CAP 6610

MACHINE LEARNING

SPRING 2016

TEAM

CHANDRA VIKAS RANGHABHATLA (UFID: 16998071)

DRUMIL DESHPANDE (UFID: 83598265)

AJINKYA RAJGURU (UFID: 91790974)

PRITHVI REDDY KOPPLU (UFID: 15789192)

ANANDA KISHORE SIRIVELLA (UFID: 99515080)

DATA SETS

1) Breast Cancer Wisconsin (Original) Data Set:

   This dataset reflects breast cancer data accumulated over eight groups of patients from 1989 to 1991. The data set has multivariate and categorical data and presents with classification problem. The attribute values in this dataset are real values and Integers. The dataset has attribute Information: id_number, Clump Thickness, Uniformity of Cell Size, Uniformity in Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses, Class. There were missing values in the dataset, so the dataset was first cleaned and then fed into different tools/programs. For cleaning the data mean of those features were used to replace the missing values. The data set had 699 patterns and 9 features.

2) Optical Recognition of Handwritten Digits Data Set:

   This dataset represents data of handwritten digits from 43 people which were then converted to 8*8 matrix. This 8*8 matrix is flattened for each pattern to give the 64 features in this dataset. The dataset has multivariate and categorical data and has Integer values. The dataset has 5620 patterns and 64 features.

3) Forest type mapping Data Set:

   This dataset contains data representing different forest characteristics. The dataset has multivariate and categorical data. The attributes contain real values and Integers. The dataset has attribute Information: Class, b1-b9 (ASTER image bands containing spectral information in the green, red, and near infrared wavelengths), pred_minus_obs_S_b1 - pred_minus_obs_S_b9, pred_minus_obs_H_b1 - pred_minus_obs_H_b9. The data set contains 525 patterns and 28 features. The dataset has class as first feature, for generic purpose the class label feature vector was moved to last position.

4) Wine Quality Data set (Red Wine dataset):

   This dataset contains data about red wine and its characteristics and presents with the problem of classification or regression on quality of the red wine. The dataset has multivariate and non-characteristic. The attribute values are real values and Integers. Dataset has attribute information: fixed acidity, volatile acidity, citric acidity, residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide, density, pH, sulphates, alcohol, and quality. The dataset contains 1600 patterns and 11 features. This data set contains highly noisy data.

5) Balance Scale Data Set:

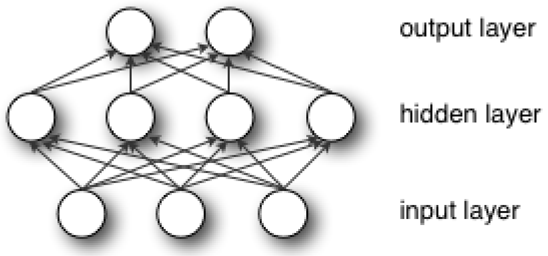
This data set was generated to model psychological experimental results. The data set contains data on weight imbalance on a balance scale. The attributes contain only integer values. The dataset has multivariate and characteristic data. Dataset has attribute information: Class name, Left Weight, Left Distance, Right Weight, Right Distance. The dataset contains 625 patterns and has 4 features.

DEEP LEARNING

Deep learning networks use a cascade of many layers of non-linear computing layers for feature extraction and transformation of features. Each layer uses the output of the previous layer as an input to operate upon. Some deep learning networks have more relations such as shared weights, feature maps between layers etc. in order to improve the learning curve. Thus deep networks learn multiple levels of representation of input data that corresponds to different levels of abstraction. In this case we have used two types of deep learning networks. First, Multi-layer perceptron which is a standard deep network for all the datasets mentioned in the report. We also found that Image data is better classified using a convolutional neural network. So, we also used CNN for the Optical Handwritten Dataset.

Multilayer Perceptron

An MLP with a single hidden layer can be represented graphically as follows:



Formally, a one-hidden-layer MLP is a function$f : R^D \rightarrow R^L$, where $D$is the size of input vector $x$and $L$is the size of the output vector$f(x)$, such that, in matrix notation:

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

with bias vectors $b^{(1)}$, $b^{(2)}$; weight matrices $W^{(1)}$, $W^{(2)}$and activation functions $G$and s.

The vector $h(x) = \Phi(x) = s(b^{(1)} + W^{(1)}x)$constitutes the hidden layer. $W^{(1)} \in R^{D \times D_h}$ is the weight matrix connecting the input vector to the hidden layer. Each column $W_{\cdot i}^{(1)}$ represents the weights from the input to the i-th hidden node. Typical choices for s include $tanh$, with $tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$, or the logistic $sigmoid$function, with $sigmoid(a) = 1/(1 + e^{-a})$. In this deep network we have used $tanh$ it gives faster training (and sometimes better local minima) than $sigmoid$. The output vector is then obtained as: $o(x) = G(b^{(2)} + W^{(2)}h(x))$.

The MLP used here has many free parameters to tune. Also it is not feasible to optimize set of values for these parameters also gradient descent cannot be readily applied to these parameters. The parameters involved in building and tuning the multi-layer perceptron are:

a) Introduction of Non-linearity:

The used MLP net uses sigmoidal and tanh functions which are one of the most common function to introduce non-linearity. They are used as they tend to produce zero-mean inputs to the next layers of the perceptron. We also found that tanh function gives better convergence properties during our literature research.

b) Weight Initialization:

Typically the initial weights chosen are small and near zero, so that the activation function acts like a linear function initially. Deep networks also usually conserve variance of activation as well as variance of back propagation from one layer to another. This allows information learned from data to flow back and forth between the layers in the deep network and reduces the discrepancies between the layers. We found that the best way to initialize weights is to use: $uniform[-\frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]$for tanh and $uniform[-4 * \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}, 4 * \frac{\sqrt{6}}{\sqrt{fan_{in}+fan_{out}}}]$for sigmoid. Where fan_{in} is the number of inputs and $fan_{out}$the number of hidden nodes in that layer.

c) Number of Hidden Layers Nodes:

This parameter is usually lot more dependent upon the dataset that the deep network is operating upon. So the more complex the distribution of the dataset is the more number of hidden nodes would be required to model the data, and hence larger would be the number of hidden layer nodes. We tried to find optimal number of hidden layers for each dataset using 100-1000 hidden layer nodes with step over of 100 nodes.

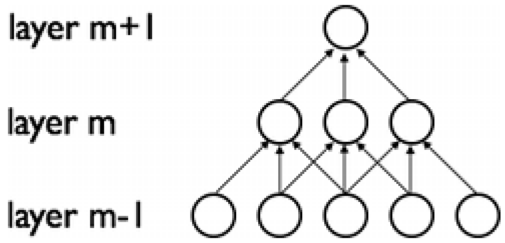
d) Learning Rate:

Usually the learning rate of a deep network is kept as a constant value in contrast to using a gradient descent or any other technique. Another way to tune the learning rate is to reduce the learning rate over time and choose a value that minimizes the validation error. In this implementation of Deep network we have used a constant learning rate.

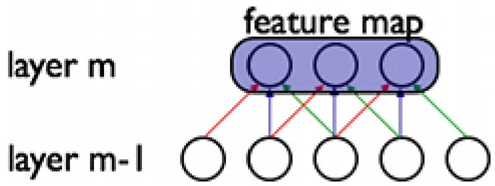
At the end of each layer a negative log likelihood optimization is used to minimize the misclassification error. This is done for a number of iterations till the misclassification error converges to some value.

Convolutional Neural Networks:

Convolutional Neural Networks are inspired from biological networks (organization of the animal visual cortex) and are variants of Multilayer Perceptron. Convolutional Neural Networks are popular when the datasets that need to be classified are images. In such cases the CNN consists of multiple layers (depending upon the data) of small neuron collection which represent portions of the test/train image pattern. These layers are called as receptive fields. The outputs of these collections are then tiled so that their input regions overlap, to obtain a better representation of the original image. This is then repeated for every layer. Tiling allows CNNs to tolerate translation of the input image. Thus CNN make use of spatial-local correlations by enforcing a local connectivity between neurons in adjacent layers. That is, the input to hidden nodes a layer (M) in the CNN are from a subset of hidden nodes in the layer (M-1) that have spatially contiguous receptive fields. This can be seen in the figure :



CNN thus contains a feature map between the consecutive layers. That is, weights of mapped hidden nodes in layer (M-1) and layer (M) are same. This weight sharing uses learning efficiency by reducing the number of free parameters to be learnt. The following image depicts feature map:



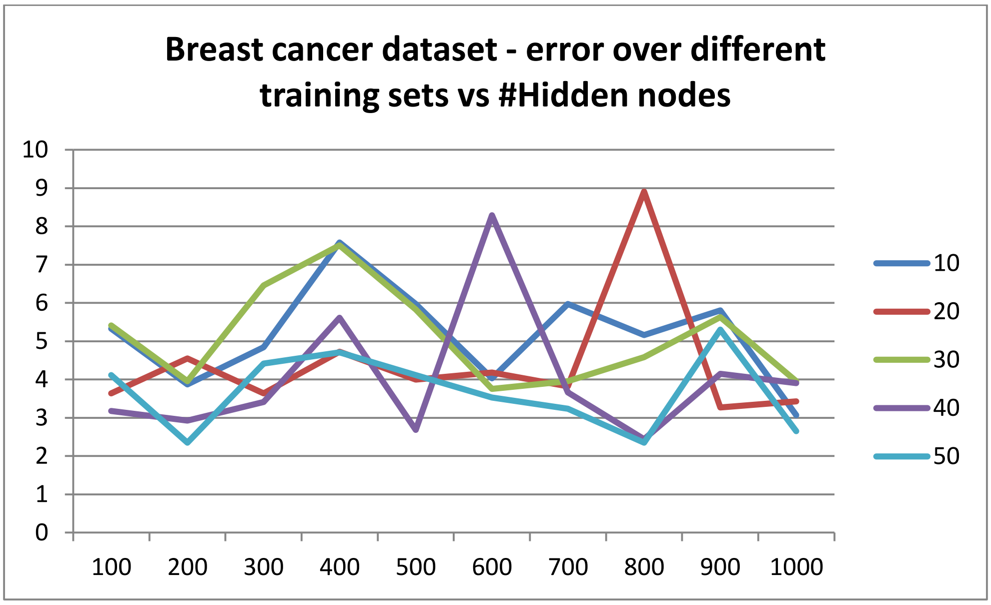
This feature map is obtained in the used CNN by recursively applying a function across sub regions of the entire input image pattern i.e. by convolution of input image pattern with a linear filter then adding a bias term and then applying a non-linearity function to it. So the k-th feature map at a given layer as h^k, whose filters are determined by the weights $W^k$and bias is b_k, then the feature map $h^k$is obtained as follows by using: $h^k_{ij} = \tanh((W^k * x)_{ij} + b_k).$ Thus, Convolutional Neural Networks are believed to work better on image datasets. However, the data of Optical Handwritten Numbers are flattened image features. So each pattern first needed to be transformed from a 64 length row vector to a 8*8 matrix representing the image pattern.

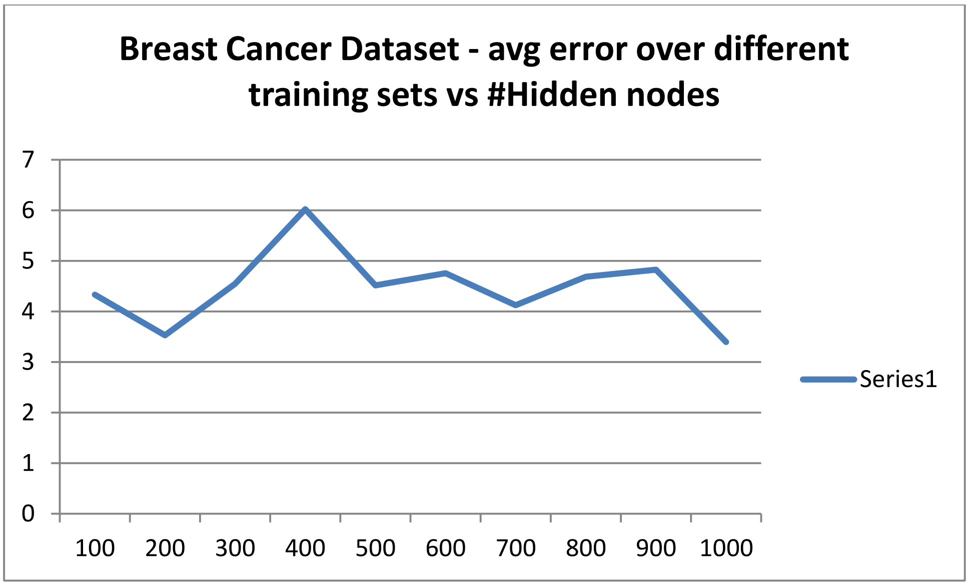
1. Breast Cancer Wisconsin (Original) Data Set:

1. Error over different datasets vs Number of Hidden Nodes.

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
100	5.323	3.636	5.413	3.173	4.117	4.3324
200	3.871	4.545	3.958	2.926	2.352	3.5304
300	4.839	3.636	6.458	3.414	4.411	4.5516
400	7.581	4.727	7.5	5.609	4.705	6.0244
500	5.968	4	5.833	2.682	4.117	4.52
600	4.032	4.181	3.75	8.292	3.529	4.7568
700	5.968	3.818	3.958	3.658	3.235	4.1274
800	5.161	8.909	4.583	2.439	2.352	4.6888
900	5.806	3.272	5.625	4.146	5.298	4.8294
1000	3.065	3.427	3.958	3.902	2.647	3.3998

This can be further illustrated by following graphs:

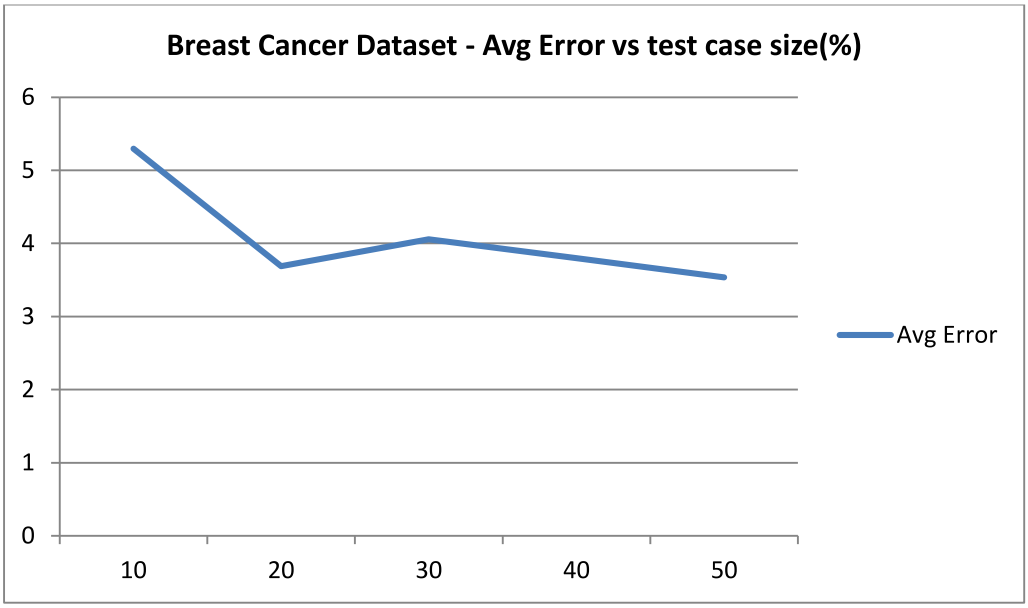




2. Average Error vs Percentage of Test Data:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	5.12	3.45	3.68	5.25	3.38
2	5.12	3.82	3.68	2.75	2.77
3	3.2	3.09	3.15	4.5	3.69
4	5.6	3.27	4.84	2.25	4.31
5	5.44	4	2.73	4.75	2.77
6	8.32	4.91	4.42	5	4.92
7	6.88	4.73	4.63	3.5	3.38
8	3.68	3.09	6.31	3.5	3.38
9	5.12	3.63	3.57	4	3.38
10	4.48	2.91	3.57	2.5	3.38
Avg. Error	5.296	3.69	4.058	3.8	3.536

This can be further illustrated by following graph:

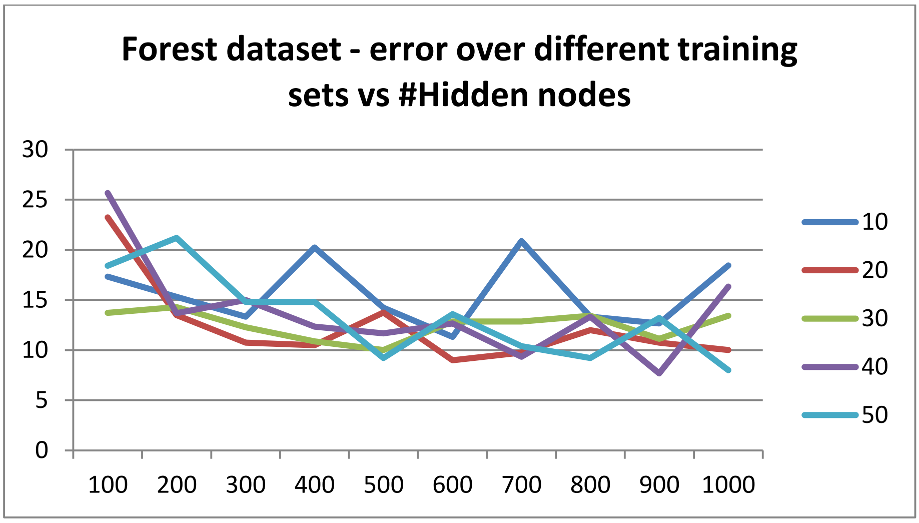


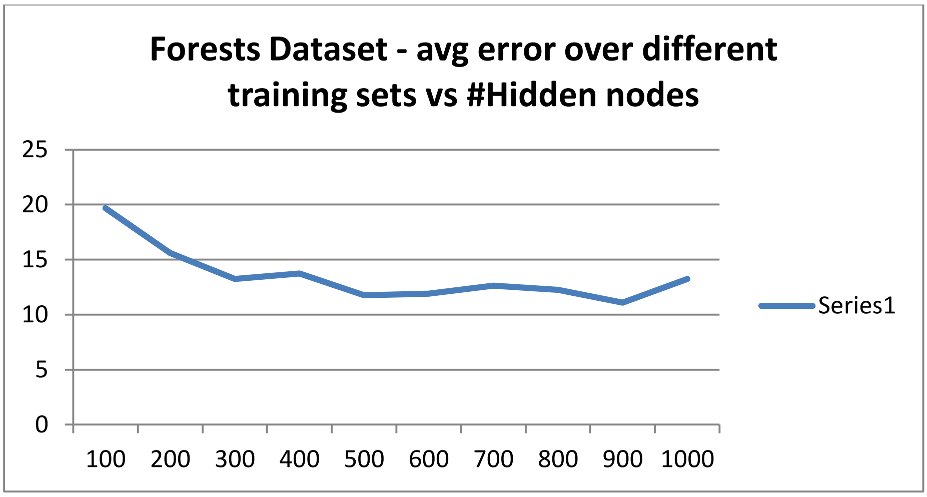
2. Forest type mapping Data Set:

1. Error over different datasets vs Number of Hidden Nodes:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
100	17.33	23.25	13.71	25.67	18.4	19.672
200	15.33	13.5	14.28	13.67	21.2	15.596
300	13.33	10.75	12.28	15	14.8	13.232
400	20.22	10.5	10.87	12.33	14.8	13.744
500	14.22	13.75	10	11.67	9.2	11.768
600	11.33	9	12.85	12.67	13.6	11.89
700	20.89	9.75	12.85	9.33	10.4	12.644
800	13.33	12	13.42	13.33	9.2	12.256
900	12.67	10.75	11.14	7.67	13.2	11.086
1000	18.44	10	13.42	16.33	8	13.238

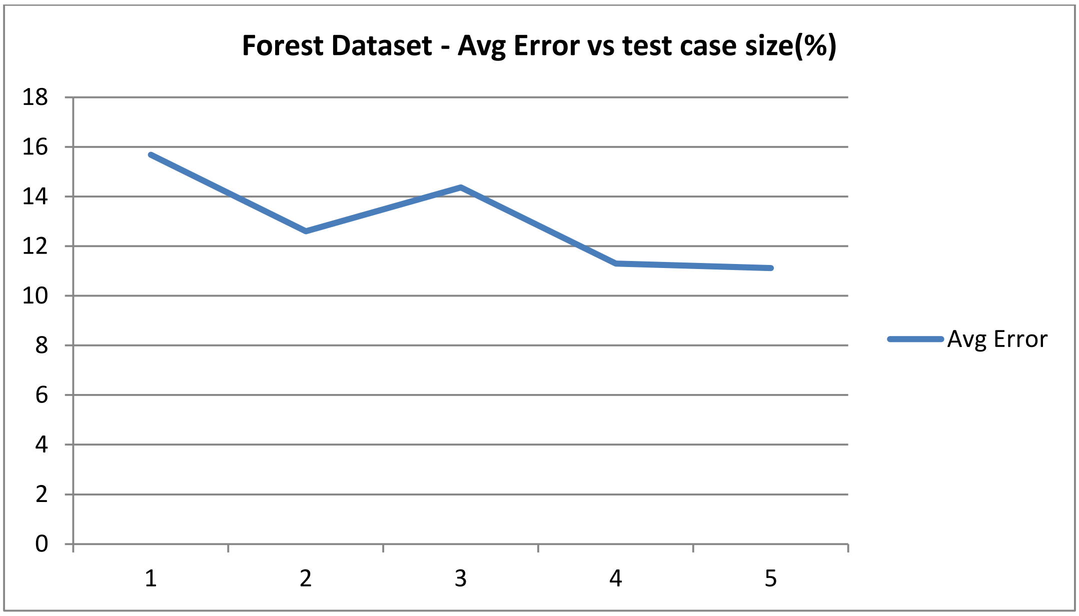
This data can be further illustrated by the figures below:





2. Average Error vs Percentage of Test Data:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	11.78	12.25	14.57	12.67	12
2	13.78	15.5	18.85	10.33	12.4
3	17.33	9.75	9.14	10.67	10.4
4	10.89	12.25	10.57	16	6.4
5	14	9.75	13.71	9.33	14
6	16.89	21.5	22.28	11.67	13.2
7	26.44	13	14.85	11	9.6
8	16.44	12.75	15.71	10	10.4
9	16.67	8.75	13.42	10	10.4
10	12.67	10.5	10.57	11.33	12.4
Avg. Error	15.689	12.6	14.367	11.3	11.12



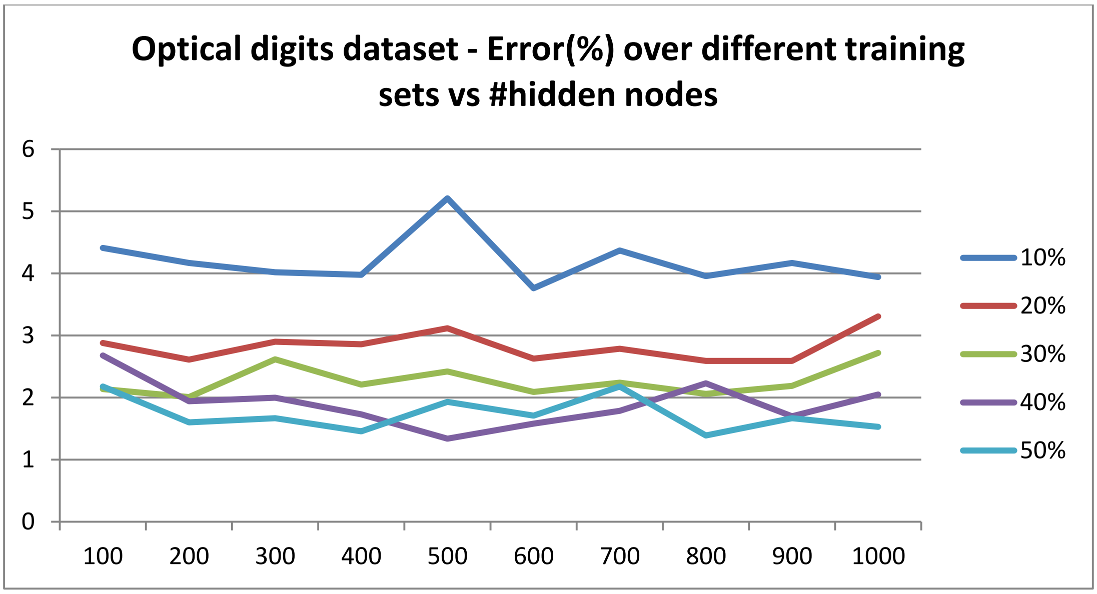
Optical Recognition of Handwritten Digits Data Set:

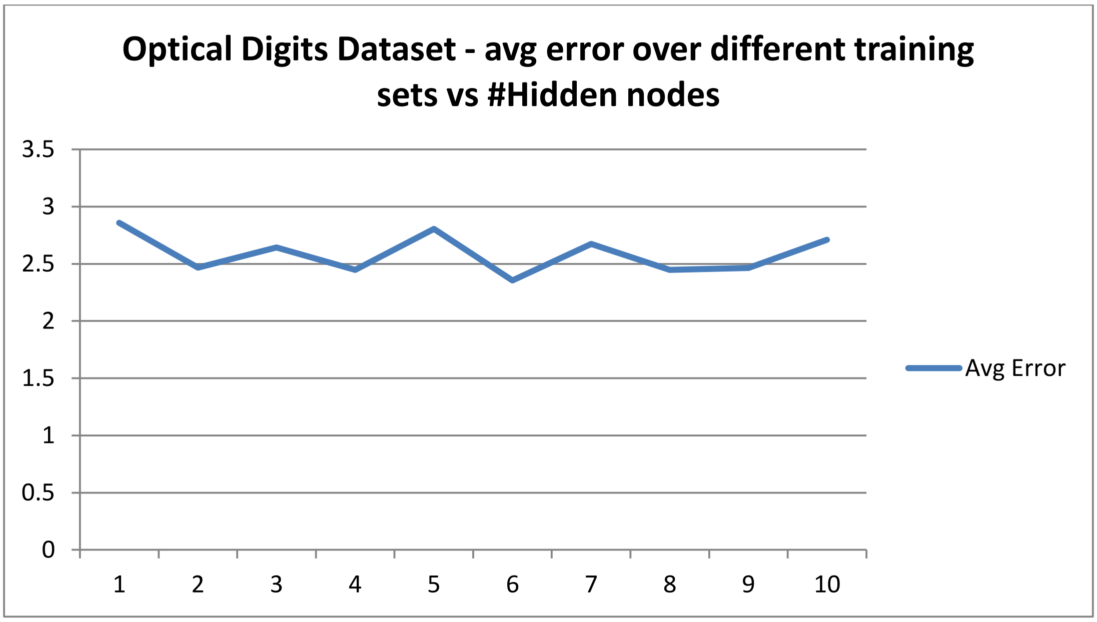
a) For Multi layer Perceptron

1. Error over different datasets vs Number of Hidden Nodes:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
100	4.41	2.88	2.14	2.68	2.18	2.858
200	4.17	2.61	2.01	1.94	1.6	2.466
300	4.02	2.9	2.62	2	1.67	2.642
400	3.98	2.86	2.21	1.73	1.46	2.448
500	5.21	3.12	2.42	1.34	1.93	2.804
600	3.76	2.63	2.09	1.58	1.71	2.354
700	4.37	2.79	2.24	1.79	2.18	2.674
800	3.96	2.59	2.06	2.23	1.39	2.446
900	4.17	2.59	2.19	1.7	1.67	2.464
1000	3.94	3.31	2.72	2.05	1.53	2.71

The following data can be Illustrated by the figures below:

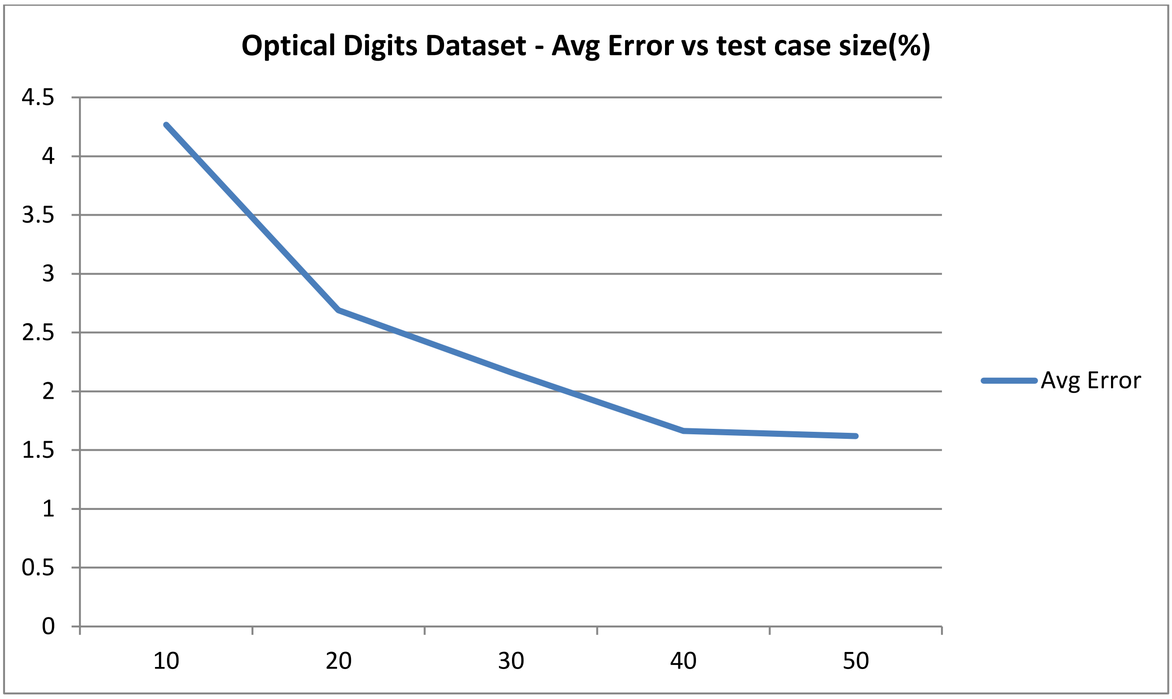




2. Average Error vs Percentage of Test Data:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	3.7	2.72	2.01	1.88	1.53
2	4.71	2.03	2.21	1.73	1.5
3	3.74	2.72	1.96	1.64	1.68
4	5.21	3.12	2.42	1.34	1.93
5	3.98	2.86	2.21	1.73	1.46
Avg. Error	4.268	2.69	2.162	1.664	1.62

This can be further illustrated by following graph:

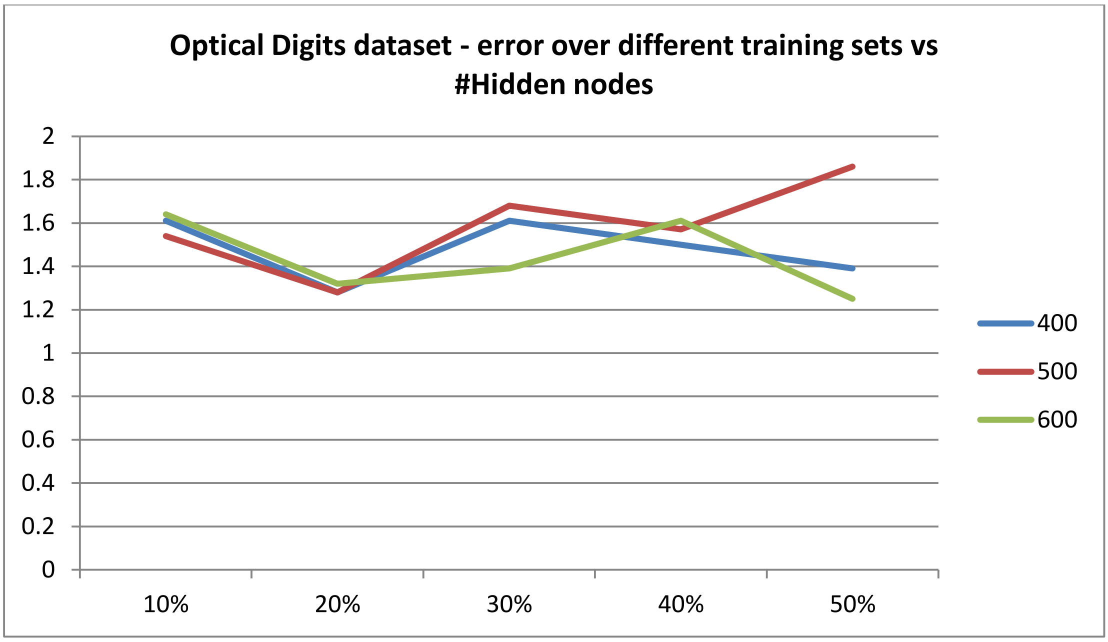


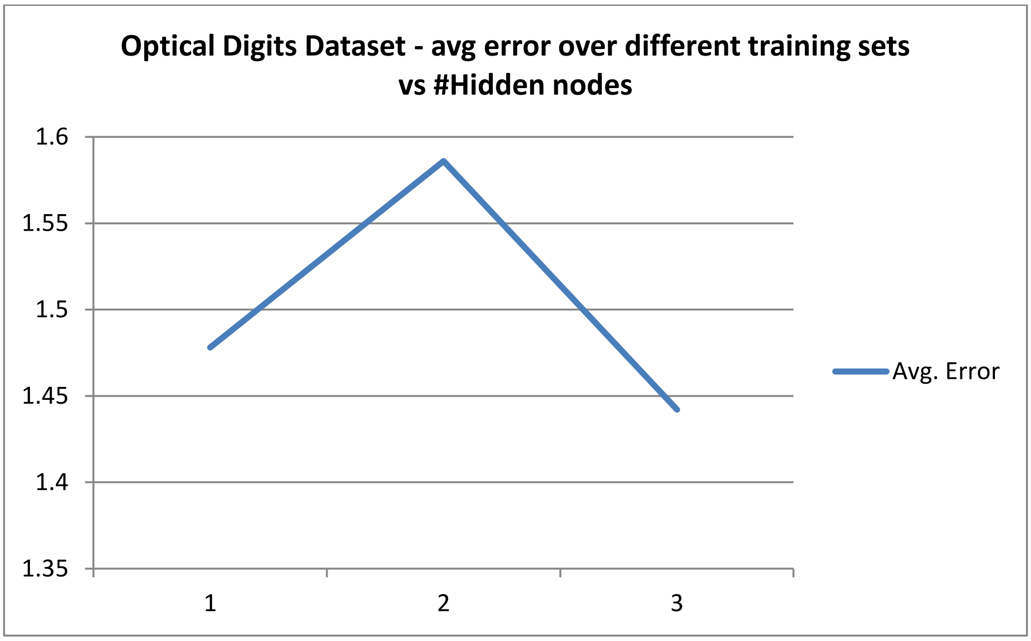
b) For Convolutional Neural Network:

Error over different datasets vs Number of Hidden Nodes:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
400	1.61	1.28	1.61	1.5	1.39	1.478
500	1.54	1.28	1.68	1.57	1.86	1.586
600	1.64	1.32	1.39	1.61	1.25	1.442

The following data can be illustrated by the figures below:

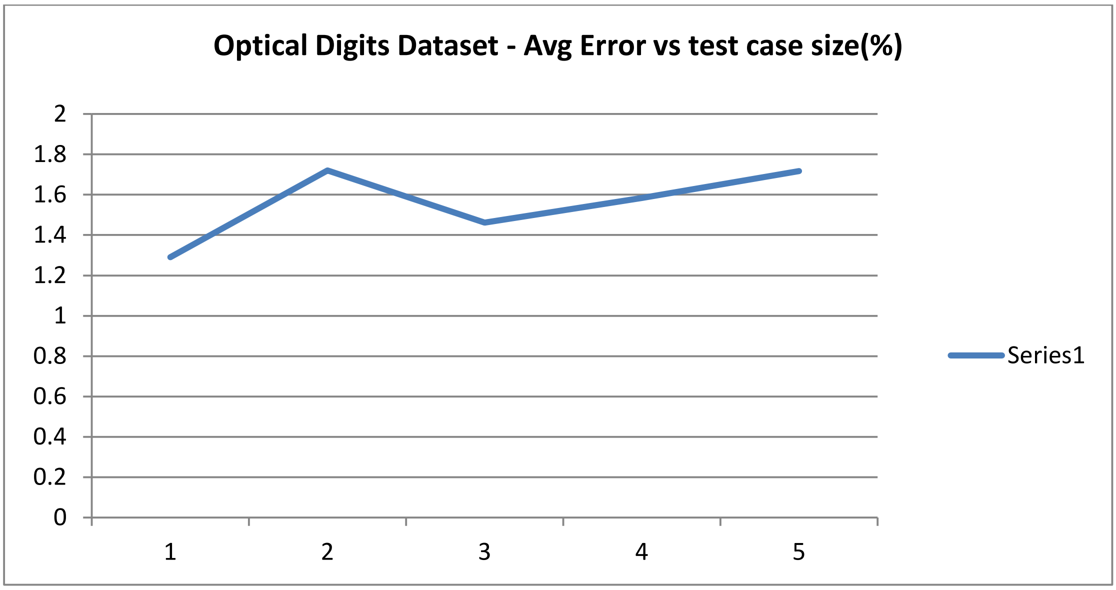




2. Average Error vs Percentage of Test Data:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	1.53	1.5	1.5	1.64	1.39
2	1.035	1.5	1.17	1.32	1.67
3	1.53	1.96	1.57	1.78	2
4	1.32	2.14	1.68	1.61	1.86
5	1.035	1.5	1.39	1.57	1.67
Avg. Error	1.29	1.72	1.462	1.584	1.718

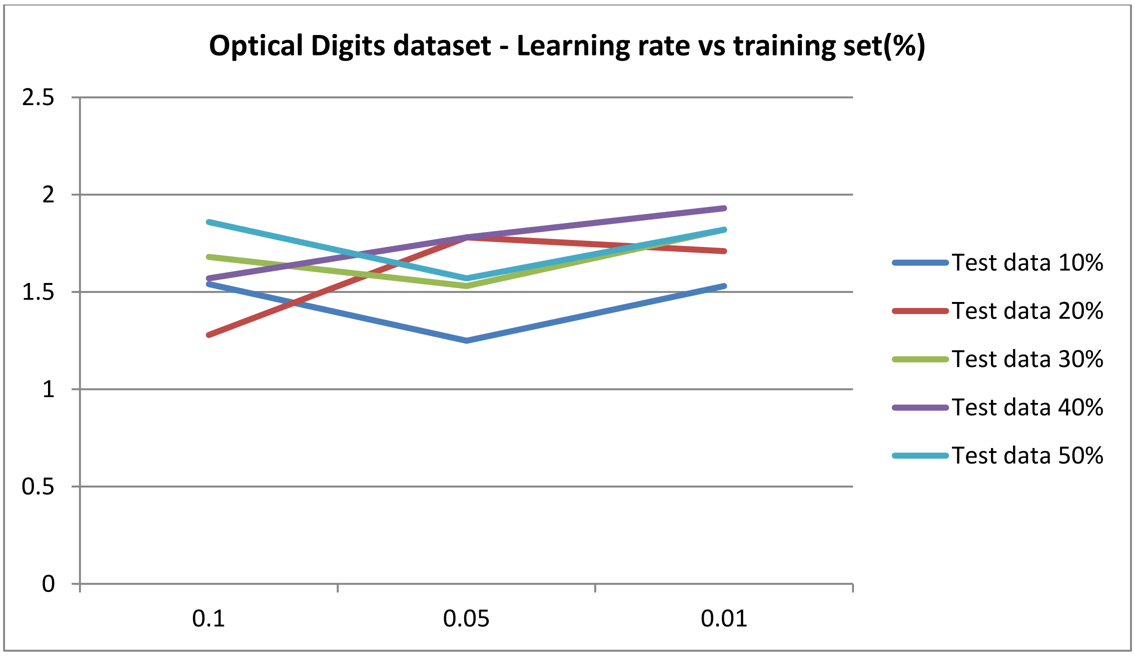
This can be further illustrated by following graph:



3. Learning rate vs Training set:

	Test data 10%	Test data 20%	Test data 30%	Test data 40%	Test data 50%
0.1	1.54	1.28	1.68	1.57	1.86
0.05	1.25	1.78	1.53	1.78	1.57
0.01	1.53	1.71	1.82	1.93	1.82

The above data is better illustrated by figure below:

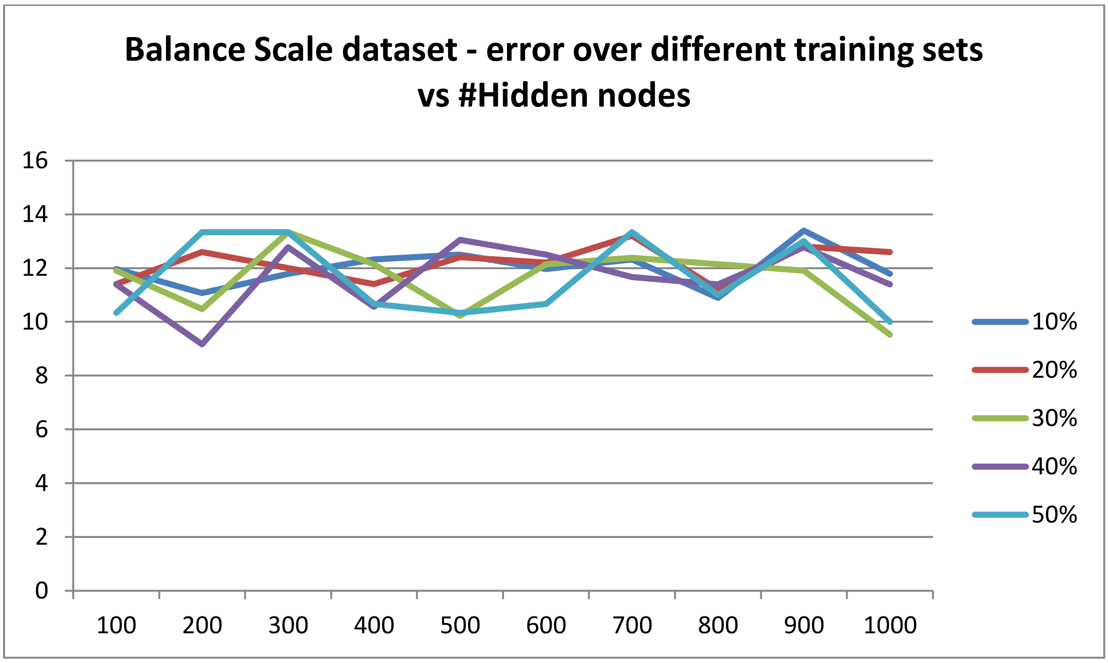


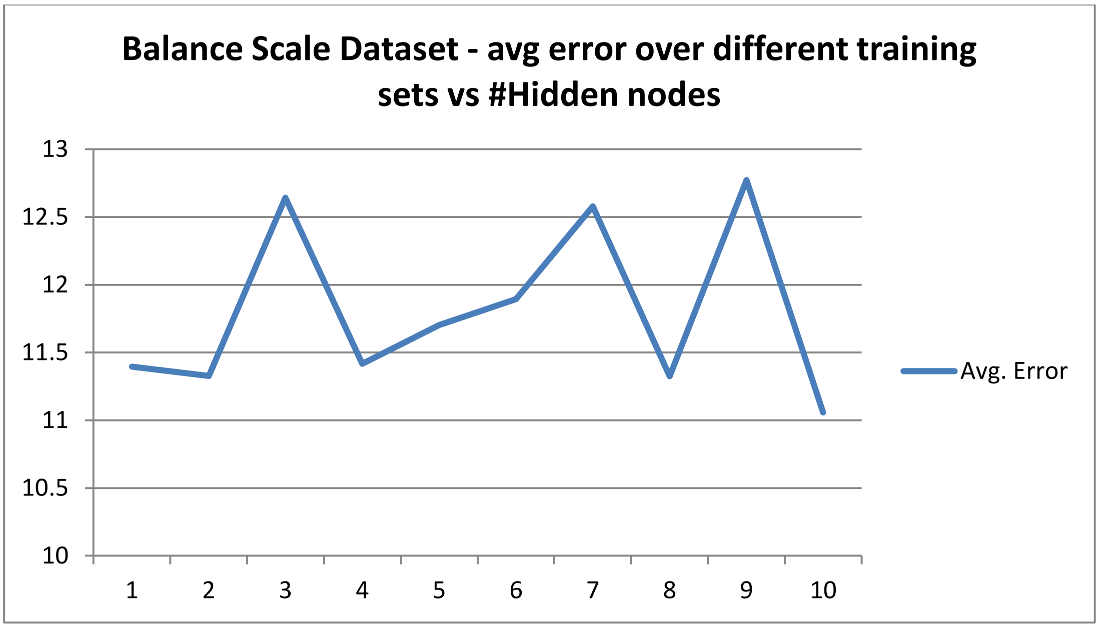
4. Balance Scale Data Set:

1. Error over different datasets vs Number of Hidden Nodes:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
100	11.96	11.4	11.9	11.39	10.33	11.396
200	11.07	12.6	10.47	9.17	13.33	11.328
300	11.78	12	13.33	12.78	13.33	12.644
400	12.32	11.4	12.14	10.56	10.67	11.418
500	12.5	12.4	10.23	13.05	10.33	11.702
600	11.96	12.2	12.14	12.5	10.67	11.894
700	12.32	13.2	12.38	11.67	13.33	12.58
800	10.89	11.2	12.14	11.39	11	11.324
900	13.39	12.8	11.9	12.78	13	12.774
1000	11.78	12.6	9.52	11.39	10	11.058

The following data can be Illustrated by the figures below:

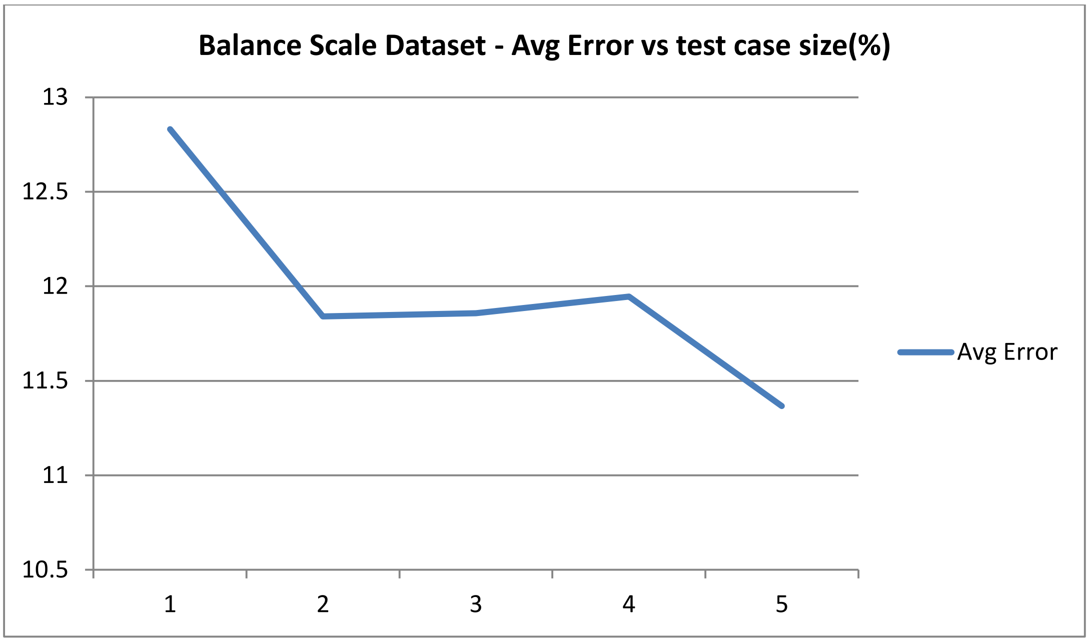




2. Average Error vs Percentage of Test Data:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	12.54	12.2	11.29	12	11.33
2	10.72	11.8	11.53	12.8	11.33
3	14.36	11.4	12.47	12	11
4	16	11.2	11.76	11.73	8
5	10.72	11.4	11.76	11.2	10.33
6	15.63	11	11.53	11.2	13.33
7	11.09	12.6	12.47	11.2	12
8	12.9	11.4	11.76	10.67	13
9	12.9	11.4	12.25	13.33	11.67
10	11.45	14	11.76	13.33	11.67
Avg. Error	12.831	11.84	11.858	11.946	11.366

This can be further illustrated by following graph:

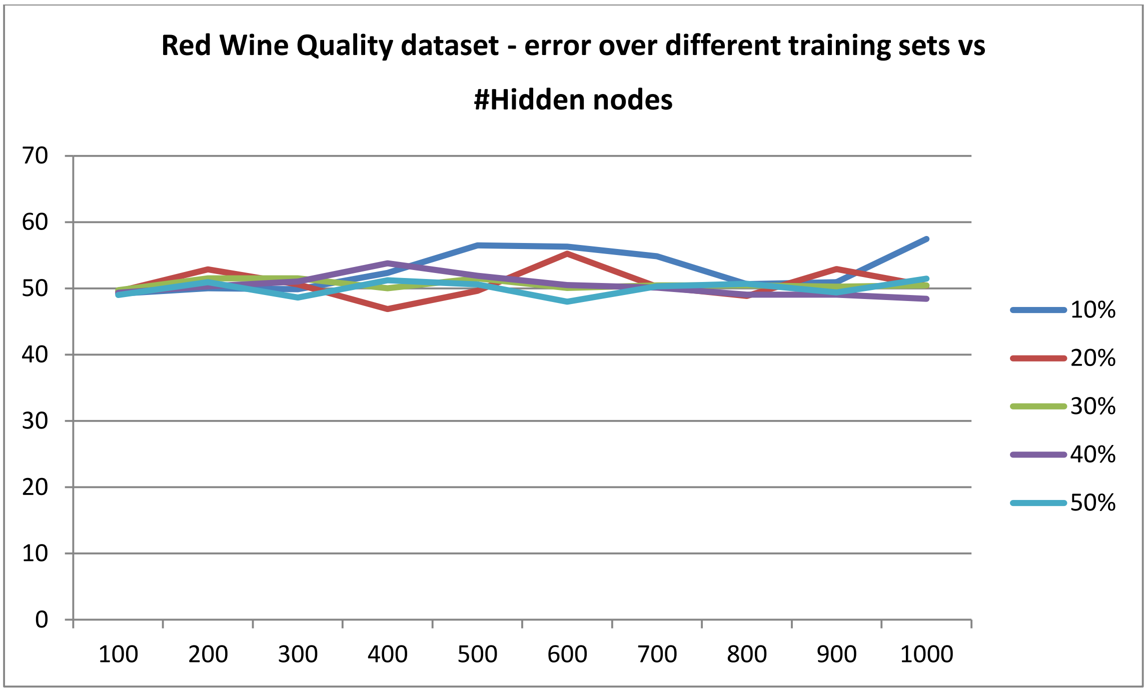


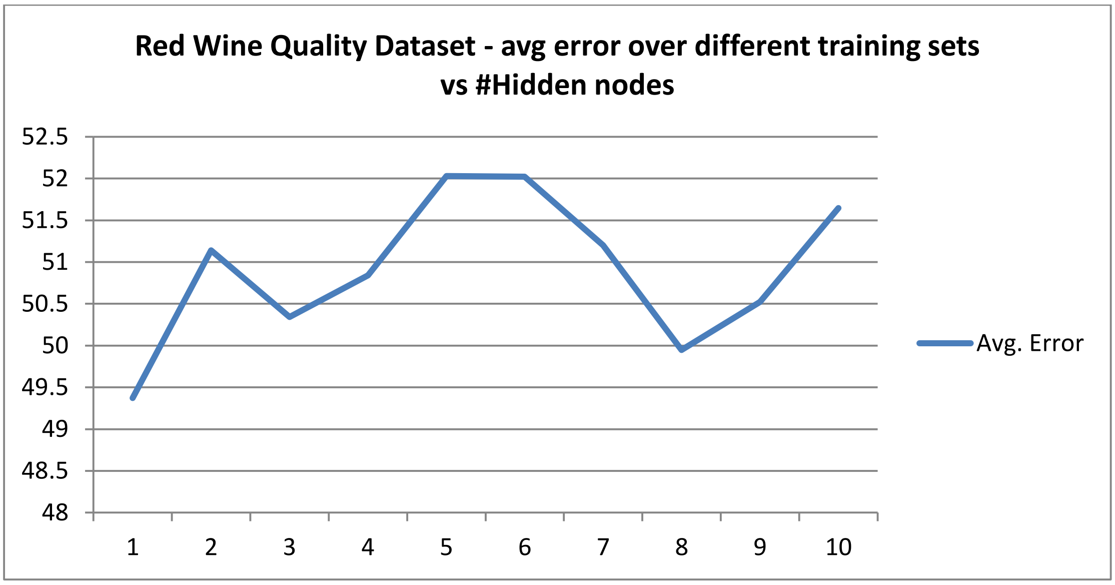
5. Red Wine Quality Data Set:

1. Error over different datasets vs Number of Hidden Nodes:

# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data	Average Error
100	49.19	49.56	49.72	49.36	49.03	49.372
200	50.03	52.86	51.54	50.31	50.96	51.14
300	49.89	50.588	51.54	51.05	48.64	50.3416
400	52.35	46.863	50	53.78	51.22	50.8426
500	56.49	49.647	51.54	51.89	50.58	52.0294
600	56.28	55.216	50.09	50.52	48	52.0212
700	54.87	50.27	50.45	50.1	50.32	51.202
800	50.67	48.86	50.45	49.05	50.71	49.948
900	50.94	52.94	50.27	49.05	49.41	50.522
1000	57.47	50.43	50.45	48.41	51.48	51.648

The following data can be Illustrated by the figures below:

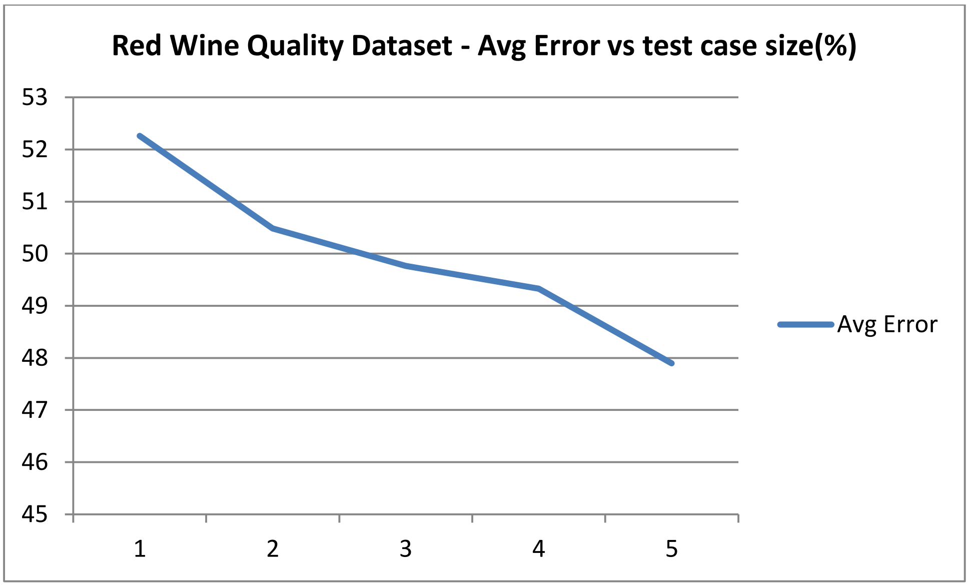

Red Wine Quality dataset - error over different training sets vs #Hidden nodes
70
60
50
40
30
20
10
0
100
200
300
400
500
600
700
800
900
1000
10%
20%
30%
40%
50%


Red Wine Quality Dataset - avg error over different training sets vs #Hidden nodes
52.5
52
51.5
51
50.5
50
49.5
49
48.5
48
1
2
3
4
5
6
7
8
9
10
Avg. Error

