MACHINE LEARNING

Febin Zachariah – 800961027

SUPERVISED LEARNING-NAIVE BAYES

INTRODUCTION

Naive Bayes classifiers are a family of simple probabilistic classifiers based on Bayes theorem. It is a supervised learning algorithm which assumes strong independence between the features. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes is an eager learning classifier and it runs fast even if the datasets are large. Naïve Bayes classifiers are suited for multi class prediction and text classifiers. Naive Bayes classifiers assumes a probabilistic model which allows us to capture uncertainty about the model in a principled way by determining the probabilities of outcomes. Naïve bayes are suited when the dimensionality of inputs is high. Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined. Bayesian classification provides a useful approach for understanding and evaluating many learning algorithms.

Algorithm:

- Classification is based on Bayes theorem.
- We are using Bayes theorem for calculating the posterior probability **P(C|X)** by using the values of P(C), P(X), P(X|C).
- Assume that the effect of the value of a predictor (X) on a given class (C) is independent of the values of other predictors. This is called class conditional independence.
- Therefore,

P(C|X) =(P(X|C) * P(C))/P(X|C)

P(C|X) is the posterior probability of class (target) given predictor (attribute).

P(C) is the prior probability of class.

P(X|C) is the likelihood which is the probability of predictor given class.

P(X) is the prior probability of predictor.

If a categorical variable has a category which was not observed in training data set, then the model will assign a zero probability and it can result in a wrong prediction. To avoid this problem, smoothing method can be used. **Laplace estimation is a smoothing technique**. Naïve Byes is mainly used in categorical input variables. For numeric variables, normal distribution is assumed. Naive Bayes is known to outperform even highly sophisticated classification methods.

We are implementing Naïve Bayes algorithm by using naivebayes library in R. We have also tried with libraries like e1071 and caret to implement the same.

naivebayes Library

For creating the model, we are using **naive_bayes** function to create the model. Different parameters of the function are listed below:

- x: numeric matrix, or a data frame of categorical and/or numeric variables.
- Y: class vector (character/factor/logical).
- formula: an object of class "formula" (or one that can be coerced to "formula") of the form class ~ predictors (class must be a factor/character/logical).
- data: numeric matrix or dataframe with categorical (character/factor/logical) or metric (numeric) predictors.
- Laplace: value used for Laplace smoothing. Defaults to 0 (no Laplace smoothing).
- usekernel: logical; if TRUE, density is used to estimate the densities of metric predictors.
- prior: vector with prior probabilities of the classes. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels.
- Subset: an optional vector specifying a subset of observations to be used in the fitting process.
- eps: double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by threshold.)
- threshold: value by which zero probabilities corresponding to metric variables are replaced (zero probabilities corresponding to categorical variables can be handled with Laplace smoothing).
- Type: if “class”, new data points are classified per the highest posterior probabilities. If “prob” the posterior probabilities for each class are returned.

Optical Recognition of Handwritten Digits Dataset

IMPLEMENTATION

1) For the implementation of Naive Bayes: **R language is used**

2) Packages used: **naivebayes.**

3) Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.

4) After that load the training data and testing data by using **read.csv** command.

5) The dataset for train and test data does not have a header. So, the heaters for the feature variables are names V1 to V64. The header for class variable is named V65.

6) Training data is stored in the **train_data** variable.

7) Test data is stored in the **test_data** variable.

8) **naive_bayes** method is used to create the model based on the training set. Following parameters are passed to the function

1) data frame 2) class vector 3) laplace value for smoothening 4) useKernel 5) prior probabilities for class levels.

```
> model <- naive_bayes(train_data[,-65], train_data[,65], laplace = 1, usekernel = T, prior = NULL)
> model
==================== Naive Bayes ====================
Call:
naive_bayes.default(x = train_data[, -65], y = train_data[, 65],
    prior = NULL, laplace = 1, usekernel = T)

A priori probabilities:

         0          1          2          3          4          5          6          7          8          9
0.09835208 0.10175255 0.09939838 0.10175255 0.10122940 0.09835208 0.09861365 0.10122940 0.09939838 0.09992153

Tables:
```

9) After the creation of the model, we are predicting the output class variable on the test data by using this model. Following parameters are set during prediction.
1) Naïve Bayes model 2) test data 3) type-either class or prob 4) threshold 5) eps value

10) We have obtained a good accuracy with this SVM function by trying with different parameter values. Results and analysis are given in the next section.

Results and Observations

- By using the above model, we have obtained an accuracy of 89.03%.

```
> model <- naive_bayes(train_data[,-65], train_data[,65], laplace = 1, usekernel = F, prior = NULL)
> pred <- predict(model, test_data, type = "class", threshold = 0.2, eps = 0.00000000000000000000001)
> mean(pred == test_data$V65)*100
[1] 89.03728
>
```

- We have tried with different values for the parameters of naive_bayes function to improve the accuracy of the model and we obtained the accuracy results as shown in the next table.
- There are 6 parameters that we have used to analyze the change in accuracy. Please see the next page for the different values that we have tried to obtain the best accuracy.

