Interesting Ensemble Systems Techniques

Vaibhav Khandelwal
35932110
School of Computing and Communications
Lancaster University
v.khandelwal@lancaster.ac.uk

Abstract

There have been several machine learning classification algorithms that have been released in the world of artificial intelligence and computational intelligence, but still ensembling techniques have made its own place which have not been replaced by anyone so far because of it's broad use-cases especially in real-time use. The main reason for utilizing such techniques to improve the accuracy of our decision making by weighing over important decisions and reaching the final result. In this report we have discussed about various ensembling techniques like random forest, adaboost, borda voting, stacked generalization, learn++, etc over various datasets and have compared together over various parameters.

1 Introduction

Started with the implementation of ensembling techniques for reducing the variance and thus the bias, but gradually it solved a numerous machine learning problems such as selection of features, estimation of confidence finding the missing feature, incremental and sequential learning, error reduction, and many others. Although this notion of ensembling is followed by humans in day to day life such as electing candidate in election which takes place using voting. Also, we often seek for expert system when we need to take some important decision or decision making like doctors, researchers, chefs, etc in order to make decision like what medicine to take in a certain infection, what topic to go through for understanding a particular concept, how much spices to be added in a soup to make it good, etc (Polikar, 2012).

We use other guidance in our daily life decision making because we all are not perfect, and we might have had some variance in the past based on our decisions. Similarly, in context of modelling not all classifiers are strong because if that would have been true then we could have directly made use of the strong classifier. So, to reduce variance and make a trade off between bias and variance we use ensembling techniques, and this could be seen from the Figure 1.

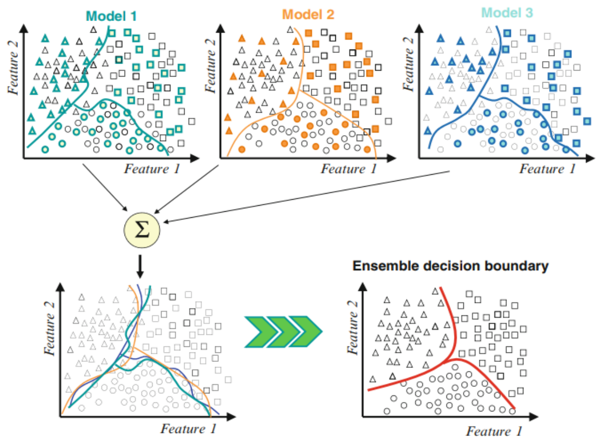


Figure 1: Variability reduction in ensemble system: Taken from (Polikar, 2012)

There are lots of data being generated at every second which needs to be transformed for the use of people, and this transformation opens a whole world of classification of data using different machine learning algorithms, but the how useful a transformation is for a person depends on the accuracy of the used algorithm. (Machová et al., 2006) suggests classification algorithm and the training set selected to be two of the main aspects to be considered. As mentioned in (Machová et al., 2006) a typical bagging algorithm for multi-class classification is as shown below:

```
1. Initialisation of training set d
2. for p classes from p =1.....P
    (a) Create a set of P equal length subsets $S_m$ of training data selected at random.
```

   (b) Learning of each classifier in S_m by a machine learning algorithm such that H_m: S_m -¿ R

3. Aggregate all the classifiers H_m based on voting.

Boosting is a method to improve a weak classifier which generates classifiers randomly, as in this the weak classifier is ran across various distribution of the training data, and then later combined or aggregated together to form a single classifier (Freund et al., 1996). Formally, the boosting algorithm works as: A set of labelled training set like (x_1, y_1),....,(x_N, y_N), where x_i are the features and y_i is the target variable are provided to the booster which on each round t= 1, 2,....,T fetches a distribution D_t over these sets and passes these to a weak classifier. After T rounds all the weak hypotheses results are combined together to get a final result.

The main comparison between bagging and boosting is that in bagging the distribution is same across all the subsets of data while boosting creates a different one. (Freund et al., 1996) suggests the two reasons for the performance of boosting algorithm better than bagging to be- different training set distribution, and getting low error when several large error hypotheses are combined together. Also as compared to bagging, boosting reduces the bias as mentioned in (Kong and Dietterich, 1995).

In this report, we propose different ensembling techniques namely- Bagging, Boosting, and Random Subspace technique. The techniques that has been discussed under the mentioned topics are- Random Forests, AdaBoost, Borda Voting, Stacked Generalization, Learn++ using single model. The 3 main innovative techniques that has been explored are: Learn++ using random model selection when doing classification, Using Borda Voting for classification in Learn++, and Implementing Stacked Generalization for classification in Learn++. The proposed classification methods have been carried out on five different binary classification datasets namely- Spambase (4601 rows of data) which consists of data if it is spam or not, most well known Iris (100 rows of data) dataset, SudentResult (395 rows of data) dataset which consists of binary classification data of whether a student fails or pass based on different subject marks, PokemonData (684 rows of data), and NBA (1329 rows of data) which is basketball data. So, basically all the methods have been performed from smallest of large datasets. All the models are then compared across all the datasets based on training and testing time, training dataset time, accuracy, precision, recall, and f1-score.

It is often experienced that we don't always have sufficient data available at an instant, so in that case our model needs to be updated iteratively ensuring it doesn't forget the past data pattern. In such case we make use of iterative ensemble approach like Learn++ and stacked generalization, thus raising the so-called stability–plasticity dilemma.

2 Related work

Authors of (Xia et al., 2018) has worked upon hyperspectral image classification using random forest ensembles because using simple supervised classification algorithms on a limited training samples might have lead to the cause of "curse of dimensionality", but ensembling techniques was considered by the author because of the various reasons mentioned by (Breiman, 2001) like- insensitivity towards high-dimensional input vectors, rapid prediction of out-of-sample, doesn't require much tuning of parameter, and it can easily rank the features based on their importance.

