Evaluating the generalizability of an MLP and SVM through a novel testing approach

Arshad Ahmed, Konstantinos Stathoulopoulos, *City University London*

***Abstract*— We evaluate the prediction accuracy of a Multilayer Perceptron and a Support Vector Machine, in classifying wines as being either red or white based on its chemical and quality attributes. We show that both these models are able to learn a robust representation of this problem, giving high accuracy and is resistant to noisy data. We test our models with additional data generated from augmenting the original data and adding noise and propose a novel Asymmetric Stress Testing Method to test the robustness of the models and gain confidence in their generalizability. We show that even with our Asymmetric Stress Testing, the models give good results and the MLP outperforms the SVM. This classification model can be incorporated as part of the wine certification process and drive automation because the system is able to classify the wines based on its chemical attributes without additional intervention.**

***Index Terms*— Artificial Neural Networks, Multilayer Perceptron, Support Vector Machine, Classification**

I. Introduction

We performed our binary classification experiment on an MLP and SVM based on the Wine Quality Dataset[1] available from the UCI Machine Learning Repository. The task is to classify a wine as either red [0] or white [1] based on 12 physicochemical attributes and 1 sensory 'Quality' attribute. We performed our analysis using IPython[2], and the Keras[3] Deep Learning Library to build the MLP, with the Theano [4], [5] library as a backend and the Scikit-Learn Library[6] for the SVM. We show that both models learn a robust representation of this data and is able to handle a good amount of noise in the input to make accurate predictions. We also introduce our novel Asymmetric Stress Testing Method for these models. The motivation for this study is that a robust classification model can be integrated in the wine classification process and reduce costs by allowing greater automation of the process. At present wines have to be chemically tested and then classified by a human expert. Our motivation is to help the industry to reduce reliance on expert systems and automate their processes as much as possible while increasing profitability. We show that our models are able to do this with a very high level of accuracy and handle a high amount of noise in the input data as well.

A. Hypothesis Statements

Our initial hypothesis (H0) is that the MLP should generalize better than the SVM despite significant noise in the data. The alternative hypothesis (H1) is that both these methods will generalise equally well despite significant noise in the input data. This paper is organized in the following manner, in **section 1**; we present a brief exploratory analysis of the dataset and our initial observations regarding the class imbalance in the dataset. In **section 2**, we discuss our choice of models the MLP and SVM, and discuss the key parameters and explain our choices based on literature and testing. In **section 3,** we summarise our data analysis methodology followed by experimental results in **section 4**, and their discussion. In **section 5** we present our conclusions and suggest further work that can be undertaken.

II. Exploratory data analysis

The dataset contains 11 continuous attributes, which are based on physicochemical tests and one output variable Quality. The dataset comes in two sets, one containing the data for red wines and one for white wines. For each set, we created an additional attribute to characterize the colour of the wine. Red wine was given a value of 0 and white wine became 1. We merged these two sets in a single dataset and used the colour class label as our target variable. The task was to predict these labels using the MLP and SVM. In the individual red and white wine datasets we 4898 samples for the white wine and 1599 samples for the red wine which gives us as class imbalance of 1:3 for white to red wine samples. To deal with this issue we perform oversampling of the red wine dataset[7], add noise and then test the models against this new data ensure they are robust[8], [9].

We also observe that there is a huge variation in scale among the variables within both the datasets shown in Fig 1. The Free Sulphur and Total Sulphur dioxide variables are dominant in both datasets and suppress the other variables. This highlights the need for data scaling. Prior to modelling we scale the data to a [0, 1] interval. The diagonal correlation matrix shown in Fig 2, highlights that there are some fairly strong relationships between the variables. Feature learning will be successful in this dataset. Since Neural Networks are universal function approximators we did not explicitly, supply it with a compressed representation of the data through a method such as PCA, as it would constitute information loss. Also the dataset is not very high dimensional so there is not an explicit need to perform any feature learning.

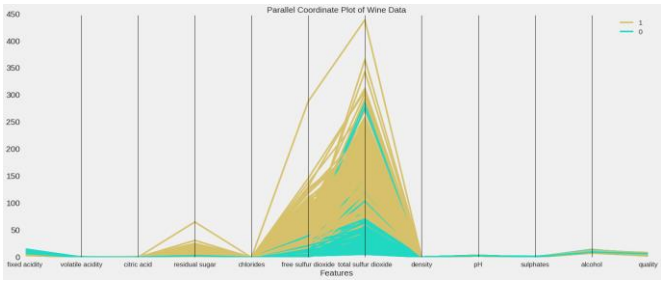

Figure 1: Parallel Coordinate Plot of data colored by Wine Class. The Sulphur variables dominate in both datasets.

