Analysis of Supervised Learning Algorithms

Author: Swati Prasad(sprasad33@gatech.edu)

Abstract: Analysis of five supervised algorithms Decision Tree, Boosting, Neural Network, K nearest neighbors & Support vector machine on UCI datasets (Pima Indian diabetes datasets and Letter recognition datasets) using weka tool is discussed in this paper. As this is classification problem, Accuracy and ROC curve is used as an analysis parameter.

DataSets Information:

Datasets 1: Pima Indians Diabetes Data Set

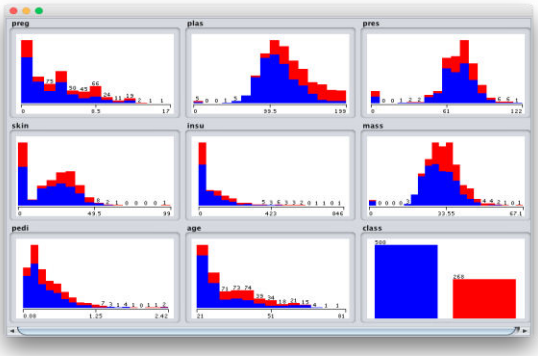
Number of Instances = 768

a) Characteristics of data after reviewing attributes in weka
   1) age : Sample population is young and 708/768 instance's data has age < 50
   2) 0 value indicates error or missing data. Plas, pres, mass has errors.
   3) There are 500 (65%) positive instances and 258 (35%) negative instances
   4) There is enough data samples. We have 8 attributes. Min number of samples = 2^8 (256) and we have 758 isntances.

b) After reviewing histogram of attributes in weka;
   1) Few attributes are normally distributed (plas, pres, mass and skin)
   2) Few attributes are exponentially distributed (preg, insu, pedi, age)
   3) Age should have been normally distributed for ideal result condition
   4) There is few big outlier values in insu, which make std deviation high

c) After reviewing Scatter plot of dataset in weka
   1) There is no direct relation between any attribute and diabetes.
   2) Higher preg usually causes diabetes

Data analysis of Dataset-1 in weka



Data Analysis of DataSet-2 in weka

DataSet 2: Letter Image Recognition Data

Number of Instances: 20000 Number of Attributes: 17 (Letter category and 16 numeric features)

a) Characteristics of data after reviewing attributes in weka
   1) There are no missing values
   2) Class distribution is fair. Each class has got instances > 730 and < 805
   3) No attributes with negative value.

b) After reviewing histogram of attributes in weka;

1) Distribution of Sample population is good. Many attributes are normally distributed. e.g x-box, width, x-bar, y-bar , x2bar, y2bar, x2ybar, xy2bar, xegvy
2) Few attributes are approximately exponentially distributed (totpix, x-ege)
3) Sample data has even distribution of 26 letter

Experiment Steps:
We will be using Inductive Learning
Steps to do analysis of datasets:
1) Parameter tuning of each algorithms based on accuracy, ROC curve.
2) Learning curve for tuned algorithm with varying training size.
3) Learning from 10-fold cross validation training datasets and applying to testing datasets. 10 fold cross validation training set to remove variance
4) Data is distributed 70/10/20. 70% data as training set 10% as validation set and 20% as testing set.

Explanation of different types of analysis tool and parameters:
i) Learning Curve: This is a classification problem so we will be using % accuracy for correct classification Vs training set instead of root mean square error. Learning curves are useful for deciding the size of the training dataset, since you can find the size of the dataset at which the classifier stops learning (and maybe degrades). So the best classifier in this case might be the one reaching the highest accuracy with the smallest dataset size.

ii) ROC curve: Under the ROC curve (AUC) is a performance metric for binary classification problem. It is plot of true positive Vs false Positive. AUC represent the model's ability to discriminate between positive and negative classes. Area 1.0 represents a model that made all predictions perfectly and area 0.5 as good as random. For dataset-2, it is created like class1 Vs group of rest of other class. Then ROC curve for class-2 Vs group of other classes.

iii) 10-fold cross validation: Model is created on 10-fold cross validation set to remove variance. In 10-fold cross-validation, divide the training set into 10 subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining 10 – 1=9 subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data, which are correctly classified. It can prevent over fitting problem.

