Group 26 MLBenchmark: An Android Application to Benchmark Machine Learning Algorithms

Ankush Kanungo
1211390203
Arizona State University
akanung1@asu.edu

Divyanshu Bandil
1211234762
Arizona State University
dbandil@asu.edu

Meha Shah
1211143970
Arizona State University
mshah17@asu.edu

Siddhant Prakash
1211092724
Arizona State University
sprakas9@asu.edu

ABSTRACT

UPDATED—January 8, 2018. Machine learning has become very popular in recent years with its application being recognized by the technology industry as a commodity. It finds application in day-to-day activity from face recognition to unlock mobile phones to sentiment analysis of our daily tweets. But machine learning on mobile devices suffers from the age old challenges of resource constraint computing especially because of its high computation cost. Thus, to aid users in this domain we present a machine learning benchmarking android application in this paper. The application uses a standard dataset to run experiments using four different machine learning algorithm. We can train any of the four model on a specified percentage of dataset provided and use the remaining percentage of data to test the performance of the model on device. The training is offloaded to cloud AWS server which implements a REST API. We report the performance of the algorithms in a separate view as well as log the outcomes of each experiment in a file stored on the external storage. The code for the project can be found here https://github.com/dev-sidd-16/ML-Benchmark

Author Keywords

Android programming, Logistic Regression, Naive Bayes, K-Nearest Neighbor, Support Vector Machine, Machine Learning, Benchmarking, Classification, Cloud Computing

INTRODUCTION

In this paper we present an Android application for benchmarking machine learning algorithms. MLBenchmark provides the user with four different machine learning algorithms. A user can train these algorithm on a cloud server and test each model on the device. The application comes with a dataset pre-installed in it over which all the experiments are run. The user can perform different experiments using our application and store result of each experiment in a log file. This can help the user to test the performance of their device towards different ML algorithms as well as aid them to compare the various machine learning algorithms out in the market. As a result, the user can find out which algorithm works best for a particular task as well as whether it is reasonable or even feasible to use it on given device.

The high cost of computation of machine learning algorithms as well as their hunger for resources is what calls for the need of such an application. In a typical machine learning experiment, we try to learn a model from large number of data samples. To learn these models researchers have come up with various different algorithms over the years. The learning part of each of these experiment is to estimate the parameters of the particular model we want to learn. Often times the models involves complex calculations well beyond simple addition and multiplication. Besides, the large amount of data required by the algorithms for learning a good representation of the data becomes a challenge in itself. Thus, the processing large amount of complex calculations on resource constraint environment such as a smartphone becomes very difficult.

In this work we try to get an estimate of the computational power we have on today's mobile devices. With advancement in mobile systems, we have now huge computational power, well beyond what we had even on desktop systems 10 years back. With our experiments that we performed, we found out that the algorithms do not do well when the training was performed on device. This is attributed to the large amount of data we need to train a model. Although, our dataset was small with only 9 dimensions, the training time it took on few hundred samples was in order of seconds. If we extrapolate these results in comparison to large datasets with hundreds of dimensions and millions of data points, it can be seen that the device will take minutes or even hours to perform training itself. Thus, we conclude that the device cannot be allowed to do the training phase, which we implement on a cloud server. Instead, we perform the testing phase on device and report it's performance as our result of the experiment, along with the time it takes to train the model on the server.

The rest of the report is organized as follows. We briefly go over the theory of machine learning algorithms we used in the backgrounds section. Explaining the role of datasets and classifiers in the experiments, we highlight how they increase the complexity of the task to be performed. We also brief over

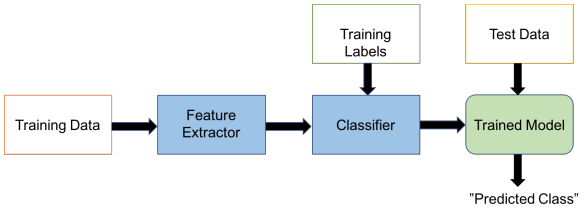


Figure 1: A standard machine learning classification pipeline.

the different performance metrics we report in our test view. Then we move on to the application structure. In this section we explain the different functionalities of our application and the work of each views. We guide the user about how to use our application and how one can perform multiple experiments in a short span of time with our intuitive interface. Next we go into the implementation details of our application. We list all the different components that we used to build this interactive application and provide an in-depth analysis of the decisions made and challenges faced while building the application. Finally, we conclude this report with a demo of the project and briefly go over the future directions where this application could be improved.

