# Deep Ensemble Learning

A Major Qualifying Project Report:

Submitted to the Faculty

of the

**WORCESTER POLYTECHNIC INSTITUTE**

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:
Juan Luis Herrero Estrada
Everett Harding
Luke Buquicchio

Advisors:
Randy Paffenroth
Stephan Sturm

Sponsor:
Jörg Osterrieder, ZHAW

Date: March 2, 2018

# Contents

## Glossary

| | |
|---|---|
| MSE | Mean Squared Error. |
| Var | Variance. |

## Notation Dictionary

$\mathbb{R}^n$ The set of all n-dimensional vectors of real numbers.

$|S|$ Number of elements in the set S.

$\mathbf{x}$ Input vector such that $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ and $\mathbf{x} \in \mathbb{R}^n$.

$y$ Target value $y \in \mathbb{R}$ for some input $\mathbf{x} \in \mathbb{R}^n$.

$\hat{y}$ Approximated model output $\hat{y} \in \mathbb{R}$ for some input $\mathbf{x} \in \mathbb{R}^n$.

$w$ Weight value associated with some input or response.

$\hat{f}$ Approximated model mapping $\mathbf{x}$ to an output $\hat{y}$.

$p(A)$ Probability of event A.

$p(A|B)$ Probability of event A conditional on event B.

$\mathbb{E}[\mathbf{X}]$ Expected value of a random variable $\mathbf{X}$.

$\varepsilon$ Error calculated from some model.

$\sigma$ Activation function.

$\sigma^2$ Total variance of a distribution.

$\mathscr{D}$ Input dataset such $\mathscr{D} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p]$ where $\mathbf{x}_j \in \mathbb{R}^n$ and $\mathscr{D} \in \mathbb{R}^{n \times p}$.

$\mathscr{D}'$ New dataset created from the original $\mathscr{D}$ with the same dimensions.

$\Phi$ Combined model $\Phi : \mathbb{R}^n \to \mathbb{R}$ of all $M$ base learners $\phi$ in the ensemble where some arbitrary learner $\phi_m : \mathbb{R}^n \to \mathbb{R}$.

## List of Figures

## List of Tables

**Abstract**

This project combines feedforward neural networks (FFNN) with ensemble learning in a classification model applied to credit card defaults. To solve this problem, we generate FFNNs as the members of the ensemble. Each FFNN in the ensemble is built using different subsets of predictors as inputs and a separate predictor as the target (instead of the label). The output from the final hidden layer of each network is aggregated into a new dataset. A classification model then uses this new dataset to predict credit defaults. The classification model is evaluated using various metrics to show that an increase in ensemble size increases the consistency of the performance of the model.

# 1 Introduction

As availability of data in the modern world increases, it is vital that the tools used to analyze data are made more perceptive in order to filter out noise and discover the underlying trends in it. One particular application of finding trends in a dataset is classification. In classification every data point is labeled with a particular category and models are trained to predict the category of an unknown data point. A method for doing so that has found a high rate of success is ensemble learning. In ensemble learning, many models train on the same dataset and vote to determine the most likely category for a given data point. Another method that has found success in this area is a feedforward neural network (described in Section 2.1). This project attempts to improve on the performance of both by combining and adapting these two techniques.

In ensemble learning, one of the key features to the success of an ensemble is the diversity of the models in the ensemble. For example, a game of BINGO can be played with one card or multiple. When playing with one card there are 10 ways to get a traditional bingo on a given arrangement of numbers (5 rows, 5 columns, and 2 diagonals). With multiple cards, the number of ways to get bingo can increase. However, if the arrangement of the numbers on each additional card is identical, the number of ways to get bingo does not increase. The more differences that exist between the cards, the more likely the player is to get bingo. Similarly, the more differences that exist between the models in an ensemble, the more likely the ensemble is to correctly categorize a data point. This project attempts to promote high diversity in the ensemble by heavily injecting randomness into the creation of each member in the ensemble. The resultant level of diversity in the ensemble is designed to improve the performance of the ensemble.

An important property of neural networks, and the motivation for their use in this application, is neural networks do no assume a linear relationship between a data point and its corresponding category exists, giving them an advantage over other methods. When used in this ensemble, the inherent complexity of the networks, combined with the randomness of their generation allows for an even higher level of diversity between the models used in the ensemble.

Another important part of an ensemble is the voting scheme, or method of combining the individual results of each model in the ensemble. For example, one method might weight the prediction of each model equally and use the most common prediction as the result of the model. Another method could weight the predictions of each model according to how often that model's prediction aligned with the majority decision in the past, and combine the current votes according to those weights. Each of these methods are ways of aggregating the relationships that models discover between data points and their corresponding categories. While most voting schemes aggregate and determine a result in one step, this project uses two steps. The first step aggregates the learned relationships in each neural network into a new dataset. The second step treats this new aggregation as a transformation of the original dataset and performs a classification that learns relationships between the transformed data points and their categories.

The two-step aggregation of the ensemble results allows the framework described to be split into two parts: the ensemble and the aggregation (called an *adapter*). The training of the ensemble in this project is unsupervised, meaning the relationships within the data are discovered without giving the models the corresponding categories of each data point. The second step, the adapter, makes use of the relationships learned in the first (unsupervised) step to perform a traditional classification, which is a supervised learning technique. Supervised learning means the model is given the data points and the corresponding categories when it is trained. The implementation of the second step is independent from the implementation of the ensemble, meaning the results of the ensemble can be examined with multiple classification methods. This project examines the differences between using another feedforward neural network and a random forest model (described below) as the adapter step.

# 2 Background

This section explains the theory and implementation of the techniques used in Section 3.2 to correctly classify the data described in Section 3.1. The section presents an explanation of feedforward neural networks and how they are used as part of an ensemble of learners described in Section 2.4. Finally, metrics to evaluate both the feedforward neural networks and the ensemble of learners are detailed in this section.

## 2.1 Feedforward Neural Network

A feedforward neural network (FFNN) is a type of specific artificial neural network (ANN). An artificial neural network, a machine learning technique, discovers relationships in data without requiring specific knowledge of, or tailoring to, the domain of the data. The structure of an ANN mimics the human brain [1]. The basic unit of an ANN is called an artificial neuron. Given an input signal, it will send an output signal to the next neurons, depending on the magnitude of the input [1]. In the context of ANN's, artificial neurons are referred to as neurons for simplicity. Neurons are organized into groups called layers. In feedforward neural networks, the layers connect sequentially, the outputs from the neurons of one layer connecting to the inputs of the neurons in the next layer. The connections between the neurons in each layer are assigned weights which express the strength of the connection between the two neurons. The combination of layers and the weighted connections between them forms the network.

The neuron $N$ receives $n$ inputs and calculates a weighted sum $s$ of the inputs,

$$s = \mathbf{w}^T \mathbf{a} + b. \tag{1}$$

In equation (1), column vector $\mathbf{a}$ is the vector of inputs received by $N$ while the vector $\mathbf{w}$ is composed of all the weights associated with each input [2]. The $i$-th input $a_i$ is weighted by the $i$-th connection weight $w_i$. $b$ is a scalar known as the bias (not to be confused with bias error in Section 2.3) used for shifting the result from a function left or right. A nonlinear activation function $\sigma(s) = z$ produces an output value $z$, which in turn is input to the following neurons.

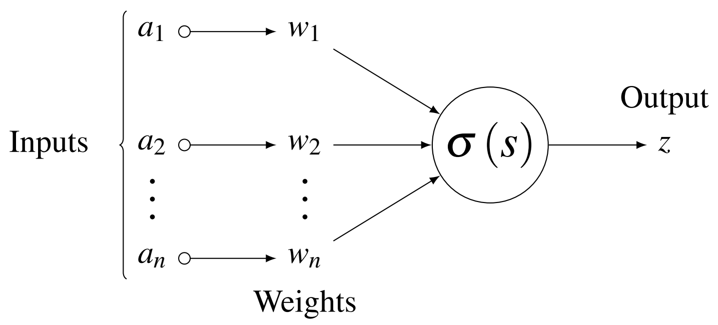

Figure 1: Neuron $N$ receives $n$ inputs $\mathbf{a}$ associated with $n$ weights $\mathbf{w}$. It performs a weighed sum $s = \mathbf{w}^T\mathbf{a}$ which is then mapped by the activation function $\sigma(s) = z$.

Neurons in the network are arranged into layers. In the case of feedforward neural networks each neuron in a layer is only connected to all the neurons in the following layer except for neurons in the output layer. See Figure 2 for an example of a feedforward neural network [2]. Equation (1) can be generalized to

$$\mathbf{s} = \mathbf{W}\mathbf{a} + \mathbf{b}$$

to consider all neurons in a layer and its connections to the following layer. $\mathbf{W}$ is here a $m \times n$ matrix containing the weight connections between two layers were $n$ it the number of outputs from the previous layer and $m$ represents the number of neurons in the current layer. $\mathbf{a}$ becomes the input coming from all neurons in the previous layer and vector $\mathbf{b}$ contains all biases associated with signals transferred from one layer to another. Again, utilizing $\sigma$, all the outputs from a layer are obtained as

$$\sigma(\mathbf{s}) = \mathbf{z}$$

where any given $z_i$ is the output of a single neuron in the layer.

An input layer, one more hidden layers, and an output layer comprise a basic ANN [2]. Together the layers classify input $\mathbf{x}$ by associating a label $y$ with the input. A number $L$ of layers transform the input $\mathbf{x}$ by

$$\mathbf{z}^{(L)} = \sigma^{(L)}(\mathbf{W}^{(L)} \dots (\sigma^{(2)}(\mathbf{W}^{(2)}\sigma^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \dots + \mathbf{b}^{(L)}))$$

where $\sigma^{(i)}$, $\mathbf{W}^{(i)}$, and $\mathbf{b}^{(i)}$ respectively represent the activation function, associated weights, and the bias vector of layer $i$. The output of the final layer, $\mathbf{z}^{(L)}$, is compared to the actual classification of the data point $y$ to determine the accuracy of the network. More on the comparison between the model's prediction and the label for the dataset can be seen in Section 2.2. Note that if there is more than one neuron in the output layer (in regression FFNN's output layer only has 1 neuron) then the result is a vector instead of scalar which is compared to a scalar $y$. $\mathbf{z}^{(L)}$ can be represented as a scalar $\hat{y}$. In the case of classification each value of $\mathbf{z}^{(L)}$ is the probability the result is a given class and the class with the highest probability is the one chosen. Thus, $\hat{y}$ is that respective label and for this paper $\hat{y}$ is always the outcome of a FFNN.