Performance Analysis On Datasets:

Baseline Algorithm

It is critically important to develop a baseline of performance when working on a machine Learning problem. A baseline provides a point of reference from which to compare other machine learning algorithm. Supervised Machine learning algorithm should perform better than baseline algorithm. From ROC Area, it is clear that zero-R algorithm is working on random selection of choice. So, accuracy and dataset-1 = 500/768 and for dataset-2 = 1/26 { as data is evenly distributed }

Baseline Performance of datasets using zero-R algorithm in weka:

Dataset-1	Dataset-2
Training Instance = 537 Validation Instance = 80	Training Instance = 14000 Validation Instance = 2000

Testing Instance = 251 Accuracy: 65.1% Avg ROC Area: .497 Root Mean Square Error: .476	Testing Instance = 4000 Accuracy : 4.0714% ROC Area: .498 Root Mean Square Error : .1923

Parameter Tuning of Supervised Algorithms:

i) Decision Tree and boosting:

Goal: Achieve perfect classification with minimum number of nodes

Tools: Weka (J48) for tree and (AdaboostM1) for Boosting are used for this analysis

Decision tree analysis based on 2 parameters tuning:

a) Confidence factor: confidence factor will be used for pruning.

b) MinNumobj: minimum number of instances per leaf

Boosting's parameter is number of iterations

Table 1-a Experiment Table on Dataset-1 for Decision Tree

Training Attempt	No. of Leaves	Size of Tree	Pruning	Confidence Factor	MinNumobj	ROC Area	Time to build	Time to query	Correctly classified training accuracy	Correctly Classified testing accuracy
1	29	57	No	.25	1	.768	.02	.01	72.81%	76.62%
2	11	21	No	.25	3	.801	.01	0	73.18%	77.92%
3/4/5	11	21	No	.25	4,5,6	.794	.01	0	72.25%	75.32%
6	11	21	Yes	.25	3	.801	.01	0	72.99%	77.92%
7	8	15	Yes	.15	3	.808	.01	0	72.25%	78.35%
8	9	17	Yes	.15	2	.784	.01	0	72.99%	77.92%
9	8	15	Yes	.1	3	.808	.01	0	71.88%	78.35%
10	5	9	Yes	.1	13	.822	.01	0	71.88%	79.22%
11	6	11	Yes	.1	23	.847	0	0	72.62%	80.08%
10	6	11	Yes	.1	40	.847	0	0	71.88%	80.08%

Table -1b Experiment Table on Dataset-1 for Boosting

Training Attempt	Set Iteartions	Performed iteartion	Confidence Factor	MinNumobj	ROC Area	Correctly classified training accuracy	Correctly Classified testing accuracy	Time To Build (sec)	Time to Query (Sec)
1	10	9	.25	2	.823	69.08%	77.92%	.02	0
2	20	9	.25	2	.823	70.76%	77.92%	.02	0
3	30	9	.25	2	.823	71.32%	77.92%	.03	0
4	10	10	.15	3	.811	72.25%	75.75%	.07	0
5	30	17	.15	3	.809	71.5%	76.62%	.09	0
6	30	5	.15	2	.812	73.74%	76.19%	.03	0
7	30	5	.15	2	.812	74.11%	76.19%	.03	.01
8	10	5	.1	23	.855	72.81%	79.22%	.03	.01
9	10	6	.1	40	.856	73.37%	80.51%	.03	.01

From Table-1a

With minNumObj = 1 caused over fitting.

From table-1B, with increasing iteration ROC area is not increasing and number of iteration is also constant. though we set iteration to 10, 30 , it finishes in 5 iteartion, It means there is improvement in accuracy.

Observation From dataset-1 for decision tree/boosting:

From Table-1a:-

No pruning and 1 instance/leaf caused over fitting. With increasing minNumobj, size of tree decreases and also decreases variance. Decreasing confidence factor from .25 to .15 decreases size of tree means more pruning. Pruned tree is giving better result for this dataset. It means, result is better for general features. Accuracy decreases with including specific features. Time to build pruned tree require less time, which make sense with decreasing tree size.

