WOLF: Automated Machine Learning Workflow Management Framework for Malware Detection and other applications

Sohaib Kiani[1], Sana Awan[1], Jun Huan[2], Fengjun Li[1], and Bo Luo[1]

sohaib.kiani@ku.edu,sanaawan@ku.edu,lukehuan@shenshangtech.com,fli@ku.edu,bluo@ku.edu

[1] University of Kansas, Lawrence, KS; [2] StylingAI Inc

ABSTRACT

Applying machine learning techniques to solve real-world problems is a highly iterative process. The process from idea to code and then to experiment may require up to thousands of iterations to find the optimum set of hyper-parameters. Also, it is hard to find best machine learning techniques for a given dataset. The WOLF framework has been designed to simultaneously automate the process of selecting the best algorithm and searching for the optimum hyper-parameters. It can be useful to both who are novice in machine learning and just want to find best algorithm for their dataset, and also to those who are experts in the field and want to compare their new features or algorithm with state of the art techniques. By incorporating the WOLF framework in their designs, it is easier for novices to apply machine learning techniques on their dataset. With a wide range of evaluation metrics provided, WOLF also helps data scientists to develop better intuition towards machine learning techniques and speed up the process of algorithm development. Another main feature of the WOLF framework is that user can easily integrate new algorithms at any stage of the machine learning pipeline. In this paper, we present the WOLF architecture, and demonstrate how it could be used for standard machine learning datasets and for Android malware detection tasks. Experimental results show the flexibility and performance of WOLF.

KEYWORDS

Classifier, Malware Detection, Parameter Selection

1 INTRODUCTION

Machine learning has been quite successful in various fields, such as health, finance, education, sports, computer vision, speech recognition, which has encouraged demand for machine learning systems amongst novices in the field. Open source libraries [10] [20] have

been built to help novices use machine learning algorithms. By definition, machine learning automates the task of learning in terms of rule induction, classification, regression etc. to draw knowledgeable insights and to forecast an event before it actually takes place. Despite this, the task of selecting the best algorithm(s) for a specific dataset is not automated [5]. With the rapidly growing pool of available machine learning algorithms, it becomes difficult for novices as well as researchers to choose the best algorithm specific to their given dataset. The main steps of any machine learning system consists of preprocessing to prepare data, followed by feature selection/extraction to remove noise or redundant information from data, and then finally applying and evaluating machine learning techniques to gather useful information about the data [19].

Automation is the fuel that drives WOLF. Automating time-consuming and repeatable tasks are the defining characteristics of the project. The rising scope of Artificial Intelligence (AI) and machine learning increases the need for automation to simplify the process and hence help researchers and data scientists to dig deeper into the problem and understanding it well rather than spending time in tweaking the algorithms and the parameters involved. The positive correlation of growing intelligence and the complexity of solutions has shifted the trend from Artificial Intelligence (AI) to Automated Intelligence, a paradigm on which WOLF is based.

WOLF has been built to have impact on a wider audience than traditionally targeted by the machine learning community. The automation of machine learning pipeline helps people with different levels of expertise and requirements, helping novices to identify the best possible combination of algorithms without even having in-depth knowledge of the inner workings of the algorithms. At the other end of the spectrum, WOLF serves as a useful supporting tool for seasoned researchers and businesses to figure out the best resulting hyper-parameters, helping them reduce the time-to-market. The project is developed with the idea behind rising interest of data scientists towards Data Science competitions, as the machine learning automation has become a necessity as competitors spend lot of time in model construction and evaluation.

Although the project so far supports the classification problems but there are a handful of classification algorithms consisting of base classifiers and ensemble methods as well that are not currently supported. WOLF framework provides the leverage of adding proprietary algorithms, making it more customizable since the scope of growing algorithms in the system is unrestricted. The process of adding an algorithm to the pipeline is pretty intuitive and straight forward. Also, the idea of not stealing the freedom of model evaluation from the user has been taken care of in the project; the result file consists of various widely used metric calculations giving user better understanding and control through easy comparison of

pipeline combinations, therefore helping the user make decisions with high confidence.

In this paper architecture of WOLF framework will be explained. The usability of the WOLF framework will be demonstrated through experiments on various types of datasets. Moreover, the results generated by WOLF will be used to support the following hypothesis:

(1) Comparison between Deep and Shallow Models: Performance comparison between deep and shallow networks explains whether using deep networks is always beneficial or not?