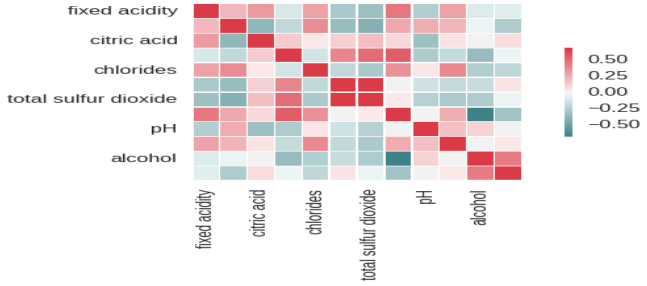

Figure 2: Correlation Matrix for Wine Data

III. Description of Models

A. *Perceptron*

Biological neural networks have some very desirable properties such as massive parallelism, distributed representation and computation, learning ability, generalization ability, adaptivity, inherent contextual information processing, fault tolerance, and low energy consumption[10] which are not present in modern computers. The motivation for developing artificial neural networks is to replicate some of these properties. Perceptrons are generalized variants of the McCulloch and Pitts neurons. These neurons computed a weighted sum of real valued inputs and generate an output of 1 if this sum is above a certain threshold U otherwise it returns 0. [10]

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right)$$

Equation 1 Mathematical neuron proposed by McCulloch and Pitts [10]

The generalization of this model, especially the threshold function, has enable the use different functions such as Sigmoid, piece wise linear and in our case rectified linear units. Artificial Neural Networks (ANN) can be visualized as a weighted directed graph. Here the nodes are the artificial neurons and the weighted directed edges are the connections between neuron outputs and inputs. Depending on the architecture these networks can be further classified into feed forward or recurrent neural networks. Simply put, recurrent networks are those that have loops thus networks without loops are feed forward[10]. Recurrent NN have applications in text classification, while feed forward networks under which MLP's fall are what is of interest to us[10], [11]. MLP are the most common type of feed forward network where the neurons are organized into layers or hidden layers that have unidirectional connections among them. Feed forward networks are static in the sense that they produce only one set of outputs for given input in contrast to a sequence. Also they are memory less in the sense that their response to the input is independent of the previous network state. [10]

Learning process for Perceptrons

Within the context of an ANN, the learning process can be viewed as a method of updating the network weights and architecture in an efficient manner so that it is able to perform the task assign to it in satisfactory manner based on the examples it encounters [10]. Learning in an ANN context is therefore the ability to approximate the underlying behavior of the training data adaptively. Generalization which is the most desirable property, is the ability of this trained model to make predictions based on new data. But this approximation ability of ANN make them susceptible to overfitting, which is the case when the model fits the training data well and fails on new data. Therefore, to reduce overfitting consider network size, sample size, model selection, and feature selection [11].

Thus for the classification case this becomes a mapping F: Rd →RM where a d-dimensional input is submitted to the network and an M-vectored network output is obtained to make the classification decision. The network is typically built such that an overall error measure, Crossentropy (CE) in our case is minimized. Cross entropy is chosen as op-posed to Mean Square Error because they have been shown to have better network performance with a shorter stagnation period [13].

Loss Functions

The key idea behind CE is to transform the original combinatorial optimization problem to one of stochastic optimization and then to solve this stochastic optimization problem as efficiently as possible using an adaptive sampling algorithm. Doing this constructs a random sequence of solutions which give probabilistic convergence to an optimal or acceptable solution. Once the stochastic optimization process is set up the CE method does the following: [14]

- Generate a sample of random data according to specified mechanism
- Update the parameters of the random mechanism on the basis of the data to produce a better sample in the next iteration.

Activation Functions

In our MLP we used the RELU function. RELU is not a symmetric function and so the mean response is therefore always not smaller than 0. RELU's were first proposed to help Restricted Boltzmann Machines better learn features from natural images with pixel intensities [15]–[17]. The RELU function is identity for positive valued inputs and zero otherwise see Fig 4. One of the key advantages is that it alleviates the vanishing gradient problem associated with backpropagation in deep networks because it has a derivative of

1 for positive values and therefore not shrinkable. However, it must be noted that RELU's are non-negative and as a result have a mean activation larger than 0[15]. Also, the RELU can produce asymmetric outputs even if the input falls under a symmetric distribution. This is due to their asymmetric nature. [12]. It must be noted that RELU have better performance than traditional sigmoid and 'tanh' activation functions, recent extensions such as Parametric RELU (PRELU), Leaky RELU[16], Exponential LU[15] have been proposed and shown to have superior performance compared to RELU. For our experiment, we note that the RELU gave good performance but we mention them here and discuss them in the Further Work section as potential extension to our work.