From table-1B :-

With Boosting, training/testing accuracy doesn't change much. With increasing number of iteration, it is almost same. Boosting only works with weak learners. Weak learner has error rate better than random (< 1/2 or 1/2-e) no matter what the distribution is. This confirms the dataset has random data distribution and noise in data and can't increase testing accuracy more than 80.51%. We may need to collect more data sample to improve the result further. ROC area is around .847 which mean algorithm is efficient. Time to create model in .03 sec with boosting, it shows the small & simple dataset. Query time in decision tree is smaller than building model time. Boosting works better with small tree or more instances per leaf. It means it is working better with generalize version of dataset.

Experiment Table on Dataset-2 for Decision Tree

Training Attempt	No. of Leaves	Size of Tree	Pruning	Confidence Factor	Min Num obj	Avg ROC Area	Time to build	Time to query	Correctly classified training accuracy	Correctly Classified testing accuracy
1	1671	3341	No	.25	1	.93	.83	.08	86.65%	86.6%
2	900	1799	No	.25	3	.93	.78	.07	85.77%	85.3%
3	486	971	Yes	.25	7	.95	.62	.05	82.92%	82.91%
4	568	1135	Yes	.2	2	.96	.76	.06	84.16%	86.11%
5	926	1851	Yes	.15	2	.96	.78	.06	86.25%	86.1%
6	728	1455	Yes	.1	3	.96	.65	.05	85.62%	77.92%
6	728	1455	Yes	.15	20	.96	.45	.05	76.71%	76.31%
6	728	1455	Yes	.1	3	.96	.65	.05	85.62%	77.92%

Experiment Table on Dataset-2 for Boosting

Training Attempt	Iteartions	Performed Iteartion	Confidence Factor	MinNumobj	Avg ROC Area	Correctly classified training accuracy	Correctly Classified testing accuracy	Time to build Model (sec)	Time to Query (sec)
1	10	10	.25	1	.998	93.98%	94.81%	9.37	.35
2	20	20	.25	1	.999	95.16%	95.91%	18.88	.6
2	40	40	.25	1	.999	95.16%	96.31%	37.34	1.1
3	10	10	.25	2	.998	94.15%	94.7%	8.17	.39
4	20	20	.25	2	.999	95.54%	95.8%	16.95	.65
7	10	10	.1	2	.998	94.24%	94.9%	9.09	.33

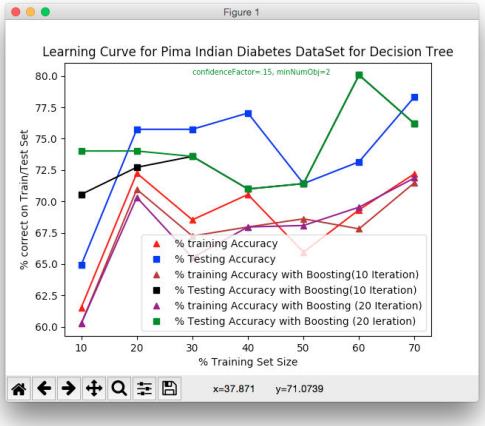
8	20	20	.1	2	.999	95.47%	95.96%	15.85	.57
8	40	20	.1	2	.999	96.3%	96.61%	33.73	1.13
8	60	60	.1	2	.999	96.3%	96.66%	51.6	1.84

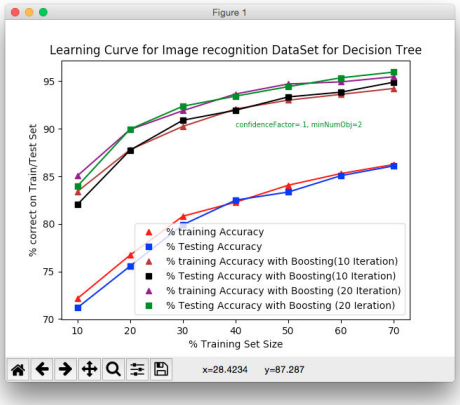
Observation from dataset-2 for decision tree/boosting:

Boosting increases training/testing accuracy by average 7% for decision tree. This means, there are lot of weak learners, which improved with iteration of boosting. Time taken to build model with boosting is significant due to more number of dataset than dataset-1. Accuracy and time to build model both increases with increasing number of iterations. With doubling number of iteration, also increase time to build the model by 2 times. Time to query is much smaller than time to build. As, we are searching in decision tree it is logarithmic running time for query. Boosting doesn't improve after approx. 96.66% for training and testing accuracy. After 40 iterations, accuracy is hardly increasing, but building model time is significant. This is very efficient algorithm as ROC area is also approx. 1.