(2) Comparison between Machine Learning Algorithms: Performance comparison between different Machine Learning algorithms supported by WOLF demonstrate if they perform differently for a given dataset?

(3) Fine-Tuning Hyper-parameters: Is there any benefit of doing fine tuning of hyper-parameters?

This paper has been organized as follows: Section II gives brief overview about similar frameworks developed in the past. Section III highlights main features of the WOLF framework. Section IV and V, provides detailed descriptions about architectural and database management aspects of WOLF. In section VI, the usability of the WOLF framework has been demonstrated by generating results for various data sets. Also, through experimental results above mentioned hypothesis regarding machine learning techniques has been described in section VII. Section VIII highlight some of the future work in order to make WOLF framework more useful.

2 RELATED WORK

In recent years, various platforms and techniques have been developed for easing cumbersome tasks in a machine learning (ML) workflow, such as data pre-processing, feature extraction and selection, and model selection. They can be categorized into two categories, ML workflow management and ML workflow discovery.

2.1 ML Workflow Management

Platforms and techniques in this category provide tools for creating, modifying, executing or sharing ML workflows. The FBLearner Flow system from Facebook was designed to be capable of easily reusing algorithms, scaling to run thousands of simultaneous custom experiments, and managing experiments with ease. KNIME [4] enables easy visual assembly and interactive execution of a data pipeline through customizable and extendable nodes. The portable format for analytics is an emerging standard for statistical models and data transformation engines [18]. PFA combines portability across systems with algorithmic flexibility: models, pre-processing, and post-processing are all represented as functions that can be arbitrarily composed, chained, or built into more complex workflows. Kepler [15] provides an graphical interface for creating a "scientific workflow" an executable representation of the steps required to generate results.

2.2 ML Workflow Discovery

The techniques in the second category is more related to WOLF in the sense that they aim to discover an optimal ML workflow for a ML task. TPOT is a Python automated machine learning tool that optimizes machine learning workflows using genetic programming

[17]. The scalability of TPOT may be problematic since its process of finding optimal workflow were not designed for distributed computation. To addressing the issue of scalability in machine learning, Tim et al. proposed MLBase framework [14] consisting of three components, ML Optimizer, MLI, and MLLib. Through ML Optimizer, an optimal learning plan can be selected for a ML task, such as classification, specified using a declarative language. Note that ML Optimizer is still under development and MLLib is specifically designed for Spark. It requires non-negligible efforts to implement an Spark compatible algorithm to achieve distributed computation. DataRobot is a proprietary data science system with software that covers the tasks that a data scientist typically performs. It is designed to automate the task of data cleaning, visualization, model construction, model evaluation, and making predictions.

3 FEATURES OF THE WOLF FRAMEWORK

The objectives of WOLF framework are quite similar to other related projects mentioned in previous section [5] [17] . However, WOLF implementation is much simpler and there are certain features of WOLF systems that makes it a more attractive automated machine learning platform to use for a wide variety of users. Following are the main features of WOLF platform that differentiate it from similar existing frameworks:

3.1 Standard Input Format

In order to make preparation of Input Data simple, Wolf requires input dataset should be converted into "*.arrf" format, which is the most common input file format among other similar platform like Auto-Weka [5].

3.2 Integration of New Algorithms

To ease the task of developing new algorithms at any stage in the machine learning pipeline(preprocessing,feature selection, machine learning algorithms) WOLF provides the user with a hands-on tool to test and compare newly designed algorithms with state of the art techniques without worrying about other stages of machine learning pipeline. Consequently, it speeds up the development phase of algorithms and saves lots of time and effort in redesigning other stages of the machine learning pipeline.

3.3 Search Optimum Hyper-parameters

It is possible to search the hyper-parameter space for the best cross validation score through WOLF. WOLF also enables users to search optimum hyper-parameters for selected machine learning algorithms. User s can provide the WOLF platform with a range of possible hyper-parameter values for which WOLF performs Grid search. Together with optimum hyper-parameter search for each algorithm, WOLF provides the user with the algorithm that performs best for given dataset using cross validation scores.

