EVALUATING MACHINE LEARNING MODELS FOR ANOMALY DETECTION OF SYSTEM LOGS

by

Joseph Porter

A project report submitted to the faculty of

The University of Utah

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

April 2020

ABSTRACT

Training Machine Learning (ML) models for anomaly detection of system logs is a promising approach that empowers system administrators to quickly and automatically identify incidents in their systems. In this work, we have explored the abilities of various ML classifiers to identify anomalies in log sequences from CloudLab logs as determined by a separate unsupervised ML classifier. While all of the models investigated come from the sci-kit learn package in Python, each model differs in its algorithmic approach, making certain models like Logistic Regression and Multi-Layer Perceptron a better classifier choice than others.

TABLE OF CONTENTS

Table of Contents

CHAPTER 1

INTRODUCTION

Monitoring and management of large systems is often a time-consuming task for system administrators. With systems growing larger and scaling on increasingly popular cloud environments, the task of monitoring systems is difficult and varies widely between systems [1]. Logs generated by the system often contain the information needed to identify the occurrence of outages, security incidents or unexpected changes in behavior as well as their root cause [2]. However, the massive amount of log data is difficult to monitor in real-time by administrators and often only leveraged to diagnosis problems after they have been identified.

Commonly applied in Intrusion Detection Systems, a number of anomaly detection tools have been leveraged by system administrators to help monitor their systems [3]. Anomaly detection is meant to create systems that can detect atypical patterns in data which, for the purpose of monitoring system logs, would mean identifying abnormal sequences or sessions of logs [4]. Anomaly detection techniques have been used to monitor systems with success, but these anomaly detection systems are often very application specific, being designed to perform detection on a specific type of system [5]. This makes it difficult to apply such detection systems in a general manner with potentially multiple uses [6].

In anomaly detection systems, machine learning (ML) seems the perfect tool to leverage as it can sift through massive amounts of data to automatically learn the underlying principles that would describe normal and abnormal examples. While ML techniques have been successfully leveraged in anomaly detection, a number of challenges still exist for such approaches. Most ML approaches use supervised learning methods, which require large amounts of accurately labeled data such as log sequences that have been identified as normal or anomalous. These datasets of labeled information can be time-consuming to create and require expert knowledge to evaluate [7]. For system administrators, the time cost would often be prohibitive to produce the necessary labeled dataset to train an ML anomaly detection system for their particular systems.

Because of this difficult in hand labeling log sequences, this project attempts to simplify that challenge by using a separate, unsupervised learning technique to provide labels for log sequences. While this labeling process may not correctly identify all anomalous and normal sequences, it does provide labels for all examples so that supervised learning techniques can be trained and their performance evaluated on the CloudLab log sequences [8]. With the labeling technique, scikit-learn models based on supervised learning can be trained using the assigned labels and their performance evaluated.

CHAPTER 2

BACKGROUND RESEARCH AND MOTIVATION

The goal of evaluating the performance of ML classifiers on sequences of system logs was based on an understanding of existing and successful approaches in anomaly detection for this data type. This included learning about preprocessing necessary to train models with log sequences, the disadvantages of certain approaches and the challenge of labeling examples with expert knowledge.

2.1 Anomaly Detection

Providing pragmatic anomaly detection to system administrators requires an understanding of existing anomaly detection techniques and the problems they are designed to address. Most commercial anomaly detection systems rely on carefully tuned threshold values that are either designed for a specific application or must be manually determined by system administrators for their particular system. For system logs, false positives are particularly costly as they require a system administrator to investigate and determine if it relates to an incident that must be addressed [2]. Anomaly detection systems can suffer from high false positive rates, because of the rarity and variety of anomalies, reducing the usefulness of the system. Additionally, the accurate detection of an anomaly does not mean that the anomaly indicates some incident that requires a response from an administrator [4].

In a number of research application, deep learning techniques have shown substantial promise in detecting anomalies in system logs. However, deep learning generally requires a huge amount of data to train because of the number and complexity of the multiple layers in the model [9]. Creating those labeled sets of log sequences can be difficult and requires expert knowledge to perform accurately [10]. The ability of unsupervised and semi-supervised learning to learn on unlabeled data would generalize the task of anomaly detection but such systems have generally underperformed in comparison to supervised learning [11].

One work that motivates this project is DeepLog, which performs anomaly detection of system logs through deep learning [4]. Several aspects of DeepLog are designed to be particularly applicable to system administrators. Firstly, in addition to detecting anomalies a log line level, DeepLog also provides a workflow model that will assist in diagnosing root cause of the anomaly. Secondly, the system provides a feedback mechanism so that system administrators can identify false positives and DeepLog would learn from that information to reduce false positives in the future. This user feedback mechanism would help the system to reduce false positives, ignore unimportant anomalies and adapt to changing system behavior.

2.2 CloudLab System Logs

CloudLab is a cloud computing testbed that provides researchers with the infrastructure to perform cloud experiments with the control and visibility needed to test new architectures [12]. CloudLab clusters have almost 15,000 cores and provide a variety of machine types spread across three sites [13]. Because of the different types of resources and the large amount of user control over experiments, the behavior of CloudLab nodes vary greatly and create a significant challenge for anomaly detection.

With the goal of eventually monitoring CloudLab system logs with anomaly detection, we have collected system logs from a variety of CloudLab nodes for over a year. Log sources currently include:

- Stated.log: reports the status of an internal state machine used in some CloudLab processes
- Bootinfo.log: records the progress of nodes in the booting process
- Dhcpd.log: records DHCP events
- Reboot.log: logs the progress of requested reboots

The system logs are gathered, processed and stored in a standard ELK (Elasticsearch, Logstash, Kibana) stack to be used later in our research [14].