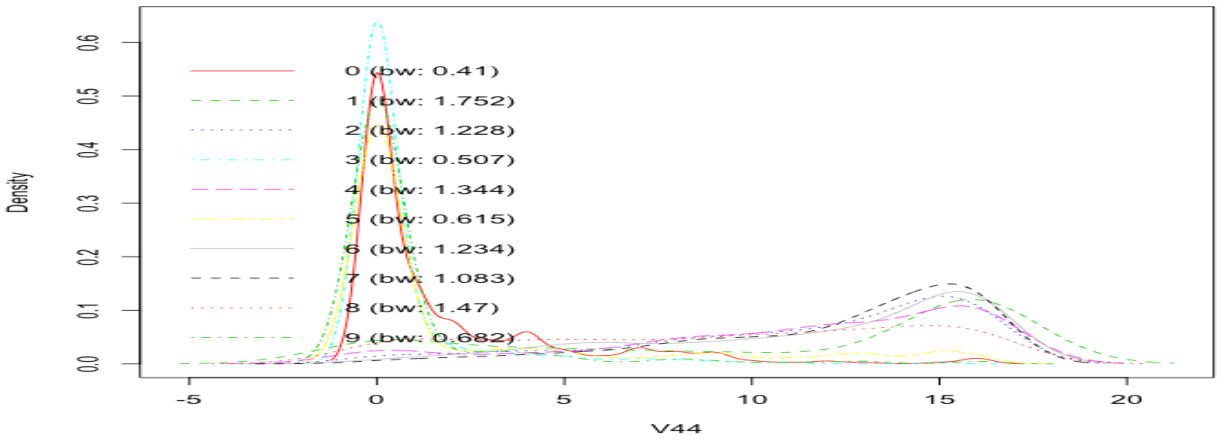
laplace	usekernel	prior	type	threshold	eps	Accuracy
1	FALSE	c(0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2)	class	0.2	1.00E-22	89.03
5	FALSE	c(0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2)	prob	0.2	1.00E-22	89.03
1	FALSE	NULL	class	0.2	1.00E-23	89.03
5	FALSE	NULL	prob	0.2	0.000000001	89.03
5	FALSE	NULL	class	0.2	1.00E-23	89.03
1	FALSE	NULL	prob	0.1	0	88.64
1	FALSE	NULL	class	0.1	0	88.64
0	FALSE	NULL	prob	0.1	0	88.64
0	FALSE	NULL	class	0.1	0	88.64
0	TRUE	NULL	class	0.1	0	88.42
0	TRUE	NULL	prob	0.1	0	88.42
5	TRUE	NULL	prob	0.2	0.000000001	88.36
5	FALSE	c(0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2)	prob	0.01	1.00E-23	86.7
5	FALSE	c(0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2)	class	0.01	1.00E-23	86.7
5	TRUE	c(0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2,0.2)	class	0.01	1.00E-23	86.7
5	FALSE	NULL	prob	0.0001	1.00E-23	76.2

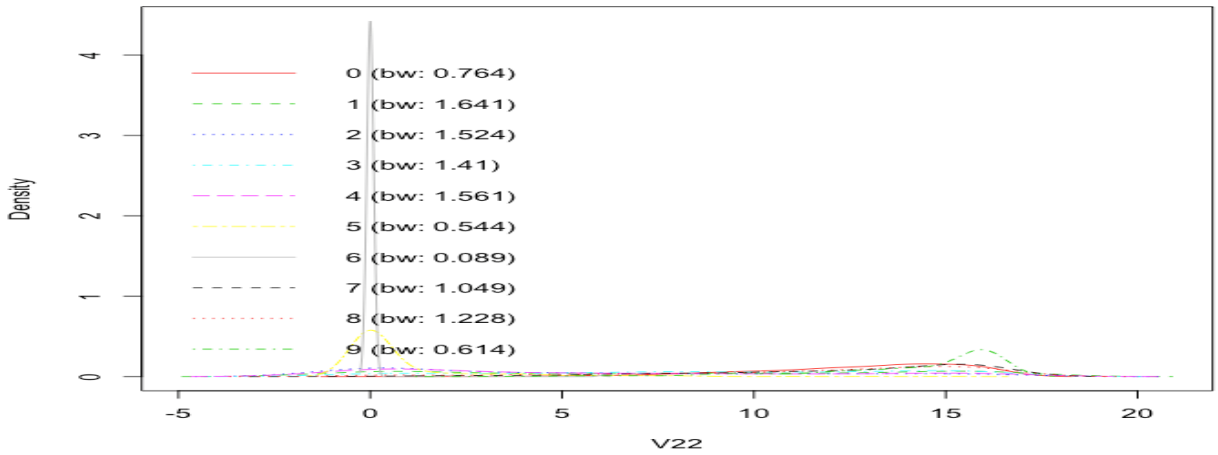
- We can see from the table that highest accuracies are obtained when usekernel value is set to false.
- Highest accuracy is obtained when threshold is >=0.2, and eps is 10^{-9} to 10^{-23}
- Laplace, type, prior doesn't have much effect on this dataset.
- Lowest accuracy we have obtained is 76.2% when the threshold is less than 0.0001.
- Tried with different values of laplace and it wasn't having any effect on the test data.
- prior is class vector specifying the prior probabilities for each class variable (this is automatically calculated if set to NULL).
- Setting different probability values for the prior parameter also doesn't have much effect on this dataset.
- Confusion matrix for this classification is given below.

