A Reservoir Computing Scheme for Multi-class Classification

Nolan J. Coble and Ning Yu
State University of New York The College at Brockport
Brockport, New York, USA
{ncobl1,nyu}@brockport.edu

ABSTRACT

A recent addition to the field of neural networks is a framework for recurrent neural networks (RNNs) called reservoir computing. Reservoir computing has proven successful in predicting dynamical systems, but little attention has been given to its possible use as a classification method. Literature has been written specifically in regard to reservoir computing for binary classification, but few papers have specifically been written about multi-class classification. This article aims to provide a generic scheme for multi-class classification based on reservoir computing. The comparable performance has shown that our proposed scheme as an alternative classifier can catch up with and even outstrip the performance of some traditional classifiers such as naïve Bayesian, decision trees, random forest, and neural network over several data sets.

CCS CONCEPTS

• **Computing Methodologies** → *Expert Systems*; • **Applied Computing** → *Computers in other systems.*

KEYWORDS

Reservoir Computing, Multi-class Classification, Recurrent Neural Network, Deep Neural Network

ACM Reference Format:
Nolan J. Coble and Ning Yu. 2020. A Reservoir Computing Scheme for Multi-class Classification. In *2020 ACM Southeast Conference (ACMSE 2020), April 2–4, 2020, Tampa, FL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3374135.3385265

1 INTRODUCTION

Since the Recurrent Neural Network (RNN) model was discovered and used for pattern recognition in 1980s, it has attested as an effective approach for various applications. In the last decade, reservoir computing, a special type of RNN, has gained much attention for its simple training process and the computing capability to handle temporal data [16].

The reservoir computing (RC) framework is also referred to as a reservoir computer. It usually consists of three layers: an input layer, reservoir layer, and output layer as depicted in Figure 1. The reservoir layer in particular houses an RNN, albeit in a different manner than traditional RNN usages. Simply put, a traditional RNN approach involves inputting a training vector, readjusting all input, perceptron, and output weights, then repeating the process until all training data has been processed. Reservoir computing greatly reduces the time complexity of a traditional RNN by randomizing the input and perceptron weights, while only training the output weights [9].

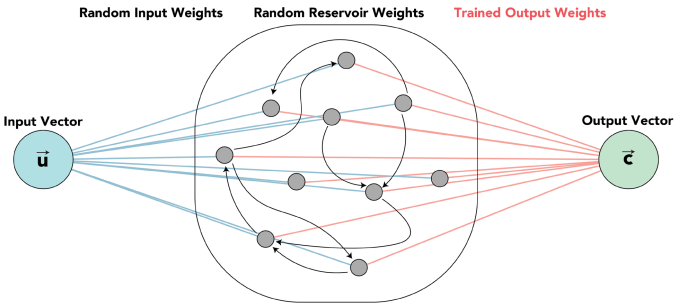

Figure 1: Illustration of Reservoir Computing

The Reservoir Computing framework has been known for over a decade as a state-of-the-art paradigm for the design of RNN. Among the models of RC instances, the echo state network (ESN) represents a type of the most widely known schemes with a strong theoretical support and many successful applications [5]. Jaeger and Haas first invented echo state networks (ESN) in 2004 for time series prediction in wireless communication channels [7]. ESN were designed to face with the challenges of RNNs by reducing the training time of artificial neural networks and avoiding the vanishing gradient problem [5]. The hidden layers were initialized randomly and constituted the reservoir, in which each neuron created its own nonlinear activation of the incoming signal, the inter-connected weights and the input weights were not learned through gradient descent, and only the output weights were tuned by using a learning algorithm such as logistic or Ridge classifier [6].