Dropout

Dropout method is a powerful technique for the improvement of generalization error in neural networks. We use it in our MLP architecture as a means to prevent overfitting and improve generalizability. Dropout encourages the network robust co-adaptations of the hidden unit feature detectors, by adding a certain type of noise in the hidden units in the forward pass of training. This noise has the effect of cancelling or zeroing out a fixed fraction of activations in the neuron in the given layer. Dropout encourages the network to redundant representations of the data and can be thought as training a large number of ANN with different connectivity patterns and tied weights for all units that are not dropped. By randomly sampling the examples that remain for each training epoch, this averaging can be performed in an efficient way. In essence, this is like adding noise to the hidden unit activation functions during the training pass. The noise tends to zero or force the dropout of a fixed fraction the activation neurons in a given layer. In our MLP we use a value of 0.2 as a trade-off between faster learning and accuracy. Dropout does not occur during testing, but instead we multiply the final output from the layer below by a factor 1/ (1-r), where r is the dropout probability for units in the layer below. We choose a relatively small value for Dropout as this tends to slow down the learning process. [12]

Learning Rule: Adam Optimizer

For the optimization method we used the Adam optimizer [18]. Adam is a first order gradient based method that aims to combine the advantages of RMSprop [19] for online and non-stationary settings and Adagrad [20] which works well with sparse gradients. The key advantages of this method is that it is computationally efficient, has low memory requirements, it is invariant to diagonal scaling, requires little and in our case no hyper parameter tuning and is well suited for large datasets or those with large number of variables. Adam computes individual adaptive learning rates for the different parameters based on estimates of first and second moments of the gradients. Also the Adam, method does not require a stationary objective function, its step sizes are bounded are approximately bounded by its step size hyper parameters and performs a form of step size annealing. [18]

As we will show the Adam method is a better choice than SGD with Nesterov Momentum for our dataset. We tested SGD against Adam for the same network architecture and found SGD to be slower and in some cases failed to converge within the maximum number of training epochs. The purpose of our comparison was to benchmark SGD and Adam so we took default values in Keras without hyper parameter tuning. Apart from being very efficient the Adam, method has robust parameter values suggested in the literature which in our case required no tuning and produced superior performance. These are the practical reasons why this is chosen over traditional backpropagation.

We used the key parameter values [18]:

- Learning Rate, $\alpha = 0.001$
- Decay Parameters, $\beta1 = 0.9$ and $\beta2=0.999$, for the moving averages of the gradient and squared gradient calculated by the algorithm
- Epsilon, $\varepsilon = 10-8$ which is a fuzz factor

Table 1: Final MLP Architecture

1.	Input Layer: 12 Input Neurons, 16 Output Neurons
2.	RELU Activation Function
3.	Dropout: r=0.2
4.	Hidden Layer: 16 Neurons
5.	RELU Activation Function
6.	Dropout: r=0.2
7.	Output Layer: 2 Neurons
8.	Softmax Activation Function

Table 2: Advantages and Disadvantages of MLP's

Advantages	**Disadvantages**
A single layer perceptron can only represent linearly separable classes, with the perceptron convergence theorem guaranteeing convergence after a certain number of iterations. But we do not know what this number of iteration are. [10]	This fails for the non-linear case and its failure to represent the XOR problem is well known in this regard.
Multilayer Perceptrons can represent non-linear decision surfaces. With three layers being capable of representing any arbitrary decision surface as a result of Kolmogorov Theorem.[21]	But the convergence property does not hold for higher number of layers.
Momentum term added to gradient descent algorithms to get the model out local minima are a way to get around this.	Gradient descent style optimization can lead to the model being stuck in local minima and miss global maximum.
Massively parallel, fault tolerant, robust to noisy data and fast to evaluate new examples.	Computationally expensive learning process

We use a small learning rate with a small amount of momentum to prevent the local minimum issue and achieve convergence during training. We used early stopping while training the MLP,

with 5 epochs with no improvement in the validation loss causing it to terminate. The fact that there is a small learning rate requires a large number of training epochs before this condition is met. We also initialize the network with random initial weights as other initializations did not yield any practical benefits in our case.

