AN ANALYSIS OF SUPERVISED MACHINE LEARNING ALGORITHMS

Zakaria Gorgis
University of California, San Diego
Email: zgorgis@ucsd.edu

1 Abstract

As the field of Machine Learning grows, so does the obsession with finding the best performing algorithm. While there are a plethora of different subsets of Machine Learning, the focus of this paper is to look at five supervised ML algorithms; Logistic Regression, Multilayered Perceptrons, kNearestNeighbors, Random Forests, and Support Vector Machines. The accuracy of these algorithms were tested using the best hyper parameters under three main error metrics, logistic loss (cross-entropy loss), Receiver Operating Characteristics, and F_1 score, on five different data sets, taking inspiration from the comparison of Supervised Learning Algorithms conducted by Rich Caruana and Alexndru Niculescu-Mizil at Cornell University.

2 Introduction

"There is no such thing as a free lunch" is a strange statement to assert, but none the less very pivotal to its derivative, the No Free Lunch Theorem for supervised machine learning algorithms. In essence, there is no singular algorithm that is capable of solving every problem. There are too many idiosyncrasies that disrupt any one method of classification. No matter how capable an algorithm might seem, it needs just one "black swan" to distort the confidence of accuracy. Because of this, it's imperative to any machine learning engineer to diversify their collection of algorithms and cater these algorithms to each problem as necessary.

3 Methods

This paper looked at a small subset of data sets, all with varying attributes and classification goals. Each algorithm was ran using a Grid Search with Stratified KFold validation on 5000 data points for a total of 5 trials. The procedure was repeated 5 times for each algorithm, for each data set. The results were subsequently recorded, and the same algorithms were ran on the testing set to better accurately indicate the effectiveness of classification by the algorithms.

3.1 Classifiers

The following are the classifiers that were used in the project and their hyper-parameters tuned using Grid Search.:
Logistic Regression : Logistic Regression classifier was trained using four different solvers; SAGA, LBFGS, NEWTON-CG, and LIBLINEAR. **SAGA** (variant of stochastic average gradient) was implemented with both L1 regularization (Lasso Regression) and L2 regularization (Ridge Regression). **LBFGS** (Limited memory Broyden-Fletcher-Goldfarb-Shanno Algorithm) was implemented with L2 regularization. **NEWTON-CG** (Newtons Method) was implemented with L2 regularization. **LIBLINEAR** (Large Linear Classification) was implemented with L2 regularization, All of these solvers utilized the inverse of regularization strength C, at interval values $[1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03, 1.e+04]$.
Multilayered Perceptrons : Multilayered Perception was trained using three different solvers; LBFGS, SGD and ADAM. **LBFGS** (optimizer using quasi-Newton methods), **SGD** (stochastic gradient descent) and **ADAM** (stochastic gradient-based optimizer) were all tested using the following hyper parameters in Grid Search; The hidden layer sizes, representative of number of neurons in a hidden layer were sequestered at $(5,),(10,),(15,),(20,)$. The activation function for the hidden layers were to either $logistic$ or $tanh$. $Logistic$ representing the logistic sigmoid function;

$$f(x) = \frac{1}{1+e^{-x}}$$

Tanh representing the hyperbolic an function;

$$f(x) = tanh(x)$$

The α, or the L2 penalty parameter was set at a default interval of $[0.0001, 0.001, 0.01, .1]$.The learning rates were invscaling, adaptive, and constant.

kNearestNeighbors: KNearestNeighbors was trained using four different metrics,

Distance Metrics Used	
Euclidean	$d_1(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$
Manhattan	$d_1(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|_1 = \sum_{i=1}^{n}\|p_i - q_i\|$
Chebyshev	$d_1(\mathbf{p},\mathbf{q}) := \max_i(\|p_i - q_i\|)$
Minkowski	$d_1(\mathbf{p},\mathbf{q}) = (\sum_{i=1}^{n}\|p_i - q_i\|^n)^{\frac{1}{n}}$

Each of these distance metrics were used with 25 different prime N values, ranging from $N = 5$ to $N = 99$. Two different prediction weights were used, uniform (points are weighted equally) and distance (points are weighted by their inverse). Within each of the distance measurements, three different algorithms were used to find nearest neighbors, Brute, ballTree, and kdTree.
Random Forest: Random Forest was trained using three different max features, used to calculate the best split, auto, sqrt, and log2. The number of estimators, (trees) was set to $[100, 200, 400]$, max depth was set to $[7, 9, 11, 13]$ and the criterion to measure split quality was set to Gini impurity or entropy.
Support Vector Machines The support vector machine was trained using four different kernels;

Support Vector Machine Kernels Used	
RBF	$k(x_i,y_j) = exp(-\gamma\|x_1 - y_j\|^2)$
Sigmoid	$k(x,y) = tanh(\alpha x^T y + c)$
Poly	$k(x_i,y_j) = (x_i x_j + 1)^d$
Linear	$k(x,y) = 1 + xy + xy \cdot min(x,y) - \frac{x+y}{2} \cdot min(x,y)^2 + \frac{1}{3} \cdot min(x,y)^3$

For each of these kernels, the Gamma value (kernel coefficient) was set to intervals of $[1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e-0]$, while the regularization parameter C was set to intervals of $[.0010, .010, .10, 1, 10, 100, 1000, 10000, 100000]$.

