2022 IEEE International Conference on Knowledge Graph (ICKG)

Deep Associative Classifier

Md Rayhan Kabir*, Samridhi Vaid†, Nitakshi Sood‡ and Osmar R. Zaiane§
Alberta Machine Intelligence Institute, University of Alberta
Edmonton Canada.
Email: *rayhan.kabir@ualberta.ca, †svaid@uallberta.ca, ‡nitakshi@ualberta.ca, §zaiane@ualberta.ca

***Abstract*—Deep neural networks architecture provides a powerful technique for solving various problems including classification. They owe their performance to the complex and layered data representation and processing built upon neural networks. The success of deep neural networks in various fields has resulted in less focus on other techniques like rule-based models, especially associative classifiers. Associative classifiers are competitive models to deep neural networks on tabular data but suffer from certain limitations i.e., require proper threshold values that differ for different datasets. Even though deep neural networks have resulted in huge success, they have complex and lengthy hyper-parameter tuning. In recent years, attempts to develop models that can compete with deep neural networks using deep representations with decision trees while reducing hyper-parameters have been tried. In this study, we propose a Deep Associative Classifier (DAC), an ensemble of associative classifiers that transforms features in a deep model representation. This model has deep neural network like architecture with associative classifiers as a base learner and overcomes some of the limitations of deep neural network architecture as well as associative classifiers. We use 10 UCI datasets and compare our approach with other existing deep neural network models, a gcForest approach, and associative classifiers. Our proposed model outperforms various state-of-the-art classifiers not only in terms of accuracy but also memory requirement and has fewer hyper-parameters to tune.**

***Index Terms*—Deep Architecture, Learning Representation, Associative Classifier, Deep Neural Network, Ensemble Learning.**

I. Introduction

Classification is a supervised machine learning technique that is used to label the data in various distinct class labels. To solve classification problems, some of the popular classification algorithms are Decision Trees, Random Forest and Support Vector Machines. Even though these models have good performance in classification tasks, they generally do not perform well for complex classification tasks. Deep learning techniques are better able to handle such tasks, with high accuracy and efficiency. Deep learning is a sub-branch of machine learning and uses deep neural networks with a layered architecture. In recent years deep neural networks have shown promising results in data classification, among other tasks, and in many cases surpassing human performance.

One of the main reasons for the high performance of deep learning techniques is their layered architecture. It has been conjectured that the success of deep learning is due to the layer-by-layer processing, the in-model feature transformation, and the complexity of the model [6]. But deep learning approaches suffer from many limitations like the necessity of hyper-parameter tuning, and the effect of the parameter tuning is hard to interpret. They also require a large amount of labelled training data, which is often not available in most domains. Zhou et al. [6] tried to utilize this deep representation of deep neural networks while eliminating some of the limitations of the deep neural networks by developing a deep model called the gcForest. gcForest is an ensemble of decision trees with the architecture of the deep neural networks. Their model showed promising results but, their model still requires some predefined hyper-parameters such as window size, number of windows, number of trees in the forest, etc.

We speculate using the idea adapted by gcForest and further exploit the layered architecture of deep neural networks. This motivated us to develop a new classifier that requires fewer hyper-parameters, is easy to tune, requires less memory, has the potential to be explainable and utilizes the advantage of the layer-wise data processing architecture of deep neural networks. Associative classifiers are one such classifier that have very good performance and is also easy to interpret. The central concept of associative classifiers is association rule mining [17] and they make use of constrained rule mining and classification to build classification models. The resulting rules are represented in the form of $X \rightarrow Y$, where X, the antecedent, is a conjunction of features, and Y, the consequent, is a class label.

In the last decade, associative classifiers have shown competitive results against the state-of-the-art classifiers. However, they suffer from many limitations inherited from association rule mining, i.e. they require proper threshold values, which vary from dataset to dataset. To remove the necessity of adjusting difficult threshold values for different datasets, Li and Zaiane [1] propose an associative classifier which is further improved by Sood and Zaiane [3]. Their models eliminate the necessity of prefixing any threshold values to generate rules from the dataset. In their models, they used the Kingfisher Algorithm [2] to find the statistically significant rules from the dataset which are used for the classification task. Though their model showed a promising result, it requires significant memory and huge run time for high-dimensional datasets.

In this paper, we endeavour in using and combining the deep representation of deep neural networks and the simplicity of the associative classifier. We propose an ensemble model, Deep Associative Classifier (DAC) that uses associative

classifiers as base learners and has a deep architecture. We implemented a two-step classifier where in the first step, we scan through the feature vectors to generate class probabilities using the associative classifier SigD2. These class probabilities are appended with the feature vector and form the input for the second step. In the second step, we adopt the general idea of deep neural networks where the input data is processed in a layer-wise fashion. Each layer in the second step is an ensemble of an associative classifier. To reduce the increase of the output vector in each layer, we introduce a filtering process eliminating the need to append probabilities of equal base learners in the feature vector output. In contrast to deep neural networks, our model does not need any predefined number of layers. The model itself defines the number of layers during the classification task.
With this approach, we seek to improve classification accuracy by utilizing the layered architecture of the deep neural networks and associative classifiers. Further, we try to improve the performance of the associative classifier in terms of memory requirement for datasets with high dimensions and reduce the number of hyper-parameter to make the model more practical to tune. Deep Associative Classifier has much fewer hyper-parameters and lower memory requirements than gcForest. It also outperforms gcForest in terms of accuracy on the majority of the datasets we tried. Moreover, our model is also able to perform well across most datasets with default settings. The contribution of our work can be stated as follows:

1) A new ensemble of the associative classifier with a layered architecture in accordance with deep neural networks, which provides better accuracy than existing associative classifiers, deep neural network classifiers and gcForest.
2) A classification approach with less memory requirement
3) A classification model that requires fewer hyper-parameters, is easy to tune and the impact of changing the hyper-parameters is easy to understand.

The rest of the paper is organized as follows. In Section II we discuss the related work. Section III describes our approach of utilizing associative classifiers and deep layered architecture. In Section IV we describe our experimental findings. Section V highlights some possible future refinements of the model, and finally, Section VI concludes our paper.

II. Related Work

This section describes related works on ensemble learning, associative classifiers and deep architectures.