Optimum number of neurons

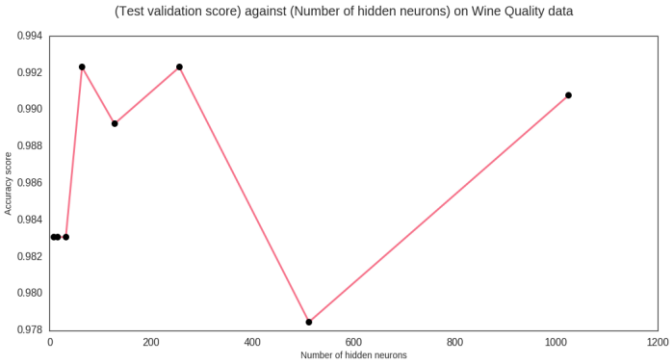

Figure 3: This plot shows the performance of a Perceptron with varying numbers of neurons in its input and hidden layer. Here we used 90/10 test train split with a 20% validation set derived by bootstrapping from the test set. The point of this was to get an idea of the performance of the network with different number of neurons. The number of neurons tested were [8, 16, 32, 64, 5256, 512, and 1024] and their accuracy score on the test set. We kept a cylindrical structure in our hidden layers for simplicity so both layers had the same number of neurons. All values for the accuracy are in the range of 97-99%. The network with 8 and 16 neurons we are able to have a very high accuracy score. Since 8 would be less than the number of input dimensions and to keep our model flexible we choose 16 neurons for all layers. This testing is performed with a single hidden layer although we test an architecture with 2 hidden layers. This is unnecessary for this problem as we show. During this testing we use Dropout at each layer to ensure that with variable number of neurons we get the best generalization error [12]. But we test a network architecture with and without Dropout for comparison.

In our experiment, we used a small batch size to ensure accurate learning along with the parameters described with the early termination over 100 training epochs.

Optimal Perceptron Architecture

We took a rigorous approach to testing the MLP architecture to derive one that learns fast, generalizes well and the model is fairly compact. In Fig 4, we show the different configurations tested. We tested the traditional 'tanh' activation units with a sigmoid function at the output layer. For this combination we used SGD in all cases and considered cases with and without Dropout and with 1 and 2 hidden layers. In our testing apart from the 2 hidden layer case the 'tanh', sigmoid combination failed to converge within the maximum number of training epochs. So we decided not to use the 'tanh' function and tried RELU. Even though Dropout increases training time as noted in the literature and in our testing, the best results we get are with the RELU and the Adam optimizer with Dropout at each layer. The RELU case with SGD is slow in comparison. The Softmax and Sigmoid cases for the RELU are interesting because even though the Softmax required more time it required around 20 less training episodes to converge when compared to the Sigmoid case. Each training case was quicker with the sigmoid but there were more iterations. We chose the RELU with the Softmax, Adam and Dropout combination for final evaluation. Also worth noting is that our RELU , Sigmoid, Adam and Dropout architecture is around 86% faster in our testing than the equivalent architecture with Tanh, Sigmoid, SGD and Dropout. Hence our RELU and Adam choices are justified based on their clearly superior performance.

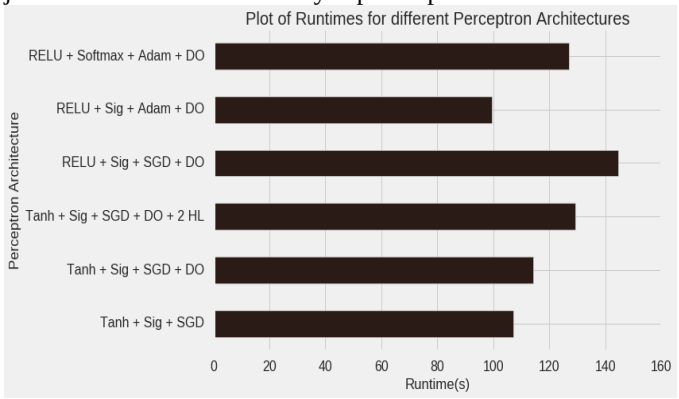

Figure 4: Training times for different Perceptron architectures.

B. Support Vector Machine

The Support Vector algorithm is a nonlinear generalization of the Generalized Portrait Algorithm developed in Russia in the sixties [22]. They are class of kernel methods that enable us to deal with non-linear problems by mapping nonlinear inputs to high dimensional spaces via linear patterns. They have the advantage of not getting stuck in local minima like multilayer neural. [23]

How the SVM works in the classification case, is by finding a hyperplane induced from the maximum margin principle, which translates to finding the hyperplane best separating the classes. In our experiment this would be the red and white wines based on the input attribute, with an error cost C > 0[23], [24]. Hinge loss is a common choice [23]. This means that during training we are trying to learn a function that will minimize training error of the empirical risk, Remp shown in Equation 2, where ι is the loss function. By doing so we are trying to minimize the test error which is the overall risk.

Equation 2: (Left) Empirical Risk for an SVM Classifier. (Right) Overall risk for an SVM. [25]

$$R_{emp}(\alpha) = \frac{1}{m}\sum_{i=1}^{m} \ell(f(x_i,\alpha),y_i) = \text{Training Error} \qquad R(\alpha) = \int \ell(f(x,\alpha),y)dP(x,y) = \text{Test Error}$$

The general approach of kernel methods has the following key aspects: [23]

- Data embedding into a Euclidean feature space
- Search for linear relations within this feature space

- These only require the inner product of vectors in this feature space
- These products can be efficiently computed via the kernel function from the data. This is often referred to as the kernel trick

Kernels provide a powerful way to modelling nonlinear patters through linear patterns. The most common types of kernels used are:[23]