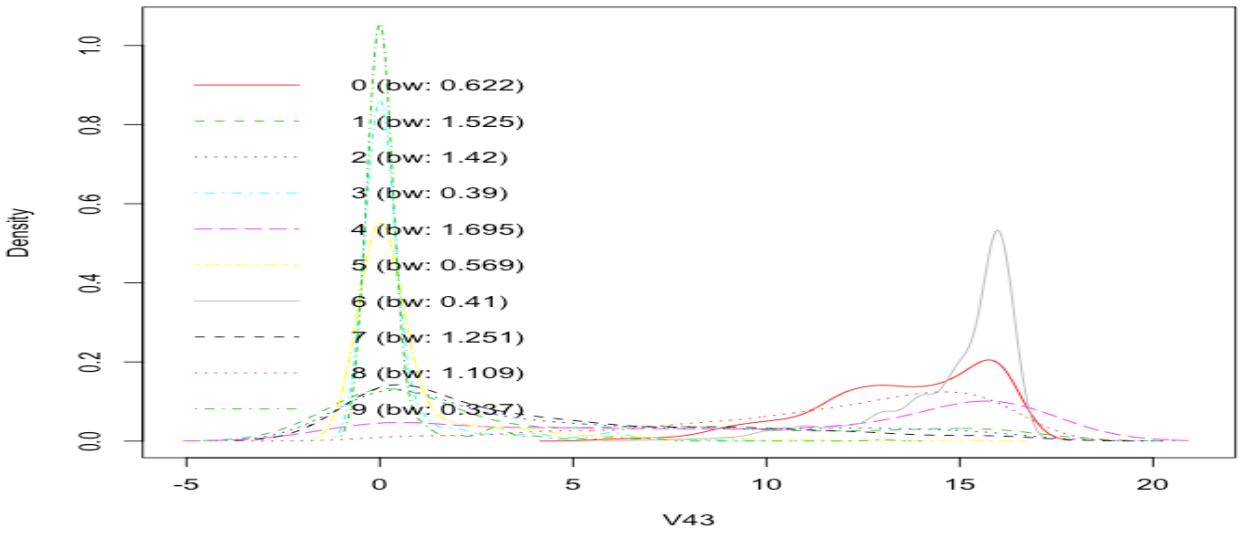
```
> confusionMatrix(idx-1, test_data$V65)
Confusion Matrix and Statistics

          Reference
Prediction   0   1   2   3   4   5   6   7   8   9
         0 171   0   0   1   0   0   1   0   0   0
         1   0 155   7   0   2   0   3   1  27   1
         2   0   8 154   0   1   1   0   1   0   0
         3   0   0   4 164   0   2   0   0   4  24
         4   6   0   2   0 173   1   3  11   2   7
         5   0   2   1   7   0 177   1   3   7   7
         6   1   3   0   0   1   0 173   0   1   0
         7   0   0   2   5   2   0   0 163   1   3
         8   0   1   5   3   1   0   0   0 123   3
         9   0  13   2   3   1   1   0   0   9 135

Overall Statistics

               Accuracy : 0.883695
                 95% CI : (0.8679643, 0.8981604)
    No Information Rate : 0.1018364
    P-Value [Acc > NIR] : < 0.00000000000000022204

                  Kappa : 0.8707507
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: 0   Class: 1   Class: 2   Class: 3   Class: 4  Class: 5   Class: 6   Class: 7
Sensitivity          0.96067416 0.85164835 0.87005650 0.89617486 0.95580110 0.9725275 0.95580110 0.91061453
Specificity          0.99876467 0.97461300 0.99320988 0.97893432 0.98019802 0.9826625 0.99628713 0.99196539
Pos Pred Value       0.98843931 0.79081633 0.93333333 0.82828283 0.84390244 0.8634146 0.96648045 0.92613636
Neg Pred Value       0.99568966 0.98313554 0.98590686 0.98811757 0.99497487 0.9968593 0.99505562 0.99012955
Prevalence           0.09905398 0.10127991 0.09849750 0.10183639 0.10072343 0.1012799 0.10072343 0.09961046
Detection Rate       0.09515860 0.08625487 0.08569839 0.09126322 0.09627156 0.0984975 0.09627156 0.09070673
Detection Prevalence 0.09627156 0.10907067 0.09181970 0.11018364 0.11407902 0.1140790 0.09961046 0.09794101
Balanced Accuracy    0.97971941 0.91313068 0.93163319 0.93755459 0.96799956 0.9775950 0.97604412 0.95128996
                     Class: 8   Class: 9
Sensitivity          0.70689655 0.75000000
Specificity          0.99199014 0.98206555
Pos Pred Value       0.90441176 0.82317073
Neg Pred Value       0.96929561 0.97244336
Prevalence           0.09682805 0.10016694
Detection Rate       0.06844741 0.07512521
Detection Prevalence 0.07568169 0.09126322
Balanced Accuracy    0.84944335 0.86603278
```

- We have 64 features in total and it's not feasible to show the graph for all the features. Following graphs displays probability density with 3 highly correlated features (v44,v22,v43).







Note: we have also tried the naïve bayes implementation using caret and e1071 library. We have obtained an accuracy like naivebayes library.

Conclusions

- We have obtained a maximum accuracy of 89.03% on the test data.
- We have constantly obtained an accuracy above 86% when threshold greater than 0.01
- We are getting more accuracy when usekernel is set to false
- Successfully implemented Naive Bayes algorithm by using naivebayes library.

Amazon Reviews Sentiment Analysis Dataset

IMPLEMENTATION

1) For the implementation of knn: **R language is used**

2) Packages used: **naivebayes.**

3) Load the packages by using **library** command in R if it is already installed, otherwise install the libraries by using **install. packages** command.

4) Headers of the datasets: name, review, rating.

5) To improve the sentiment score accuracy, we implemented sentiment calculation by using Vader Sentiment library perform and it was giving better performance compared to RSentiment library from R.

6) The SentimentIntensityAnalyzer method takes the text review as input and output four values, namely, “pos” (positive score of the review), “neg” (negative score of the review), “neu” (neutral score of the review), “compound” (normalized value of the sum of all sentiment scores).

7) After calculating the sentiment scores of both training data and test data, sentiment scores of both the files are stored in separated files namely sentiment_train.csv and sentiment_test.csv respectively.

8) Now we are loading this files in our R script by using read.csv command.

9) Training data is stored in the **train_data** variable.

10) Test data is stored in the **test_data** variable.

11) **naive_bayes** method is used to create the model based on the training set. Following parameters are passed to the function:

1) data frame 2) class vector 3) laplace value for smoothening 4) useKernel 5) prior probabilities for class levels.

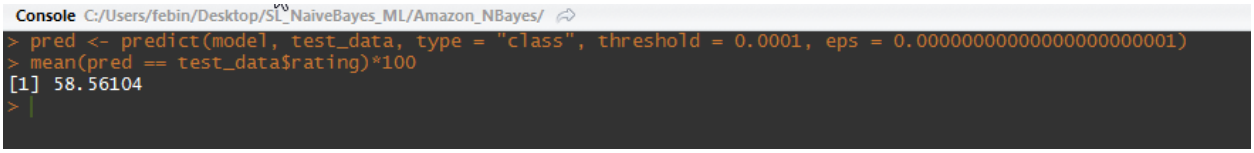
```
> model <- naive_bayes(rating~negative_score+positive_score+neutral_score+compound_value, data = train_data,
+                      laplace = 5, usekernel = F, prior = NULL)
> model
==================== Naive Bayes ====================
Call:
naive_bayes.formula(formula = rating ~ negative_score + positive_score +
    neutral_score + compound_value, data = train_data, prior = NULL,
    laplace = 5, usekernel = F)

A priori probabilities:

        1         2         3         4         5
0.08272490 0.06157032 0.09102054 0.18054950 0.58413475
```

12) 9) After the creation of the model, we are predicting the output class variable on the test data by using this model. Following parameters are set during prediction.
1) Naïve Bayes model 2) test data 3) type-either class or prob 4) threshold 5) eps value

13) We have obtained a good accuracy with this SVM function by trying with different parameter values. Results and analysis are given in the next section.

