UNIVERSITY OF
CENTRAL MISSOURI.
LEARNING TO A GREATER DEGREE

COLLEGE OF HEALTH, SCIENCE AND TECHNOLOGY
DEPARTMENT: COMPUTER SCIENCE

PROJECT:
Classification Techniques.
(Crowdsourced Mapping Dataset)

SUBMITTED TO:
Prof Dr. Lianwen Wang

SUBMITTED BY:
Alpa Arvind Bandekar (axb85820@ucmo.edu)
Sayali Shridhar Vakil (sxv71740@ucmo.edu)
Swapnali Sunil Wakchaure (sxw33070@ucmo.edu)

Contents

1. Contribution to the project by each group member

Name	**Responsibilities**
Alpa Arvind Bandekar	• Worked on R code for all classification techniques (over skype calls) • Documentation created for Bagging, • Documentation created for Random Forest • Comparison of multiple classification techniques
Sayali Shridhar Vakil	• Worked on R code for all classification techniques (over skype calls) • An introduction to the project • Documentation created for Decision Tree Using Hold-Out Method • Documentation created for Result Decision Tree Using Boosting Method
Swapnali Sunil Wakchaure	• Worked on R code for all classification techniques (over skype calls) • Documentation created for Naive-Bayes Classification • Documentation created for Result of support vector machine using linear kernel with different costs • Potential performance issues • Possible Future Study • Documentation created for Result of support vector machine using radial kernel with different costs and gammas • Documentation created for Result of support vector machine using polynomial kernels with different costs and gammas • Conclusion of the project • References

2. An introduction to the project

Crowdsourced Mapping Data Set Description:

This dataset was derived from geospatial data from two sources:

1. Landsat time-series satellite imagery from the years 2014-2015, and
2. Crowdsourced georeferenced polygons with land cover labels obtained from OpenStreetMap.

The crowdsourced polygons cover only a small part of the image area, and are used to extract training data from the image for classifying the rest of the image. The main challenge with the dataset is that both the imagery and the crowdsourced data contain noise (due to cloud cover in the images and inaccurate labeling/digitizing of polygons).

Attribute Information:

- **Class:** The following class labels are covered by land
    1. Impervious
    2. Farm
    3. Forest
    4. Grass
    5. Orchard
    6. Water

  This responsible variable **class** has 6 labels. The dataset contains around 10546 records. We have selected '**Farm'** and '**Forest'** as two labels which have maximum number of records (total records in this updated dataset are 8873 after deleting data for the labels 'Impervious', 'Grass', 'Orchard', 'Water')

- **max_ndvi:** the maximum NDVI (normalized difference vegetation index) value derived from the time-series of satellite images.

- **20150720_N to 20140101_N:** NDVI values extracted from satellite images acquired between January 2014 and July 2015, in reverse chronological order (dates given in the format yyyymmdd).

Note: We have run below query to rename variable names class to class1 and 20150720_N to Jul_20_2015 for better understanding.

```
> CrowdSourceData=read.csv("C://Users//sayal//OneDrive//Desktop//Data Mining//DM Proje
ct//Crowdsourced Mapping//training.csv", header=T, na.strings ="?")
>
> ColumnName <- c("class1","max_ndvi", "Jul_20_2015",     "Jun_02_2015", "May_17_2015"
,    "May_01_2015"   ,"Apr_15_2015","Mar_30_2015", "Mar_14_2015", "Feb_26_2015", "Feb_1
0_2015",     "Jan_25_2015", "Jan_09_2015", "Nov_17_2014", "Nov_01_2014", "Oct_16_2014"
,    "Sep_30_2014", "Aug_13_2014", "Jun_26_2014", "Jun_10_2014","May_25_2014", "May_09_
2014", "Apr_23_2014", "Apr_07_2014", "Mar_22_2014", "Feb_18_2014", "Feb_02_2014", "Jan
_17_2014", "Jan_01_2014")
> names(CrowdSourceData) <- ColumnName
```

Crowdsourced Dataset with few observations:

```
> fix(CrowdSourceData)
```

	class1	max_ndvi	Jul_20_2015	Jun_02_2015	May_17_2015	May_01_2015	Apr_15_2015	Mar_30_2015	Mar_14_2015	Feb_26_2015	Feb_10_2015	Jan_25_2015	Jan_09_2015	Nov_17_2014	Nov_01_2014
7407	forest	7964.63	7560.46	7052.52	7687.62	7524.78	1548.72	7597.85	7174.62	7604.02	7964.63	7649.72	2859.54	5710.45	7027.34
7408	forest	8021.49	7652.52	7680.48	6948.36	6931.49	758.257	7220.87	6974.97	7556.3	7982.15	7801.63	2452.51	6200.01	7365.52
7409	forest	8139.9	7724.36	3385.45	7838.42	7798.32	211.755	7950.51	7380.79	8139.9	8082.07	7990.79	1742.05	1433.1	5661.23
7410	forest	7992.4	7735.22	7567.1	7778.22	7664.34	1736.83	7612.44	7235.38	7887.6	7992.4	7833.93	2101.92	4863.41	7371.07
7411	forest	7701.02	7338.9	7040.28	7474.11	6652.97	579.109	7521.96	6696.15	7142.69	7616.98	7200.35	2058.65	2762.73	7223.34
7412	forest	6713.92	6238.19	2799.2	6095.02	6008.86	322.519	5664.34	6009.54	6544.54	6713.92	6459.36	2045.01	1500.24	4750.25
7413	forest	8014.66	7663.64	7664.47	7612.63	7511.98	1099.56	7672.46	7227.1	7317.56	8014.66	7734.77	2375.92	4553.29	7166.03
7414	forest	7978.62	7522.99	5280.05	7978.62	7715.35	560.602	7726.3	7270.96	7853.73	7923.09	7712.02	1753.16	1824.52	7182.86
7415	forest	8186.01	7776.89	3445.61	8098.99	7965.3	622.578	7978.81	7540.72	8046.08	8186.01	7837.8	3003.96	2501.99	7402.02
7416	forest	8094.21	7717.07	1821.55	7743.38	7595.53	799.219	7730.34	7265.53	7891.75	8094.21	7723.58	2595.99	2479.73	7194.54
7417	forest	7921.65	7549.64	7719.42	7921.65	7771.18	368.642	7521.44	6953.31	7561.19	7591.08	7052.51	2271.71	3152.15	6559.64
7418	forest	8377.89	7730.45	8104.08	8237.54	8120.34	437.401	8075.36	7628.36	8210.97	8377.89	8054.36	3016.1	2342.86	7443.58
7419	forest	7961.23	7393.82	4291.57	7928.05	7792.79	428.153	7795.01	7337.65	7880.31	7961.23	7640.85	1820.8	1488.49	6944.97
7420	forest	7812.45	7394.97	5162.07	6272.29	6504.44	374.486	5868.97	5820.24	7305.52	7626.56	7248.38	1687.08	1247.06	7156.45
7421	forest	8083.74	7704.83	3550.96	7877.28	7617.78	607.189	7752.22	7349.79	7104.8	8083.74	7974.75	2679.9	2294.21	7304.98
7422	forest	8181.82	7834.98	2536.03	8088.35	7998.74	678.793	8012.23	7643.17	8080.45	8181.82	7904.09	3062.48	2158.74	7441.06
7423	forest	3869.63	3869.63	1525.31	3373.49	3719.85	408.436	2977.29	3278.8	3436.78	3239.91	3167.45	3064.18	3161.66	3207.68
7424	forest	7596.69	7202.44	7179.8	566.597	2210.37	493.755	6522.21	6335.43	906.176	7311.1	7258.73	5794.37	1460.04	6745.3
7425	forest	7367.88	6724.3	7367.88	7344.73	6859.15	1130.4	6227.98	6209.42	6782.71	6358.67	6337.39	6399.6	4424.87	5214.82
7426	farm	1025.54	72.2547	-468.917	324.181	-487.765	479.369	-1383.12	-1677.6	540.812	-2049.15	-1211.05	288.37	512.216	-57.768
7427	farm	712.481	-318.798	-1097.91	300.552	-1328.86	508.487	-1809.25	-1574.97	712.481	-2810.35	-1311.61	323.619	413.51	-86.018
7428	farm	959.202	0	-651.52	310.689	-236.591	511.714	-1392.62	-1504.05	481.536	-1424.31	-793.992	330.857	512.988	28.7468
7429	farm	6978.23	5318.72	5017.15	281.061	5545.42	533.336	2356.69	5824	631.767	6978.23	6237.76	348.68	932.921	2818.7
7430	farm	7772.05	4393.83	3802.37	3163.38	5824.26	1062.94	7772.05	1249.81	6204.91	377.734	1452.59	454.368	919.388	1824.87
7431	farm	7363.24	1845.43	4589.56	3028.67	6510.99	1995.72	7363.24	3462.06	4107.01	2350.79	1780.55	619.581	1106.28	1369.67
7432	farm	7300.37	2418.15	4219.12	3139.14	5946.05	4055.82	7300.37	3515.35	4902.34	2382.79	2434.14	620.337	1028.19	1575.31
7433	farm	7188.6	2517.21	4469.85	2213.78	4548.46	799.598	7188.6	1540.17	6081.86	2217.32	2414.75	540.669	1084.48	1208.67
7434	farm	7399.55	3068.68	4225.58	2674.11	3850.54	1493.66	7027.73	4993.49	7010.34	2952.58	3633.32	542.312	1093.97	1478.59
7435	farm	6265.79	3199.29	6265.79	184.207	5895.33	4909.34	4946.57	1187.83	462.433	3011.46	3501.63	674.188	960.832	364.292
7436	farm	4515.04	782.319	4515.04	248.476	3927.02	2212.68	1948.69	943.589	549.954	1128.51	829.13	516.603	1014.64	330.971
7437	farm	5442.59	3874.59	4645.34	218.305	2810.64	4132.87	5238.85	1913.36	464.979	5071.34	3345.15	445.108	1273.55	366.021
7438	farm	6989.83	6658.81	2091.55	245.505	5008.23	5079.51	6077.69	4128.46	5856.1	918.748	2827.64	6989.83	5805.11	4143.06
7439	farm	4064.69	2304	3664.63	412.646	1683.49	2690.92	4064.69	3398.31	3190.2	953.722	1462.44	915.252	1745.29	2303.96
7440	farm	6852.58	6852.58	5541.59	701.45	5118.68	4876.47	5465.25	1655.31	6016.63	5450.42	1927.21	886.622	1669.06	518.737
7441	farm	7613.54	7613.54	7196.23	1659.41	5872.02	5982.68	6416.09	5696.02	6962.54	3808.18	2146.36	865.659	1473.89	1464.76
7442	farm	7549.93	7029.2	2888.79	2389.9	5447.72	3272	5525.07	746.684	2301.1	4907.33	703.993	570.324	2819.65	541.089