During the processing of the log messages, each message is checked against a series of regular expressions to determine which event ID to assign to the message. An event ID, also referred to as a log key, is a value assigned to a message that matches a specific log pattern. Since most log messages are of a particular format, types of log messages, ie. State_changed or DHCP_request, tend to have identical patterns that separate types of messages. It is a common technique in log parsing and often automated to extract both the event ID and the values of any parameters contained in the pattern [15], [16]. For our research, the patterns were manually specified to ensure accurate patterns and to increase our understanding of the log data. With the five log sources, we identified approximately three dozen log patterns and assigned each a unique event ID.

As part of the background research, invariant mining was applied to CloudLab system logs as a method to detect anomalies. Invariant mining utilizes counts of the event IDs to determine linear relationships between the counts that hold true for a large threshold of the data [17]. For example, the count of “open file” event IDs would equal the number of “close file” counts. To perform invariant mining, set of event IDs had to be grouped into sessions from a specific range of timestamps and machine locations, which are used to count the occurrences of each event ID for mining invariants [2]. While some system logs may have very distinct sessions, such as for a particular job or task, CloudLab logs do not have similar delineations. Grouping at the experiment level does not provide an acceptable level of granularity since experiments can run for extended periods of time and only allow anomaly detection after the experiment has ended. To form sessions of the log sequences, each session was formed of the five target logs for a particular machine node and a specified day. This enables session-based anomaly detectors to identify normal or abnormal behavior at a daily, node level.

2.3 Moving Forward

The background detailed in this section provided several important elements that influenced the direction of experiments. First, the background research identified important preprocessing steps needed for logs to give structure to the unstructured logs messages by assigning event IDs as well as methods of grouping into sessions. These steps are necessary for many anomaly detection techniques to be applied on this type of data and opened up the possibility of evaluating a great many methods. Secondly, the background work found disadvantages or weaknesses in a number of anomaly detection methods that emphasized the importance of reducing false positives as well as the challenge of identifying anomalies in continuous sequences. This motivated experiments to reduce false positives and analysis to explore how classifiers performed throughout an anomalous sequence.

CHAPTER 3

EXPERIMENT METHODS

The experiments for this project consisted of several distinct efforts to reach its final goal. From the CloudLab logs previously gathered, a data set was selected from a subset and formed into usable examples. After forming the data, it was labeled using an unsupervised learning technique, invariant mining. This data set was then used in a variety of experiments that trained and evaluated classifiers.

3.1 Data Set

From the previously discussed CloudLab log data, a data set was formed from the logs of the 200 HP type machines for the year of 2019. This set was then further subdivided into training and testing sets. The training set contained logs from January to September 2019, resulting in 24,430 sessions, while the test set contained all logs from October to December 2019, containing 9,708 sessions. The log entries were grouped based on their assigned session ID, which formed groups of logs for each machine for each day. The event IDs of these sessions were then ordered based on their timestamp and offset, an indicator of order when timestamps are identical, to form the sessions into chronological examples.

To train ML models with these examples, they first had to converted to features vectors. Features vectors were formed in several ways depending on the experiment's intended purpose. One method to convert to feature vectors was a simple unigram representation; a count of the occurrences of each event ID in a session. This resulted in a vector based on the number of event IDs with each item in the vector representing the count of that event ID. This method provides a simple representation that preserves only the counts of event IDs and disregards the ordering of the logs. Another approach created feature vectors using bigram counts. In this case, each bigram represents a transition from one event ID to another. This creates a feature vector where each entry represents some unique, chronological transition X->Y where X and Y are both event IDs. Unlike the unigram representation, the bigram representation preserves some information about the ordering of messages because transitions are counted. The final method still creates feature vectors of event ID counts but does so with partial sessions instead of complete day-long sessions. With this method, the log messages are processed chronologically, and a new example feature vector is created for the log sequence up to that point. This allows the examples to be considered at any point in the sequence, thus performing similarly to a real-time system that performs anomaly detection as new log messages occur.

3.2 Labeling with Invariant Miner

To use supervised learning approaches, each example needs to be assigned a label for the learning process to function. These experiments used an invariant miner, the previously mentioned unsupervised learning technique, to label the examples. While the labels supplied by the invariant miner may not reflect the true labels of the examples, they are a suitable starting point to explore the efficacy of these learning techniques. In a related paper recently submitted for publication, systems experts labeled a set of logs sequences to compare to the invariant miner. The results:

"The accuracy of the invariant miner was reasonable: it correctly labeled 70% of sessions identified by the administrators as normal, and 73% of the sessions labeled as anomalous. This gives us an overall false positive rate of 30% and false negative rate of 27%."

The data set was labeled with a single set of invariants to ensure consistent labeling across the training and test set. This labeling method resulted in 1,972 out of the 24,430 sessions in the training set to be labeled as anomalous while the test set was labeled as containing 455 anomalous sessions out of 9,708.

3.3 Classifier Selection

For this project, all classifiers were selected from those supplied in the scikit-learn [18] Python library. The library provides simple, open source tools for data analysis built on other common Python libraries NumPy [19], SciPy [20] and matplotlib [21]. The classifiers selected for this project had characteristics that seemed suitable for this type of classification problem but were also varied enough to warrant exploration of each.