The choice of $\sigma$ defines the behavior of a neural network layer. Note that even though the input of $\sigma$ is a vector the operation is applied entry-wise. If $\sigma$ is a linear function such as the identity function

$$\sigma(\mathbf{x}) = \mathbf{x},$$

then $\mathbf{z} = \mathbf{s}$ and

$$\mathbf{z} = \mathbf{W}\mathbf{a} + \mathbf{b}.$$

With linear $\sigma$ the layer maps $\mathbf{a} \in \mathbb{R}^n$ to $\mathbf{z} \in \mathbb{R}^m$ and if $m < n$, then the layer approximately performs a principal component analysis (PCA), a technique to map higher dimensional data to lower dimensions where the most variance from the original dataset is preserved [3]. However, there are differences between standard PCA and this variation. First, $\mathbf{W}$ is not the $m$ eigenvectors or principal components of $\mathscr{D}\mathscr{D}^T$, where $\mathbf{a}$ belongs to dataset $\mathscr{D} \in \mathbb{R}^{n \times p}$. Second, bias $\mathbf{b}$, inexistent in classical PCA, shifts data points along the respective $m$ axes. For more on standard PCA see [3]. If $\sigma$ is nonlinear, then the network layer can better approximate nonlinear relationships which are more complex and prevalent in practical settings [4]. Two common nonlinear functions used in FFNNs are the rectified linear unit (ReLU) defined as

$$\sigma(\mathbf{x}) = max(0, \mathbf{x})$$

and the sigmoid function

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}}.$$

The more hidden layers with nonlinear $\sigma$ an FNNN has, the deeper the network is considered. Deeper networks have a higher representational power than networks with fewer layers. Adding more layers mimics the universal approximation theorem which states that

$$G(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i),$$

where $\alpha_i$ is a weight and $\sigma$ is any continuous sigmoidal function, approximates function $f$ [5]. However, the difference between this theorem and modern FFNNs is that for FFNNs $\sigma$ in every layer can be different and the output of any layer is used input as for the next layer unlike a weighed sum of all layer outputs. For more on the universal approximation theorem, see [5].

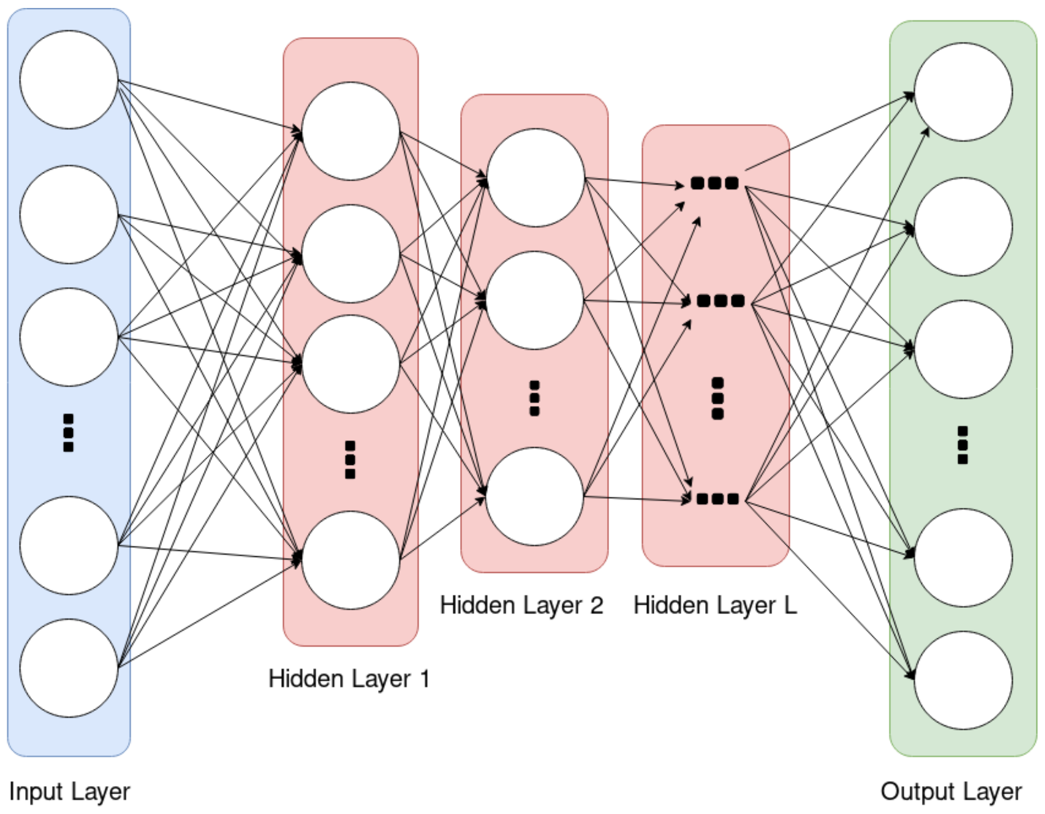

Figure 2: A Sample Feedforward Neural etwork

## 2.2 Performance Evaluation

For a data point $\mathbf{x} \in \mathbb{R}^n$ with $n$ being the number of features in the dataset, there exists a target $y \in \mathbb{R}$ and a function $f : \mathbb{R}^n \to \mathbb{R}$ such that

$$y = f(\mathbf{x}) + \varepsilon$$

where $\varepsilon$ is a normally distributed error term with zero mean that cannot be diminished known as the irreducible error [6]. This mapping function $f$ is unknown and the best that can be done is to approximate it with another function $\hat{f} : \mathbb{R}^n \to \mathbb{R}$. The result from $\hat{f}$ known as $\hat{y}$ and defined as

$$\hat{y} = \hat{f}(\mathbf{x})$$

is compared to $y$. Specifically, for FFNNs $\hat{f}$ is calculated as the output from the network

$$\hat{f}(\mathbf{x}) = \sigma^{(L)}(\mathbf{W}^{(L)} \ldots (\sigma^{(2)}(\mathbf{W}^{(2)}\sigma^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \ldots + \mathbf{b}^{(L)})).$$

The two functions are compared using a performance metric. The metric depends on if $f$ and $\hat{f}$ are classifiers or regressors. Classifiers associate a label, from a finite set, to the input given to the model. Metrics for this kind of model work in terms of the number of correct and incorrect associations. Regressors produce a continuous output attempting to approximate $y$. Performance for this kind of model is based on the Euclidean distance, defined in Section 2.2.2, between $\hat{y}$ and $y$.

### 2.2.1 Classifier Evaluation

For classifiers, having categorical target variables, the performance of a neural network is evaluated by counting how many of its predictions are true positives, true negatives, false positives (type I error) and false negatives (type II errors). These four values come from comparing the model's output label $\hat{y}$ and the ground truth label $y$. In the case of binary classification, if both labels give the true label then it is a true positive (tp) and if both labels give the false label it is a true negative (tn). If the labels disagree then the result is either a false negative (fn) or a false positive (fp) depending on the ground truth label. These results are summarized in a table known as the confusion matrix, described in Figure 3.

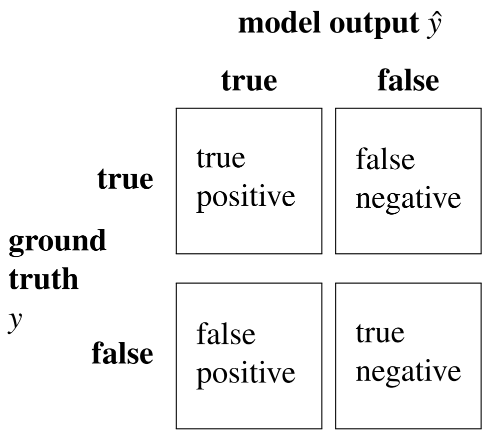

Figure 3: Confusion Matrix of a Classifier

The counts are compared in three steps. The first, called *precision* $p$, uses the ratio of true positives to total positive predictions:

$$p = \frac{tp}{tp + fp}.$$

The second, called *recall* $r$, is the ratio of correctly identified positives to total possible positives

$$r = \frac{tp}{tp + fn}.$$

These two metrics, precision and recall, are combined through their harmonic mean as the $F_1$ Score. The harmonic mean of precision $p$ and recall $r$ is calculated as

$$F_1 = 2\frac{pr}{p + r}.$$

The advantage of the harmonic mean is its resistance to a large discrepancy between the two values, meaning a high precision value and a low recall value will still result in a low $F_1$ score. This metric ranges between 0 and 1 with 0 being the worst $F_1$ score and 1 the best possible score.

Generally, the accuracy, the percentage of correct instances compared to the total number of samples is defined as

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

and is used as a basic metric to evaluate a classifier. However, if the dataset contains unequal instances of the different classes, such as the one in shown in Section 3.1, then a very naive classifier will label all input as the majority class and will get a high accuracy percentage. This high accuracy is misleading. The labeling of all instances as the majority category indicates all errors committed will be exclusively false negatives (type I error) or false positives (type II error). Depending on the context, one type of error can be harmless to make while the other could lead to a decision incurring in extraordinary losses. Given the financial context and the unbalanced classification dataset in Section 3.1, the $F_1$ score with its resistance to large discrepancies, provides a more insightful measurement of the classifier's predictive ability than basic accuracy [7].