MACHINE LEARNING BACKGROUND

A supervised machine learning framework takes labelled data as input and generates discrete output corresponding to each data sample. A typical example is a classification task. In a classification task, we classify each data sample into different classes. We generally group classification task into two groups, viz. 2-class classification and multi-class classification As the name suggest, 2-class classification involves classifying the data into 2 classes while we can have any number of classes for multi-class problems. To demonstrate our use case, we have selected a 2-class classification problem and built our application over it.

We show the data flow in a standard machine learning classification system in Figure 1. The dataset is first divided into a training and test set. Feature extraction is normally done on raw training data to extract features for various purposes like reducing dimensionality of dataset for faster computation. After extracting the features, a classifier is learnt on the dataset using the training samples and the training label. After each learning an error is calculated and using the error the parameters or "weights" of the model are updated to reduce the training error. Thus, after the training is complete we have a trained model which represents the data it has been trained on. Now in the testing phase, the test data is fed into the system and the model predicts a class for each of the data samples. Each prediction results in a class output, which may or may not be correct. The error incurred on the test dataset is called test error.

The quality of training a good model depends on a number of factors, of which the size of training set is a major one. If we have less training data, the model will not be able to learn the data representation correctly leading to learning of too simplistic model, a phenomenon called **"under-fitting"**.

On the other hand, if we train a model to learn too complex a representation over the training set, we observe that the training error keeps on decreasing, while the test error, initially decreasing, starts increasing after attaining a saddle point. The resultant phenomena is termed as **"over-fitting"**. Thus we see that having a large training data is desirable, but when it comes to mobile phones, we need to take the resource constraints into consideration too.

Now we will describe the dataset we used as well as give a background on the classifiers we have implemented. We will also discuss the performance metrics we have used to express the performance of our test results in brief.

Dataset

We have used the **"Breast Cancer Wisconsin (Original) Data Set"** from the UCI Machine Learning repository [1][2][3][4]. The dataset consist of 699 data points in total. It is a 2-class category problem with class 1 "Benign" consisting of 458 samples (65.5%) while class 2 "Malignant" consisting of 241 samples (34.5%). The dataset consist of 10 attribute with the first one being the sample ID. We discard the sample ID as an attribute since it does not really give us any information about the data. Thus, we use a 9 attribute dataset to learn our model for prediction 2 given class.

Classifiers

We have used four different classifiers in our application to experiment on. We have used the classifier implementation of "Weka Library" in our application. The four different algorithms we have used are:

Logistic Regression

The logistic regression classifier uses the logistic (sigmoid) function to build a model for classification. This is a probabilistic approach which computes the probability of a data point lying in a given class, $p(y|x)$. The conditional distribution is a Bernoulli given by,

$$p(y|x) = \mu(x)^y(1-\mu(x))^{(1-y)} \tag{1}$$

where μ is the logistic function and is given by,

$$\mu = \frac{1}{1+e^{-w^T x}} \tag{2}$$

The training of a logistic regression classifier involves estimating the parameters **w** which will maximize the conditional likelihood of the training data.

The final objective function is given by the following equation,

$$\begin{aligned} l(w) &= \text{argmax}_w \sum_{i=1}^{N} ln(P(x_i|y_i;w)) \\ &= \sum_{i=1}^{N}(y_i-1)w^T x_i - \sum_{i=1}^{N} ln(1+e^{-w^T x_i}) \end{aligned} \tag{3}$$

Thus, to train the model we learn the weights **w** by optimizing the given objective function using the gradient ascent algorithm.