3.4 Built to Support Wide Range of Tasks

WOLF has been designed to cover a wide range of machine learning tasks, which include binary and multi-class classification. Different metric evaluation criteria have been integrated into WOLF to develop better intuition of machine learning models for a given dataset.
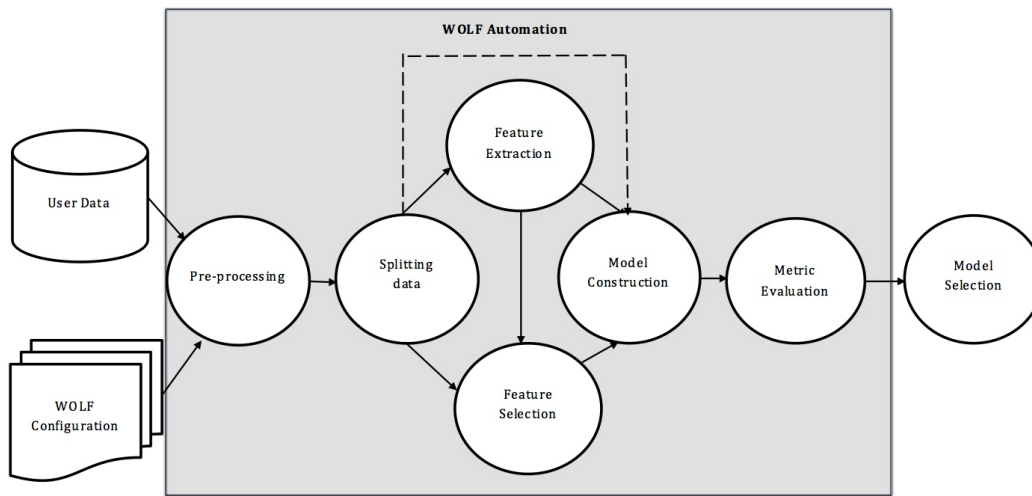
Figure 1: The WOLF framework and its transactions.

4 THE WOLF ARCHITECTURE

WOLF works in transactions to figure out the best pipeline for the user's data. A transaction is a common machine learning task performed to build the best solution for a data problem. Some of those tasks include data preprocessing, cross validation, feature engineering, model construction and evaluation. Each transaction provides choice of various algorithms. The WOLF platform runs all transactions based on the configuration provided by the user, and selects a pipeline consisting of best performing algorithm for each transaction individually. Each transaction in the project is isolated but is dependent on the output generated by its prior task. WOLF's workflow is shown in Figure 1.

A typical WOLF pipeline consists of multiple transactions, not all transactions are mandatory. The dotted line in Figure 1 is to show alternate paths or the transactions that can skipped in a pipeline. Each transaction performs a meaningful task.

The WOLF transactions are dependent on the user input configuration. To keep the consistency across all user configurations, the project follows a protocol which defines the structure and keywords used for creating a configuration file.

Following are the list of transactions in machine learning pipeline adopted in WOLF framework:

4.1 Preprocessing

There are some common data preprocessing operations which are performed on the data file provided by user. This step includes operations like getting rid of NaN (Not a Number) values or replacing them with user specified values. Label encoding of categorical features and scaling of numerical features is also performed in the preprocessing step.

4.2 Splitting Data

The splitting data transaction replicates the task of cross validation. After pre-processing of user input data it is then split into multiple pairs of train and test files. The number of train and test data file combinations generated depends on the values provided by the user for the number of folds and repetitions parameter under split data configuration. The data file is always randomized before splitting which leads to distinct pairs of train and test files.

4.3 Feature Extraction

The data extraction is mostly meant for dimensionality reduction of data set or for creating new features by combining multiple features which may be more meaningful to model construction algorithms. The train and test file combinations created by splitting data transactions are treated as input for data extraction phase. Because not all the data sets requires data extraction or the user may want to take control of the feature extraction step, this transaction has been made optional.

The method used for dimensionality reduction is Principal Component Analysis (PCA).

4.4 Feature Selection

Taking the train and test files pairs generated after data extraction or splitting data transaction (if data extraction is skipped) the operation of feature selection is performed on them. The idea behind this transaction is to get rid of noisy data or the features which are not meaningful to model construction algorithms. The task of feature selection is also optional for the same reasons as feature extraction. After performing feature selection, pairs of train and test files are created but this time the output files consist of meaningful features only.