Results and Observations

- By using the above model, we have obtained an accuracy of 58.56%



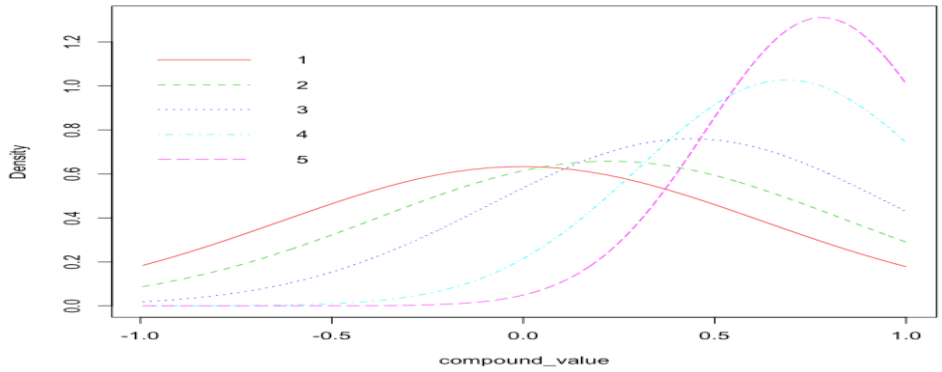
- We have tried with different values for the parameters of naive_bayes function to improve the accuracy of the model and we obtained the accuracy results as shown in the next table.
- There are 6 parameters that we have used to analyze the change in accuracy.

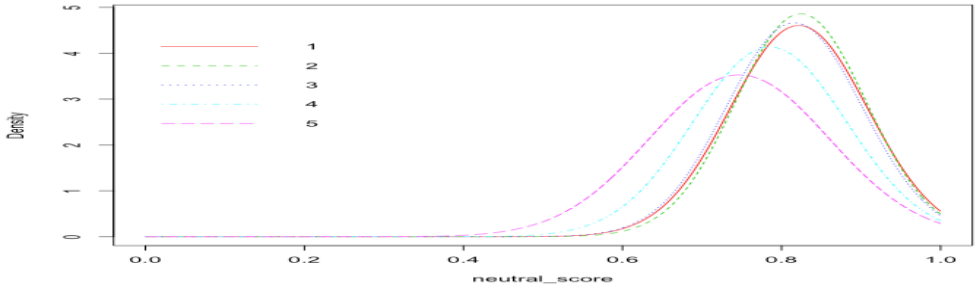
laplace	usekernel	prior	type	threshold	eps	Accuracy (%)
5	FALSE	NULL	prob	0.2	0.000000001	58.56
5	FALSE	NULL	prob	0.3	1.00E-23	58.56
5	FALSE	NULL	prob	0.001	1.00E-23	58.56
5	FALSE	NULL	prob	0.0001	1.00E-23	58.56
0	FALSE	NULL	class	0.1	0	58.56
1	FALSE	NULL	prob	0.1	0	58.56
1	TRUE	NULL	prob	0.01	0.1	58.06
5	TRUE	NULL	prob	0.2	0.000000001	58.06
0	TRUE	NULL	prob	0.1	0	58.05
1	TRUE	NULL	prob	0.1	0	58.05
1	TRUE	NULL	class	0.1	0	58.05
5	TRUE	c(0.2,0.2,0.2,0.2,0.2)	prob	0.01	1.00E-23	48.39
5	TRUE	c(0.2,0.2,0.2,0.2,0.2)	prob	0.2	1.00E-23	48.38
5	TRUE	c(0.2,0.2,0.2,0.2,0.2)	class	0.2	1.00E-23	48.38
5	FALSE	c(0.2,0.2,0.2,0.2,0.2)	class	0.01	1.00E-23	45.37
5	FALSE	c(0.2,0.2,0.2,0.2,0.2)	prob	0.1	1.00E-23	45.37
5	FALSE	c(0.2,0.2,0.2,0.2,0.2)	prob	0.2	1.00E-23	45.37

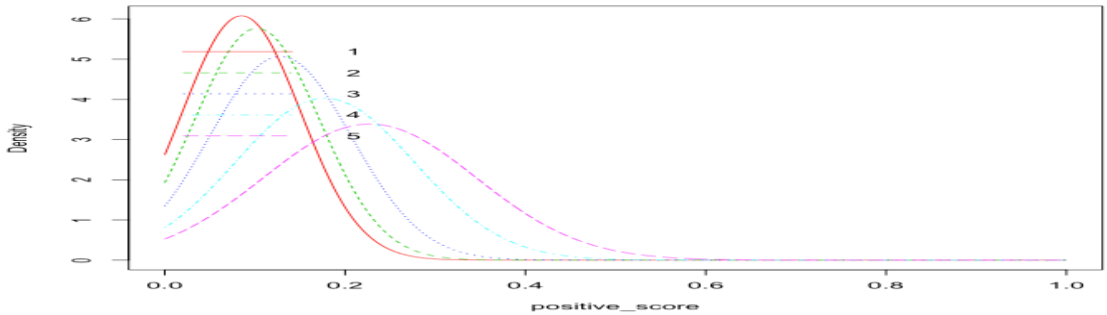
- For this dataset, we are obtaining the highest accuracy of 58.56% when the usekernel is set to false.
- For threshold >=0.1, we are getting high accuracy.
- Laplace, prior, type doesn't make much difference in the accuracy value.
- Least accuracy obtained for amazon dataset is 45.37%.
- prior is class vector specifying the prior probabilities for each class variable (this is automatically calculated if set to NULL).
- Setting different probability values for the prior parameter also doesn't have much effect on this dataset.
- Running time for naïve bayes algorithm is really less for amazon dataset as compared to other algorithms.
- Confusion Matrix for this classification is shown in the next page.