Fawaz et al. presented an empirical study of the most recent DNN architectures for Time Series Classification (TSC) and gave an overview of the most successful deep learning applications in various time series domains including the RC under a unified taxonomy of DNNs for TSC [3]. V. Krylov and S. Krylov applied a RC framework to binary classification [9]. Notably, they made the decision to represent two distinct classes with the vectors (1, 0) and (0, 1). Though this should have improved prediction result over using a labeling scheme of 0 and 1, their results were less than fruitful. However, it has shown that RC is a potentially promising candidate for generic classification. Schaetii et al. investigated RC when applied to MNIST digit recognition [14]. Although not generic classification, their method verifies the ability of RC to classify with

ACMSE 2020, April 2–4, 2020, Tampa, FL, USA

ACM ISBN 978-1-4503-7105-6/20/03…$15.00
https://doi.org/10.1145/3374135.3385265

a high success rate. This temporalization method along with the RC framework is applied to the MNIST handwritten digit dataset. Chouikhi et al. studied the application of a reservoir computer as an autoencoder for aiding in time-series classification problems [2]. Their autoencoder method followed exactly as in typical RC frameworks, but the objective of their work was not to train an output layer to achieve prediction capabilities. Rather, their goal was to collect reservoir states and pass those states into a classifier. In terms of RC terminology, they generated reservoir states for each step of time-series data and used these reservoir states to represent the time-series data. Due to the attributes of the nonlinearity and the echo-state property of the reservoir layer, their RC autoencoder was able to capture and represent hidden features of data. To extract even more hidden features, they also implemented a multi-layer reservoir system. In this framework there were multiple reservoirs connected together. The output from one reservoir was directly input to another reservoir, and so forth for as many layers as desired. Results for their autoencoder were promising when compared to an extreme learning machine (ELM) autoencoder. Ma et al. introduced a deep echo state network method to improve reservoir computing's ability to capture multi-scale dynamics of time-series data [12]. Multi-layer RC frameworks have been studied before, but these systems simply apply multiple reservoirs in a row to nonlinearly transform time series data. Multiple reservoirs were used in a row, but the output from one reservoir was passed through an autoencoder before entering the next reservoir. The additional nonlinear transformation allowed this RC to capture dynamics on multiple timescales, an important requirement for reservoir computing to become realistically useful. Their system greatly improved time-series prediction over a single ESN framework and a general multi-layer ESN framework. In a study of Mackey-Glass system predictions (a common chaotic system used in prediction experiments), their system improved predictions by an order of magnitude in this particular instance while in other instances, they found only modest improvement. Nonetheless, their method provides an interesting combination of the simplicity of training and predicting for reservoir computers and the multi-scale capabilities of traditional deep-learning methods.

In addition to these RC applications in RNN, Tanaka et al. importantly introduced the various physical implementations of reservoir computing [15]. Any high dimensional and nonlinear system with short-term memory could serve as a physical implementation for reservoir computing [13]. Many types of physical reservoir computers were being studied, including coupled-oscillators and neuromemristive circuits.

Despite the numerous research works devoted to reservoir computing, few have focused on applications of reservoir computing to classification problems. Even among those works, the problem of generic multi-class classification has not been handled to date. In this article, we try to fit the niche and propose a generic multi-class classifier. In the following sections, Section 2 describes the methods used in this article. Section 3 illustrates the performance of the proposed generic scheme comparing with other common classifiers. At last, Section 4 makes a discussion and concludes this article.