The authors in (Freund et al., 1996) applied boosting on the nearest neighbors classifier to show that a training set selected via boosting shows the same result as shown by the complete dataset. He also compared the boosting with the test error of Condensed Nearest Neighbor (Hart, 1968).

In (Freund, 1995), the author carried out boosting but without considering the prior accuracies of weak hypotheses. SO, to improve in this (Freund and Schapire, 1997a) took the notion of this and generates weighted majority hypotheses. (Schapire, 1990a) showed that weak classifiers when grouped together can give a better result which later was taken forward by (Freund, 1995) to implement "book-by-majority" algorithm.

The ensembling system was initilized by (Dasarathy and Sheela, 1979), who used multiple classifiers to partition the feature space, which was later taken by (Hansen and Salamon, 1990) and demonstrated that using neural network ensemble system classification could be improved. But the foundation of boosting using AdaBoost was laid by (Schapire, 1990b). Later by (Freund and Schapire, 1997b) it was implemented on regression problems. Later, various flavours of ensembling techniques got developed like mixture of experts (Jacobs et al., 1991), stacked generalization (Wolpert, 1992a), neural networks committee (Drucker et al., 1994)etc.

3 Methodology

3.1 Bagging Random Forest:

Random Forest is a bagging technique constructed via plethora of individual decision trees, such that bootstrapped sample is used to train every individual tree, while each tree leaf are splitted using randomly selected features (Xia et al., 2018).

As mentioned in (Kotsiantis, 2013), the decision tree models run sequentially by combining a sequence of simple decisions ensuring that in each decision a variable is compared that is if the variable is distinct then it should be compared to a threshold and if it's categorical then it should be compared to a set of values. (Safavian and Landgrebe, 1991) described a way to handle a non-parametric classification under supervised learning by breaking down a very complex problem into simpler chunks using decision making which also makes the comprehensibility of this algorithm a black-box neural network.

A typical decision tree mentioned by (Safavian and Landgrebe, 1991) is as shown in the Figure 2.

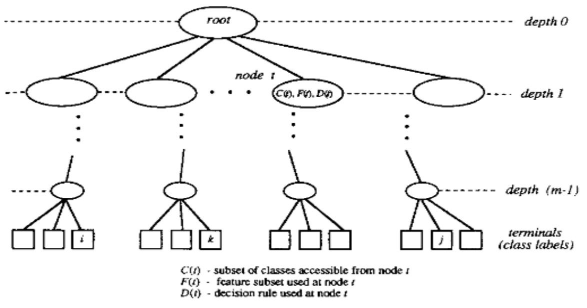

root
depth 0
node t
C(t), F(t), D(t)
depth 1
depth (m-1)
i
k
j
terminals (class labels)
C(t) - subset of classes accessible from node t
F(t) - feature subset used at node t
D(t) - decision rule used at node t

Figure 2: A typical Decision Tree: Taken from (Safavian and Landgrebe, 1991)

From the Figure 2 taken from (Safavian and Landgrebe, 1991), it can be seen that a decision tree consists of three different types of nodes namely-Root nodes, Internal nodes, and the leaf node. Root Node is the starting node (also known as top or parent node) where decision making starts, i.e. at this node first decision or condition is applied to split the features. It has no incoming edge but can have zero or more outgoing edges. Internal Nodes lies at the middle or intermediate level which are generate by the outcomes of several decisions applied at various parent nodes, and can possess multiple outgoing but single incoming edge. Leaf nodes (also known as terminal nodes) are the nodes which lie at the bottom of the tree or we can say at the end of the tree with no outgoing edge but only one incoming edge. At this node labels are predicted which are taken as an output.

When a set of 'n' features are given to a decision tree, then to decide the parent node n trees are formed as initially each feature is considered to be the parent node, but based on splitting criterion it is decided which tree to be selected. As under random forests, several decision trees are formed with each tree given a random sample of features on which all the decisions per decision tree are applied. According to (Safavian and Landgrebe, 1991) depth of the tree plays a very important role as shorter tree avoids over-fitting and tends towards easy generalization, but a longer tree leads to the over-fitting.

(Kotsiantis, 2013) and (Du and Zhan, 2002) mentions about entropy and gini index as a splitting criterion as shown in 1 and 2. Entropy is basically a metrics to calculate the randomness which at a particular node of a tree can be calculated using the 1, while gini index is probabilistic metric which calculates the number of times a classification would be made incorrect.

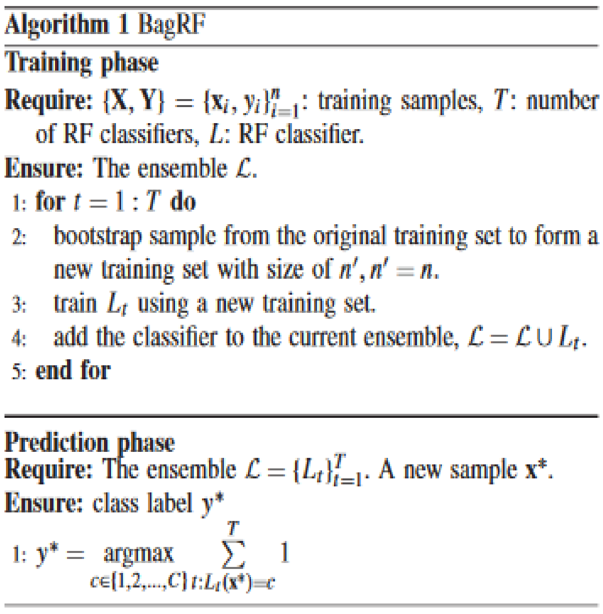
$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i) \tag{1}$$

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{2}$$

When we receive all the outputs from all the decision trees, then based on majority voting that is based on the output given by maximum of the decision trees are considered as the final output for a particular data. To improve the performance of the random forest several extensions have been performed like reducing feature space by random feature selection (Tuv et al., 2009), etc. The complete algorithm of bagging random forest can be seen from Figure 3.