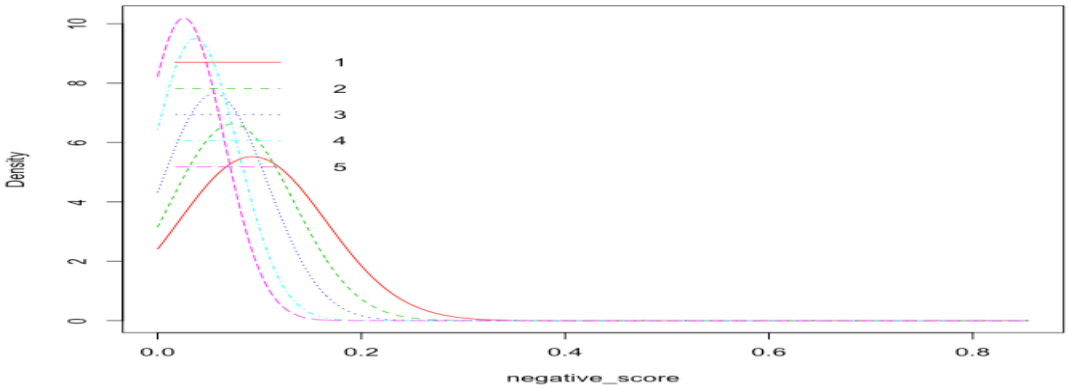
```
Console C:/Users/febin/Desktop/SL_NaiveBayes_ML/Amazon_NBayes/
> confusionMatrix(pred, test_data$rating)
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1  1664   939   900   675  1205
         2     0     0     0     0     0
         3   251   216   278   313   482
         4    43    37    34    40    88
         5  1079  1078  2203  5668 19514

Overall Statistics

               Accuracy : 0.5856
                 95% CI : (0.5806, 0.5907)
    No Information Rate : 0.58
    P-Value [Acc > NIR] : 0.01445

                  Kappa : 0.1969
 Mcnemar's Test P-Value : < 2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity           0.54791  0.00000 0.081406 0.005974   0.9166
Specificity           0.88955  1.00000 0.962093 0.993269   0.3496
Pos Pred Value        0.30912      NaN 0.180519 0.165289   0.6606
Neg Pred Value        0.95617  0.93816 0.910797 0.817469   0.7523
Prevalence            0.08274  0.06184 0.093034 0.182418   0.5800
Detection Rate        0.04533  0.00000 0.007573 0.001090   0.5316
Detection Prevalence  0.14665  0.00000 0.041954 0.006593   0.8048
Balanced Accuracy     0.71873  0.50000 0.521749 0.499621   0.6331
> |
```

- Following graphs displays probability density with different parameters (compound_value, neutral score, positive score and negative score)


Density
compound_value
1
2
3
4
5


Density
neutral_score
1
2
3
4
5


Density
positive_score
1
2
3
4
5


Density
negative_score
1
2
3
4
5

Conclusions

- We have obtained a maximum accuracy of 58.56% on the test data.
- Accuracy hasn't improved as expected but we can see that the running time for amazon dataset has reduced as compared to other algorithms.
- Successfully implemented Naive A algorithm by using naivebayes library.

Summary of Supervised Machine Learning Algorithms

I have implemented 6 supervised learning algorithms to perform classification in two domains.

Comparison of Algorithms

Decision Tree Classification

1) **Rpart library** is used to create the model. Parameters used: formula, data, params, method, minbucket, minsplit.
2) Pruning is done to avoid overfitting of the data. **Pruning is done by using the prune method**. Decision tree generated is passed. "Cp" parameter is used to trim the decision tree passed. The rpart object which holds the decision tree also has a list of Cp values, from which we can select the one having the least cross-validated error. This value is dynamically selected based on the rpart object to get the most optimal pruned tree.
3) There are two types of pruning: Pre-pruning, which prevents the generating of non-significant branches. Pre-pruning involves a stopping criteria to determine when it is desirable to stop expanding some of the branches prematurely as the tree is generated. This requires calculating the goodness of a split when constructing a tree. If a split fall below a pre-specified threshold, then the branch is not expanded further. Post-pruning, where the tree is first generated completely and then the non-significant branches are removed. One way to do this is to replace some of its subtrees with leaf nodes, and hence we get a smaller pruned tree.
4) Results & Observations:
   OCR Dataset: Training speed and prediction speed for digit dataset is fast since we have just 3823 observations. **Cross validation dataset obtained an accuracy of 78.45% and test data obtained an accuracy of 75.04%.**
   Amazon Dataset: Training speed becomes extremely slow for amazon dataset and this can be attributed to the huge number of observations available. I have obtained **an accuracy of 61.54% on the cross-validation data and 61.07% accuracy on the test data.**

Neural Network Classification

1) **Neuralnet** library is used to create the model by using the training data. It provides a custom-choice of activation and error function.

2) Resilient backpropagation with weight backtracking is the algorithm used to train the neural network model. Other available algorithms are backpropagation, Resilient backpropagation without weight backtracking etc...
3) Hidden parameter is used to specify the number of hidden layers to be used while creating the model. Best result is obtained when hidden is set to 10. I have also tried with values like 5,10,20,30,50,75,100. Algorithm was not converging when hidden value is set to 100.
4) Threshold parameter is set to 0.01. I have tried with different values for the parameter and the best result is obtained when it is set 0.01. Threshold value determines the stopping criteria for the partial derivatives of the error function.
5) Training speed is generally slow for neural networks and can increase exponentially as the number of observations increase. The training time for amazon dataset when using only 50% of the dataset was just about 15 minutes, but when using the full dataset, it is 47 minutes.
6) Result & Observations:
   ORC Dataset: **Obtained an accuracy of 90% on the test data and 93% accuracy on the cross-validation data.**
   Amazon Dataset: **Obtained an accuracy of 56.1% on the cross-validation dataset and 52.09% accuracy on the actual test data.**

K-Nearest Neighbors:

1) Class library is used to implement knn algorithm.
2) k-Nearest Neighbor (k-NN) algorithm is widely used for predictive analytics. It can be used for both classification and regression predictive problems. Some characteristics of k-NN are, 1) It does not make any assumption on the underlying data distribution. 2) It is a lazy algorithm, which means it does not use training data points to make any generalization.
3) The train dataset, test dataset, and class variables to the train dataset is passed as parameters to "knn". The parameter "l" (whose default value is 0), specifies the minimum vote for a definite decision. The value of "l" needs to be less than k-1. Have tried different values for "l", but the default value of 0 gives the best result. Have also tried k value up to 100 and the accuracy for each iteration is recorded.
4) The number "k" in this algorithm decides how many neighbors influence the classification. The value of "k" is usually an odd number if the number of classes is 2.
5) Memory issue was one problem while implementing knn.
6) For amazon dataset, if k value is more than 3 we are getting an error saying, "too many ties in knn".
7) For k-NN, training speed is fast irrespective of the number of observations since there is no explicit training phase. Prediction speed depends on the value of n. For digit dataset, I could do prediction for values of k up to 100. But for amazon dataset, a value of k greater than 3 will not work as the algorithm does not converge. Also, excluding feature selection, this algorithm does not need any parameter tuning.
8) Result & Observations:
   OCR Dataset: K value is not restricted to an odd number since we have more than two class types. **Highest accuracy obtained is 98.44% on the test data and we can see that as**

k increases accuracy is getting decreased. Even for k=100, we have obtained an accuracy of 95%.