3.3.1 Cross Validation

Each ML classifier requires a set of hyper-parameters that influence the learning process for that classifier. These hyperparameters are not learned directly during the training of the model and are instead evaluated in a process called cross validation to improve the performance and generality of the classifier. In this project, hyperparameters were evaluated in a grid search, meaning that each set of proposed hyperparameters were evaluated and the best set of parameters was chosen for the final training of the model. To ensure that each set of hyperparameters were evaluated correctly, cross validation was performed with 5-fold cross validation. For each set of parameters, the data was divided into five splits of data were formed from the training set. Then, five models were trained with the specified hyperparameters and all but one of the splits of data. The held out split is used for testing of the model to evaluate performance and all of the scores were averaged together to select the best set of hyperparameters.

CHAPTER 4

EXPERIMENT RESULTS

This section includes results for the various experiments performed to evaluate the performance of classifiers from the Python scikit-learn package. The experiments begin with evaluations of SVM and logistic regression classifiers to establish a baseline. Later experiments work with different classifiers and analyze performance with an emphasis on reducing false positives and observing partial sequences.

4.1 Classification with Logistic Regression and SVM

The first experiments for this project began by testing the efficacy of the support vector machines (SVMs) [22] and logistic regression [23] classifiers from the sci-kit learn package on this data set. Because the data is labeled with a series of linear relationships established by the invariant miner, SVMs appeared to have a high chance of success given their strength in linear classification while also being very effective in high dimensional spaces [24]. Similarly, logistic regression was chosen because it is a linear model and was expected to perform well based on the labeling scheme [25].

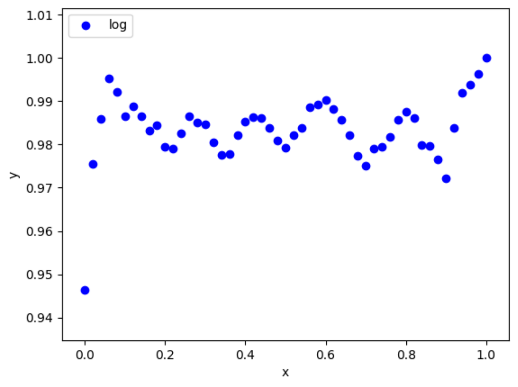
These models were trained with features vectors of unigram event IDs and cross validation was performed to select the best hyperparameters. When evaluating on the test data, the results showed that both classifiers were trained to correctly label all examples in the test set. Given the rarity of classifiers perfectly labeling a test set, the models were investigated, and both showed that they had learned linear relationships that enabled them to achieve this result. The performance of these models was partially a result of labeling method as well as the linear nature of these classifiers.

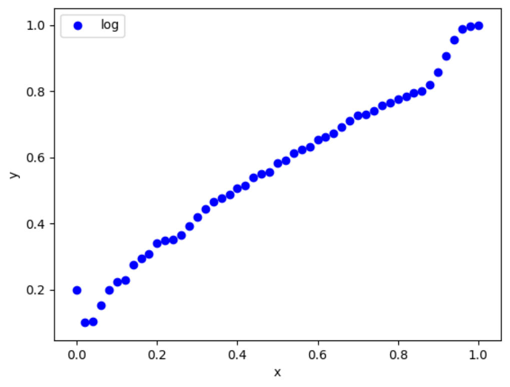
4.2 Further Logistic Regression Experiments

Since the previous experiment demonstrated that logistic regression and SVM classifiers could perfectly classify entire sessions of logs, the next experiment was meant to explore how much data in a log sequence was needed for the classifier to correctly label it. In this experiment, log sequences from the test set were used to generate partial examples so that predictions could be made on all progressions of the sequence resulting in 3,151,655 examples. For each feature vector, the percent of progress in the sequence was tracked to enable further data analysis. Below is the confusion matrix showing the performance of the logistic regression classifier on the partial examples from the test set:

	Labeled Normal	Labeled Anomaly
Predicted Normal	2,976,781 (94.4%)	55,455 (1.76%)
Predicted Anomaly	50,430 (1.60%)	68,989 (2.19%)

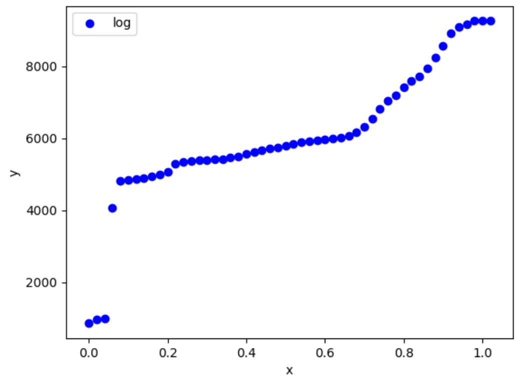
These results show that 42% of identified anomalies are actually normal sequences. This high false positive rate is alarming because of the implications if this was actually a real-time system. The below graphs show the percent of correct examples for each label based on the percent of progress through the log sequence.

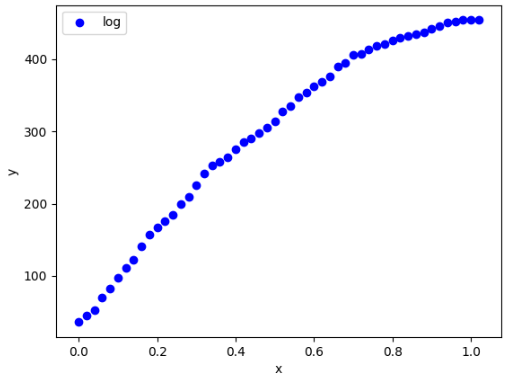




The above graphs demonstrate several interesting points about the performance of the classifier on partial sequences. When labeling normal sequences, the classifier is highly accurate throughout but varies before correctly classifying all sequences at the end. For anomalous sequences, the classifier gradually becomes more accurate as it receives more of the sequence. This is likely because the anomalous behavior occurs at a specific point in the sequence and would appear normal before that.