Performance Analysis based on Training size for decision tree :

Dataset-1: For small training size, training and testing accuracy is low, due to it is dealing with under fitting. It means high bias and low variance. >50% of dataset, boosting increase accuracy rate for training and testing dataset. Boosting is not able to cross testing accuracy of 77.92% , which means there is noise in data. As with increasing number of iteration testing accuracy lines overlaps each other but doesn't improve accuracy further. 60% -70% of training data, testing accuracy decreases while training accuracy increases, it shows over fitting property.





Dataset-2: Boosting's effect is much evident in this data set. Without boosting, training/testing accuracy is in range 70-80% while with boosting 85-96%. Even further improvement in accuracy with increasing number of iteration. Low training and testing accuracy with small dataset shows under fitting.

Decision tree searches through the set of possibly hypotheses and preferentially selects those hypotheses that lead to a smaller decision tree. This type of bias is called a preference (or search) bias. Dataset-1 has more preference bias than dataset-2. Confusion Matrix gives accuracy as well as number of instances classified wrong. Roc curve is based on confusion matrix.

Confusion Matrix of Binary Dataset (Datset-1) for decision tree with boosting

```
a   b   <-- classified as
 132  20 |   a = 0
  25  54 |   b = 1
```

ii) Neural Network

Tool: Weka (function (Multilayer perceptron))

Network Design Parameters:

Gradient descent optimization algorithm to minimize the error function to reach a global minima. Global minima is achieved by tuning following parameters:

a) Learning Rate (0 <= a <= 1.0): Learning rate is a procedure which assesses the relative contribution of each weight to the total error. The selection of Learning Rate is critical importance in finding the true global minimum of the error distance. If it is too small- LR will make really slow progress. And if it is too large it will proceed much faster.

c) Momentum : To help avoiding settling on local minima, momentum rate allows network the network to skip through the local minima. Momentum rate can be helpful in speeding the convergence and avoiding local minimum.

d) No. of input node: Number of input attributes. These are independent variables

e) No. of output Nodes: In this classification example for binary classification, it is 0/1

f) No. of middle or hidden layers: Hidden layer allow number of potentially different combinations of input might result in low outputs. Nodes in the middle layer is connected to all previous and following nodes

Experiment Table of Dataset-1 for NN

Training Attempt	No. of hidden neurons/ Hidden Layers	No. of epoch	Error Per Epoch	Learning Rate	Mom e-ntum	ROC Area	Correctly Classified Training Accuracy	Correctly Classified Testing Accuracy
1	5/1	15	.163	.2	.7	.793	75.79%	78.78%
2	5/1	15	.163	.3	.7	.801	74.3%	79.22%
3	5/1	15	.162	.4	.7	.796	74.48%	77.48%
4	5/1	15	.16	.3	.1	.805	75.23%	78.78%
5	5/1	15	.16	.3	.3	.804	75.23%	79.22%
6	5/1	15	.168	.3	.5	.802	74.86%	77.92%
7	5/1	15	.164	.3	.8	.797	75.6%	78.78%
8	5/1	20	.161	.3	.3	.804	75.04%	78.35%
9	5/1	40	.156	.3	.3	.808	76.35%	79.22%
10	5/1	60	.153	.3	.3	.814	76.16%	77.92%
11	5,2/2	40	.164	.3	.3	.793	75.79%	79.22%
12	5,2,2/3	40	.23	.3	.3	.521	64.8%	65.8%
13	0	40	.162	.3	.3	.808	75.79%	79.65%

Observation From Dataset-1 for Neural Network:

Increasing the learning rate didn't affect the training accuracy much but started decreasing testing accuracy. It may be finding local minima for network solution. High momentum may skip the global minima and stuck in local minima. This is proved by high testing accuracy and low training accuracy with momentum .8.