Amazon Dataset: **Highest accuracy obtained is 51.98% when k is set to 3.**

Boosting

1) Libraries used: caret, caretEnsemble, C50 to implement boosting on two classifier algorithms namely, C5.0 and gradient boosting.
2) A boosting technique combines a set of weak learners (a classifier slightly correlated with true classification) and converts it into a strong learner (a classifier well correlated with the true classification).
3) Train method is used to train the data by specifying the parameters: formula, method (either C5.0/gbm), metric (Accuracy/Kappa Classification) and trainControl object for cross-validation.
4) If we are using C 5.0, global pruning is performed by default. Two stages of pruning are there:1) Local pruning stage for examining the sub tress and collapses branches 2) Global pruning stage which considers the tree as whole.
5) Training speed for boosting techniques is generally slow while creating a model since its just combining a set of weak learners to create a strong learner, but the prediction speed is fast.
6) Result & Observations:

   OCR Dataset: C5.0 method with boosting gives an **accuracy of 95.49%** accuracy **96.3% accuracy with gradient boosting**. We can clearly see that boosting has improved the accuracy of the dataset. We got only 77% accuracy with decision tree algorithm and we are getting an accuracy of 95.49% by using boosting with C5.0 model.

   Amazon Dataset: We have obtained an accuracy of 60.1% accuracy with C5.0 and gradient boosting respectively.

Support Vector Machines

1) Library used: e1071.
2) SVM is a discriminative classifier formally defined by a separating hyperplane. It is a supervised machine learning algorithm which can be used for both classification and regression problems, but is mainly used for classification problems.
3) By using the **svm function**, we are creating the model based on the training dataset. Following parameters are passed to the svm function

   1)Dataset 2) Kernel (linear, polynomial, sigmoid) 3) type (c classification, nu-classification) 4) cross (number of folds for cross validation) 5) degree (used when kernel is polynomial).

4) Result & Observations:

   OCR Dataset: On trying with different combinations of the parameters, we have obtained a highest accuracy of 97.88% when type = C-classification, kernel = polynomial, gamma = 1000, cost = 0.0001.

Amazon Dataset: We have obtained accuracy of 60.09% for the test data when, type = C-classification, kernel = linear, cost = 0.01, gamma = 100

Naive Bayes Classification

1) Library used: naivebayes
2) Naïve Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes theorem with strong independence assumptions between the features.
3) **naive_bayes** method is used to create the model based on the training set. Following parameters are passed to the function
   1) data frame 2) class vector 3) laplace value for smoothening 4) useKernel 5) prior probabilities for class levels.
4) After the creation of the model, we are predicting the output class variable on the test data by using this model. Following parameters are set during prediction.
   1) Naïve Bayes model 2) test data 3) type-either class or prob 4) threshold 5) eps value
5) Result & Observations:
   OCR Dataset: We have tried with different parameters and obtained an accuracy of 89.03% on the test data. The accuracy rate is marginally higher when the usekernel is False (kernel is not used), threshold is >=0.2, and eps is 10^{-9} to 10^{-23}. The values for laplace, prior and type does not make much of a difference. Lowest accuracy obtained is 76.2%.
   Amazon Dataset: We have tried with different parameters and the highest accuracy obtained is 58.56. The accuracy rate is marginally higher when the usekernel is False (kernel is not used). Accuracy is high when threshold is >=0.1 most of the time, except for two scenarios when threshold is 0.001 and 0.0001. eps are 10^{-9} to 10^{-23}. The values for laplace, prior and type does not make much of a difference. The accuracy hasn't improved even if the algorithm runs fast.

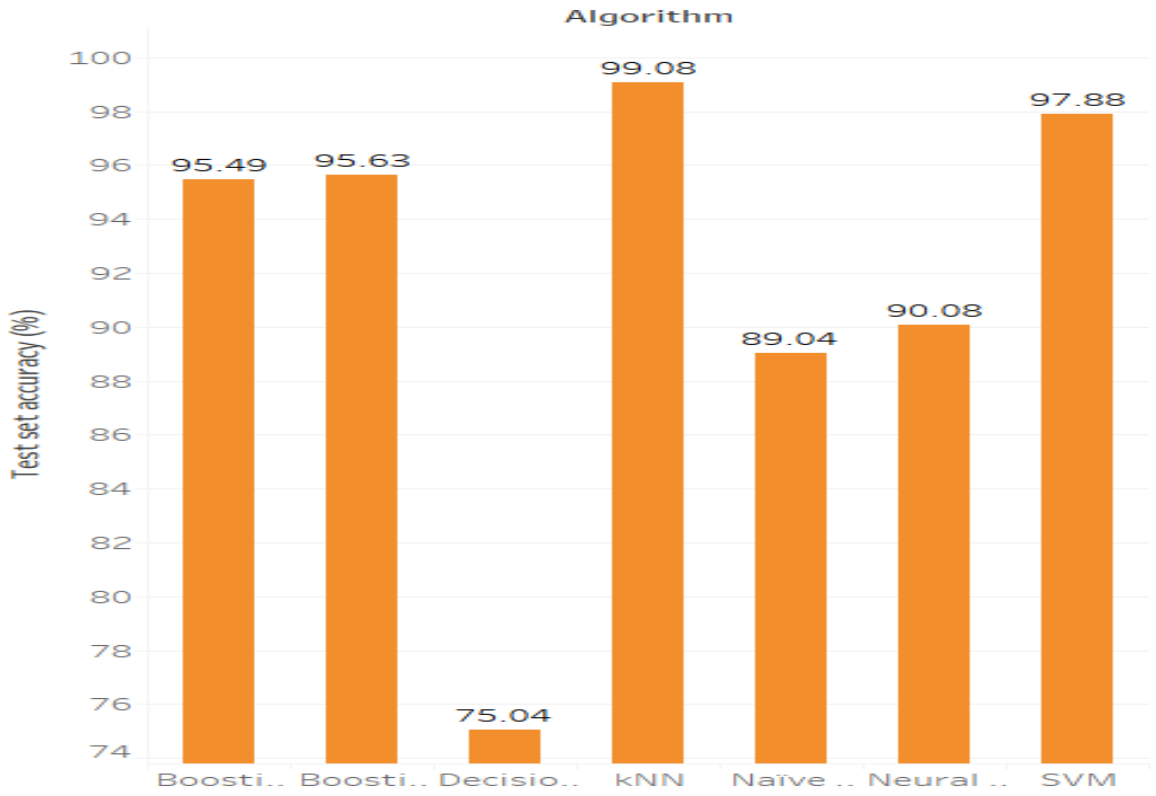
6) Naïve Bayes runs fast as compared to other algorithms.
7) Training speed and prediction speed is the fastest for naïve Bayes classifier.