Using this same data, an analysis was performed to determine at what point the classifier ‘converges’ for each example, that is, all further predictions for that session are correct. This is especially important when trying to understand the frequency of false positives, normal sequences that are incorrectly labeled as anomalies, when the sessions are considered as partial examples.





The convergence graph for the normal sequences demonstrates that the classifier frequently mislabels normal sequences before converging at the end. For anomalies, the convergence graph shows steady progress as more sequences converge with longer sequences.

While the logistic regression model predicts correctly for complete sessions, it does predict a large number of false positives in the partial sequences. This would be particularly troublesome if such a model was leveraged as a real-time system to identify anomalies as they occurred. The model was trained on complete sessions because the model is known to predict the correct label at the end, so the model is not designed to label partial results. Similarly, the anomalous examples cannot always be labeled early in the sequence because the anomalous behavior has not occurred or been predicted. Overall, the classifier does show promise based on its results on full sessions, but the high false positive rate is concerning for this model with partial sequences.

4.2.1 Logistic Regression with Bigrams

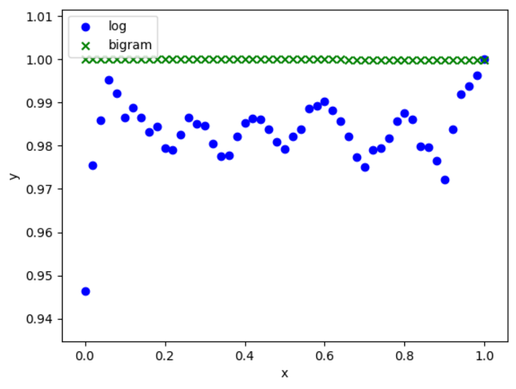
After observing how the logistic regression model performs on partial data, this experiment explores if that performance can be improved by providing more information to the model. In this experiment, each example is now formed using both unigram and bigram information so that the feature vectors now hold information about event ID counts and counts of event ID transitions. Training the model with these new bigram type vectors results in the following confusion matrix for the complete sessions in the test set.

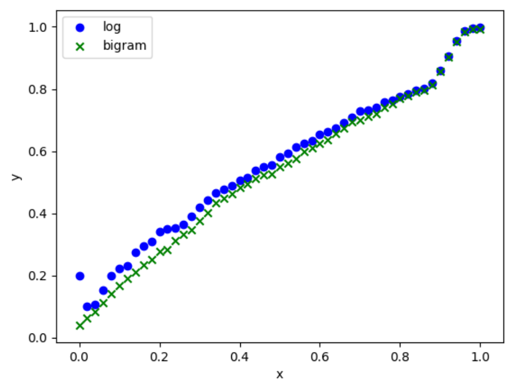
	Labeled Normal	Labeled Anomaly
Predicted Normal	9252 (95.3%)	4 (0.04%)
Predicted Anomaly	1 (0.01%)	451 (4.64%)

Unlike the classifier trained with unigram features, this classifier mislabels several of the sessions but does not severely degrade the performance. This is likely a consequence of the expanded feature space, which would require more examples to predict all complete sessions correctly. When predicting on partial examples, the bigram model results in the following confusion matrix:

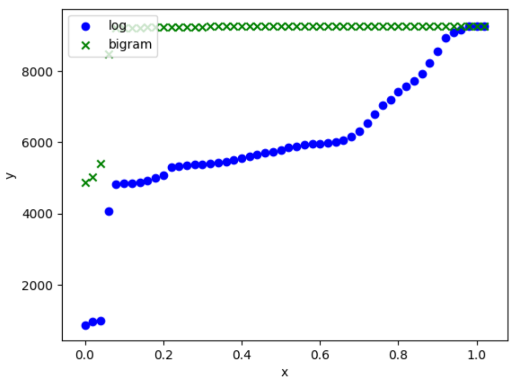
	Labeled Normal	Labeled Anomaly
Predicted Normal	3,027,165 (96.1%)	59,134 (1.88%)
Predicted Anomaly	46 (0.001%)	65,310 (2.07%)

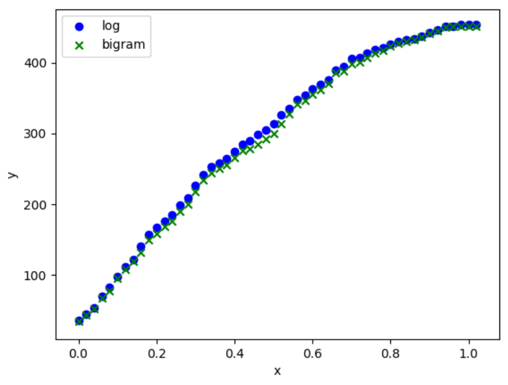
The confusion matrix shows that when given bigram features as well, the model performs slightly worse in identifying anomalies but correctly labels nearly all of the normal examples. This is also seen in the graphs showing percent correct:





The graphs demonstrate that the classifier correctly labels nearly all normal sequences throughout while only slightly underperforming the previous classifier on anomalous sequences. Similarly, in the convergence graphs:





These results are quite interesting because they indicate that such a classifier could perform well in a real-time system since nearly every partial sequence labeled anomalous is actually labeled as an anomaly. The larger feature space does appear to have an effect of reducing the performance on complete sessions, but this could potentially be solved with additional training data or creating an ensemble of this classifier and the classifier trained on just unigram features.