Naive Bayes

Naive Bayes is another probabilistic approach in which we use Bayes rule for computation of posterior probability $P(Y|X)$ to classify data based on the input X to one of the classes given by Y [5]. The difference between Bayes and Naive Bayes is the assumption of **conditional independence** which states that, X is conditionally independent of Y given Z, if the probability distribution governing X is independent of the value of Y, given the value of Z. Mathematically, conditional independence is expressed as given in Equation 4,

$$P(X = i|Y = j, Z = k) = P(X = i|Z = k) \qquad \forall(i, j, k) \tag{4}$$

Thus, in Naive Bayes classification, the interpretation of conditional independence assumption is that each feature becomes independent of the other given a class. Thus, the joint likelihood of a feature given a class becomes,

$$P(X_1, ..., X_2|Y) = \Pi_i P(X_i|Y) \tag{5}$$

So the decision rule for Naive Bayes becomes,

$$\begin{aligned} y^* &= \text{argmax}_y P(y)P(x_1, ..., x_n|y) \\ &= \text{argmax}_y P(y)\Pi_i P(x_i|y) \end{aligned} \tag{6}$$

Training of naive bayes classifier involves the calculation of class likelihood to find out the posterior probabilities for each class. The testing phase predicts the outcome of each data point by finding out the most likely class in which the data should fall based on the class posterior probabilities.

k-Nearest Neighbor

k-NN is an instance based learning algorithm . Each data point is represented as a vertex point on a graph and the edge weights of the graph represents the similarity between the nodes. If the weights are less, the nodes are more similar and vice versa. For symmetric k-nearest neighbor, the graph is an unweighted graph. In this algorithm we connect each vertex to its k-nearest neighbors, i.e the k most similar data points in its vicinity are grouped together. The similarity is calculated using a distance function between two nodes. The lower the distance function, the similar the nodes are. The most commonly used distance function is L_2-norm which is the default metric used in the library implementation. We have used the same metric in our implementation of the k-NN classifier.

Support Vector Machines

Support Vector Machine [6] is a supervised learning technique in which the goal is to come up with a decision boundary to classify all data points in the given classes. In the training process the goal is to learn the decision boundary which will minimize the error in classification of data points.

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Table 1: Confusion Matrix

The algorithm works by estimating the support vectors, which are the data points closest to the decision hyperplane, separating the classes. The error is calculated by summing the distance of the mis-classified examples from the decision hyperplane. Support vector machine finds hyperplane for classifying two classes. Thus, the algorithm works well for our classification task.

The decision boundary in SVM is given by the equation,

$$w^T x + b = 0 \tag{7}$$

in which w and b are the parameters to be learnt. In SVM we maximize the margin c for the classification task, between the decision hyperplane and the support vectors. Thus, the joint formulation can be defined as follows.

$$(w^T x_i + b)y_i > c$$

The final maximum margin optimization for the classification problem is given by the following equation.

$$\begin{aligned} \max_{w,b} \quad & \frac{c}{||w||} \\ \text{s.t} \quad & y_i(w^T x_i + b) \geq c, \qquad \forall i \end{aligned} \tag{8}$$

This is the Support Vector Machines objective, which we try to solve using quadratic programming, and learn the weights **w** and the bias **b** in the training phase. Thus the model becomes the decision hyperplane given by equation 7.

Performance Metrics

Confusion matrix, shown in Table 1, is one of the methods in machine learning used to evaluate performance of a (2-class) classification problem. As shown in the table, the columns are predicted class and the rows are actual class. Over a dataset of finite samples, the count of correctly classified negative samples is termed as True Negative (TN), while the count of incorrectly classified negative samples is termed as False Positive (FP). Similarly, the count of incorrectly classified positive sample is termed as False Negative (FN), while the count of correctly classified positive sample is termed as True Positive (TP).