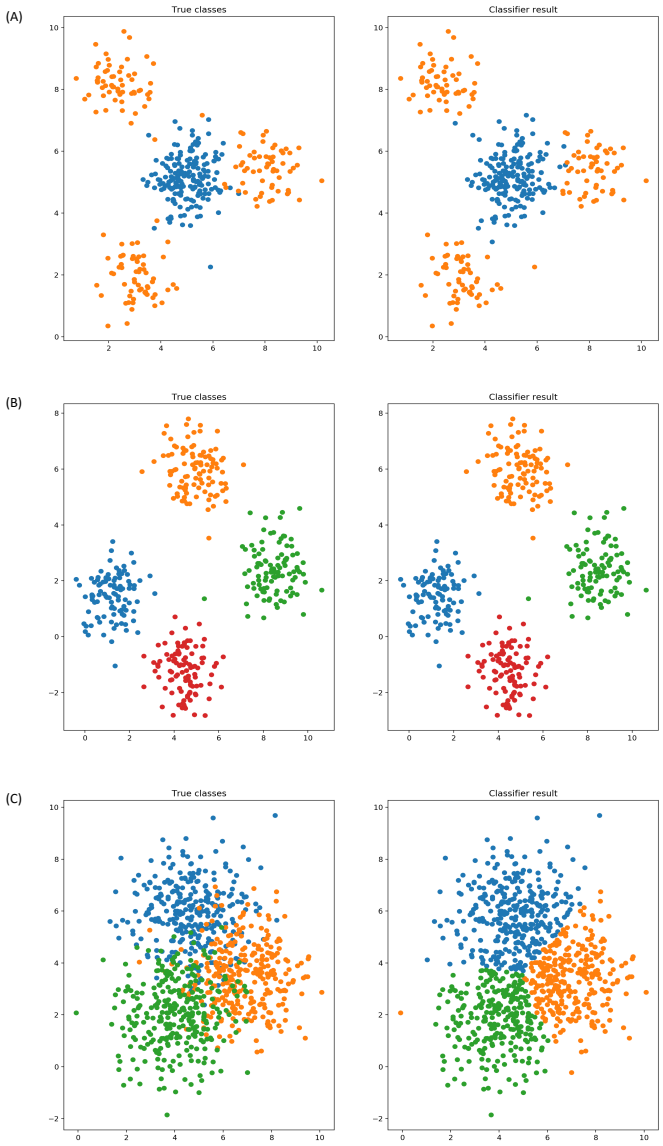

Figure 2: True Classes and Classifier Predictions for (A) Dtree Data Set, (B) Multivar Data Set, and (C) Rforest Data Set

2 METHODS

The training input for classification is a matrix of (# of training vectors) × (# of properties). There are many ways to represent the class that a vector may belong to. Common in other classification methods is assigning classes as 0, 1, 2, etc., but this labeling scheme proves to be ineffective for reservoir computing. A labeling scheme of (1, 0) and (0, 1) was used to represent two distinct classes in [9]. This method allowed the reservoir computer to better differentiate between classes. In this article we proposed an extension of the RC method as: n different classes correspond to orthogonal unit vectors of length n. Namely, we will use the rows of I_n the identity matrix

Table 1: Average RC Prediction Performance

Data Set	Mean precision	Mean recall	Mean F1-score
Dtree	0.965	0.970	0.970
Multivar	1.000	1.000	1.000
Rforest	0.870	0.870	0.870
Pulsar-star	0.965	0.900	0.930
Banknote	0.985	0.980	0.980
Iris	0.977	0.973	0.977
Shuttle	0.764	0.676	0.687
Seeds	0.927	0.930	0.927

$precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$, $F1 = 2 \times \frac{precision \times recall}{precision+recall}$

of dimension n to represent the different classes. At this point, we have a matrix of input solutions and each training input has an associate class vector. The i-th input vector will be referenced as $\vec{u}(i)$.

The input layer consists of the input weights to each reservoir perceptron. It is a randomly generated matrix of size (# of properties) × (# of perceptrons), with values uniformly distributed between 0 and δ, where δ is a scaling factor. This scaling factor allows us to adjust the reservoir computing method to training data of various sizes. The input layer matrix is denoted as W_{in}.

