CSC 466: Analysis of Machine Learning Methods for Anomaly Detection in AWS using the Numenta Anomaly Benchmark Dataset

Damon Lin
dlin42@calpoly.edu

Jerry Tang
jtang72@calpoly.edu

Ritvik Durgempudi
rdurgemp@calpoly.edu

Wesley Tam
wtam08@calpoly.edu

Abstract

This report covers the performance of four different machine learning models for anomaly detection using an already classified anomaly dataset from AWS. The end goal of this project is to identify the best machine learning models to use for anomaly detection. The four models: logistic regression, single-layer perceptron, multi-layer perceptron, and k-means clustering, were trained on the same dataset and their accuracy, recall, precision, and F1-scores were recorded and compared. The results of the report showed that linear classifiers such as logistic regression and single-layer perceptron performed poorly in classifying anomalies as they were not correctly learning from the data. On the other hand, models that supported non-linear data such as multi-layered perceptron performed significantly better than their linear-based counterparts. This implies that the best models to use to correctly classify anomalies are those that can handle data that is not linearly separable and as such are best suited for anomaly detection.

1. Introduction

To ensure that software applications are deployed reliably, anomalous behavior must be quickly identified and addressed. By leveraging state-of-the-art machine learning methods to identify patterns within software deployment, software developers and professionals can ensure that their applications are running correctly. In this paper, our team will demonstrate four separate machine learning models in order to identify methods particularly efficient at anomaly detection and to pinpoint the qualities that make them effective. Drawing on data from AWS, an industry leader in cloud services, we will train and test our machine-learning models on a time-series dataset comprised of timestamps, CPU Utilization, and labels indicating whether an anomaly occurred or not. From there, we will evaluate the performance of each model, and measure the method's ability to learn from the data and to identify anomalies using metrics such as Accuracy, F1-Score, Precision, and Recall. Afterward, we will examine the qualities of each method in order to explain the model's performance. By the end of this research paper, we will conclude which of our four methods perform the best, and the qualities that enable their success.

2. Related Work

For this research project, we decided to analyze data from AWS using the Numenta Anomaly Benchmark (NAB) dataset. In the past, there have been many analyses on this dataset, most notably the Numenta team themselves, which created their own machine learning model called the Hierarchical Temporal Memory (HTM) [4]. This algorithm takes inspiration from the neocortex and was designed to work well with time-series data. The HTM algorithm scored the highest out of all the algorithms tested by the Numenta team, even when prioritizing low FPs or FNs. This shows the importance of using the time-series aspect of the data in their analysis. We decided to ignore the time-series aspect in our analysis of the dataset because our previous attempts failed to properly utilize the time-series component of our dataset and we wanted to highlight the distinctions in different types of machine learning models. Another interesting research paper is Experimental Comparison of Online Anomaly Detection Algorithms [1], which, similar to our goal, wanted to "choose the optimal anomaly detection methods based on the characteristics the time-series displays". They tested much more advanced algorithms such as RNN and SARIMA, and in the end, concluded that the best algorithm for a given dataset depends heavily on the data itself.

3. Methods

In this paper, we explore four different machine learning methods: Logistic Regression, Perceptron, Multi-Layer Perceptron, and K-Means Clustering. For each of our models, we split the dataset into stratified training and testing sets. Since our dataset is unbalanced, as shown in Figure 1, we chose to stratify the training and testing splits in order to ensure that an equal proportion of anomalous and non anomalous samples are in both sets. In addition, we also stratified our training and validation folds during

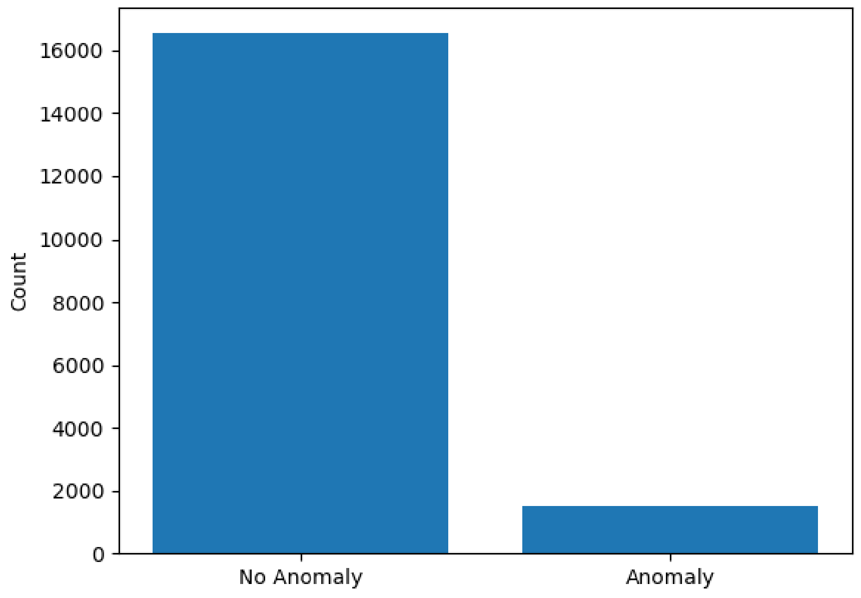


Figure 1. As shown above, our dataset consists of significantly less anomalous samples, indicating that our dataset is very unbalanced.

cross-validation to ensure they contain equal proportions as well. After we completed the training and testing for each model, we then evaluated our model based on the F1-score, Precision, Recall, and Accuracy, presenting the confusion matrices for each method to better visualize our results. Moreover, for Perceptron, Logistic Regression, and the Multi-Layer Perceptron, we trained our models using CPU Utilization as our only feature and ignored the time-series component of the dataset. Furthermore, in this paper, we relied on mainly the Sklearn library [5] for the implementations of the Perceptron, the Multi-Layer Perceptron, K-Means Clustering, Logistic Regression, and many of the utilities, such as Grid Search, that we used to train, test, and evaluate our models.