For any classification task, predictive accuracy is defined as the total number of correctly classified samples over total number of samples. Mathematically, it is given by,

$$Accuracy = \frac{TP + TN}{TN + FP + FN + TP}$$

In machine learning, we evaluate the performance of a classifier by its error rate which is given by,

$$ErrorRate = 1 - Accuracy$$

We have reported the True Accept Rate (TAR), True Reject Rate (TRR), False Accept Rate (FAR) and the False Reject Rate (FRR) as our evaluation metric. These metrics are given as follows,

$$TAR = \frac{\text{Correctly predicted positive}}{\text{Total positive}} = \frac{TP}{TP+FN}$$

$$TRR = \frac{\text{Correctly predicted negative}}{\text{Total negative}} = \frac{TN}{TN+FP}$$

$$FAR = \frac{\text{Incorrectly predicted positive}}{\text{Total negative}} = \frac{FP}{TN+FP}$$

$$FRR = \frac{\text{Incorrectly predicted negative}}{\text{Total positive}} = \frac{FN}{TP+FN}$$

We can see that,

$$TAR + FRR = 1$$

and,

$$TRR + FAR = 1$$

We also report the HTER metric which is given as,

$$HTER = \frac{FAR+FRR}{2}$$

The above metrics give an idea about the accuracy of prediction which reflects how well the learnt model represent the data distribution. In addition to these metrics we also report the time taken by each algorithm for both training as well as testing phase. Since the training is done over cloud, we report the complete round trip time it takes for uploading the training dataset as well as downloading the learnt model on the uploaded dataset as our training time.

APPLICATION STRUCTURE

We have divided the application into 3 major sections, viz. Main Screen, Cloud Interface and the Testing View. Here we describe the basic UI to navigate through the application and explain different component of each section.

Main Screen

The first screen that we get when we run the application is the main screen. This is where we decide all our parameters for running an experiment. As we can see in Figure 2c, the main screen consists of many different views. Let us go through each one sequentially from top.

The first view from the top is the data split percentage input. This is the place where we need to specify the percentage of data that should be used for **training** the classifiers. We should note, that this is independent of the classifier we use. Thus, it comes before selection of the classifiers. The values that we can take should be in the range of **0.01 - 0.99** and must be a floating point value. If we specify the value as 0.75, this means that 75% of the total samples in the dataset will go for the training and the rest 25% will be used for testing. The default value for this parameter is 0.5.

Next we have the classifier selection drop down menu. It is a straight-forward drop down menu with which we can select either of the 4 algorithms that we have implemented, viz. Logistic Regression, Naive Bayes Classifier, k-Nearest Neighbor and Support Vector Machines. The default algorithm selected at the start of the application is Logistic Regression. Next is the create dataset split button, which splits the dataset into training set and testing set. When the button is clicked it creates and stores the training set and testing set files on the external storage. The files are retrieved when we need to perform the respective tasks as we will see later.

The above views, i.e. **the data split percentage and the classifier selection are mandatory** to specify for any experiments. Once we specify the data, we need to click on the create dataset split button to ensure that our training file and testing file is correctly split. To conduct and experiment, these two values must be input. Although they have default values to handle cases when input is not given, the best performance with correct result will be achieved when these two parameters are specified.

The parameters of the classifiers comes next. We have specified a cross-validation parameter which is true for all classifiers. Cross-Validation is basically an integer number. We have performed n-fold cross validation in our training and the number specified here is the n for the cross-validation. If we select k-Nearest neighbor classifier, we can see a new parameter will become visible .Here we specify the number of nearest neighbor that should be considered for k-NN classification. Similarly, if we select Support Vector Machines, we will be able to see another parameter. This time it is a drop down menu which takes the kernel to be used for SVM classification as input. The option of kernels currently available in our application are linear, polynomial (degree=2), polynomial (degree=3) and radial basis function (RBF) kernels. The default value of these parameters are 5 for cross-validation, 5 for number of nearest neighbor and linear kernel for SVM.

We also have a training model available view which specifies whether a trained model is already available in the memory with the specific configuration that we have specified. The authors will like to point out here that we have specified each experiment run as with a fresh start by deleting the previous models trained. Thus, the view shows “Yes" only after we train a model and it is available for the application to start the testing phase. We have a radio button which when selected, performs multi-threaded on-device training. This is a feature we have been trying to implement and more details can be found about it in the future works section.

