on Higgs Dataset

	10	**30**	**50**
Accuracy	82.88	84.86	84.85
NumHiddenLayers	4	5	4
NumHiddenunit(neurons)	8	10	5
Numepochs	100	100	10



Stack Auto Encoder Conclusion

- We get best result for Stacked Encoder if Number of neuron(Number of Hidden) is close to Number of feature . If we take too many neurons dataset performs very perform badly

- We used tanh,softmax and sigmoid activation functions and they give almost same result.

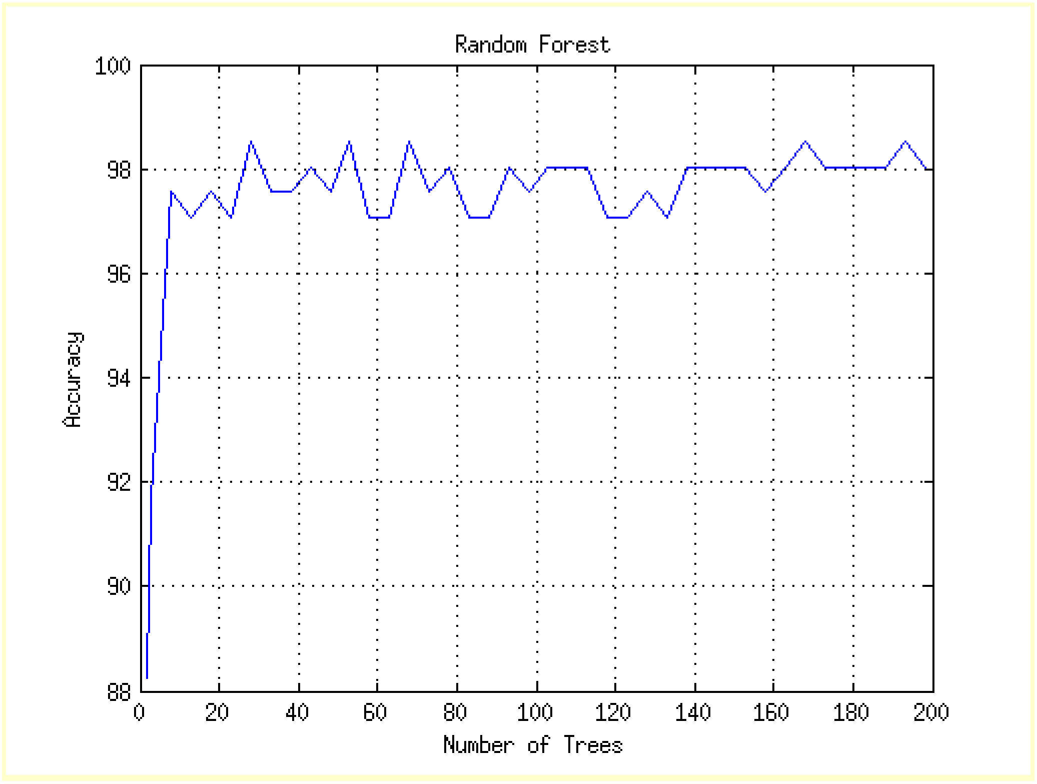
- Increasing Number of Hidden layer on a simple dataset like Breast cancer from a too large value decreases accuracy.

2.3 Random Forest

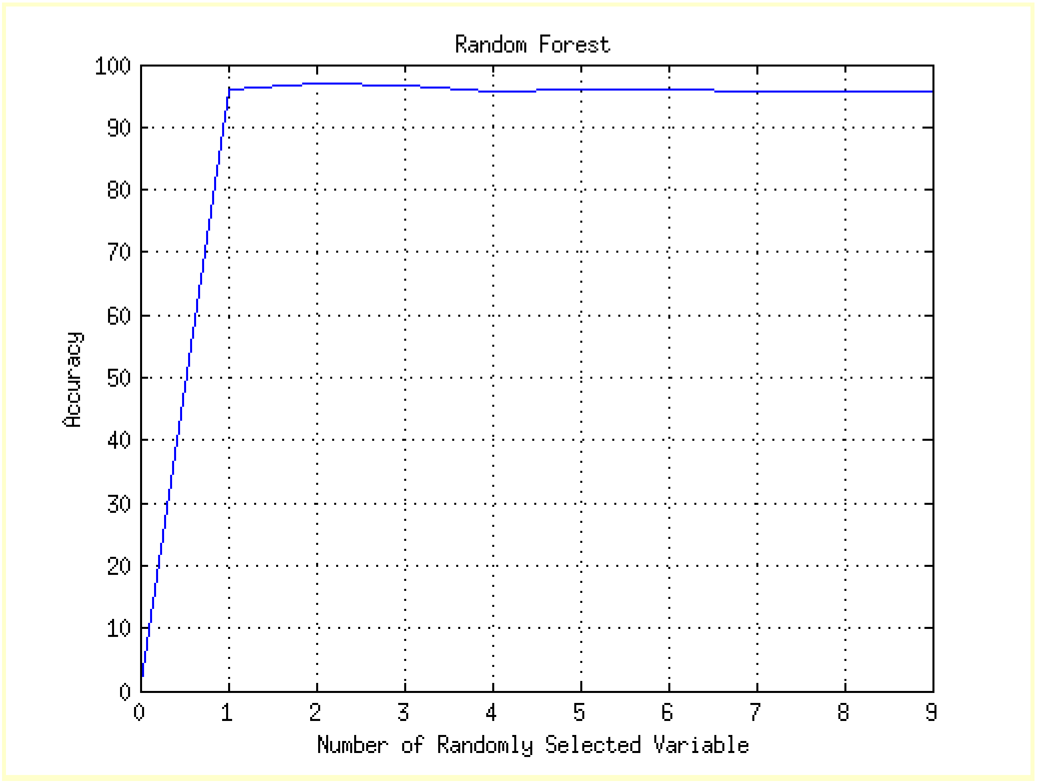
The Random Forest generates a set of random trees.. The resulting forest model contains a specified number of random tree models. The *number of trees* parameter specifies the required number of trees. The resulting model is a voting model of all the random trees.

Breast Cancer Dataset-

Random Forest	**10**	**30**	**50**
Accuracy	98.529412	99.50	98.7
Number of Randomly Selected Variable	4	4	5
Method for splitting node	Gini Index	Gini Index	Gini Index
Number of Trees	120	100	120

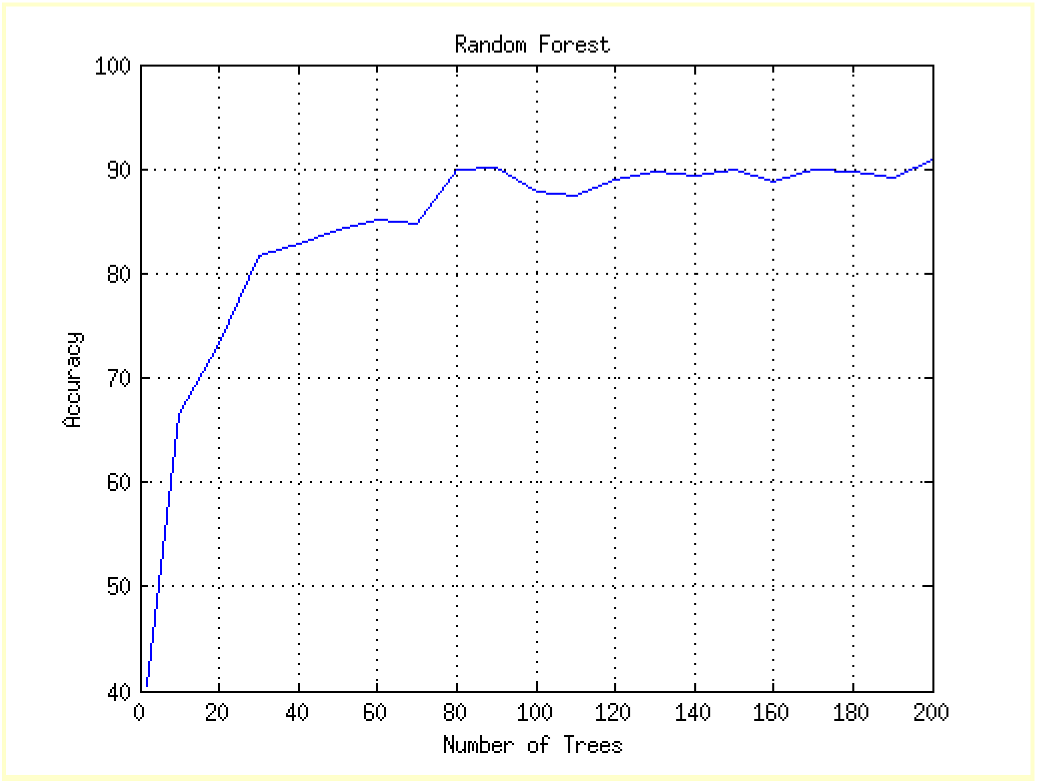


Number of Randomly Selected Variable=4

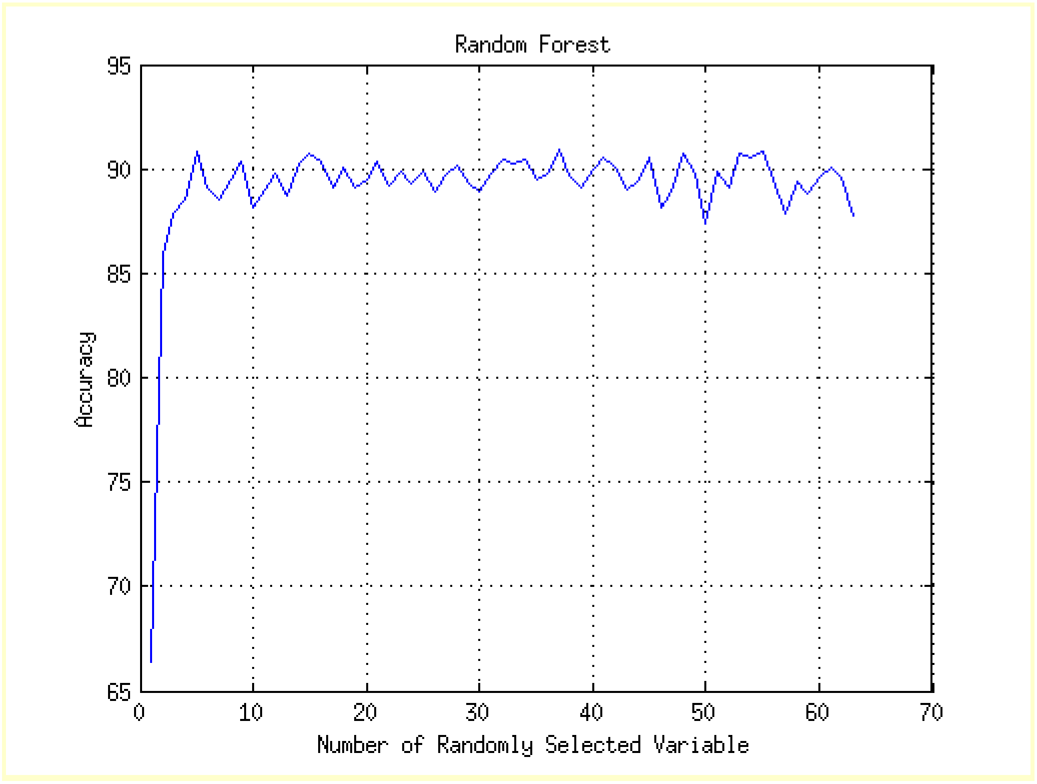


OptDigitDataset

Random Forest	**10**	**30**	**50**
Accuracy	90.25	93.17	91.85
Number of Randomly Selected Variable	14	17	16
Method for splitting node	Gini Index	Mean Squared Error	Gini Index
Number of Trees	220	300	400