4. Overview

To investigate the performance of different machine learning models, the subsequent four sections will describe each method we are using, our experiments, our results, and then a discussion on whether the method is effective. Afterwards, we will reflect on some of the challenges and shortcomings we experienced, and present a final conclusion on our findings.

5. Logistic Regression

We decided to test a logistic regression model on the Numenta dataset because we wanted to know how a relatively simple algorithm scored in comparison to our other models; similar to a control group in a scientific study. As a review, logistic regression runs through the entire dataset and, based on an initial set of weights and biases, predicts a binary outcome given the input features. The model then calculates its error and updates its weights accordingly. Then it repeats this process for a given amount of loops until it can accurately predict outcomes. A logistic regression model is designed for a linearly separable dataset, so we expect our logistic regression model to perform poorly on this dataset. Furthermore, we will only be using the CPU Utilization feature when training our models to keep our results comparable to the other algorithms tested.

5.1. Experiments

The dataset used to train the logistic regression model is the aforementioned NAB dataset. We created our own logistic regression algorithm in addition to Sklearn's in-built algorithm to easily control the learning rate and epoch count as well as easily graph the errors per epoch. In terms of data preprocessing, we stratified our dataset on the target field so that we would have an equal amount of target classes in our train and test splits. We also ignored the time stamp field as mentioned earlier and normalized the CPU Utilization. We used Sklearn's in-built logistic algorithm for 10-fold cross-validation and our own algorithm for final accuracy testing. For both algorithms, we used an epoch count of 1000. For our own algorithm, the initial weights and biases are set to 0 and the learning rate was set to 0.01 after some light testing.

5.2. Results

Both logistic regression models scored equally as poorly in that both failed to predict anomalies in the dataset, as expected since the data was nonlinear. The accuracies of both algorithms were deceptively high, predicting around 91% of the targets correctly. However, upon further analysis, it was discovered that the only reason why the algorithm was able to achieve such a high accuracy score was because the dataset itself was heavily unbalanced. Since the dataset consisted of almost 90% non-anomalous samples, an algorithm only predicting non-anomalies would inherently be correct 90% of the time. Looking at Figure 1, the confusion matrix of the logistic regression model, we can see that the model indeed did not ever predict an anomalous sample.

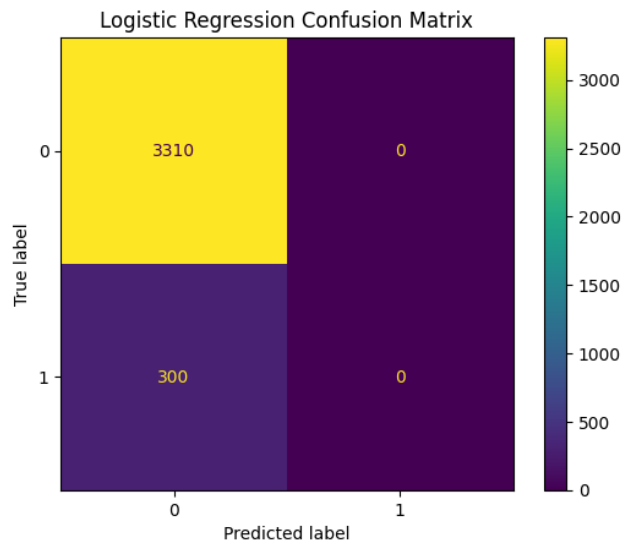


Figure 2. Confusion matrix of the logistic regression algorithm

This gives us an accuracy score of 91.69% but a recall, precision, and F1-score of 0%, making logistic regression a terrible algorithm to use on the anomaly detection dataset. Looking at the loss graph in Figure 2 may show a reason why our logistic regression algorithms underperformed.

As we can see, the logistic regression model seems to have predicted all anomalies in the beginning, then updated its weights and predicted all non-anomalies thereafter. After the first update of the weights, the model seems to have reached a non-optimal local minima and remained there for the remainder of the iterations. Several different learning rates were tried, but none fixed the suboptimal minima. Overall, using logistic regression is not advised when analyzing nonlinear datasets.

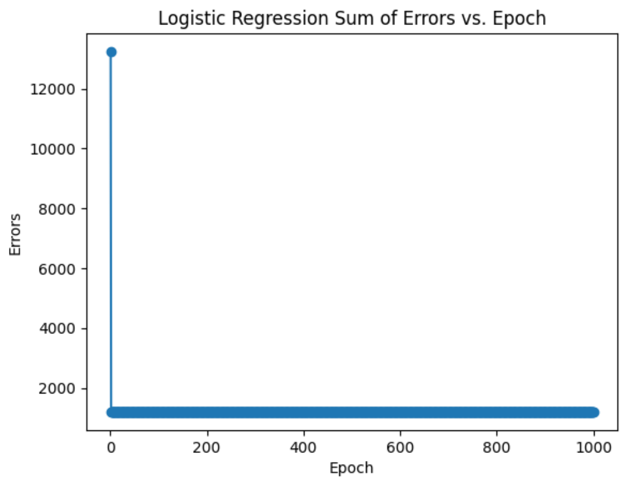


Figure 3. Plot of the number of misclassifications with respect to epoch in the logistic regression algorithm

6. Single-Layer Perceptron

Another proposed method that we wanted to test was a single-layer perceptron. Given the dataset's unbalanced nature and inability to be linearly separated, we were curious to see whether the classifier would perform similarly to Logistic Regression since they are both linear classifiers. The perceptron classifier is modeled after a neuron found in the brain. As Figure 4 visualizes, the model takes in features, weights, and biases which are used to calculate a value through summation. This value is then put through an activation function. The activation function is essentially a threshold function that decides if the "neuron" fires and determines the model's prediction. Perceptron is a linear classification model, as such we still wanted to test its performance on a non-linearly separable dataset to compare it to other nonlinear models.