### 2.2.2 Regressor Evaluation

For a regressor, a model with a continuous target variable, the performance is evaluated based on the Euclidean distance between the prediction $\hat{y}$ and the actual value $y$ calculated as

$$d(y, \hat{y}) = \sqrt{(y - \hat{y})^2}.$$

Using the Euclidean distance $d$ of the outputs, a metric called loss function calculates the mean squared error (MSE) as

$$MSE = \frac{1}{2p} \sum_{i=1}^{p} d^2(y_i, \hat{y}_i),$$

where $\hat{y}_i \in \mathbb{R}$ and $y_i \in \mathbb{R}$ correspond to some $\mathbf{x}_i$ in a training dataset $\mathscr{D} \in \mathbb{R}^{n \times p}$. Since all inputs come from a training dataset it means that $f$ and $\hat{f}$ have seen these data points before and tweaked their parameters accordingly. To calculate the MSE of the models on testing dataset, meaning data that the model has not seen before, then it is defined as

$$MSE = \mathbb{E}\left[\left(\hat{f} - y\right)^2\right]$$

where $\mathbb{E}$ is the expected value operation which is described in Section 2.3. MSE is used for the performance evaluation of the continuous outputs or targets for the feedforward neural networks that make up the ensemble (see Section 2.4 for more on this) attempting to predict if an individual will default. The larger the MSE, the worse the FFNN approximated the ground truth.

## 2.3 Bias-Variance Trade-off

Using an FFNN (described in Section 2.1), a model $\hat{f}$ is built. The model $\hat{f}$ is an estimator parameterized by the data points $\mathbf{x} \in \mathscr{D}_{train}$, the training dataset, and trained to be used on more datasets [8]. We abbreviate notation by letting $\hat{f} = \hat{f}(\mathbf{x})$ and $f = f(\mathbf{x})$. As $\hat{f}$ is dependent on the training data provided, it is thus a random variable whose expected value is defined by the formula

$$\mathbb{E}[\hat{f}] = \sum_{i=1}^{n} \hat{y}_i p(\hat{f} = \hat{y}_i)$$

where $\hat{y}_i$ is a possible outcome of the random variable and $p(\hat{f} = \hat{y}_i)$ is the probability of the outcome $\hat{y}_i$.

The closer the output $\hat{y}$ of model $\hat{f}$ is to the output $y$ of $f$, the better $\hat{f}$ is. $\hat{f}$ can only approximate $f$, meaning there will be always some error or difference between the outcomes of the two models. Error in any prediction model can be attributed to three major sources: bias, variance and irreducible error. Bias, defined as

$$Bias(f, \hat{f}) = \mathbb{E}[\hat{f}] - \mathbb{E}[f]$$

or simply

$$Bias(f, \hat{f}) = \mathbb{E}[\hat{f}] - f \tag{2}$$

is the inability of the model to recognize the relationship between the target label $y$ and its predictors $\{x_1, x_2, \ldots, x_n\}$ and is measured as the average difference between model prediction and the expected output value for a givens set of predictors. Variance, defined as

$$Var(\hat{f}) = \mathbb{E}[\big(\hat{f} - \mathbb{E}[\hat{f}]\big)^2]$$

is the level to which the model considers random noise to be part of the relationship and is measured as the expected difference between $\hat{f}$ and $f$ [8]. Bias and variance are considered reducible error or part of the error because they can be diminished by adjusting the parameters of a model. The third type of error is irreducible error meaning it cannot be decreased, no matter how close $\hat{f}$ is to $f$. The irreducible error appears because in many cases there are lurking variables that do not appear in the dataset used to train the model.

The precision of a model and its ability to generalize to similar datasets can be measured by the MSE of an estimator. Subsection 2.3.1 decomposes the mean-squared error in terms of its reducible error to understand how much each type of error offsets the model [8].

### 2.3.1 Bias-Variance Decomposition

For the following decomposition we introduce the constant $\tau = \mathbb{E}[\hat{f}]$.

$$\begin{aligned} MSE &= \mathbb{E}\left[(\hat{f}-y)^2\right] \\ &= \mathbb{E}\left[(\hat{f}-\tau+\tau-y)^2\right] \\ &= \mathbb{E}\left[((\hat{f}-\tau)+(\tau-y))^2\right] \\ &= \mathbb{E}\left[(\hat{f}-\tau)^2+2(\hat{f}-\tau)(\tau-y)+(\tau-y)^2\right] \\ &= \mathbb{E}\left[(\hat{f}-\tau)^2\right]+2\mathbb{E}\left[(\hat{f}-\tau)(\tau-y)\right]+\mathbb{E}\left[(\tau-y)^2\right] \end{aligned}$$

by the property of linearity and since $2\mathbb{E}\left[(\hat{f}-\tau)(\tau-y)\right]=0$ by

$$\begin{aligned} 2\mathbb{E}\left[(\hat{f}-\tau)(\tau-y)\right] &= 2\mathbb{E}\left[(\hat{f}-\tau)\right]\mathbb{E}\left[(\tau-y)\right] \\ &= 2\mathbb{E}\left[(\hat{f}-\mathbb{E}[\hat{f}])\right]\mathbb{E}\left[(\tau-y)\right] \\ &= 2\big(\mathbb{E}\left[\hat{f}\right]-\mathbb{E}\left[\mathbb{E}[\hat{f}]\right]\big)\mathbb{E}\left[(\tau-y)\right] \\ &= 2\big(\mathbb{E}\left[\hat{f}\right]-\mathbb{E}\left[\hat{f}\right]\big)\mathbb{E}\left[(\tau-y)\right] \\ &= 0 \end{aligned}$$

given that $(\hat{f}-\tau)$ and $(\tau-y)$ are independent by assumption. We can simplify the decomposition

$$\begin{aligned}
MSE &= \mathbb{E}\left[(\hat{f}-\tau)^2\right] + \mathbb{E}\left[(\tau-y)^2\right] \\
&= \mathbb{E}\left[\hat{f}^2 - 2\hat{f}\tau + \tau^2\right] + \mathbb{E}\left[\tau^2 - 2y\tau + y^2\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - 2\tau\tau + \tau^2 + \tau^2 - 2\tau\mathbb{E}\left[y\right] + \mathbb{E}\left[y^2\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - \tau^2 + \tau^2 + 2\tau\mathbb{E}\left[y\right] + \mathbb{E}\left[y^2\right].
\end{aligned}$$

Given that $y = f + \varepsilon$,

$$\begin{aligned}
MSE &= \mathbb{E}\left[\hat{f}^2\right] - \tau^2 + \tau^2 + 2\tau\mathbb{E}\left[f+\varepsilon\right] + \mathbb{E}\left[(f+\varepsilon)^2\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - \tau^2 - 2\tau\left(\mathbb{E}\left[f\right] + \mathbb{E}\left[\varepsilon\right]\right) + \mathbb{E}\left[f^2\right] + \mathbb{E}\left[2f\varepsilon\right] + \mathbb{E}\left[\varepsilon^2\right].
\end{aligned}$$

$\mathbb{E}[f] = f$ and $\mathbb{E}[\varepsilon] = 0$ since $f$ is constant and $\varepsilon$ has mean 0 by assumption. Therefore,

$$\begin{aligned}
MSE &= \mathbb{E}\left[\hat{f}^2\right] - \tau^2 + \tau^2 - 2\tau f + f^2 + \mathbb{E}\left[\varepsilon^2\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - \tau^2 + (\tau - f)^2 + \mathbb{E}\left[\varepsilon^2\right].
\end{aligned}$$

Thus, by equation (2) and

$$\begin{aligned}
Var(\hat{f}) &= \mathbb{E}\left[\left(\hat{f} - \mathbb{E}[\hat{f}]\right)^2\right] \\
&= \mathbb{E}\left[\left(\hat{f} - \mathbb{E}[\hat{f}]\right)\left(\hat{f} - \mathbb{E}[\hat{f}]\right)\right] \\
&= \mathbb{E}\left[\hat{f}^2 - 2\hat{f}\mathbb{E}\left[\hat{f}\right] + \mathbb{E}^2\left[\hat{f}\right]\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - 2\mathbb{E}\left[\hat{f}\right]\mathbb{E}\left[\hat{f}\right] + \mathbb{E}^2\left[\hat{f}\right] \\
&= \mathbb{E}\left[\hat{f}^2\right] - \mathbb{E}^2\left[\hat{f}\right]
\end{aligned}$$

we can then conclude

$$MSE = Var(\hat{f}) + Bias^2(\hat{f}, f) + \mathbb{E}\left[\varepsilon^2\right],$$

where $\mathbb{E}\left[\varepsilon^2\right]$ is the irreducible error.

By understanding the composition of the error, it can be minimized through techniques that address those error components. For instance, in Section 2.4.1 it is shown how bootstrap aggregation can reduce variance error. If the decomposed error was comprised solely of bias error then bootstrap aggregation would not improve the model.

## 2.4 Ensemble Learning

A classifier or regressor can be built and later trained on dataset $\mathscr{D}_{train}$. This regressor or classifier, known as a base learner $\varphi : \mathbb{R}^n \to \mathbb{R}$ for its membership to an ensemble, approximates the true mapping $y = f(\mathbf{x}) + \varepsilon$ [9]. An ensemble is the set of all base learners which are combined into a single learner $\Phi : \mathbb{R}^n \to \mathbb{R}$. The base learners that are classifiers are combined through a voting mechanism in which the output label from each base learner counts as vote for that label and the label that has simple majority becomes the output of combined ensemble. If the base learners are regressors then they are combined through a sum

$$\Phi(\mathbf{x}) = \sum_{m=1}^{M} w_m \varphi_m(\mathbf{x}),$$

where $w_m$ is weight dependent on the type ensemble learning technique used [10]. For example, $w_m = \frac{1}{M}$ represents a voting scheme where each learner's vote counts equally.

The performance of the ensemble $\Phi$ should be better than the performance of any individual base learner [9]. Thus, the accuracy of the base learners becomes a baseline for the overall result. Another metric of ensemble learning is diversity, defined as the number of coincident errors committed by base learners on input $\mathbf{x}$ [9]. For more on diversity see Section 2.4.2.