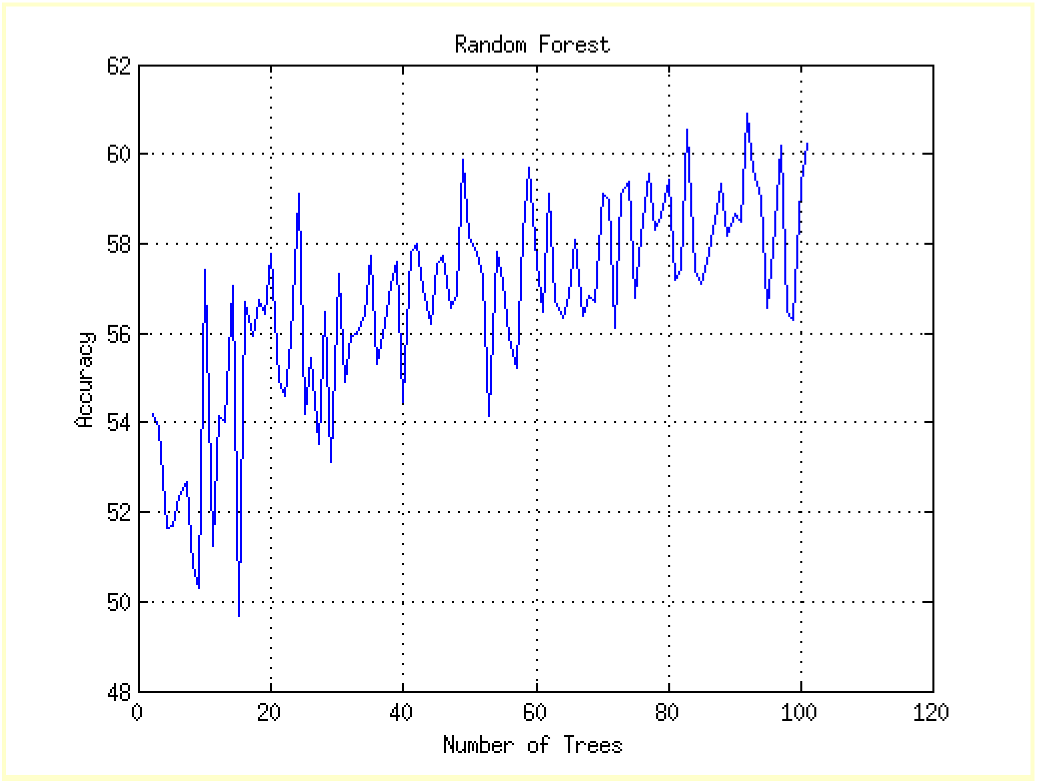
Number of Randomly Selected feature=17

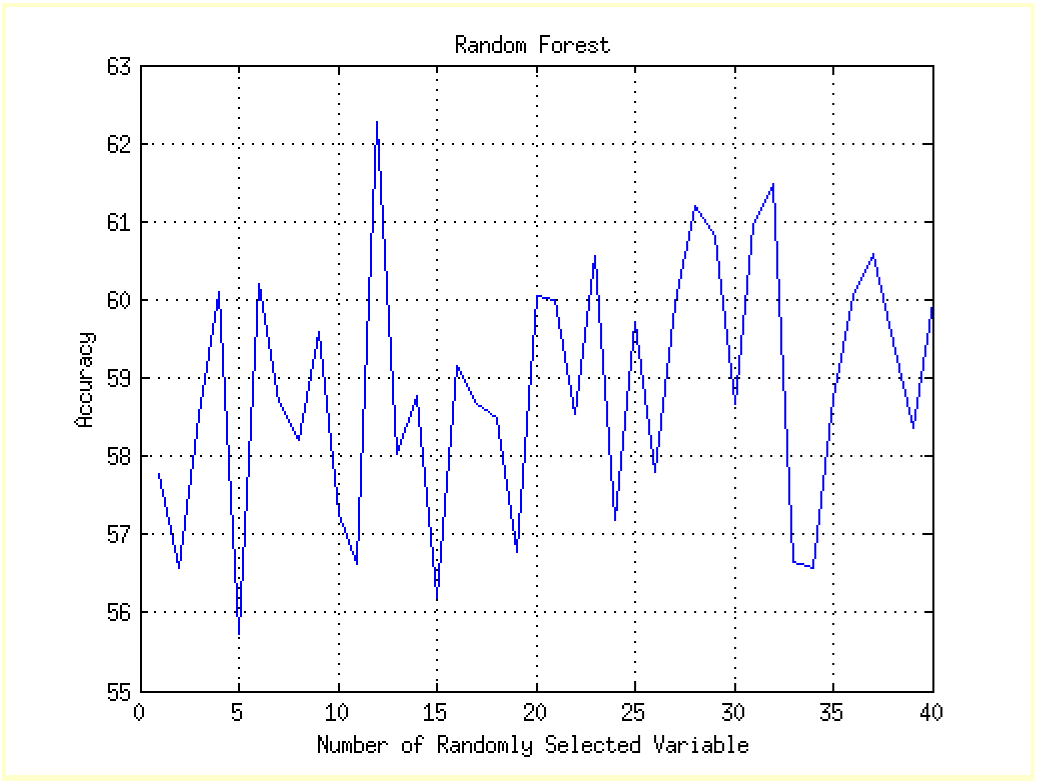


Number of Trees=200

3) Higgs Dataset

	10	**30**	**50**
Accuracy	60.92	61.92	62.92
Number of Randomly Selected Variable	11	10	10
Method for splitting node	Gini Index	Gini Index	Gini index
Number of Trees	112	92	100


Random Forest
Accuracy
Number of Trees
62
60
58
56
54
52
50
48
0
20
40
60
80
100
120


Random Forest
Accuracy
Number of Randomly Selected Variable
63
62
61
60
59
58
57
56
55
0
5
10
15
20
25
30
35
40

Random Forest Conclusion-

a) Usually more trees is perform better with than less trees but increase Time Complexity of Algorithm. Hence with smaller datasets breast cancer and optical character digit recognition datasets performed quickly compared to higgs dataset.

b).Deeper trees are almost always better subject to requiring more trees for similar performance. Based on our observations we found out that accuracy after selecting fewer features becomes more or less constant after a certain point. **e.g in the OCR dataset if we selected 5 features or more, accuracy remains more or less constant.**

c) Deeper trees reduces the bias; more trees reduces the variance.

d) Most important parameter is how many features to test for each split. We can sort of tune it via Out of Bag Error(OOB)estimates if you just want to know your performance on your training data and there is no twinning (~repeated measures). Even though this is the most important parameter, it's optimum is still usually fairly close to the original suggest defaults (sqrt(p) or (p/3) for classification/regression).

e) We don't even need to do exhaustive split searches inside a feature to get good performance. Just try a few cut points for each selected feature and move on. This makes training even faster.

2.4 Ensemble Methods

We compared the following algorithms performance with different number of learners.

- **AdaBoostM1(AdaBoostM2 for multiclass)** :

AdaBoostM1 is a very popular boosting algorithm for binary classification. The algorithm trains learners sequentially. For every learner with index *t*, AdaBoostM1 computes the weighted classification error

$$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} \mathbb{I}\left(y_n \neq h_t(x_n)\right),$$

where

x_n is a vector of predictor values for observation *n*.

y_n is the true class label.

h_t is the prediction of learner (hypothesis) with index *t*.

$\mathbb{I}$ is the indicator function.

$d_n^{(t)}$ is the weight of observation *n* at step *t*.

AdaBoostM1 then increases weights for observations misclassified by learner *t* and reduces weights for observations correctly classified by learner *t*. The next learner *t* + 1 is then trained on the data with updated weights $d_n^{(t+1)}$.

After training finishes, AdaBoostM1 computes prediction for new data using

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x),$$

where

$$\alpha_t = \frac{1}{2} \log \frac{1-\varepsilon_t}{\varepsilon_t}$$

are weights of the weak hypotheses in the ensemble.

Training by AdaBoostM1 can be viewed as stagewise minimization of the exponential loss

$$\sum_{n=1}^{N} w_n \exp\left(-y_n f(x_n)\right),$$

- **Subspace**

Use random subspace ensembles (Subspace) to improve the accuracy of discriminant analysis (ClassificationDiscriminant) or *k*-nearest neighbor (ClassificationKNN) classifiers. Subspace ensembles also have the advantage of using less memory than ensembles with all predictors, and can handle missing values (NaNs).

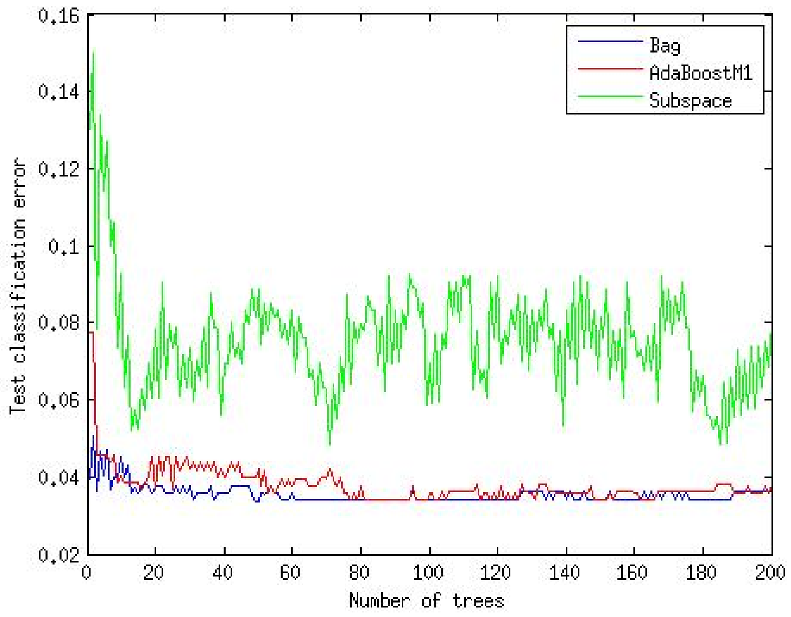
- **Bag**

Bagging, which stands for "bootstrap aggregation," is a type of ensemble learning. To bag a weak learner such as a decision tree on a dataset, generate many bootstrap replicas of this dataset and grow decision trees on these replicas.