Started Multilayer perceptron with default setting. It sets the initial hidden layer as

Number of hidden layer = (number of Attributes + Number of output)/2 = (8 + 2)/2 = 5 neurons/hidden layer

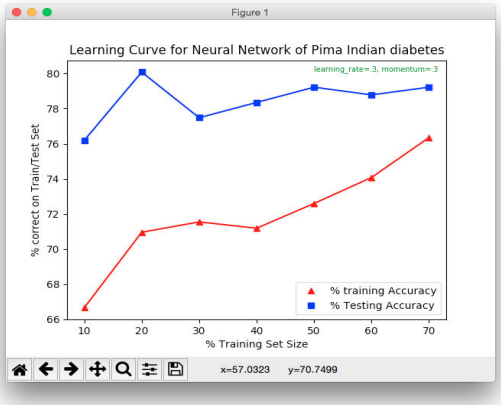
Over complicated network structure from training attempt 10-12 with 3 hidden layers decreases training/testing result significantly due to low bias and high variance. Training attempt 13 in this example, even no hidden layer. Simple input to output connecting gives me good training and testing accuracy due to simple dataset. Further tested with increasing number of iterations 40, 80, 120, 160 and 200 testing accuracy is around 78 -79%. Accuracy doesn't get improved after 40 iterations.

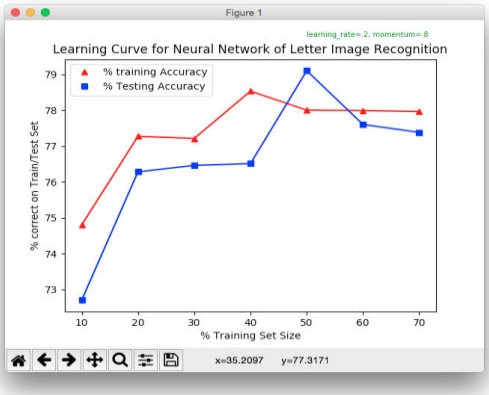
Experiment Table of Dataset-2 for NN

Training Attempt	No. of hidden Neurons/ Hidden Layers	No. of Epoch	Error Per Epoch	Learning Rate	Momentum	ROC Area	Correctly Classified Training Accuracy	Correctly Classified Testing Accuracy
1	17/1	40	.0098	.2	.7	.94	81.3%	81.25%
3	17/1	40	.01	.4	.7	.94	80.47%	78.76%
4	17/1	40	.01	.2	.1	.9	77.98%	78.78%
5	17/1	40	.01	.2	.4	.89	77.45%	78.8%
6	17/1	40	.01	.2	.6	.89	78.5%	79.7%
7	17/1	40	.009	.2	.8	.91	80.94%	80.21%
11	17,8/2	40	.1	.2	.7	.87	73.97%	71.95%
13	0	40	.019	.2	.7	.85	68.85%	68.53%

Observations From Dataset-2 for Neural Network:

For dataset-2, learning rate, momentum tradeoff is .2/.8, at which training and testing accuracy is highest. This means low learning rate and high momentum gives the global minima for this dataset. A large momentum means convergence will happen fast. Further tested with increasing number of iterations 40, 80, 120, 160, 200 & 400 testing accuracy is around 80-81%. Accuracy doesn't get improved after 40 iterations.





Dataset-1: With very small training set size, training and testing accuracy is low. Training accuracy has increased further with training size compared to testing accuracy. After 50% of training data, training accuracy has increased but testing accuracy has decreased and that is the over fitting area. There is a significant gap in training and testing accuracy for small training size. Number of training data plays important role in accuracy.

Dataset-2: Training and testing accuracy is almost parallel with varying training size. But again there is improvement in accuracy with increasing number of training datsets. There is low training/testing accuracy with small dataset, which shows underfit area.

iii) KNN (Instance Based Learning):

KNN doesn't have training phase. All instances are stored. Training may involves indexes to find neighbors quickly. During testing, KNN algo has to find k nearest neighbors quickly. This is time consuming, if we do exhaustive searching addition storing all instances in memory, necessary to compute k nearest neighbors takes lot of space too. That's why KNN is lazy learner.

An obvious issue with k nearest neighbor is how to choose a suitable value for the number of nearest neighbors used.