CrowdSourced mapping data summary:

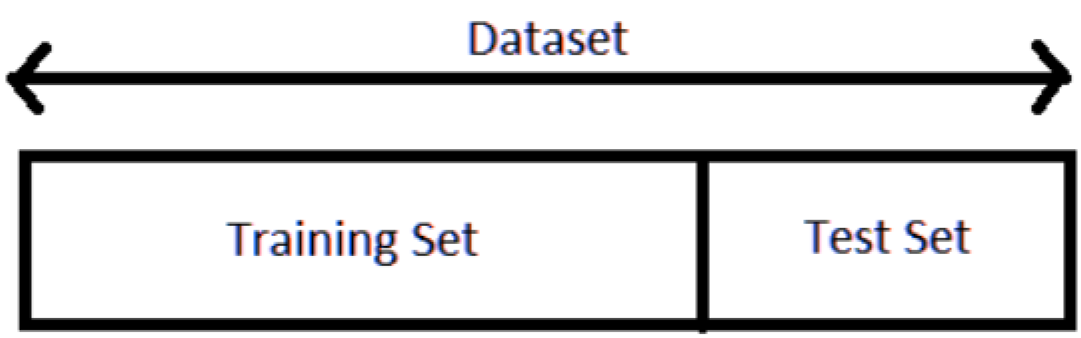
```
> summary(CrowdSourceData)
   class1        max_ndvi        Jul_20_2015      Jun_02_2015      May_17_2015      May_01_2015      Apr_15_2015
 farm  :1441   Min.   : 712.5   Min.   :-318.8   Min.   :-1098   Min.   :-633.7   Min.   :-1329   Min.   : -18.33
 forest:7431   1st Qu.:7648.5   1st Qu.:5026.4   1st Qu.: 2647   1st Qu.:1849.2   1st Qu.: 3749   1st Qu.: 491.97
               Median :7961.7   Median :7059.2   Median : 5835   Median :5236.8   Median : 6154   Median :1601.29
               Mean   :7807.3   Mean   :6085.8   Mean   : 5125   Mean   :4753.1   Mean   : 5518   Mean   :3021.11
               3rd Qu.:8157.3   3rd Qu.:7655.1   3rd Qu.: 7643   3rd Qu.:7595.0   3rd Qu.: 7597   3rd Qu.:5998.70
               Max.   :8650.5   Max.   :8377.7   Max.   : 8566   Max.   :8650.5   Max.   : 8516   Max.   :8267.12
  Mar_30_2015    Mar_14_2015    Feb_26_2015    Feb_10_2015    Jan_25_2015    Jan_09_2015      Nov_17_2014
 Min.   :-1809  Min.   :-1678  Min.   :-1157  Min.   :-2810  Min.   :-2850  Min.   :-3099.7  Min.   :-1571
 1st Qu.: 3332  1st Qu.: 1055  1st Qu.: 3271  1st Qu.: 1513  1st Qu.: 3372  1st Qu.:  596.2  1st Qu.: 1062
 Median : 6169  Median : 3220  Median : 6266  Median : 4907  Median : 6780  Median : 1277.3  Median : 2282
 Mean   : 5272  Mean   : 3511  Mean   : 5363  Mean   : 4501  Mean   : 5471  Mean   : 2301.6  Mean   : 3347
 3rd Qu.: 7385  3rd Qu.: 5861  3rd Qu.: 7528  3rd Qu.: 7309  3rd Qu.: 7629  3rd Qu.: 3447.8  3rd Qu.: 5640
 Max.   : 8499  Max.   : 8002  Max.   : 8452  Max.   : 8422  Max.   : 8401  Max.   : 8477.6  Max.   : 8625
  Nov_01_2014      Oct_16_2014      Sep_30_2014     Aug_13_2014      Jun_26_2014     Jun_10_2014
 Min.   : -86.02  Min.   : -33.84  Min.   :-456.6  Min.   :-1004.2  Min.   : 414.1  Min.   :-1520
 1st Qu.: 577.20  1st Qu.: 948.61  1st Qu.: 478.2  1st Qu.:  750.9  1st Qu.:1691.8  1st Qu.: 2563
 Median :1763.91  Median :1531.43  Median :1079.3  Median : 1343.0  Median :2934.6  Median : 5983
 Mean   :2676.34  Mean   :2799.37  Mean   :2456.5  Mean   : 1615.2  Mean   :3142.1  Mean   : 5163
 3rd Qu.:4693.16  3rd Qu.:3979.81  3rd Qu.:4447.6  3rd Qu.: 2123.0  3rd Qu.:4397.4  3rd Qu.: 7679
 Max.   :7932.69  Max.   :8630.42  Max.   :8210.2  Max.   : 5915.7  Max.   :7491.2  Max.   : 8490
  May_25_2014     May_09_2014     Apr_23_2014     Apr_07_2014       Mar_22_2014      Feb_18_2014     Feb_02_2014
 Min.   :-300.1  Min.   :-2223  Min.   :-958.4  Min.   :  -6.222  Min.   :  69.01  Min.   : 128.4  Min.   :-
2596
 1st Qu.:1461.9  1st Qu.: 1492  1st Qu.:1092.9  1st Qu.: 434.043  1st Qu.: 848.16  1st Qu.: 525.4  1st Qu.:
6297
 Median :3993.7  Median : 2809  Median :3095.3  Median :1245.665  Median :1849.66  Median :1048.8  Median :
7028
 Mean   :3832.3  Mean   : 3145  Mean   :3258.9  Mean   :2180.516  Mean   :2976.09  Mean   :2263.0  Mean   :
6637
```

```
 3rd Qu.:6054.7   3rd Qu.: 4316   3rd Qu.:5133.4   3rd Qu.:3404.637   3rd Qu.:5271.03   3rd Qu.:3612.1   3rd Qu.:
7440
 Max.   :7981.8   Max.   : 8445   Max.   :7919.1   Max.   :8206.780   Max.   :8235.40   Max.   :8247.6   Max.   :
8410
  Jan_17_2014       Jan_01_2014
 Min.   :-2139.9   Min.   :-3647.6
 1st Qu.:  675.5   1st Qu.:  734.6
 Median : 1492.1   Median : 1562.7
 Mean   : 2642.6   Mean   : 2687.7
 3rd Qu.: 4502.8   3rd Qu.: 4345.0
 Max.   : 8418.2   Max.   : 8502.0
```

Introduction to Concepts:

- **Holdout Method:**

  Hold-out is when you split up your dataset into a ‘train’ and ‘test’ set. The training set is what the model is trained on, and the test set is used to see how well that model performs on unseen data. A common split when using the hold-out method is using 80% of data for training and the remaining 20% of the data for testing.
  In our ‘Crowdsourced Mapping’ dataset We have split 2/3rd data to the training set (5900 records) and around 1/3rd data to the testing set (2972 records)



- **Bagging:**

  When training a model, no matter if we are dealing with a classification or a regression problem, we obtain a function that takes an input, returns an output and that is defined with respect to the training dataset. Due to the theoretical variance of the training dataset (we remind that a dataset is an observed sample coming from a true unknown underlying distribution), the fitted model is also subject to variability: if another dataset had been observed, we would have obtained a different model. The idea of bagging is then simple: we want to fit several independent models and “average” their predictions in order to obtain a model with a lower variance

- **Random Forest:**

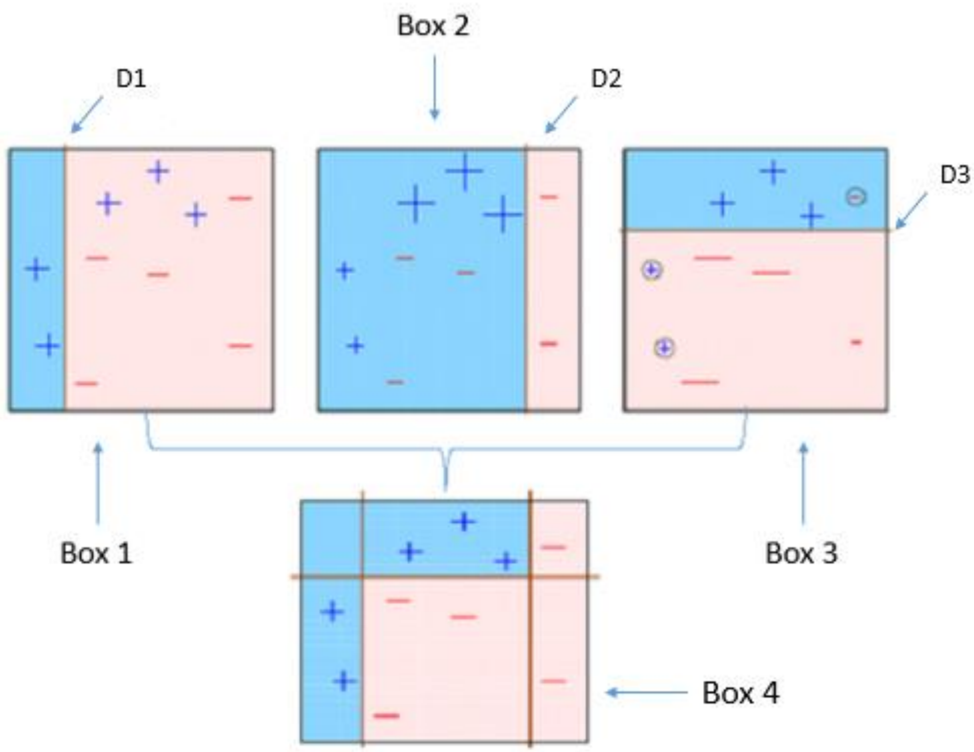
  Random Forests are an improvement over bagged decision trees. A problem with decision trees like CART is that they are greedy. They choose which variable to split on using a greedy algorithm that minimizes error. As such, even with Bagging, the decision trees can have a lot of structural similarities and in turn have high correlation in their predictions. Combining predictions from multiple models in ensembles works better if the predictions from the sub-models are uncorrelated or at best weakly correlated. Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation. It is a simple tweak. In CART, when selecting a split point, the learning algorithm is allowed to look through all variables and all variable values in order to select the most optimal split-point. The random forest algorithm changes this procedure so that the learning algorithm is limited to a random sample of features of which to search.