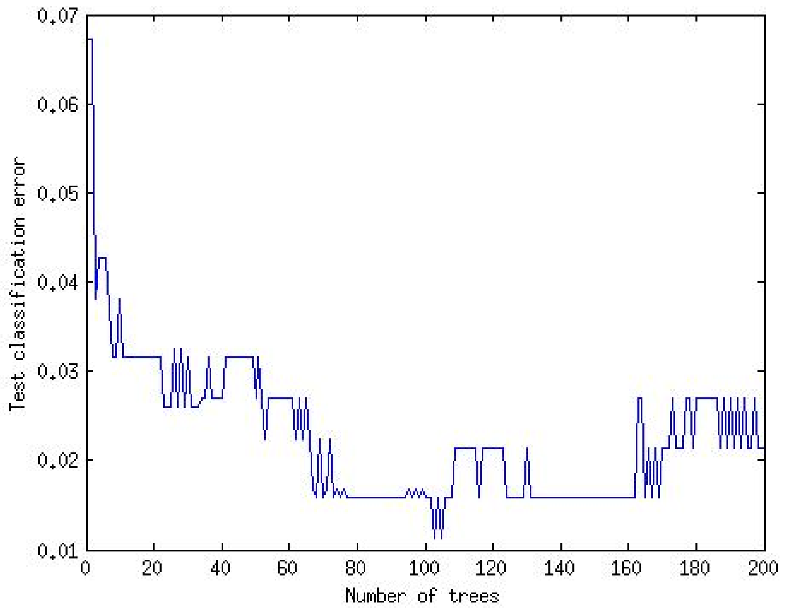
Experiments on Datasets

We tried over above 3 ensemble methods to measure the performance of each on all the three different data sets. On the best performing algorithm we tuned it for "number of iterations" and noted the results.

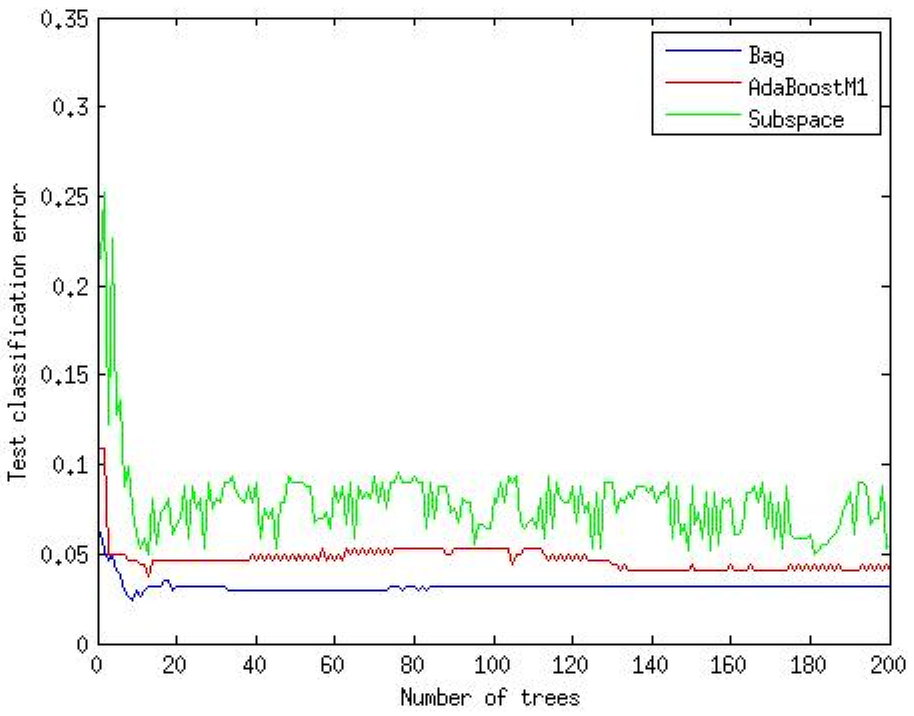
Breast Cancer
Training : 30%



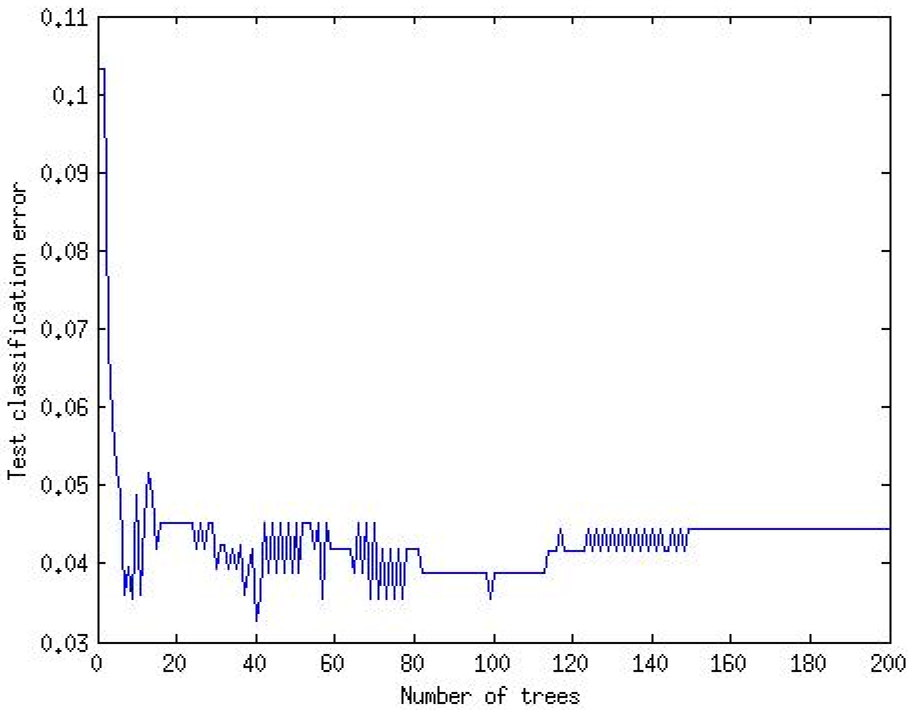
Best performance on : Bag ;   Optimal number of learners   : 100 ;Accuracy   : 94.979079



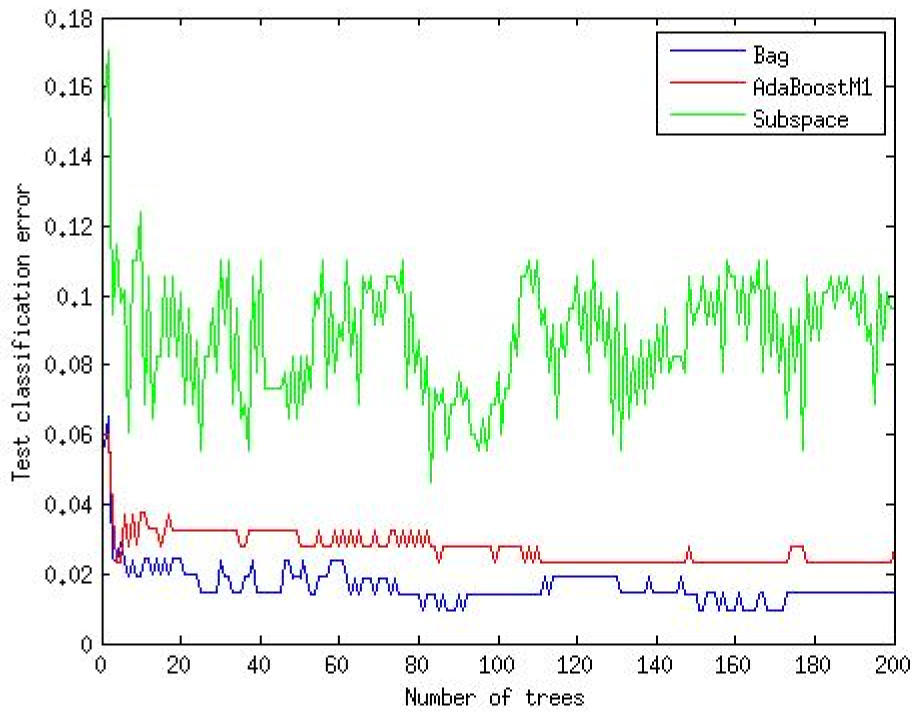
Training 50%



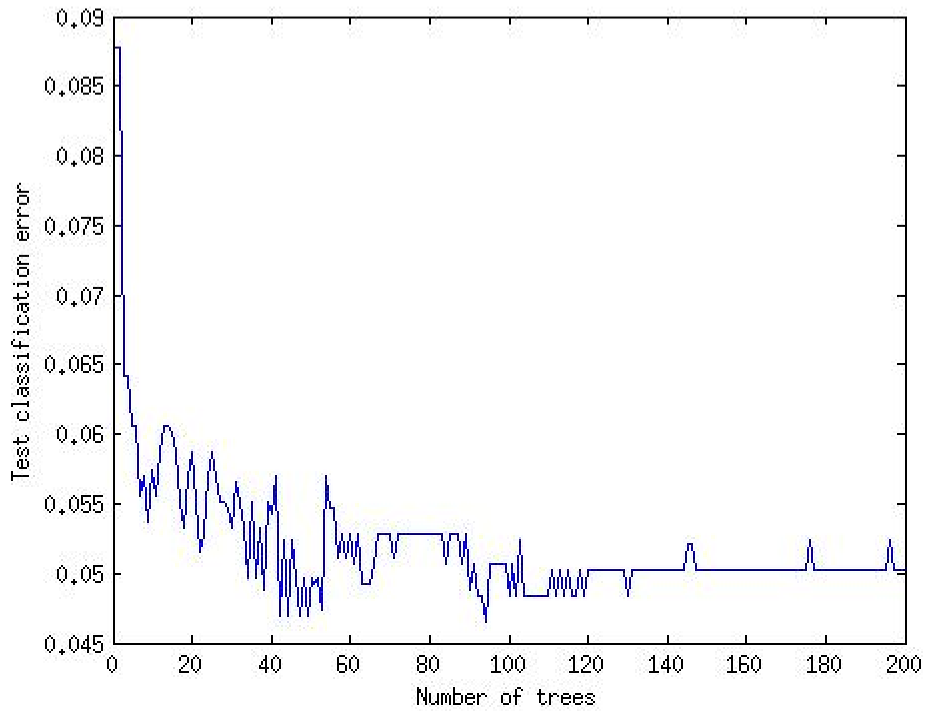
Best performance on : Bag; Optimal number of learners : 40; Accuracy : 96.480938