6.1. Experiments

For our specific dataset, we first trained and tested our in-built Sklearn model using cross-validation. We trained for 10 folds to see how well that model performed. We then used the dataset to test the model separately for the 10-fold cross-validation to see how well it performed there. Most of the data preprocessing for the perceptron model was the same that we used on the other models. We stratified the data to account for its unbalanced nature. We also normalized the values because it didn't affect the performance of the model to do so since we were only using one feature. After running the model without cross-validation we then calculated the model's confusion matrix. The setup for Perceptron was fairly basic and didn't require anything extra than what was done for the other models. One thing we chose not to do for Perceptron was hyperparameter tuning with grid search that we used for the multi-layer perceptron.

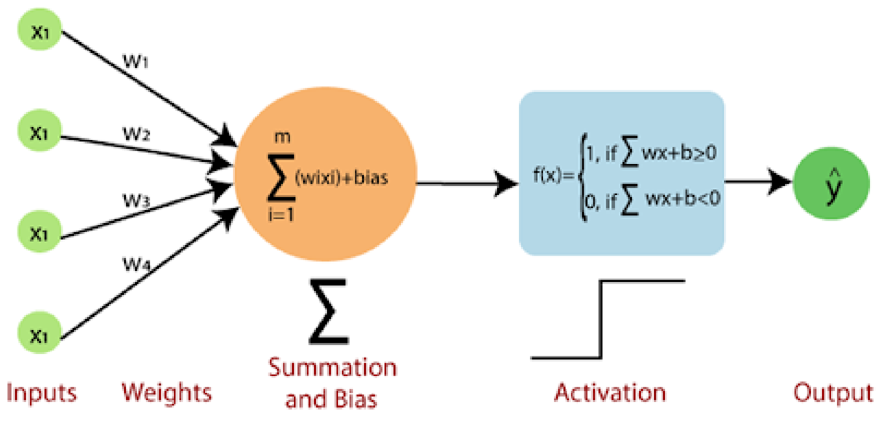


Figure 4. Diagram that depicts the structure of a single-layer perceptron model [3]

We deemed that it was not as necessary as it did not have as many hyperparameters as its multilayered counterpart.

6.2. Results

The results of utilizing the perceptron model were somewhat similar to the results of the logistic regression model. The model boasted a very high accuracy of 93% at first glance. However, looking deeper into the model reveals that the model never makes any false positives and only classifies a small sample of anomalies as anomalies. So, although it did slightly better than the logistic regression model which only predicted samples as nonanomalous, the perceptron model still did not perform very well. The two other metrics that display how poorly the model did are the F1 score and the recall. The model had an F1 score of 29.545% and a recall of 17.33%. These scores show the model is not learning much from the data. The model has a high recall because it has many false negative classifications. In other words, the model does not predict anomalies as anomalies and marks them as nonanomalous, much like the logistic regression model. This is depicted in Figure 5, which shows the confusion matrix of the model. As seen in this figure, the model predominantly guesses nonanomalous for samples which has led to a very low recall. With this new information factored in, it makes sense why the F1 score is so low. The F1 score is a much better performance metric to judge this model upon as it takes into account both the precision and the recall. Even though the precision for the model was 100%, that was because the model barely predicted samples as anomalous which led to having no false positives. The accuracy and precision are false metrics that don't show the whole picture of the model's performance.

So the question remains as to why the model performed

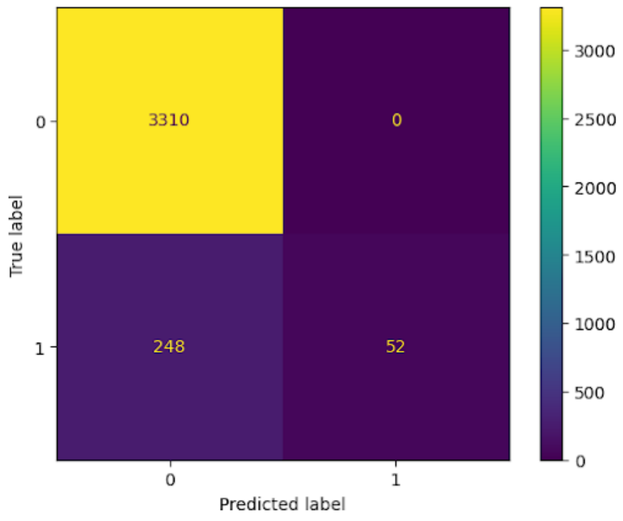


Figure 5. Confusion matrix produced by Single-layer Perceptron

so poorly. As stated before, Perceptron is a linear-based model. Our AWS anomaly dataset is very much not linearly separable. This is most likely the primary reason that the model performed so poorly. This further strengthens our findings that linear-based models are not the best suited for anomaly detection, at least in the case of this dataset. Compared to some other models that we will go into later, we deemed that single-layer perceptron is not a good model to use as its performance metrics were much too low to be acceptable.