Experiment Table of Dataset -1 for KNN

Training Attempt	K (No. of nearest neighbor)	Search Algorithm Distance Function	Distance Weighing	ROC Area	Time to build	Time to query	Correctly Classified Training Accuracy	Correctly classified Testing Accuracy
1	1	Euclidian	False	.649	0	.02	70.57%	70.56%
2	3	Euclidian	False	.733	0	.02	70.39%	77.05%
3	5	Euclidian	False	.753	0	.02	71.13%	75.75%
4	7	Euclidian	False	.752	0	.02	69.64%	77.05%
5	10	Euclidian	False	.77	0	.02	72.25%	74.45%
5	12	Euclidian	False	.77	0	.02	73.37%	75.75%
6	100	Euclidian	False	.794	0	.04	69.83%	73.59%
7	300	Euclidian	False	.763	0	.07	64.8%	65.8%
8	3	Manhattan	False	.707	0	.02	69.83%	74.89%
9	7	Manhattan	False	.74	0	.02	70.76%	75.32%
10	10	Manhattan	False	.757	0	.02	71.69%	74.89%
11	100	Manhattan	False	.798	0	.04	69.08%	72.72%
12	7	Euclidian	Weight by 1-distance	.768	0	.02	69.64%	77.05%
13	10	Euclidian	Weight by 1-distance	.781	0	.02	71.32%	77.05%
14	12	Euclidian	Weight by 1-distance	.779	0	.02	71.88%	77.48%
15	10	Euclidian	Weight by 1/distance	.780	0	.02	71.69%	76.62%

Observation from dataset-1 for KNN:

a) When k is too small, the method is susceptible to noise in the data. Without cross fold training data accuracy goes to 100%, but testing accuracy is 70%. Overfit area is evident with very small k.
b) With increasing number of nearest neighbors up to 12, it shows improvement. Increasing nearest neighbors to big number 100, 300 performance start decreasing. K = 7, 10, 12 showed optimize behavior.
c) Compared Euclidian Distance and Manhattan distance algorithm. For this dataset, Euclidian Distance algorithm gives better result

d) Selecting distance weighing algorithm especially weight by 1-distance gives better result than no weighing algorithm. It means nearest neighbors get more weightage than far neighbors. For same k, Weight by 1/distance gives poor result than Weight by 1-distance. This shows preference bias.
e) Time to build model is 0 as we are only storing points, while time to query depends upon number of neighbors considered for calculating distance.

Experiment Table of DataSet-2 for KNN

Training Attempt	K (No. of nearest neighbor)	Search Algorithm Distance Function	Distance Weighing	ROC Area	Time to build	Time to query	Correctly Classified Training Accuracy	Correctly classified Testing Accuracy
1	1	Euclidian	False	.98	0	9.15	94.91%	95.51%
2	3	Euclidian	False	.98	0	10.01	94.27%	94.7%
3	5	Euclidian	False	.998	0	10.72	93.95%	94.26%
4	10	Euclidian	False	.998	0	11.83	93.05%	93.83%
5	100	Euclidian	False	.995	0.17	18.65	79.58%	80.8%
8	2	Manhattan	False	.997	0	12.22	92.71%	93.88%
9	3	Manhattan	False		0	13.37	94%	94.48%
10	5	Euclidian	Weight by 1-distance	.999	0	11.13	94.51%	95.03%
11	5	Euclidian	Weight by 1/distance	.999	0	11.16	94.78%	95.31%

Observation From dataset-2 for KNN:

Dataset-2 has less noise and KNN works better than dataset-1. With very low k value, it tries over fit, but both training and testing accuracy is high. If k is too large, the decision is smeared out, covering too great an area of instance space e.g K =100 gives lowest accuracy. Accuracy further improved by giving much weightage to nearest neighbors by weight distance algorithm.

Conclusion: Increasing k will decrease variance and increase bias (underfit). While decreasing k will increase variance and decrease bias (overfit). How variable the predictions are for different data sets at low K. As k increases this variability is reduced. But if we increase k too much, then we no longer follow the true boundary line and we observe high bias. This is the nature of Bias-Variance Tradeoff. Time to build is 0 , but to query depends upon K(number of neighbors). poor choice of distance function gives poor result. Locally weighted regression works better {gives more weight to near points}. This means KNN has preference bias.

iv) Support Vector Machine:
Parameters: 2 differet type of non linear kernel is used
a) PolyKernel (Exponent, c)
b) (Gaussian kernel) RBF (Gamma, C)

The goal is to identify good (C, γ) for RBF and (C, E) for poly kernel, so that the classifier can accurately predict unknown data (i.e. testing data).