A. Ensemble learning

Ensemble learning is a technique that utilizes multiple models to solve computationally difficult problems and provide high accuracy [7], [8]. Kowsari et al. [9] used an ensemble of deep architecture to predict the class of different types of data like image, text and tabular data. Bagging [18] and boosting [19] are popular ensemble techniques used for improving the performance by adopting different ways to combine weak learners to produce a strong learner. In [20] the authors showed that bagging and boosting improved classification performance for demographic classification of handwriting. Prusa et al. [21] evaluated 12 different techniques that improve the classification performance on 3 datasets for twitter sentiment classification. Based on their experimental findings, they concluded that bagging provided superior performance as compared to the rest of the techniques and significantly improved the classification performance.

B. Associative classifiers

Associative classifiers are a type of rule-based classifiers that provide a classification approach that is intuitive and easily understandable. Liu et al. [12] first came up with the idea to utilize the association rules for classification. They made use of the Apriori algorithm to generate classification association rules. Following their work, several other rule-based classifiers have been developed i.e., CMAR [13], ARC [14]. These models are easy to interpret and also have very good performance. But as mentioned earlier, they suffer from the limitation of the requirement of predefined threshold values. These approaches sometimes tend to generate meaningless rules, which leads to poor classification accuracy. To overcome these limitations of rule-based classifiers, Li and Zaiane [1] proposed SigDirect, an approach that mines statistically significant rules without the need for support and confidence measures. This methodology was derived by pushing the rule constraints of the Kingfisher Algorithm [2]. Sood and Zaiane [3] further extended this work by proposing SigD2 with a novel rule pruning strategy to remove the noisy classification rules. Their model achieved better results than the original Sigdirect and other rule-based classifiers, including decision trees.

C. Deep architectures

Deep neural networks are a powerful technique used for classification. They not only provide high accuracy but are highly efficient. However, they are complex, have a tedious hyper-parameter tuning process and require high computational power. In addition, deep neural networks require a high volume of data for training purposes. Training a deep model with a low volume of data results in overfitting [4]. Thus deep model cannot perform well if the dataset is not large enough. Zhou et al. [6] tried to overcome some of these limitations of deep neural networks by proposing gcForest, an ensemble of decision trees that exploits the significant features of a neural network like representation learning, ability to deal with high dimensional data and model complexity. Tanno et al. [15] proposed another architecture where they used adaptive neural trees whose architecture is hierarchical and can learn the parameters of the model over the progressive growth. The parameters are auto-tuned according to the size of the dataset. Sun et al. [16] also proposed a model which is inspired by the gcForest [6] where they select important features by following some measures for classification. The subsequent phases remain the same as gcForest, but in each step, they choose the important features only. Both the work of Tanno

et el. [15] and Sun et al. [16] focus more on image data. Their architecture performs well in the case of classification of an image dataset. Though our focus is on tabular data, their works provide a significant direction for our model because, in our case, we want to develop a model which is efficient for datasets of different sizes.

As we highlighted earlier one of the limitations of the associative classifiers such as Sigdirect and SigD2, is that they cannot handle data well if the feature vector is large. To solve this limitation, we utilize the idea of gcForest [6] by building an ensemble but of associative classifiers, namely SigD2.

III. Methodology

This section introduces the proposed model, the Deep Associative Classifier (DAC). To develop our model we followed the deep neural network architecture. In case of deep neural networks, the input data are processed at multiple layers which might be a reason for their good performance. Gcforest proposed by Zhou et al. [6] utilizes the layered architecture of deep neural networks and builds an ensemble using a decision tree as a base learner. While our architecture is different, we borrow the idea of the ensemble but use the associative classifier SigD2 as the base learner. In SigD2, an additional rule pruning technique is used to prune the noisy rules which enhances the performance of the model while significantly reducing the number of classification rules in the model. We use different ensembles, one in each layer of our architecture and another one at the start while scrutinizing the feature space. Building our layered architecture follows different steps.

In the first step, our architecture performs a scanning through the feature vectors of the dataset, this is the scanning phase of our architecture. In this phase, a fixed-size window slides over the input feature vector and selects a pre-specified number of feature vectors sequentially. In the work of Zhou et al. [6], they used 3 windows at the same time. We experimented with a different number of windows and found that only one window is sufficient for the model. This is discussed in detail in the experimental result section. As we are taking the subset of the feature vector for each base learner, our reasoning behind adding the class probabilities created by the base learners is that they would serve as a heuristic for the base learners of the following layer to more accurately predict the class label. In the scanning phase, subsets of the feature vectors are formed and with each subset, a base learner SigD2 is trained. Each base learner provides class probabilities for the instances and the class probabilities are concatenated to the original input vector. For example, if we have a dataset with a feature vector size of 100 and the dataset consists of the instances of binary classes, with a sliding window size of 30, we will have, 71 subsets of the dataset. Each of the SigD2 base learners provides class probabilities at the end of the learning. That is as the data is of binary class, each base learner generates two-class probabilities - one for each class. Thus, with 71 base learners, we have a total of 142 class probabilities. These class probabilities are then concatenated with the original features resulting in 242 features at the end of the scanning phase. The newly formed dataset with the concatenated features is provided as an input to the next phase, the cascading phase. Algorithm 1 shows the steps performed in the scanning phase. Figure 1 illustrates the scanning phase of our architecture.