The algorithm used for this purpose is Support Vector Machine Recursive Feature Elimination (SVM_RFE). Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of

features and weights are assigned to each one of them. Then, the features whose absolute weights are the smallest are pruned from the current set features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

4.5 Model Construction

The input for model construction transaction is train and test file combinations generated by feature selection or feature extraction or the splitting data transaction based on the users configuration if any of the transaction is skipped. This transaction consists of multiple algorithms and each model construction algorithm runs in parallel. The output generated by this transaction consists of result files which contain true values and predictions made for test files. The number of generated files depend on the number of algorithms chosen by the user for experimentation and also the freedom of hyper-parameters as configured by users. So far, the following classifiers have been integrated into the WOLF framework:

(1) AdaBoost [11]
(2) Bernoulli and Gaussian Naive Bayes [9]
(3) Decision Tree [22]
(4) Logistic Regression (LR) [7]
(5) Random Forest (RF) [8]
(6) Linear, Nu and C-Support Vector Machines [6]
(7) Linear and Quadratic Discriminant Analyses (LDA) [12]
(8) Deep Neural Network (DNN) [1]

4.6 Model Evaluation

After performing all the important tasks of machine learning, evaluation of results is undertaken. In this step, metric calculation is performed for all possible combinations formed by prior transactions. WOLF consists of metrics like precision, accuracy, area under curve (AUC), Matthews correlation coefficient (MCC), f1-score and many more. The output of this transaction is used for decision making or choosing the best (based on the chosen metrics) predicting combination of algorithms.

4.7 Model Selection

This is the moment of truth; the user is provided with a result file consisting of metrics calculated for all possible combinations based on the configuration file. The result file also consists of graph representation of all metrics for each configuration combination which may be further pruned by the user. This gives users the leverage of analyzing the result for appropriate choice of the algorithm which performs best according to their metric selection.

5 WOLF DATABASE MANAGEMENT

In WOLF, the configuration of all the transactions as well as the metric evaluation results of each configuration are saved into the database. The idea behind integrating database to the project is to ease the tracking of each run. It also helps to ease the process of result generation after the complete pipeline completion, which is then utilized for model selection. As the configuration of each user differs, using relational database could have caused huge waste of space. To conserve space, a non-relational database, MongoDB is

used, where only the parameters for a transaction that are specified by users are saved.

The use-case of each collection is:

5.1 Split Data

The user's configuration for "Splitting Data" transaction of the project gets saved into this collection. The collections consists of data file information by storing the input file name and also the parameters like number of folds and number of repetitions set by user in the configuration file.

5.2 Files

This collection keeps track of the files name that get created after splitting data transaction. It just consists of train file name and test file name for each pair generated. The idea behind this is to keep track of best performing pair if required.

5.3 Feature Extraction

The configuration passed for "Feature Extraction" transaction by user gets saved into this collection. The information stored in the collection consists of, executable (name of algorithm that get executed) and parameters for the chosen executable.

5.4 Feature Selection

The feature selection collection provides information about the users configuration for "Feature Selection" transaction. It consists of executable (name of algorithm that get executed) information and also consists of parameters values provided for chosen algorithm.

5.5 Algorithm

The information related to the "Model Construction" transaction gets stored into this collection. The stored information consists of executable (name of user selected algorithm) and parameters for the selected algorithm.

5.6 Result

As the name explains this collection contains output generated by "Model Evaluation" transaction. All the metrics value that get calculated in evaluation step like mcc, f1-score, precision, and recall etc. gets stored into the collection. The result collection provides moment of truth; it is used to query the results of all pipeline combinations executed by user and generate the report for model selection.

All the collections in WOLF database schema except Result has one to many relationship with Result collection. As Result collection contains the metric evaluation result so it makes sense to refer each result document to the configuration it is related to. The Result collection along with metric evaluation output, consists of unique ids' for all the other collections like split data, feature extraction and selection, algorithm. The unique IDs' of other collections are the foreign keys that point to the particular configuration used for the generated result.