3.2 AdaBoost:

Mentioned in (Freund et al., 1996) that an Adaptive boosting (AdaBoost) ensembling technique uses weight factor to process the data such that the samples of which misclassification has been done are increased so that learning algorithm gets concentrate on those samples. By changing the distribution of those samples dynamically, adaboost

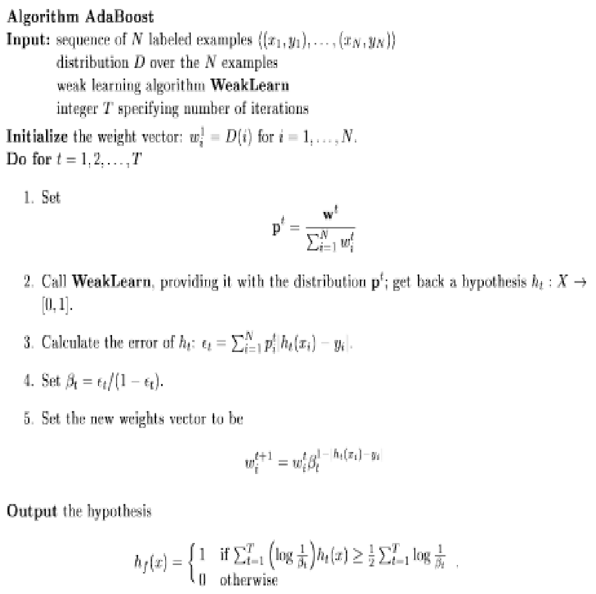


```
Algorithm 1 BagRF
Training phase
Require: $\{\mathbf{X},\mathbf{Y}\} = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$: training samples, $T$: number
    of RF classifiers, $L$: RF classifier.
Ensure: The ensemble $\mathcal{L}$.
1: for $t = 1 : T$ do
2:     bootstrap sample from the original training set to form a
       new training set with size of $n'$, $n' = n$.
3:     train $L_t$ using a new training set.
4:     add the classifier to the current ensemble, $\mathcal{L} = \mathcal{L} \cup L_t$.
5: end for

Prediction phase
Require: The ensemble $\mathcal{L} = \{L_t\}_{t=1}^{T}$. A new sample $\mathbf{x}^*$.
Ensure: class label $y^*$
1: $y^* = \underset{c \in \{1,2,\ldots,C\}}{\operatorname{argmax}} \sum_{t : L_t(\mathbf{x}^*) = c}^{T} 1$
```

Figure 3: Algorithm of Bagging Random Forest: Taken from (Xia et al., 2018)

proves to provide the most informative samples of training thus, reducing the training error. This algorithm reduces the learning error of any algorithm which generates the classifiers consistently as compared to the ones with random guessing (Freund and Schapire, 1997a). The algorithm of the adaboost implemented in this report is as shown in Figure 4.



```
Algorithm AdaBoost
Input: sequence of $N$ labeled examples $\langle(x_1, y_1), \ldots, (x_N, y_N)\rangle$
       distribution $D$ over the $N$ examples
       weak learning algorithm WeakLearn
       integer $T$ specifying number of iterations
Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$.
Do for $t = 1, 2, \ldots, T$

  1. Set
       $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$

  2. Call WeakLearn, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t : X \to$
     $[0, 1]$.

  3. Calculate the error of $h_t$: $\epsilon_t = \sum_{i=1}^{N} p_i^t |h_t(x_i) - y_i|$.

  4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.

  5. Set the new weights vector to be
       $w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$

Output the hypothesis
       $h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \log \frac{1}{\beta_t} \\ 0 & \text{otherwise} \end{cases}$ .
```

Figure 4: Algorithm of AdaBoost: Taken from (Freund and Schapire, 1997a)

3.3 Borda Voting:

(Polikar, 2012) explains the difference between voting and borda voting as a voting uses a winner-take-all strategy that is only majority of the votes corresponding to a class is taken and rest other classes are not taken into account, whereas in borda voting a classifier gives rank to all the classes such that the class which has higher chance of coming is given the maximum rank and vice-versa, and at the end rank given by all the classifiers to all the classes are summed up, and the class with the maximum sum of weights are taken to be the output.

3.4 Stacked Generalization:

Till now whatever techniques we have covered were using non-trainable combiners that is we are unaware of which part of the feature space has been learnt by which classifier. But (Wolpert, 1992a) suggested a technique known as stacked generalization which can be seen from Figure 5.

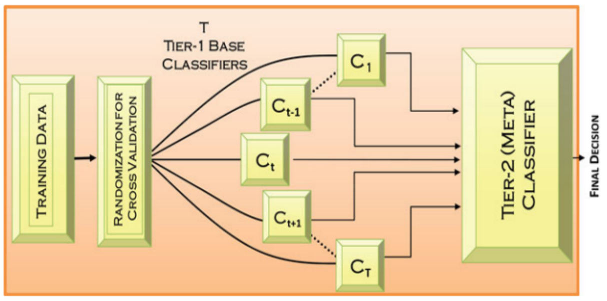


Figure 5: Stacked Generalization: Taken from (Wolpert, 1992b)

This technique uses trainable combiners which could be used based on our use case that is suppose out of 20 features only selected 5 we want and for those particular 5 features we can use the classifiers that would give good results with those 5 and combine them accordingly.