3.2 Performance Metrics

During each sweep of Grid Search, three different error metrics were calculated. Log Loss, ROCAUC and F1.
Log Loss Error: In a binary classifications scenario, the Log loss is formulated as;

$$L_{\log}(y,p) = -(y\log(p) + (1-y)\log(1-p))$$

While for general cases involving multi-class classification, it is known as cross-entropy loss and is formulated as;

$$H(p,q) = -\sum_i p_i \log q_i = -y\log\hat{y} - (1-y)\log(1-\hat{y})$$

It is important to show both of these equations and their similarity, as not all of the data sets in the paper are binary classification problem. Namely, The Dry Bean Data Set from *UCI* is a multi-class classification problem. The Log Loss error is helpful to determine how close the predicted classification is in correspondence to the actual classification. This again can be represented as a binary or multi-class.
ROCAUC: The Receiver operating characteristic is generally recognized as a probability curve in which a relationship is drawn between the True positive rate, and the False positive rate. The true positive rate (also known as sensitivity or recall) is calculated as;

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN} = 1 - FNR$$

Where TP is the true positive, TN is the true negative, FP is the false positive, and FN is the false negative. The False positive rate (also known as the fall-out) is calculated as;

$$FPR = \frac{FP}{N} = \frac{FP}{FP+TN} = 1 - RNR$$

Where TP is the true positive, TN is the true negative, FP is the false positive, and FN is the false negative. The area under curve, or AUC represents the measure of separability between the false positive rate and true positive rate. The ROCAUC is an important error metric to study a classifier in relation to the discrimination threshold.
F1 Score: The F1 score is an important extension of the ROC curve, as it exemplifies the mean of recall and precision.More commonly known as;

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Understood to be a harmonic mean, the $F1$ is more formally calculated as;

$$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR}$$

Where PPV is the positive predictive value and TPR is the true positive rate.
To attempt to simulate real world situations in which data does not always come clean and pre-processing, the selection of data sets was done in hopes of varying degrees of attributes, sizes, and classification information. Because of this, all of the data sets went under some pre-processing stage, which will be described.

The first data set used was the **Fire Wall Classification Data set**. The goal of this data set was to classify internet traffic records to determine if the incoming data was either allowed, requires an action, drop, or reset. The dependent y, in which the classification was made was categorical. To work around this, the data was pre-processed by using Label-Encoder to transform the y column into discrete numeric values ranging from $[0, 1, 2, 3]$.(4 classifications) The second data