Table 1: The datasets for binary classification

Dataset Name	No. of Attributes	No. of samples	Pos. / Neg. Samples	Percent Positive Samples
Bank note authentication	5	1372	610 / 762	0.445
Blood Transfusion Service Center	5	748	178 / 570	0.237967914
Climate Model Simulation Crashes	19	540	494 / 46	0.914814815
Connectionist (Sonar, Mines vs. Rocks)	61	208	111 / 97	0.533653846
default of credit card clients	24	30000	6636 / 23364	0.2212
Fertility	10	100	12 / 88	0.12
LSVT Voice Rehabilitation	311	126	42 / 84	0.333333333
Pima Indians Diabetes	8	768	268 / 500	0.348958333
Spambase	58	4601	1813 / 2788	0.394044773
Vertebral Column	7	310	210 / 100	0.677419355
Wholesale customers	8	440	142 / 298	0.322727273

Table 2: Results generated by WOLF for the Spambase Dataset

Algorithm Name	Accuracy	f1 score	mcc	Precision	Recall	roc auc
Random Forest	0.940491943	0.922081025	0.875282549	0.952318939	0.893881254	0.932342067
Decision Tree	0.905932828	0.881324856	0.803675827	0.87639301	0.88665373	0.902562047
AdaBoost Classifier	0.938579417	0.921548713	0.871293486	0.927807641	0.915611007	0.934563288
Deep Neural Network	0.723321129	0.538341427	0.405704617	0.776242927	0.421798091	0.670599051
Bernouli Naive Bayes	0.885960569	0.849226568	0.759761205	0.886503696	0.815392448	0.873621528
Linear Support Vector	0.832480662	0.776085134	0.673030914	0.821482253	0.791323227	0.825286975
Logistic Regression	0.927605137	0.906666538	0.847999572	0.921425047	0.892614492	0.921487069
Linear Discriminant Analysis	0.887568983	0.846374519	0.764242867	0.916240821	0.78682252	0.869953218
Gaussian Naive Bayes	0.821346197	0.808553048	0.676682579	0.700090522	0.957035295	0.845072219

6 EXPERIMENTS

The main focus of carrying out the following experiments is to present main features of WOLF framework to compare performance of state of the art machine learning techniques with optimum hyper-parameters. As WOLF is an automated machine learning platform, so a user will only provide data in .arff format along with the configuration file. The user can tell which algorithm to execute in each transaction and the range of hyper-parameters for each algorithm in the configuration file.

6.1 Binary Classification Tasks

In order to demonstrate applicability of WOLF platform, different families of datasets from UCI repository [16] for binary classification tasks were selected. In order to pass those data sets to WOLF, they were first converted into .arff data format.

Table 1 provides a list of datasets that has been evaluated by WOLF. The number of attributes in Table 1 corresponds to feature points in each dataset. The number of feature points are not too big, that's why feature selection or extraction transactions were not used. Also, it is useful to note the total number of samples and ratio of positive samples. For a dataset with high percentage of one class of samples, a machine learning model which is biased towards most likely label will give high accuracy. So other evaluation metrics like precision will give better insight in such cases.

Table 2 provides evaluation metrics generated by WOLF framework for all algorithms that have been applied to Spambase Dataset.

There is no dependency among algorithms, so, all these algorithms were executed and evaluated in parallel. Evaluation metrics calculated for binary classification are the most popular ones to evaluate algorithm performance. For instance, in some applications it's only desirable to have a high accuracy score, however for critical applications it can also be desirable to have a high precision score. Precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples. It is also worth noting if the machine learning model is predicting randomly or has learned something from training data. For this purpose, (roc_auc) score can give better intuition and model is considered good if the score is close to 1. For the Spambase dataset, Random Forest and Ada Boost classifier performed better as compared to other algorithms in terms of accuracy.

Table 2 provides information about which family of algorithms are superior to others for a given dataset. DNN performance is quite poor for this particular application which is evident from roc_auc score which is close to 0.5. The possible reason could be the size of training dataset. In summary, the evaluation metrics for binary classification provided by WOLF framework may help data scientists to develop better intuition of different machine learning techniques.

Table 3 demonstrates another feature of the WOLF framework, which is to search for optimum hyper-parameters through the grid

Table 3: Results of deep neural networks with various hyper-parameters for the vertebral dataset

Accuracy	Learning Rate	Batch Size	No. Neurons in each layer	roc auc
0.829677419	0.001	200	[100,100,100]	.796761905
0.836129032	0.001	100	[100,100,100,100]	0.797333333
0.410322581	0.01	100	[100,100,100,100]	0.506619048
0.61483871	0.01	100	[100,100,100]	0.510380952
0.831290323	0.001	100	[100,100,100]	0.795595238
0.523870968	0.01	50	[100,100,100]	0.507142857
0.63483871	0.01	200	[100,100,100,100]	0.596380952
0.84	0.001	200	[100,100,100,100]	0.810142857
0.329677419	0.01	50	[100,100,100,100]	0.5
0.809032258	0.001	50	[100,100,100]	0.795666667
0.745806452	0.01	200	[100,100,100]	0.668857143
0.816129032	0.001	50	[100,100,100,100]	0.804309524