The reservoir layer is where the actual RNN is held. The reservoir can contain any number of perceptrons, though typical applications use a value between 500 to 1000. For classification applications, it appears as though smaller values (around 50-200) work best. The current, i-th, state of the reservoir is described as a length (# of perceptrons) vector, referenced as $\vec{r}(i)$. The perceptron weights are described by a connectivity/adjacency matrix, A. This matrix is an Erdös-Rényi randomly generated matrix. The eigenvalues of the matrix are normalized, and scaled by a factor, ρ, known as the spectral radius. The spectral radius of the connectivity matrix determines how the reservoir adapts to changing dynamics. Usually, a spectral radius of around 0.9 is used in dynamics-prediction applications, although finding an optimal spectral radius has not been directly studied. The activation function for the reservoir is given in Equation 1.

$$\vec{r}(i) = \tanh[\vec{r}(i-1) \times A + \vec{u}(i) \times W_{in}], \tag{1}$$

where $\vec{r}(i-1) \times A$ represents the reservoir feedback and is what allows the reservoir to adapt to new input vectors. The $\vec{u}(i) \times W_{in}$ term represents the input to the reservoir. The input term and activation function used varies greatly between different implementations of reservoir computing. A similar linear mapping with input weights was given in the literature [9] and the activation function used was the hyperbolic tangent.

The output layer is how a prediction is obtained from the reservoir. The output layer is a (# of perceptrons) ×(# of classes) matrix, referred to as W_{out}. The output is obtained by Equation 2.

$$output = \vec{r}(i) \times W_{out} \tag{2}$$

The process of reservoir computing involves three stages: (1) listening, (2) training, and (3) predicting.

Table 2: Comparison for Dtree Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	**1.00**	0.97	**0.99**
Reservoir	0.93	**1.00**	0.97
Decision Tree	0.87	0.87	0.87
Random Forest	0.97	0.90	0.93
Neural Network	0.90	1.00	0.95

Class 1			
Method	Precision	Recall	F1-score
Bayesian	0.97	**1.00**	**0.99**
Reservoir	**1.00**	0.94	0.97
Decision Tree	1.00	0.94	0.97
Random Forest	1.00	0.94	0.97
Neural Network	1.00	0.89	0.94

In the listening phase, training data is input to the reservoir one by one to generate associated reservoir states. Given one piece of training data, $\vec{u}(i)$, we generate an associated reservoir state, $\vec{r}(i)$, via the activation function given in Equation 1. Note that a new reservoir state depends on the current input data, as well as the previous reservoir state. This feedback term is what allows the reservoir states to capture the dynamics, or properties, of the input data stream. After all of the training data has been input to the reservoir, we are left with a few things: a matrix of input data $\vec{u}$, a matrix of associated classes $\vec{c}$, a matrix of reservoir states $\vec{r}$, and the final state of the reservoir, $\vec{r}(end)$. At this point, the matrix of input data serves no purpose, but helpful to recall its presence. It should also be noted that $\vec{r}(end)$ is certainly contained within the reservoir state matrix (it is the last row of this matrix), but we will find that this vector itself is important during the predicting phase.

The training phase is the result of a few simple logical arguments. We have a matrix of reservoir states that supposedly capture the dynamics of our input data, and we have a matrix of class vectors that are associated with the input data. We would expect the relation between the class vectors and the reservoir states as the following Equation 3.

$$\vec{r} \times W_{out} = \vec{c} \tag{3}$$

If our reservoir has, in fact, captured the properties of the training data, then when we retrieve information from the reservoir, we would expect to get our matrix of class vectors. We do not know the output matrix, but we know the reservoir states and the class matrix. We can find the output matrix via Equation 4. By using the Moore-Penrose pseudoinverse, we are effectively minimizing the error in producing the class matrix from our reservoir states.

$$W_{out} = pseudoinverse(\vec{r}) \times \vec{c} \tag{4}$$

The predicting phase is relatively straightforward. Given a new input with unknown class, we generate a new reservoir state via Equation 5. From this new reservoir state, we obtain the predicted class via Equation 6.

$$\vec{r}(new) = \tanh[\vec{r}(end) \times A + \vec{u}(input) \times W_{in}] \tag{5}$$

Table 3: Comparison for Multivar Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	**1.00**	**1.00**	**1.00**
Decision Tree	1.00	1.00	1.00
Random Forest	1.00	1.00	1.00
Neural Network	1.00	1.00	1.00