7. Multi-Layer Perceptron

The Multi-layer Perceptron is a machine learning method similar to Logistic Regression and the Perceptron but consists of additional layers of neurons, also known as

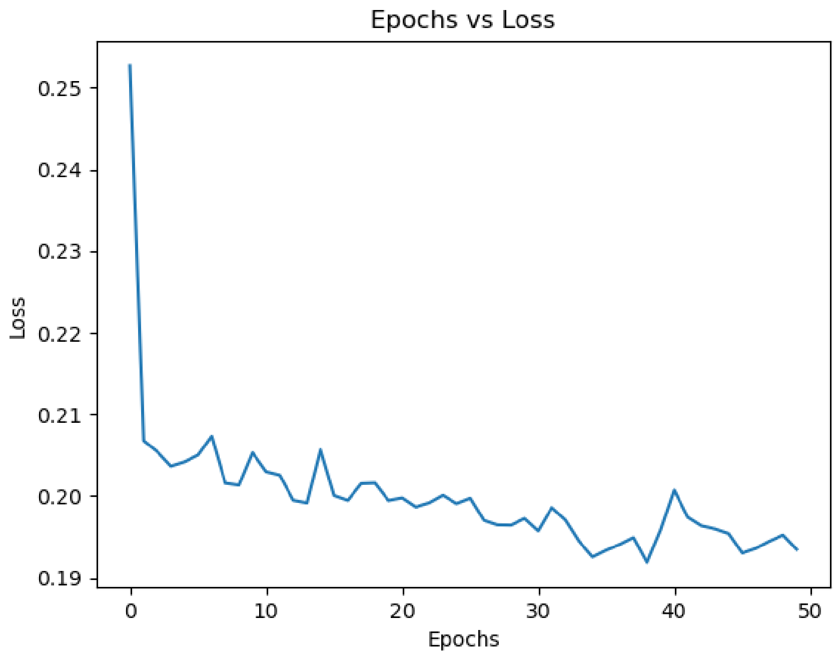


Figure 6. The above figure illustrates the model's loss per epoch. Since the loss is decreasing as the number of epochs increase, we believe that the model is training correctly.

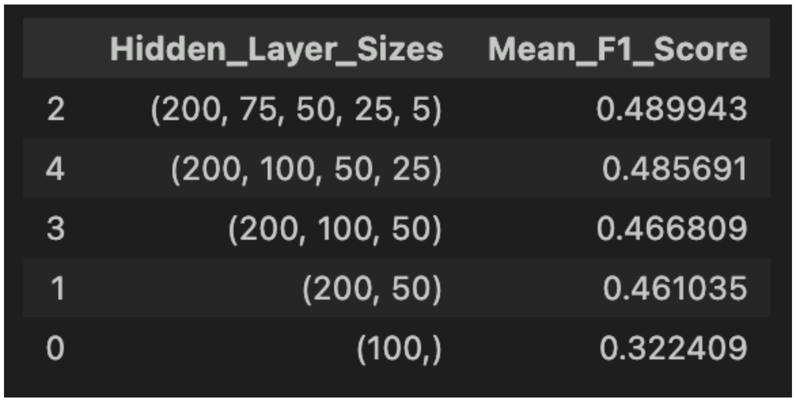
the hidden layers, between the input layer and the output layer. Each hidden layer consists of a linear transformation of the previous layer's output and the learned parameters at that layer. In addition, after the linear transformation of the input and weights, each layer also applies an activation function, such as sigmoid or ReLU, before passing its output into the next layer. By processing the input through multiple layers, the Multi-Layer Perceptron is able to be an effective non-linear classifier [2], unlike Logistic Regression and the Perceptron.

7.1. Experiments

For our Multi-Layer Perceptron, we chose ReLU for our activation function since it is nonlinear, and we chose Adam for our optimizer. After selecting our base model, we split our data into a training and testing split, stratifying the samples to ensure that both the train and test set have equal proportions of anomalous and non-anomalous samples. Then, we used Grid Search to select the set of hyperparameters, such as the learning rate, the number of hidden layers, and the size of each hidden layer, that best models the data. Since the dataset is largely unbalanced, we selected the model based on the highest F1 score. Then, we evaluated our model with cross-validation using the hyperparameters we previously selected. Again, to ensure that each fold contains the same proportion of samples containing anomalies, we stratified the training fold and the validation fold inside the cross-validation. In addition, we also graphed the loss curve with respect to epochs to confirm that our model is converging to a minimum loss. After we achieved satisfactory accuracy during cross-validation, we retrained the model using the entire training set and then completed the final evaluation of our model on the test set in order to confirm that our model generalized effectively from the training data. Then, we calculated the number of true positives, true negatives, false positives, and false negatives that the model predicted on the test set to calculate the Precision, Recall, and F1-Score. We also graphed the confusion matrix in order to visualize the model's performance.

7.2. Results

Using Grid Search, we measured the best layer sizes to be 200, 75, 50, 25, and 5 as illustrated in Figure 7. In addition, we also observed the most effective learning rate to be 0.005. After we selected the hyperparameters, we observed 93.2% accuracy during cross-validation, and 93.7% accuracy on the test set. Since our model performs similarly during cross-validation and when evaluated on the test set, we believe that our model is able to generalize effectively to unknown data. In addition to accuracy, we measured the recall to be 30.33%, the precision to be 84.26%, and the F1-Score to be 44.61%. From the loss curve in Figure 6, we also observe that the loss is generally decreasing as the number of epochs increases, confirming that our model can



	Hidden_Layer_Sizes	Mean_F1_Score
2	(200, 75, 50, 25, 5)	0.489943
4	(200, 100, 50, 25)	0.485691
3	(200, 100, 50)	0.466809
1	(200, 50)	0.461035
0	(100,)	0.322409

Figure 7. The above figure illustrates the performance differences between different hidden layer sizes and different quantities of hidden layers. Our results suggest that four hidden layers consisting of 200, 75, 50, 25, and 5 nodes respectively is the most effective.

learn from the data and optimize correctly. Moreover, the confusion matrix in Figure 8 also illustrates that our model is able to easily identify true negatives, events that do not contain an anomaly, but struggles to identify true positives, events that do contain an anomaly, as indicated by the poor recall and the large number of false negatives, similarly to the Single-layer Perceptron and Logistic Regression.