Table 4: Accuracy scores for all binary classification datasets

Dataset Name	RF	Linear SVM	DNN	LR	Bernoulli Naive Bayes	LDA	Ada Boost	DT
Bank note auth.	0.9923	0.9889	0.9998	0.9896	0.8419	0.9786	0.996	0.9810
Blood Transfusion	0.6279	0.5323	0.5003	0.5495	0.4993	0.5419	0.6181	0.5852
Climate Sim. Crashes	0.5501	0.7941	0.6581	0.5773	0.5	0.7158	0.7535	0.6527
Sonar, Mines/Rocks	0.8167	0.7692	0.61	0.7507	0.5051	0.7358	0.7954	0.6919
Default of credit card	0.6540	0.5217	0.5	0.4999	0.6731	0.6127	0.6388	0.6081
Fertility	0.5531	0.4960	0.5223	0.4988	0.5	0.4878	0.5375	0.4964
Voice Rehabilitation	0.7831	0.5058	0.6344	0.5529	0.6854	0.7256	0.7916	0.7351
Pima Indians Diabetes	0.7216	0.5597	0.6864	0.7151	0.5035	0.7253	0.7131	0.6412
Spambase	0.9323	0.8252	0.6706	0.9215	0.8736	0.8699	0.9345	0.9025
Vertebral Column	0.8058	0.7261	0.8101	0.8095	0.6438	0.8017	0.7917	0.7592
Wholesale customers	0.9053	0.6939	0.5	0.8765	0.5	0.7748	0.8800	0.8520

search. Except for Deep Neural Networks (DNN), the grid search is simple and efficient to find optimum hyper-parameters.

Table 3 provides details about the performance of DNN on the Vertebral dataset for a range of hyper-parameters provided by the user in the configuration file. It can be deduced from the table that the learning rate plays a vital role in DNN's performance, compared to batch size and depth. This helps data scientists to determine which hyper-parameters plays a vital role in performance and provides a better understanding about DNN for a given problem. This would help them to adopt good implementation practices in the future. More advanced parametric search algorithms, such as the Bayesian optimization and active optimization, will be integrated into the WOLF framework as part of the future work.

Table 4 summarizes the results for all datasets mentioned in Table 1. It can be seen that the WOLF framework is applicable to different families of datasets and automates the process of finding the best machine learning algorithm with optimum hyper-parameters. It can be observed that not a single machine learning model outperform in all datasets. So the user needs an automated platform like WOLF to choose best algorithm for each application. Also, if data scientists develop a new algorithm, it can easily be compared with existing commercial or proprietary algorithms.

6.2 Android Malware Detection

For all experiments, a dataset of real Android applications and real malware requested from the authors of Drebin [3] is considered. The final dataset contains about 40,000 benign applications and 3000 malware samples. This is one of the largest malware datasets that has been used to evaluate a malware detection method on Android [3].

The app features are extracted in two ways: from permission information of given sample (extracted from manifest.xml file) and from system call information. Detailed description of the feature extraction procedure is given below. Figure 3 shows the Area under curve (AUC) values of different machine learning algorithms analyzed for the given feature vectors consisting of permission bits and API calls.

From the experiment results we can see that the Decision Tree, has given an AUC value closest to 1.0.

Feature Extraction The appropriateness of extracted features determines the accuracy of the classification results. The features are extracted in two phases described below.

Feature Extraction using Static Analysis Android applications come in an Android package (.apk) archive. This .apk file is a zip bundle of AndroidManifest.xml, classes.dex and other resources and folders.

Table 5: The datasets for multiclass classification

Dataset Name	No. of Attributes	No. of samples	Classes
Dermatology	35	358	6
Iris	5	150	3
Leaf	16	340	30
Page Blocks Classification	11	5473	5
Wine Quality	12	4898	11

Table 6: Accuracy scores for all multiclass classification datasets

Algorithm Name	Dermatology	Iris	Leaf	Page Blocks Classification	Wine Quality
Random Forest	0.97	0.929	0.745	0.852	0.412
Linear SVM	0.956	0.934	0.578	0.667	0.0801
DNN	0.613	0.571	0.585	0.119	0.050
Gaussian Naive Bayes	0.843	0.933	0.713	0.495	0.219
LR	0.97	0.93	0.385	0.732	0.210
LDA	0.956	0.964	0.7937	0.669	0.253
AdaBoost	0.48	0.905	0.0853	0.554	0.114
Decision Tree	0.97	0.929	0.745	0.852	0.412