Algorithm 1 Algorithm **Scanning phase**

Input: Features: All features in dataset
Output: Features concatenated with class probabilities

```
WindowSize=30
NewFeatures=[]
n=0
while n+WindowSize ≤ features.length do
    features_to_train ← Features[n:n+WindowSize]
    NewFeatures.append(Sigd2.classProbabilities(
    features_to_train)
    n=n+1
end while
Features ← Features.concatenate(NewFeatures)
return Features
```

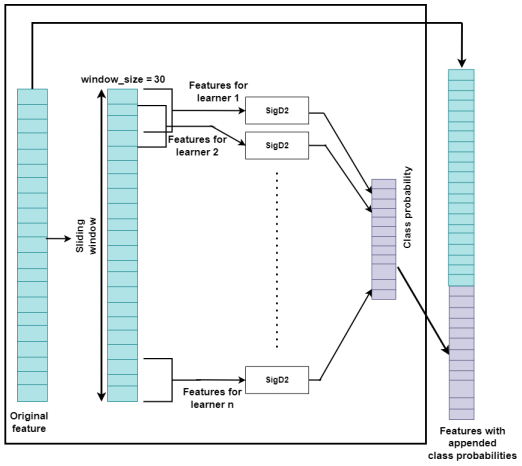

Fig. 1: Scanning phase of Deep associative classifier

The next step of our model is the cascade phase. Like the scanning phase, we again use Sigd2 as the base learner. In this step, the feature vector generated in the scanning phase acts as an input. The features are processed layer by layer. At each layer, we use 100 SigD2 base learners. For each base learner, 30 features are selected randomly with shuffle and replacement from the input feature vector. For the scanning phase and the cascading phase, we selected 30 features because if the feature vector size is very large, the memory requirement and run time of SigD2 increases, that is the effectiveness and efficiency of SigD2 tend to decrease with a dimension larger than 30 in terms of memory requirement and run time. Thus we decided to run the experiment with 30 features, a hyper-parameter in our model. After training, each of the

base learners calculates the class probabilities. For example, if we consider our previously mentioned example of a dataset with binary classes, each of the base learners produces two class probability. With 100 base learners, we have 200 class probabilities which we append to the features after the filtering process and weight assignment. In the cascading phase, we also predict the class label of the test dataset with each of the base learners to evaluate the accuracy at the current layer. The goal is to have an increase in accuracy from one layer to the other. We perform a max vote strategy to find the final class label of the instances and calculate the accuracy. Figure 2 shows the architecture of a single layer in the model, in fact, the very first layer of the cascading phase. The difference is that for the first layer the input feature vector comes from the scanning phase while for any other layer its input feature vector is the output vector of the previous layer. For example, at the end of the first layer, new class probabilities generated by the base learners of the first layer are appended with the input feature vector of the first layer. This updated feature vector is the input for the second layer. The output feature vector of the second layer acts as an input to the third layer and so on. Figure 3 shows the input feature vector and output feature vector at each layer.

Before forwarding the data to the next layer, we append the class probabilities from the base learners as a feature, similar to the scanning phase. But before appending, we introduce a filter to assign weights to the repetitive class probabilities. From our analysis, we found there are many base learners which produce the same class probability for each of the classes for each of the instances. Adding them directly to the output vector fed to the next layers produces more identical base learners and reduces the diversity in the ensemble. Thus instead of appending the identical class probabilities, we append the matched class probabilities multiplied with a weight factor. The weight factor is simply the count of learners generating the same class probabilities. In other words, If there is a set of identical class probabilities repeated n times, we only append the class probabilities multiplied by n to the output vector, making the feature vector increase with a smaller extension. While comparing the class probabilities, we compare them up-to 7 digits precision. Algorithm 2 shows the ensemble procedure and Algorithm 3 reveals the filtering procedure for the new features.

The number of layers in our architecture is not preset but determined automatically. At each layer, a prediction on test data is performed. The goal is to have an improvement of the prediction accuracy from one layer to the other. If no improvement is noted at a given layer compared to the previous layer, the creation of a new layer is halted and the layer with the best accuracy is considered the final layer. Since each layer is an ensemble of SigD2 classifiers, the prediction of a layer is a max voting among base learners. This prediction is not appended in the output vector but its sole purpose is to decide whether to continue creating another layer if the accuracy improves or to halt and ignore the last layer if no improvement is noted. Because we cease adding new layers

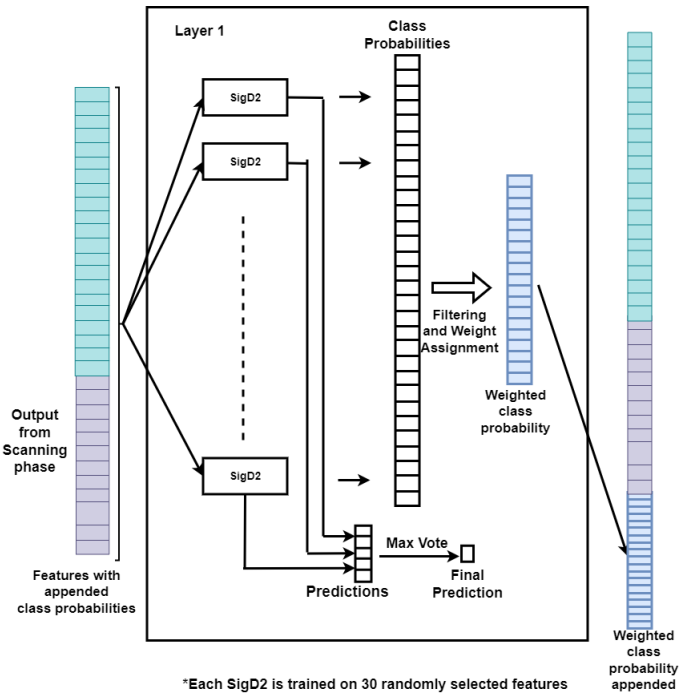

Fig. 2: Architecture of one single layer in the cascade phase

Algorithm 2 Algorithm **Cascade**

Input: Features: Concatenated features from scanning phase:
Output: Accuracy, Concatenated features for next layer

```
NewFeatures=[]
prediction=[]
for i in range 1 to 100 do
    subsample → randomly select 30 features from Features
    Sigd2.train(train_data)
    predict ← Sigd2.predict(Test_data)
    prediction.append(predict)
    NewFeatures.append(Sigd2.classProbabilities(subsample)
end for
finalprediction ← max_vote(predictions)
accuracycalculate_accuracy(test_label, finalprediction)
if (accuracy > previous_layer_accuracy) do
    NewFeatures← weightedNewFeature(NewFeatures)
    Features ← Features.concatenate(NewFeatures)
else
    accuracy ← previouslayer_accuracy
end if
return Features, accuracy
```

when there is no progress in the accuracy, the model will converge to a final output at the end of a particular layer. No improvement means that we cease adding additional layers even if the accuracy is the same as the layer before. This prevents the model from running indefinitely. The highest level of accuracy we can get is 1 if performance keeps improving in additional layers. As a result, the model will eventually