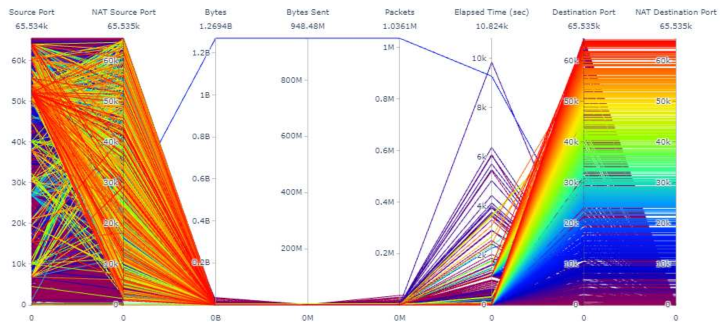

Fig. 1. Parallel Distribution of Fire Wall Data set

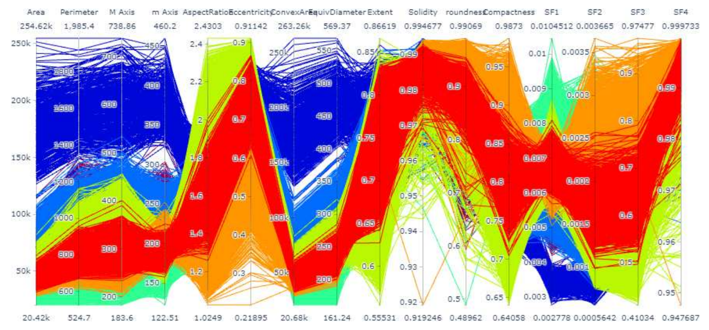

Fig. 2. Parallel Distribution of Dry Beans Data set

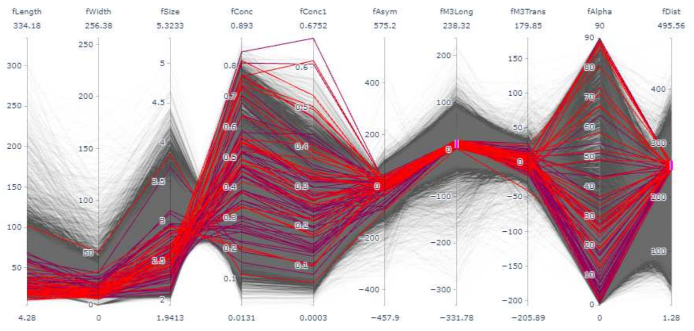

Fig. 3. Parallel Distribution of Gamma Data set

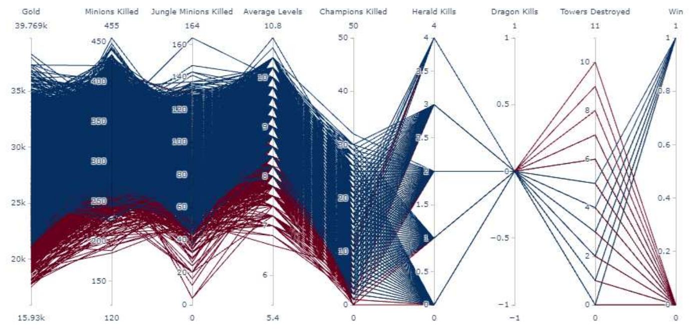

Fig. 4. Parallel Distribution of LoL Data set

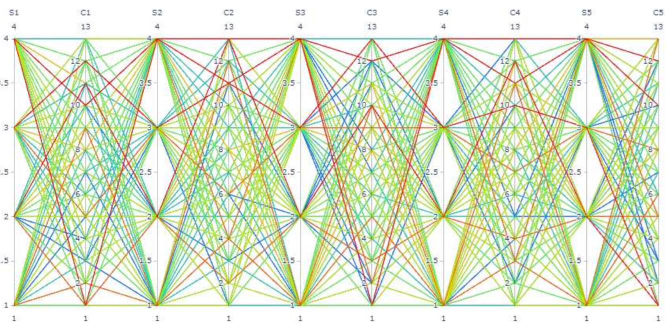

Fig. 5. Parallel Distribution of Poker Data set

set that was used was the **Dry Bean Data set**. The goal of this data set was to classify dry beans based on various characteristics such as area, eccentricity, and convexity. Like that of the previous set, the classification was categorical. To combat this, the data was pre-processed by using Label-Encoder to transform the y column into discrete numeric values ranging from $[0, 1, 2, 3, 4, 5, 6]$ (7 classifications). The third data set that was used was the **MAGIC Gamma Telescope Data Set** . The goal of this data set was to determine whether or not the high energy gamma particles found in the atmosphere were either gamma (h) or hadron (h) (binary classification). Again, the *y* values were once again categorical, quickly corrected by using Label-Encoder. The Gamma set also had an unnecessary column of labeled column values that were dropped to maintain accuracy. The fourth data set that was used was the **League of Legends Diamond Games (First 15 Minutes)**.
This data set gave various game statistics at the 15-minute mark, and the goal of the classification is to determine whether or not the blue team wins (binary classification). Two columns from the data set were dropped, one contained a column numerical label while the other was the match ID, which has no value to the classification goal.
The final data set is a wild card, both metaphorically and literally. The final data set that was used was the **Poker Hand Data Set**. The original goal of this data set was to predict the class of a 5-card hand, given all of the cards and their suits. Because of this, the data set had a total of 10 classes ranging from nothing to Royal flush. However, because of the rarity of the Royal flush, errors would arise from the Grid Search implementation as the $yTest$ values size vector would not be equal to the $yTrain$ values size vector. A solution to the problem was found, which will be discussed in the results section. But because of the volatility of the data set, and the skewness of the accuracy results through all three metrics, the data set will be discussed separately, addressing the importance of data selection.

Table 1 : Data Set Information

Name	# ATTR	Train Size	Test Size	y	%
FIRE WALL	12	5000	60532	ALLOW	57%
				DENY	23%
				OTHER	20%
DRY BEANS	17	5000	7611	DERMASON	28.26%
				SIRA	20.90%
				SEKER	16.07%
				HOROZ	15.28%
				CALI	12.92%
				BARBUNYA	10.48%
				BOMBAY	04.13%
GAMMA	11	5000	14020	GAMMA	64.83%
				HADRON	35.17%
LoL	19	5000	45000	WON	50.53%
				LOST	49.47%
POKER	11	5000	20000	2/3/4	06.87%
				ELSE	93.13%

Fig. 6. Since some data sets are not binary classifications, the y dependent vector is split into its respective classifications with percentages.

4 Results and Analysis of Algorithms

4.1 Random Forest

From table 2 and table 3, the assumption can be made that Random Forests generally had a high accuracy across all three error metrics, and across all data sets. Its accuracy when using LOG LOSS hyper parameters was .87, ROCAUC at .91, and F1 at ,90. The Random Forest classifier is built on the decision tree algorithm. Random Forests classifiers are generally good at classification with data that is unprocessed and data that has magnitudes of variation/imbalance. In essence, there is no need for data normalization. Although the data was standard scaled, it probably would not have made a big difference in relation to Random Forests. More over, all of the categorical data was converted to discrete integer values in order to run against other algorithms. But this step could of also been avoided with Random forests. RFs are also generally quick with computations when compared to something like Support Vector Machines. For example, in the Dry Beans data set, a Stratified 5Fold on 5000 data points using Grid Search (with the hyper-parameters listed above) took RF on average about 90 seconds to compute, while SVM on the other hand generally finished from 5 to 6 minutes. A few magnitudes of order slower than RF. Before moving on, there are a few limitations to RFs that are important to touch on. First, RFs are notorious for being easily "over-fit" on training data and might become lack luster on testing data, with variations of classification. This was seen in some of the training data collected. RF sometimes would average an accuracy of upwards of 97 to 99 percent, while getting barely 90 to 92 percent on testing sets.That being said, 90 percent is impressive. While this isn't directly related, another possible pitfall of RFs is that they will generally not return the most optimal classification/solution. This is because the algorithm is designed on a greedy heuristic, AKA a greedy algorithm. It's this inherent paradox that the algorithm in hopes of finding a globally optimal solution will subsequently make the locally optimal choice. This could have its benefits, like the aforementioned run time, but might fall short through sub-optimal classifications/regression when dealing with gigabytes of features and data, this however is just speculation using limited knowledge.