As can be seen from the figure that first training data are cross-validated into D blocks which are then used to create C_1, C_2,....,C_T classifiers in tier-1. Each classifier in tier-1 are trained through D-1 blocks and then each classifier make a prediction on D-1th block which was not given during training. The output on D-1th block data by the classifiers in tier-1 will acts as make feature set for training the classifier in tier-2. Now classifiers in tier-1 are trained using all the training data and then they make prediction on testing set, whose outputs will be given as an input set for the classifier in tier-2 which was trained initially. Now using the output on test set by tier-1 classifier, the tier-2 classifier will make prediction which would be the resultant prediction.

3.5 Learn++ using single type classifier:

Learn++ is an incremental approach of learning the data by learning only the novel part from the new batch of data that has not been learnt from the previous batch of data. But it becomes difficult if new batch of data introduces a new class which would affect our previous learning and we would make a wrong classification of the pattern of data learnt previously.

The algorithm of Lean++ from (Polikar et al., 2001) and (Mohammed et al., 2006) is as follows:

1. Out of n features f features are selected at random to train each classifier.
2. Initialize the probability of each n features by 1/n.
3. For each classifier:
    (a) Select f features randomly, and train each classifier.
    (b) Update the probability distribution on each sequential iteration by decreasing the probability of the selected f features by multiplying it with f/n.
    (c) Normalize the probabilities so that sum of the probability vector comes to 1

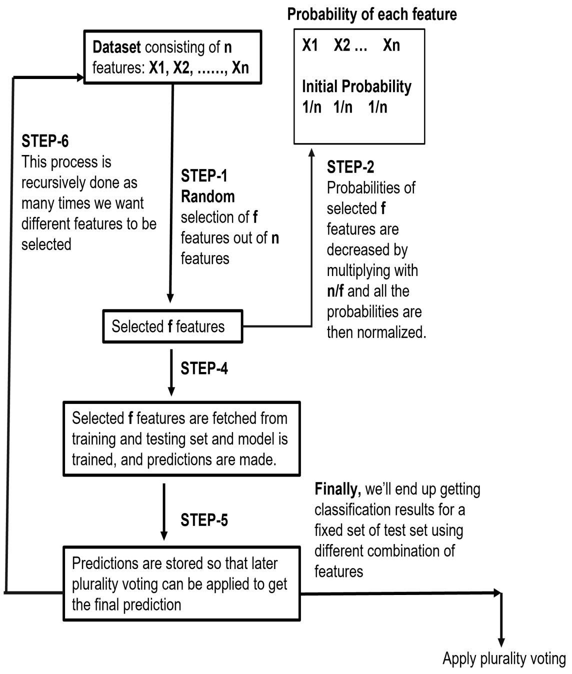
The above algorithm can be viewed in a flowchart in Figure 6. Here, for this we have used only a single type of classifier for classification.

3.6 Learn++ using random classifier model selection:

It is the extension of the previous technique with a small modification made in choosing the classifier. Here whenever the f selected features are trained then, training can be done via any model selected at random which can be seen from the code.

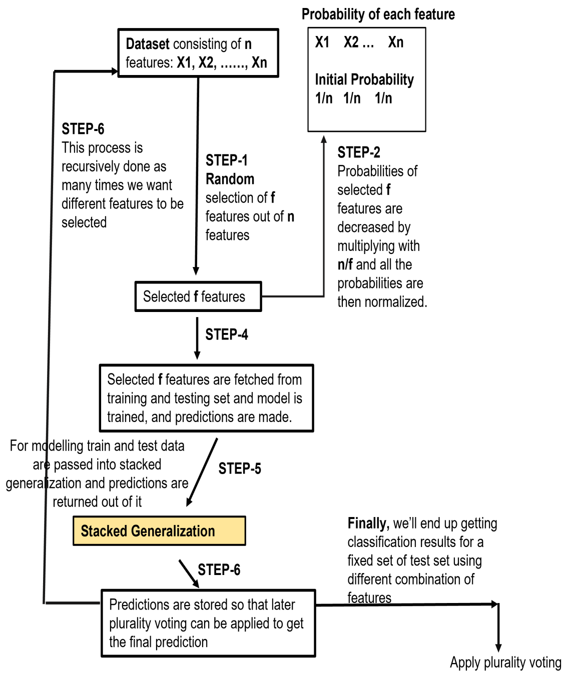
3.7 Learn++ with Stacked Generalization:

It is also an extension of above discussed technique which is Learn++. Here for modelling stacked generalization is taken as a classifier which can be seen highlighted in yellow box in the flow chart shown in Figure 7.



Probability of each feature
X1 X2 ... Xn
Initial Probability
1/n 1/n 1/n
Dataset consisting of n features: X1, X2, ..., Xn
STEP-6
This process is recursively done as many times we want different features to be selected
STEP-1
Random selection of f features out of n features
STEP-2
Probabilities of selected f features are decreased by multiplying with n/f and all the probabilities are then normalized.
Selected f features
STEP-4
Selected f features are fetched from training and testing set and model is trained, and predictions are made.
STEP-5
Finally, we'll end up getting classification results for a fixed set of test set using different combination of features
Predictions are stored so that later plurality voting can be applied to get the final prediction
Apply plurality voting