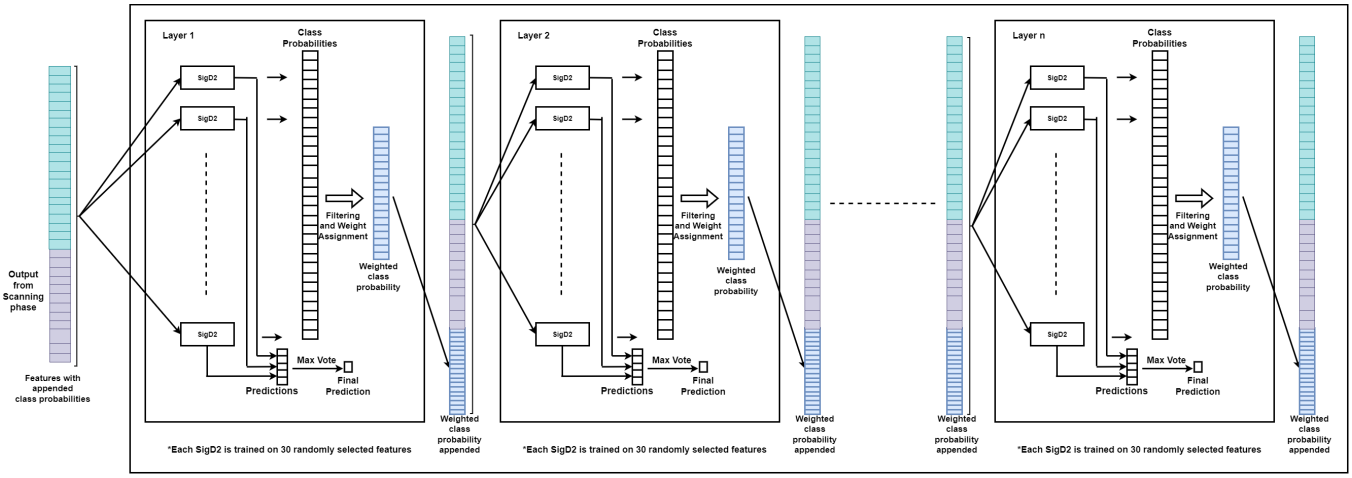

Fig. 3: Cascade phase of Deep associative classifier

Algorithm 3 Algorithm **weightedNewFeature**

Input: NewFeatures: generated class probabilities
Output: weightedfeature weighted class probabilities as features

```
unique_features ← unique value set from NewFeatures
for unique in unique_features do:
    count ← unique in NewFeatures
    weightedfeature.append(unique * count)
end for
return weightedfeature
```

converge to the desired result. Figure 3 shows the architecture of the cascading phase of the model and the whole architecture is provided in Figure 4.

IV. Experimental Results

In this section, we provide an in-depth analysis of our experimental findings.

A. Dataset and Performance Measure

We use 10 different datasets from the UCI repository [10] to evaluate the performance of our proposed model. Before applying the algorithms to the datasets, we discretize the numerical attributes of the dataset as stated in [11]. We then vectorize the features. In the result, the reported feature vector size is the size of the feature after discretization and vectorization. For all the experiments, we use the same discretized values so that the performance is measured across the same format of the dataset. We select these datasets so that we can evaluate the performance of our model on different types of datasets i.e., datasets having a different number of records and a different number of features. By evaluating the performance of our proposed model on these datasets, we try to ensure that our proposed approach works well across datasets with different sizes in records and features.

B. Results

Our proposed model is assessed in terms of accuracy and memory requirement. For this comparison, we select random forest, gcForest proposed by Zhou et al [6], SigD2 proposed by Sood and Zaiane [3] and three different deep neural network architectures. From the rule-based classifier, we select only SigD2 because it was shown to outperform other existing rule-based classifiers. We also calculate the memory requirements of the model and show a comparative analysis. Finally, we conduct a statistical analysis on the accuracy to show the significance of our results. In our experiments, we divide the dataset into training and testing data, 80% and 20% respectively. Among the 10 datasets, two datasets, breast and flare have a feature vector size 18 and 30 respectively, therefore smaller than our window size. Thus for these two datasets we use a window size of 15. For all other datasets, we use a window size of 30. In each layer of the cascade phase, we use 100 SigD2 base learners. In the case of gcForest, we use the implementation [5] of gcForest proposed by Zhou et al [6]. As we use discretized datasets, the reported results can be slightly different from the results reported in the mentioned papers.

Among the deep neural network models, The first deep neural network (DNN1) has one hidden layer and the second deep neural network (DNN2) has two hidden layers and DNN_N where we incorporated N hidden layers. In the case of DNN_N, N represents the number of layers equal to the number of layers determined by our proposed model DAC for each dataset. DNN1 consists of one hidden layer with 256 nodes in the layer with a dropout of 0.3 and ReLU activation function. In the output layer, we use the softmax activation function. We use the Adam optimizer with a learning rate of .0001. In DNN2 and DNN_N we use the same parameters with additional hidden layers with 256 nodes. With DNN1 in DNN2, one more hidden layer is added, and with DNN1 in DNN_N, N-1 more hidden layers are added. For each dataset, we utilise a different number of layers in the DNN_N model, which is dataset-specific and the number of layers are