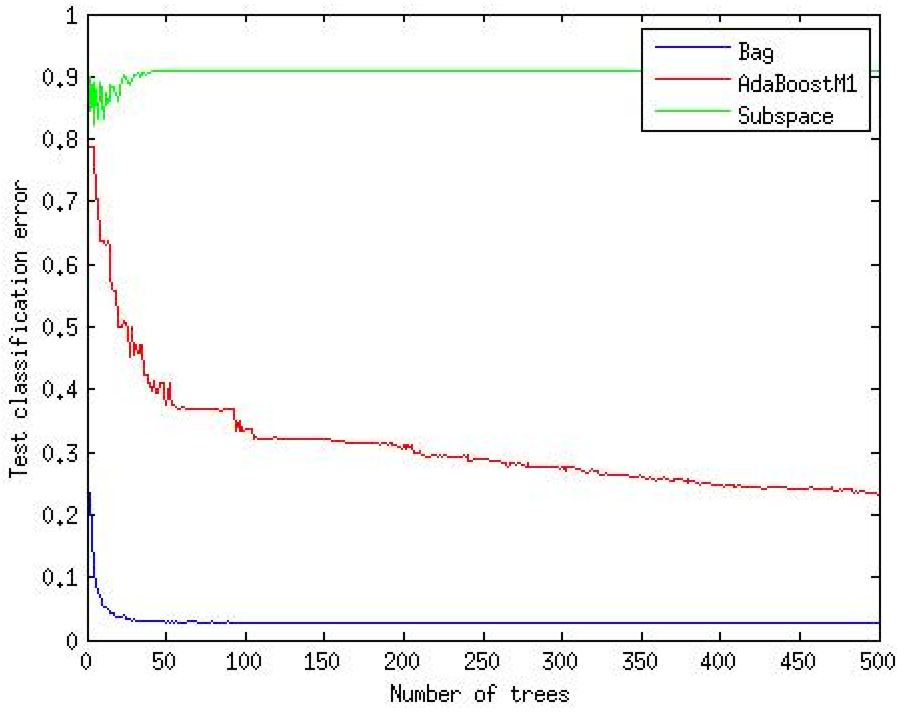
70% training

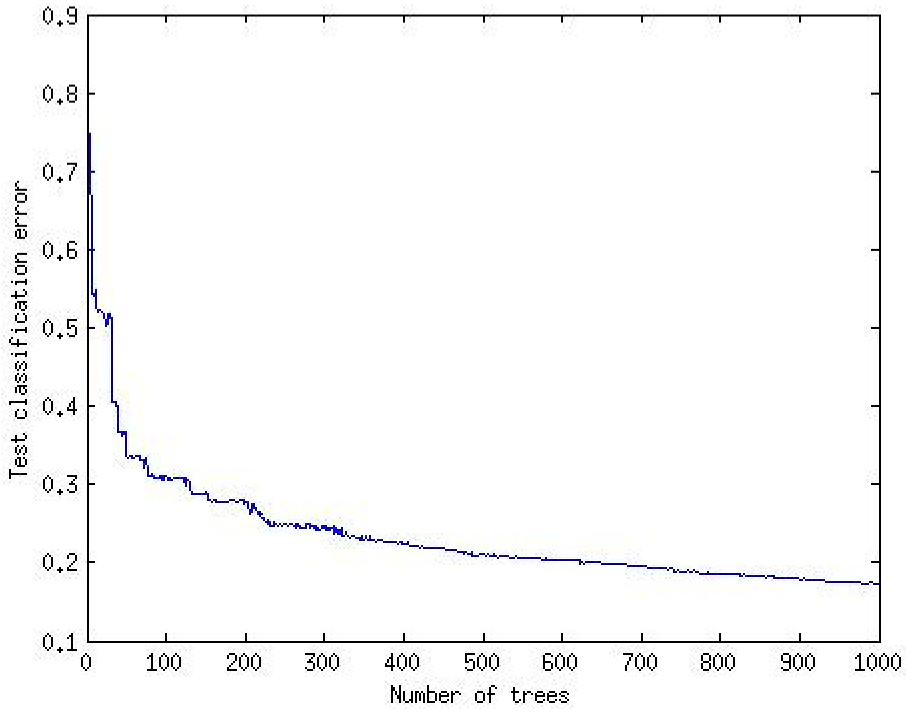


Optimal number of learners : 90; Accuracy : 98.529412



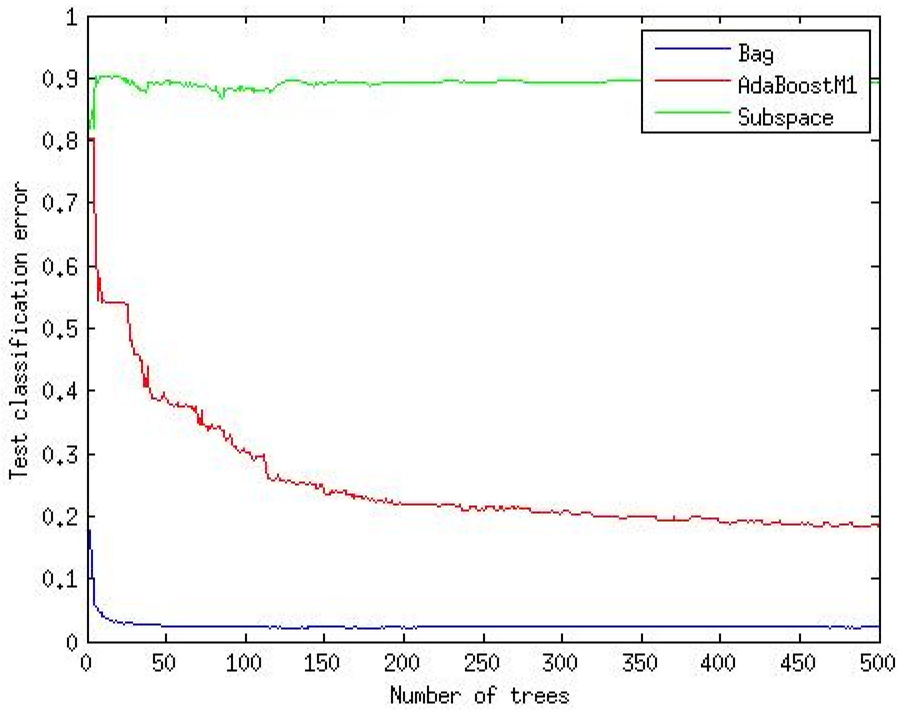
OCR Digits
30% training

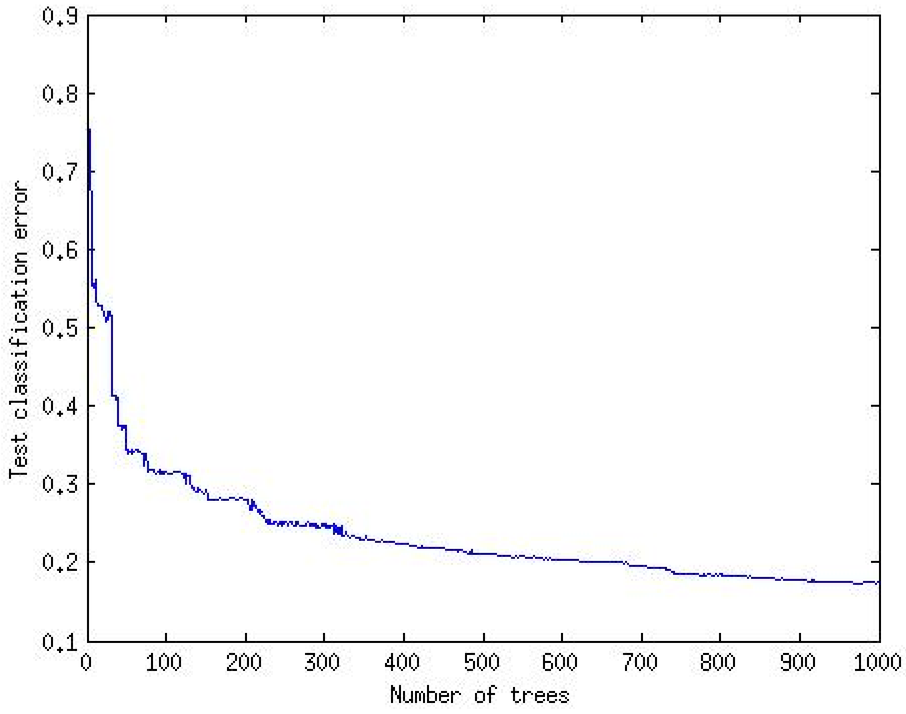




Accuracy: 97.431986

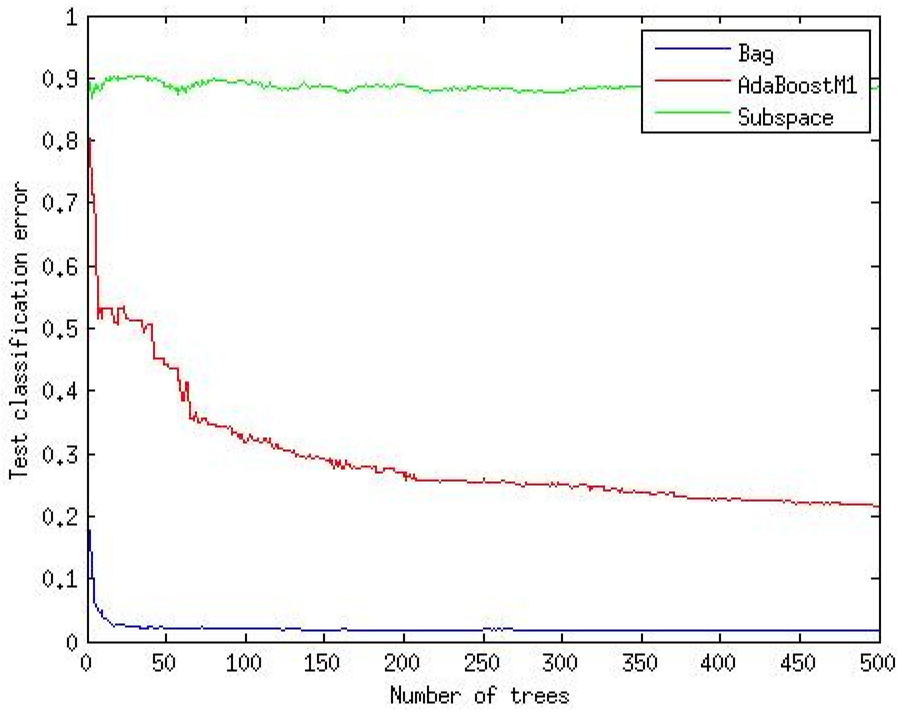
50% training

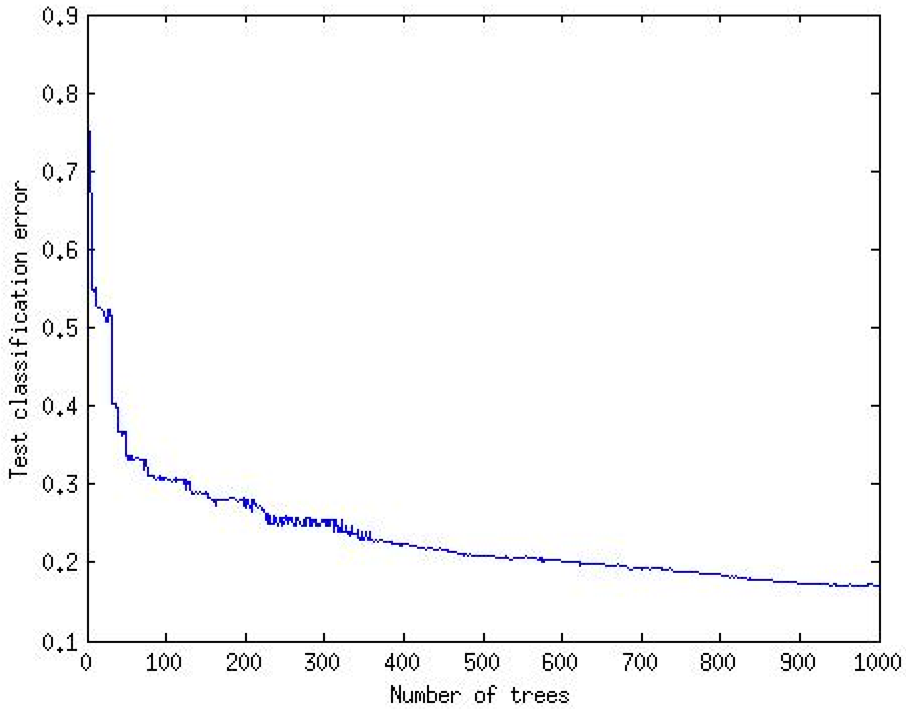




Accuracy: 98.185053

70% training

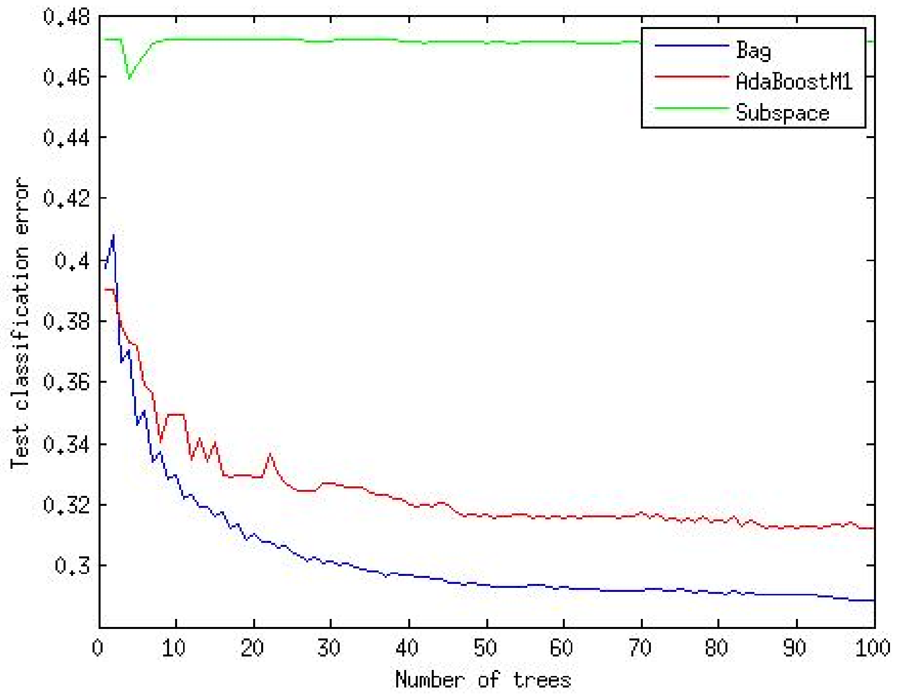




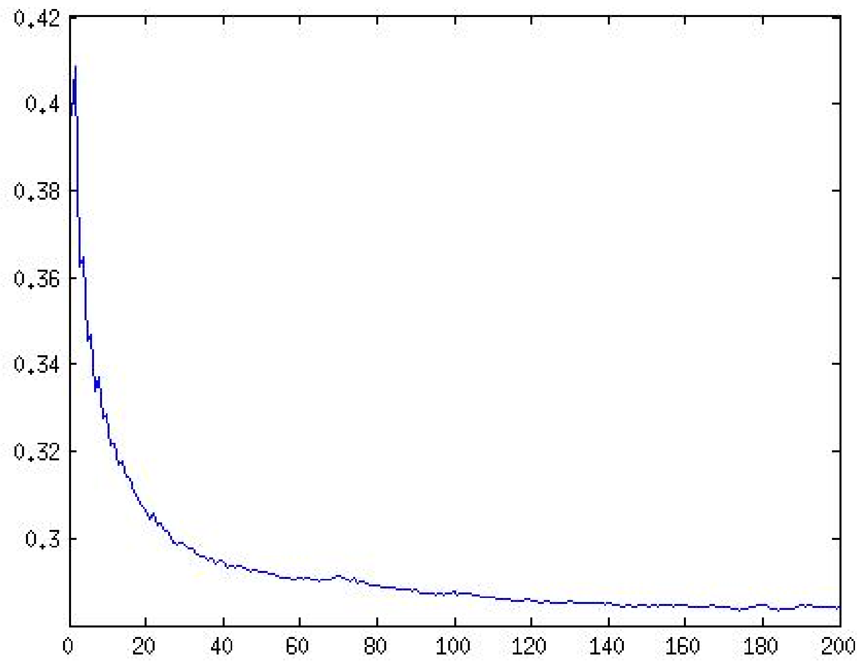
Accuracy: 98.398577

HIGGS

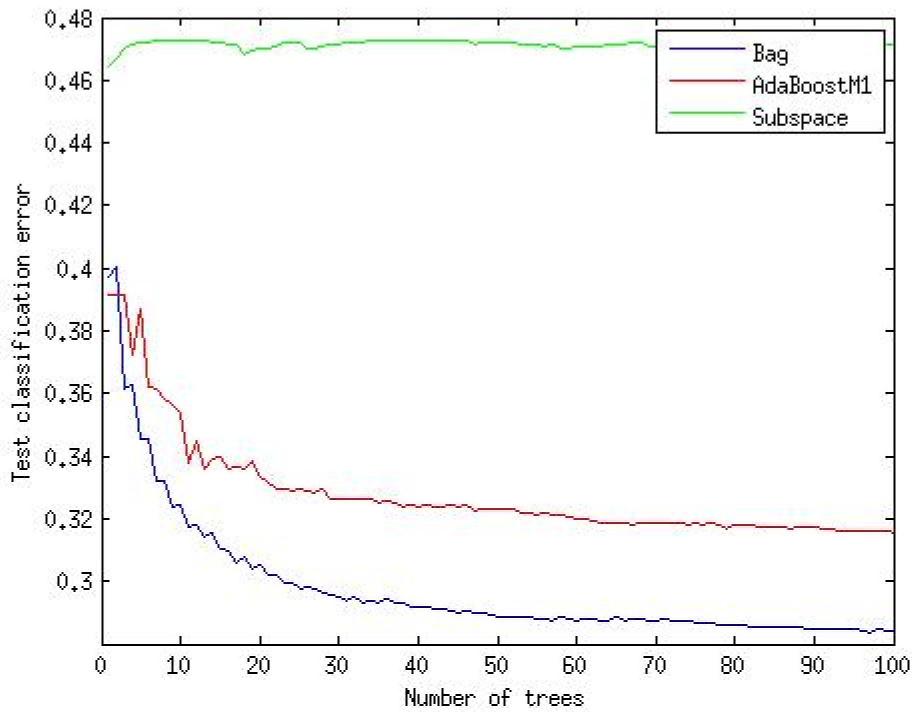
30% training



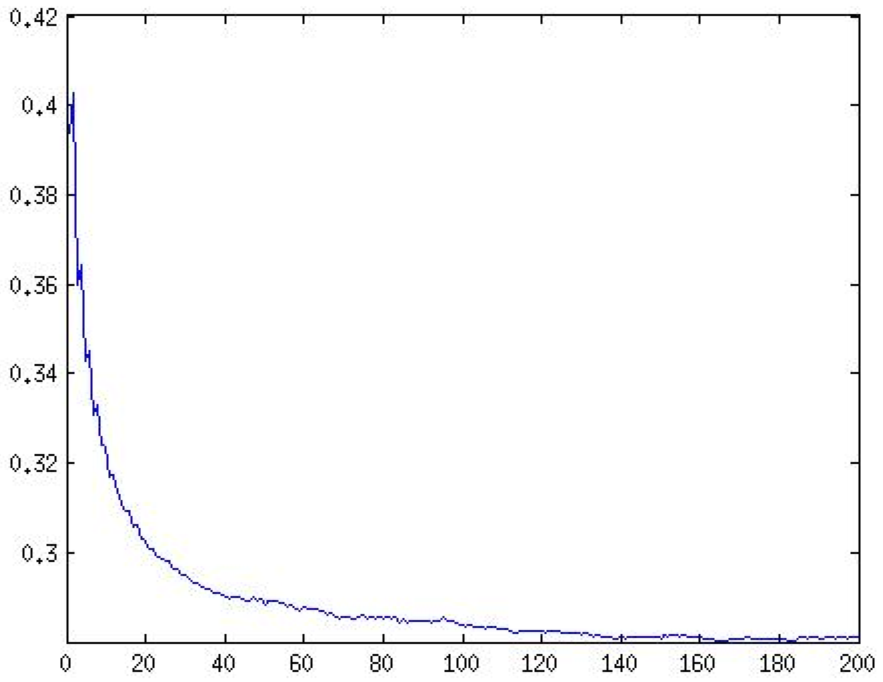
Accuracy: 71.172857



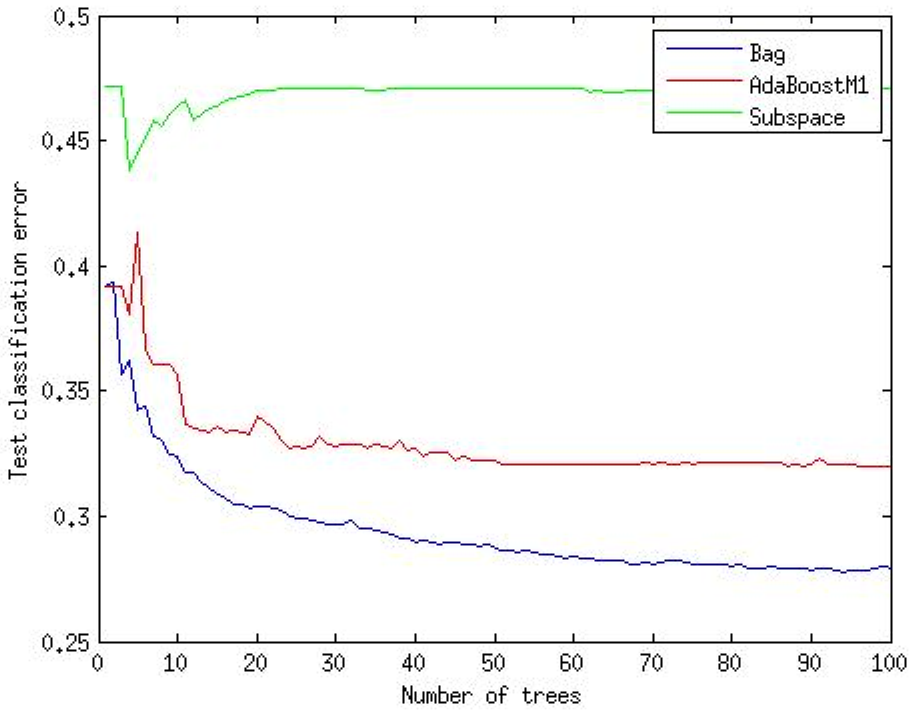
50% training



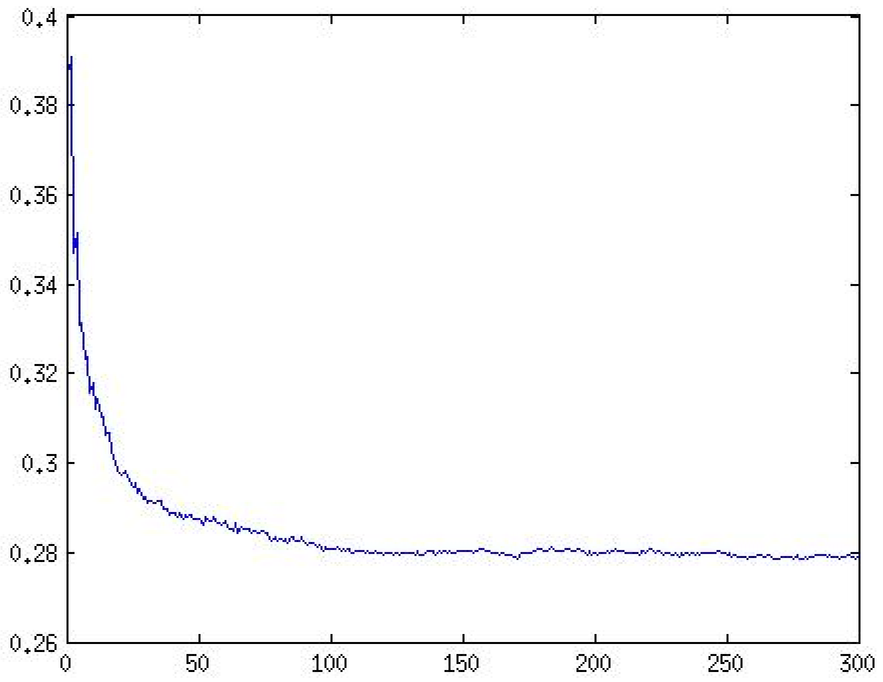
Accuracy: 71.570000