The number of features that can be searched at each split point (m) must be specified as a parameter to the algorithm. We can try different values and tune it using cross validation.
For classification a good default is: m = sqrt(p)
For regression a good default is: m = p/3

Where m is the number of randomly selected features that can be searched at a split point and p is the number of input variables. For example, if a dataset had 25 input variables for a classification problem, then:
m = sqrt (25)
m = 5

❖ Boosting:

The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners. Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. Boosting means that each tree is dependent on prior trees. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage. It does not involve bootstrap sampling, instead each tree is fit on a modified version of the training data set.



❖ Naïve Bayes Classifiers:

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Algorithm
Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier 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 assumption is called class conditional independence.

Likelihood   Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

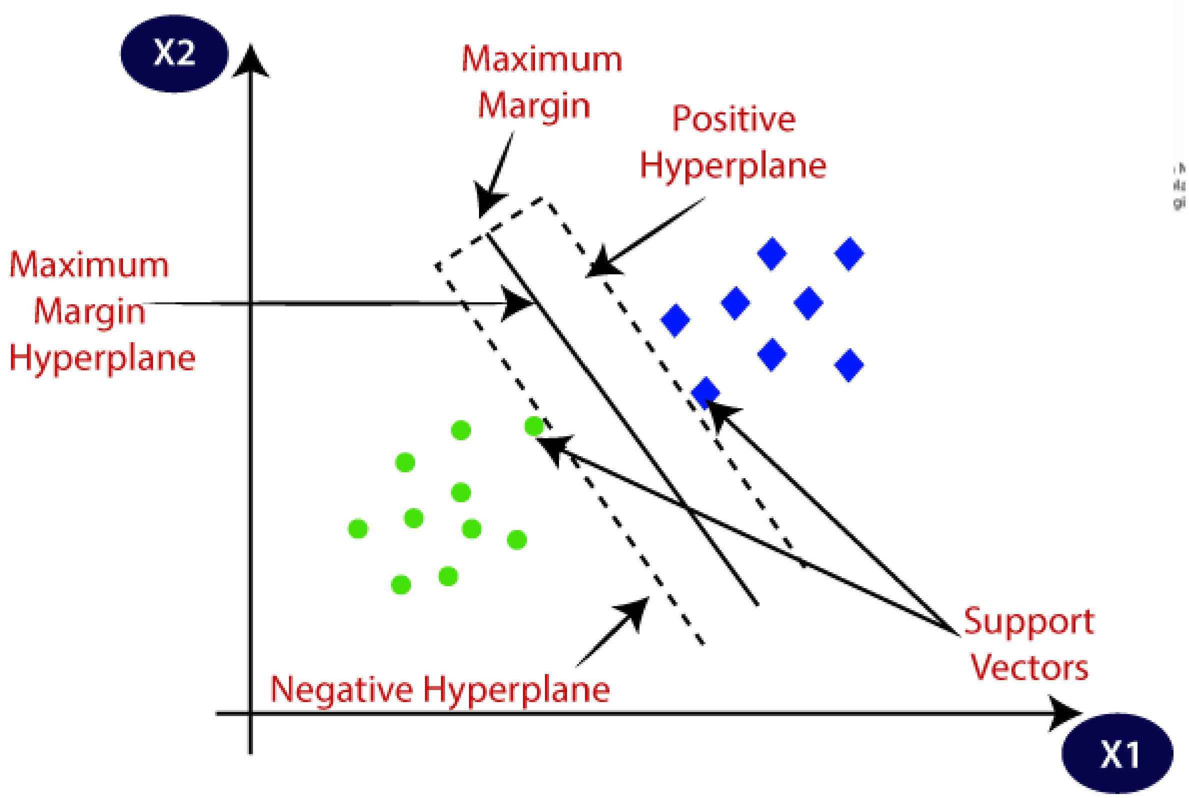
Posterior Probability   Predictor Prior Probability

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- 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.

❖ Support Vector Machines:

Support vector machine (SVM) is the popular and most important technique of classification and was developed by Vladimir Vapnik. It is based on statistical learning theory. In the classification of small datasets, SVM has yielded excellent performance that is hardly provided by any other method and able to solve practical problems such as high dimension, over learning and local minima. The standard Support vector machine algorithm is leads to a quadratic optimization problem with bound constraints and one linear equality constraint. However, when the datasets are large with large number of data points, the quadratic programming solvers become very difficult, because their time requirements and memory are highly dependent on the size of the training datasets. This is the only limitation of support vector machine. Support vector machine works on the kernel function which is used to project the data points to higher dimensions for better accuracy of classification.



SVM which is kernel based algorithms have achieved considerable success in various problems in the classification where all the training data is available in advance. Support vector machine combines the kernel trick with the large margin idea. Various kernels are used to classify the data by support vector machine such as linear kernel, sigmoid kernel, polynomial kernel, radial basis function kernel etc. Support vector machines (SVMs) and kernel methods (KMs) have become very popular as techniques for learning. New kernel expert system is also introduced by R.Zhang, W.Wang for better classification performance. These kernels basically depend on the number of support vectors. There are some kernels present in the literature those are independent of number of support vectors namely: intersection kernel, chi-squared kernel, additive kernel. SVMs (Support Vector Machines) are the efficient technique for data classification and prediction. It works on the principle of supervised learning. As we discussed that kernel function plays an important role in the classification by support vector machine.

Kernels are used to project the data points in the higher dimensions for better classification of the datasets as shown in fig.1. Some kernel functions are present in support vector machine algorithm are based on neural networks. Support vector machine is considered easier to use than neural networks but time taken by support vector machine is more compared to neural network [2]. The radial basis kernel, polynomial kernel and sigmoid kernel of support vector machine is used for non-linear separation and works on the principle of neural networks.

Kernels:

Kernel function is used to project the data points to higher dimensional space for better classification. There are various kernels which are used to improve the performance of classification by support vector machine namely: linear kernel, Radial basis function kernel, polynomial kernel. Kernel is used to project the data point to higher dimensional space to improve its ability to find best hyperplane to separate the data points of different classes. The kernel function used are described below:

1. **Linear Kernel:** Linear kernel separates the data points linearly by using straight line. Linear kernel is good at classifying two classes at a time. Mapping of data points to higher dimension is not required.

2. **Polynomial Kernel:** The polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that is similar to vectors (training samples) in a feature space over polynomials of the original variables. It generally works with non-linearly separable data

3. **Radial basis function kernel:** The radial basis function network is an artificial neural network that uses radial basis functions as activation functions. It mainly used to classify non-linear data. Radial basis function networks have many uses, including time series prediction, function approximation, classification and system control.

The art is to choose a model with optimum variance and bias. Therefore, you need to choose the values of C and **Gamma** accordingly. For SVM, a High value of Gamma leads to more accuracy but biased results and vice-versa. Similarly, a large value of **Cost** parameter (C) indicates poor accuracy but low bias and vice-versa.

The SVM in particular defines the criterion to be looking for a decision surface that is maximally far away from any data point. This distance from the decision surface to the closest data point determines the **margin** of the classifier.

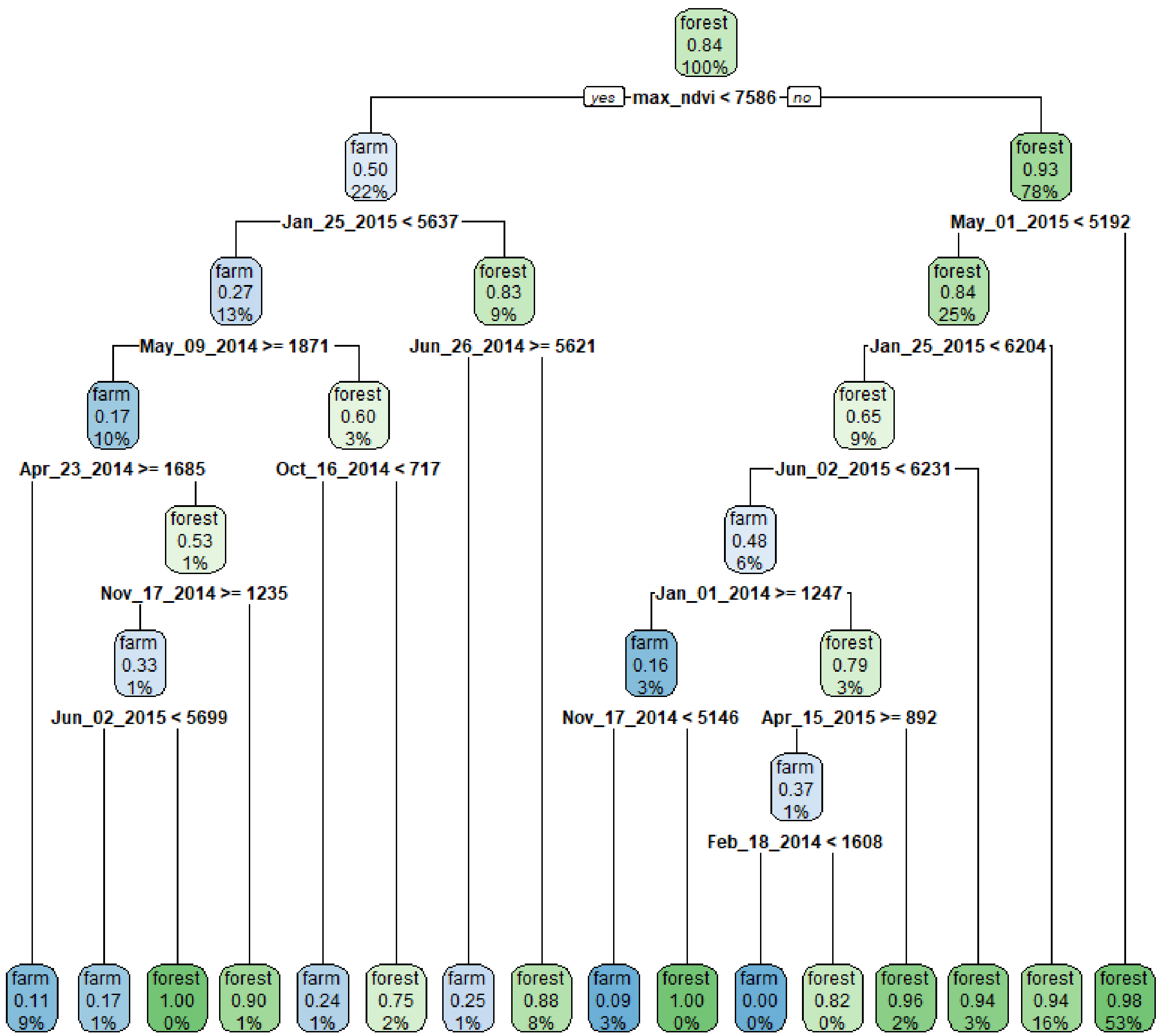
A **hyperplane** in an n-dimensional Euclidean space is a flat, n-1 dimensional subset of that space that divides the space into two disconnected parts