### 2.4.1 Bootstrap Aggregation

Bootstrap Aggregation, or bagging, is an ensemble learning technique meant to reduce the variance error of a model by generating new sample datasets from the original dataset [10]. Bagging creates each new dataset by sampling uniformly and with replacement from the original dataset. Every new dataset generated has size $|\mathscr{D}_i'| \leq |\mathscr{D}|$ were $\mathscr{D}_i'$ represents the generated dataset and $\mathscr{D}$ represents the original dataset. Each base learner $\varphi_i$ is trained on the corresponding subset $\mathscr{D}_i'$ [11]. Once all the base learners have been trained, they solve the regression or classification problem desired. If the model is solving a classification problem, then the final prediction of the class label of $y$ is the class most learners output [11]. Otherwise if it is solving a regression problem, the prediction is

$$\Phi_{agg}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \varphi_m(\mathbf{x}).$$

Bootstrap aggregation reduces the variance error defined in Section 2.3.1 by performing an average of the learners' results [10]. By averaging independent and identically distributed random variables, bootstrapping reduces the model's variance. This can be demonstrated as:

$$\begin{aligned}
Var\left(\Phi_{agg}(\mathbf{x})\right) &= Var\left(\frac{1}{M}\sum_{m=1}^{M}\varphi_m(\mathbf{x})\right) && (3)\\
&= \frac{1}{M^2}Var\left(\sum_{m=1}^{M}\varphi_m(\mathbf{x})\right)\\
&= \frac{1}{M^2}\sum_{m=1}^{M}Var(\varphi_m(\mathbf{x})) && (4)\\
&= \frac{1}{M^2}\sigma^2 M\\
&= \frac{\sigma^2}{M} && (5)
\end{aligned}$$

First, in (3) the learners of the ensemble are averaged. Given that each $\varphi_m(\mathbf{x})$ in (4) acts on the same group of data points and are independent and identically distributed random variables[1], they have all have variance $\sigma^2$. In (5), the variance is reduced by a factor $M$ the size of the ensemble.

### 2.4.2 Diversity in Ensemble Learning

There exists no model that can perfectly match the true function $f$. Ensembles, similarly to the universal approximation theorem mentioned in Section 2.1, approximate $f$ by including additional base learners. However, the additional base learners do not increase the accuracy or reduce the loss of $\Phi$ if they are identical to other base learners already in the ensemble [11]. Thus, diversity must be introduced to improve the overall result.

Diversity can loosely be described as how different two or more objects are [11]. The diversity of an ensemble or difference between its base learners is quantified based on the errors committed by the learners [9]. For simplicity, in this

[1]These learners are not necessarily independent given that they come from the same dataset. Independence is the baseline situation but in reality the best that could be expected is for learners to be pairwise negatively correlated.

section $\varphi_i$ and $\varphi_j$ are arbitrary base learners which are also binary classifiers. As well, the following quantities are defined as

$$\begin{aligned}
a &= p(\varphi_i = 1 \cap \varphi_j = 1),\\
b &= p(\varphi_i = 1 \cap \varphi_j = 0),\\
c &= p(\varphi_i = 0 \cap \varphi_j = 1),\\
d &= p(\varphi_i = 0 \cap \varphi_j = 0).
\end{aligned}$$

$\varphi_i$ and $\varphi_j$ are further away from each other if they have less coincident errors. There exist different diversity measurements that follow the axioms mentioned before. See [11] for additional metrics. One simple example is the disagreement measurement defined as

$$D_{\varphi_i,\varphi_j} = b + c.$$

If the ensemble has $M > 2$ base learners then the diversity of $\frac{M(M-1)}{2}$ pairs of base learners is measured and all of those results are then averaged together.

The correlation coefficient $\rho$ of $\varphi_i$ and $\varphi_j$ defined as

$$\rho_{\varphi_i,\varphi_j} = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}}$$

ranges between $-1$ indicating complete independence and 1 indicating complete dependence. The greater diversity among base learners, the more independent they are [11].

To illustrate the relationship between diversity, correlation, and accuracy consider an ensemble with 21 base learners acting on input $\mathbf{x}$. The accuracy of each base learner is 70% and their misclassifications are completely independent. The error rate ($1 - \textit{accuracy}$) of the combined model $\Phi$ is equal to the area under the binomial distribution between $\frac{M}{2}$ and $M$, where $M$ is the number of learners in the ensemble [9]. See Figure 4 for more detail on the error rate of the ensemble of 21 base learners. The diverse ensemble has a lower error rate than the homogeneous ensemble in Figure 5 that also has 21 base learners each with 70% accuracy.

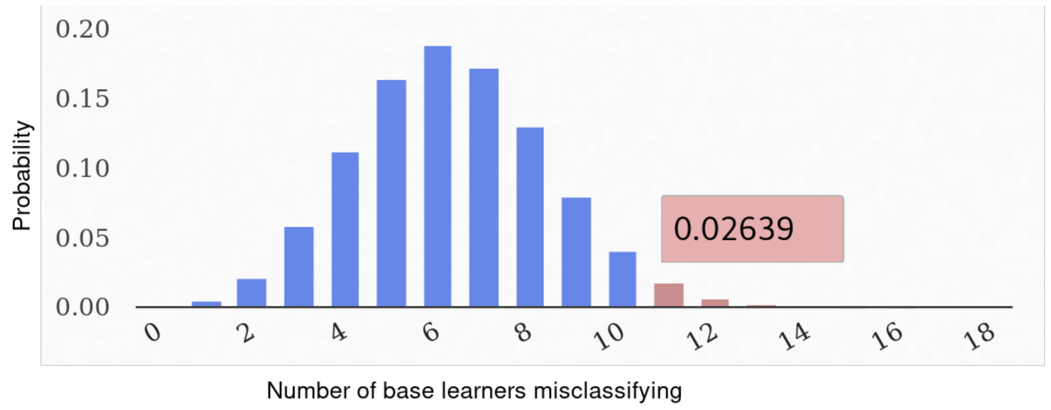

Figure 4: Binomial distribution of 21 base learners each one with an accuracy rate of 70% and independent misclassifications. The probability of more than half of the learners misclassifying **x**, meaning the error rate of the ensemble, is 2.639%. The more base learners with independent misclassifications and 70% accuracy will reduce this error rate even more [9].

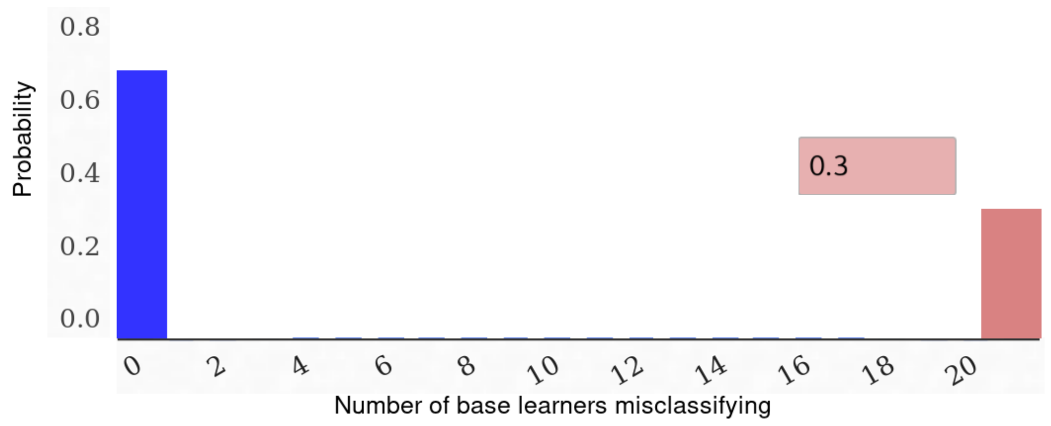

Figure 5: Distribution of 21 base learners each one with an accuracy rate of 70% and dependent misclassifications. The probability of more than half of the learners misclassifying **x**, meaning the error rate of the ensemble, is 30%. The error rate stays at 30% if more dependent base learners are added with the same accuracy [9].

# 3 Model Development

## 3.1 Data

All training and testing of the framework used the Credit Card Default dataset from the University of California, Irvine Machine Learning Repository. The dataset contains 30,000 monthly credit card records for consumers using credit cards in Taiwan from May through October 2005. Each record consists of 23 features covering demographic information, as well as the amount of bill statements, bill payments, credit limit, and repayment status for the six months preceding the record. Each of the 23 features are either categorical or numerically valued. A full description of all features can be found in Appendix A. The label for each record is either 1.0 indicating the consumer defaulted on their next credit card payment or 0.0 indicating the consumer did not default on their next payment. There are 6,636 records in the dataset representing a default in the next month, forming 22.12% of the dataset, while the remaining records in the dataset representing non-defaults, form the other 77.88% of the data. There are approximately four times as many records in the non-default classification as there are in the default classification. Therefore, non-default is named the majority class and default the minority class.

Established methods to deal with imbalanced classes in a classification problem include oversampling, undersampling and generating synthetic data points. The last technique is popular for dealing with unbalanced datasets and thus bears mention; however, it is beyond the extent of this project. Each method generates a new dataset from the original where each class is equally represented. Oversampling randomly samples with replacement from the minority class until there are as many examples of the minority class in the new dataset as there are samples of the majority class. Oversampling does not add any new information to the dataset, but by repeating examples from the minority class to balance the distribution, it guards against the tendency of predictive models to err towards the more prevalent class. The repetition of the minority examples can potentially make the model fit more tightly to those examples, running the risk of over-fitting in the minority class. Undersampling randomly samples with replacement from the majority class only as many examples as there are minority class examples. Undersampling avoids the issue of overfitting in the minority class, at the expense of some amount of information from the majority class. In the case of the credit card default dataset, undersampling would exclude almost three-fourths of the examples in the majority class. Generating synthetic data points is a general approach that encompasses many different algorithms. A popular algorithm, SMOTE (Synthetic Minority Oversampling Technique) generates synthetic data points along lines drawn between existing examples [12]. It makes a key assumption that the shape of the data in the minority class is convex. Convex data lies in a region where for any line segment drawn between two of the data points, all the points on that line lie within the same general region as the data [13].