70% training



Accuracy: 72.120000



Conclusions

a.) Bagged decision trees always performed better than boosted trees for all three datasets. Possible reasons can be because boosted trees are prone to overfitting on training data set.

b.) We tried checking the quality of ensemble by plotting accuracy with number of iterations of the ensemble and found the optimum values for each data set. We found the performance converging around 20 for Breast Cancer data set, around 90 for Optical Digit Recognition and 150 for Higgs dataset.

c.) As the training size increased accuracy on the test set increased.

d.) As the number of features increased it required more number of decision trees to get the optimum performance as was observed in the case of OCR digits with 64 features it needed around 40 trees to give the best performance while just 20 were needed for breast cancer data set.

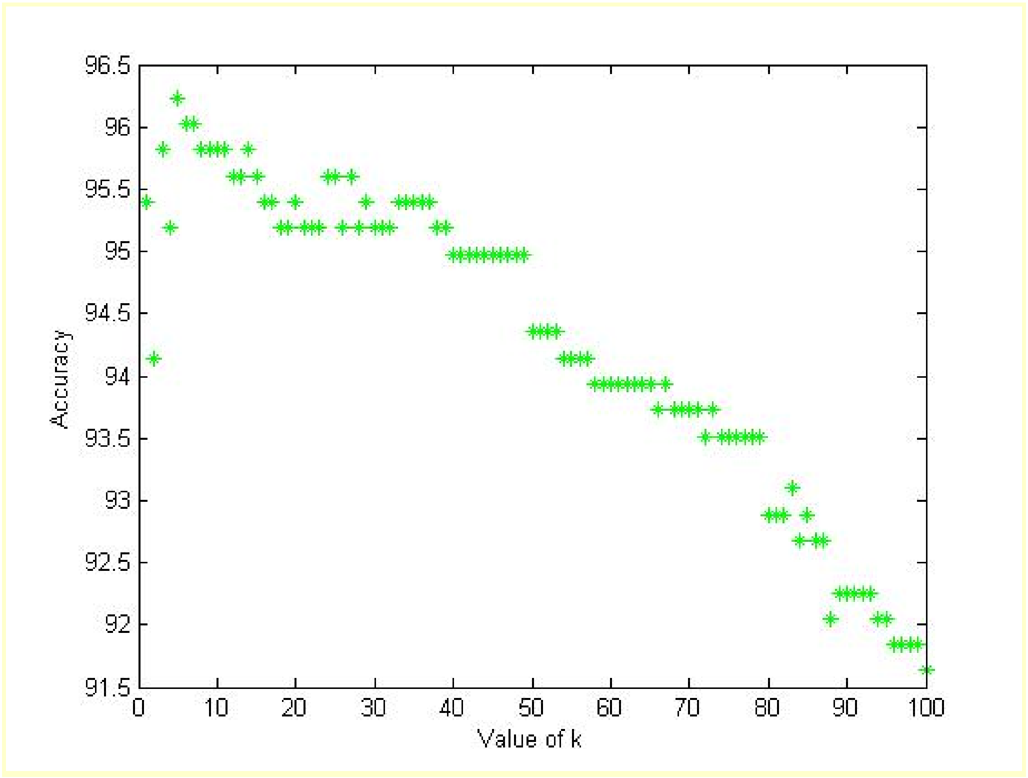
2.5 K Nearest Neighbour

Introduction

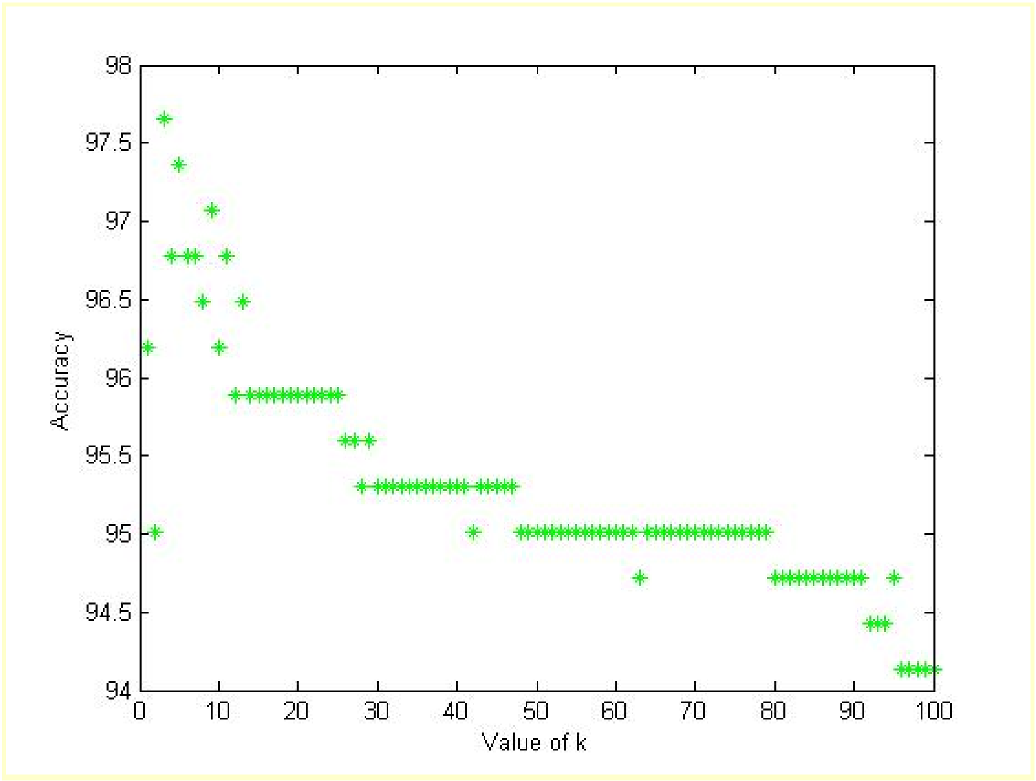
An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If*k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

We experimented by varying the value of k from 1 to 100 and comparing the corresponding predicted accuracy. The following graphs depict the variation of accuracy with the value of K.
Please note that the distance used was the euclidean distance.

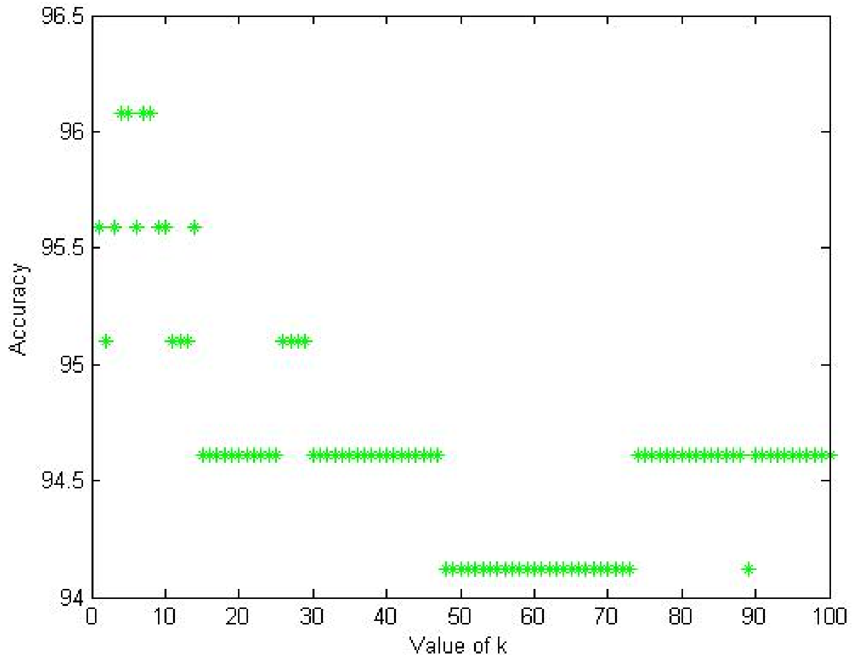
Breast Cancer data set (70% training data set)



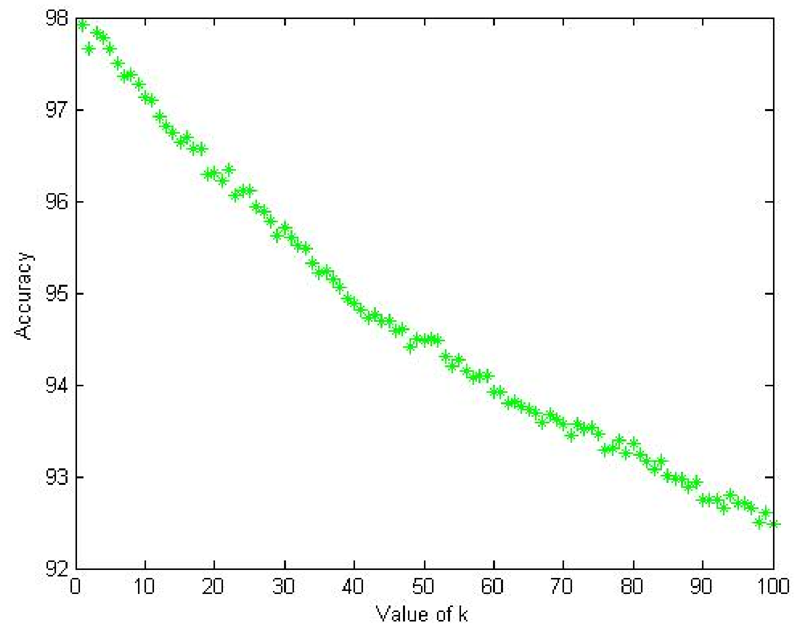
Breast Cancer data set (50% training data set)