3. Decision Tree Using Hold-Out Method

Dim results :

> dim(CrowdSourceData)	> dim(CrowdSourceData.train)	> dim(CrowdSourceData.test)
[1] 8872 29	[1] 5900 29	[1] 2972 29

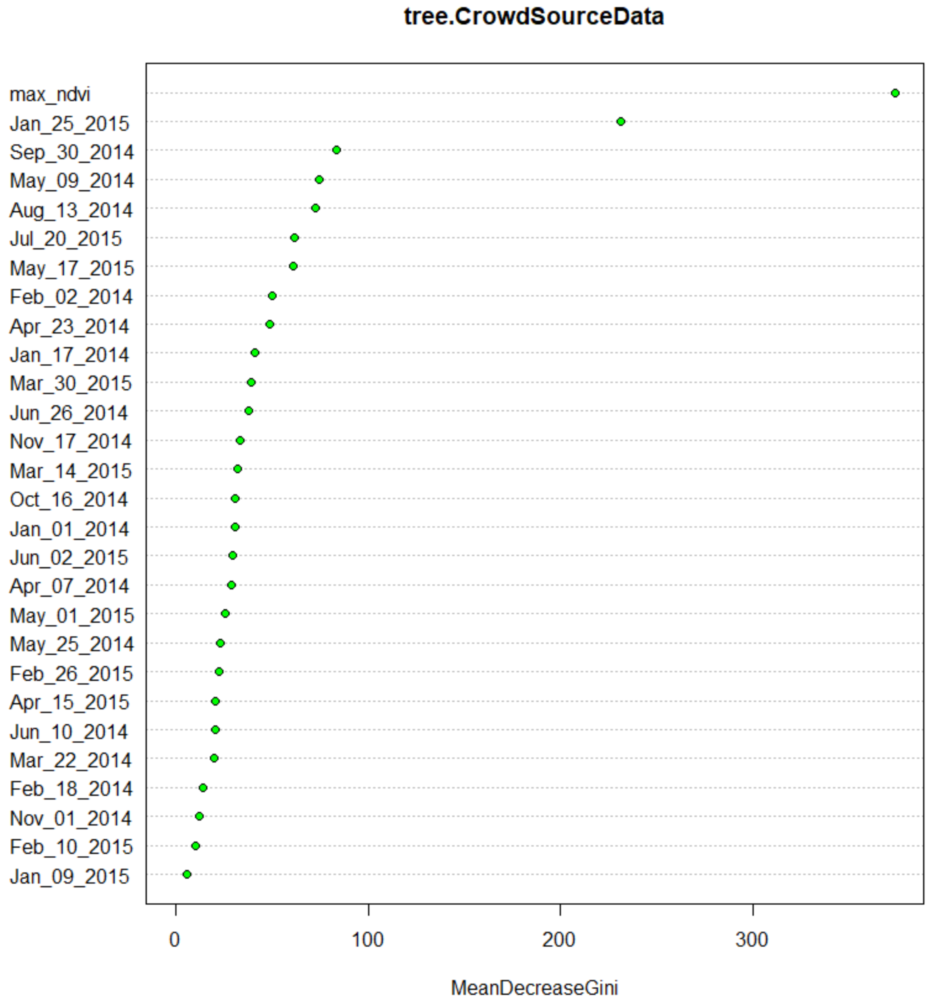
Tree plot :



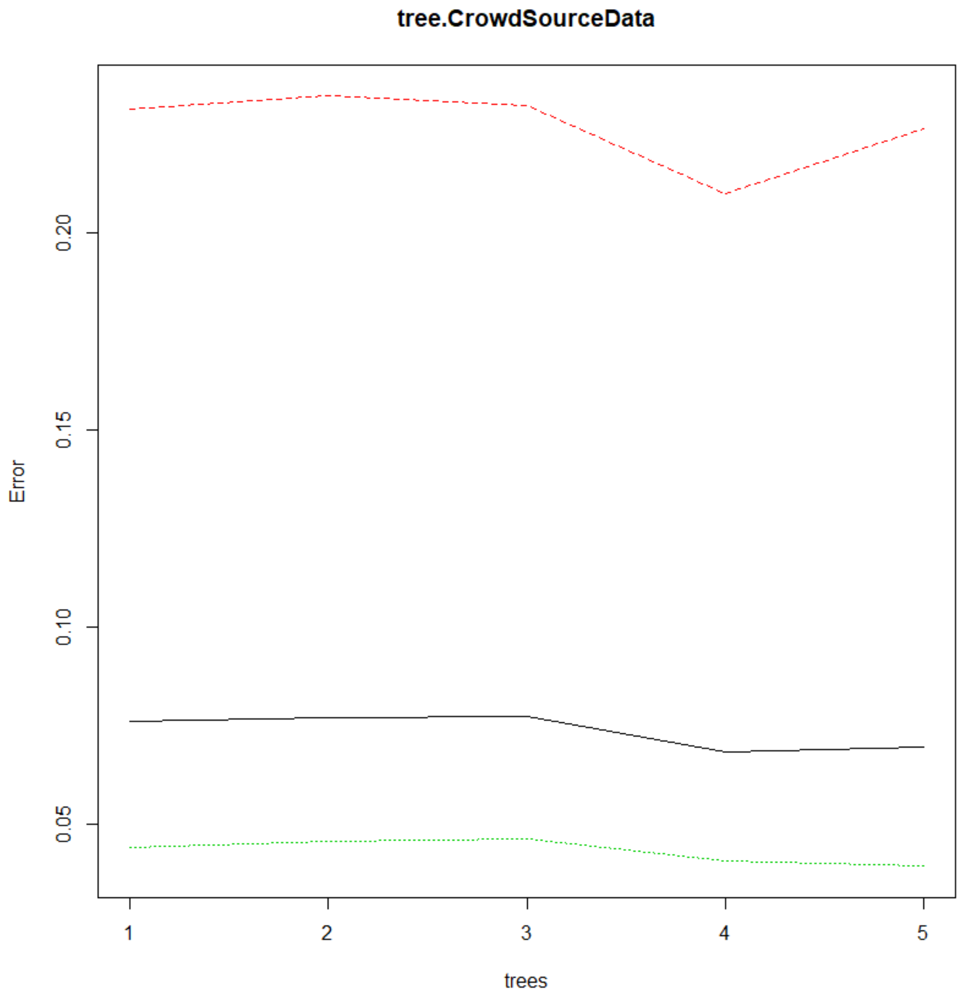
4. Decision Tree Using Bagging Method

<u>Our Observations:</u>

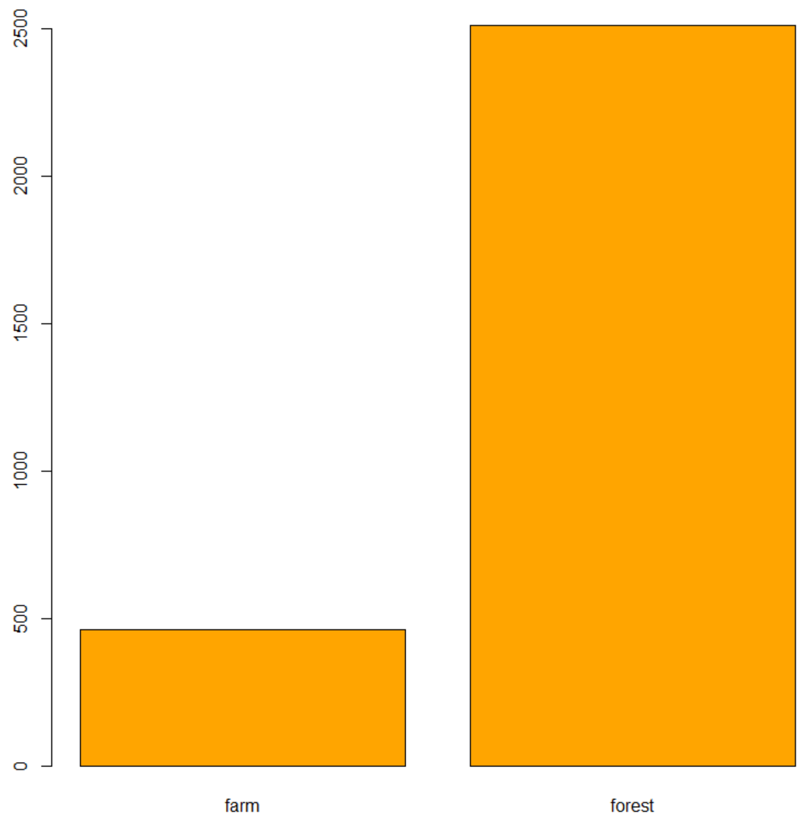
```
> table(tree.pred,class1.test)
              class1.test
    tree.pred farm forest
       farm    413     48
       forest   83   2428
> mean(tree.pred!=class1.test)
    [1] 0.04407806

> varImpPlot(tree.CrowdSourceData,bg="green")
```



```
> plot(tree.CrowdSourceData) #train data
```



```
> plot(tree.pred, col="orange") #test data
```