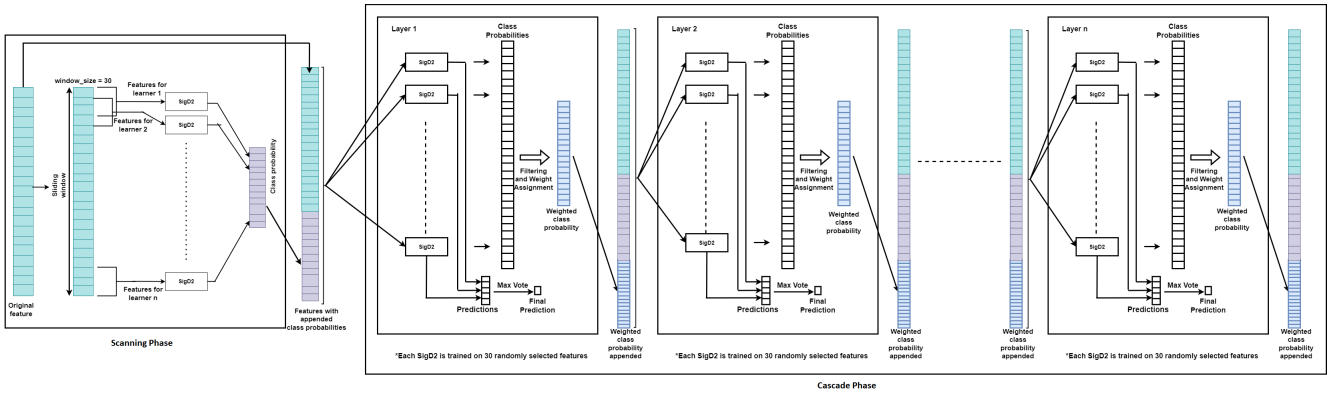

Fig. 4: Architecture of Deep associative Classifier

determined by our proposed model. We execute the model for 100 epochs for each dataset. We take these models as a general representation of the deep neural network family. With different combination of hyper-parameters for the deep neural network architectures, the performance may outperform the classifiers mentioned in this paper including our proposed model. But the hyper-parameter tuning might be dataset-specific and it is not an easy task to find proper set of hyper-parameters. We consider this as a limitation of the deep neural networks. For this reason, we do not perform hyper-parameter tuning for each dataset for deep neural networks rather use simple architectures for all the datasets.

C. Accuracy

A comparison of the accuracy of our model compared to other models is given in the Table I
If we do not consider dataset specific model DNN_N then in 5 out of 10 datasets, DAC outperforms other models. Among the rest 5, in breast dataset, DAC has the same performance as Random forest. In 2 of them DNN2 has better performance than other models. Only in 2 datasets, dataset specific model DNN_N outperforms all other models. In the Flare dataset, SigD2 outperforms other classifiers. In the glass dataset deep neural network model with two hidden layers performs better than other models. For the glass dataset, the accuracy of the associative classifier and the gcForest model is quite low as compared to the deep neural network models. In this dataset, the decision tree-based architecture gcForest, rule-based classifier SigD2 and our proposed model could not perform well. The core architecture of the deep neural network models might have an advantage over the rule-based models and the gcForest model for this dataset. In the case of the Anneal dataset, we can see that gcForest architecture outperforms all other models. In this dataset, the Random forest also has the same performance as the gcForest. As the base learner for both models is the decision tree, in the case of this dataset, the method of the decision tree has an advantage over the other classifiers. On average our model DAC outperforms all others.

To see the effect of filtering and weighting of the augmented feature, we tested the performance of our model by removing the filtering phase. We only generated the class probabilities and without further analysis, we added all the class probabilities in the dataset as features. Similarly, we also tested the performance of our model without the scanning phase. In this case, we used the same parameters and settings as mentioned above but just removed the scanning phase. The results of these tests are provided in Figure 5.

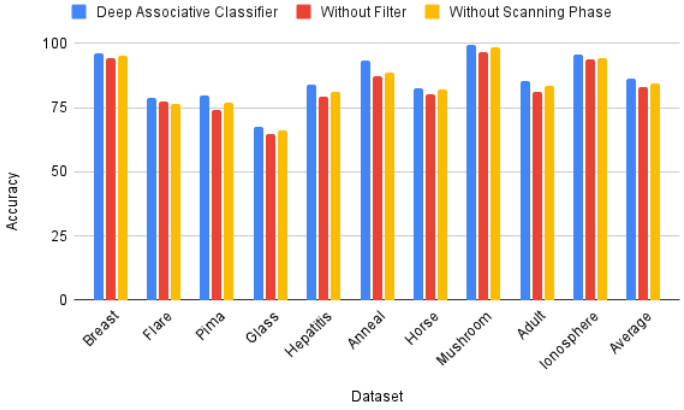

Fig. 5: Comparison of accuracy with and without different component of the proposed model

From Figure 5, we can see that removing the filtering or the weighting of the augmented feature components from the model has an impact on the performance of the model. Among the two modules, filtering has more impact on the results. In all the datasets except, Flare, removing the filtering step reduces the accuracy more than removing the scanning step. In the case of implementing the model without filtering, when the model generates class probabilities, we found many base learners produce the same class probabilities for all the instances. To make a good ensemble we know diversity among the base learners is a crucial factor. But without filtering many of the base learners produce duplicate results. With those results, in

Dataset	#cls	#rec	Feature vector size	Random Forest	gcForest	DNN1	DNN2	DNN_N	SigD2	DAC
Breast	2	699	18	96.42	95.76	94.43	96.14	**97.15**	94.29	96.42
Flare	9	1389	30	69.73	78.41	48.65	53.38	53.38	**84.39**	78.77
Pima	2	768	36	67.41	75.32	44.21	66.32	60.00	76.44	**79.87**
Glass	7	214	41	72.09	69.77	67.9	**88.89**	**88.89**	48.84	67.79
Hepatitis	2	155	54	78.38	80.65	35.42	81.25	**87.50**	81.54	82.26
Anneal	6	898	67	**98.89**	**98.89**	98.33	96.74	96.74	92.22	93.33
Horse	2	368	83	75.67	78.38	76.82	77.03	79.70	75.68	**82.43**
Mushroom	2	8124	88	69.51	99.17	78.27	79.89	79.60	99.03	**99.37**
Adult	2	48842	95	84.97	85.12	84.86	85.13	85.25	84.59	**85.41**
Ionosphere	2	336	155	95.78	94.37	85.34	**96.55**	**96.55**	90.14	95.78
Average				80.89	85.58	71.42	82.13	82.48	82.73	**86.14**

TABLE I. Accuracy of proposed method compared to other models

the subsequent layers, they produce even more duplicate base learners which hampers the model.