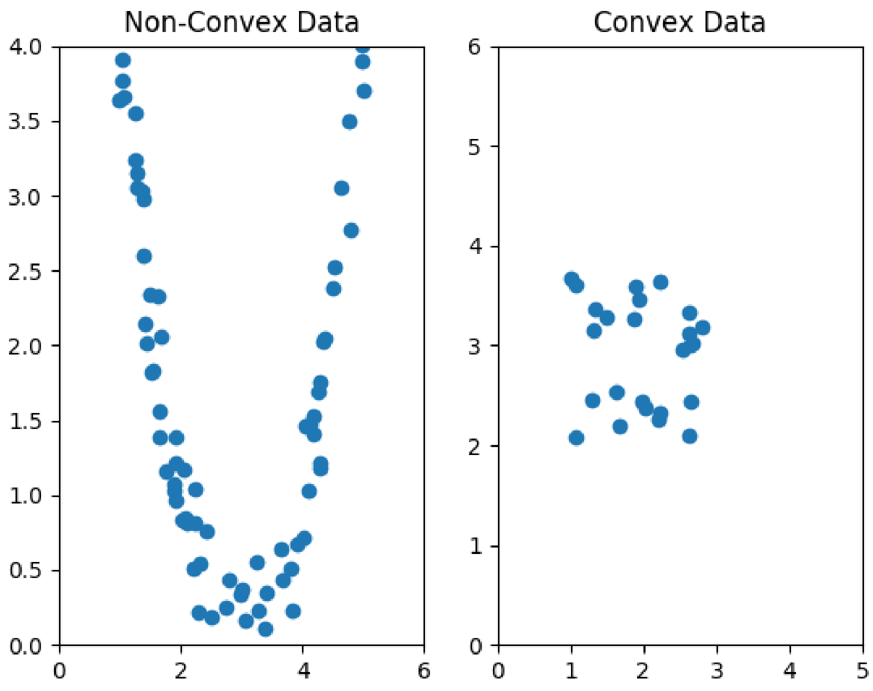

Figure 6: An example of non-convex data and convex data. All points in the convex data can be connected by a line that is approximately contained in the general region of the data. In the non-convex data, where the shape follows a parabola, a line drawn from one side of the parabola to another would clearly cross outside the region occupied by the data.

In higher dimensional spaces it becomes infeasible to fully understand the shape of the dataset, therefore impossible to determine if the base assumption of SMOTE holds.

Of the techniques outlined above, undersampling naturally lends itself to ensemble learning through combination with bagging. When bagging generates each new sample dataset, undersampling can impose the restriction that each new dataset contains an equal number of examples from both the majority and minority classes. Individually, the new datasets lack the information of the whole dataset. When combined in the ensemble, the entire dataset is represented in the model, even though no single base learner sees all the information. Using bagging in combination with undersampling both deals with the issue of the imbalanced classes and retains all the information in the original dataset.

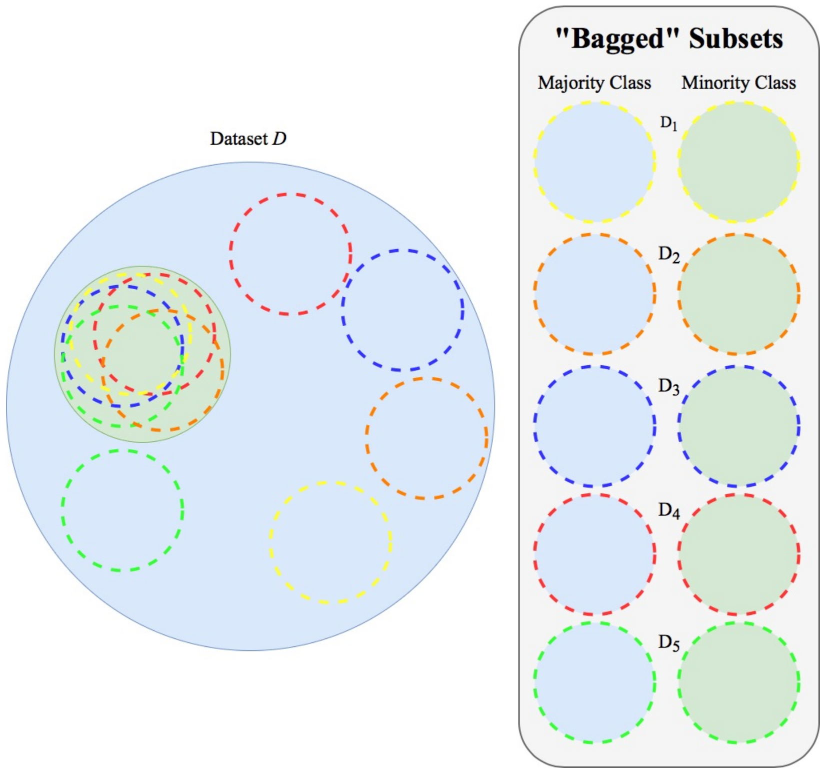

Figure 7: A visualization of bagging with undersampling. Each subset $X_i$ of the original dataset represents a smaller amount of data taken from the training examples. A single learner could not view all the data contained in the dataset from one of these subsets. In aggregation, the subsets cover the extent of the data. Therefore, in aggregating the learners built from each subset the ensemble can form a prediction without the class imbalance bias but with all the information contained in the dataset.

In a concrete sense, consider as an example of an ensemble of learners: a group (the ensemble) of friends (the learners) examines a collection of photos and attempts to model the best way to determine if the subject of a photo is a cat or a dog. They have many pictures of dogs (the majority class) but relatively few photos of cats (the minority class). If taken all at once, the model will have a higher likelihood for predicting the label "dog" in a less obvious example. With bagging and undersampling, each friend takes a random selection of equal size from each category (dogs and cats). Based on those images, the friend comes up with a model of how to tell the difference between the two categories. After coming up with their model, the friend replaces the photos, and another friend repeats the process. With equal numbers of photos from both categories, the consistent bias towards the "dog" label disappears.

As more learners in the group take random selections from each class, the selections cover more of the original datasets majority and minority class. The result is a larger portion of the available information about the majority class is used in the development of the model. The data in the minority class are likely to be repeated across the learners, as there are fewer data points in the minority class from which to sample. The repeated sampling of the majority class yields more information because there are more data points sampled from it than simple undersampling. The repeated sampling of the minority class yields no further information than is available. Additionally, this means there is a size of ensemble where every point in the majority class has been sampled at least once, and further bagging does not add any information to the ensemble model.

Furthermore, we hypothesize that the underlying reason our model is viable stems from the idea that from these random combinations, we make the underlying structure of the data more accessible to the models making predictions. By including a combination of predictors ultimately representing each possible target, we hope to improve the information given to our model and its overall accuracy.

## 3.2 Model Structure

The goal of the model creation is to construct a two-step framework for classification. In the first step, the model determines some underlying structure of the dataset by repeatedly attempting to approximate the mapping from a subset of the dataset features to a single other feature instead of the associated label. By aggregating the results of the approximations, the model yields an expanded feature set which shows some underlying structure of the dataset. In the second step, the model uses its determination of the underlying structure to perform classification on the testing samples in the dataset.

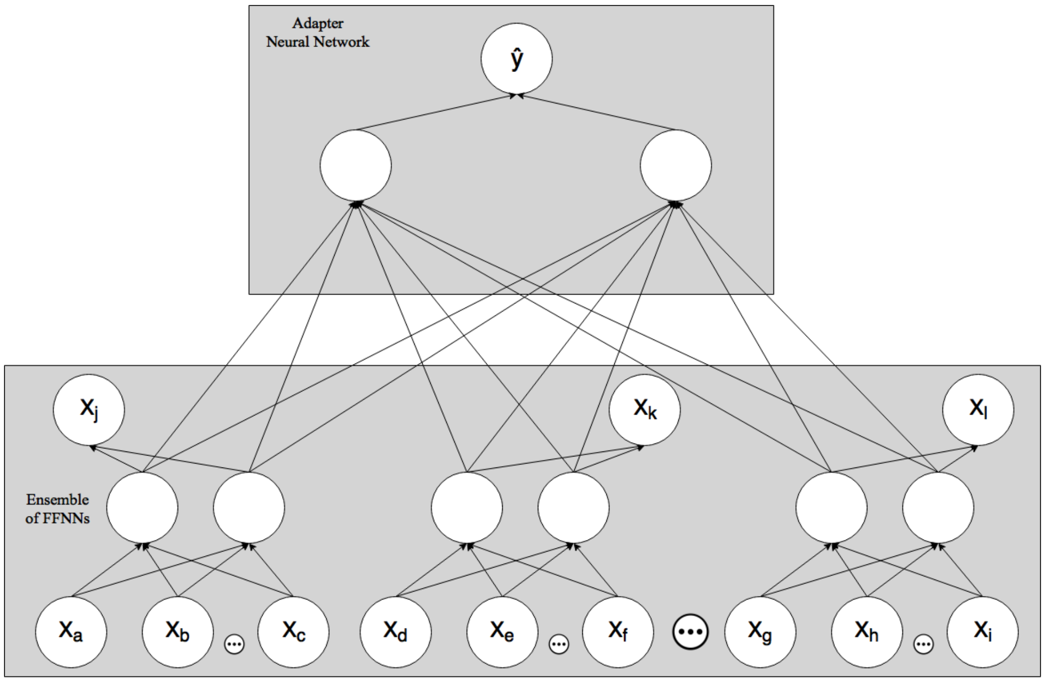

Figure 8: A deep ensemble learning network with an adapter network. The outputs from the final hidden layer of each network in a trained ensemble of FFNNs are concatenated and used as inputs to the final adapter network, which performs the classification.

The first step to the framework builds an ensemble of FFNN's using subsets of the training set. The datasets are generated using bagging with undersampling. Each neural network trains on a subset of the features in the dataset, attempting to predict the value of a single other feature from the values of the subset. For example, one base learner in the ensemble might use the AGE, SEX, EDUCATION, and LIMIT_BAL features as inputs and use the MARRIAGE feature as the target variable. Another learner might use PAY_5, PAY_2, BILL_AMT1, and MARRIAGE features as inputs and use the AGE feature as the target variable.