2. Average Error vs Percentage of Test Data:

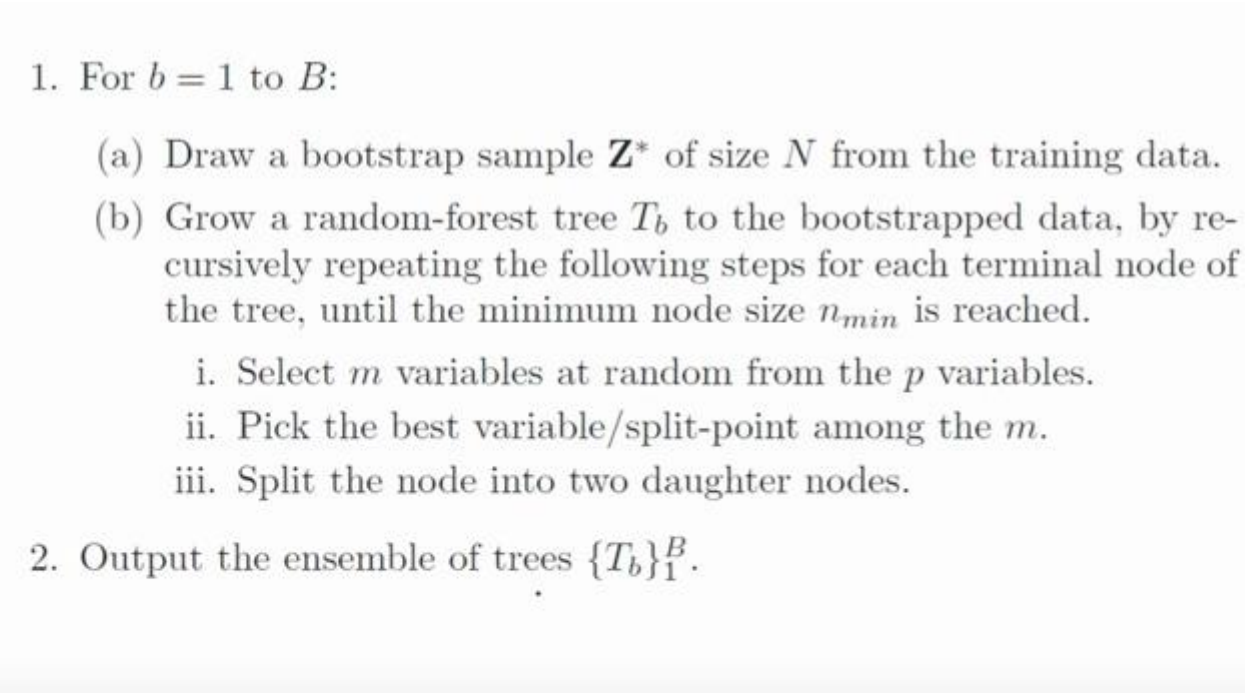
# Hidden Nodes	10% Test data	20% Test data	30% Test data	40% Test data	50% Test data
1	50.24	51.21	50.09	53.26	48.38
2	54.1	52	49.81	49.05	45.41
3	51.64	50.03	51.45	49.26	49.93
4	58.87	50.5	52.54	49.57	51.22
5	53.19	52.31	50.09	50.1	51.22
6	52.24	51.21	49.81	48.05	47.38
7	50.87	49.5	48.45	48.57	48.38
8	51.64	50.03	49.81	50.1	45.41
9	49.64	48.03	48.09	47.26	45.41
10	50.19	50.03	47.54	48.05	46.22
Avg. Error	52.262	50.485	49.768	49.327	47.896

This can be further illustrated by following graph:



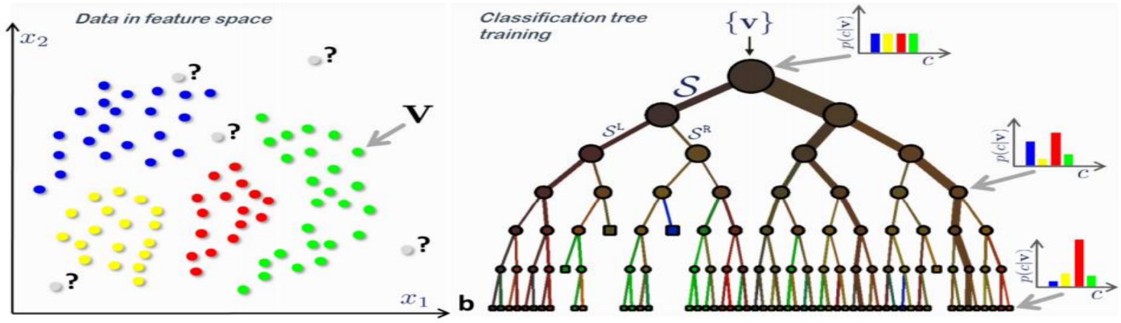
Random Forests

Given a dataset X of n points, we randomly draw m (<n) set of data points with replacement to form ‘k’ random datasets. These data points in each of these datasets(Bootstrap sample) are selected in a way that the correlation between any two datasets is minimum. This process is known as bagging. Then we utilize these datasets to build a forest of random trees (otherwise known as an ensemble).The algorithm for generating a forest is as follows:

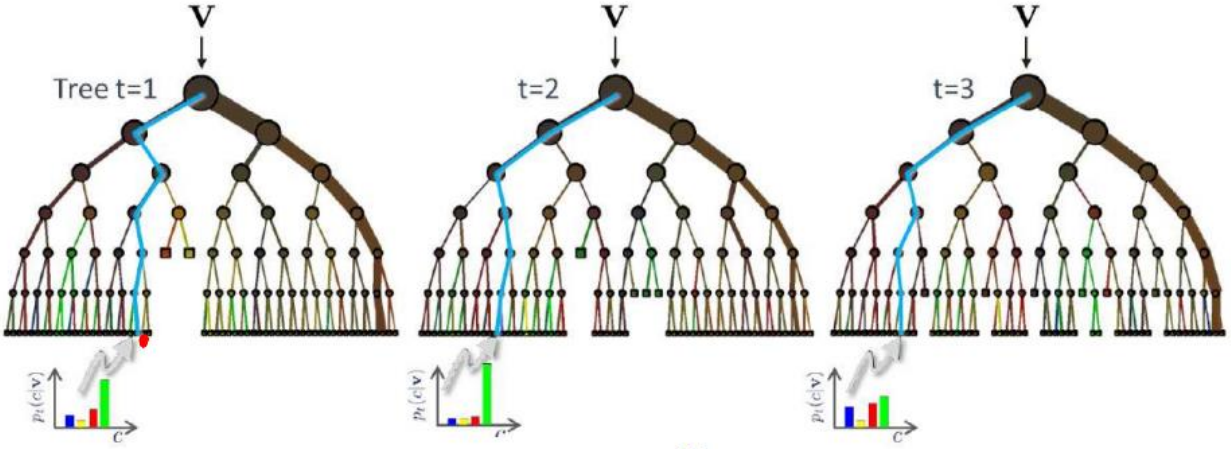


```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
        i. Select $m$ variables at random from the $p$ variables.
        ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

- ‘B’ indicates the number of trees in the forest. In the first step, for each tree in the forest draw a bootstrap sample ‘Z*’
- Then build a tree using this bootstrap sample. This is done by randomly selecting ‘m’ attributes (generally m=sqrt(p))out of p attributes and picking the best split. This split is decided based upon the quantity called information gain. The more the classes are separated at a node, the more the information gain.
- Once the features are decided, the node is split into two daughter nodes. This way the data points in the sample Z* are used to build a random tree. The probability distribution at each node in the random tree built gives the probability that a new test data point belongs to each class.



- Once a test point arrives, it is tested against all these tree classifiers (trees from the forest) and probability it belongs to a particular class is recorded at each of the trees. Then an average of these measures is taken and then the test point is classified accordingly.
- Suppose the forest consists of 3 trees, the new point is tested at each of the classifiers and then an average of the probability distribution is taken to determine the label of the new data point as shown in the figure below.



$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} p_t(c|\mathbf{v})$$

[Criminisi et al, 2011]

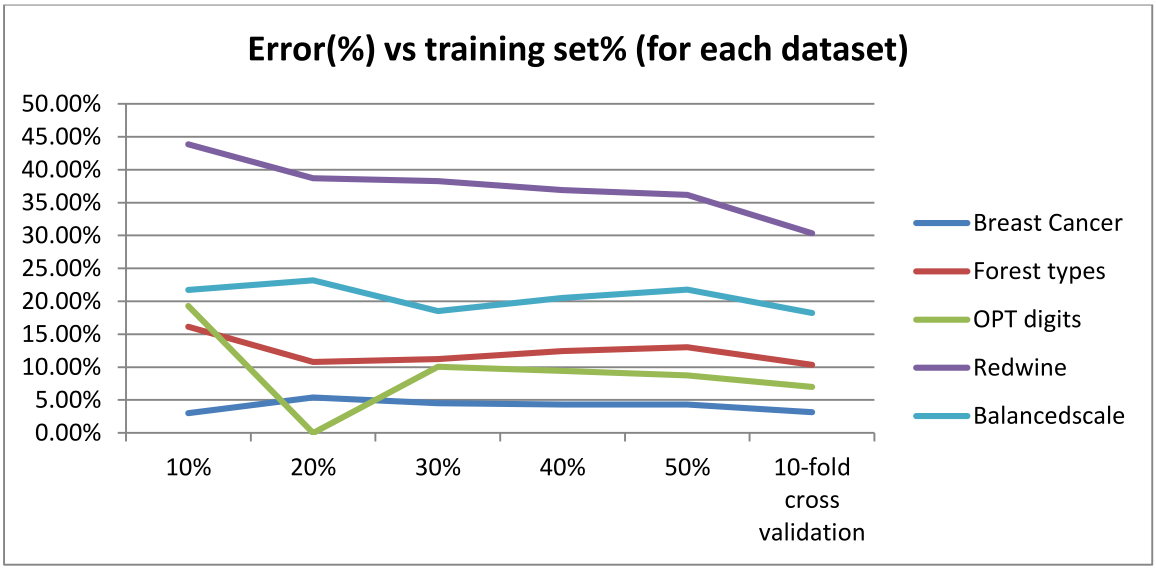
Performance on various datasets:

Random forests have 3 main tuning parameters :

- Number of trees
- Depth of the tree
- Numbers of features to be selected at each node when building a tree

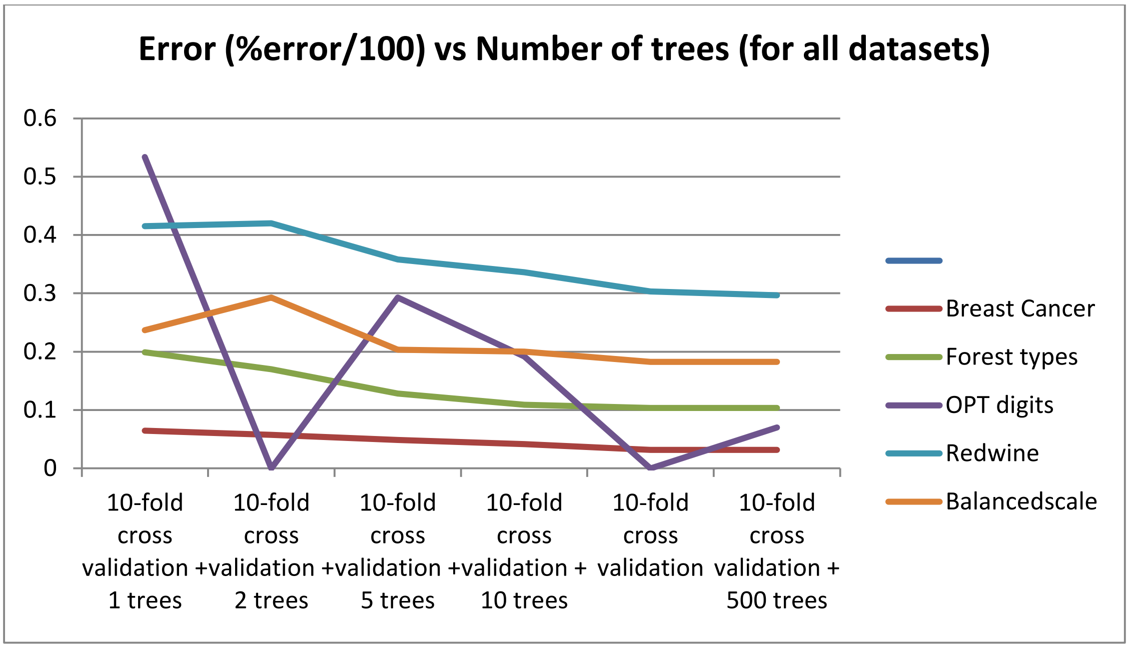
The following are the results obtained when the parameters are set to their default values, that is number of trees: 100, Depth of the tree is unlimited and the number of features are selected such that the error is minimized.

Misclassification error (Training set %)	*10%*	*20%*	*30%*	*40%*	*50%*	*10-fold cross validation*
Breast Cancer	3.021%	5.3667%	4.499%	4.2959%	4.298%	3.1474%
Forest types	16.1359%	10.7656%	11.2022%	12.4204%	13.0268%	10.325%
OPT digits	19.2764%	13.145%f	10.0661%	9.4306%	8.7189%	6.9929%
Redwine	43.85%	38.7021%	38.2484%	36.9135%	36.1702%	30.3315%
Balancedscale	21.7082%	23.2%	18.5355%	20.5333%	21.7949%	18.24%



As I vary the parameter number of trees mentioned above, we get the following results:

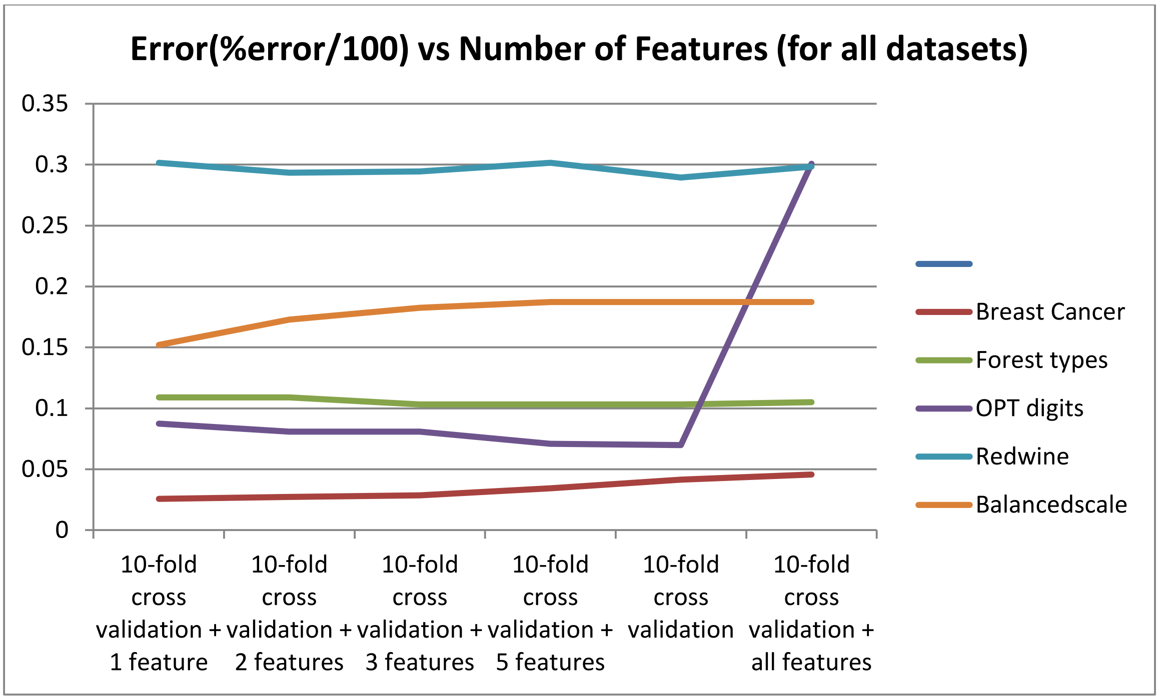
Misclassification error (Training set % +number of trees)	*10-fold cross validation + 1 trees*	*10-fold cross validation + 2 trees*	*10-fold cross validation + 5 trees*	*10-fold cross validation + 10 trees*	*10-fold cross validation +100 trees*	*10-fold cross validation + 500 trees*
Breast Cancer	6.4378%	5.7225%	4.8641%	4.1488%	3.1474 %	3.1474%
Forest types	19.8853%	17.0172%	12.8107%	10.8987%	10.325%	10.325%
OPT digits	53.3986%	46.6726%f	29.30661%	19.1993%	6. 9929%	6.9929%
Redwine	41.526%	42.0263%	35.8349%	33.646%	30.3315%	29.6435%
Balancedscale	23.68%	29.28%	20.32%	20%	18.24%	18.24%



Conclusion: As the number of trees increase, the error decreases and finally the error rate is stabilized.

Now I vary the parameter number of features:

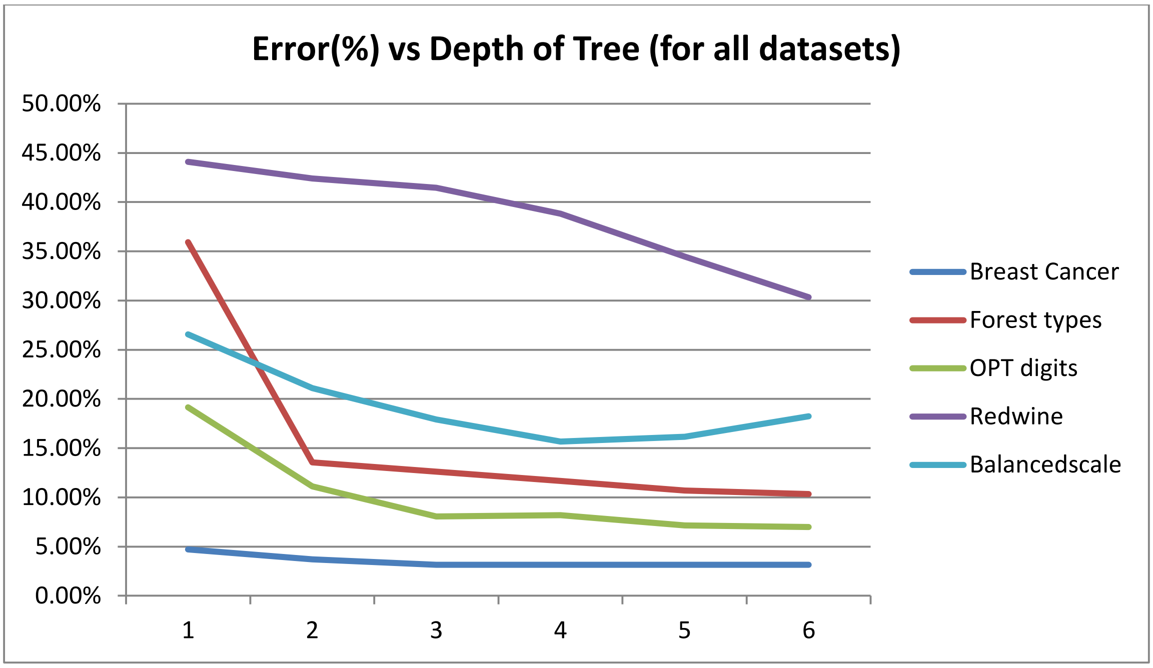
Misclassification error (Training set % +number of features)	*10-fold cross validation + 1 feature*	*10-fold cross validation + 2 features*	*10-fold cross validation + 3 features*	*10-fold cross validation + 5 features*	*10-fold cross validation + 7 features*	*10-fold cross validation + all features*
Breast Cancer	2.5751%	2.7182%	2.8612%	3.4335%	4.1488%	4.578%
Forest types	10.8987%	10.8987%	10.325%	10.325%	10.325%	10.5163%
OPT digits	8.7367%	8.0961%	8.0783%	7.0819%	6.9929%	30.0712%
Redwine	30.1438%	29.3388%	29.4459%	30.1438%	28.9556%	29.8311%
Balancedscale	15.2%	17.28%	18.24%	18.72%	18.72%	18.72%



Conclusion: As we increase the number of features, the error rate increases.

The performance of random forest as I vary the depth of the tree:

Misclassification error (Training set %+depth of tree)	*10-fold cross validation + 1*	*10-fold cross validation + 2*	*10-fold cross validation + 3*	*10-fold cross validation + 5*	*10-fold cross validation + 7*	*10-fold cross validation + unlimited depth*
Breast Cancer	4.721%	3.7196%	3.1474%	3.1474%	3.1474%	3.1474%
Forest types	35.9465%	13.5755%	12.6195%	11.6635%	10.7075%	10.325%
OPT digits	19.1637%	11.121%	8.0783%	8.2028%	7.153%	6.9929%
Redwine	44.0901%	42.4015%	41.4634%	38.8368%	34.459%	30.3315%
Balancedscale	26.56%	21.12%	17.92%	15.68%	16.16%	18.24%



Conclusion: As we increase the depth of the trees the error rate decreases. On the noisy dataset Balancedscale , there is divergence in the expected pattern.

Support Vector Machine using Platt’s Sequential Minimal Optimization (SMO)

Learning in a Support Vector Machine involves solving a constraint optimization problem. This optimization problem gives rise to the well-known SVM –Dual. The formulation of the dual is as follows:

$$\begin{aligned} \max \quad & \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i \alpha_j Q_{ij} \\ \text{subject to :} \quad & \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C, \ \forall i \end{aligned}$$

Where $Q_{ij}=y_iy_jK$ (x_i, xj) and K is the positive definite Kernel function chosen by the user. This is the famous Quadratic programming problem and there are several specialized ways one can approach it. The set of α_i should satisfy the optimality conditions of the SVM problem listed below:

$$\begin{aligned} \alpha_i = 0 \quad & \Longrightarrow \quad y_i f(x_i) \geq 1 \\ 0 < \alpha_i < C \quad & \Longrightarrow \quad y_i f(x_i) = 1 \\ \alpha_i = C \quad & \Longrightarrow \quad y_i f(x_i) \leq 1 \end{aligned}$$

The running time of a Quadratic programming problem is O (n^3) in the worst case, where n is number of variables + number of constraints, here 2l+1. When working with huge amounts of data, solving the above would be a herculean task even for modern computers. This lead to active research in solving this problem and this lead to the ideas stated below:

One Interesting idea is known as chunking wherein the user chooses to optimize only small number of variables at a time. Suppose the ‘l’ in the quadratic programming problem (will be referred to as QP problem from now on) is 1000, the user then chooses to solve only say 25 variables at a time. In the case of a QP with 1000 variables Q_{ij} is a matrix of dimensions 1000*1000 and the cost of computing is very high as compared to solving a QP with 25 variables which involves a Q_{ij} matrix of dimensions 25*25.

Therefore by reduced the number of optimization variables we can increases the speed of our classifier. This idea of chunking is used in SVM light. Now Platt’s Sequential Minimal Optimization (from here on, SMO) takes this idea of chunking to the extreme. SMO takes the smallest chunk of variables we can optimize on and this smallest chunk turns out to be 2. SMO assumes that at a particular point in the optimization that there are only two variables and the rest are treated as fixed. This algorithm needs to solve for only two variables at each iteration and very less time is needed in doing so. Effort is needed to determine which 2 variables are to be chosen out of the possible ‘l’ variables. Say there are ‘m’ variables out of the possible ‘l’ which satisfy the optimality conditions stated above, we need to have a heuristic to choose the 2 best variables. The optimization problem is as follows:

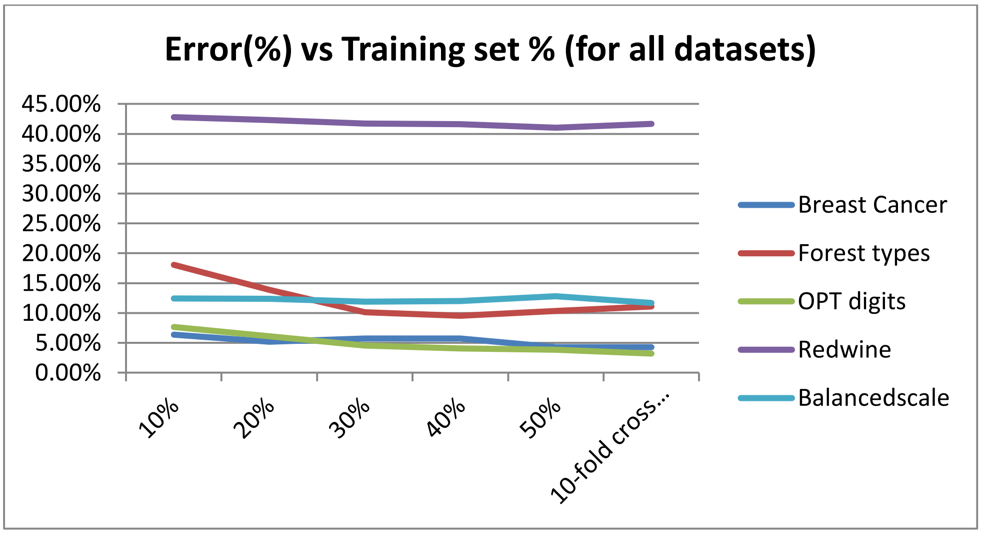
$$\min_{\alpha_a,\alpha_b} \quad \frac{1}{2}K_{aa}\alpha_a^2 + \frac{1}{2}K_{bb}\alpha_b^2 + \frac{1}{2}\alpha_a y_a \sum_{j\neq a} y_j\alpha_j K_{aj} + \frac{1}{2}\alpha_b y_b \sum_{j\neq b} y_j\alpha_j K_{bj} - \alpha_a - \alpha_b$$

$$y_a\alpha_a + y_b\alpha_b = -\sum_{i\neq a,b}\alpha_i y_i \qquad 0 \leq \alpha_a, \alpha_b \leq C$$

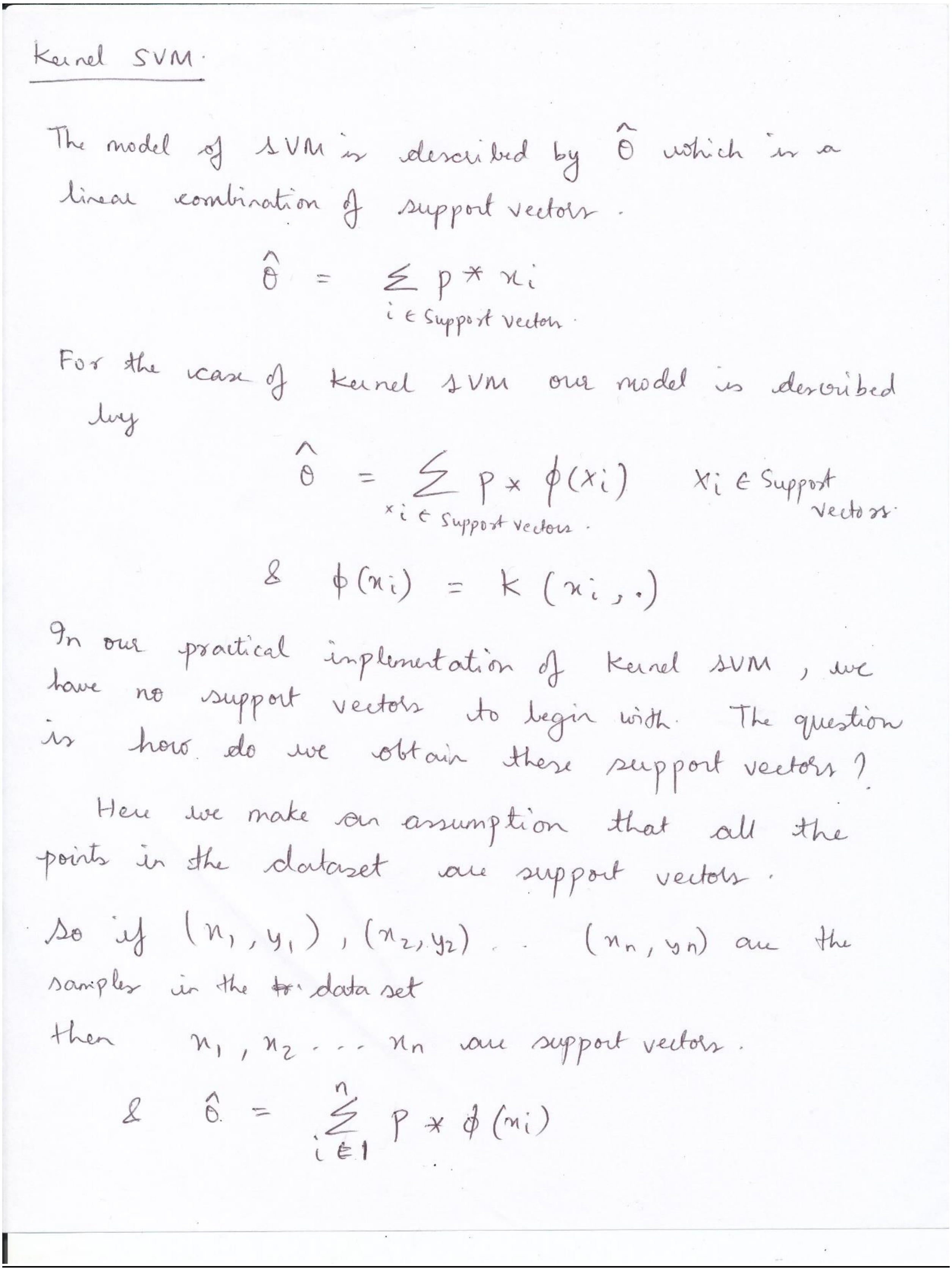
Where α_a, α_b are the two chosen variables at this iteration.

- Platt’s SMO uses two heuristics each to pick one variable at each iteration.