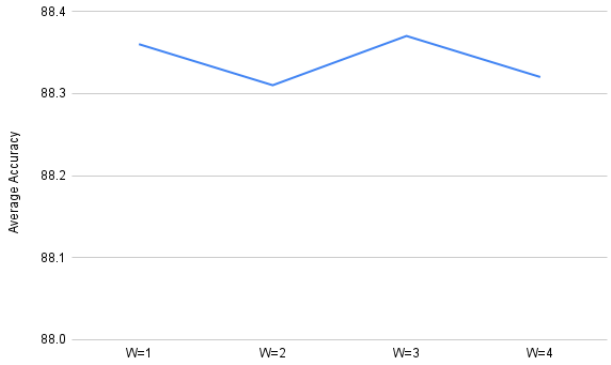

Fig. 6: Effect of number of windows in the scanning phase

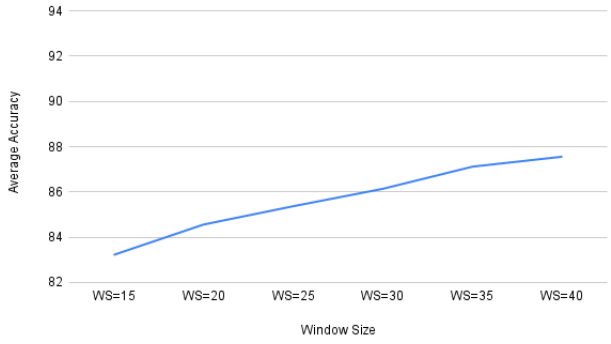

Fig. 7: Effect of different window size on accuracy

We analysed the performance of our model when using a different number of windows in the initial scanning step scrutinising the feature space. We experimented with two, three and four windows. The average accuracy for the different windows is provided in Figure 6.

In Figure 6, we can see, that increasing the number of windows in the scanning does not have a significant impact on the performance of the model in terms of accuracy. However, it impacts the execution time of the model. In our model, the layers and the base learners are trained sequentially. Thus increasing the number of windows increases the execution time of the model.

We also analyzed the different window sizes by experimenting with window sizes from 15 to 40 increasing by 5 at each step. We plot in Figure 7 the average accuracy of the 10 datasets with the increase of the window size. From the figure we can see that the accuracy increases with the increase in the window size. However, there is a compromise to make. As the window size increases, so does the feature vector output in each layer and therefore the memory requirement.

Finally, we analyse the effect of the number of base learners in each cascading layer. Since we use a majority voting scheme, the number of base learners can have a big impact on the performance of the model. This is a very important hyper-parameter for our model. In the case of the number of base learners, we perform the test varying the number of base learners from 25 to 200 increasing by 25 base learners in each step. Figure 8 shows the effect of the number of base learners on accuracy.

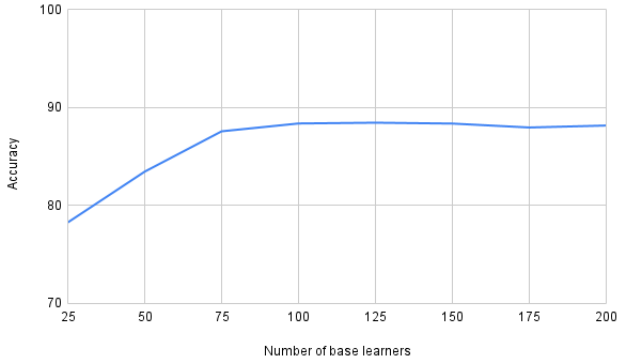

Fig. 8: Effect of number of base learners on accuracy

From Figure 8, we can see with the increase in the number of base learners, the accuracy increases sharply then plateaus beyond 100. In our model, we introduce filtering at each layer of the cascading layer which keeps only one of the base learners and increases redundancy by assigning some weight

Dataset	#cls	#rec	Feature vector Size	Random Forest	gcForest	DNN1	DNN2	DNN_N	SigD2	DAC
Breast	2	699	18	127	112	1736	1760	623	**92**	104
Flare	9	1389	30	132	132	1762	1777	1777	**106**	108
Pima	2	768	36	128	119	1774	1792	660	**101**	105
Glass	7	214	41	128	113	1719	1741	1741	110	**101**
Hepatitis	2	155	54	127	111	1786	1807	688	113	**104**
Anneal	6	898	67	129	117	1802	1827	1827	150	**116**
Horse	2	368	83	128	137	1813	1833	717	235	**112**
Mushroom	2	8124	88	142	**131**	1837	1859	731	993	162
Adult	2	48842	95	253	**374**	1871	1876	1992	825	492
Ionosphere	2	336	155	128	276	1838	1859	1859	2519	**113**

TABLE II. Memory requirement (in MB) of proposed method compared to other models

to that one learner. Thus when we increase the number of base learners, many redundant base learners are produced and most of them are filtered out. By increasing the number of base learners beyond 100 there is no significant change in the accuracy of the model.

D. Memory Requirement

To see how efficient our proposed model is in terms of memory requirement, we also measure the memory needed for each model. Deep neural networks require a high amount of memory for processing data at each layer. SigD2 also requires a high amount of memory with the increase of the feature vector size. We can see the memory requirements of the different models in Table II. To calculate the memory requirement for each of the model we used the python library *psutil*

From the table, we can observe the high memory requirement of deep neural networks. The memory requirement is also high for SigD2 for a dataset with a large feature vector. The gcForest architecture largely decreases the memory requirements for all types of datasets. Our proposed model further decreases the memory requirements. In the case of a dataset having a very small number of features, SigD2 requires less memory than our proposed model. But when the feature vector size increases, we can see a decrease in the memory requirement by our model in contrast to SigD2. Our model also shows competitive performance in comparison with random forest.

To get a better understanding of the memory requirement of our model, we tried different window sizes since SigD2 is sensitive to the size of feature vectors. Thus we tested different window sizes from 15 to 40 increasing by 5 at each step. Figure 9 shows the average memory requirement by our proposed model as we increase the window size. By analyzing Figure 7 and Figure 9, we can see the improvement in the accuracy and higher memory requirement with the increased window size. After window size 30 though the accuracy increases but the memory requirement also rises and the trend of increasing memory requirement is much higher than the accuracy. For this reason, window size 30 can be considered as an optimum window size for using our proposed model.