Table 3: Normalized scores of each learning algorithm by DATASETS

	FIREWALL	DRY BEANS	GAMMA	LoL	MEAN
LR	0.971407	0.917339	0.7764521	0.7810038	0.861550
MLP	0.978785	**0.9234616**	0.8116067	0.66412856	0.8444956
KNN	0.967443	0.908658	0.8008225	0.7736338	0.862639
RF	**0.98923**	0.914075	**0.870282**	**0.8098206**	**0.89585212**
SVM	0.9614266	0.922079	0.769674	0.785403	0.859646

Fig. 7. The table shows all of the normalized scores for each algorithm based on Data sets (Notice the low accuracy rate of MLP for LoL). using 5 Algorithms, LR, MLP, KNN, RF, SVM

Table 2: Normalized scores for each learning algorithm by METRICS

	LOG LOSS	ROCAUC	F1	MEAN
LR	0.86509605	0.85508330	0.8519588	0.85737941
MLP	0.8214603	0.84929791	0.8124604	0.8277395
KNN	0.8632551	0.870525	0.8581378	0.8639729
RF	**0.8719237**	**0.9131673**	**0.9074237**	**0.8975049**
SVM	0.86969704	0.8463066	0.84893434	0.8549793

Fig. 8. The table shows all of the normalized scoers for each algorithm based on metrics. The metrics are as following LOG LOSS, ROCAUC, and F1 using 5 Algorithms, LR, MLP, KNN, RF, SVM

4.1.1 Complexity of Random Forest

Because Random forests are built on decision trees the worst case can be done by analyzing decision trees using masters theorem of a recurrence relation given as;

$$\begin{cases} T(1) = c & iff \quad n < d \\ T(n) = aT\frac{n}{b} + f(d) & iff \quad n \geq d \end{cases}$$

If we look at the worst case recurrence of a decision tree, we are able to proof its complexity, where t(n) is the time taken to build a tree from n samples, c is the constant time taken to making a leaf node, and c(n) is the time taken to finding a split;

$$\begin{cases} T(1) = c \\ T(n) = c(n) + T(1) + T(n-1) \end{cases}$$

Taken from *Understanding Random Forests: From Theory to Practice*, written by Gilles Louppe, we begin the proof;

$$T(n) \leq cn\log n + c + T(n-1) = \sum_{x=1}^{n} c + xc\log x$$

$$\leq cn + c\log n \sum_{x=1}^{n} x = cn + c\log n \frac{n(n+1)}{2}$$

$$= O(n^2 \log n)$$

This worse case can be generalized to Random Forests, where k is the number of randomized trees, and m is the number of variables needed at each node;

$$= O(kmn^2 \log n)$$

4.2 kNearest Neighbors

While Random Forests took first place in terms of mean accuracy overall by a margin of about 4 to 5 percent, Nearest Neighbors proved also to be impressive, as it was the second most accurate over all the data sets. One of the big worries with the implementation of Nearest Neighbors is its tendency to become over fit when choosing a small k value. Generally, small k values do not always mean over fitting, especially with binary data that is perfectly separated by some decision boundary. In the case of the presented data sets, all of them have multiple attributes with tight overlapping data features. Case in point, the parallel distribution graphs of all of the data sets clearly show intersecting features. Because of this, the testing and training sets showed signs of astute similarity. An interesting point of contention is what metric classifier is best in terms of accuracy and speed.Four metrics were used, but only two constantly were utilized by the Grid Search for best performers; Euclidean and Manhattan, more formally in ML as the L_2 norm (Euclidean) and L_1 norm (Manhattan). While there is general debate as to which metric is best, there are important factors that contribute to choosing the most accurate. According to *On the Surprising Behavior of Distance Metrics in High Dimensional Space* written by Aggarwall et al, the use of L_1 or L_2 depends solely on the dimensionality of the data sets, i.e the number of attributes. This leads into the theory of grid heuristics, in which a heuristic function estimates the minimal cost from any vertex m to vertex n, such that $A* = f(n) = g(n) + h(n)$. Because of this constraints, the Manhattan distance proves to be the best scenario when dealing with the lowest cost in high dimensional data, as such deeming the Manhattan distance as being admissible. The intuition for this is there. For example, if we were to initialize an alpha term α to signify the minimum cost, then the goal in any path-finding algorithm is to try to minimize α. In an L_1 distance measurement (Manhattan), when a step is executed, the $g(n)$ parameter is increased by α, and the $h(n)$ is decreased by α. When these two values are summed, they should not change signifying that the loss function (generally understood to be the cost function) and heuristic stay the same. However, in high dimensionality, when using euclidean distance, the loss function $g(n)$ will not be the same as the heuristic function $h(n)$. This means that while the L_2 norm (Euclidean) might be able to find shorter paths, its time complexity is much larger (*Stanford*). While the theory is there, the data sets that were chosen leave this to be rather inconclusive. All of the data sets have 10 plus attributes, but this is no where near enough to suffer the effects of high dimensionality. More over, the ratio of L_1 to L_2 distance measurements throughout all of the trials was 4 to 6 drawing the conclusion of nearly a coin flip. If any of these data sets had thousands of attributes with hundreds of thousands of data points, it would be safe to conclude that using L_1 (Manhattan) as a distance metric would be the best possible choice preserving a good balance of efficiency and accuracy.