- Linear Kernel defined as , $\kappa(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$ k is the kernel function
- Polynomial Kernel defined as, where d is the degree of the polynomial and parameter R ∈ R $\kappa(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + R)^d$
- Gaussian Radial Basis Function (RBF) $\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$, $\sigma > 0$ $= K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$.
- Sigmoid $K(x_i, x_j) = \tanh(a x_i^T x_j + r)$ [24]

The advantages of SVM's are as follows:

- With the introduction of kernels, SVM become flexible since they can solve a non-linear problem in a higher dimensional space.
- It is an effective method even when the number of dimensions (features) is bigger that the number of observations.
- SVM provide a good out-of-sample generalization if the hyper parameters are chosen appropriately. [22]
- The result of Support Vector Machines is unique, in contrast to the outcome of a Neural Network. The latter might produce multiple solutions by falling into local minima which will reduce the generalization capabilities of the model. [22]
- SVM can also be memory efficient since they use a subset of the available training samples in the decision function.

The disadvantages of SVM's are as follows:

- Support Vector Machines share a drawback with other non-parametric algorithms. The results aren't transparent because the answer was probably given in a higher dimension. However, it is possible to interpret the results by visualizing the outcome of the procedure. [22]
- They don't provide probability estimates directly but these have to be calculated using cross-validation. The application of cross validation is computationally costly, especially if the dataset is large.
- The model will perform poorly if the number of dimensions is much higher than the sample size.

Grid Search for SVM Parameters

The SVM is a sophisticated model that requires a lot of parameter tuning. As we have noted that there are a choice of kernels, with their associated regularisation values (C) and kernel parameters (Gamma) where applicable. Therefore to derive the best parameters for this dataset we perform a grid search for linear, polynomial and RBF kernels and with C and Gamma values [0.001, 0.1, 1, and 10]. Grid Search method implements an exhaustive search through these values. We used Stratified Random Split in order to validate the results of this model. This method returns stratified random folds. We chose to do 5 reshuffling and splitting iterations while the size of the validation set was the 20% of the training. The 5 iterations are used for practical considerations.

Our grid search returned a **C value of 10.0, a Gamma of 10.0 and the RBF kernel** as the best values for our problem. We restrict our discussion to the RBF kernel because this is the best kernel to use for our data according to the grid search.

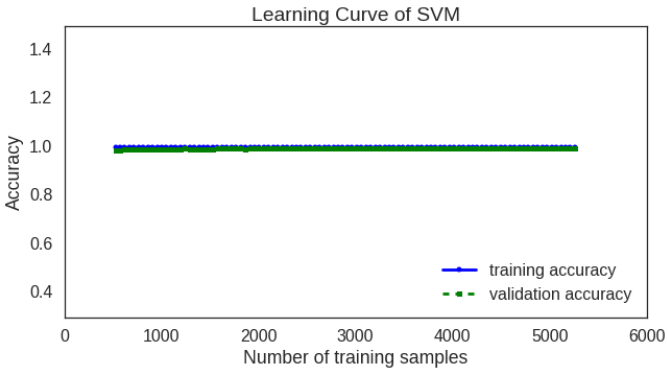

Figure 4: Shows the learning curve for the SVM with a Gaussian kernel and parameters chosen via grid search. This learning curve is generated with 90/10 train test split with 10 fold cross validation. The Cross validation performs bootstrapping from the training data and helps to reduce overfitting in the model by training and testing the model on different parts of the data.

IV. Data Processing and Modelling

In this section we describe our analysis methodology step by step. Since most of the technical details for the models and their justifications have been discussed in previous sequence. They are included here so they can be seen in context of the whole analytical pipeline.

Step 1: Data Import

1. Import red and white wine data from UCI and add class labels [0, 1] for red and white wines.
2. Check for null values. There are no null values in this dataset
3. Compute basic statistics such as mean, standard deviation and visualise using boxplots and scatter matrices of features, heat maps of correlation.
4. Merge red and white wine data into unified dataset. At this stage we do not apply any sampling strategies such as SMOTE [7] or over sampling. This is the under sampling case.
5. Separate data into features and targets
6. There are lots of variations on scale in the data and there outliers so use Min Max Scaling to scale data to [0, 1] interval.
7. Visualise feature scaling using boxplots to ensure data scaled correctly show in Fig 3.
8. Pick out outliers using IQR. The Robust Covariance Determinant Method for calculating outliers based on Mahalanobis distance was not applicable due to non-Gaussian nature of data.