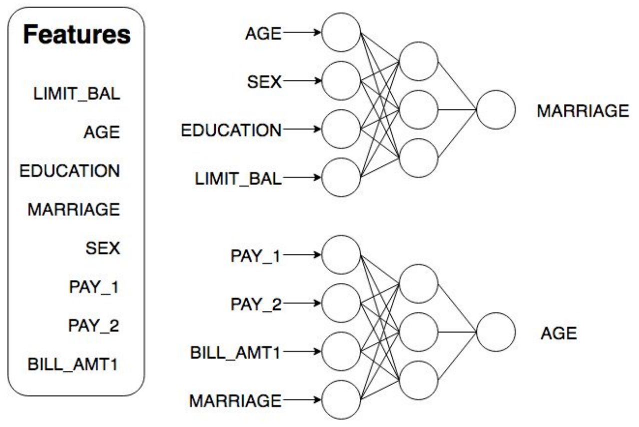

Figure 9: Simple FFNNs for the ensemble. Each learner in the ensemble takes a subset of the available features from an example as inputs. It uses these inputs to predict the value of another feature from the example. The predicted feature cannot be part of the subset used as inputs to the learner.

The networks learn a representation of the feature selected as their target variable in relation to a subset of the other features in the dataset. Formally, each network's construction follows these steps:

1. Randomly select a single feature to use as a target variable for the network, denoted here as $x_t$
2. Select a random subset of features $\mathbf{x_a}$ from the given set of features $\mathbf{x}$ such that $|\mathbf{x_a}| < |\mathbf{x}|$ and $x_t \notin \mathbf{x_a}$. The size of the subset $\mathbf{x_a}$ is $0 < |\mathbf{x_a}| < n$ where $n$ is an integer representing the total number of features in the dataset.
3. Randomly generate the number of hidden layers in the network and the number of units in each of those layers to create the network structure.

4. Train the network using an under-sampled subset of the training data with only the features included in $\mathbf{x_a}$.

After completing the training of the ensemble, the outputs from the final hidden layer of each network are extracted and concatenated to create an expanded version of each example in the original dataset. To be concrete, say there are three networks in the fully-trained ensemble, two with three layers (an input layer, a hidden layer, and an output layer) and one with four layers (an input layer, a first hidden layer, a second hidden layer, and an output layer). The first three-layer network takes AGE, SEX, EDUCATION, and LIMIT_BAL as inputs with MARRIAGE as its target variable. The second three-layer network takes PAY_5, PAY_2, BILL_AMT1, and MARRIAGE as inputs with AGE as its target variable. The four-layer network takes BILL_AMT2, BILL_AMT4, EDUCATION, and PAY_3 as inputs with LIMIT_BAL as its target variable. As shown in Figure 3.2, given an input example to the ensemble $\mathbf{x_i}$, the value of each feature in the example is used as an input to the corresponding input of each learner in the ensemble which uses that feature. The corresponding outputs from the hidden layer of the first two networks and the outputs from the second hidden layer of the third network are concatenated into a single vector, notated as $\mathbf{z_i}$. The vector $\mathbf{z_i}$ is given the same label as the feature vector from which it was generated. The completely expanded dataset is notated as $\mathbf{Z}$ where each entry corresponds to the original label. Formally:

$$\mathbf{Z} = \psi(\mathbf{X})$$

where $\psi(\mathbf{X})$ denotes the transformation of $\mathbf{X}$ by the trained ensemble.

The second step of the framework trains a FFNN classifier to predict the original labels $Y$ from $\mathbf{Z}$. In other words, the FFNN approximates the relationship $f(\mathbf{Z}) = Y$ as:

$$\hat{f}(\mathbf{Z}) = Y$$

The FFNN classifier consists of an input layer, three hidden layers, and an output layer. The connections between the each of the three hidden layers have a dropout rate of 30%. This means for any given evaluation of an example by the network, a random 30% of the neurons in each layer will be disconnected from the others and are neither used nor updated during that computation. Each hidden layer contains fewer total neurons than the layer preceding it. The number of neurons in each layer is calculated relative to the number of features in $\mathbf{Z}$.

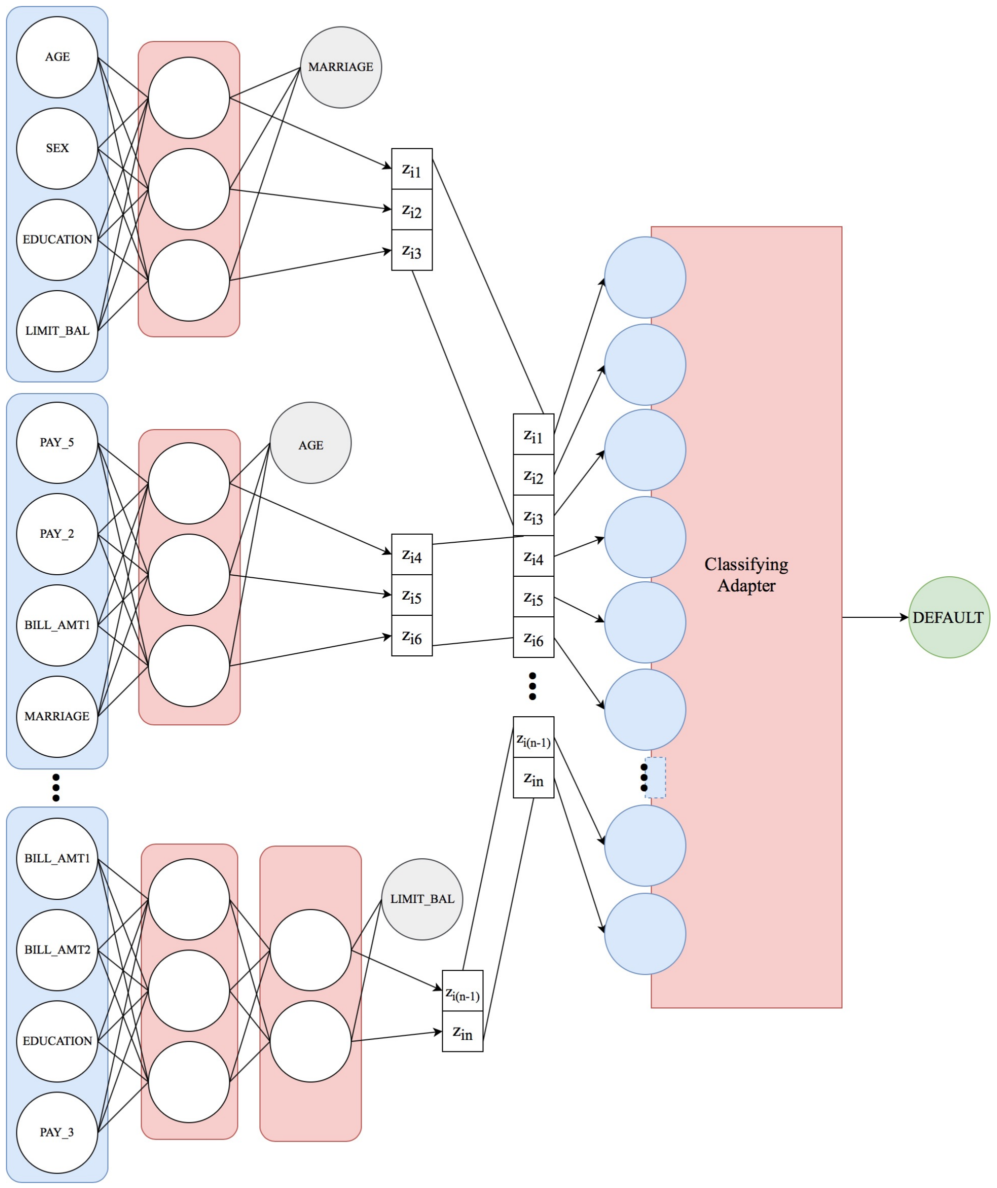

Figure 10: Expansion of data for classification. The values of each feature from an example are used as input values to the corresponding inputs of each learner in the ensemble. The outputs of the final hidden layer of each learner are concatenated to form the expanded example **z**. Each example **z** is then concatenated to form the expanded dataset **Z**. Finally, the classifying adapter layer of the framework is trained to predict the value of DEFAULT from **Z**.

## 3.3 Benchmark Models

In order to place the results of the developed framework in perspective, it was compared to the results of several other models. These models bear similarity to the structure of the framework in implementation and in functionality. They were selected to provide a reasonable comparison of the framework to established models.

### 3.3.1 Classical Auto-encoding

The first of these models is comprised of an auto-encoder and a basic FFNN. The auto-encoder mirrors the behavior of the ensemble of networks, taking in the features and producing approximations of them. The hidden layer of the auto-encoder is extracted to be used as the new base dimension for the classifier, just as the hidden layer of the ensemble networks is extracted to be used as the new base dimension for the adapter network. The classifier (and adapter network) use this new base dimension to perform their predictions of whether or not the example will result in a default next month.

### 3.3.2 Classical Ensemble

The second is an ensemble of simple neural networks, each attempting to predict the label of the samples. The structure of the ensemble of neural networks mirrors the structure of the framework, as both are built on ensembles of simple networks. The difference is in the targets and structure of the networks in each of the ensembles. While the benchmark ensemble is comprised of networks with randomized structure attempting to predict the label from all the features, the framework ensemble is comprised of networks with uniform structure (excepting the inputs) attempting to predict a feature of the dataset from a subset of the other features. The results from the ensembles are aggregated and used to perform the final classifications in both models.

### 3.3.3 Random Forest as Adapter

A third benchmark was constructed to compare the functionality of different types of adapters as the ensemble size changed. In this benchmark the neural network adapter was replaced with a random forest classifier with a fixed number of trees. To compare the performance of the FFNN to the performance of the random forest, the framework's ensemble was trained with sizes ranging from 1 to 50 inclusive. Once trained, the expanded data from the framework's ensemble was used for both the neural network and the random forest classifier.

# 4 Results

## 4.1 Experiments

The effectiveness of the framework was tested with several experiments. These experiments investigated the effect of different hyper parameters of the model on results with the UCI Credit Card Default Dataset. The parameters tested are: ensemble size, number of hidden layers in the adapter network, the sizes of each of those layers, the loss function applied, and the activation functions used at each layer of the adapter. Each experiment observes the change in the different performance metrics of the adapter network, the average accuracy of the ensemble, and the total time to train the framework.