Next we come across the two most important button of the main view. The “Start Train" button when clicked starts the training phase for the application. Assuming multi-thread processing is unchecked and we are training on cloud, on clicking the train button the training file that was created on splitting the dataset will be sent to the cloud server for training. The cloud server will train the model as specified in the parameters

and send back the model file generated. The application will download the trained model and store it in the external storage directory. Now the application is ready for testing. On clicking "Start Testing", the application loads the testing file and the model generated and evaluates the model over the test set. Before evaluating the application check for the availability of the particular model file asked for with the specified parameters. If the file is not present it returns a error message to the user that the model file is not present and the user should train the model before testing. Thus, if the test button is clicked before the train button, the application handles the case automatically.

Cloud Interface

On the server side, we have implemented the RESTful Web service by using Spring Boot in combination with Spring Web MVC. RESTful web services works best on the Web. In the REST architecture, data and functionality are considered to be the resources which can be accessed using Uniform Resource Identifiers (URIs) which links on the client side. The resources are acted upon by using a set of simple, well-defined operations. The REST architecture is the basis of the client/server architecture and is designed to use a stateless communication protocol, HTTP in our case where the client and server exchange resources by using a standardized interface and protocol.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

- Resource identification through URI: A RESTful web service exposes a set of resources identified by URIs that identify the targets of the interaction with its clients.
- Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations - PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.
- Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others. We are using the JSON format in our application.
- Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained.

Secondly, we have used Amazon EC2 server to host our RESTful web service. Following reasons encouraged us to select EC2 server for hosting our application:

1. Security: Data uploaded by the application user is our first priority and the network architecture and data center built by the Amazon EC2 team provides security-sensitive robust networking functionality for our resource.
2. Scalability: Amazon EC2 also provides resizable compute capacity in the cloud. Therefore, it is easy to train large size big data sets on the server.
3. Reliability: Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned.
4. Complete Control: We have complete access of the server including the root access just like any local machine.

Testing View

The testing view is a simple view which reports the outcome of a particular experiment. It consist of different section where different performance metrics are reported. In the heading the test view specifies the algorithm on which the experiment is run. Next comes the split dataset percentage that has been used for training the model of which the results is being shown.

Then we show the outcome of the experiments in a sequential manner. First comes the performance metrics in the following order: True Reject Rate, True Accept Rate, False Reject Rate, False Accept Rate and HTER. Next we specify the time elapsed in the training phase to generate and download the model followed by the time elapsed for testing the data on device.

The last two buttons are the navigation buttons from this screen. The "Back" button will lead back to the main screen and the current experiment run configuration will be cleaned. The "View Logfile" button will take the user to the next view where we can see the log file that is created after each experiment run. We can see all the experiments performance measure in the file and compare the run results as we please. We can navigate back and forth between the test screen and the view log file screen without loss of any data.

IMPLEMENTATION DETAILS

In this section we will go into our implementation details, design choices and the reason behind the choices we made to come up with this application.

General Android Settings

The application was implemented using Android Studio 2.3.3. The minimum SDK version supporting the app is **Android 17** and the target SDK version on which the app was built is **Android 26**. We have tested the application on 3 Android smartphone and 1 emulator, with the following configurations.

1. OnePlus 3T - Android 8.0.0 API26
2. Nexus 5 - Android 5.1.1 API22
3. Nexus 4 - Android 5.1.1 API22
4. Nexus 5 - Android 5.1.1 API22 - Emulator

Classifiers

We used "Weka Library" [7] for the implementation of the classifiers. The version of weka we used is "weka-stable-3.8.0". Although we obtained "weka-stable-3.8.1" library jar file, we didn't use it because we found the library was not stable and gave us compilation issue when we tried it. We used the library to implement both training on the server as well as testing on device. We imported the library as a jar file and added it to our project. The exact steps we followed to do the same can be found here [8].

Now let us get into the application structure. Before building the app we discussed and decided we will have two screens, one for main activity where we will have all the parameters and the other for showing the test result. The training part was needed to be built on a cloud server and since one of our team members had a Amazon Web Services account, we went ahead with implementing an EC2 server on it. There was some discussion over how to send the training parameters, and how to input it. We would have liked to add different screens for different classifiers because each one would have required a different set of parameters. But finally we settled on a single parameter screen, with all the parameters listed on the same. That became our main activity screen.