Observations & Key Findings

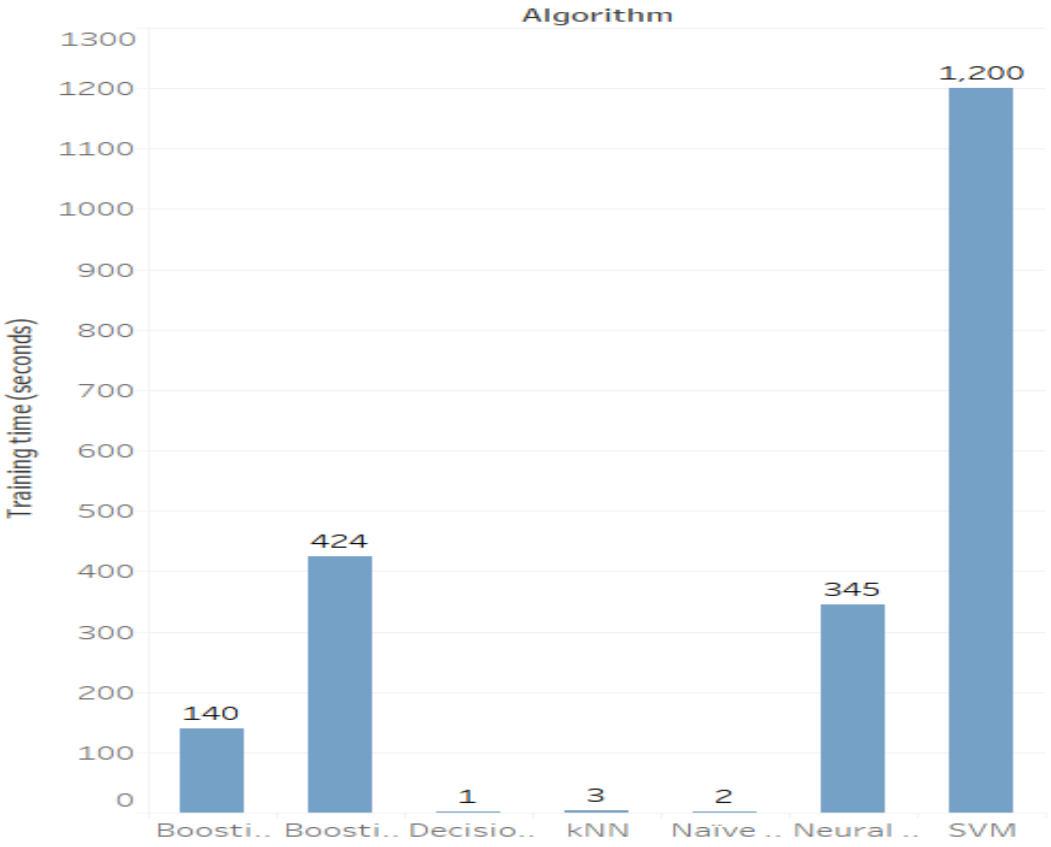
Digit Recognition Dataset

Digit Recognition				
Algorithm	Training size (observations)	Training set accuracy (%)	Test set accuracy (%)	Training time (seconds)
Decision tree	3823	76.86	75.04	1
Neural network	3823	93.11	90.08	345
kNN	3823	-	99.08	3
Boosting-C5.0	3823	97.2	95.49	140
Boosting-GBM	3823	97.3	95.63	424
SVM	3823	99.08	97.88	1200
Naïve Bayes	3823	-	89.04	2

Following Graph represents the Accuracy of each algorithm on Test data.



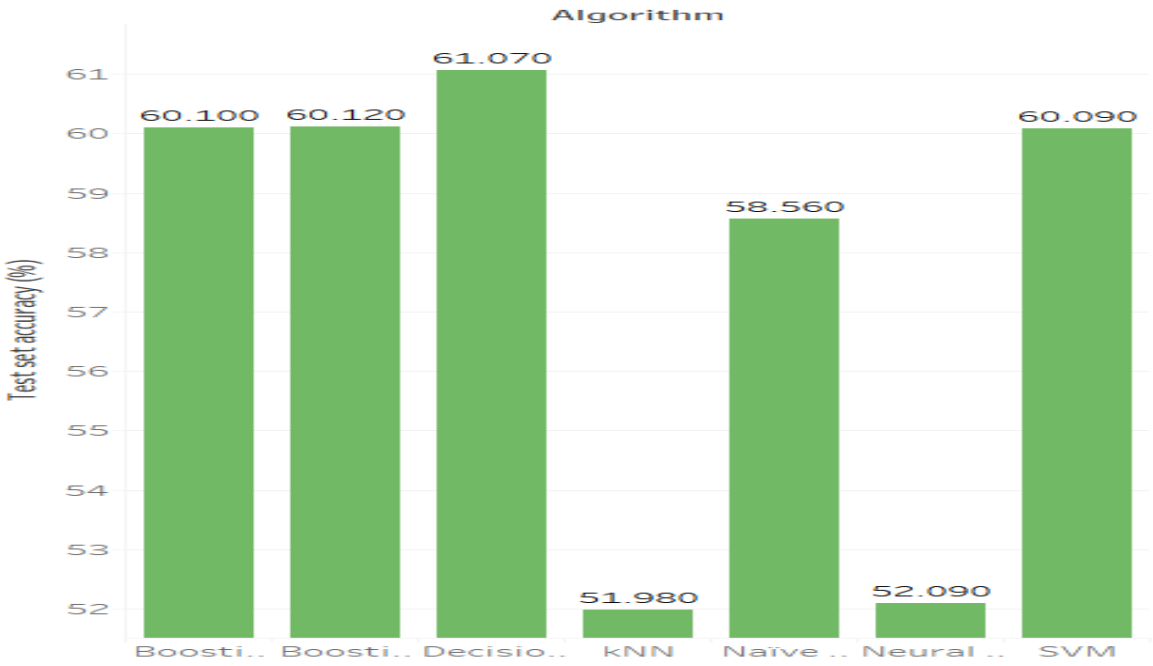
Following Graph represents the Time Taken(Seconds) for each algorithm on Test data.