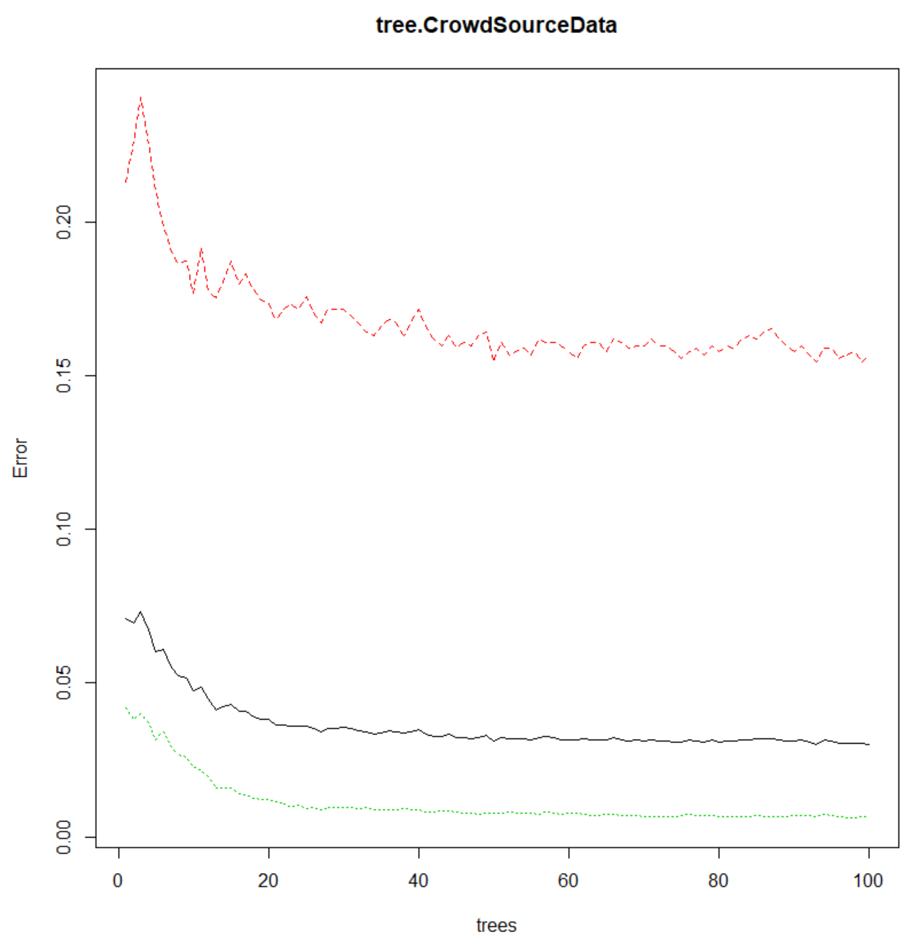
5. Result Decision Tree Using RandomForest Method

We have 28 predictors and 1 response variable. To decide on mtry value we can do “sqrt(28) “ which is 5.29

<u>Our Observations:</u>

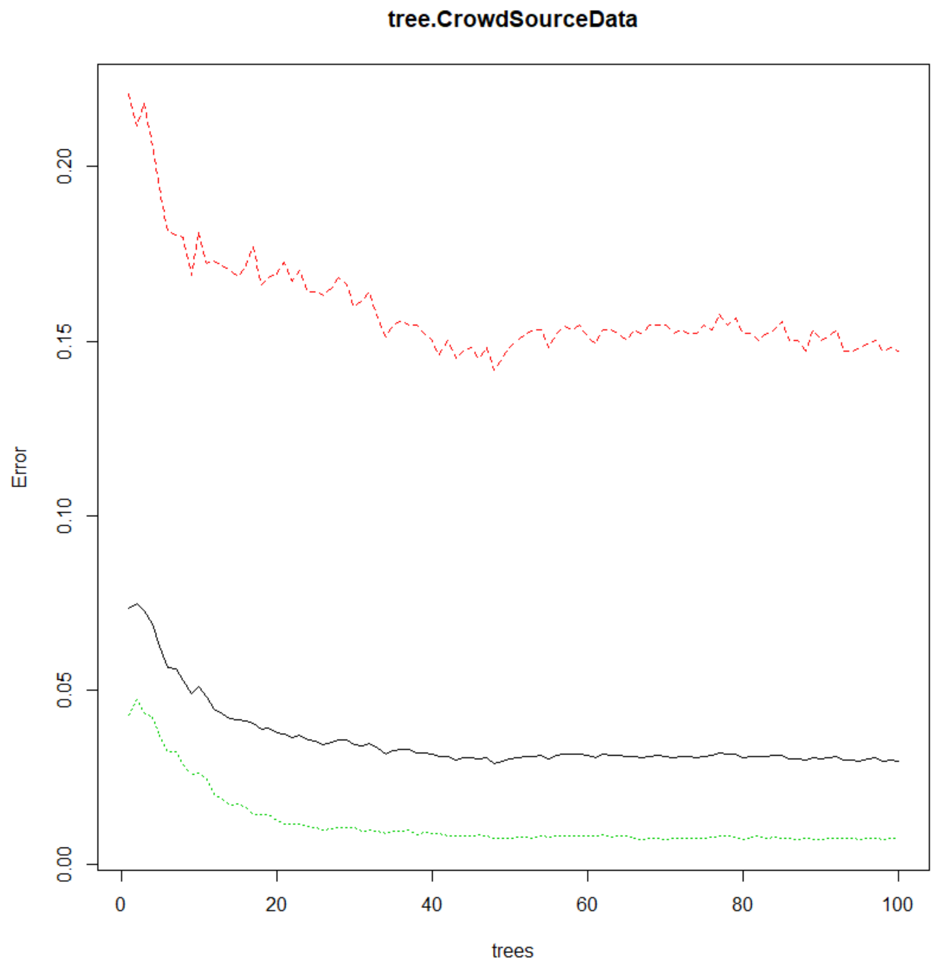
#Randomforest ntree=100, mtry=5

```
> table(tree.pred,class1.test)
         class1.test
tree.pred farm forest
   farm    431     10
   forest   65   2466
> mean(tree.pred!=class1.test)
[1] 0.02523553
```



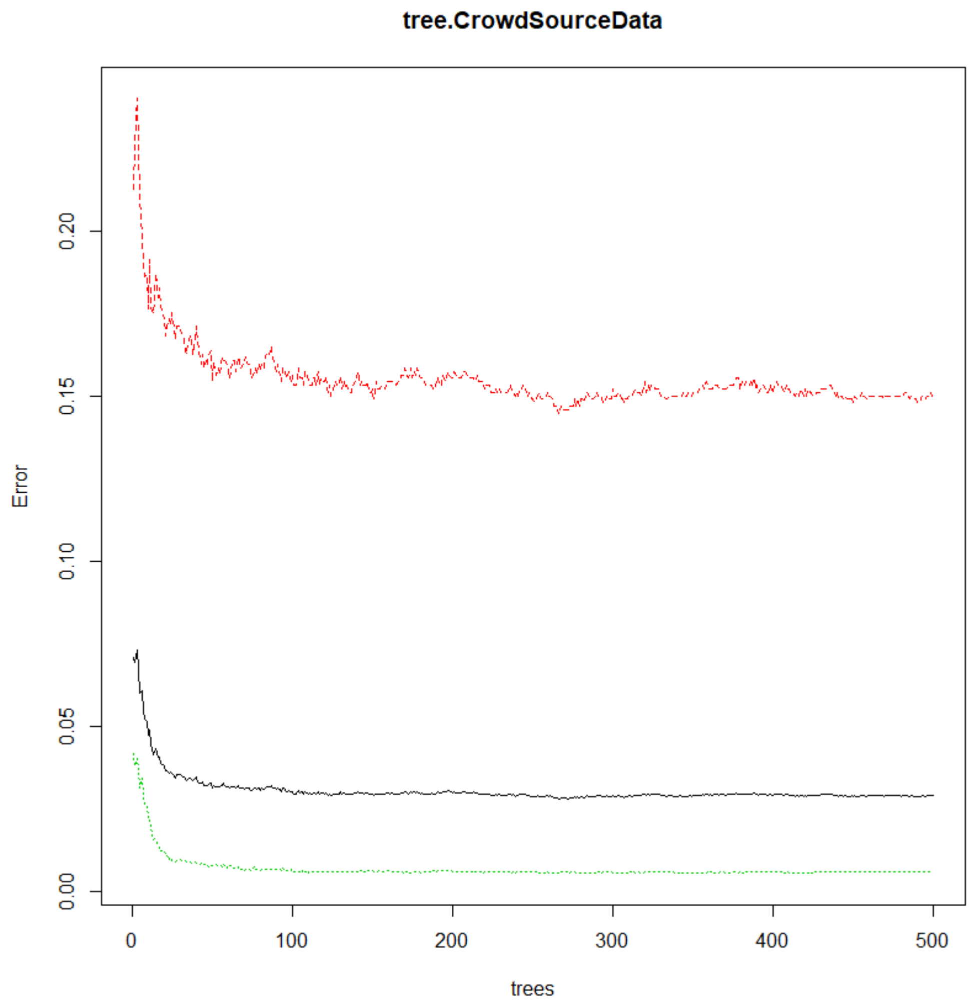
#Randomforest ntree=100, mtry=6

```
> table(tree.pred,class1.test)
         class1.test
tree.pred farm forest
   farm    435     14
   forest   61   2462
> mean(tree.pred!=class1.test)
[1] 0.02523553
```



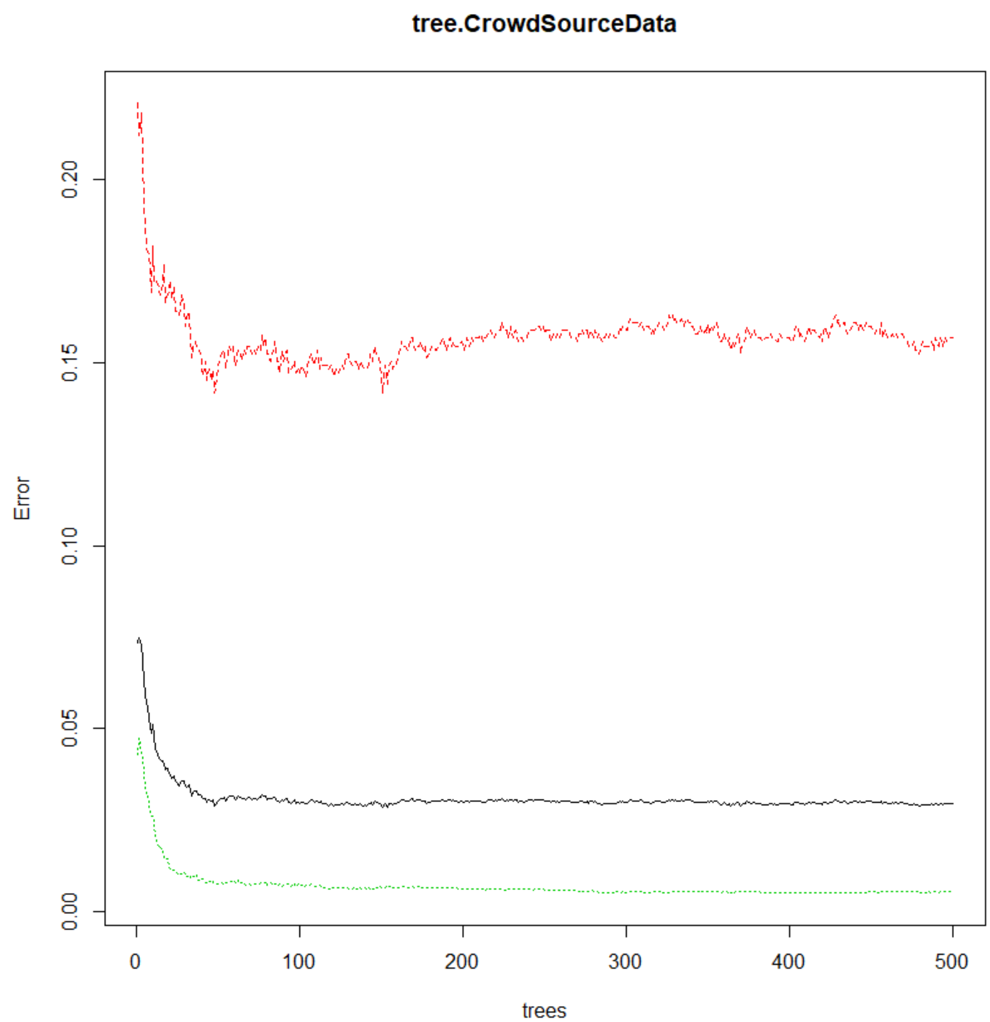
#Randomforest ntree=500, mtry=5

```
> table(tree.pred,class1.test)
         class1.test
tree.pred farm forest
   farm    439     12
   forest   57   2464
> mean(tree.pred!=class1.test)
[1] 0.02321669
```