Main Activity

We first implemented the application view in the "activity_main.xml" file. The dataset split input was created using "EditText" widget. For the choice of classifiers we wanted to add a drop down button. Thus, we used the "Spinner" widget which lets us handle drop-down menu quite easily. The next hurdle we faced was when to update the dataset split. Most of the time when we input edit text, we change the value immediately by implementing a "setOnEditorActionListener". But we wanted to give an exclusive button to specify when the dataset is split and the split train and test dataset file is created. Thus, next we added a button to include the above functionality.

The next step was to take the input split percentage and split the data file. To do so, we first downloaded the dataset file from the UCI repository **??**. The next decision we needed to make is how to access the dataset. We decided not to download the dataset from the repository every time and include the dataset as a resource with our application. Since, the dataset we choose was roughly around 20KB, it was small enough to be included with the application. Besides, we also needed to convert the raw data file that we downloaded into 'ARFF' format file. We changed the dataset file accordingly and added the file to our assets folder.

But we needed to store all other files in external storage since the training, testing and model files could become very large. So we have added the functionality that an application folder will be created in "sdcard/Android/data/" with the name of **"MLBenchmark"** and all the files will be stored in it. When the "Split Dataset" button is clicked, the app checks if the application folder exists or not. If it does not exists, ti creates the folder and copies the dataset file from assets to the external storage. Next, the "getDataFromResource" function is called, which takes the split percentage as the input, creates and stores the dataset split files in the application folder.

Next we added the views for the parameters with the cross-validation parameter and k-nn parameter as edit text, while the SVM kernels were implemented as a drop-down list. We have added the functionality that only when the parameters are required will they appear. For example, when we select naive bayes or logistic regression classifier, we do not need the k-NN and SVM parameters. Thus, we make all parameters that are not required invisible by setting the visibility of the corresponding view to "INVISIBLE". When the classifier is selected, say SVM, the corresponding parameter's view becomes visible, in this case kernels.

We have also added a view to check if the model of the selected classifier is present in the application folder location, it will indicate "Yes" else it will indicate "No". Since we did not want a new experiment to use the trained model of a previous experiment we delete the previous model as soon as the new model is selected from the drop-down menu if it exists. Thus, the model available check returns true only after training is performed and the model is downloaded.

Next we implemented the training and the testing buttons. On clicking the training button we read the training file from the application directory and upload it to the server. After we receive OK from the server confirming the upload, we then hit the server for training with the train parameters appended to the end point. We have designed the end point of the server to inform the server of the classifier as well as the parameters as listed,

1. Logistic regression: "Server/LR_crossvalidationfolds"
2. Naive Bayes: "Server/NB_crossvalidationfolds"
3. k-NN Classifier: "Server/KNN_K_crossvalidationfolds"
4. SVM CLassifier: "Server/SVM_kernel_crossvalidationfolds"

where "Server/ : AWS_address:8080/".

Thus, the values are extracted on the server side, corresponding training is done and the model file is sent as the output of the hit request. Meanwhile, on the client side, the device keeps checking if the file model file is sent back or not. After, receiving the file, it stores in the application directory as "<classifier>.model". This completes our training process on the client side. The training implementation on the server side is covered in the cloud interface implementation description in the subsection.

Coming to the testing button, when we click it, the train and test file is read from the application directory and loaded as individual instances. The model is checked for availability based on the classifier that is selected. Once the model file is found, we load it and build the evaluation model on the training instance. Finally we test the test instance on the evaluation model built. Next we bundle all the parameters, viz., the timestamp, experiment parameters, data split percentage and the time elapsed in each phase along with the evaluation model and send it to the testing view, attaching it to a new Intent as a Serializable blob. Thus, this completes the test phase and we are ready to view the result in the testing view.

Cloud Interface

The server side of the project is responsible to perform the data processing task i.e. train the selected classification model using training data and send back the trained model to the client (android mobile phone).