- The first variable (Lagrange multiplier) is obtained by first iterating over the entire training set and then iterating over the Lagrange multipliers which are 0 or C multiples times (multiple passes show interest in the unbounded support vectors which may need re-optimization)and then selecting one which violates the Karush-Kuhn-Tucker conditions.
- The second variable is chosen such that the step size is maximized during joint optimization.
- Repeat the selection process until convergence.

Misclassification error (Training set %)	*10%*	*20%*	*30%*	*40%*	*50%*	*10-fold cross validation*
Breast Cancer	6.3593 %	5.1878%	5.726%	5.7279%	4.298%	4.2918 %
Forest types	18.0467%	13.8756%	10.1093%	9.5541%	10.3448%	11.0899%
OPT digits	7.6512%	6.1165%f	4.5755%	4.0629%	3.8434%	3.2028%
Redwine	42.8075%	42.2987%	41.7337%	41.6058%	41.0513%	41.651%
Balancedscale	12.4555%	12.4%	11.8993%	12%	12.8205%	11.68%



Kernel SVM



Kernel SVM.

The model of SVM is described by $\hat{\theta}$ which is a linear combination of support vectors.

$$\hat{\theta} = \sum_{i \in \text{Support vector}} p * x_i$$

For the case of kernel SVM our model is described by

$$\hat{\theta} = \sum_{x_i \in \text{Support vectors}} p * \phi(x_i) \qquad x_i \in \text{Support vectors}$$

$$\& \quad \phi(x_i) = K(x_i, \cdot)$$

In our practical implementation of Kernel SVM, we have no support vectors to begin with. The question is how do we obtain these support vectors?

Here we make an assumption that all the points in the dataset are support vectors.

So if $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$ are the samples in the ~~tr~~ data set

then $x_1, x_2 \ldots x_n$ are support vectors.

$$\& \quad \hat{\theta} = \sum_{i \in 1}^{n} p * \phi(x_i)$$

We are considering a Gaussian Kernel first.

Therefore $\phi(x_i) = k(x_i, .) = e^{-\frac{\| \cdot - x_i \|^2}{2\sigma^2}}$

where x_i's are support vectors.

Now let us compute the similarity of a point with each support vector.

Let 'x' be the point.

Its similarity with x_1 is $K(x, x_1)$

$$= e^{-\frac{\|x - x_1\|^2}{2\sigma^2}} = f_1$$

similarly with $x_2 = e^{-\frac{\|x - x_2\|^2}{2\sigma^2}} = f_2$

$$\vdots \qquad \vdots$$

Similarily with $x_n = e^{-\frac{\|x - x_n\|^2}{2\sigma^2}} = f_n$.

~~He~~ Now we represent x with $\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$

Hence we have project x into a higher dimensional space. and creating new feature space.

If $x \simeq x_i$ (any support vector)

$$K(x, x_i) = e^0 = 1.$$

If x is far from x_i $\quad k(x, x_i) = e^{-\text{large number}} = 0$.

for x_1, the first data point.

$$f_1^{1} = k(x_1, x_1) = 1$$

$$f_2^{1} = k(x_1, x_2) \simeq 0.$$

$$\vdots$$

$$f_n^{1} = k(x_n, x_1) \simeq 0.$$

Taking bias into consideration

$$x_1 = \begin{bmatrix} f_1^{1} \\ f_2^{1} \\ \vdots \\ f_n^{1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \tilde{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\tilde{x}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Similarly

$$x_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad x_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad x_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Our initial feature space is replaced by $f_1, f_2 \cdots f_n$.

Our original objective function $\theta^T x$ is replaced by

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \cdots \theta_n f_n = \theta^T f_i$$

The formulation of SVM optimization is given by .

$$\text{Hinge loss} = \frac{1}{2}\theta^T\theta + C\sum_n \left[1 - y_n(\theta^T x_i + \theta_0)\right]_+$$

Now our feature space 'x_i' is transformed into feature space 'f_n'. $(i < n)$ [going into higher dimensions].

Each x_i now is a $(m \times 1)$ dimensional vector, if m is the number of training samples.

The feature space $(x_1, x_2 \ldots x_i)$ is transformed to $(f_1, f_2 \cdots f_m)$ $(i < m)$.

~~Mo~~ Kernel Hinge loss

$$= \frac{1}{2}\theta^T\theta + C\sum_n \left[1 - y_n(\theta^T f_n + \theta_0)\right]_+ \quad \hookrightarrow (1)$$

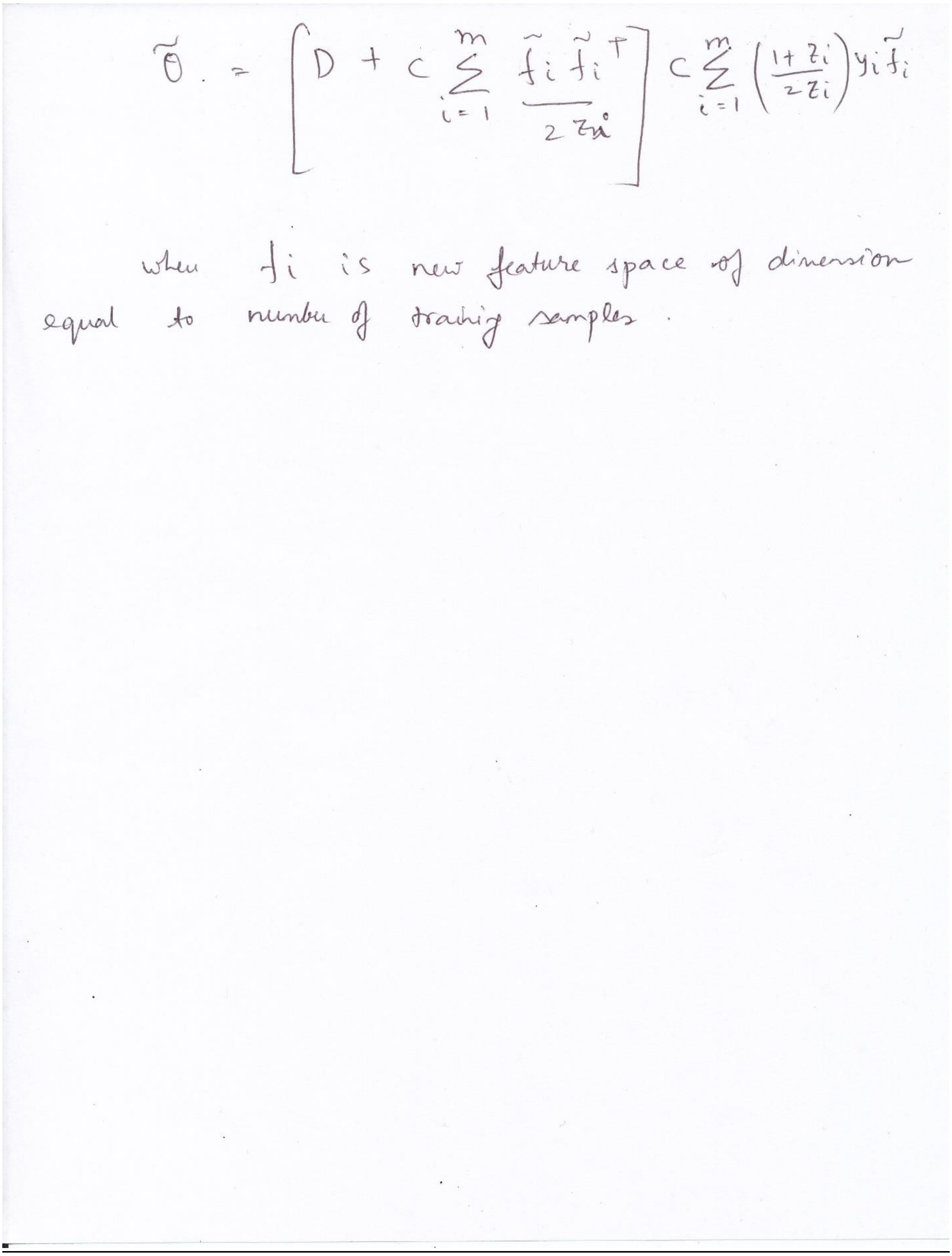
where $f_n = K(x, x_n)$.

The majorized version of (1) is

$$\min_\theta \frac{1}{2}\theta^T D\theta + C\sum \frac{\left[1 - y_n(\theta^T f_n + \theta_0) + z_n\right]^2}{4z_n}$$

$$\text{Where } D = \begin{bmatrix} 1 & & & \\ & 1 & & 0 \\ 0 & & 1 & \\ & & & 0 \end{bmatrix}$$

The solution is derived in the class and the solution



$$\tilde{\theta} \, . = \left[D + c \sum_{i=1}^{m} \frac{\tilde{f}_i \tilde{f}_i^{T}}{2 z_i} \right] c \sum_{i=1}^{m} \left(\frac{1+z_i}{2 z_i} \right) y_i \tilde{f}_i$$

when f_i is new feature space of dimension equal to number of training samples.

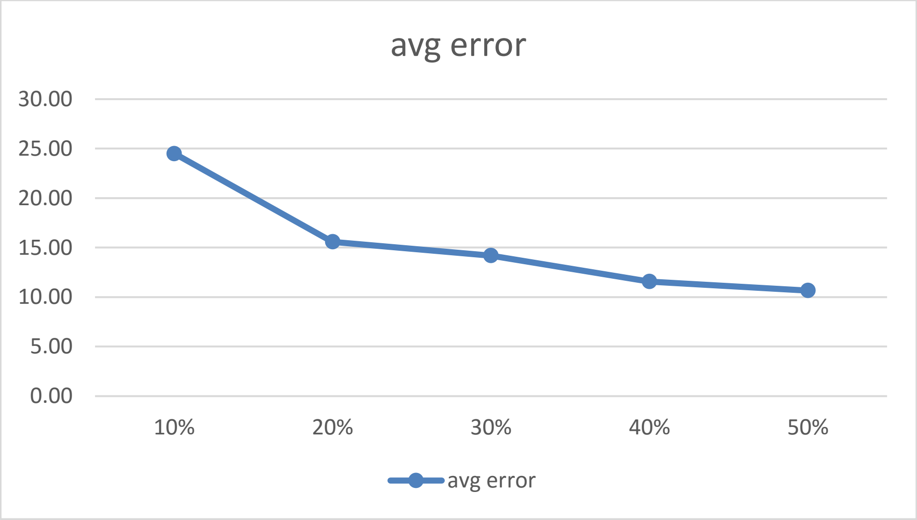
By using the theta above we create one classifier for each of the classes present. This is done by changing the ‘y’ attribute. If we are building a classifier for class 1, the labels corresponding to this class in the training set will have 1 and remaining are assigned a label of -1.Similarily we use a different ‘y’ for each of the classes and compute corresponding theta.

Once all the thetas are calculated we evaluate the new test point with each of the theta and the winner claims the point.

The following are the result we got by using the Gaussian kernel for the data samples:

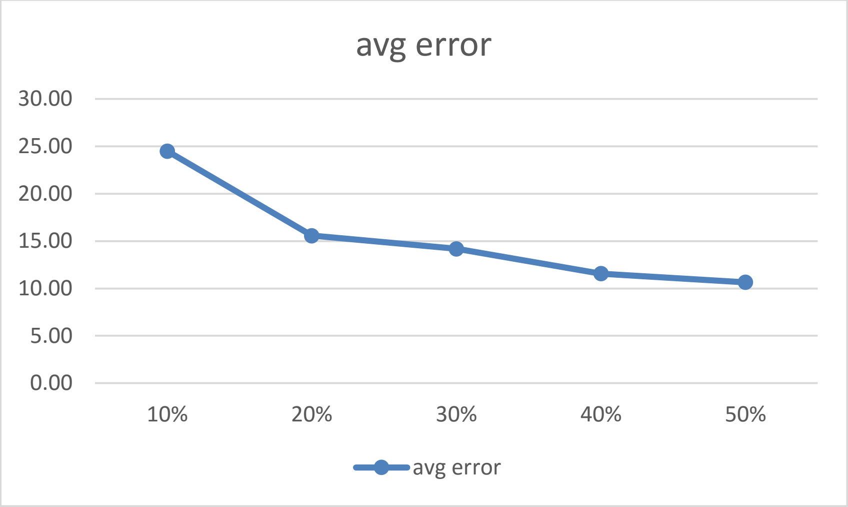
Training Set %		10%	20%	30%	40%	50%
Misclassification error for each Trial	1	17.17	15.74	11.45	8.35	8.60
	2	17.49	16.64	9.61	9.31	10.32
	3	10.02	12.88	10.43	9.31	9.74
	4	19.24	13.24	12.68	14.08	8.60
	5	18.28	15.03	10.84	9.79	9.17
	6	15.74	12.34	11.66	14.80	8.02
	7	18.60	11.45	14.52	10.98	10.89
	8	17.49	14.85	13.91	12.17	10.03
	9	12.40	12.70	11.66	10.50	10.60
	10	16.85	18.07	14.11	8.59	10.03

Table 1: *Breast Cancer Data Set*



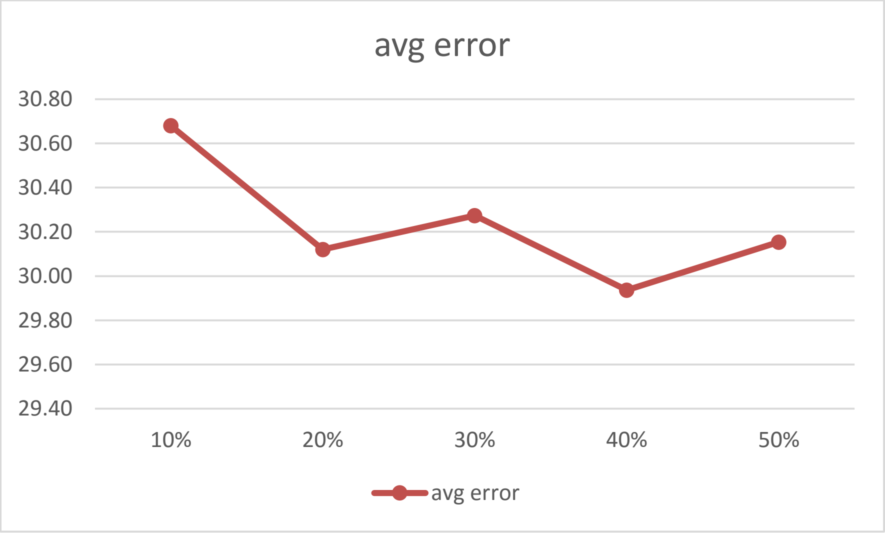
Training Set %		10%	20%	30%	40%	50%
Misclassification error for each Trial	1	23.31	13.80	11.44	11.20	9.94
	2	22.60	15.80	10.53	11.20	9.29
	3	17.08	17.60	15.10	11.47	11.54
	4	16.19	15.60	17.16	10.13	9.62
	5	30.61	15.00	14.87	12.80	11.22
	6	21.17	15.20	13.73	14.13	9.62
	7	36.12	14.60	13.04	12.53	14.10
	8	35.59	16.60	16.93	9.60	9.62
	9	24.02	17.00	14.65	12.00	11.54
	10	18.33	14.60	14.42	10.67	10.16

Table 2: ***Balanced scale Data Set***



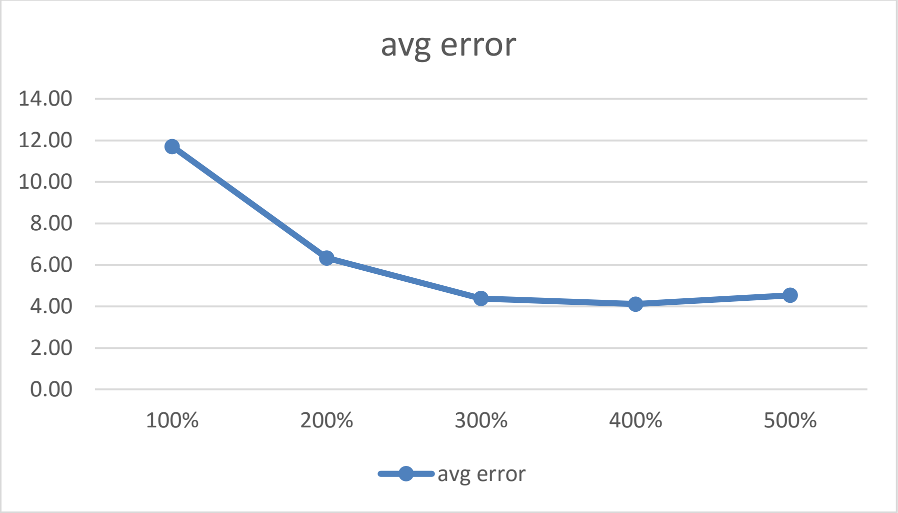
Training Set %		10%	20%	30%	40%	50%
Misclassification error for each Trial	1	31.21	31.10	30.33	29.62	30.27
	2	31.42	29.67	28.96	30.89	31.42
	3	30.15	29.90	29.24	31.21	31.03
	4	29.72	28.71	29.78	28.66	26.44
	5	30.15	31.10	29.24	30.89	31.42
	6	31.42	30.14	27.60	29.62	32.57
	7	30.15	29.43	30.05	28.66	29.50
	8	30.79	29.90	33.88	27.07	29.89
	9	31.00	30.86	31.42	30.89	30.65
	10	30.79	30.38	32.24	31.85	28.35

Table 3: ***Forest type mapping***



Training Set %		10%	20%	30%	40%	50%
Misclassification error for each Trial	1	7.41	2.50	8.57	3.33	5.33
	2	11.85	4.17	5.71	4.44	2.67
	3	11.11	7.50	4.76	3.33	4.00
	4	21.48	3.33	4.76	4.44	5.33
	5	7.41	5.83	4.76	5.56	1.33
	6	13.33	5.83	1.90	3.33	6.67
	7	11.11	10.83	2.86	5.56	4.00
	8	8.89	2.50	2.86	2.22	5.33
	9	15.56	14.17	1.90	4.44	5.33
	10	8.89	6.67	5.71	4.44	5.33

Table 4: ***Iris Data Set***



SVM Light

The SVMLight is an implementation of the support vector machine proposed in the Vapnik 1995. The SVMLight is a fast optimization algorithm whose working set is selected based on the steepest feasible descent.

To use the SVMLight using the packaged binaries we will need to preprocess the data so that the first labels are +1 or -1 depending on the class currently being looked at where +1 implies that the pattern belongs to the class and -1 otherwise.

Using a C function to read the CSV and convert to the SVMLight format we create key value pairs where the keys represent the index of the attribute allowing us to ignore the 0 values reducing the space complexity of the algorithm especially when dealing with image data. We specify a class label location to indicate the location of the label and whether there are any identifiers that need to be removed before running the SVM.

The data is partitioned into a training and test data set by using a randomizing function in python which creates the datasets of the desired size. The python script allows to specify the size of the training data as a fraction of the entire data set.

We iterate for each choice of the size and each of the class labels and 10 times for generalization and calculate the accuracy of the test data set with respect to the model. The iterations are created with a batch file