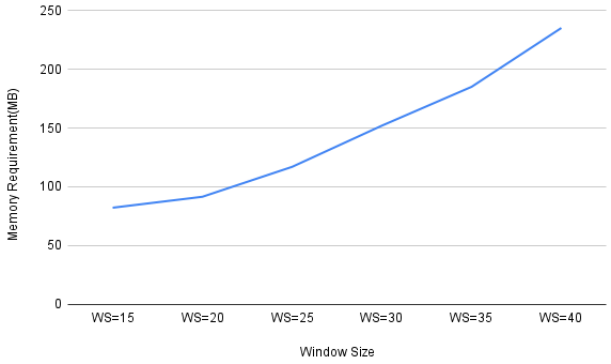

Fig. 9: Effect of window size in the memory requirement

E. Number of hyper-parameter and tuning feasibility

In previous sections, we mentioned, that deep neural network models have a complicated hyper-parameter tuning process and they differ for different datasets, In our experiment, we used a simple version of the deep learning model and we used the same model for all the datasets, these models of deep neural networks could not perform well in most of the datasets. But with proper hyper-parameters, deep neural networks might perform better. Most of the time the hyper-parameter tuning for deep neural networks is dataset-specific. That is, for each of the datasets the hyper-parameters could be different. Tuning the hyper-parameters is a tedious procedure. Moreover, in the case of deep neural network models, it is very hard to understand the effect of each hyper-parameter on the performance of the model.

On the other hand, gcForest architecture requires fewer hyper-parameters than the deep neural network models. Still, several hyper-parameters are present in their model. For the experiment, the authors found a setting where they showed their model achieved a better performance on different types of datasets, There can be some other settings of the hyper-parameter by which the performance of their model can be further improved. In their case, tuning 4 hyper-parameters to find a suitable setting is also a complicated and time-consuming process and like deep neural networks, there is no way to assume the hyper-parameters without testing. In our

model, we further reduced the number of hyper-parameters. We have two hyper-parameters which are the number of base learners in each of the cascading layers, and the window size. The number of layers is determined automatically. With experiments, we found a window size 30 is a very good choice for our model as decreasing from 30 decreases the accuracy of the model. Increasing the window size highly increases the memory requirement which we can see from Figure 7 and Figure 9. Still, for some other datasets, different window sizes may improve the performance of the model. Table III shows the number of hyper-parameter in the case of the deep neural network models, gcForest and our proposed model.

Deep neural networks	gcForest	DAC
• Activation functions • Number of hidden layer • Number of Nodes in hidden layer • Number Feature maps • Kernel size • Learning rate • Dropout ratio • Momentum • weight regularization penalty • Weight initialization • Batch size	• Number of Forests • Number of Trees in each forest • Tree growth • Sliding window sizes (3 different sizes)	• Number of base learners • Window size

TABLE III. Hyper-parameters

Considering the complexity of tuning hyper-parameters for deep neural networks and gcForest, changing the hyper-parameters for our proposed model is quite an easy task. In the case of our model, if we increase the number of base learners at each layer, the effect of the change that is whether there is an improvement in the performance can be directly observed. In the case of changing the window size, the change in the accuracy and memory requirements is also directly discernable from the result. Decreasing the window size makes the model faster but decreases the accuracy, and increasing the window size requires much memory while still improving the accuracy. By changing the window size it is easy to determine a good window size for a dataset which is an easier task than testing the model with different settings of many hyper-parameters in the case of deep neural networks and gcForest.

F. Statistical Analysis

Finally, we conducted a statistical analysis to show the significance of the improvements in our proposed model. For this, we performed the student paired t-test with the null hypothesis, that the improvement in the accuracy of our model is not significant. We run the whole experiment 20 times and calculated the average accuracy for each model. In this case, we used a window size of 30 and 100 base learners in each layer of cascading phase. Then we calculated the differences in accuracy by $\delta = \mathcal{M}_{DAC}(test\_data) - \mathcal{M}_i(test\_data)$. Here $\mathcal{M}_{DAC}$ is our proposed model and the subscript i stands for the other models. Running the models on test data provides the accuracy of the models and δ is the difference in the accuracy. Then with the differences, we performed the paired t-test which is inspired by the work of Henry et al. [23]. We used their model to calculate p-values for each of the mentioned classification models against our proposed model. The calculated p-values are provided in Table IV.

Algorithm	p-value
DAC vs Random Forest	1.2543e-21
DAC vs gcForest	0.0165
DAC vs SigD2	1.45335e-06
DAC vs DNN1	4.5374e-26
DAC vs DNN2	0.0009
DAC vs DNN_N	0.0002

TABLE IV. Statistical result

From Table IV we can see the p-value for each pair is very low. Thus we can reject the null hypothesis and conclude that the improvement in accuracy of our proposed model is significant.

V. Perspectives for Future Work

The structure of individual layers of our deep architecture is consistent and the number of layers is determined automatically by adding layers so long as the accuracy is improved. Once the accuracy of the last layer declines, adding layers stops. The hyper-parameters remain consistent among layers. However, once the accuracy declines, instead of stopping, the number of base learners and the window size (i.e. the number of features per base learner) do not need to remain the same for subsequent layers and could be tuned again to improve the accuracy of the last layer if a decline is noted. It is worth investigating hyper-parameter auto-tuning per additional layer to insure improvement of accuracy while increasing the depth of the model.

In the cascading step, we employ a filtering process to ensure that identical class probabilities are not appended more than once in the output feature vector. Our current filtering strategy simply avoids repetition by multiplying the probabilities by a weight equal to the count of repetitions of identical predictions. The performance of our model could be improved by exploring other filtering methods that still ensure diversity in the subsequent ensemble and safeguard the feature vector size in the layer output. Moreover, the random sampling of features for the base learners in the layer ensemble during the cascading step makes the model less stable across different runs. A better strategy of feature sampling that guarantees diversity and coverage could not only ensure stability but also eliminate the need to select the size of the ensemble as a hyper-parameter but instead automatically determine this size per layer.

SigD2 was shown to outperform SVM and decision trees, but more importantly other rule-based classifiers such as Ripper [22] or other associative classifiers. This was the main reason for selecting it as a base learner in our ensembles. However, SigD2 has drawbacks, particularly its limitations vis-à-vis large dimensional data. This is not a significant restriction in our current model as tested, however, this constitutes

a possible obstacle for hyper-parameter tuning (i.e. finding an optimal window size beyond 30). Exploring other rule-based classifiers as base learners or directly addressing the shortcomings of SigD2 could be an interesting open problem to address.