For extracting these features, the .apk files need to be reverse engineered. This is done using the apktool. The AndroidManifest.xml file contains a lot of features that can be used for static analysis. One of the main features is the permissions requested by the application. In order to extract these permissions, regular xml parsers cannot be used since Android has its own proprietary binary xml format. A special xml parser developed by the Drebin [3] authors is used for extracting permission features from the AndroidManifest.xml file of the application.

Feature Vectors Let R be a vector containing a set of all Android permissions requested by an application. For every ith application in the Android apps dataset, we generate a binary sequence $R = \{r_1, r_2, r_3, ..., r_j\}$, where:

$$r_j = \begin{cases} 1, & \text{if } j_{th} \text{ permission exists .} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The permissions identified are stored as a binary sequence of 0 or 1 in a comma separated form in the final .arff file submitted to the WOLF Machine Learning Platform. This sequence typically contains comma separated permission bits which denote 1 if the corresponding permission is present or 0 if it is absent. In addition, we consider a variable C denoting the class of the application, where $C = 0$ for a benign application and $C = 1$ for malware app.

Figure 2 explains the feature extraction method from [3]. We list the steps to extract features as fellows:

(1) Download the Drebin dataset of all malware and benign files.
(2) Reverse engineer the android applications in the dataset. This reverse engineering is achieved using the APK tool (already done by Drebin authors).
(3) Drebin uses an xml parser is used to extract the permission request features from the AndroidManifest.xml, along with
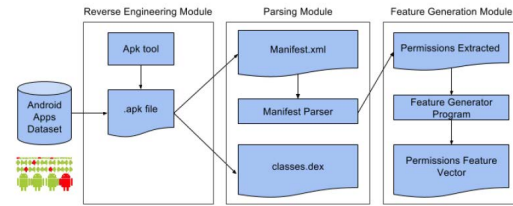
**Figure 2: Flow chart of permission feature extraction [3]**

other features such as hardware resources and filtered intents requested.

(4) The permissions requested by each Android application are then sent to the Feature Vector generator program where the application's feature vector is generated using the method discussed above.

(5) We finally build a matrix that maps the permissions requested by each application in the dataset to the corresponding application. This file also contains the class (benign / malware) information of the application and stores it in an ARFF [3] file format.

6.3 Multiclass Classification Tasks

For Mutliclass Classification, few datasets were acquired from UCI repository [16] and the details about each data set are provided in Table 5. Results generated by WOLF platform are shown in Table 6.

Again, it can be observed selection of suitable algorithm is an important step. For instance for Dermatology dataset, performance of AdaBoost classifier is quite poor as compare to other algorithms. Again, in Leaf dataset it's performance degrades severely.
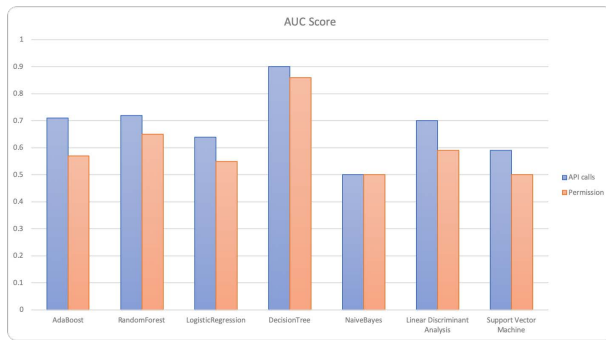
Figure 3: Performance of Android malware detection

In summary, the WOLF framework is able to find optimum algorithms for all datasets except the Wine Quality set. This indicates that it can provide a good insight about the dataset to the data scientists.

7 HYPOTHESIS

From the results described in previous section, the following hypothesis can be postulated:

(1) Comparison between Shallow and Deep Models: In this experiment, we try to compare performance of deep models with standard shallow machine learning models to develop better understanding about the applicability of deep models. It is evident that deep models do not perform better when data size is small. Also, if ratio of positive to negative samples is not closer to 0.5, DNN models are more likely to over fit.