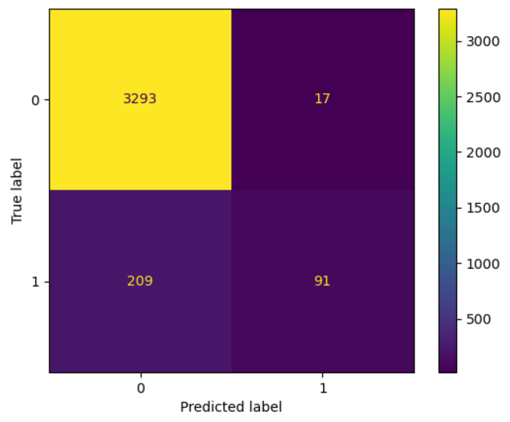


Figure 8. The figure above demonstrates the confusion matrix for the Multi-layer Perception, drawing on predictions made on the test set. As indicated, the model appears to predict mostly negative (non-anomalous events).

7.3. Discussion

Although the Multi-Layer Perceptron has low recall relative to its high precision, the Multi-Layer Perceptron demonstrates that it is able to identify anomalies at a much higher rate than Logistic Regression and the Single-Layer Perceptron, as illustrated by its much higher F1-Score. We believe the difference in performance can be attributed to the Multi-Layer Perceptron's ability to classify and identify patterns within data that is not linearly separable, unlike Logistic Regression and the Single-Layer Perceptron, both of which rely on the data to be linearly separable for it to be effective. This suggests that the Multi-Layer Perceptron's ability to classify data that is not linearly separable is crucial for this dataset, and by extension, the problem of Anomaly Detection.

Although we believe our model is effective, we believe our model selection process can be improved by using a heuristic or approximation algorithm to estimate potential candidates for our hyperparameters. Since we did not use an algorithm to approximate the learning rate, the hidden layer sizes, and the number of hidden layers, the hyperparameters were chosen arbitrarily. In the future, we believe we can produce a potentially better model by attempting to estimate and deduce more effective hyperparameters.

8. K-Means Clustering

K-means clustering is an unsupervised learning algorithm that categorizes a set of data points D into a predetermined K number of sets, also widely known as clusters, each focused around a centroid C with the priority of minimizing the sum of square distance for each cluster. The algorithm requires an initial set of data points to act as the centroids, which can either be data points in the overall set given or randomly generated points. The algorithm then assigns every data point to a cluster based on which centroid the data point is closest to. This distance can be calculated in different manners, but for the sake of simplicity, we will be using and referring to the Euclidean distance between two points. Once every point has been assigned a cluster, the centroids for each cluster are recalculated by taking the mean of every point within the cluster. Once the new centroids for each cluster have been calculated, the algorithm then attempts to re-assign every data point to a cluster. Some data points may be assigned to a new cluster if they are closer to a different centroid after the recalculation, while some may remain in the same cluster. From here, if any data point has changed clusters, the algorithm will then re-calculate the centroids once again and repeat the entire process. However, if every data point remains in the same cluster it was assigned in the previous iteration, the algorithm then understands that it can no longer make progress and has converged.

8.1. Experiments

In our search for finding an effective method to predict anomalies from the given features and data, we decided to employ k-means clustering to determine whether or not clustering patterns could be detected and used to identify future anomalies. Given that we had a total of three features in the dataset, we decided to take a quick look at the visualization of each combination of features before deciding how to proceed with training our model, which are displayed in figures 9, 10, and 11.