Step 2: Modelling

9. We split the features and targets from the previous step into training and test set. We use a 90/10 split for the train/test in our first pass model building and evaluation
10. We start with a Multilayer Perceptron, with 2 hidden layers and with a (RELU) [12] and the Adam optimizer [18] and a Dropout (p=0.2) at each layer [12] and Cross entropy loss function with a 2 way softmax at the output layer.
11. We test the network with 1024, 512, 256, 128, 64, 32, 16, 8 neurons at each layer to determine the minimum number we need for this problem. We determine that 16 neurons is sufficient giving 98% accuracy with 64 neurons being required for 99% shown in Fig 4. Since the performance difference is marginal we go for the smaller network.
12. MLP Cross Validation: The validation split parameter was added in the MLP model. It accepts a float number between 0 and 1 which represents the fraction of the training data that will be used for the validation set.
13. We compare our MLP with a SVM Classifier. We perform a grid search for best values for the SVM to find the best kernel, gamma and C value for our dataset. Which we determine to be a RBF kernel, with C = 10.0 and gamma=10.0.
14. SVM Cross Validation: We used Stratified Random Split in order to validate the results of this model. This method returns stratified random folds. We chose to do 5 reshuffling and splitting iterations while the size of the validation set was the 20% of the training.
15. To evaluate our classifier we use common classification metrics such as Precision, Recall, F1-Score, Confusion Matrix and ROC AUC Score.

Step 3: Testing and Evaluation

16. Since we have noted a class imbalance, we repeat our training and testing of our models with oversampling of the red wine data, with z-score normalization and with added random noise. [9], [16]
    <u>This extra pass of training and testing is referred to as 2nd pass from now on.</u>
17. We take this transformed feature data, split it into training and test set and we evaluate our models again.
18. First, we do the usual train/test split of 90/10, then to make it more difficult we do a 60/40 split and to make it even harder we do a 20/80 split and evaluate the models. Even at the 20/80 split end the MLP and SVM give AUC of 0.98-0.99. This gives us confidence in the robustness of our models to noise in the data and ability to handle class imbalance.

V. RESULTS AND EVALUATION

For model evaluation we use common classification metrics such as Precision, Recall, F1-Score, Receiver Operator Characteristic, Area under the Curve and Confusion Matrices for both our models for both phases of evaluation shown in Fig 5, 6 and Table 3. The 1st pass referrers to the initial model building and evaluation stage. Here we take the red and white wine data as it is, merge and scale it to [0, 1] interval and then we train and test both the models. In the 2nd pass we use our AST methodology and quote the results at the 20/80 end where we observe some performance differences.

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$
$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Equation 3: **Precision (top)**: 'tp' means True Positives and represents the number that were correctly classified in the specific class and 'fp' means False Positives and represents the sum of examples that were incorrectly labelled to a given class. **Recall (mid)**: 'tp' means True Positives and represents the number that were correctly classified in the specific class and fp means False Positives and represents the sum of examples that was incorrectly labelled to a given class. Moreover, 'fn' means False Negative which is the number of items that were not classified in a specific class, but they should have. **F1 – Score (bottom)** is used to compute test's accuracy. It combines Precision and Recall since it calculates their harmonic mean.

We see that within this evaluation phase, the class imbalance identified previously is still present and the sample size for the overall dataset is fairly small and consists of a total 6497 samples of these 650 make it to the test set. Within this sample we have a 1:3 class imbalance of the white wine to the red wine so we can be skeptical of the high scores across the all metrics for both classes for both models. We note almost identical performance of both the SVM and MLP for this particular instance of the evaluation with the data issues highlighted.
The Precision, Recall and F1 measures are heavily biased towards the class with the most samples. Since all the metrics are so good for our models, we wanted to test them further and truly determine how robust our models so we propose a novel Asymmetric Stress Testing methodology.

Asymmetric Stress Testing

We propose a novel model evaluation framework based on asymmetric splits of the training and test data. We show that using this method when presented with two seemingly equally good models we are able to distinguish between them. In this case our methodology enables us to prove our initial hypothesis that the MLP is a more robust method even when there is significant noise present in the training data. The first step of this process is to generate new data from the existing data. We do this by taking the raw data, applying a z-score normalization and then adding some random noise.

To address the class imbalance noted previously we oversample the underrepresented class in this data and shuffle the items. There is no reason why other commonly used data transformation methods such square root, log, inverse and maximum absolute scaling cannot be used. As an extension one might even combine the outputs of these separate normalizations into one cohesive dataset to perform this evaluation. Another extension to this step could be to apply a Radon transform to the data, which is commonly used in Image Analysis and Seismic Data Analysis for rotation and noise removal, to compute a rotations of this data. Other common mathematical transforms such as Fourier and Hilbert Transforms could be used as well to compute interesting representations of the input data for further testing. Discussion

of these methods is beyond the scope of this paper so we leave it to the reader to refer to literature.

The next step is to split this data into features and target and then split the data into training and test set. We start with a 90/10 train and test split we evaluate our models. We repeat this step again this time with a 60/40 split and then again with a 20/80 split. The idea is to simulate the model being in an environment where it encounters new data and has less and less to learn from each time. It is at this extreme end where we note the interesting differences that we discuss later. At this end of our testing we see a significant divergence in the performance of the SVM with respect to the MLP so we quote and discuss these results from our 2nd pass (20/80 split).