Figure 6: Learn++ algorithm flowchart



Probability of each feature
X1 X2 ... Xn
Initial Probability
1/n 1/n 1/n
Dataset consisting of n features: X1, X2, ..., Xn
STEP-6
This process is recursively done as many times we want different features to be selected
STEP-1
Random selection of f features out of n features
STEP-2
Probabilities of selected f features are decreased by multiplying with n/f and all the probabilities are then normalized.
Selected f features
STEP-4
Selected f features are fetched from training and testing set and model is trained, and predictions are made.
For modelling train and test data are passed into stacked generalization and predictions are returned out of it
STEP-5
Stacked Generalization
STEP-6
Finally, we'll end up getting classification results for a fixed set of test set using different combination of features
Predictions are stored so that later plurality voting can be applied to get the final prediction
Apply plurality voting

Figure 7: Learn++ with Stacked Generalization algorithm flowchart

3.8 Learn++ with Borda Voting:

It is also an extension of above discussed technique which is Learn++. Here for modelling Borda Voting is taken as a classifier which can be seen highlighted in yellow box in the flow chart shown in Figure 8.

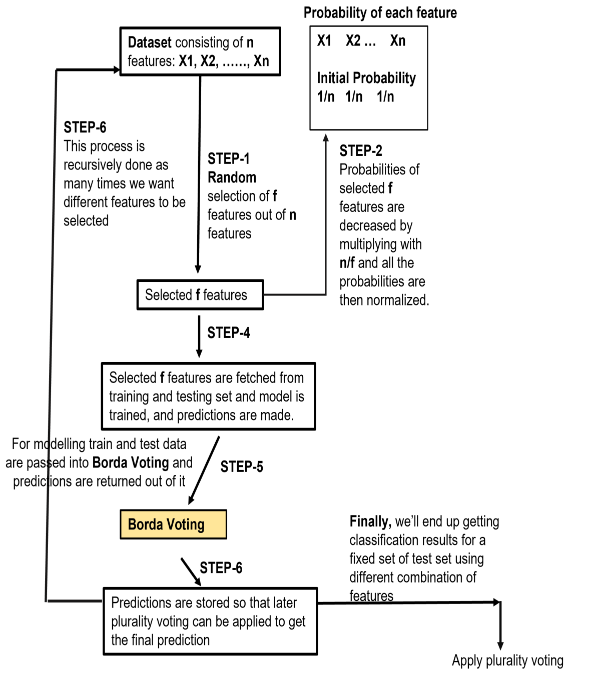

Probability of each feature
X1 X2 ... Xn
Initial Probability
1/n 1/n 1/n
Dataset consisting of n features: X1, X2, ..., Xn
STEP-6
This process is recursively done as many times we want different features to be selected
STEP-1
Random selection of f features out of n features
STEP-2
Probabilities of selected f features are decreased by multiplying with n/f and all the probabilities are then normalized.
Selected f features
STEP-4
Selected f features are fetched from training and testing set and model is trained, and predictions are made.
For modelling train and test data are passed into Borda Voting and predictions are returned out of it
STEP-5
Borda Voting
STEP-6
Finally, we'll end up getting classification results for a fixed set of test set using different combination of features
Predictions are stored so that later plurality voting can be applied to get the final prediction
Apply plurality voting

Figure 8: Learn++ with Borda Voting algorithm flowchart

4 Results/ Analysis

The dataset used for the analysis are- Spambase, Iris, StudentResult, PokemonData, and NBA as mentioned in the introduction. This section is divided into two parts- Individual technique results and Comparison of all techniques.

4.1 Individual Technique Analysis

4.1.1 Random Forest

When Random Forest is run across all the five dataFigure 9 at different training size then following pattern was observed as shown in Figure 9.

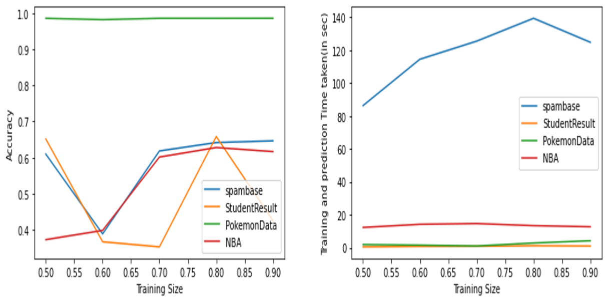


Figure 9: Accuracy and Time in Random Forest

It can be seen that for a data with very less rows of around 100, the accuracy seems to be almost 99% for all the training size, but for others trend seems to be fluctuating. For medium size data as training size increased accuracy also increased, but for large sized data as training size increased accuracy first increased but then started decreasing after 60% training size. Also, with the increase in training size the average accuracy was found to be around 65%.

Except for Spambase dataset, the average time taken by algorithm is around 20 seconds but for Spambase it ranges between 80 to 140 seconds on different training size.

4.1.2 AdaBoost

When the algorithm is ran for number of classifiers in the algorithm as 3 across various training size, it was observed that accuracy and recall followed the similar pattern, and precision and f1-score followed the same pattern across different training size. Except for Spambase dataset, all other dataset showed a good accuracy and recall across all the training size as compared to fluctuaing value in case of precision and f1-score, and comparatively low value as compared to other datatset. Thus, this algorithm shows an accuracy of around 95% for large dataset and almost 100% for very small dataset as shown in Figure 10.

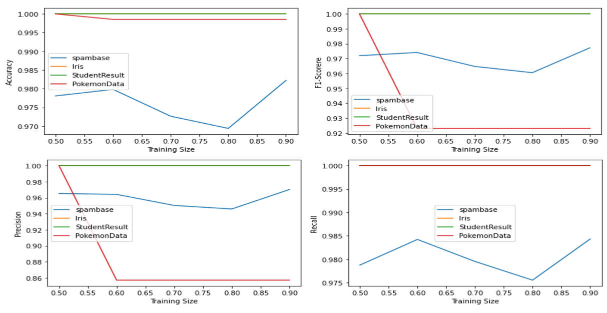


Figure 10: Metrics of AdaBoost

Except for Spambase dataset, the average time taken by algorithm is around 0.05 seconds but for Spambase it ranges between 0.2 to 0.35 seconds on different training size as can be seen from Figure 11.