4.2.1 Complexity of kNearest Neighbors

When computing the time complexity of kNN, there are two subsections to address, using Brute force, or some data structure like kdTree. In terms of brute search, the complexity is generally understood as being;

$$O(nm)$$

Where n is the number of data points, and m is the number of dimensions (i.e attributes of each data point). Some researchers have argued that kNN actually has a general complexity of $O(nm+k)$, where k is the nearest neighbor but by and large, $O(nm+k)$ can be reduced to $O(nm)$. kNNs runtime can be further reduced by taking advantage of abstract data types, such as a priority queue and kdTrees. The following pseudocode does exactly that;

```
Result: Worst case $O(nlogk)$
initialize priority queue PQ;
initialize bestDistance at $\infty$ ;
initialize bestNode and KDTree Root ;
push root into PQ ;
while Priority Queue is not empty do
    set (node, bound) = PQ.pop();
    if if bound $\geq$ bestDistance then
        return bestNode;
    end
    initialize distance = dist(Node, node.bestNode)
    if if bound $<$ bestDistance then
        bestNode = Node
    end
    if if Node then
        PQ.push(node.left, Node[node.val] -
          node.thresh);
        PQ.push(node.right, 0)
    else
        PQ.push(node.left,0);
        PQ.push(node,right, node.thresh -
          Node[node.val])
    end
end
return bestNode;
```

Algorithm 1: kNN using KDtree priority queue

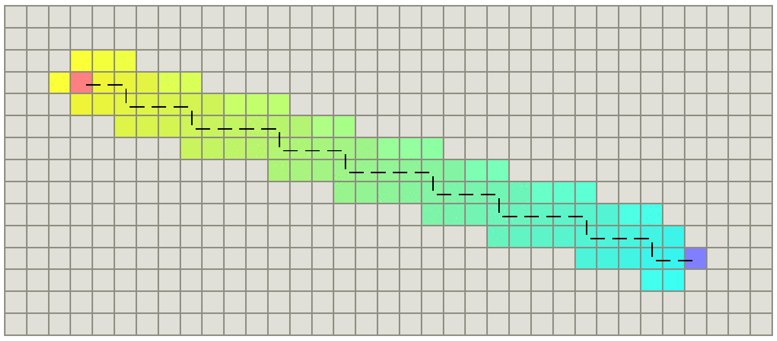
Fig. 9. Manhattan distance representation on a 2D Grid (Stanford)

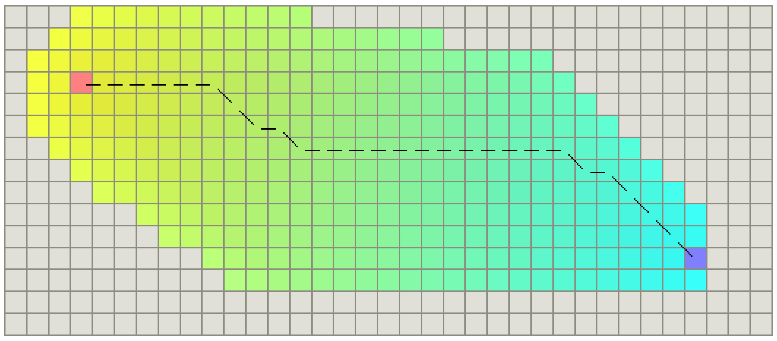
Fig. 10. Euclidean distance representation on a 2D Grid (Stanford)

4.3 Multilayered Perceptron

The next algorithm that needs to be analyzed is Multilayered Perceptrons. MLPs are what every aspiring machine learning engineer wants to perfect as they are a gateway into neural networks, deep learning and everything in between. Generally over the data sets, MLP performed the worst, which was a surprising result. In fact, the algorithm had one of the lowest accuracy percentages amongst all error metrics, over all data sets except Firewall and Dry Beans, where it performed exceptionally well. The discrepancies between each data set, each error metric, and accuracy output of MLP left much to be investigated. The alpha values, solvers, and activation parameters were pretty centralized to a few main parameters. Generally the Grid Search opted for logistic or tanh activation functions, with either stochastic gradient descent or a quasi-Newton optimizer. However, what encountered much variation was the sizes of the hidden layers. When the data overlapped heavily like in the League of Legends set, the hidden activation layers were usually set to $(5,) \vee (20,)$, depending on the solver, while with less sequestered data sets, like the Dry Beans set, the hidden activation layers were instead set to $(10,)$ (generally). The MLP had the worst accuracy score when classifying the LoL data set, often times reaching thresholds of barely above 60 percent with hidden layers 20. One possible explanation to this horrible accuracy rate is the low amount of Epochs. Generally, each MLP was capped at around 250 to 350 epochs (On the training set), which was significant enough to classify with impressive accuracy for other data sets. Because of the possible lack of convergence, the MLP classifier might have instead opted to counteract with high amounts of hidden layers for a more intensive computation. Had Epochs been set to something in the thousands, it could be possible that the classification rate would be much higher, but this is speculation. With respect to hidden layers, a question often times is brought up, as to what the "perfect" amount of layers equates to. A circular question, as it cannot be solved with a blanket suggestion. In *Deep Learning*, a MIT Press book written by Goodfellow et al, they discuss this problem of over-complication. Like many philosophers, they opt to invoke Occam's razor (principle of parsimony), when determining the best refinements for an algorithm, as to react with discretion to not over-fit or under fit the hyper-parameter. In the case of MLP, it is without a doubt that it suffered from severe under fitting in the case of the LoL data set.