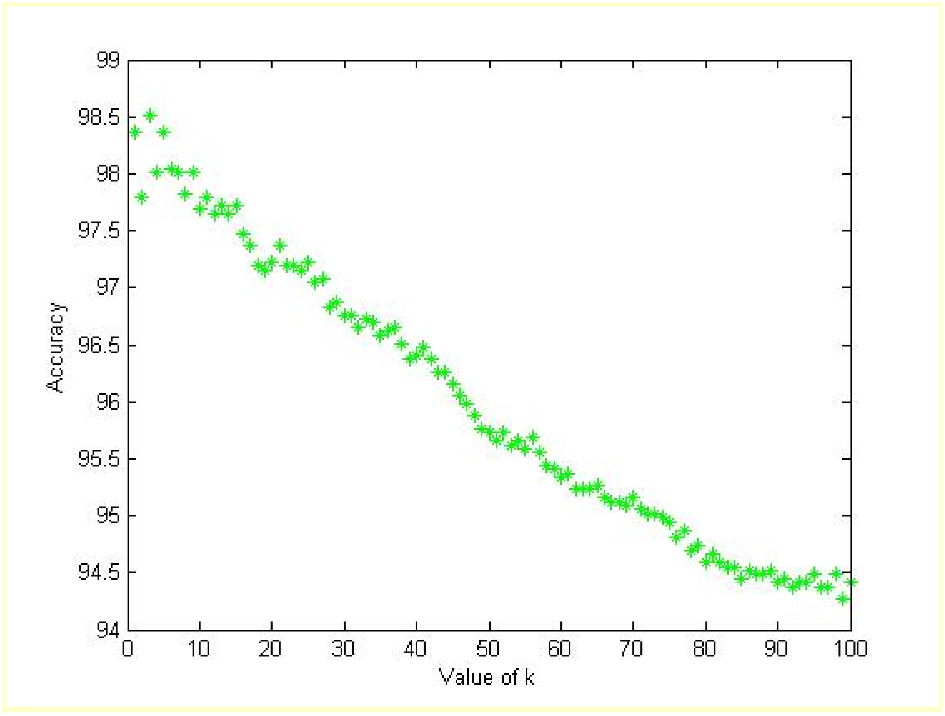
Breast Cancer data set (30% training data set)



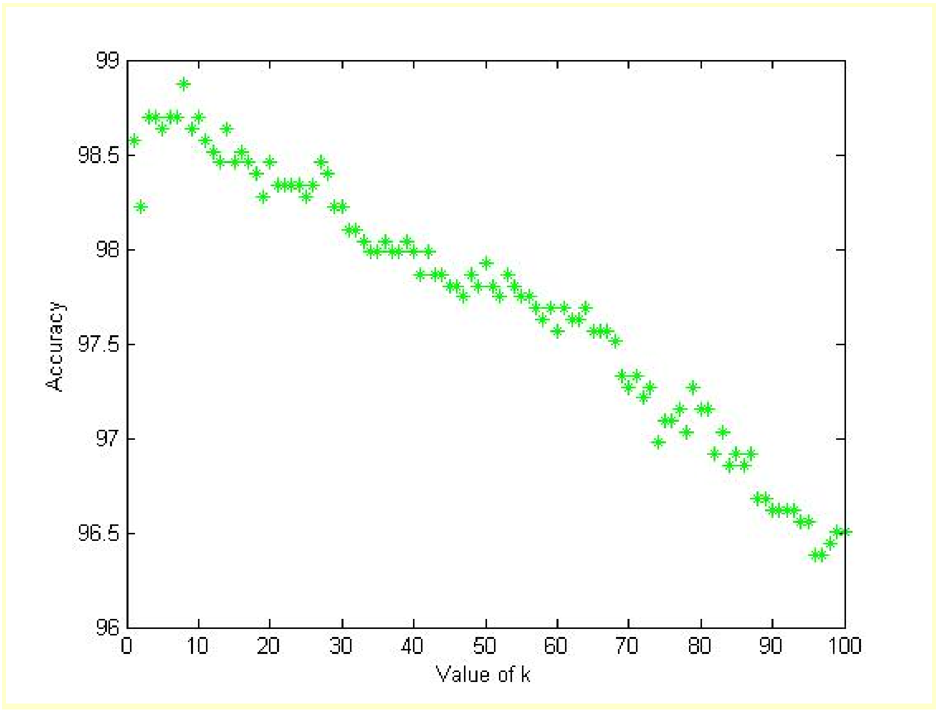
OCR data set with 70% training data



OCR data set with 50% training data

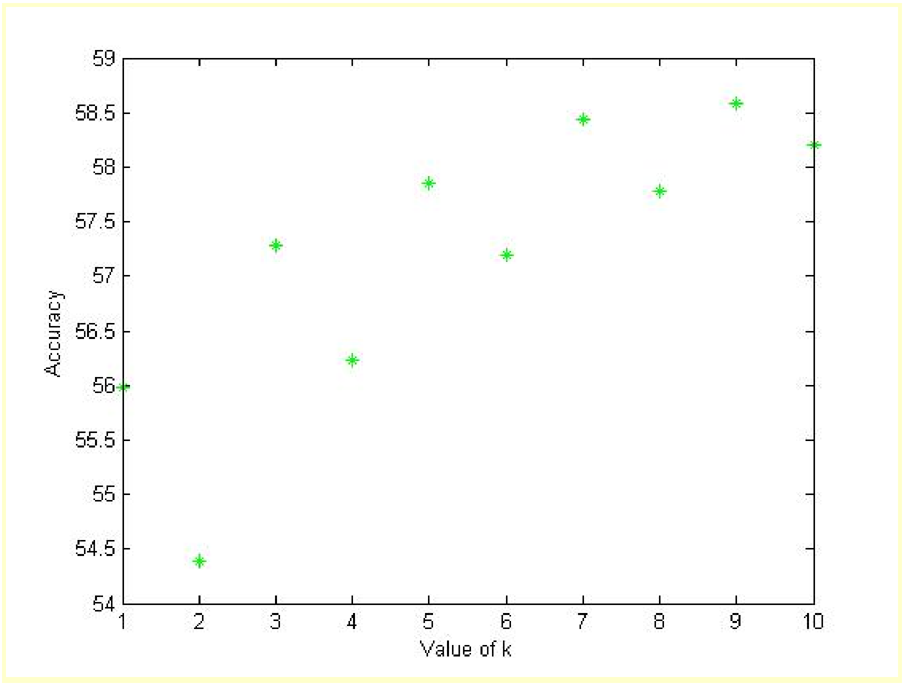


OCR data set with 30% training data

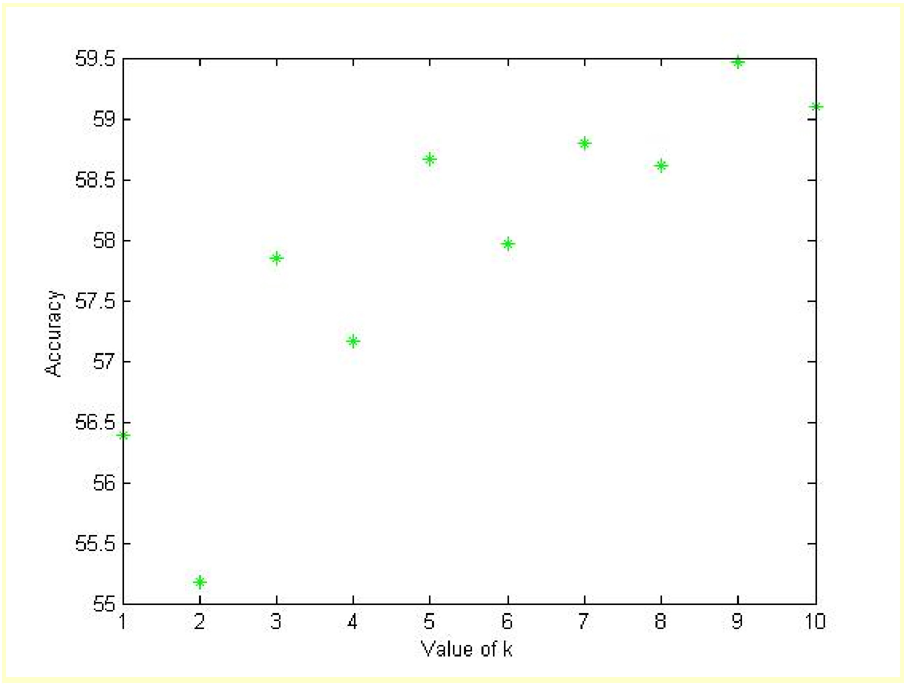


Note : For Higgs Boson data set, the values of k were varied from 1 to 10 because of performance constraints.

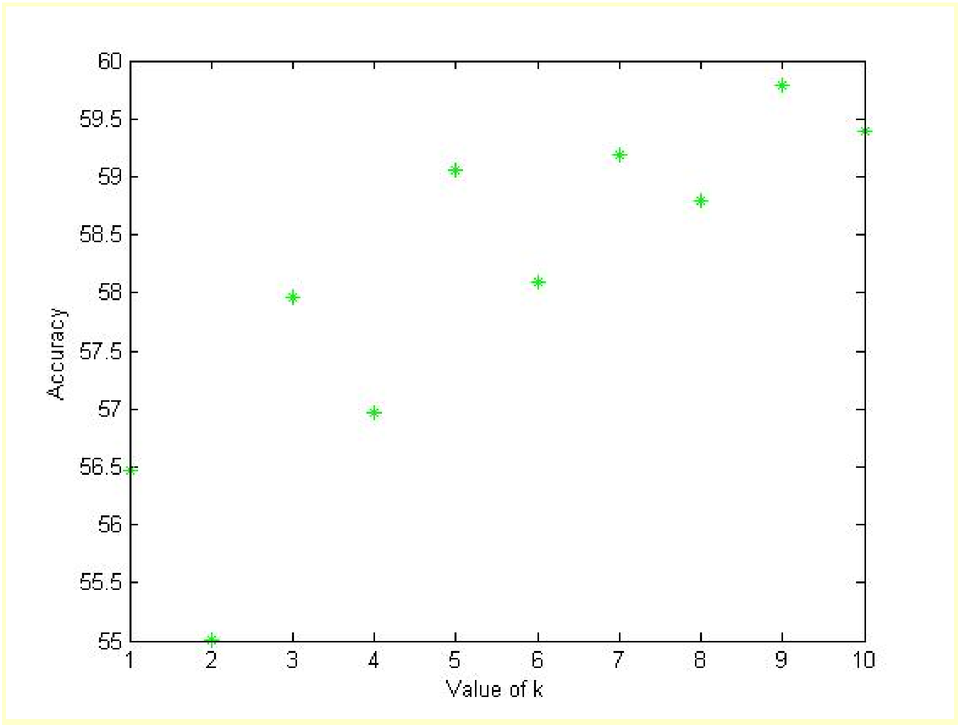
Higgs Boson data set with 70% training data



Higgs Boson data set with 50% training data



Higgs Boson data set with 30% training data



Conclusions.