(2) Fine tuning Hyper-Parameters is essential: This is the essential part of getting good performance from a given machine learning model. It is evident from Table 3 that model's performance drastically suffer with wrong set of hyper-parameters.

(3) No standard Machine Learning Algorithm for all applications: It can be observed from Results shown in Table 4, that no single machine learning algorithm can be applied to all datasets. So there will always be a need to find which algorithm is better suited for a particular application.

8 FUTURE WORK

In order to make deep learning more effective for smaller datasets, fine tunning of pre-trained models also known as transfer learning [23] needs to be integrated into WOLF. Moreover, grid search for optimum hyper-parameters for DNN requires lots of resources. To address this issue, better parameter optimization techniques like Bayesian optimization [21] or latest techniques like active optimization [13] and learning to learn [2] need to be integrated into the WOLF framework. Efforts are being made to make the WOLF framework publicly available online, where users can upload the project directly to get the results, or contribute to the project.

REFERENCES

[1] I. Sutskever A. Krizhevsky and G. Hinton. 2012.
[2] Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. 2016. Learning to Learn by Gradient Descent by Gradient Descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 3988–3996.
[3] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck. 2014. Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium (NDSS)*.
[4] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. 2008. The Konstanz Information Miner (KNIME). In *Data Analysis, Machine Learning and Applications. Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin, Heidelberg.*
[5] H. Hoos K. Leyton-Brown C. Thornton, F. Hutter. 2013. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining.* 847–855.
[6] Chih-Chung Chang and Chin-Jen Lin. 2011.
[7] C.Y.Peng, J.Lee, K. Land, and G.M. Ingersoll. 2002.
[8] Misha Denil, David Matheson, and Nando de Frietas. 2002.
[9] S. Eyheramendy, D.D. Lewis, and D. Madigan. 2003. On the Naive Bayes model for text categorization. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (2003), pp. 332-339.*
[10] Pedregosa F., Varoquaux G., Gramfort A., and Grisel Duchesnay E. Michel V., Thirion. 2011. Scikit-learn: Machine learning in Python. In *The Journal of Machine Learning Research.* 2825–2830.
[11] Y. Freund and Schapire R. 1997. A decision theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Sciences 119–139.*
[12] Benyamin Ghojogh and Mark Crowley. 2019. Linear and Quadratic Discriminant Analysis: Tutorial. arXiv:1906.02590 [stat.ML]
[13] Kirthevasan Kandasamy, Jeff Schneider, and Barnabas Poczos. 2017. Query Efficient Posterior Estimation in Scientific Experiments via Bayesian Active Learning. In *Artificial Intelligence Journal (AIJ).*
[14] Tim Kraska, Ameet Talwalkar, Rean Griffith John Duchi, Michael Franklin, and Michael Jordan. 2013. MLbase: A Distributed Machine-learning System. In *6th Biennial Conference on Innovative Data Systems Research (CIDR).*
[15] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. 2006. Scientific Workflow Management and the Kepler System: Research Articles. *Concurr. Comput.: Pract. Exper.* 18, 10 (Aug. 2006), 1039–1065.
[16] Lichman M. 2013. *UCI Machine Learning Repository.* Retrieved February 28, 2019 from http://archive.ics.uci.edu/ml
[17] Randal S. Olson and Jason H. Moore. 2016. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. In *The Journal of Machine Learning Research (JMLR).* 66–74.
[18] Jim Pivarski, Collin Bennett, and Robert L. Grossman. 2016. Deploying Analytics with the Portable Format for Analytics (PFA). In *Proceedings of the 122nd ACM SIGKDD international conference on Knowledge discovery and data mining.* 579–588.
[19] Sebastian Raschka. 2015. *Python machine learning : unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics* (1st. ed.). Packt Publishing Ltd, irmingham B3 2PB, UK.
[20] T. Schaul, J. Bayer, D. Wierstra, F. Sehnke T. Rückstieß Y. Sun, M. Felder, and J. Schmidhuber. 2010. PyBrain. In *The Journal of Machine Learning Research (JMLR).*
[21] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems.*
[22] Yan-Yan Song and Ying Lu. 2015. Decision tree methods: applications for classification and prediction. In *Shanghai archives of psychiatry vol. 27,2 (2015): 130-5.*
[23] Qiang Yang and Sinno Jialin Pan. 2010. A Survey on Transfer Learning. In *IEEE Transactions on Knowledge & Data Engineering.* 1345–1359.