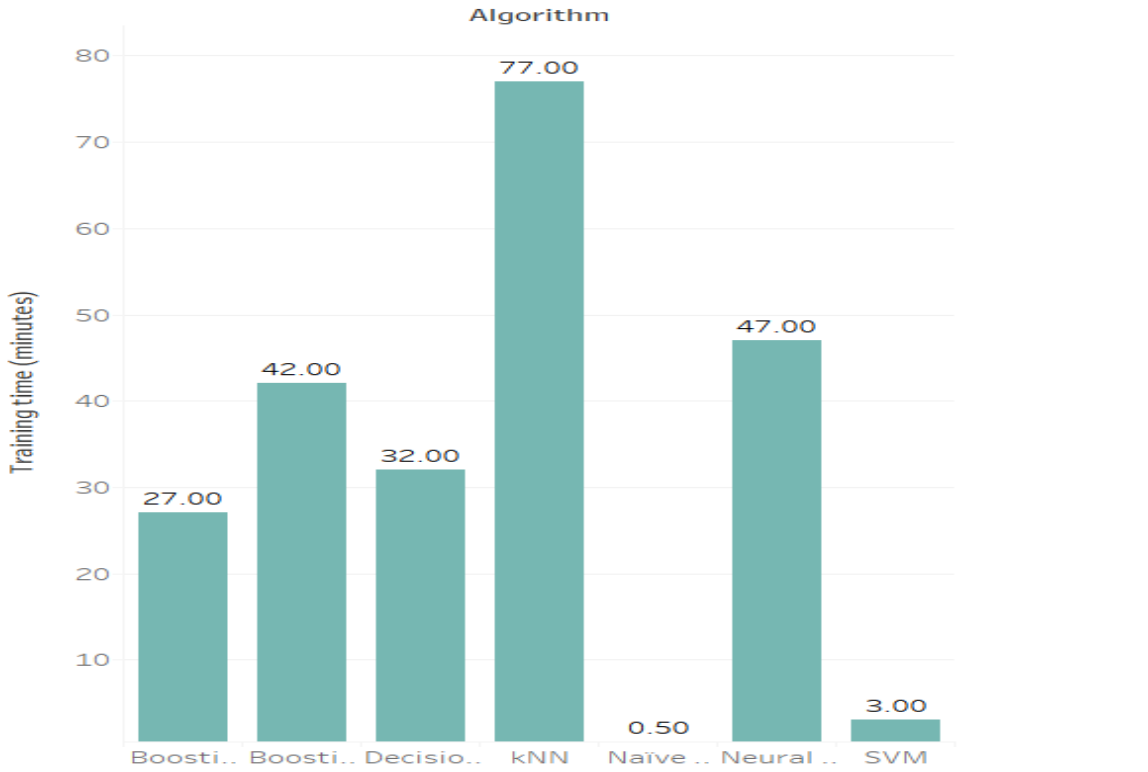
Amazon Baby Product Review Dataset

Amazon review				
Algorithm	**Training size (observations)**	**Training set accuracy (%)**	**Test set accuracy (%)**	**Training time (minutes)**
Decision tree	117459	61.54	61.07	32
Neural network	146824	56.01	52.09	47
kNN	51388	-	51.98	77
Boosting-C5.0	51388	60.22	60.1	27
Boosting-GBM	51388	60.15	60.12	42
SVM	22023	60.75	60.09	3
Naïve Bayes	146824	-	58.56	0.5

Following Graph represents the Accuracy of each algorithm on Test data.



Following Graph represents the Time Taken(Seconds) for each algorithm on Test data.



Results for Digit Recognition Dataset

- I have created 6 models for **digit recognition dataset**. **In terms of wall clock time**, fastest running algorithm was decision tree model. Accuracy for decision tree model was the lowest. Naive Bayes and support vector machines also took very less time.

- The **best performance in terms of accuracy**, achieved on the digit recognition test data was by k-Nearest Neighbors, which is 99.08%, and the wall clock time for this algorithm was about 3 seconds, which is not much in comparison to the decision tree algorithm.

- There were other algorithms which also had good accuracy rates, but on the account of having around 64 features to work with, their running time was significantly very high. SVM took 1200 seconds to create the model whereas Gradient boosting took about 424 seconds.

- Digit Recognition dataset contains all numerical data. Therefore, it was easy to implement the algorithms without any conversion.

- In case of k-Nearest Neighbors, there is a property named "axiom of neighborliness", which states that, for every finite set S, whose elements are classified by some class, we can say that S satisfies the axiom of neighborliness if for every element x in S, element y is the closest to x and the class of x and y are same. For knn, I have tested with different values of k and all were giving an accuracy above 94% which is good.

- We have used cross-validation for all the algorithms except for knn. Knn is not having cross-validation since it does not have a training phase

Results for Amazon Baby Product Review Dataset

- **In terms of wall clock time, the fastest running algorithm was Naïve Bayes,** which took less than a minute to create the model. Support Vector machine also created the model in less than 5 minutes.
- **The best performance in terms of accuracy was obtained with decision tree algorithm** and all other models other than neural network and knn gave an accuracy above 58.5%.
- Least accuracy was obtained knn, which is about 51.98%.
- A better sentiment score calculation can give better accuracy result for amazon dataset.
- I was not able to use the full dataset available for generating model. For SVM, I could use only up to 25% of the training dataset. k-NN, boosting techniques could handle only 30% of the data. Decision tree could take up to 80%. Only Naïve Bayes classifier and Neural

networks could handle the full training dataset, and this was mainly because of less number of features available for amazon dataset.

- Memory issue with knn, since all the training data is stored in RAM for computation. I faced memory issues while running amazon dataset.

- k-Nearest Neighbors to the highest amount of wall clock time for amazon dataset and the algorithm would not converge for values of k greater than 3.

Conclusion: From all the above analysis and findings, I can say that the best algorithm is Support Vector machine, because it performed really well both the classification domains with good accuracy and less training time.

Future Improvements

Following changes can be implemented to improve the performance of the algorithm.

- Currently for decision tree, we are creating the model in two steps. First, we are creating a tree which overfits the data and in the second step we are pruning the tree to reduce the tree size and avoid overfitting. It will be good if we can do it in single step.
- Neural network is also having the problem of overfitting. To improve the generalization of neural networks, train multiple neural networks and average their output.
- For knn, the main problem that we faced was the running time issue with amazon dataset. We can improve this by assigning weights for features which are important.
- We have seen that for amazon dataset, we are not getting accuracy above 62% for all the models. We can improve this by developing a better sentiment analyzer. After the conclusion of this project, I am planning to research on this to improve the sentiment calculation.