4.3.1 Complexity of MLP

Generally speaking, the Multilayered Perceptron classifier functions through forward propagation (Inference). Under ideal conditions with an input vector $X \in R^n$, any input from the vector is treated as an activation matrix. Let's assume that an arbitrary multilayered perceptron has i inputs, j hidden layers and k outputs. In any Neural Network, any calculation occurs in the hidden and output layers. With this in mind, each hidden layer neuron has the following computation;

$$output_k = \sigma_1 \left(\sum_{s}^{i} w_{js} x_s \right)$$

With i inputs and j hidden neurons, the multiplications are ij for each layer (on average). The output layer has the same calculation with a possible different activation function (although this isn't always the case);

$$output_k = \sigma_2 \left(\sum_{s}^{i} w_{js} x_s \right)$$

With the output layer, the multiplications are jk where j is the number of hidden neurons and k is the number of output neurons. This can be generalized to a Multilayered perceptron with n inputs, M hidden layers, which each layer containing m_i neurons, and k outputs, written in Big-Θ (Tight bound) notation as;

$$\Theta \left(nm_1 + m_M k + \sum_{i=1}^{M-1} m_i m_{i+1} \right)$$

Thus, the time complexity of multilayered perceptron is directly correlated with the number of hidden layers and the number of hidden neurons in each layer.

4.4 Support Vector Machines

Support Vector Machines are extremely good at classification problems due to their utilization of margins and disregard for noise outside of said margins. SVM had good accuracy regardless of what error metric was utilized, sitting at around 85 ± 1. When running grid search on the data sets, one of the more common SVM kernels that was utilized was RBF. This could be because of the ease of access when it comes to hyper-parameter tuning, and is generally used as the default kernel. However, one of the biggest pit falls is the time complexity of the algorithm, especially when running Grid Search on four different kernels on a various collection of C and γ values. That being said, the run time of SVM sparked interest into optimization.

4.4.1 Complexity of Support Vector Machines

The training step of SVM falls under the quadratic programming (optimization) problem. In which the goal is to either maximize or minimize a function under constraints. In a Support Vector Machine, the goal is to maximize the margin in order to minimize the magnitude. Thus by mathematical definition, the argmin function of SVM is;

$$argmin(w,b,\lambda) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{k}(\lambda_i y_i(W^T X_i + b - 1)$$

Where λ is known as the Lagrange multiplier. This introduction of the Lagrange multiplier lets the SVM optimization problem be rewritten in dual form (the primal problem/ dual problem).

$$\max_{\lambda} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(x_i, x_j)$$

Under the constraint where the Lagrange multiplier must be less than or equal to the hyper-parameter

$$0 \le \lambda_i \le C, \quad \text{for } i = 1,2,\ldots,n, \sum_{i=1}^{n} y_i \lambda_i = 0$$

This is done in order to obtain the maximum lower bound optimum value by maximizing the dual function instead of solving the primal problem (minimizing). With an understanding of how the SVM optimization problem is formulated, the next step is to understand what some Machine Learning libraries like ScikitLean and LIBSVM use to optimize complexity. This optimization algorithm is known as sequential minimal optimization. Introduced in 1998, its purpose serves to split the SVM optimization equation into small sub problems;

```
Result: Worst case $O(n^3)$
initialization;
while Not converged do
    Find a Lagrange multiplier, $\lambda_1$ that violate first
      derivative tests (known as
      Karush–Kuhn–Tucker (KKT) conditions) ;
    Find a second Lagrange multiplier, $\lambda_2$ and
      optimize $(\lambda_1, \lambda_2)$ ;
end
```

Algorithm 2: Sequential minimal optimization

With all of the insane math and optimization that is involved, Support Vector Machines utilizing the SMO algorithm have a worst case run time of $O(n^3)$. This run time makes it painful to use the algorithm for any massive data sets with many features. On the data sets presented, with on average $10 \le n \le 20$ attributes and only 5000 data points, training the SVM took much longer than any other algorithm, wow.