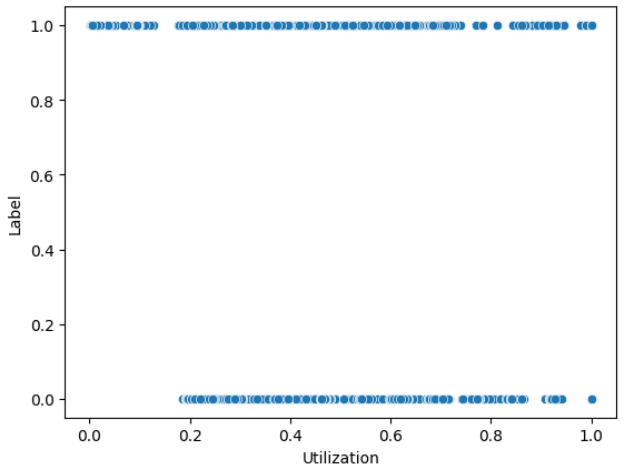


Figure 9. Visualization of Data by Utilization and Label

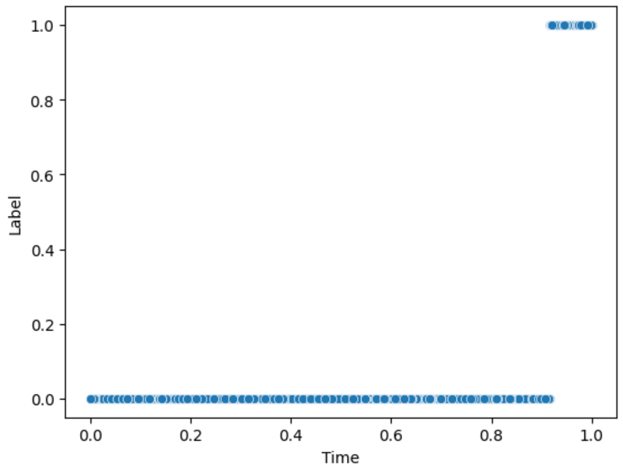


Figure 10. Visualization of Data by Time and Label

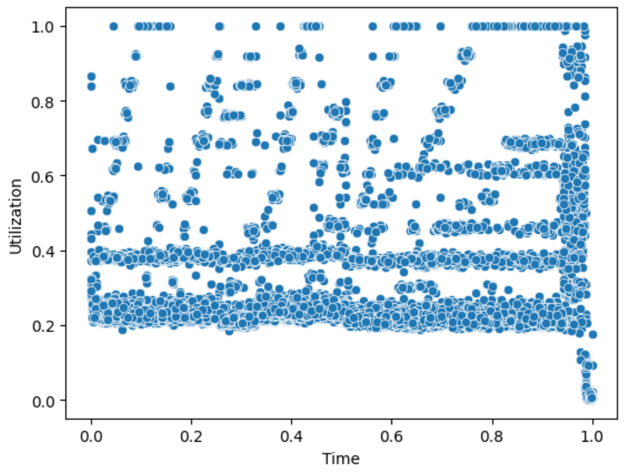


Figure 11. Visualization of Data by Time and Utilization

Both Figure 9 and 10 do not indicate that k-means will produce very effective results, as they use the binary "Label" feature used for reporting the data points as anomalous. "Label" being confined to two outcomes means that it doesn't suit k-means, due to how k-means relies on minimizing Euclidean distance. "Label" is either a 1 or 0, thus resulting in clusters having a heavy skew towards whichever outcome is more prevalent within the dataset. Furthermore, k-means tends to work better on data that can potentially form spherical shapes, which is not present in either combination with "Label" because of how its binary nature leads the data to be confined to two lines. As a side note, we acknowledge that there are other issues with using "Label" as a feature for training our model which will be addressed in the analysis after the results. Figure 11, on the other hand, plots the timestamp of each data point's measurement to the recorded percentage of the CPU's utilization. Neither of these features are binary, thus resulting in a much more spread-out visualization of data. Furthermore, although the data seems to be concentrated at certain levels of utilization, it still seems to vary quite a bit, much more so than the binary nature of the "Label" feature. Therefore, due to how the data doesn't utilize a binary feature and has somewhat of a spread, we moved forward with the combination of the timestamp and utilization features to train our model with.

Originally, a mistake was made with calculating the silhouette score to end up with a K of 9. Experiments done with a K = 9 ended up creating a model that performed very well and provided strong statistics as a result. However, upon further examination, the method used to choose a K of 9 was incorrect, and upon revision, it was determined through the elbow method that a K of 3 would be better. This ended up making the statistics of the model much worse, but the explanation for why this supposedly better K of 3 did worse statistically will be covered along with the other shortcomings of k-means clustering later on. For now, know that the following results are from the model trained with K = 3.

The difficulty of using K-means for this binary classification problem was how to determine the accuracy of the model, as the model predicts which cluster the data point belongs to and not if the point is an anomaly. A workaround for this was to calculate the percentage of anomalies in each cluster and to then classify them as anomalous clusters if the percentage of anomalies exceeded a certain confidence level. In the initial run where K = 9, the confidence level of 66% was used, as it seemed like a cluster where over 2/3rds of the data points were anomalies was a good indication of an anomalous cluster. Upon changing to K = 3, a confidence level of 66% was not a very good indicator, as none of the clusters had a very high concentration of anomalies in them. The confidence level was then shifted to 20%, as that was necessary for the model to at least classify points as anomalous rather than predict every point to be non-anomalous.

8.2. Results

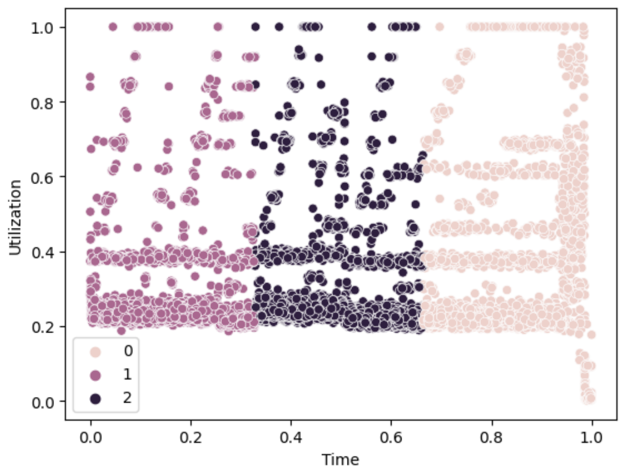


Figure 12. Visualization of Data by Time and Utilization, post K-Means Clustering at K = 3

Figure 12 is the visualization for the results of a K-mean clustering model with K = 3. Figure 13 is the confusion matrix for the predictions made by it. From our confusion matrix, we calculated the precision to be 25.84%, the recall to be 100%, the F1 score to be 41.07%, and the accuracy to be 74.57%. As shown through the statistics, the model can accurately identify non-anomalous points, as indicated by the high recall. However, it fails to identify many non-anomalous points as non-anomalies, mistakenly classifying them as anomalies instead, thus resulting in an extremely poor precision score. This inability to mark regular points as anomalies drags down the accuracy and the F1 score, especially for the latter. The model's high false positive rate

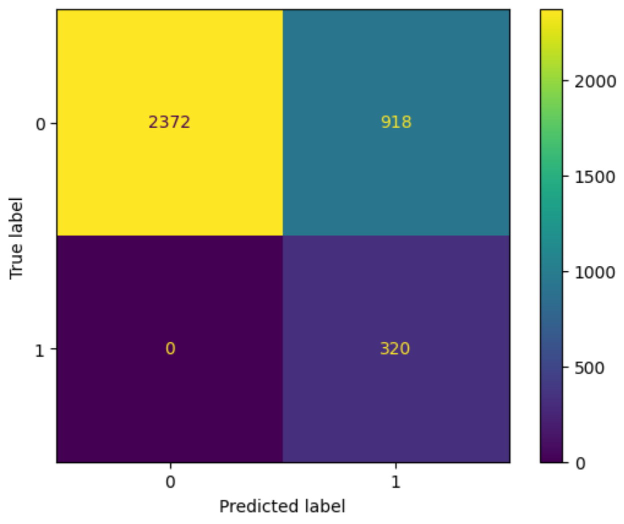


Figure 13. Confusion Matrix of Predictions from K-Means Clustering Model

can likely be attributed to the method in which it deems points as anomalous, as it classifies anything that belongs in an anomalous cluster to be an anomaly. This, paired with how there are only 3 clusters used for this model, means that the anomalous cluster(s) the model uses to predict anomalies must contain a considerable amount of non-anomalous points as well, thus meaning its cluster has not been refined enough to where many regular points are being caught in it. This is further proved by taking a look at the model's confidence level for anomaly percentage per cluster, which had to be modified to 20% in order to properly classify anomalies rather than just predict everything as a non-anomalous point. This means that as long as the cluster was made up of 20% anomalies, everything predicted to belong to the cluster would be considered an anomaly, which explains the extremely poor precision of the model.