Class 1			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	**1.00**	**1.00**	**1.00**
Decision Tree	0.96	1.00	0.98
Random Forest	1.00	1.00	1.00
Neural Network	1.00	1.00	1.00

Class 2			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	**1.00**	**1.00**	**1.00**
Decision Tree	1.00	0.96	0.98
Random Forest	1.00	1.00	1.00
Neural Network	1.00	1.00	1.00

Class 3			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	**1.00**	**1.00**	**1.00**
Decision Tree	1.00	1.00	1.00
Random Forest	1.00	1.00	1.00
Neural Network	1.00	1.00	1.00

Table 4: Comparison for Rforest Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	**0.97**	**0.87**	**0.91**
Reservoir	0.91	0.86	0.88
Decision Tree	0.96	0.84	0.89
Random Forest	0.92	0.85	0.88
Neural Network	0.91	0.89	0.90

Class 1			
Method	Precision	Recall	F1-score
Bayesian	0.84	**0.88**	**0.86**
Reservoir	0.84	0.83	0.84
Decision Tree	0.83	0.86	0.85
Random Forest	**0.86**	0.84	0.85
Neural Network	0.84	0.83	0.83

Class 2			
Method	Precision	Recall	F1-score
Bayesian	0.84	0.91	0.87
Reservoir	**0.86**	**0.92**	**0.89**
Decision Tree	0.82	0.91	0.86
Random Forest	0.84	0.92	0.88
Neural Network	0.89	0.92	0.90

$$predictedclass = \vec{r}(new) \times W_{out} \tag{6}$$

It is important to note that we do not keep the new reservoir state after a prediction has been made. This is starkly different from predicting dynamical data using reservoir computing. We do not want to update the last reservoir state during classification, since we do not know whether or not the prediction that was made is correct. If the prediction was correct, then the reservoir performance would improve slightly, but even a single incorrect prediction would drastically affect results [9]. For this reason, we do not update the reservoir state after a prediction has been made.

3 RESULTS

All reservoir computing classification results are summarized in Table 1. The reservoir computer classification scheme was tested on eight data sets. Sample data was split so that 75% of data was used for training, with the remaining 25% reserved for testing/validation. An input scaling of 0.3 and a spectral radius of 0.9 were used. The reservoir size was 50 perceptrons. Performance in Table 1 was measured as the mean precision, recall, and F1-score across the different classes of each data set.

The proposed RC scheme was validated against four other well known classification methods: naïve Bayesian, decision tree, random forest, and a neural network. The naïve Bayesian classifier is a probabilistic model which is based on Bayes' rule for probabilities. Due to the probabilistic nature, the prediction results are the same every time the classifier is trained and run. The decision tree classifier is an entropy-based model. This model effectively classifies all of the training data in a tree model. Test instances then follow this tree model, to determine which class they belong to. The random forest model uses an ensemble of decision trees. In many applications, the random forest model can drastically increase classification accuracy. The neural network model is a widely used classification method. A 3-layer neural network model was used for this classification test. Each layer consists of 50 perceptrons. The hidden layer activation function was a rectified linear function, and the output layer activation was softmax. The model was trained over 2000 iterations. Each of the 5 classifiers were tested on the 8 data sets. Tables 2-9 summarize the results.

Dtree data consists of 360 instances, each with 2 attributes [8]. These instances are labeled in two equally sized classes. Comparison of true classes and reservoir computer classifier results is given in Figure 2 (A). The classifier predicted with strong performance, missing only fringe data points. Table 2 shows the comparison of the RC classifier against the other classifiers.

Table 5: Comparison for Pulsar-star Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	0.98	0.96	0.97
Reservoir	0.98	**1.00**	**0.99**
Decision Tree	**0.99**	0.99	0.99
Random Forest	0.98	0.99	0.98
Neural Network	0.98	1.00	0.99

Class 1			
Method	Precision	Recall	F1-score
Bayesian	0.67	0.83	0.74
Reservoir	**0.95**	0.80	0.87
Decision Tree	0.93	**0.85**	**0.89**
Random Forest	0.94	0.82	0.87
Neural Network	0.98	0.76	0.85

Multivar data consists of 400 instances, each with two attributes [8]. These instances are labeled in four equally sized classes, all linearly separable. Comparison of true classes and reservoir computer classifier results is given in Figure 2 (B). The classifier predicted without error. Table 3 shows the comparison of the RC classifier against the other classifiers.