#Randomforest ntree=500, mtry=6

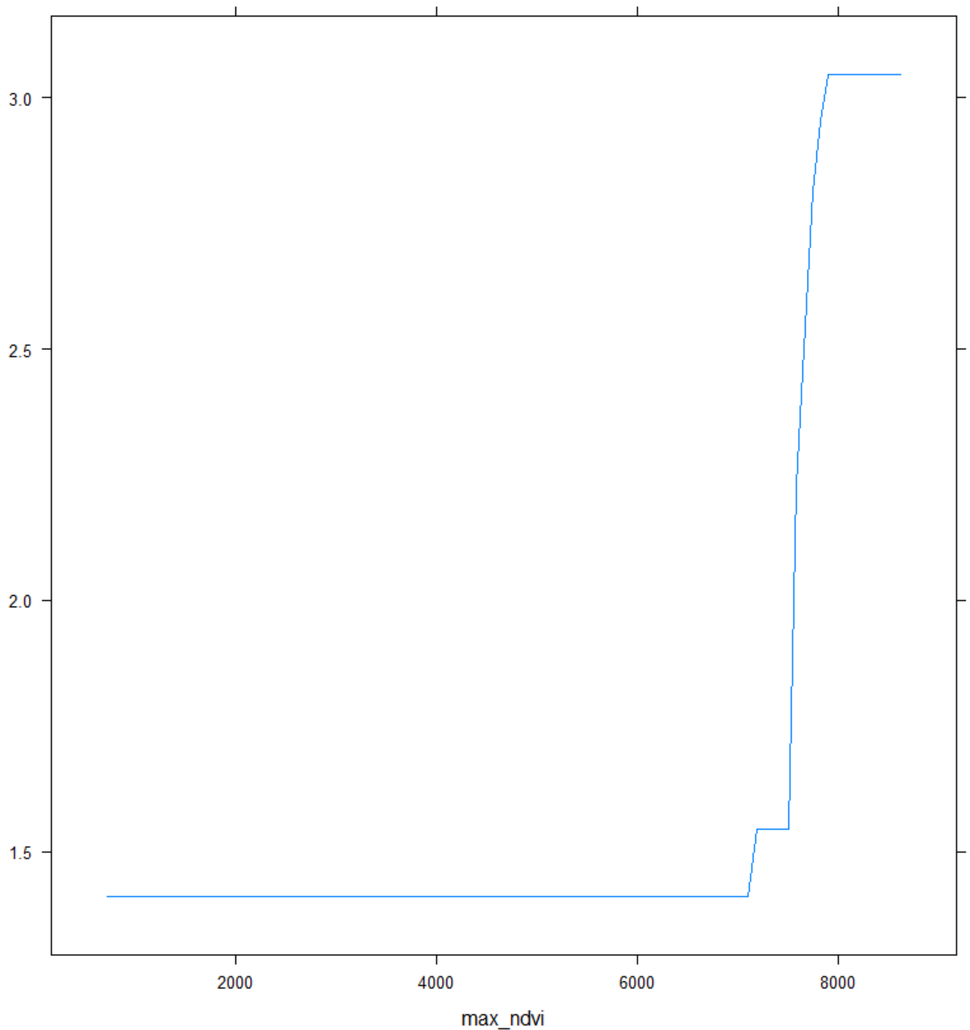
```
> table(tree.pred,class1.test)
         class1.test
tree.pred farm forest
   farm    438     12
   forest   58   2464
> mean(tree.pred!=class1.test)
[1] 0.02355316
```


tree.CrowdSourceData
Error
0.00
0.05
0.10
0.15
0.20
0
100
200
300
400
500
trees

6. Result Decision Tree Using Boosting Method

<u>Our Observations:</u>

```
> mean(tree.pred!=classtype.test)
[1] 1
```



7. Naive-Bayes Classification

Our Observations:

```
> Naive_Bayes_Model=naiveBayes(CrowdSourceData$class1~., CrowdSourceData)
> Naive_Bayes_Model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
     farm    forest
0.1624211 0.8375789

Conditional probabilities:
        max_ndvi
Y            [,1]     [,2]
  farm   7161.091 812.5605
  forest 7932.653 419.0747

        Jul_20_2015
Y            [,1]     [,2]
  farm   4805.204 1779.533
  forest 6334.154 2038.927

        Jun_02_2015
Y            [,1]     [,2]
  farm   4151.646 1796.775
  forest 5313.166 2769.619

        May_17_2015
Y            [,1]     [,2]
  farm   3309.915 1971.376
  forest 5032.994 2865.194

        May_01_2015
Y            [,1]     [,2]
  farm   4289.054 1687.096
  forest 5755.728 2365.285

        Apr_15_2015
Y            [,1]     [,2]
  farm   2071.011 1834.947
  forest 3205.350 2917.733

        Mar_30_2015
Y            [,1]     [,2]
  farm   5187.149 1871.608
  forest 5287.956 2557.845

        Mar_14_2015
Y            [,1]     [,2]
  farm   4365.980 2220.935
  forest 3344.742 2492.792

        Feb_26_2015
Y            [,1]     [,2]
  farm   5093.113 2345.456
  forest 5415.573 2520.883

        Feb_10_2015
Y            [,1]     [,2]
  farm   3397.546 2167.514
```

```
  forest 4714.970 2839.372

       Jan_25_2015
Y             [,1]     [,2]
  farm   3296.428 2230.865
  forest 5892.801 2546.437

       Jan_09_2015
Y             [,1]     [,2]
  farm   1636.525 1587.064
  forest 2430.550 2313.866

       Nov_17_2014
Y             [,1]     [,2]
  farm   2822.005 1641.397
  forest 3448.432 2833.984

       Nov_01_2014
Y             [,1]     [,2]
  farm   2499.162 1804.092
  forest 2710.702 2414.547

       Oct_16_2014
Y             [,1]     [,2]
  farm   1885.803 1616.726
  forest 2976.522 2596.731

       Sep_30_2014
Y             [,1]     [,2]
  farm   2256.107 2222.723
  forest 2495.341 2555.368

       Aug_13_2014
Y             [,1]      [,2]
  farm   1371.424  784.0479
  forest 1662.515 1096.1095

       Jun_26_2014
Y             [,1]     [,2]
  farm   3726.861 1484.727
  forest 3028.646 1689.219

       Jun_10_2014
Y            [,1]     [,2]
  farm   4084.20 1930.509
  forest 5372.04 2758.011

       May_25_2014
Y             [,1]     [,2]
  farm   3501.464 1669.008
  forest 3896.409 2451.056

       May_09_2014
Y             [,1]     [,2]
  farm   3870.923 1649.472
  forest 3004.245 2128.288

       Apr_23_2014
Y             [,1]    [,2]
  farm   3832.743 1449.75
  forest 3147.677 2294.23

       Apr_07_2014
Y             [,1]     [,2]
  farm   2155.215 1613.663
  forest 2185.422 2210.453

       Mar_22_2014
Y             [,1]     [,2]
  farm   2134.270 1870.340
  forest 3139.338 2558.969
```

```
        Feb_18_2014
Y            [,1]     [,2]
  farm   1200.002 1215.704
  forest 2469.085 2419.044

        Feb_02_2014
Y            [,1]      [,2]
  farm   5399.701 1931.6035
  forest 6877.256  940.4063

        Jan_17_2014
Y            [,1]     [,2]
  farm   1404.955 1357.355
  forest 2882.556 2505.525

        Jan_01_2014
Y            [,1]     [,2]
  farm   2751.494 2120.329
  forest 2675.386 2487.872

> NB_Predictions=predict(Naive_Bayes_Model,CrowdSourceData)
> table(NB_Predictions,CrowdSourceData$class1)

NB_Predictions farm forest
        farm   1225    619
        forest  216   6812
> mean(NB_Predictions!=CrowdSourceData$class1)
[1] 0.09411632
>
> RNGkind(sample.kind = "Rounding")
Warning message:
In RNGkind(sample.kind = "Rounding") : non-uniform 'Rounding' sampler used
> set.seed(123)
> train=sample(1:nrow(CrowdSourceData),5900)
> trainSet=CrowdSourceData[train,]
> testSet=CrowdSourceData[-train,]
>
> class1.test=CrowdSourceData$class1[-train]
> NB_2=naiveBayes(class1~.,trainSet)
> NB_Predictions_2=predict(NB_2,testSet)
> table(NB_Predictions_2,class1.test)
                class1.test
NB_Predictions_2 farm forest
          farm    417    205
          forest   79   2271
> mean(NB_Predictions_2!=class1.test)
[1] 0.09555855
```