We have divided the implementation of the cloud server into three parts:

RESTful web service

The REST web service will typically connect the client and server machines to exchange data resources and information given by the application user needed to perform the model training by using the HTTP protocol. The web service will perform two main operations:

1. It will accept HTTP POST request from the client machine to fetch the training data file generated on the client side and as soon as the upload is complete, it will send a success response ("File uploaded") back to the client machine.

2. It will accept the HTTP GET request from the client machine to fetch the type of classifier that the application user wants to train on the fetched training data and sends back the trained model file in response, to the client machine.

After extracting the training data set and the type of classifier, the web service will send the information to the training class file.

Classification Model Training

We have implemented our four classification models, Logistic Regression, Naive Bayes Classifier, k-Nearest Neighbor and Support Vector Machines on the server side.

To implement these algorithms, we have used the machine learning libraries provided by the WEKA suite. These libraries will take the '.arff' training file as the input, train the respective machine learning algorithm on the input data file and creates the '.model' file in the output. We can use this file directly on the test data to predict the data classes.

Cloud server setup

We have used Amazon EC2 server to host the web service. To setup the server we performed the following steps:

1. Sign Up for AWS: We created the AWS account.

2. Create IAM user: We created the IAM users (for the team members) in the AWS account and added them to the IAM group with administrative permissions. By performing this step, we can access AWS using a special URL and the credentials for the IAM user.

3. Create a key-pair: AWS uses public-key cryptography to secure the login information for our instance. A Linux instance uses a key pair to log in to the instance securely. We specify the name of the key pair during the instance launch and then provide the private key while logging in to the instance using SSH.

4. Add security groups: Security groups act as a firewall for the EC2 server instance controlling both inbound and outbound traffic at the instance level. We added the rules to a security group that enable us to connect to our instance from any IP address using SSH. We can also added rules that allowed inbound and outbound HTTP and HTTPS access from any device.

Testing View

In the testing view, we report all the performance metrics and the time elapsed for a particular experiment. We deserialize all the bundle parameters we get from the Main Activity. Based on the values that we get we create respective variables for the same. Since we receive tested evaluation model of the experiment, we need to extract out each metric from it. We do so using the listed functions from "weka.evaluation" class.

1. True Reject rate: eval.trueNegativeRate(0)

2. True Accept rate: eval.truePositiveRate(0)

3. False Reject rate: eval.falseNegativeRate(0)

4. False Accept rate: eval.falsePositiveRate(0)

The parameter '0' is the class label for the positive class. We multiply the number by 100 and report the percentage on the screen using TextView widget.

Log File View

We have implemented a scroll view activity to read the log file and display it on the application.

CONCLUSION & FUTURE WORK

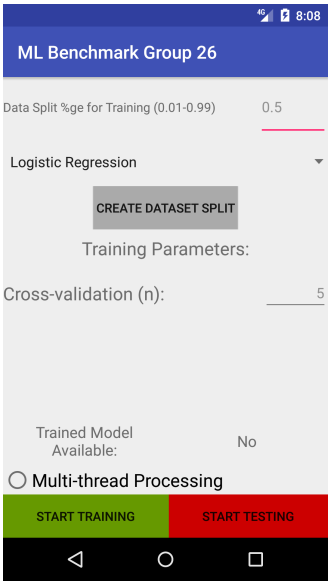
The primary goal of this project is to introduce an ongoing work for benchmarking Machine Learning algorithms. We have exhibited a comprehensive analysis of the performance of various standard Machine Learning algorithms for the data set given by the user, which may be used as a baseline for evaluating and comparing newly developed Machine Learning algorithms. Moreover, our project will also help in assessing the diversity of existing benchmark datasets to identify shortcomings to be addressed by the subsequent addition of further benchmarks in a future release.

For the future work, we have identified that the domain of the Machine Learning benchmarking tool should be expanded further by providing more number of machine learning algorithms for the comparison. Also, we are planning to provide individual application screen with respect to each Machine Learning algorithm where user can also customize the selected Machine Learning algorithm by giving different values of the internal parameters like maximum number of passes over the data, shuffle type, type of activation function for neural networks, learning rate, etc used in training the model. Lastly, we are also planning to make our application on GPU so that we can save the cost of external servers and perform the training on the mobile itself.