Rforest data consists of 900 instances, each with two attributes [8]. These instances are labeled in three equally sized classes. All three classes overlap with each other, reducing accuracy of all classification methods. Comparison of true classes and reservoir computer classifier results is given in Figure 2 (C). Table 4 shows the comparison of the RC classifier against the other classifiers.

Pulsar-stars consists of 17898 instances, each with 8 attributes [11]. These instances are labeled in two unequal classes. The 8 attributes are statistics obtained from integrated pulse profiles and DM-SNR curves for candidate pulsars. Each instance is labeled as either yes, a pulsar star, or no, not a pulsar star. 16,259 instances are not pulsar stars and 1,639 instances are. This represents approximately 9% positive occurrences, a relatively unbalanced set to test binary classification. Table 5 shows the comparison of the RC classifier against the other classifiers. All classifiers were able to predict the larger class with high precision. The Bayesian classifier had the worst performance on the smaller class, while the other four classifiers performed on par with each other.

Banknote data consists of 1372 instances, each with 4 attributes [10]. These instances are labeled in two classes; 762 instances belong to class 0 and 610 instances belong to class 2. The 4 attributes are statistics obtained from wavelet transformed images of both real and fraudulent banknotes. Table 6 shows the comparison of the RC classifier against the other classifiers. The neural network classifier performed perfectly, with the RC classifier not far behind. The other methods showed only slightly less precision.

Iris data consists of 150 instances, each with 4 attributes [4]. These instances are labeled in three equal classes representing different iris species. The 4 attributes are various physical measurements of the iris plant. Class 0 is linearly separable from the other two, but class 1 and class 2 are not separable. Table 7 shows the comparison of the RC classifier against the other classifiers. All classifiers predicted Class 0 perfectly, but the RC classifier scored higher than the other classifiers for both remaining classes.

Table 6: Comparison for Banknote Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	0.89	0.88	0.89
Reservoir	0.97	1.00	0.98
Decision Tree	0.95	0.95	0.95
Random Forest	0.98	0.95	0.95
Neural Network	**1.00**	**1.00**	**1.00**

Class 1			
Method	Precision	Recall	F1-score
Bayesian	0.82	0.83	0.82
Reservoir	1.00	0.96	0.98
Decision Tree	0.94	0.94	0.94
Random Forest	0.94	0.97	0.96
Neural Network	**1.00**	**1.00**	**1.00**

Table 7: Comparison for Iris Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	**1.00**	**1.00**	**1.00**
Decision Tree	1.00	1.00	1.00
Random Forest	1.00	1.00	1.00
Neural Network	1.00	1.00	1.00

Class 1			
Method	Precision	Recall	F1-score
Bayesian	0.91	1.00	0.95
Reservoir	**0.93**	**1.00**	**0.97**
Decision Tree	0.92	0.86	0.89
Random Forest	0.87	0.93	0.90
Neural Network	0.93	0.93	0.93

Class 2			
Method	Precision	Recall	F1-score
Bayesian	1.00	0.90	0.95
Reservoir	**1.00**	**0.92**	**0.96**
Decision Tree	0.85	0.92	0.88
Random Forest	0.91	0.83	0.87
Neural Network	0.92	0.92	0.92

Shuttle data consists of 43,500 instances, each with 9 attributes [1]. These instances are labeled in 7 unequal classes. The 9 attributes are various numerical attributes. 34,108 (approximately 80%) instances correspond to class 1, which should be the most accurately predicted

[1] https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

Table 8: Comparison for Shuttle Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	0.99	0.68	0.81
Reservoir	0.99	0.99	0.99
Decision Tree	1.00	1.00	1.00
Random Forest	**1.00**	**1.00**	**1.00**
Neural Network	0.99	1.00	0.99