8.3. Discussion

As a whole, k-means clustering was not suited to analyze this dataset in the slightest. The combination of low features that were inappropriate for k-means meant that it was not going to perform well in the first place, and yet it did very well at an irregular K level. The following will cover what factors exactly made k-means a poor choice, as well as why it performed so well despite that at a high K level.

First and foremost, time is not a feature that works well with Euclidean k-means clustering. A soft assumption in k-means clustering is that the data points are independent and identically distributed. However, the data we used had time as a feature, and k-means doesn't work well with time sequences due to how there are temporal dependencies between data points. That is to say, not only can some data points be related to others based on when they were taken, but also by the very nature of the points being related by time, there is already a pattern of points being close together due to the temporal nature of them bringing them somewhat close in proximity. If we had used some measure of distance that accounted for and worked well with time sequences, such as dynamic time warping, this would have been alright; however, our model utilized Euclidean distance for the sake of simplicity. Thus, our Euclidean k-means model failed to properly understand and analyze the data because it was not well-suited to analyze time as a feature.

Furthermore, the binary feature "Label" was not a feature that worked well with Euclidean k-means clustering either. Since the "Label" feature was only either a 0 or 1, this results in a clump of data points on 0 and 1, which influences the centroids to gravitate towards them separately. Because our model uses Euclidean distance, the feature either being a 1 or 0 makes it so that the distance on one dimension between points is either extremely close, given they are of the same "Label," or extremely far, given they are not the same. This disrupts our model, as Euclidean distance is sensitive to the scale of features; it fails to properly calculate meaningful distance when one of the dimensions it relies on is either extremely high or extremely low. Additionally, k-means clustering works best when the model is more spread out, or when it can be possible to assume a spherical shape exists within the data. When utilizing a binary feature, the data is segregated into two lines, thus making the shape of the data not something that works well with k-means. As the finisher, because of how the label is the feature that actually indicates if the data point is an anomaly or not, the model would likely catch on to this fact and start only predicting data points with label = 1 as anomalies. This would be true, but counterintuitive to our purpose, as we want to predict future anomalies that we don't know about, which would not be classified and given a label beforehand.

Given the thorough reasoning and explanation behind why two of the three features provided in the dataset do not work well with Euclidean k-means clustering, it should be easy to see that the algorithm was doomed from the start. Euclidean k-means clustering heavily depends upon using two features to find clusters and classify anomalies. However, after acknowledging that two of the three are ineligible, we are only left with one proper feature to use, that being utilization. Thus, the very nature of the AWS dataset is unfit for anomaly detection via K-means clustering, as there are not enough features to provide an accurate and meaningful analysis of the data.

Originally, the K level was set to 9, meaning the model had 9 centroids to cluster the data with. The statistics from this model with K = 9 were spectacular, with a recall of 100%, a precision of 91.43%, an F1 score of 95.52%, and an accuracy of 99.17%. Although these stats are very strong and seem to indicate that the model was working well, this

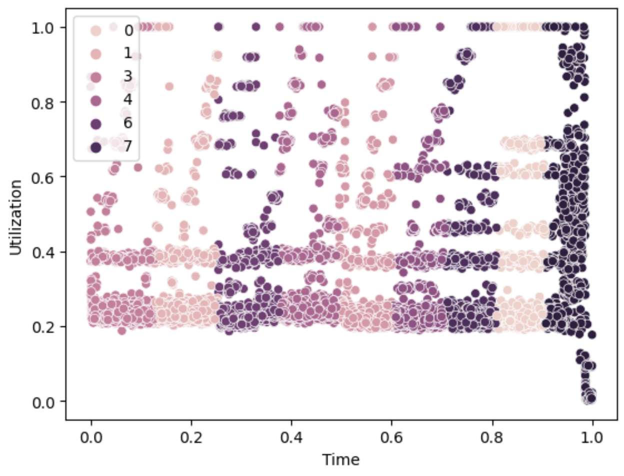


Figure 14. Visualization of Data by Time and Utilization, post K-Means Clustering at K = 9

is incorrect for two reasons. As stated earlier, k-means is not suited for this dataset. Thus, regardless of how well these stats are, they are not indicative of actually performing and predicting well, as the model has still failed to take into account the temporal patterns that inherently come with the time feature. In addition to this, if a closer look is taken at Figure 10, which graphs time and label, it can be see there is a heavy correlation with anomalies and data points past a specific timestamp. The cluster visualization for K = 9, provided in Figure 14, also seems to back up the idea that the model had just clustered data points based on time proximity, and had managed to isolate the time period responsible for anomalies as a cluster. Although this proved to work well with the test portion of the data, it would not work well with future data, as future data would have a different time, and its current manner for predicting anomalies is reliant on time periods that it knows.

As stated earlier, the choice of K was the idea that K = 3 was better than K = 9 as a result of the elbow method. However, the confusion matrix statistics for K = 3 are severely worse than those for K = 9. How, then, can K = 3 be "better" for the algorithm than K = 9? First, these statistics are misleading; they have been proven to not be indicative of much as stated earlier. Furthermore, a high K value comes with the risk of overfitting the model to the data and predicting the training data too well. That comes with the risk of overlooking important patterns and failing to properly predict future unseen data. As for this specific example, K = 3 performed worse due to how its anomalous cluster encapsulated many non-anomalous points, resulting in it predicting several regular points as anomalies. K = 9 did not have this issue, as it had enough clusters to where it could isolate the time period that contained anomalies, letting it predict false positives at a much lower rate. Our model and experiments aren't an example of overfitting, as for this respective data set it managed to capture the correct time period that anomalies were related to. However, both of these still failed to make any meaningful analysis or predictions, as they both still utilized time to predict and classify anomalies without factoring or respecting the temporal patterns that the time sequence includes.