1. We observed that the accuracy of the predicted results increases when we increase k from 1, but starts falling off after a certain point as the value of k increases.

2. The values of k for which the accuracy was noticed to be highest in a particular run for each of the data sets was found to be usually between 3 and 6 for Breast Cancer and OCR data sets and between 7-9 for the Higgs Boson Data set.

3. A very small value of K (such as 1 or 2) implies that the noise will have a higher influence in determining the result. A very high value of k (50 and beyond) increases the number of computations to be made and is not very efficient in terms of the running time of the algorithm and also in terms of accuracy, because in such cases the result will tend to favour the class which has higher probability than the other class.

	30%	50%	70%
Optimum accuracy (Breast Cancer)	96.24	97.37	96.24
Optimum k (Breast Cancer)	6,8	3	5
Optimum accuracy (OCR)	98.76	98.51	97.96
Optimum k (OCR)	8	3	1
Optimum accuracy (Higgs Boson)	59.78	59.5	58.58
Optimum k (Higgs Boson)	8	9	8

2.6 Naive Bayes

Naive Bayes classifier is designed for use when features are independent of one another within each class, but it appears to work well in practice even when that independence assumption is not valid. It is possible to use various distributions with each feature.

Normal (Gaussian) Distribution

The 'normal' distribution is appropriate for features that have normal distributions in each class. For each feature you model with a normal distribution, the Naive Bayes classifier estimates a separate normal distribution for each class by computing the mean and standard deviation of the training data in that class.

Kernel Distribution

The 'kernel' distribution is appropriate for features that have a continuous distribution. It does not require a strong assumption such as a normal distribution and you can use it in cases where the distribution of a feature may be skewed or have multiple peaks or modes

Multinomial Distribution

The multinomial distribution (specify with the 'mn' keyword) is appropriate when all features represent counts of a set of words or tokens. This is sometimes called the "bag of words" model. For the multinomial option, each feature represents the count of one token. The classifier counts the set of relative token probabilities separately for each class. The classifier defines the multinomial distribution for each row by the vector of probabilities for the corresponding class, and by N, the total token count for that row.

We compared the results by fitting different distributions over our features.
For each distribution it was compared with 30%, 50% and 70% training datasets.

Breast Cancer

Training Percentages	**30**	**50**	**70**
Normal Distribution	96.443515	96.480938	95.098039
Kernel Distribution	94.979079	95.601173	96.078431
Multinomial Distributior	86.192469	89.442815	85.294118
Multivariate Multinomial Distributior	97.280335	97.360704	98.039216

OCR Digits

Training Percentages	**30**	**50**	**70**
Normal Distribution	**-**	-	-
Kernel Distribution	86.117468	81.672598	86.358244
Multinomial Distributior	90.694127	90.249110	91.814947
Multivariate Multinomial Distributior	90.134757	90.676157	91.696323

Higgs

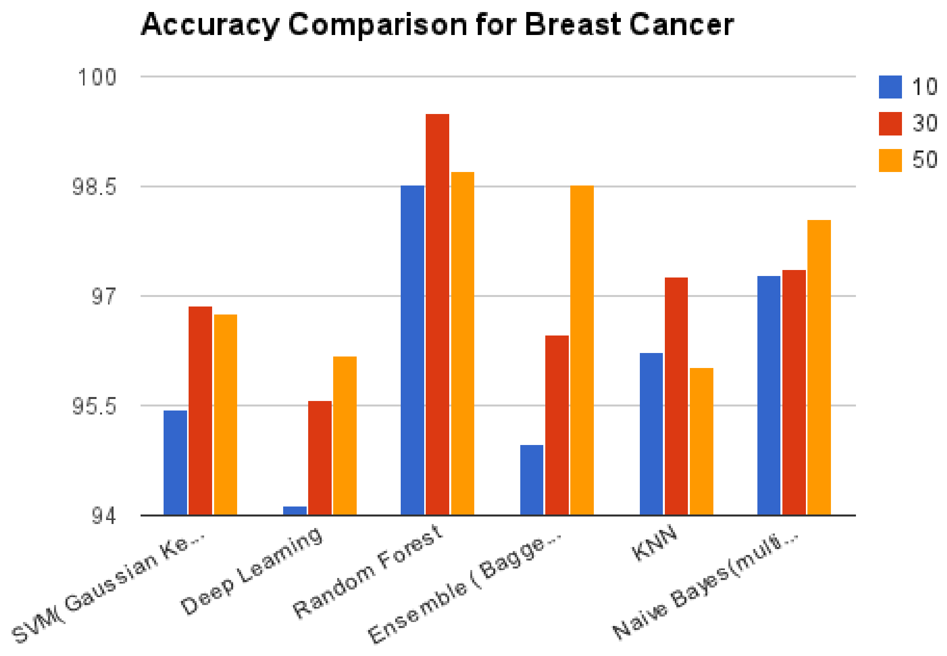
Training Percentages	**30**	**50**	**70**
Normal Distribution	60.237143	60.718000	65.098039
Kernel Distribution	66.245714	66.324000	69.078431
Multinomial Distributior	-	-	-
Multivariate Multinomial Distributior	-	-	-

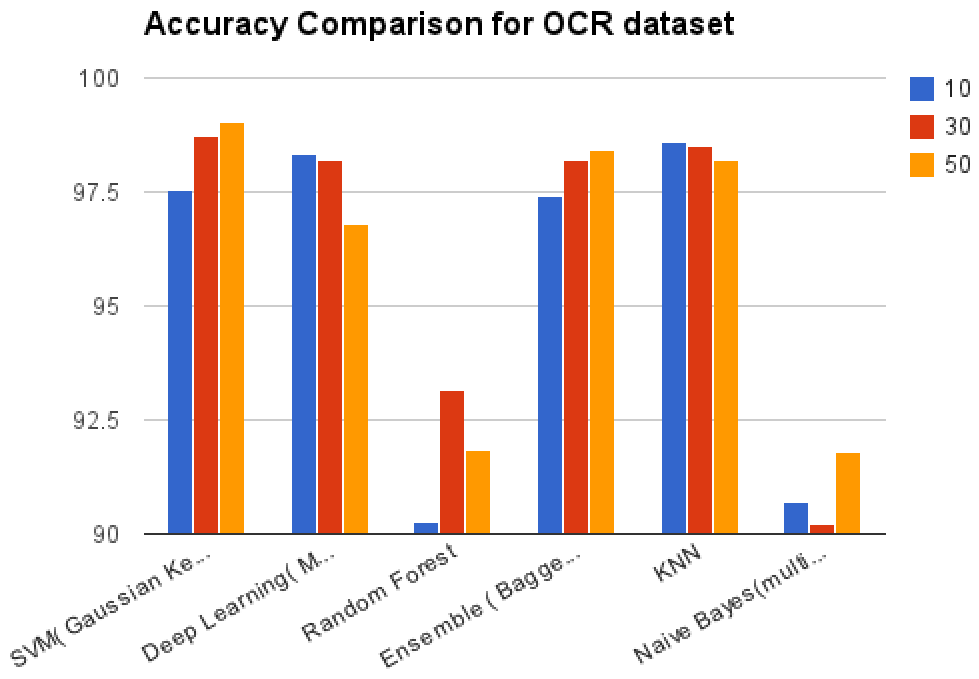
Conclusions

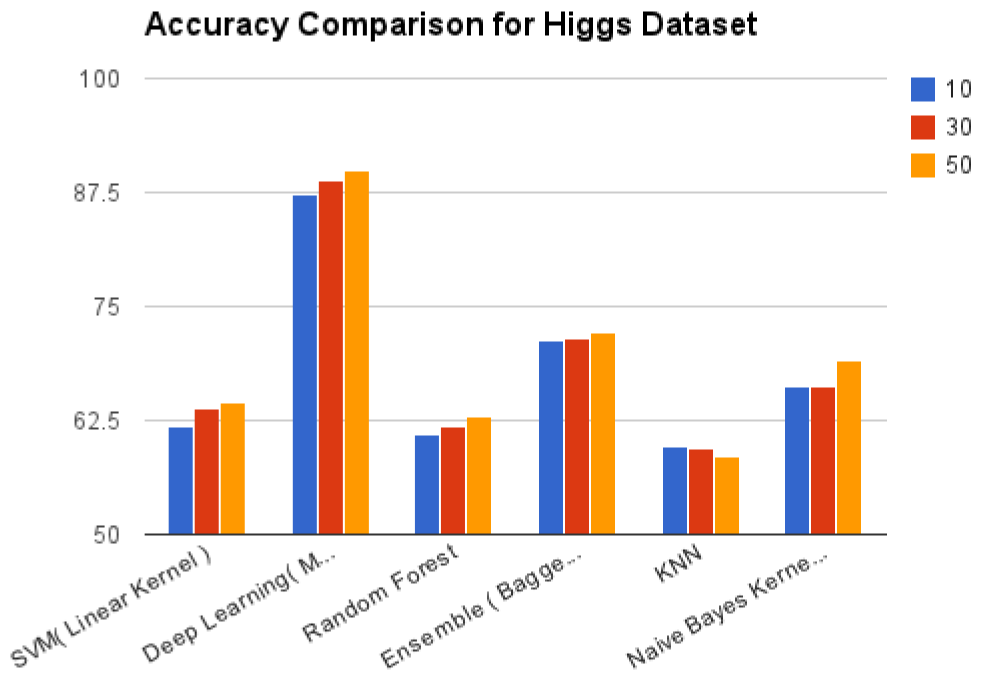
a.) For OCR data set : Using normal distribution was not possible because for using normal distribution within-class variance in each feature of TRAINING must be positive. While t**here were some features in OCR which do not follow normal distribution.**

b.) For Higgs data set : Data set for mn and **mvmn** distribution should only contain positive values. (MATLAB) Constraint.

3. Final Comparison of Accuracy Across Algorithms


Accuracy Comparison for Breast Cancer
100
98.5
97
95.5
94
10
30
50
SVM(Gaussian Ke...
Deep Learning
Random Forest
Ensemble (Bagge...
KNN
Naive Bayes(multi...


Accuracy Comparison for OCR dataset
100
97.5
95
92.5
90
10
30
50
SVM(Gaussian Ke...
Deep Learning(M...
Random Forest
Ensemble (Bagge...
KNN
Naive Bayes(multi...



Conclusion

a.) **Gaussian** kernel performed better than **linear** kernel for Breast Cancer Dataset and OCR data set while linear kernel better for higgs boson dataset.
b.) In Deep Learning "**Multi Layer Perceptron** " outperformed "**Stacked Auto Encoder**" across all datasets.
c.) **Bagged** Ensemble Methods outperformed **Boosted** Decision Tree across all data sets.
d.) Random Forest gave the best results for breast cancer dataset.
e.) Gaussian Kernel(SVM) gave the best results for OCR dataset.
f.) Multi Layer Perceptron gave the best results for Higgs Dataset.
g.) Since the breast cancer data set is linearly separable, an ensemble type algorithms e.g random forest, bagged decision trees performed better than other algorithms.
h.) Since OCR data set is well clustered, gaussian kernel SVM performed better than any other algorithm.
i.) Higgs data is very convoluted therefore deep learning algorithms like Multi layer perceptron gave better results than other algorithms.