4.1.3 Borda Voting

The classification models put into algorithm are- Logistic regression, decision tree with entropy as splitting criterion, decision tree with gini index as splitting criterion, and Neural network. From the Figure 12 it can be seen when the algorithm is ran across various training size. It is observed that as training size increased the accuracy, precision, recall and f1-score of all the dataset increases except

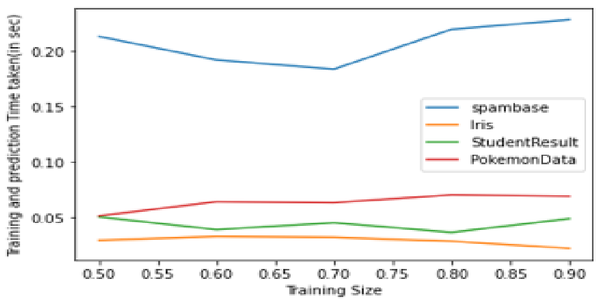


Figure 11: Time taken in AdaBoost

the smallest dataset which is of Pokemon. So, this method doesn't worl well on small dataset.

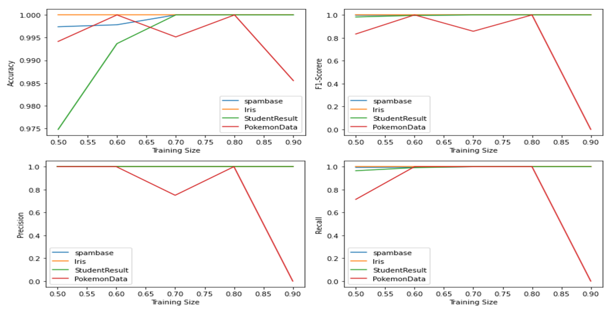


Figure 12: Metrics of Borda Voting

When seen the Figure 13, it can be seen depending on the size of the dataset, as dataset size increase time taken for training and classification also increases.

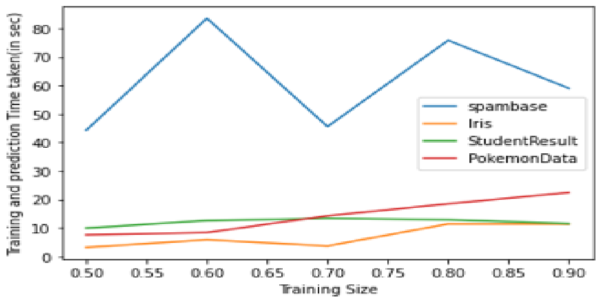


Figure 13: Time taken in Borda Voting

4.1.4 Stacked Generalization

The classification models put into algorithm are- Logistic regression, decision tree with entropy as splitting criterion, decision tree with gini index as splitting criterion, and Neural network. As can be seen from Figure 14 with this algorithm on all the dataset a very high accuracy was observed across all the training size.

When seen the Figure 15, it can be seen depending on the size of the dataset, as dataset size increase time taken for training and classification also increases.

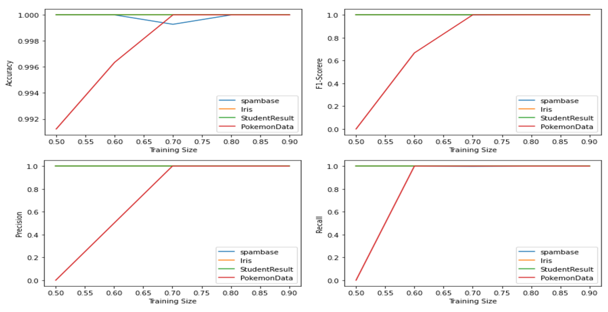


Figure 14: Metrics of Stacked Generalization

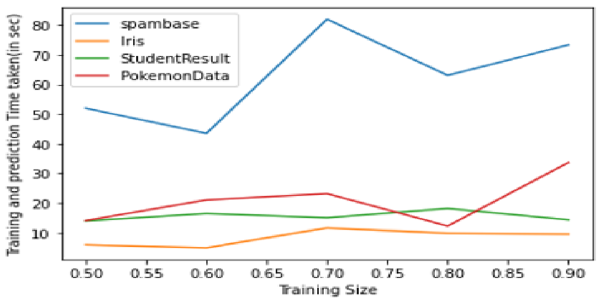


Figure 15: Time taken in Stacked Generalization

4.1.5 Learn++ (Single model classifier)

The classification models put into algorithm are- Logistic regression, decision tree with entropy as splitting criterion, decision tree with gini index as splitting criterion, and Neural network. Here when see the Figure 16 it is seen that accuracy was good either for very low training size or high training size. But for mid range training size of around 70% there was a very huge drop in accuracy.

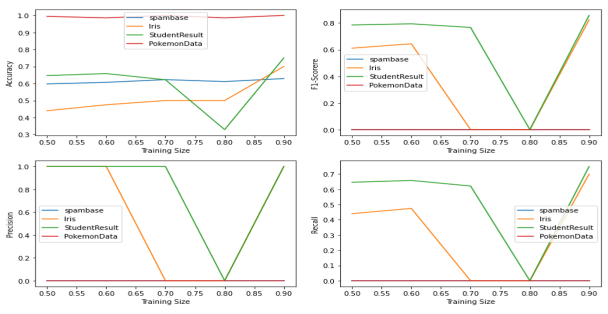


Figure 16: Metrics of Learn++ (Single model classifier)

When seen the Figure 17, it can be seen depending on the size of the dataset and type of model used as a classifier, as dataset size increase time taken for training and classification also increases.