Class 1			
Method	Precision	Recall	F1-score
Bayesian	1.00	0.69	0.82
Reservoir	0.97	1.00	0.98
Decision Tree	1.00	1.00	1.00
Random Forest	**1.00**	**1.00**	**1.00**
Neural Network	0.99	1.00	0.99

Class 2			
Method	Precision	Recall	F1-score
Bayesian	0.02	1.00	0.04
Reservoir	0.00	0.00	0.00
Decision Tree	0.50	0.14	0.22
Random Forest	1.00	0.14	0.25
Neural Network	**1.00**	**0.43**	**0.60**

Class 3			
Method	Precision	Recall	F1-score
Bayesian	0.21	0.59	0.30
Reservoir	0.81	0.35	0.49
Decision Tree	0.79	0.89	0.84
Random Forest	1.00	0.11	0.20
Neural Network	**1.00**	**0.81**	**0.90**

Class 4			
Method	Precision	Recall	F1-score
Bayesian	0.44	0.80	0.57
Reservoir	0.98	0.89	0.93
Decision Tree	1.00	1.00	1.00
Random Forest	**1.00**	**1.00**	**1.00**
Neural Network	1.00	0.97	0.99

Class 5			
Method	Precision	Recall	F1-score
Bayesian	0.40	1.00	0.57
Reservoir	**1.00**	**0.50**	**0.67**
Decision Tree	0.00	0.00	0.00
Random Forest	0.00	0.00	0.00
Neural Network	1.00	0.50	0.67

Class 6			
Method	Precision	Recall	F1-score
Bayesian	0.00	1.00	0.01
Reservoir	**0.60**	**1.00**	**0.75**
Decision Tree	0.00	0.00	0.00
Random Forest	0.00	0.00	0.00
Neural Network	0.60	1.00	0.75

Table 9: Comparison for Seeds Data Set

Class 0			
Method	Precision	Recall	F1-score
Bayesian	0.93	0.93	**0.93**
Reservoir	**0.95**	0.90	0.92
Decision Tree	0.85	0.85	0.85
Random Forest	0.76	**0.95**	0.84
Neural Network	0.90	0.90	0.90

Class 1			
Method	Precision	Recall	F1-score
Bayesian	1.00	1.00	1.00
Reservoir	0.94	1.00	0.97
Decision Tree	1.00	1.00	1.00
Random Forest	**1.00**	**1.00**	**1.00**
Neural Network	0.94	1.00	0.97

Class 2			
Method	Precision	Recall	F1-score
Bayesian	0.92	**0.92**	**0.92**
Reservoir	0.89	0.89	0.89
Decision Tree	0.83	0.83	0.83
Random Forest	0.92	0.67	0.77
Neural Network	**0.94**	0.89	0.91

class. 6,748 correspond to class 4, 2,458 to class 0, 132 to class 3, 37 to class 2, 11 to class 6, and 6 to class 5. Due to number of instances, classes 2, 5, and 6 should be the most difficult for the classifiers to predict accurately. Table 8 shows the comparison of the RC classifier against the other classifiers. All of the models struggle with the small classes, but the neural network clearly performs the best. The reservoir computer is on par with the NN for most of the classes.

Seeds data consists of 210 instances, each with 7 attributes [1]. These instances are labeled in 3 equal classes. The 7 attributes represent geometric parameters of seeds of 3 different wheat varieties. Table 9 shows the comparison of the RC classifier against the other classifiers. All of the classifiers predicted relatively similar with each other, with the Bayesian classifier appearing to perform the most accurately.

The Python code of the RC scheme can be found in Github: https://github.com/ncoble98/reservoir_computing_classifier.git.