We observe that the MLP is more robust, in this case that faced with noisy data it is still able to maintain an AUC score of 0.99. The MLP performs consistently across all the evaluation metrics but the SVM experiences a 33% fall in the precision for Class 1 which we oversampled in the 2nd pass. Also the Recall for Class 0 is 29% worse than before. The F1-Score for accuracy which is the harmonic mean of Precision and Recall shows an average of 20% or more fall for both classes for the SVM compared to the 1st pass. No such change is observed from the MLP for this phase of testing. Hence, we show that our method is very useful in model selection and in gaining greater insight into their generalization ability.

The biggest limitation for the SVM is the kernel choice even though we determined the RBF kernel as being the best from the grid search but this was based on the previous data. The only other parameter that is of significance is the regularization term C, which we determined to be 10. This is a very high value and would probably restrict the flexibility of the SVM. It could be investigated whether a lower C value results in better SVM performance under the AST framework in the future.

Table 3: Classification metrics for MLP and SVM for both passes of training

	Pass 1 (No AST)				Pass 2 (With AST)			
	MLP		SVM		MLP		SVM	
Metrics	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
Precision	1	1	1	1	1	0.99	1	0.67
Recall	1	1	0.99	1	0.99	1	0.61	1
F1-Score	1	1	1	1	0.99	0.99	0.76	0.8

VI. FURTHER WORK AND CONCLUSION

This work can be extended in a number of ways and this is not meant to be exhaustive. There is a well identified issue with class imbalance and this dataset and we have shown how over sampling and under sampling affects model results. This can be extended further by utilising the SMOTE method[7] and repeating the testing and evaluation as proposed. This data can also be augmented in other such as with different scaling methods as has been already suggested. Also, the novel Asymmetric Stress Test methodology can be applied to new datasets and machine learning models and can serve to give us

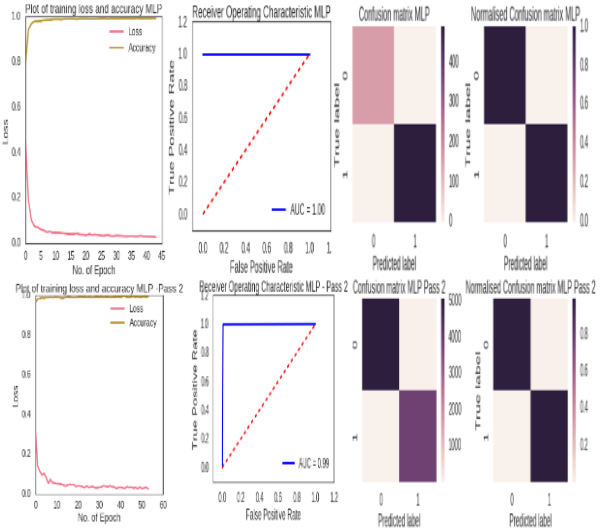

Figure 5: **:** Shows plots of (left) training loss and accuracy, (mid) ROC plot and AUC score, (right) Confusion Matrix both regular and normalized for both passes of MLP training and evaluation. We see the MLP performs very well across both phases of training and evaluation.

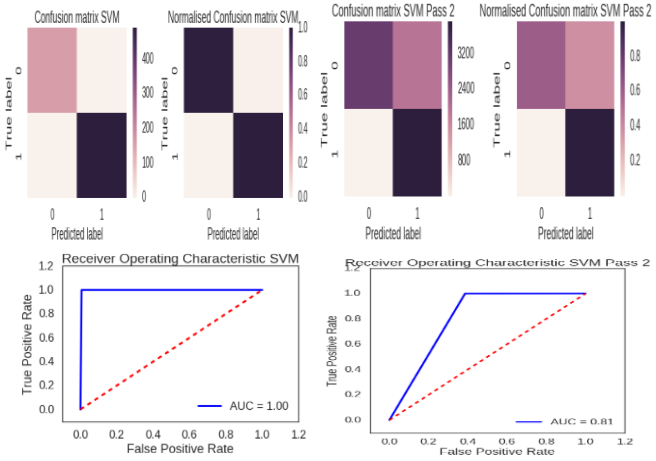

Figure 6: Shows Confusion Matrices for both passes of evaluation for the SVM and the associated ROC Curve and AUC scores and the Learning Curve for the first pass of SVM training. We see that the SVM performs well on the initial data which we have had hyper parameter optimization but does not generalise as well as the MLP in the second phase possibly due to the initial parameterization being too rigid for this case.

additional insight when it is feasible. This may or may not be feasible depending on the dataset size. As previously noted, there are a number of extensions that have been proposed in literature and shown to have better performance than the RELU activation we used. Some of these could be investigated with variations of the network architecture and their performance can be evaluated as we have done. We have utilised CE and the Adam optimizer other loss functions such as Hinge Loss and Cosine proximity could be investigated. Also other optimization routines such as RMSprop[19], Adagrad[20], Adadelta, Adamax, [18] could be explored and their rate of convergence compared with the Adam optimizer. Other evaluation metrics such as Cohens Kappa[27] and Gwet's AC1[28] can be applied as alternatives to ones used here.