We expect this future work to lead to a more comprehensive benchmark application that will guide the users in discovering the strengths and weaknesses of various Machine Learning algorithms by giving them the honest comparisons and faster results. The task list and division of tasks amongst the group members are listed in Table 2.

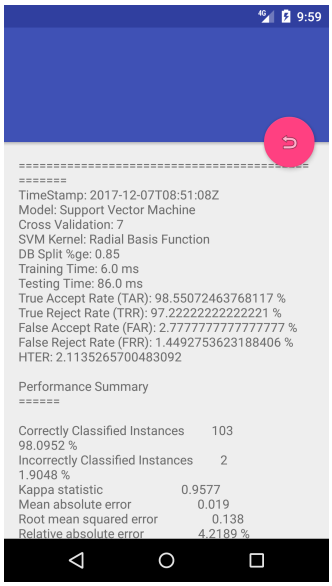
ACKNOWLEDGMENTS

We would like to thank Professor Ayan Banerjee and the TA Junghyo Lee for providing their valuable guidance. We appreciate their considerate efforts which helped us in completing the project successfully.



(a) Main Screen

(b) Test Screen



(c) Log Screen

Figure 2: The different screens in MLBenchmark Application. (a)The main screen that we see at the start of application. We can specify the parameters for experiments on this screen. (b)The test screen we get after the testing is completed. The test results are shown along with the run configuration of the experiment. (c) The log screen where logs of all the experiments performed can be viewed.

REFERENCES

1. O. L. Mangasarian and W. H. Wolberg: "Cancer diagnosis via linear programming", SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.
2. William H. Wolberg and O.L. Mangasarian: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193-9196.
3. O. L. Mangasarian, R. Setiono, and W.H. Wolberg: "Pattern recognition via linear programming: Theory and application to medical diagnosis", in: "Large-scale numerical optimization", Thomas F. Coleman and Yuying Li, editors, SIAM Publications, Philadelphia 1990, pp 22-30.
4. K. P. Bennett & O. L. Mangasarian: "Robust linear programming discrimination of two linearly inseparable sets", Optimization Methods and Software 1, 1992, 23-34 (Gordon Breach Science Publishers).
5. “Naive Bayes classifier." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 7 December 2017.

Table 2: Task Lists; Ankush Kanungo: AK; Divyanshu Bandil: DB; Meha Shah: MS; Siddhant Prakash: SP

S. No	Activities	Name
1	Weka Integration on Server side	DB
2	Server-side implementation of Machine Learning Algorithms	DB
3	Server-side coding for Handling System parameters for different models/algorithms	DB
4	Serializing the parameters of trained model into a file	DB
5	Sending the serialized model file to device	DB
6	Deployment on AWS server	AK
7	Preprocessing of breast cancer dataset and create file in ARFF format	AK
8	Weka integration on Android required for training/testing	AK
9	UI enhancement for enabling parameters only specific to a model and disable all other irrelevant parameters not specific to that model	AK
10	Random splitting of input dataset into training/testing dataset	AK
11	Read ARFF training file using weka	SP
12	Upload the ARFF train file to server from device	SP
13	Send system parameters for different models to server	SP
14	UI development for different models (SVM, KNN, Logistic Regression, NaÃŕve Bayes)	SP
15	Receive the serialized model file (trained model) from the server on device	SP
16	DE serialize the model file and read it using weka	MS
17	Display the tested results in a separate activity for that model	MS
18	Create Log file for saving the testing results	MS
19	UI development for different models (SVM, Logistic Regression, NaÃŕve Bayes, KNN)	MS
20	Video preparation for demonstrating the application	MS

6. “Support Vector Machine." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 7 December 2017.
7. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.
8. *“How to add external libraries to Android Project in Android Studio?"*. http://o7planning.org/en/10525/how-to-add-external-libraries-to-android-project-in-android-stud