4.5 Logisitic Regression

Logistic Regression was the final algorithm that was analyzed. On average, its performance over all data sets using all error metrics was good. It performed better than MLP and SVM (by a slight margin), and was outperformed by RF and KNN. LR is generally understood as being a powerhouse algorithm when it comes to predicting probabilities on binary classification problems. It performed exceptionally well in the LoL data set, just behind RF. LR also can easily be adapted to classification problems beyond binary, like that of the Bean data set. Out of all of the algorithms, LR was the least prone to over fitting. This lack of over fitting could of been due to Grid Search finding the best hyper parameter, and more importantly, choosing between L_1 or L_2 regularization to minimize the significance of outlier to subsequently reduce the need to "over-fit" the data per se. In general, LR was the only algorithm in which the change in error metric did not significantly change the required hyper-parameters. This can be seen in figure 2, in which LR had a LOG LOSS accuracy of .86, ROCAUC accuracy of .85 F1 accuracy of .85 and MEAN of .85. A very consistent algorithm, however, where it was great in consistency, it lacked very high accuracy. This could be because of the unbalanced data sets that were not linearly separable. LR tends to work very well with binary classification data sets like mail spam or cancer classification.

4.5.1 Complexity of Logistic Regression

Without taking into account any specific regularization, the time complexity of logistic regression is pretty straight forward. Generally its worse case is;

$$O(nmk)$$

Where n is the number of attributes per data point, m is the number of dependent classes y vector, and k is the number of data points. It's run-time is very similar to that of linear regression, the only main difference being the addition of an activation function denoted as;

$$\sigma = \frac{1}{1+e^{-x}}$$

5 Analysis of Data Sets

One of the biggest draw backs of Machine Learning and its derivatives in the 80s and 90s was the lack of usable data. Data collection and utilization is the backbone of machine learning, as it needs this information to make decisions, and with more information comes more accuracy. That being said, there comes a point where the data is not useful. This could be due to various things, it could be that the data was pre-processed incorrectly, gathered incorrectly, noise in dependent variables that lead to inconclusive data, the list goes on. In the case of the four "active" data sets that were talked about in the paper, The League of Legends, Dry Beans, Gamma Particles, and Firewall all had many good features in common. Each set was catered for classification and was balanced. pre-processing was very basic, and each set performed as expected during the Grid Search with Stratified KFold validation on 5000 data points for a total of 5 trials for each algorithm. However, the Poker data set proved to be troublesome, and it highlighted the importance of using balanced data sets with validation. When Grid Search was ran, a ValueError would continuously show up over and over again. This error had its roots with ROCAUC, one of the error metrics that was used. This error was puzzling as it only occurred with the Poker Data set. After much investigation, the problem was caused by a heavily imbalanced data set. In table 1, we can see that the Poker data set has a near 97 percent negative class, with only about 6 percent positive class. This discrepancy is only after the y values were encoded from a multi class classification to binary. For example, prior to this encoding, getting a CLASS 9 in the y vector would equate to getting a Royal Flush. To be dealt a royal flush, the probability is $4/2,598,960 = 1/649,740 = 0.00015\%$. Or more mathematically proven using Combinatorics;

$$= \frac{5!}{(13 \cdot 51 \cdot 50 \cdot 49 \cdot 48)}$$

$$= \frac{1}{13} \cdot \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{51 \cdot 50 \cdot 49 \cdot 48}$$

$$= \frac{52}{13} \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{51 \cdot 50 \cdot 49 \cdot 48}$$

$$= \frac{4}{C(52,5)} = \frac{C(4,1)}{C(52,5)} = 0.000001539.$$

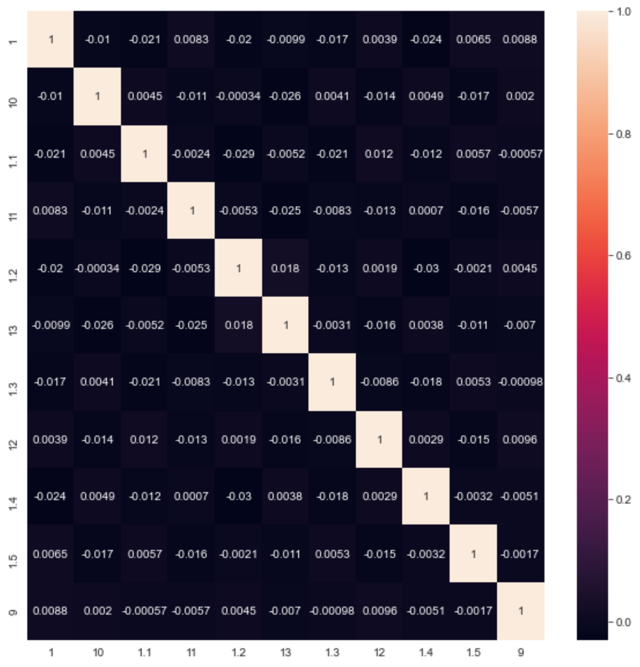

Fig. 11. Correlation matrix, notice the very low negative/positive correlation factors of each attribute.

When actually looking into the probabilities of the data set, it becomes much more clear why the ValueError was showing up. The data set is first sampled with 5000 random data sets. Then, Grid Search does a Stratified KFold 5 times. It is then the discrepancy between the KFolded data points and the rest of the data set that causes the ValueError. In other words, when KFold is ran, the sub data sets do not have all of the y values. For example, lets say sub data set 1 is split into 1000 data points. In this set, it contains all y values except for $8,9$. sub data set 2 is split into 1000 data points, but in this set, it contains all y values except for 9. When running ROCAUC, it will throw an error as the $y_{T}rue$ will not match up with the $y_{T}est$ because the y vectors size do not match. This error was fixed by using a sample size of 25000 data points, and using a Stratified 2Fold Grid Search. Because these restrictions did not apply to all data sets, it was better to investigate this data set singularly to keep the validity of the analysis strong.