In conclusion, we have shown that MLP and SVM can achieve high levels of accuracy in wine classification based on chemical and quality attributes. We have shown that MLP in this case outperformed SVM in terms or robustness and generalizability. We also show prove our working hypothesis through our experiment that the MLP is a more robust method. It can be said that an MLP model is better suited to be implemented as a classifier in the wine certification process and is better placed to aid in automation efforts in this regard and reduce reliance on expert systems. We have also proposed a novel way to Asymmetrically Stress Test our models and have shown practically its effectiveness.

REFERENCES

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decis. Support Syst.*, vol. 47, no. 4, pp. 547–553, 2009.

[2] F. Pérez and B. E. Granger, "IPython: a System for Interactive Scientific Computing," *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 21–29, May 2007.

[3] "Keras: Deep Learning library for Theano and TensorFlow." [Online]. Available: http://keras.io/. [Accessed: 13-Feb-2016].

[4] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio, "Theano: new features and speed improvements," *arXiv Prepr. arXiv …*, pp. 1–10, 2012.

[5] J. Bergstra, O. Breuleux, F. F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math compiler in Python," *Proc. Python Sci. Comput. Conf.*, no. Scipy, pp. 1–7, 2010.

[6] F. Pedregosa and G. Varoquaux, "Scikit-learn: Machine Learning in Python," *J. Mach. …*, vol. 12, pp. 2825–2830, 2011.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, 2002. [Online]. Available: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02a-html/chawla2002.html. [Accessed: 13-Feb-2016].

[8] A. Krizhevsky, Ii. Sulskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.

[9] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Doc. Anal. Recognition, 2003. Proceedings. Seventh Int. Conf.*, pp. 958–963, 2003.

[10] J. Mao, "Why artificial neural networks?," *Communications*, vol. 29, pp. 31–44, 1996.

[11] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Trans. Syst. Man Cybern. Part C (Applications Rev.*, vol. 30, no. 4, pp. 451–462, 2000.

[12] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, pp. 8609–8613, 2013.

[13] G. Nasr, E. Badr, and C. Joun, "Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand.," *FLAIRS Conf.*, pp. 381–384, 2002.

[14] S. Mannor, D. Peleg, and R. Rubinstein, "The cross entropy method for classification," *Proc. 22nd Int. Conf. Mach. Learn. - ICML '05*, pp. 561–568, 2005.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *Under Rev. ICLR2016，提出了ELU*, no. 1997, pp. 1–13, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th International …*, 2010. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf. [Accessed: 15-Feb-2016].

[18] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv1412.6980 [cs]*, pp. 1–15, 2014.

[19] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop Divide the gradient by a running average of its recent magnitude," *COURSERA Neural Networks Mach. Learn.*, vol. 4, p. 2, 2012.

[20] John Duchi,Elad Hazan, *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. 2010.

[21] V. Kůrková, "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*, vol. 5, no. 3, pp. 501–506, Jan. 1992.

[22] B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, 2004. [Online]. Available: http://www.springerlink.com/index/KM7KRM46802R2114.pdf\nfile:///Users/dr/Documents/Papers2/Articles/2004/Sch?lkopf/Statistics and Computing 2004 Sch?lkopf.pdf\npapers2://publication/uuid/B1760828-D940-47C8-B75B-2EC82283920E. [Accessed: 16-Feb-2016].

[23] J. Shawe-Taylor and S. Sun, *Academic Press Library in Signal Processing: Volume 1 - Signal Processing Theory and Machine Learning*, vol. 1. Elsevier, 2014.

[24] H. Lin and C. Lin, "A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods," *Submitt. to Neural Comput.*, pp. 1–32, 2003.

[25] J. Weston, "Support Vector Machine (and Statistical Learning Theory) Tutorial," 2011. [Online]. Available: http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf. [Accessed: 16-Feb-2016].

[26] C. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[27] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes," *Pattern Recognit.*, vol. 44, no. 8, pp. 1761–1776, Aug. 2011.

[28] N. Wongpakaran, T. Wongpakaran, D. Wedding, and K. L. Gwet, "A comparison of Cohen's Kappa and Gwet's AC1 when calculating inter-rater reliability coefficients: a study conducted with personality disorder samples.," *BMC Med. Res. Methodol.*, vol. 13, p. 61, 2013.