```
FOR /L %%C IN (10,10,50) DO (

     FOR  /L %%B IN  (0,1,9) DO (

          FOR /L %%A IN (1,1,10) DO (
               python sample.py %%C
               convert optdigits.csv.tra 64 %%B false >
               digitsTra.dat
               convert optdigits.csv.tes 64 %%B false >
               digitsTes.dat

               cd svm_light_windows64
               svm_learn ../digitsTra.dat modelDigits
               svm_classify ../digitsTes.dat modelDigits
               output >> dataDigits%%C

               cd ..
          )
     )

)
```

10%

Digits	Wisconsin	Balance	Forest
98.104	97.08	90.64	60.44
97.908	97.38	90.42	59.41
97.894	97.67	89.72	58.61
98.046	96.79	90.04	62.1
98.029	97.08	89.39	62.7
98.068	96.21	89.39	61.82
97.774	96.79	90	59.81
98.086	95.63	89.61	60.58
98.198	96.5	89.93	59.79
97.981	96.79	90.53	59.14

20%

Digits	Wisconsin	Balance	Forest
98.571	97.38	88.97	62.91
98.544	97.67	90.76	62.73
98.719	97.96	91.1	63.11
98.577	95.34	90.74	60.41
98.54	97.38	89.8	61.87
98.543	97.08	89.88	62.27
98.674	97.96	90.36	61.32
98.557	96.79	89.41	62.34
98.599	97.38	89.68	63.11
98.521	97.96	89.64	62.87

30%

Digits	Wisconsin	Balance	Forest
98.792	97.38	89.55	66.8
98.675	97.38	90.68	68.14
98.734	97.67	90.6	67.93
98.663	97.67	91.66	66.82
98.795	97.67	90.76	67.38
98.76	97.67	91.49	67.1
98.645	97.08	90.61	66.94
98.67	98.54	88.96	68.53
98.801	97.38	88.1	67.57
98.651	97.67	89.78	68.21

40%

Digits	Wisconsin	Balance	Forest
98.792	98.83	90.22	70.7
98.675	98.25	89.32	71.83
98.734	97.96	87.49	71.22
98.663	98.54	90.05	73.7
98.795	98.54	88.57	72.48
98.76	98.54	90.88	72.83
98.645	97.38	89.35	71.8
98.67	98.83	89.33	73.1
98.801	98.54	90.68	71.39
98.651	98.83	91.04	70.7

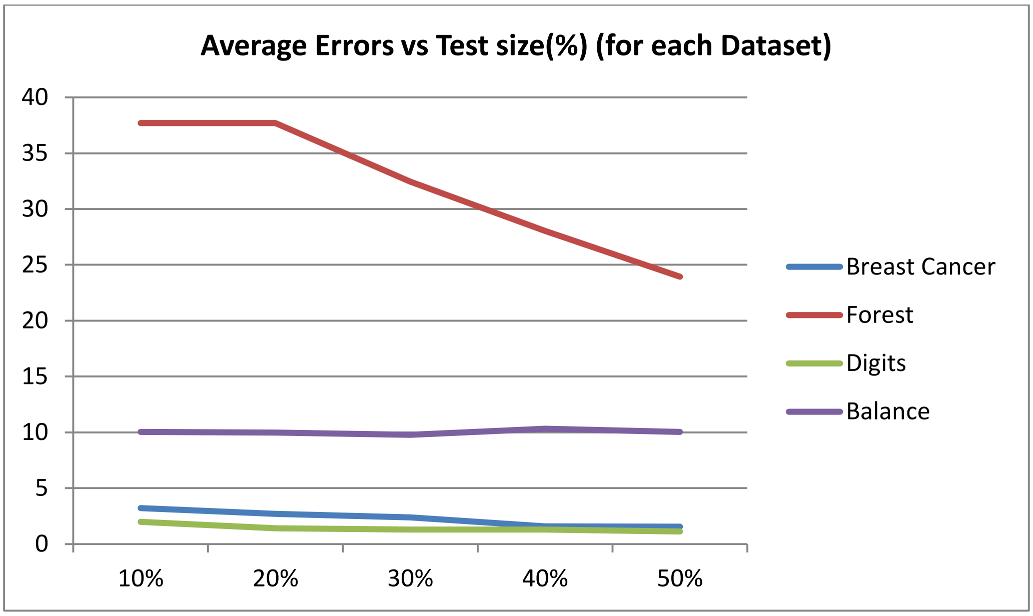
50%

Digits	Wisconsin	Balance	Forest
98.855	99.13	87.12	75.09
98.897	98.25	89.28	77.49
98.81	98.25	91.42	76.03
98.865	98.54	90.06	77.37
98.965	98.25	91.65	77.29
98.755	98.25	90.41	73.22
98.962	99.13	88.7	79.62
98.896	97.96	91.4	72.3
98.946	99.13	88.77	78.14
98.934	97.67	91.23	74.2

Summing up the information from these runs we get the following data:

	Test size 10%	Test size 20%	Test size 30%	Test size 40%	Test size 50%
Breast Cancer	3.21	2.71	2.39	1.58	1.55
Forest	37.706	37.71	32.458	28.03	23.93
Digits	1.98	1.4155	1.2876	1.2845	1.12
Balance	10.04	9.97	9.79	10.31	10.03

These data can be easily visualised in the figure below:



Comparison and Analysis

1. Breast Cancer:

	10%	20%	30%	40%	50%
SVM (SVMLight)	3.21	2.71	2.39	1.58	1.55
Random Forest	3.22	5.37	4.5	4.3	4.3
Deep Learning	5.296	3.69	4.058	3.8	3.53
Kernel SVM	16.85	15.03	14.52	12.17	10.6
SMO	6.36	5.19	5.73	5.73	4.3

For the breast cancer data set, apart from the industrial algorithm SVM light, Random Forests and Deep Learning algorithms performed better compared to kernel SVM.

2. Forest Data Set:

	10%	20%	30%	40%	50%
SVM (SVMLight)	37.706	37.71	32.458	28.03	23.93
Random Forest	16.14	10.77	11.2	12.42	13.03
Deep Learning	15.689	12.6	14.36	11.3	11.12
Kernel SVM	30.42	30.14	30.24	29.62	30.27
SMO	18.05	13.88	10.11	9.55	10.34

For the forest data set, Random forests and deep learning provided much better than SVM Light and kernel SVM.

3. Optical Digits Data Set:

	10%	20%	30%	40%	50%
SVM (SVMLight)	1.98	1.4155	1.2876	1.2845	1.12
Random Forest	19.28	13.145	10.07	9.43	8.72
Deep Learning	4.26	2.69	2.16	1.66	1.62
Kernel SVM	7.94	6.63	-	-	-
SMO	7.65	6.116	4.58	4.06	3.84

For the optical digit recognition data set, apart from SVM light, using convolutional neural networks provided near perfect classification. The implementation provided earlier in this report for kernel SVM was unable to train training sets larger than 20% of the data set within considerable time limits since the approach used by us requires the input space to be transformed to the kernel space which would imply working on inverses of square matrices of large dimensions which is a very costly operation.

4. Balance Scale Dataset:

	10%	20%	30%	40%	50%
SVM (SVMLight)	10.04	9.97	9.79	10.31	10.03
Random Forest	21.71	23.2	18.54	20.53	21.79
Deep Learning	12.83	11.84	11.858	11.94	11.36
Kernel SVM	24.59	15.2	14.65	12	10.16
SMO	12.46	12.4	11.9	12	12.82

Deep Learning and SVM Light produced better classifiers for the balanced scale data set while random forests produced considerably higher misclassification errors.

5. Red Wine Dataset:

	10%	20%	30%	40%	50%
SVM (SVMLight)	-	-	-	-	-
Random Forest	43.85	38.7	38.25	36.99	36.17
Deep Learning	52.26	50.48	49.76	49.32	47.89
Kernel SVM	-	-	-	-	-
SMO	42.81	42.3	41.73	41.61	41.05

This data set had highly noisy data and hence didn’t significantly perform better than even a baseline classifier for any of the above algorithms.

References

Y.Bengio, X. Glorot, Understanding the difficulty of training deep feedforward neuralnetworks, AISTATS 2010.

H.Lee, R. Grosse, R. Ranganath, and A.Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.”, ICML 2009

Deeplearning.net

kaggle.com/c/digit-recognizer/forums/t/10552/convolutional-neural-networks-using-theano

http://www.cs.ubc.ca/~nando/340-2012/lectures/l25.pdf

Random Forests by Leo Breiman (2001)

Everything old is new again: A fresh look at historical approaches in machine learning byRyan Michael Rifkin

https://www.youtube.com/watch?v=LHzqW0EolzE

Andrew Ng lecture on Kernel SVM: https://www.youtube.com/watch?v=LHzqW0EolzE

http://svmlight.joachims.org/

http://image.diku.dk/imagecanon/material/cortes_vapnik95.pdf

Support Vector-Networks by Corinna Cortes and Vladmir Vapnik

Code File Attachments

MulticlassKernelSVM.m : MATLAB code file for the kernel SVM implementation.

neural_net.py : contains the python/theano code used to build the neural nets

conv_neural_net.py : containes the python/theano code used to build convolutional neural networks.

sample.py : Python code snippet for random sampling and partitioning of data.

convert.c : Code to convert the data to format acceptable to SVM Light.