8. Result of support vector machine using liner kernel with different costs

Our Observations:

```
> svmfit=svm(class1~., data=trainSet, kernel="linear", cost=0.1, scale=TRUE)
> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    359     70
     forest  137   2406
> mean(Class1_pred!=testSet$class1)
[1] 0.06965007

> #SVM on train data linear cost =0.01
> svmfit=svm(class1~., data=trainSet, kernel="linear", cost=0.01, scale=TRUE)
> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    347     65
     forest  149   2411
> mean(Class1_pred!=testSet$class1)
[1] 0.07200538

> #SVM on train data linear cost = 1
> svmfit=svm(class1~., data=trainSet, kernel="linear", cost=1, scale=TRUE)
> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    363     74
     forest  133   2402
> mean(Class1_pred!=testSet$class1)
[1] 0.06965007

> tune.out=tune(svm,class1~., data=trainSet, kernel="linear", ranges=list(cost=c(0.001
,0.01,0.1,1,5,10,100)),scale=TRUE)

WARNING: reaching max number of iterations

WARNING: reaching max number of iterations
> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
    5

- best performance: 0.06559322

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.09067797 0.01584441
2 1e-02 0.06610169 0.01308004
3 1e-01 0.06610169 0.01390795
4 1e+00 0.06593220 0.01347549
5 5e+00 0.06559322 0.01317850
6 1e+01 0.06576271 0.01329305
7 1e+02 0.06559322 0.01365433

> #best model
```

```
> bestmod=tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = class1 ~ ., data = trainSet, ranges = list(cost = c(
0.001,
    0.01, 0.1, 1, 5, 10, 100)), kernel = "linear", scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  5

Number of Support Vectors:  1044

 ( 526 518 )


Number of Classes:  2

Levels:
 farm forest



>
> #SVM on test data  linear
> Class1_pred=predict(bestmod,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    363     74
     forest  133   2402
> mean(Class1_pred!=testSet$class1)
[1] 0.06965007
```

9. Result of support vector machine using radial kernel with different costs and gammas

<u>Our Observations:</u>

```
#SVM on test data radial cost 0.1

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm      0      0
     forest  496   2476
> mean(Class1_pred!=testSet$class1)
[1] 0.166891

#SVM on test data radial cost 1

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm      0      0
     forest  496   2476
> mean(Class1_pred!=testSet$class1)
[1] 0.166891

#SVM on test data radial cost 10

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm      8      0
     forest  488   2476
> mean(Class1_pred!=testSet$class1)
[1] 0.1641992


 > tune.out=tune(svm, class1~., data=trainSet, kernel="radial",
 +               ranges=list(cost=c(0.1,1,10,100,1000), gamma=c(0.5,1,2,3,4)))
 > summary(tune.out)

 Parameter tuning of 'svm':

 - sampling method: 10-fold cross validation

 - best parameters:
  cost gamma
    10   0.5

 - best performance: 0.1276271

 - Detailed performance results:
    cost gamma     error dispersion
 1  1e-01   0.5 0.1601695 0.01188457
 2  1e+00   0.5 0.1377966 0.01101478
 3  1e+01   0.5 0.1276271 0.01032678
 4  1e+02   0.5 0.1276271 0.01032678
 5  1e+03   0.5 0.1276271 0.01032678
 6  1e-01   1.0 0.1601695 0.01188457
 7  1e+00   1.0 0.1598305 0.01177122
 8  1e+01   1.0 0.1567797 0.01193816
 9  1e+02   1.0 0.1567797 0.01193816
 10 1e+03   1.0 0.1567797 0.01193816
 11 1e-01   2.0 0.1601695 0.01188457
```

```
12 1e+00   2.0 0.1601695 0.01188457
13 1e+01   2.0 0.1601695 0.01188457
14 1e+02   2.0 0.1601695 0.01188457
15 1e+03   2.0 0.1601695 0.01188457
16 1e-01   3.0 0.1601695 0.01188457
17 1e+00   3.0 0.1601695 0.01188457
18 1e+01   3.0 0.1601695 0.01188457
19 1e+02   3.0 0.1601695 0.01188457
20 1e+03   3.0 0.1601695 0.01188457
21 1e-01   4.0 0.1601695 0.01188457
22 1e+00   4.0 0.1601695 0.01188457
23 1e+01   4.0 0.1601695 0.01188457
24 1e+02   4.0 0.1601695 0.01188457
25 1e+03   4.0 0.1601695 0.01188457

> bestmod=tune.out$best.model
> bestmode <- tune.out$best.model
> summary(bestmod)

Call:
best.tune(method = svm, train.x = class1 ~ ., data = trainSet,
ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)),
kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  5811

 ( 4874 937 )


Number of Classes:  2

Levels:
 farm forest



> #Test performance
> class1_pred=predict(tune.out$best.model,testSet)
> table(class1_pred,testSet$class1)

class1_pred farm forest
     farm    108      0
     forest  388   2476
> mean(class1_pred!=testSet$class1)
[1] 0.1305518
```

10. Result of support vector machine using Polynomial kernel with different costs and gammas

Our Observations:

```
#SVM on test data polynomial degree 2 & cost 0.01

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm     42      4
     forest  447   2479
> mean(Class1_pred!=testSet$class1)
[1] 0.1517497

#SVM on test data polynomial degree 2 & cost 0.1

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    308     19
     forest  181   2464
> mean(Class1_pred!=testSet$class1)
[1] 0.06729475

#SVM on test data polynomial degree 2 & cost 1

> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    426     34
     forest   63   2449
> mean(Class1_pred!=testSet$class1)
[1] 0.03263795


> tune.out=tune(svm, class1~., data=trainSet, kernel="polynomial",
+               ranges=list(degree=c(0.1,0.5,1,1.5,2),ranges=list(cost=c(0.01,0.1,1,10
,100))))

> summary(tune.out)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 degree                            ranges
      2 1e-02, 1e-01, 1e+00, 1e+01, 1e+02

- best performance: 0.0259322

- Detailed performance results:
  degree                            ranges      error  dispersion
1    0.1 1e-02, 1e-01, 1e+00, 1e+01, 1e+02 0.16135593 0.015197635
2    0.5 1e-02, 1e-01, 1e+00, 1e+01, 1e+02 0.16135593 0.015197635
3    1.0 1e-02, 1e-01, 1e+00, 1e+01, 1e+02 0.06288136 0.009536147
4    1.5 1e-02, 1e-01, 1e+00, 1e+01, 1e+02 0.06288136 0.009536147
5    2.0 1e-02, 1e-01, 1e+00, 1e+01, 1e+02 0.02593220 0.006639320

> bestmod=tune.out$best.model
> summary(bestmod)
```

```
Call:
best.tune(method = svm, train.x = class1 ~ ., data = trainSet, ranges = list(degree =
c(0.1, 0.5, 1, 1.5, 2), ranges = list(cost = c(0.01, 0.1, 1, 10, 100))), kernel = "pol
ynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  2
     coef.0:  0

Number of Support Vectors:  870

 ( 463 407 )


Number of Classes:  2

Levels:
 farm forest



>
> Class1_pred=predict(svmfit,testSet)
> table(Class1_pred,testSet$class1)

Class1_pred farm forest
     farm    426     34
     forest   63   2449
> mean(Class1_pred!=testSet$class1)
[1] 0.03263795
```

11. Potential performance issues

The Crowdsourced dataset is a dataset that is derived from satellite imagery. As mentioned, it has noise in them because of cloud cover in the images and inaccurate labeling/digitizing of polygons. This might cause some performance issues when doing data classification and accuracy may vary. This can be corrected by two ways by taking sample of data from various sources and reducing noise in them with help of some noise filtration algorithms.

12. Possible Future Study

This project can be extended to do some future research and analysis in couple of areas. We can incorporate other similar image datasets possibly with less cloud contamination from different continent or areas. In addition to that we can also incorporate datasets not only obtained by satellites but also by other sources like radars and sonars. By combining and doing analysis on both of them we can vastly improve accuracy and have better predictions. We can also extend the project by using other data classification algorithms and evaluating overall the accuracy and prediction levels.

13. Comparison of multiple classification techniques

- We have taken records for two classes “forest” and “farm” with 8872 records.
- We see minimum error for decision tree techniques like Random Forest, SVM (Kernel=Polynomial) and Bagging. Random Forest with 500 trees gives better result than 100 trees generation for “mtry” value of 5.
- Dataset takes time for SVM techniques (linear, radial) but SVM technique with polynomial gives faster result and the error rate is less than linear and radial.
- Naïve Bayes error rate is less than SVM with radial kernel.

Technique	Mean	Cost	Gamma	Degree
Bagging	0.04407	NA	NA	NA
Random Forest (ntree = 100, mtry = 5)	0.02523	NA	NA	NA
Random Forest (ntree = 100, mtry = 6)	0.02523	NA	NA	NA
Random Forest (ntree = 500, mtry = 5)	0.02321	NA	NA	NA
Random Forest (ntree = 500, mtry = 6)	0.02355	NA	NA	NA
Boosting	1	NA	NA	NA
Naïve Bayes classifier	0.09555	NA	NA	NA
support vector machine using linear kernel (Cost = 0.01)	0.0720	0.01	NA	NA
support vector machine using linear kernel (Cost = 0.1)	0.06965	0.1	NA	NA
support vector machine using linear kernel	0.06965	1	NA	NA

(Cost = 1)				
support vector machine using radial kernel (Cost = 0.01)	0.1668	0.1	1	NA
support vector machine using radial kernel (Cost = 0.1)	0.1668	1	1	NA
support vector machine using radial kernel (Cost = 1)	0.1641	10	1	NA
support vector machine using polynomial kernel (Cost = 0.01)	0.1517	0.01	NA	2
support vector machine using polynomial kernel (Cost = 0.1)	0.0672	0.1	NA	2
support vector machine using polynomial kernel (Cost = 1)	0.0326	1	NA	2

14. Conclusion of the project

We evaluated the potential for rapid and automated land use land cover using the given dataset. We used various data classification algorithms like Naïve Bayes (NB), random forest (RF), Support Vector Machine (SVM) and others to calculate accuracy and for better prediction. This will further help for automated and rapid mapping of lands that fall in classes of forest and farm areas.