9. Challenges, Shortcomings, and Future Work

Throughout the development of this paper, we encountered numerous challenges and obstacles, such as measuring the accuracy of a dataset that is heavily unbalanced, designing models to utilize the time-series aspect of the data, and selecting possible candidates for hyperparameters. At the beginning of our experiments, we ignored the imbalance in samples within the dataset, and we measured the models simply on accuracy. As a result, many of our models recorded high levels of accuracy, since they learned to only predict that there was no anomaly due to the fact that the majority of our samples contained non anomalous events. Although this impeded our development at first, we later mitigated this problem by selecting our models based on F1-score instead of Accuracy and by stratifying the splits within the training and testing sets, as well as the folds within cross-validation. Moreover, we were also challenged by the time-series aspect of the dataset. Initially, we assumed we could use timestamp as a feature and integrated it into our model by converting each timestamp into its UNIX equivalent. However, we later realized that we misunderstood the role of time as a feature, and later chose to ignore it from our experiments. In the future, we would be interested to see the performance of models such as LSTMs that are better suited to time-series data, and we are also interested to see whether we could train our Multi-Layer Perceptron to utilize the time-series aspect of the dataset by partitioning the samples into timeframes and passing the partitions separately at a time. Third, we also had difficulties choosing different sets of hyperparameters to test. Although we used Grid Search to select the best hyperparameters of the candidates we arbitrarily chose, we did not know an effective way to pick sets of hyperparameters that could potentially be optimal. In the future, we hope to research heuristics that we could have used to identify candidates for hyperparameter selection. Lastly, our initial plan for this experiment was to optimize for high recall. Clearly, our results showed that our models optimized for precision instead. Since the dataset contained significantly more samples that did not indicate an anomaly, we believe that our model was skewed toward predicting that an anomaly did not occur. In the future, we would want to investigate further methods for optimizing recall instead of precision.

10. Conclusions

This research project was conducted with the goal of determining which models were best suited to detect anomalous behavior from deployed software applications, and what exactly made them well-suited to do so. This was done through the process of training four different models on data provided by industry leader AWS and reviewing the results these models had once fully trained. The four methods used for these models were Logistic Regression, Single-Layer Perceptron, Multi-Layer Perceptron, and K-Means Clustering.

As our report details, each of these methods had distinct reasons for succeeding and failing to predict anomalies. Logistic Regression didn't perform well due to the data set not being linearly separable, which is necessary for Logistic Regression to be able to work well. In a similar vein, Single-Layer Perceptron also failed for the same reason of the data not being linearly separable. Both of these methods, as linear-based models, heavily rely on finding a linear relationship within the data in order to classify it. Therefore, training them with data that is unable to be linearly separated, such as the AWS data we used, hinders them greatly in producing any meaningful result. The third method we used, Multi-Layer Perceptron, had much more promising results than its previous counterpart. The Multi-Layer Perceptron's success is largely attributed to how its use of multiple layers allows it to classify data that is not linearly separable. Finally, the last method of K-Means Clustering failed to be properly applied to the data. Many of the data's features were unsuited for K-Means Clustering, rendering its strong performance useless as it failed to take the context of those features into account.

Our findings show that the best models for anomaly detection are nonlinear classifiers such as Multi-Layer Perceptron, which are able to detect patterns within data that is not linearly separable. In comparison, linear classifiers such as Logistic Regression and Single-Layer Perceptron are not well-suited to detect anomalies. Their heavy reliance on data being linearly separable prevents them from performing well on data that may not have a linear relationship present. As an outlier, although K-Means Clustering performed well statistically, its results cannot be taken at face value due to it failing to recognize the context of the data's features. Therefore, no conclusions on its performance can be drawn.

11. Contributions

11.1. Damon Lin

Damon worked on the development of the Multi-Layer Perceptron, and he wrote about the method, his experiments, his results, and the discussion on its performance in the section on the Multi Layer Percepton. In addition, he also wrote the report's introduction, the report's overview, the report's methods, and the discussion on the team's challenges and shortcomings.

11.2. Ritvik Durgempudi

Ritvik contributed to the project by doing the work on the single-layer perceptron model. He worked on running the model and collecting its results, as well as interpreting the results of the model and adding them to the presentation as well as to the report. In addition to his work on the single-layer perceptron portions, he also worked on writing the abstract for the report.

11.3. Wesley Tam

Wesley Tam worked on the training of the Logistic Regression and wrote about the method, experiments, and results of the experiment. He also researched other related works and papers regarding the Numenta Anomaly Benchmark dataset and handled most of the bibliography.

11.4. Jerry Tang

Jerry Tang handled the development of the K-Means Clustering model and wrote about the methods, experiments, results, and discussions relating to it as well. Further contributions include the conclusion.

12. GitHub

All code is hosted on GitHub at this repository: https://github.com/oprylord/466-project

References

[1] C. Freeman, J. Merriman, I. Beaver, and A. Mueen. Experimental comparison of online anomaly detection algorithms. In R. Barták and K. W. Brawner, editors, *Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference, Sarasota, Florida, USA, May 19-22 2019*, pages 364–369. AAAI Press, 2019.

[2] M. Gardner and S. Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627–2636, 1998.

[3] Kılıç. Perceptron model: The foundation of neural networks, Sep 2023.

[4] A. Lavin and S. Ahmad. Evaluating real-time anomaly detection algorithms - the numenta anomaly benchmark. Dec. 2015.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.