The experiments intentionally avoid tuning the structure of the ensemble networks. The ensemble networks are randomized in the combinations of features they use as inputs and the feature they selected as the target feature. Therefore, tuning each network in the hopes of increasing accuracy would likely have to be done individually for each generated network, which is computationally infeasible.

| **Model** | **Accuracy** | **Time** |
|---|---|---|
| Unsupervised Ensemble with FFNN Adapter | 70.03% | 24230s |
| Classical Auto-Encoder | 64.43% | 306s |
| Supervised Ensemble of 200 FFNN | 70.87% | 21304s |

Table 2: The results from the first two benchmark models compared to the Unsupervised Ensemble with a FFNN Adapter. The proposed framework outperformed a classical dimension reduction with an autoencoder before classification, and performed approximately equivalent to a carefully tuned, supervised ensemble of 200 FFNNs.

## 4.2 Model Performance

This section is broken into two subsections: one displaying the results for the Neural Network adapter and one for a random forest adapter. Both sets of tests were run with the same hyperparameters and on the same split of training and testing data.

### 4.2.1 Neural Network Adapter Results

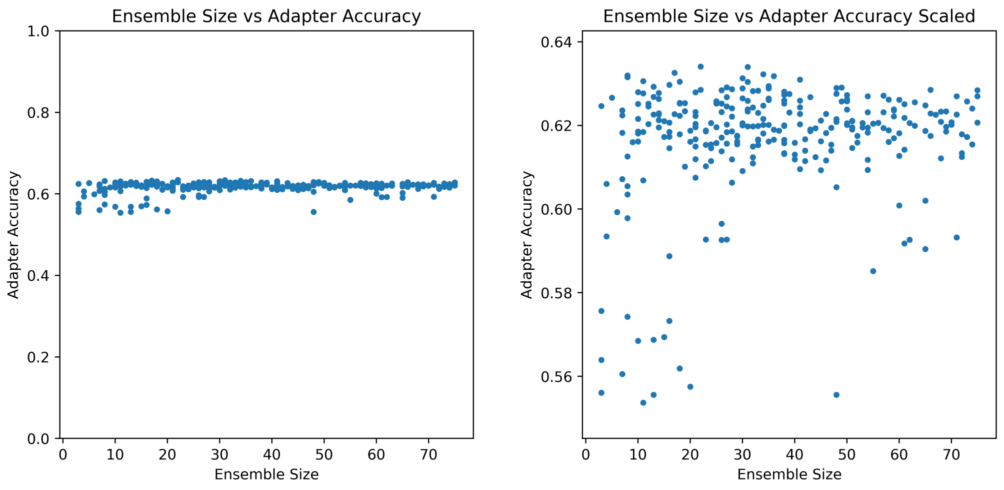

Figure 11: Adapter Accuracy v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and feedforward neural network adapter accuracy with an asymptote at roughly 0.64.

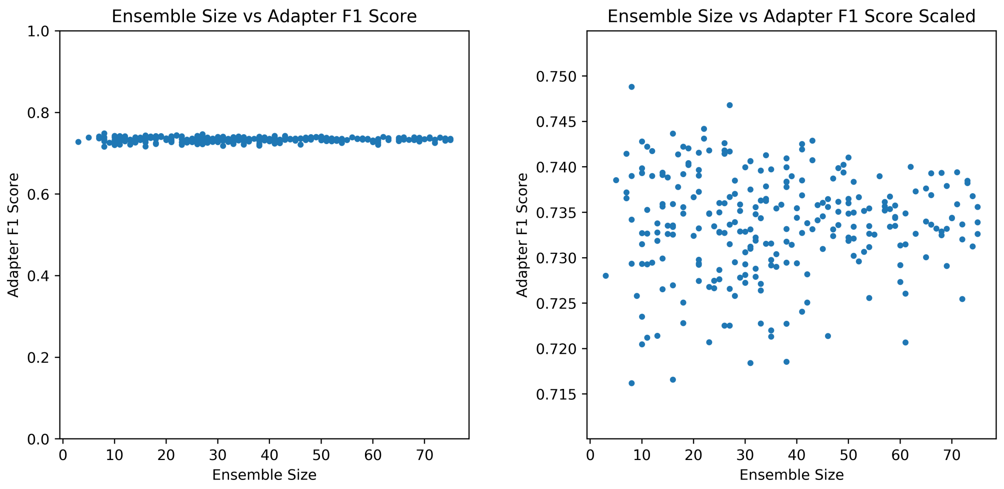

Figure 12: Adapter F1 Score v Ensemble Size. From the data above, we see a decrease in variance as the number of networks in the ensemble increases.

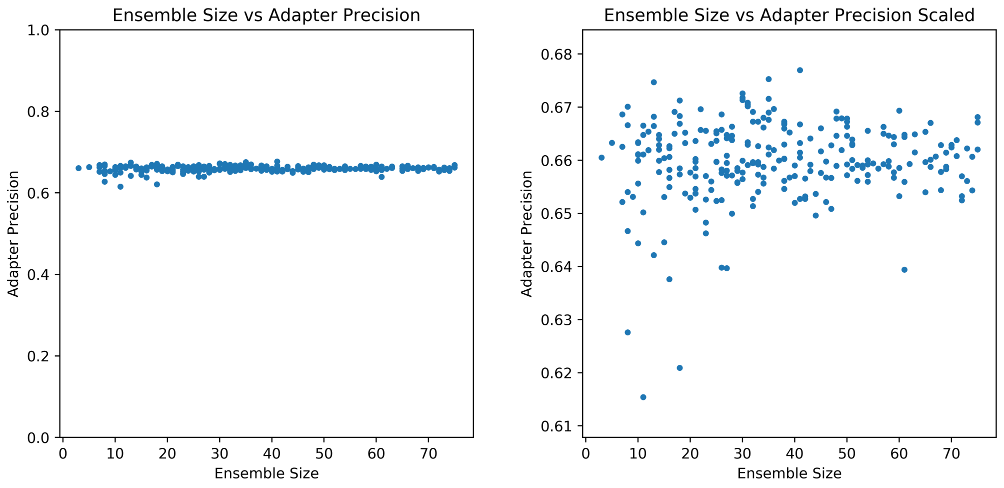

Figure 13: Adapter Precision v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and feedforward neural network adapter precision with an asymptote at roughly 0.68.

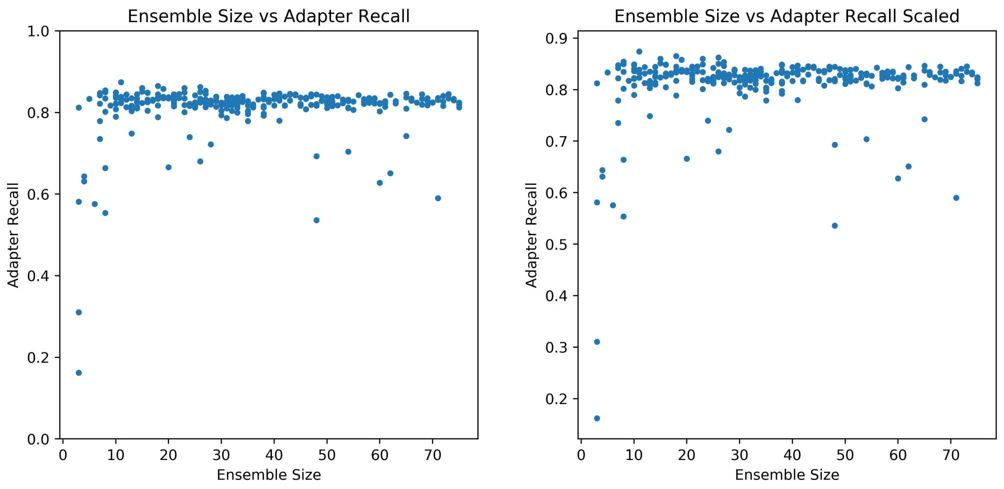

Figure 14: Adapter Recall v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and feedforward neural network adapter recall with an asymptote at roughly 0.88.

### 4.2.2 Random Forest Adapter Results

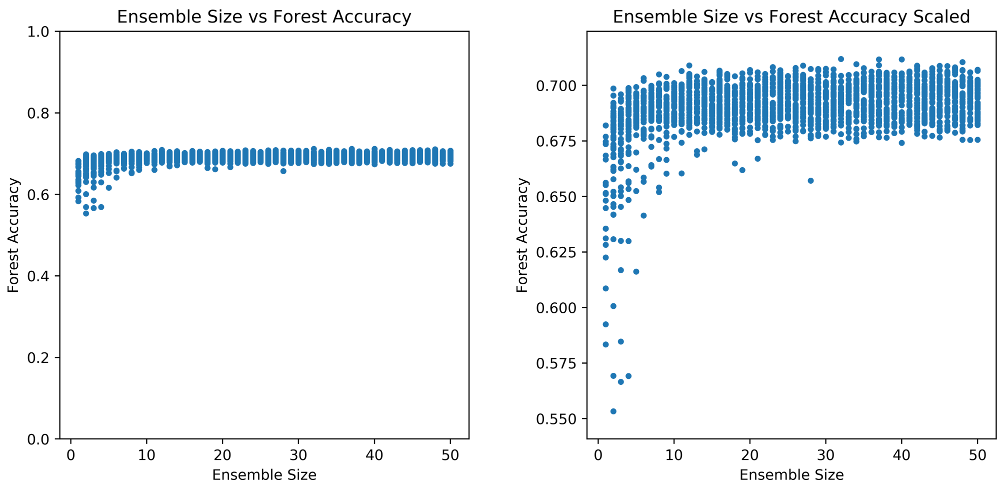

Figure 15: Adapter Accuracy v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and random forest adapter accuracy with an asymptote at roughly 0.72.