4.2.2 Logistic Regression Trained on Partial Examples

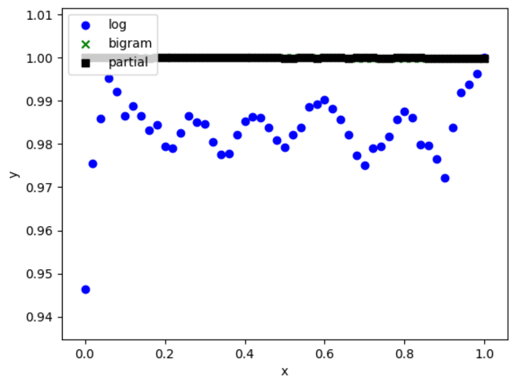
In the previous experiments, the classifier was trained using only examples formed from complete sessions. This was done because identifying every anomaly is important and the initial logistic regression classifier was able to do this using only complete sessions as training examples. However, after analyzing the results on partial data, a next step is to see if training on partial data improves the results predictions on partial data. This experiment does exactly that; training and testing the classifier using partial examples. The confusion matrix for the resulting classifier on complete sessions is shown below:

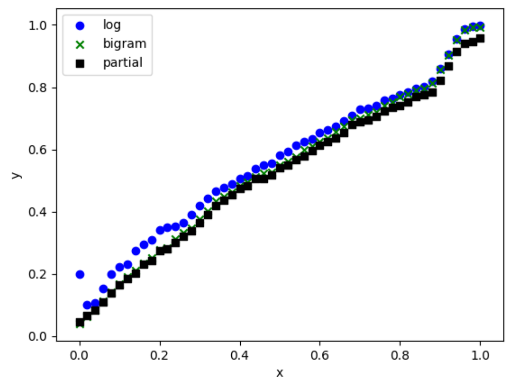
	Labeled Normal	Labeled Anomaly
Predicted Normal	9252 (95.3%)	19 (0.20%)
Predicted Anomaly	1 (0.01%)	436 (4.49%)

This classifier does mislabel more anomalies than the previous classifier, but the change is not extreme. The confusion matrix for the partial sequences are shown below:

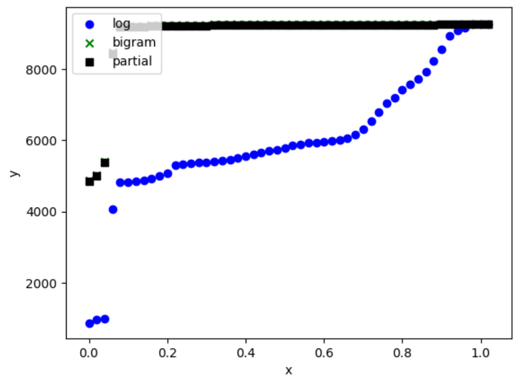
	Labeled Normal	Labeled Anomaly
Predicted Normal	3,027,128 (96.0%)	60,874 (1.93%)
Predicted Anomaly	83 (0.003%)	63,570 (2.02%)

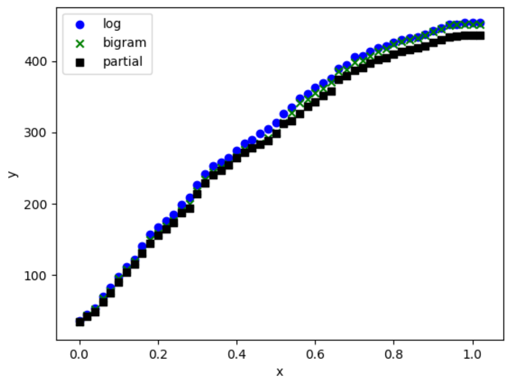
Similar to the classifier trained with bigram features, this classifier greatly reduces the false positives since nearly all predicted anomalies are labeled as anomalies. This model does not perform as well as the bigram model, but it does demonstrate that training the model with partial examples does have benefits when evaluating partial examples. Again, this is also observable in the percent correct graph:





As well as in the convergence graph:





These results demonstrate that training with partial sequences, similar to adding bigram features, significantly reduces the false positive rate while slightly reducing the accuracy in labeling anomalies.

4.3 Other Classifier Experiments

After exploring the performance of multiple approaches with logistic regression, our experiments shifted to finding other appropriate classifiers to compare. These classifiers also came from the sci-kit learn library and each was selected because of its different characteristics and high potential for success.

4.3.1 Naïve Bayes

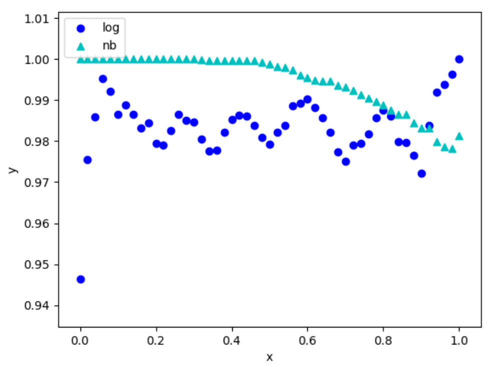
Naïve Bayes classifiers are supervised learning algorithms that leverage Bayes' theorem and make the naïve assumption of conditional independence between features. Sci-kit learn has a number of Naïve Bayes classifiers, several of which were evaluated for this experiment. Initially, the Complement Naïve Bayes algorithm, which is described as being suited for imbalanced data sets, and Multinomial Naïve Bayes algorithm, which is a variant used in text classification, seemed like the most promising approaches [26]. However, both proved to perform very poorly for this dataset. Instead, the Gaussian Naïve Bayes algorithm performed the best and is evaluated in this experiment [27]. As its name suggests, the Gaussian Naïve Bayes assume the likelihood of a feature to be Gaussian. The classifier was trained on complete sessions with unigram features; its performance on the complete sequences in the test set are below:

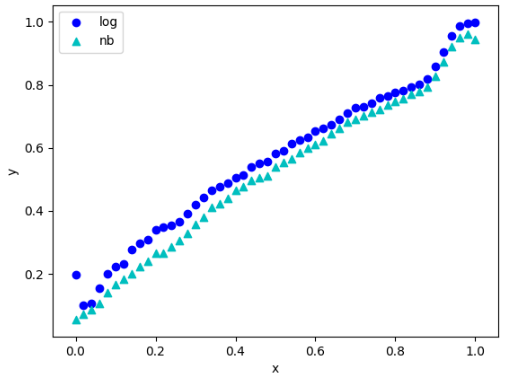
	Labeled Normal	Labeled Anomaly
Predicted Normal	9081 (93.54%)	25 (0.26%)
Predicted Anomaly	172 (1.77%)	430 (4.43%)

This is the worst performance of any of the classifiers thus far, but the model is fairly simplistic and does reasonably well. The number of mislabeled normal sequences jumps from previous classifiers. For the partial examples, the confusion matrix is below:

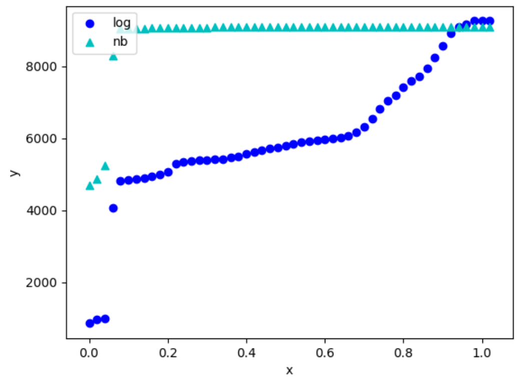
	Labeled Normal	Labeled Anomaly
Predicted Normal	3,012,161 (95.6%)	61,015 (1.94%)
Predicted Anomaly	15,050 (0.48%)	63,429 (2.01%)

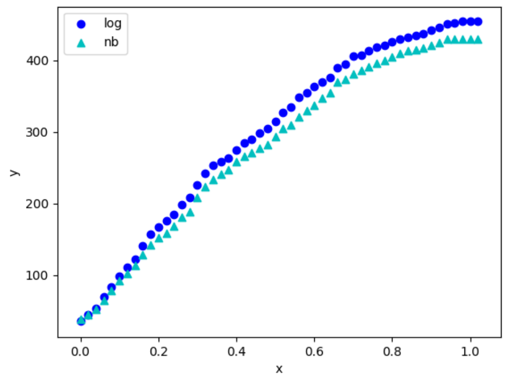
The results appear fairly average as a classifier using just unigram features, predicting the normal sequences better than the logistic regression but predicting the anomalies with less accuracy. These results are more noteworthy when the percent correct graphs are considered:





Interestingly, the percent of correct predictions for normal examples decreases as the sequence progresses. The percent of correct predictions for anomalies is slightly lower than logistic regression but follows the same trend. The convergence graphs are below:





Despite the differences in the percent correct graph, the convergence for normal sequences jumps very quickly, although it never converges for some of the sequences. The convergence graph for anomalous sequences is very similar to the logistic regression with slightly less accuracy. This classifier does perform rather well considering the assumptions made by the model but may not be the most appropriate classifier for this problem.

4.3.2 Multi-Layer Perceptron

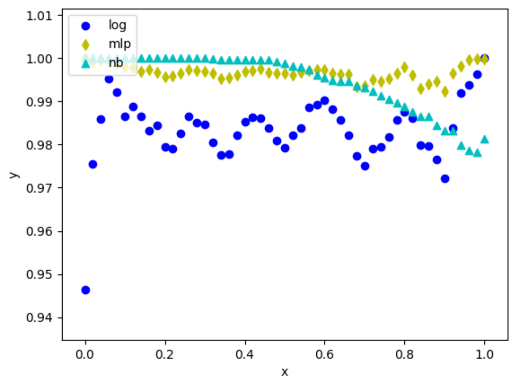
The Multi-layer Perceptron (MLP) algorithm differs from the other models used because it is a type of neural network. The MLP algorithm contains the hidden layers typical of a neural network and has the advantage of being able to learn non-linear models [28]. As the only neural network model in sci-kit learn, MLP is clearly different from the models used in previous experiments [29]. After evaluating multiple hidden layer designs, the model performed best with a simple (20,5) layout. This layout has twenty nodes in the first layer and five in the second. The results on complete sessions are shown below:

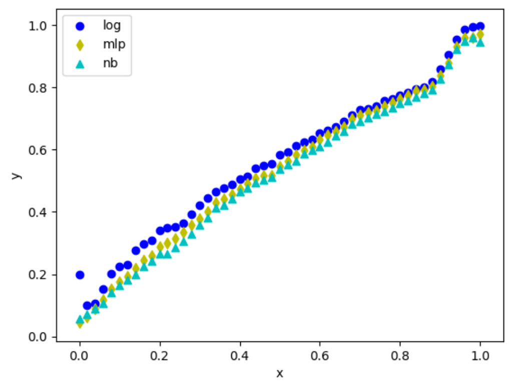
	Labeled Normal	Labeled Anomaly
Predicted Normal	9252 (95.3%)	13 (0.13%)
Predicted Anomaly	1 (0.01%)	442 (4.55%)

These results show the model to predict nearly all of the normal sequences correctly while incorrectly classifying several anomalies. The confusion matrix for MLP on partial sequences in below:

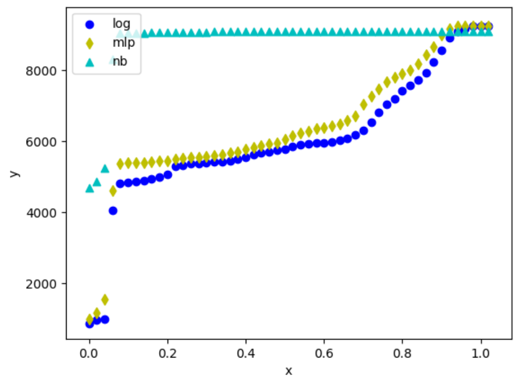
	Labeled Normal	Labeled Anomaly
Predicted Normal	3,017,168 (95.7%)	59,161 (1.88%)
Predicted Anomaly	10,043 (0.32%)	65,283 (2.07%)

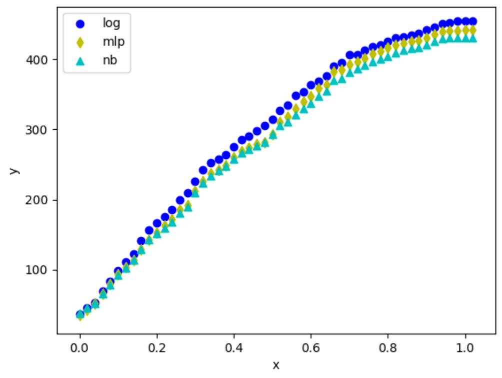
The MLP classifier   identified anomalies as well as any of the models evaluated and had fewer false positives than the classifiers using Naïve Bayes or Logistic Regressions and unigram features. The percent correct graphs are shown below:





The MLP classifier does better in labeling normal sequences but still varies slightly like the logistic classifier while slightly underperforming logistic regression in identifying anomalies. The convergence graphs are below:

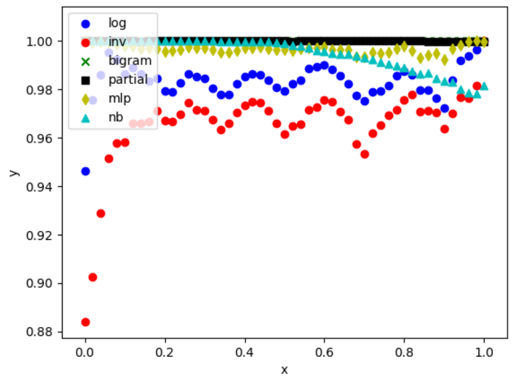


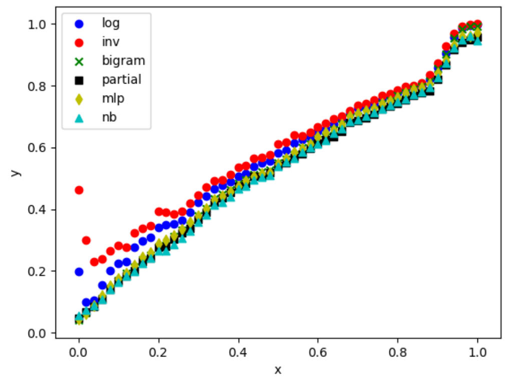


Again, the MLP classifier converges more quickly for normal sequences and does slightly worse for normal sequences than the logistic regression model. These results do suggest that the MLP classifier could be a good choice for classifying this data set. The results would likely improve with more examples as neural networks typically require large numbers of examples to train all of the hidden layers.

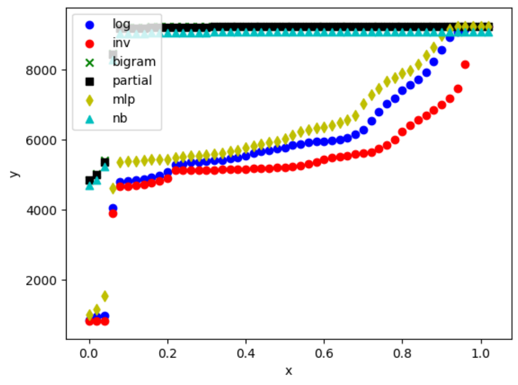
4.4 Comparing Classifiers

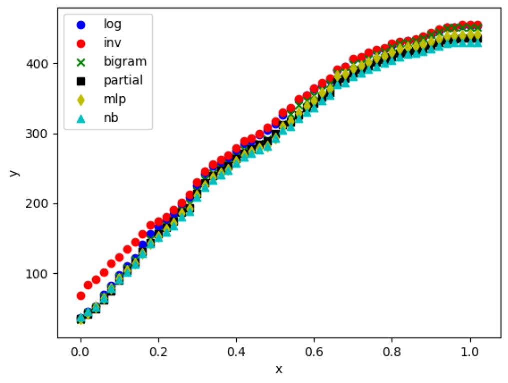
When comparing all of the results for these experiments, it is important to consider that each experiment uses a fundamentally different classifier or provides a different set of information to the model. Below are the percent correct graphs for all experiments including an entry for the invariant miner, the classifier used to provide labels:





Overall, the percent correct graph for the normal sequences shows more variety in performance, although the performance is generally very high. Logistic regression clearly outperforms invariant mining but MLP clearly has even better performance. This indicates the ranking of each model for predicting invariant miner labels using partial sequences. The models using bigram features and partial sequences for training demonstrate that providing this type of information greatly reduces the number of false positives. For anomalies, the classifiers do not vary as greatly with only small differences in performance between models but the same upward trend in identifying anomalies.





The convergence graphs for all classifier do not offer any additional information that is not shown in the percent correct graph. They do emphasize how quickly the models with bigram and partial training data identify normal sequences.