Dataset-1 With 70% of data

[Gridserach for (C, γ) for RBF kernel for dataset-1]

Training Attempt	C	γ	ROC Area	%Training Accuracy	%Testing Accuracy	Build Time (sec)	Query Time (sec)
1	25000	.2		76.91%		24.18	
2	2500	.01		76.91%		164.08	
3	2500	.5		76.91%		24.93	
5	25000	.1	.664	71.13	76.19	.12	.01
6	25000	.3	.708	75.6	78.35	.14	.14
7	25000	.5	.720	77.28%	78.35	.08	.01
8	25000	.7	.720	76.72	78.35	.09	.01
9	25000	.9	.733	76.16	79.22%	.09	.01
10	25000	1	.733	75.23%	79.22%	.15	.03

[Grid search for (C, E) for Poly kernel for dataset-1]

Training Attempt	C	E	ROC Area	%Training Accuracy	%Testing Accuracy	Build time (sec)	Query Time (sec)
1	25000	1.0	.727	79.675%	79.22%	1.96	0
2	2	.25		71.35%	67.8%	47.35	
3	25000	1.5	.717	76.53%	77.92%	.12	.01
4	25000	2	.711	77.09%	77.92%	.09	.01
5	25000	2.5	.701	76.90%	77.05%	.14	.01
6	25000	3	.701	76.35%	77.05%	.15	.01
7	25000	3.5	.720	76.53%	78.78	.16	.01
8	25000	5.5	.727	75.04%	79.22%	.33	.01
9	25000	7.5	.704	75.23%	77.48%	1.27	.01

[Gridserach for (C, γ) for RBF kernel for dataset-2]

40% of data is used for training

Training Attempt	C	γ	Avg ROC ARea	%Training Accuracy	%Testing Accuracy	Build time (sec)	Query Time (Sec)
1	25000	.01	.91	29.26%	28.31%	62.62	147.4
2	25000	.1	.96	70.23%	67.8%	15.54	69.57
3	25000	.5	.97	83.48%	81.06%	7.93	41.32
5	25000	.7	.98	85.51%	82.95%	9.27	48.06
6	25000	.9	.987	84.5%	84.33%	7.11	40.96
7	25000	1.2	.992	85.8%	86.31%	8.46	46.58
6	25000	1.6	.994	86.8%	87.68%	7.05	36.14
7	25000	2.5	.997	94.26%	90.28%	7.75	39.45
8	25000	5	.998	96.45%	92.78%	10.42	44.23
7	25000	15	.9994	97.13%	94.01%	23.76	39.45

[Grid search for (C, E) for Poly kernel for dataset-2]

70% of training data is used for training

Training Attempt	C	E	ROC Area	%Training Accuracy	%Testing Accuracy	Build time (sec)	Query Time (sec)

1	2500	2	.981	87.62%	87.56%	26.75	32
1	25000	2	.981	87.62%	87.56%	26.23	31.5
2	25000	3	.995	92.19%	92.13%	36.21	33.16
3	25000	5	.998	94.6%	94.66%	52.68	22.77
4	25000	7	.997	93.12%	93.68%	53.98	22.32
5	25000	7.5	.996	93.5%	93.45%	54.93	22.85

Observations:
Dataset-1&2 {Large C}
The C (slack) parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, which we have chosen 25000, for Dataset-1 and 2, the optimization will choose a smaller-margin hyperplane. Large C gives low bias and high variance. Low bias because we penalize the cost of miss classification a lot. Low value of C is taking much long in build and query time because it is trying to look for vectors that can be clearly classified.

Dataset-1 { low value of gamma }: We have tuned for high C and low gamma. Gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' Small gamma gives low bias and high variance. Very small Gamma can't capture the complexity of model. We can see accuracy rate for very small gamma is low. With low value, model can be made complex with large value of C. That's why we have got good accuracy of around 80%. RBF and poly kernel both has given same accuracy.