Another important direction to explore is to investigate how we can exploit the interpretability of the rule-based classifier, SigD2, to make DAC explainable. Understanding the important features in a decision is tricky with a cascade of layers each with an ensemble of deciders voting by majority. Moreover, the salient features composing the vectors in each layer are made up not only of the initial input features but also collected weighted probabilities from the ensemble of the previous layers. Finding the salient features is a problem in itself but translating those back to the original features is another open problem worth investigating.

VI. Conclusion

In this paper we present an attempt at exploit one of the main reasons behind the success of deep learning: the layered and deep representation learning. Our proposal expands on the idea of the gcForest by using a rule-based classifier, SigD2, as a base learner, introduces a filtering process to tackle diversity in ensembles, and further reducing hyper-parameters. One of the main questions we try to answer through this paper is *Can we have a deep architecture with fewer hyper-parameters, low memory requirement and exploit a layer-wise data processing architecture of deep neural networks while making use of a rule-based base learner?*. To answer this question and overcome some of the limitations of deep neural networks, gcForest, and some hindrances of associative classifiers, we designed a deep associative classifier architecture using SigD2 as a base learner.

We compared the performance of our approach with the other state-of-the-art models in terms of accuracy, memory requirements and hyper-parameter tuning feasibility on 10 different UCI datasets. We explored different sampling and tuning methods and added a filtering phase along with scanning and cascading phase to the ensemble architecture to improve the accuracy of our proposed model. From our experiments, we show that our model performs better than other existing models including gcForest and deep neural networks for different datasets not only in terms of accuracy but also in terms of memory needs. We also show that our model reduces the memory requirement to a great extent in comparison to deep neural networks and SigD2 which makes our model suitable to run in low-resource hardware. Our proposed model has only two hyper-parameters to tune as compared to gcForest which has four to tune and deep neural networks which have several hyper-parameters. Further, our model achieves excellent performance across various datasets in terms of accuracy.

There are several avenues for future research in the context of our proposed model which we came across while implementing and testing our model. We have highlighted some of them that we want to explore in future studies.

VII. Acknowledgement

Osmar Zaiane gratefully acknowledges funding from the Canada CIFAR AI Chairs Program, the Alberta Machine Intelligence Institute (Amii), and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] Jundong Li and Osmar R. Zaiane, "Exploiting statistically significant dependent rules for associative classification." Intelligent data analysis 21.5 (2017): 1155-1172.

[2] Wilhelmiina Wilhelmiina and Matti Nykänen. "Efficient discovery of statistically significant association rules." 2008 Eighth IEEE international conference on data mining. IEEE, 2008.

[3] Nitakshi Sood and Osmar Zaiane. "Building a competitive associative classifier." International Conference on Big Data Analytics and Knowledge Discovery. Springer, Cham, 2020.

[4] Xue Ying, "An overview of overfitting and its solutions." Journal of Physics: Conference Series. Vol. 1168. No. 2. IOP Publishing, 2019.

[5] LAMDA-NJU, Deep-Forest, GitHub repository, https://github.com/LAMDA-NJU/Deep-Forest

[6] Zhou, Zhi-Hua, and Ji Feng. "Deep forest." National Science Review 6.1 (2019): 74-86.

[7] Omer Sagi and Lior Rokach. "Ensemble learning: A survey." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8.4 (2018): e1249.

[8] Dietterich, Thomas G. "Ensemble learning." The handbook of brain theory and neural networks 2.1 (2002): 110-125.

[9] Kamran Kowsari, et al. "RMLD: Random multimodel deep learning for classification." Proceedings of the 2nd International Conference on Information System and Data Mining. 2018.

[10] Dheeru Dua and Casey Graff: UCI machine learning repository (2017), http://archive.ics. uci.edu/ml

[11] Frans Coenen, The lucs-kdd software library, 2004. [Online]. Available: https://cgi.csc.liv.ac.uk/frans/KDD/Software/LUCS-KDD-DN/lucs-kdd_DN.html

[12] Bing Liu and Wynne Hsu and Yiming Ma, "Integrating classification and association rule mining", in International Conference on Knowledge Discovery and Data Mining, 1998.

[13] Wenmin Li and Jiawei Han and Jian Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in IEEE International Conference on Data Mining, ICDM, 2001, pp. 369–376.

[14] Maria-Luiza Antonie and Osmar R. Zaiane. "Text document categorization by term association." 2002 IEEE International Conference on Data Mining, 2002. Proceedings.. IEEE, 2002.

[15] Ryutaro Tanno and Kai Arulkumaran and Daniel Alexander and Antonio Criminisi and Aditya Nori, "Adaptive neural trees." International Conference on Machine Learning. PMLR, 2019.

[16] Liang Sun, et al. "Adaptive feature selection guided deep forest for covid-19 classification with chest ct." IEEE Journal of Biomedical and Health Informatics 24.10 (2020): 2798-2805.

[17] Rakesh Agrawal and Srikant Ramakrishnan. "Fast Algorithms for Mining Association Rules in Large Databases." Paper presented at the meeting of the Proceedings of the 20th International Conference on Very Large Data Bases, San Francisco, CA, USA, 1994.

[18] Leo Breiman, "Bagging predictors," Machine learning, vol. 24, no. 2, pp. 123–140, 1996.

[19] Yoav Freund and Robert E. Schapire, "Experiments with a new boosting algorithm," in International Conference on Machine Learning, vol. 96, 1996, pp. 148–156.

[20] Karthik R. Bandi and Sargur N. Srihari. "Writer demographic classification using bagging and boosting." Proc. 12th Int. Graphonomics Society Conference. 2005.

[21] Joseph Prusa and Taghi M. Khoshgoftaar and Amri Napolitano, "Utilizing ensemble, data sampling and feature selection techniques for improving classification performance on tweet sentiment data." In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pp. 535-542. IEEE, 2015.

[22] William W. Cohen, Fast Effective Rule Induction, Twelfth International Conference on Machine Learning, 1995.

[23] Henry Hsu, and Peter A. Lachenbruch. "Paired t test." Wiley StatsRef: statistics reference online (2014).