Chapter 5

CONCLUSION

The experiments in this project explore the performance of a number of sci-kit learn classifiers on a data set of on a collection of CloudLab system logs. The evaluations explored using day-long sessions and partial sessions that better reflect a real time monitoring system. Based on the results, logistic regression, and multi-layer perceptron show promise in both session-based identification while the variations of logistic regression and multi-layer perceptron models demonstrate potential for operating real time anomaly detectors.

Future work with these experiments would suggest several avenues to improve this work or leverage these results. A better classifier could almost certainly be created by creating an ensemble, a ML technique that leverages multiple classifiers to make a single prediction, to combine the strengths of several of the promising models. Additionally, evaluating the various training methods used for the logistic regression model could be applied to the Naïve Bayes and MLP classifiers to note their effects. Further work could also be done to evaluate implementing a real-time monitoring system for CloudLab data. Several issues regarding the number of false positives and identifying anomalies that require system administrator intervention would need to be explored.

REFERENCES

[1] B. Canner, “4 Challenges of Traditional Log Management Solutions,” 2018. [Online]. Available: https://solutionsreview.com/security-information-event-management/4-challenges-traditional-log-management-solutions/. [Accessed: 13-Nov-2019].

[2] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience Report: System Log Analysis for Anomaly Detection,” in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2016, pp. 207–218.

[3] M. Tanase, “One of These Things is not Like the Others: The State of Anomaly Detection | Symantec Connect Community,” 2002. [Online]. Available: https://www.symantec.com/connect/articles/one-these-things-not-others-state-anomaly-detection. [Accessed: 14-Nov-2019].

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the ACM Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[5] D. Balaban, “Why anomaly detection should be your number one priority in 2019 | Articles | Big Data | Innovation Enterprise,” 2019. [Online]. Available: https://channels.theinnovationenterprise.com/articles/why-anomaly-detection-should-be-your-number-one-priority-in-2019. [Accessed: 14-Nov-2019].

[6] A. (UC B. Oliner, A. (Splunk) Ganapathi, and W. (Google) Xu, “Advances and Challenges in Log Analysis - ACM Queue.” [Online]. Available: https://queue.acm.org/detail.cfm?id=2082137. [Accessed: 13-Nov-2019].

[7] D. Li, D. Chen, J. Goh, and S. Ng, “Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series,” in *BigMine 2018 : 7th BigMine Workshop on Big Data Mining (at SIGKDD 2018)*, 2018.

[8] “CloudLab.” [Online]. Available: https://www.cloudlab.us/. [Accessed: 14-Nov-2019].

[9] “Why go large with Data for Deep Learning? - Towards Data Science.” [Online]. Available: https://towardsdatascience.com/why-go-large-with-data-for-deep-learning-12eee16f708. [Accessed: 13-Nov-2019].

[10] F. Di Mattia, P. Galeone, M. De Simoni, and E. Ghelfi, “A Survey on GANs for Anomaly Detection,” Jun. 2019.

[11] S. Curtis, “Should We Be Rethinking Unsupervised Learning?,” 2016. [Online]. Available: https://www.kdnuggets.com/2016/08/rethinking-unsupervised-learning.html. [Accessed: 15-Nov-2019].

[12] “CloudLab - Software Technology.” [Online]. Available: https://www.cloudlab.us/technology.php. [Accessed: 14-Nov-2019].

[13] “CloudLab - Hardware.” [Online]. Available: https://www.cloudlab.us/hardware.php. [Accessed: 14-Nov-2019].

[14] “ELK Stack: Elasticsearch, Logstash, Kibana | Elastic.” [Online]. Available: https://www.elastic.co/what-is/elk-stack. [Accessed: 14-Nov-2019].

[15] “GitHub - logpai/logparser: A toolkit for automated log parsing [ICSE’19, TDSC’18, DSN’16].” [Online]. Available: https://github.com/logpai/logparser. [Accessed: 14-Nov-2019].

[16] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “An evaluation study on log parsing and

its use in log mining," in *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, 2016, pp. 654–661.

[17] A. Pecchia, S. Russo, and S. Sarkar, "Assessing Invariant Mining Techniques for Cloud-based Utility Computing Systems," *IEEE Trans. Serv. Comput.*, Mar. 2017.

[18] "scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation." [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 10-Apr-2020].

[19] "NumPy — NumPy." [Online]. Available: https://numpy.org/. [Accessed: 10-Apr-2020].

[20] "SciPy.org — SciPy.org." [Online]. Available: https://www.scipy.org/. [Accessed: 10-Apr-2020].

[21] "Matplotlib: Python plotting — Matplotlib 3.2.1 documentation." [Online]. Available: https://matplotlib.org/. [Accessed: 10-Apr-2020].

[22] "sklearn.svm.SVC — scikit-learn 0.22.2 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 10-Apr-2020].

[23] "sklearn.linear_model.LogisticRegression — scikit-learn 0.22.2 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed: 10-Apr-2020].

[24] "1.4. Support Vector Machines — scikit-learn 0.22.2 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: 10-Apr-2020].

[25] "1.1. Linear Models — scikit-learn 0.22.2 documentation." [Online]. Available:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. [Accessed: 10-Apr-2020].

[26] “1.9. Naive Bayes — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed: 10-Apr-2020].

[27] “sklearn.naive_bayes.GaussianNB — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. [Accessed: 10-Apr-2020].

[28] “sklearn.neural_network.MLPClassifier — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. [Accessed: 10-Apr-2020].

[29] “1.17. Neural network models (supervised) — scikit-learn 0.22.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Accessed: 10-Apr-2020].