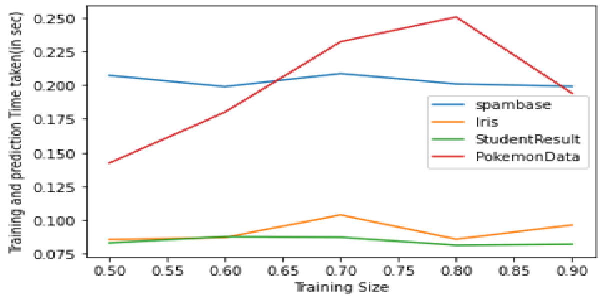


Figure 17: Time taken in Learn++ (Single model classifier)

4.1.6 Learn++ (Random model classifier)

From the Figure 18, it can be seen that this algorithm is not giving any good output for any of the training size except for a very small datat size.

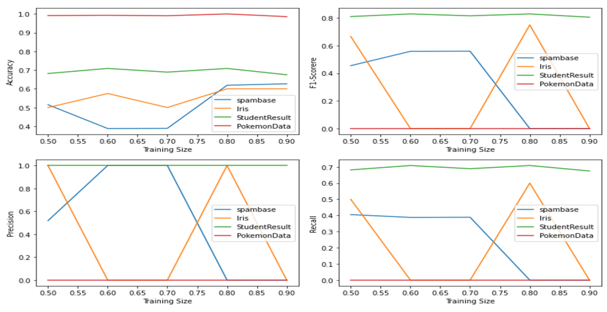


Figure 18: Metrics of Learn++ (Random model classifier)

As can be seen from Figure 19, here training time somewhat depends on the training size, but also on the model that are selected randomly which is black-box.

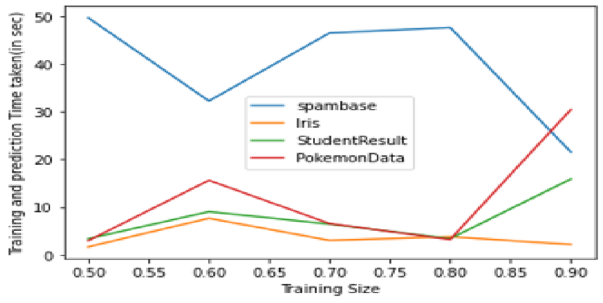


Figure 19: Time taken in Learn++ (Random model classifier)

4.1.7 Learn++ with Borda Voting

In Figure 20 it can be seen that this model only works good for smaller dataset, but for large dataset it's not giving good accuracy.

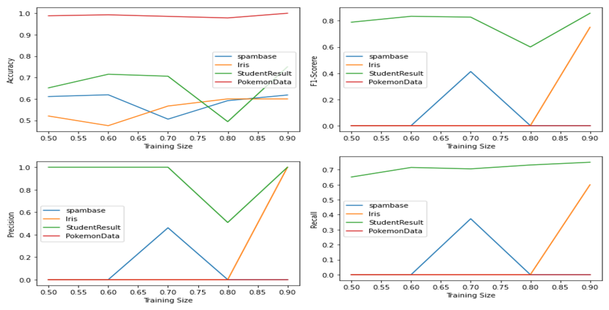


Figure 20: Metrics of Learn++ with Borda Voting

From the Figure 21, it is observed that as training size increases training and prediction time also increased but for a larger dataset.

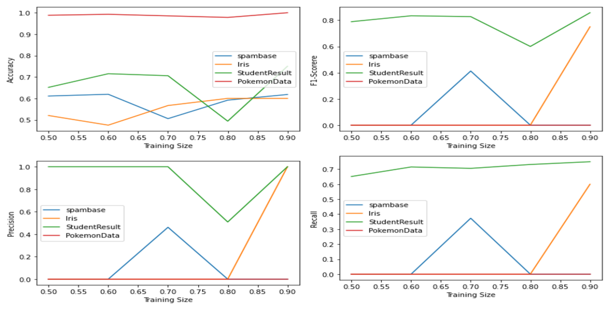


Figure 21: Time taken in Learn++ with Borda Voting

4.1.8 Learn+ with Stacked Generalization

From Figure 22, it can be seen that the model overfitted a lot due to which we aren't getting good accuracy for any of the data, even with increase in the training size.

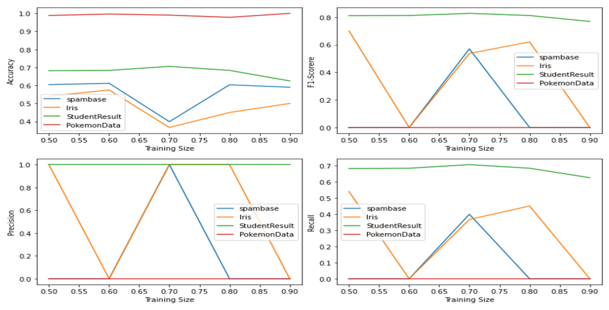


Figure 22: Metrics of Learn+ with Stacked Generalization

As training size increased the total training time started to decrease at a very low rate as seen from Figure 23.

4.2 Comparison of all Technique Analysis

As can be seen from above at training size 60% we are getting good accuracy. So for combined comparison we'll be considering that training size.