6 References

[1] “A*'s Use of the Heuristic.” Heuristics,theory.stanford.edu/ amitp/GameProgramming/Heuristics

[2] Adamczyk, Jakub. “k Nearest Neighbors Computational Complexity.” Medium, Towards Data Science, 14 Sept. 2020, towardsdatascience.com/k-nearest-neighbors-computational-complexity-502d2c440d5.

[3] Aggarwal, Charu C, et al. “On the Surprising Behavior of Distance Metrics in High Dimensional Space.” Https://Bib.dbvis.de/UploadedFiles/155.Pdf, IBM T.J Watson Research Center.

[4] “What Is the Time Complexity of the Forward Pass Algorithm of a Feedforward Neural Network?” Artificial Intelligence Stack Exchange, 1 June 1968, ai.stackexchange.com/questions/13612/what-is-the-time-complexity-of-the-forward-pass-algorithm-of-a-feedforward-neura.

[5] Banerjee, Alekhyo. “Computational Complexity of SVM.” Medium, Medium, 31 Aug. 2020, alekhyo.medium.com/computational-complexity-of-svm-4d3cacf2f952.

[6] Banerjee, Writuparna. “Train/Test Complexity and Space Complexity of Logistic Regression.” Medium, Level Up Coding, 27 Aug. 2020, levelup.gitconnected.com/train-test-complexity-and-space-complexity-of-logistic-regression-2cb3de762054: :text=So

[7] Bulso, Nocola, et al. On the Complexity of Logistic Regression Models. arxiv.org/pdf/1903.00386.pdf.

[8] Chiramana, Sathvik. “SVM DUAL FORMULATION.” Medium, Medium, 1 Oct. 2019, medium.com/@sathvikchiramana/svm-dual-formulation-7535caa84f17.

[9] “Computational Complexity of Machine Learning Algorithms.” The Kernel Trip, 16 Apr. 2018, www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/.

[10] Daniel López and josliber“k-NN Computational Complexity.” Cross Validated, stats.stackexchange.com/questions/219655/k-nn-computational-complexity.

[11] “What Is the Time Complexity for Training a Neural Network Using Back-Propagation?” Artificial Intelligence Stack Exchange, ai.stackexchange.com/questions/5728/what-is-the-time-complexity-for-training-a-neural-network-using-back-propagation/5730.

[12] Fredenslund, Kasper. “Computational Complexity Of Neural Networks.” Kasperfred, kasperfred.com/series/computational-complexity/computational-complexity-of-neural-networks.

[13] Fredenslund, Kasper. “Just How Fast Is Your Algorithm?” Kasperfred, kasperfred.com/series/computational-complexity/just-how-fast-is-your-algorithmbig-o-matrix-multiplication.

[14] Goodfellow, Ian, et al. Deep Learning, MIT Press, 2016, www.deeplearningbook.org/.

[15] “Heuristics From Amit's Thoughts on Pathfinding.” Heuristics, theory.stanford.edu/ amitp/GameProgramming/Heuristics.html.

[16] KOWALCZYK, Alexandre, et al. “SVM - Understanding the Math: Duality and Lagrange Multipliers.” SVM Tutorial, 26 July 2020, www.svm-tutorial.com/2016/09/duality-lagrange-multipliers/.

[17] Kumar, Paritosh. “Time Complexity of ML Models.” Medium, Medium, 17 Dec. 2019, medium.com/@paritoshkumar5426/time-complexity-of-ml-models-4ec39fad2770.

[18] Kumawat, Dinesh. “Introduction to Logistic Regression - Sigmoid Function, Code Explanation.” Analytics Steps, www.analyticssteps.com/blogs/introduction-logistic-regression-sigmoid-function-code-explanation.

[19] Louppe, Gilles. Understanding Random Forests; From Theory to Practice. July 2014, arxiv.org/pdf/1407.7502.pdf.

[20] Muller, Emily. “Mathematical Underpinnings: SVMs + Optimisation.” Medium, Towards Data Science, 18 Sept. 2018, towardsdatascience.com/mathematical-underpinnings-svms-optimisation-6495776215c3.

[21] The Nearest Neighbor Algorithm. Oregon State University, web.engr.oregonstate.edu/ tgd/classes/534/slides/part3.pdf.

[22] “Sequential Minimal Optimization.” Wikipedia, Wikimedia Foundation, 21 June 2020, en.wikipedia.org/wiki/Sequentialminimaloptimization.

[23] “The Simplified SMO Algorithm.” CS229 Simplified SMO Algorithm 1, Stanford, cs229.stanford.edu/materials/smo.pdf.

[24] Taschka, Sebastian. STAT 479: Machine Learning Lecture Nodes. Department of Statistics University of Wisconsin–Madison, Fall 2018, sebastianraschka.com/pdf/lecture-notes/stat479fs18/02knnnotes.pdf.