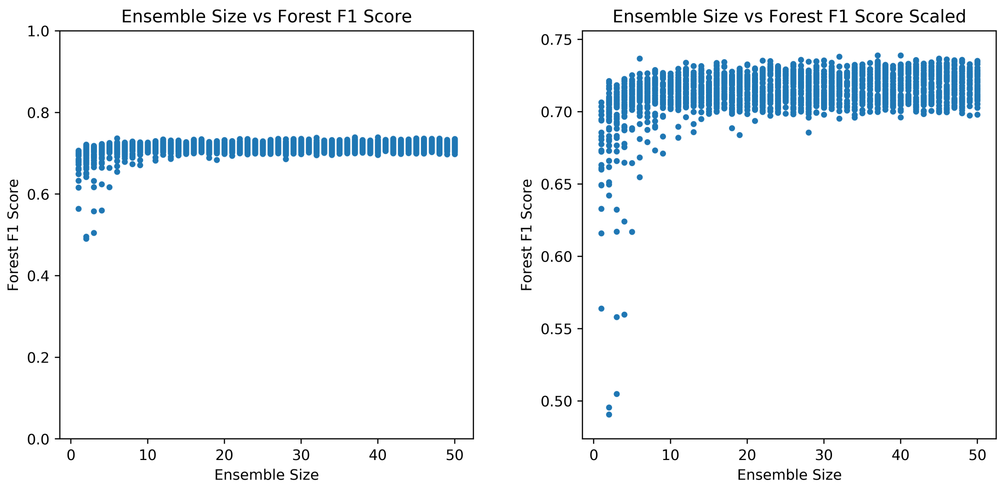

Figure 16: Adapter F1 Score v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and random forest adapter F1 score with an asymptote at roughly 0.75.

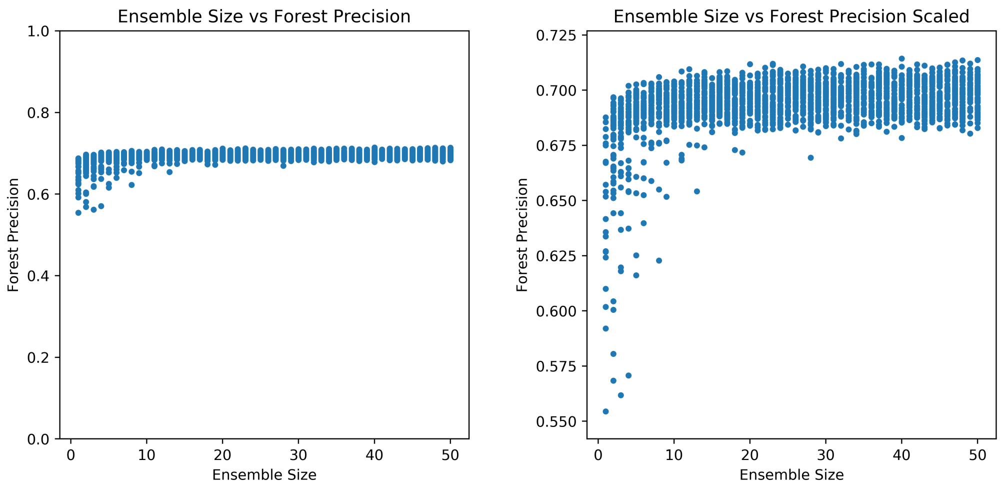

Figure 17: Adapter Precision v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and random forest adapter precision with an asymptote at roughly 0.72.

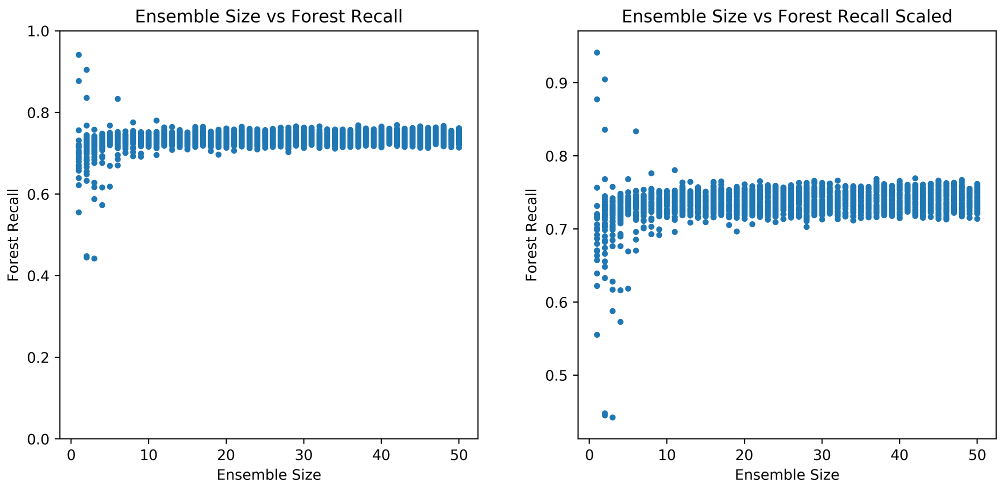

Figure 18: Adapter Recall v Ensemble Size. From the data above, we see a roughly positive logarithmic correlation between ensemble size and random forest adapter recall. Interestingly, there were some test cases that seemed to follow a decay leading to the same asymptote with an asymptote at roughly 0.76.

Something noteworthy for the figures above is that each metric appears to converge when the ensemble size reaches 23. We suspect this is due to the original training set having 23 features acting in conjunction with the design of our program. It specifies that while the ensemble size is less than 23, each feature can only be selected once as a target for an ensemble member.

## 4.3 Analysis

Benchmarking showed that the performance of the random forest adapter was dependent on the ensemble size until there were roughly 23 nets in the ensemble. This contrasts with our neural network adapter that seemed to lack dependence on ensemble size for certain metrics. The effect of an increased ensemble size for the neural network adapter on F1 Score and Precision was a decrease in variance around an almost flat trendline, whereas the effect on Percent-Correct and Recall was positive. One possible explanation for this behavior is that the output of the ensemble is not making the underlying structure or information about the data more accessible to the random forest. Because the ensemble was built so that the first 23 members were attempting to predict each of the different columns, it logically follows that the random forest would keep improving with the addition of more columns. However, it appears that the neural network adapter was able to achieve peak performance before the ensemble size was equal to the number of columns in the original data set.

## 5 Conclusion and Future Work

The results show that for both the FFNN implementation and the random forest implementation of the adapter, a larger ensemble reduces the variance of the model's performance as expected. The added randomness showed a high level of error for small networks but as the size of the ensemble increased the aggregation of the relationships learned within the data proved sufficient to achieve more consistent results. The implementation of an unsupervised ensemble, combined with a supervised classifier and no specific tuning of hyperparameters yielded comparable results to a highly tuned supervised ensemble. With more extensive tuning, the framework could achieve higher performance.

With slight tuning of hyperparameters, the random forest slightly outperformed the FFNN for ensembles with greater than 20 members. For ensembles with fewer than 20 members the random forest showed a higher sensitivity to the size of the FFNN ensemble than the FFNN adapter. Since both adapters showed less variance across the other hyperparameters as the ensemble sized increased, the new structure of the ensemble follows the expected behavior of ensembles.

Much of the possible future work for this framework involves different implementations for the structure of the ensemble networks, different methods for aggregation of the hidden layers, and other classifiers used as the adapter. For example, one possible implementation could use an ensemble of FFNNs that map the subset of features to a subset of different features, aggregated by averaging the outputs from their hidden layers, and classified using k-nearest neighbors. This adaptations still follows the general progression from an unsupervised ensemble to a supervised classifier. There are many more combinations that follow this framework to be explored.

## References

[1] C. H. Dagli and P. Poshyanonda, "Basic artificial neural network architectures," *Artificial Neural Networks for Intelligent Manufacturing*, p. 39–65, 1994.

[2] B. C. Bangal, "Automatic generation control of interconnected power systems using artificial neural network techniques," May 2009.

[3] *Principal Component Analysis and Factor Analysis*, pp. 150–166. New York, NY: Springer New York, 2002.

[4] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Eötvös Loránd University, Hungary*, vol. 24, p. 48, 2001.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[6] S. Fortmann-Roe, "Understanding the bias-variance tradeoff," 2012.

[7] L. A. Jeni, J. F. Cohn, and F. D. L. Torre, "Facing imbalanced data–recommendations for the use of performance metrics," *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, 2013.

[8] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, p. 1–58, 1992.

[9] T. G. Dietterich, "Ensemble methods in machine learning," *Multiple Classifier Systems Lecture Notes in Computer Science*, p. 1–15, 2000.

[10] P. Bühlmann, "Bagging, boosting and ensemble methods," *Handbook of Computational Statistics*, p. 985–1022, 2011.

[11] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. John Wiley and Sons Inc, 2014.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote:synthetic minority oversampling technique," *Journal of Artificial Intelligence Research*, p. 321–357, Jun 6AD.

[13] C. C. Morris and R. M. Stark, *Finite mathematics: models and applications*. Wiley, 1 ed., 2016.

## A Description of Dataset

Table 3: Description of Dataset Features

| Feature Name | Feature Type | Feature Description |
|---|---|---|
| LIMIT_BAL | Numerical | Amount of total credit |
| SEX | Binary | 1=male, 2=female |
| EDUCATION | Categorical | 1=grad school, 2=university, 3=high school, 4=others, 5, 6=unknown |
| MARRIAGE | Categorical | 1=married, 2=single, 3=others |
| AGE | Numerical | age in years |
| PAY_1 | Categorical | September, 2005 Repayment Status |
| PAY_2 | | August, 2005 |
| PAY_3 | | July, 2005 |
| PAY_4 | | June, 2005 |
| PAY_5 | | May, 2005 |
| PAY_6 | | April, 2005 |
| BILL_AMT1 | Numerical | September, 2005 bill statement amount (NT dollar) |
| BILL_AMT2 | | August, 2005 |
| BILL_AMT3 | | July, 2005 |
| BILL_AMT4 | | June, 2005 |
| BILL_AMT5 | | May, 2005 |
| BILL_AMT6 | | April, 2005 |
| PAY_AMT1 | | Amount of previous payment in: September, 2005 |
| PAY_AMT2 | | August, 2005 |
| PAY_AMT3 | | July, 2005 |
| PAY_AMT4 | | June, 2005 |
| PAY_AMT5 | | May, 2005 |
| PAY_AMT6 | | April, 2005 |