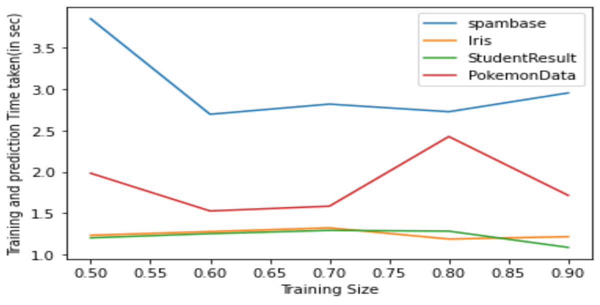


Figure 23: Time taken in Learn+ with Stacked Generalization

As can be seen from Figure 24, that when all the techniques were applied on larger dataet which is Spam dataset and the smaller dataset which is the Pokemon one, we found that the creative models showed less accuracy of around 60% as compared to other models which showed around 90% on larger datatset. For smaller dataset, creative models performed better than Random Forest and normal Learn++.

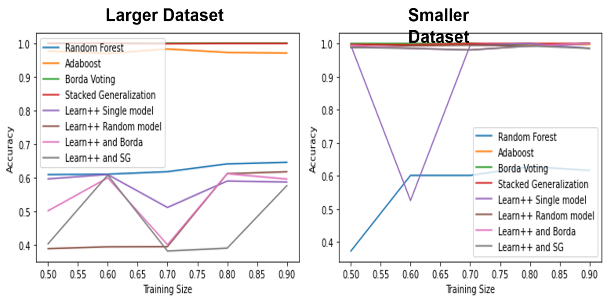


Figure 24: Combined Accuracy of all Techniques

From Figure 25 it can be seen that f1-score for larger dataset shows the same observation as accuracy but F-1 score for smaller dataset of creative model is less as compared to other models.

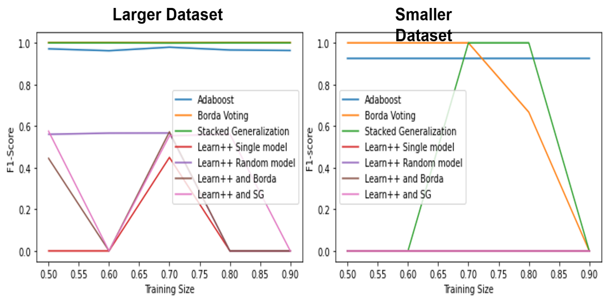


Figure 25: Combined F1-Score of all Techniques

5 Discussion

From the above discussion we can say that our creative models namely- Learn++ with random model, Learn++ with Borda Voting, and Learn++ with stacked Generalization overfits the model due to high complex network formed while putting Borda Voting and Stacked Generalization in between thus, leading to bad test accuracy and f1-score. Also, compared to rest of models and few models from the creative part performed quite better than the self created random forest. With the increase in training size from 50% to 80% all the models showed improvement.

6 Conclusion

In this task of implementation of ensembling techniques it is observed that Adaboost, Learn++, Stacked Generalization tends to show good behaviour towards all types of data as compared to the creative ones namely Learn++ with random model, Learn++ with Borda Voting, and Learn++ with stacked Generalization which shows quite a good behaviour with an accuracy of around 60% with larger data but with smaller datatset it showed a poor behaviour.

Future improvements that could be carried out to try combining two or more techniques as an ensemble but making sure that it doesn't over-fit.

References

Leo Breiman. 2001. Random forest, vol. 45. *Mach Learn*, 1.

Belur V Dasarathy and Belur V Sheela. 1979. A composite classifier system design: Concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713.

Harris Drucker, Corinna Cortes, Lawrence D Jackel, Yann LeCun, and Vladimir Vapnik. 1994. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301.

Wenliang Du and Zhijun Zhan. 2002. Building decision tree classifier on private data.

Yoav Freund. 1995. Data filtering and distribution modeling algorithms for machine learning.

Yoav Freund and Robert E Schapire. 1997a. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Yoav Freund and Robert E Schapire. 1997b. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Yoav Freund, Robert E Schapire, et al. 1996. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer.

Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001.

Peter Hart. 1968. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Eun Bae Kong and Thomas G Dietterich. 1995. Error-correcting output coding corrects bias and variance. In *Machine learning proceedings 1995*, pages 313–321. Elsevier.

Sotiris B Kotsiantis. 2013. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283.

Kristína Machová, Frantisek Barcak, and Peter Bednár. 2006. A bagging method using decision trees in the role of base classifiers. *Acta Polytechnica Hungarica*, 3(2):121–132.

Hussein Syed Mohammed, James Leander, Matthew Marbach, and Robi Polikar. 2006. Can adaboost. m1 learn incrementally? a comparison to learn++ under different combination rules. In *International Conference on Artificial Neural Networks*, pages 254–263. Springer.

Robi Polikar. 2012. Ensemble learning. In *Ensemble machine learning*, pages 1–34. Springer.

Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. 2001. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508.

S Rasoul Safavian and David Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Robert E Schapire. 1990a. The strength of weak learnability. *Machine learning*, 5(2):197–227.

Robert E Schapire. 1990b. The strength of weak learnability. *Machine learning*, 5(2):197–227.

Eugene Tuv, Alexander Borisov, George Runger, and Kari Torkkola. 2009. Feature selection with ensembles, artificial variables, and redundancy elimination. *The Journal of Machine Learning Research*, 10:1341–1366.

David H Wolpert. 1992a. Stacked generalization. *Neural networks*, 5(2):241–259.

David H Wolpert. 1992b. Stacked generalization. *Neural networks*, 5(2):241–259.

Junshi Xia, Pedram Ghamisi, Naoto Yokoya, and Akira Iwasaki. 2018. Random forest ensembles and extended multiextinction profiles for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 56(1):202–216.