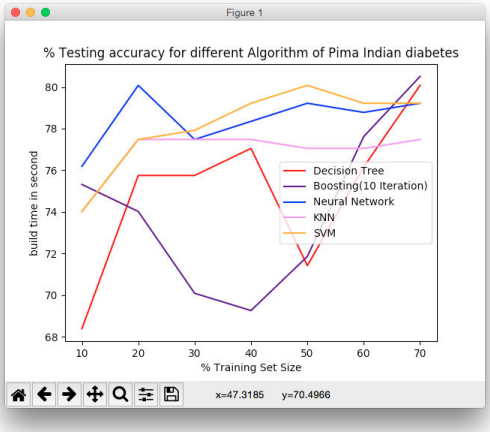
Dataset-2 {Intermediate value of gamma}
High value of gamma tells the influence of single training example doesn't reaches far. Large gamma values gives complex model, hence higher bias and low variance. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.
For the polynomial kernel, the only parameter is "exponent", which controls the degree of the polynomial. E.g Set to 1 for the linear kernel, Set to 2 for the quadratic kernel and 3 for the cubic kernel. For dataset-1, 5.5 Exponent value gives highest accuracy while 7 for dataset-2. Higher order polynomial works for both the datasets. RBF and poly kernel has given approximate same accuracy.

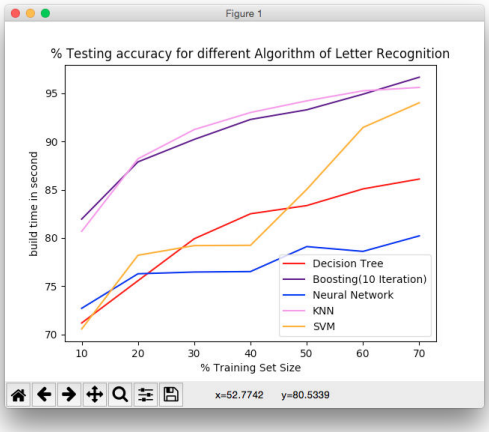
SVM is not suitable for larger datasets because the training time with SVMs is high and much more computationally intensive. SVMs can work effectively on smaller training datasets as they don't rely on the entire dataset. We can see for dataset-2, Query time is much high as training time is low due to 40% data is chosen for training. Dataset-1 has much better build and query time due to smaller dataset. They are less effective on noisier datasets that have overlapping classes, which we can see on dataset-1. Accuracy rate doesn't improve from 80%.
γ (the width of the Gaussian if using an RBF kernel) =0.9 which is high gives better result.

Comparison Of multiple Algorithms and Conclusions:
Reason is already discussed in above sections. We will draw only conclusions from above training attempts
Build Time comparison of DataSet-1:- KNN < Decision Tree < Boosting < NN < SVM
Build Time comparison of DataSet-2:- KNN < Decision Tree < Boosting << NN < SVM

Query Time comparison of Dataset-1:- Decision Tree < SVM = NN = Boosting < KNN
Query Time comparison of Dataset-2:- Decision Tree < NN < Boosting < KNN< SVM

%Testing Accuracy of Dataset-1: KNN < SVM < Decision Tree < NN < SVM
% Testing Accuracy of Dataset-2: NN < SVM < Decision Tree < Boosting < KNN





Best Algorithm and their tuned parameters:

Sr No	Learning Algo	DataSet-1	Avg ROC1 /Accuracy 1	DataSet-2	Avg ROC2/Accuracy 2
1	Decision Tree	Confidence Factor=.15, minNumobj = 23, Pruning=Yes	.847/80.08 %	Confidence Factor=0.15, minNumObj=2, Pruning=Yes	.96/86.1 %
2	Boosting	Iterations =10	.856/80.51 %	Iterations =60	.999/96.66%
3	NN	No of neuron =5, No of hidden layer =1, no. of epoch 40, learning rate = 0.3, momentum = 0.3	.808/79.22 %	No of neuron =17, No of hidden layer =1, no. of epoch 40, learning rate = 0.2, momentum = 0.8	.91/80.21%
4	KNN	K=12, Search Algo= Euclidian Distance, distance weighing = Weight by 1-distance	.779/77.48 %	K=5, Search Algo= Euclidian Distance, distance weighing = Weight by 1/distance	.999/95.31%
5	SVM	PolyKernel Gamma=0.9, C=25000	.733/79.22 %	PolyKernel gamma=7, C=25000	.997/94.01%

References:

https://archive.ics.uci.edu/ml/datasets/Pima%20Indians%20Diabetes
https://archive.ics.uci.edu/ml/datasets/letter+recognition
https://www.hindawi.com/journals/jam/2014/675806/
http://scikit-learn.org/stable/modules/learning_curve.html
http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf
https://matplotlib.org/users/pyplot_tutorial.htm
UDACITY Lectures on Machine Learning