15. References

1. Textbook Introduction to Data Mining, 2nd Edition, by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar, ISBN 978-0133128901, Pearson, 2019
2. http://archive.ics.uci.edu/ml/datasets/Crowdsourced+Mapping
3. https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/
4. https://www.saedsayad.com/naive_bayesian.htm
5. https://www.ijraset.com/fileserve.php?FID=3040

16. Appendix of R source codes

```
#Dataset: CrowdSourced Mapping Dataset

library(tree)
library(rpart)
library(rpart.plot)

FilePath="C://Users//sayal//OneDrive//Desktop//Data Mining//DM Project//Crowdsourced Mapping//dataset.csv"
CrowdSourceData=read.csv(FilePath, header=T, na.strings ="?")

ColumnName <- c("class1",  "max_ndvi", "Jul_20_2015",       "Jun_02_2015","May_17_2015",
      "May_01_2015"           ,"Apr_15_2015","Mar_30_2015",          "Mar_14_2015",           "Feb_26_2015",
      "Feb_10_2015",            "Jan_25_2015","Jan_09_2015","Nov_17_2014",           "Nov_01_2014",
      "Oct_16_2014","Sep_30_2014", "Aug_13_2014", "Jun_26_2014", "Jun_10_2014","May_25_2014",
"May_09_2014", "Apr_23_2014", "Apr_07_2014", "Mar_22_2014", "Feb_18_2014", "Feb_02_2014",
"Jan_17_2014", "Jan_01_2014")
names(CrowdSourceData) <- ColumnName

summary(CrowdSourceData)
CrowdSourceData=na.omit(CrowdSourceData)
dim(CrowdSourceData)
fix(CrowdSourceData)

#Holdout Method
RNGkind(sample.kind = "Rounding")
set.seed(123)
train=sample(1:nrow(CrowdSourceData), 5900)
CrowdSourceData.train = CrowdSourceData[train,]
CrowdSourceData.test = CrowdSourceData[-train,]
dim(CrowdSourceData.train)
dim(CrowdSourceData.test)

#decision tree on train set
RNGkind(sample.kind = "Rounding")
set.seed(123)
tree.CrowdSourceData=rpart(class1~., data=CrowdSourceData.train, method = 'class')
rpart.plot(tree.CrowdSourceData)
summary(tree.CrowdSourceData)


#bagging
RNGkind(sample.kind = "Rounding")
set.seed(123)
library(randomForest)
tree.CrowdSourceData=randomForest(class1~.,CrowdSourceData.train, ntree=5,mtry=28)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test,type="class")
class1.test=CrowdSourceData$class1[-train]
table(tree.pred,class1.test)
```

```
mean(tree.pred!=class1.test)
importance(tree.CrowdSourceData)
varImpPlot(tree.CrowdSourceData, bg="green")
plot(tree.CrowdSourceData)
plot(tree.pred, col="orange")


sqrt(28)
#Randomforest ntree=100, mtry=5
RNGkind(sample.kind = "Rounding")
set.seed(123)
tree.CrowdSourceData=randomForest(class1~.,CrowdSourceData.train, ntree=100,mtry=5)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test,type="class")
table(tree.pred,class1.test)
mean(tree.pred!=class1.test)
plot(tree.CrowdSourceData)

#Randomforest ntree=100, mtry=6
RNGkind(sample.kind = "Rounding")
set.seed(123)
tree.CrowdSourceData=randomForest(class1~.,CrowdSourceData.train, ntree=100,mtry=6)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test,type="class")
table(tree.pred,class1.test)
mean(tree.pred!=class1.test)
plot(tree.CrowdSourceData)

#Randomforest ntree=500, mtry=5
RNGkind(sample.kind = "Rounding")
set.seed(123)
tree.CrowdSourceData=randomForest(class1~.,CrowdSourceData.train, ntree=500,mtry=5)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test,type="class")
table(tree.pred,class1.test)
mean(tree.pred!=class1.test)
plot(tree.CrowdSourceData)

#Randomforest ntree=500, mtry=6
RNGkind(sample.kind = "Rounding")
set.seed(123)
tree.CrowdSourceData=randomForest(class1~.,CrowdSourceData.train, ntree=500,mtry=6)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test,type="class")
table(tree.pred,class1.test)
mean(tree.pred!=class1.test)
plot(tree.CrowdSourceData)

#Boosting
library(gbm)
RNGkind(sample.kind = "Rounding")
set.seed(123)
#Adding another column for boosting
classtype=ifelse(CrowdSourceData$class1=="farm","No","Yes")
CrowdSourceData=data.frame(CrowdSourceData,classtype)
CrowdSourceData$classtype=ifelse(CrowdSourceData$classtype=="Yes",1,0)
CrowdSourceData=CrowdSourceData[,-1]
```

```
train=sample(1:nrow(CrowdSourceData), 5900)
CrowdSourceData.train = CrowdSourceData[train,]
CrowdSourceData.test = CrowdSourceData[-train,]
classtype.test=CrowdSourceData$classtype[-train]
dim(CrowdSourceData.train)
dim(CrowdSourceData.test)

tree.CrowdSourceData=gbm(classtype~., CrowdSourceData.train, distribution="bernoulli",n.trees=100)
tree.pred=predict(tree.CrowdSourceData,CrowdSourceData.test, n.trees=100, type="response")
mean(tree.pred!=classtype.test)
plot(tree.CrowdSourceData)


#Naive bayes
CrowdSourceData=read.csv(FilePath, header=T, na.strings ="?")
ColumnName <- c("class1",  "max_ndvi", "Jul_20_2015",       "Jun_02_2015","May_17_2015",
    "May_01_2015"         ,"Apr_15_2015","Mar_30_2015",       "Mar_14_2015",       "Feb_26_2015",
    "Feb_10_2015",         "Jan_25_2015", "Jan_09_2015", "Nov_17_2014",       "Nov_01_2014",
    "Oct_16_2014","Sep_30_2014", "Aug_13_2014", "Jun_26_2014", "Jun_10_2014","May_25_2014",
"May_09_2014", "Apr_23_2014", "Apr_07_2014", "Mar_22_2014", "Feb_18_2014", "Feb_02_2014",
"Jan_17_2014", "Jan_01_2014")
names(CrowdSourceData) <- ColumnName
CrowdSourceData=na.omit(CrowdSourceData)

library(e1071)
RNGkind(sample.kind = "Rounding")
set.seed(123)

Naive_Bayes_Model=naiveBayes(CrowdSourceData$class1~., CrowdSourceData)
Naive_Bayes_Model
NB_Predictions=predict(Naive_Bayes_Model,CrowdSourceData)
table(NB_Predictions,CrowdSourceData$class1)
mean(NB_Predictions!=CrowdSourceData$class1)

RNGkind(sample.kind = "Rounding")
set.seed(123)
train=sample(1:nrow(CrowdSourceData),5900)
trainSet=CrowdSourceData[train,]
testSet=CrowdSourceData[-train,]

class1.test=CrowdSourceData$class1[-train]
NB_2=naiveBayes(class1~.,trainSet)
NB_Predictions_2=predict(NB_2,testSet)
table(NB_Predictions_2,class1.test)
mean(NB_Predictions_2!=class1.test)



#SVM Linear
CrowdSourceData=read.csv(FilePath, header=T, na.strings ="?")
ColumnName <- c("class1",  "max_ndvi", "Jul_20_2015",       "Jun_02_2015","May_17_2015",
    "May_01_2015"         ,"Apr_15_2015","Mar_30_2015",       "Mar_14_2015",       "Feb_26_2015",
```

```
	"Feb_10_2015",		"Jan_25_2015","Jan_09_2015","Nov_17_2014",		"Nov_01_2014",
	"Oct_16_2014","Sep_30_2014", "Aug_13_2014", "Jun_26_2014", "Jun_10_2014","May_25_2014",
"May_09_2014", "Apr_23_2014", "Apr_07_2014", "Mar_22_2014", "Feb_18_2014", "Feb_02_2014",
"Jan_17_2014", "Jan_01_2014")
names(CrowdSourceData) <- ColumnName

#svm
library(e1071)
RNGkind(sample.kind = "Rounding")
set.seed(123)
train=sample(1:nrow(CrowdSourceData),5900)
trainSet=CrowdSourceData[train,]
testSet=CrowdSourceData[-train,]
dim(trainSet)
dim(testSet)
#SVM on train data linear cost =0.1
svmfit=svm(class1~., data=trainSet, kernel="linear", cost=0.1, scale=TRUE)
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

#SVM on train data linear cost =0.01
svmfit=svm(class1~., data=trainSet, kernel="linear", cost=0.01, scale=TRUE)
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

#SVM on train data linear cost = 1
svmfit=svm(class1~., data=trainSet, kernel="linear", cost=1, scale=TRUE)
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)


tune.out=tune(svm,class1~., data=trainSet, kernel="linear",
ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)),scale=TRUE)
summary(tune.out)
#best model
bestmod=tune.out$best.model
summary(bestmod)

#SVM on test data  linear
Class1_pred=predict(bestmod,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)


#radial
library(e1071)
RNGkind(sample.kind = "Rounding")
set.seed(123)
svmfit=svm(class1~., data=trainSet, kernel="radial", gamma=1, cost=0.1)
#SVM on test data  radial cost 0.1
```

```
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

svmfit=svm(class1~., data=trainSet, kernel="radial", gamma=1, cost=1)
#SVM on test data  radial cost 1
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

svmfit=svm(class1~., data=trainSet, kernel="radial", gamma=1, cost=10)
#SVM on test data  radial cost 10
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

tune.out=tune(svm, class1~., data=trainSet, kernel="radial",
        ranges=list(cost=c(0.1,1,10,100,1000), gamma=c(0.5,1,2,3,4)))
summary(tune.out)

bestmod=tune.out$best.model
summary(bestmod)

#SVM on test data polynomial
Class1_pred=predict(bestmod,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)


#polynomial
library(e1071)
RNGkind(sample.kind = "Rounding")
set.seed(123)
svmfit=svm(class1~., data=trainSet, kernel="polynomial", degree=2,cost=0.01)
#SVM on test data polynomial cost 0.01
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

svmfit=svm(class1~., data=trainSet, kernel="polynomial", degree=2,cost=0.1)
#SVM on test data polynomial cost 0.1
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)

svmfit=svm(class1~., data=trainSet, kernel="polynomial", degree=2,cost=1)
#SVM on test data polynomial cost 1
Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)


tune.out=tune(svm, class1~., data=trainSet, kernel="polynomial",
```

```
    ranges=list(degree=c(0.1,0.5,1,1.5,2),ranges=list(cost=c(0.01,0.1,1,10,100))))

summary(tune.out)
bestmod=tune.out$best.model
summary(bestmod)

Class1_pred=predict(svmfit,testSet)
table(Class1_pred,testSet$class1)
mean(Class1_pred!=testSet$class1)
```