4 CONCLUSION

The results and validation show that the RC scheme as an alternative multi-class classifier can certainly compete with other classification methods and even outperforms other traditional machine learning methods over some data sets, without increase on computational complexity. There are several things to keep in mind about the prior results. The naïve Bayesian classifier—which is based on statistical arguments—and the decision tree classifier—which is based on entropy arguments—perform consistently each time they are run. The bases for their schemes is not random, and as a result the classifications will always produce the same results (aside from

the randomness of splitting the training data). This can be useful, but it also limits those methods from improvement. The random forest and reservoir computing methods can vary greatly with each run. This is a result of the inherent randomness of both schemes (a 'random' forest and a random reservoir).

Although reservoir computing may not have performed best in each trial, there are many factors that could affect this performance. As mentioned in previous sections, the input scaling factor and spectral radius of the reservoir will drastically change the outcome of results. We did not vary these values throughout the tests, mainly due to the lack of literature on how to optimize parameters. Further research could be done to optimize the input scaling and spectral radius for classification, or even for dynamical predictions.

In addition to optimizing parameters, there is some literature devoted to additions to the reservoir computing prediction scheme for dynamic systems, that could theoretically be extended to the classification use. A deep reservoir network scheme was described in [12] whereby reservoir states subsequently passed through an auto-encoder into another reservoir, and so forth. This method could be adapted to classification to some extent.

REFERENCES

[1] M. Charytanowicz, J. Niewczas, P. Kulczycki, P. A. Kowalski, S. Łukasik, and S. Żak. 2012. Seeds Dataset.

[2] N. Chouikhi, B. Ammar, and A. M. Alimi. 2018. Genesis of Basic and Multi-layer Echo State Network Recurrent Autoencoders for Efficient Data Representations. *arXiv preprint arXiv:1804.08996* (2018).

[3] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. 2019. Deep Learning for Time Series Classification: A Teview. *Data Mining and Knowledge Discovery* 33, 4 (2019), 917–963.

[4] R. A. Fisher. 1988. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Iris

[5] C. Gallicchio and A. Micheli. 2017. Deep Echo State Network DeepESN: A Brief Survey. *arXiv preprint arXiv:1712.04323* (2017).

[6] A. E. Hoerl and R. W. Kennard. 1970. Ridge Regression: Applications to Nonorthogonal Problems. *Technometrics* 12, 1 (1970), 69–82.

[7] H. Jaeger and H. Haas. 2004. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *science* 304, 5667 (2004), 78–80.

[8] P. Joshi. 2017. *Artificial Intelligence with Python.* Packt Publishing Ltd.

[9] V. Krylov and S. Krylov. 2018. Reservoir Computing Echo State Network Classifier Training. In *Journal of Physics: Conference Series*, Vol. 1117. IOP Publishing, Moscow, Russian Federation, 012005.

[10] V. Lohweg. 2013. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/banknote+authentication

[11] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. 2016. Fifty Years of Pulsar Candidate Selection: From Simple Filters to a New Principled Real-time Classification Approach. *Monthly Notices of the Royal Astronomical Society* 459, 1 (04 2016), 1104–1123. https://doi.org/10.1093/mnras/stw656 arXiv:https://academic.oup.com/mnras/article-pdf/459/1/1104/8115310/stw656.pdf

[12] Q. Ma, L. Shen, and G. W. Cottrell. 2017. Deep-esn: A Multiple Projection-encoding Hierarchical Reservoir Computing Framework. *arXiv preprint arXiv:1711.05255* (2017).

[13] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott. 2018. Model-free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach. *Physical review letters* 120, 2 (2018), 024102.

[14] N. Schaetti, M. Salomon, and R. Couturier. 2016. Echo State Networks-based Reservoir Computing for MNIST Handwritten Digits Recognition. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. IEEE, Paris, France, 484–491.

[15] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose. 2019. Recent Advances in Physical Reservoir Computing: A Review. *Neural Networks* (2019).

[16] A. Tong and G. Tanaka. 2018. Reservoir Computing with Untrained Convolutional Neural Networks for Image Recognition. In *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, Beijing, China, 1289–1294.