Comparison of Supervised Learning Algorithms in Predicting Onset Type 2 Diabetes on AWS Platform

Team #16: Andrew McLean[1], Nick Peng[2], Jitesh Chawla[3]

[1,2,3]Ming Hsieh Department of Electrical Engineering, University of Southern California,
3740 McClintock Ave, Los Angeles, CA 90089

[1]*amclean@usc.edu*, [2]*npeng@usc.edu*, [3]*jchawla@usc.edu*

Abstract

In this project report, we detail the process of developing predictive models to determine the diagnoses of diabetes in patients. Various supervised learning algorithms are evaluated, tested, and compared to achieve the most accurate classification model. Important health parameters relevant towards diabetes are often times missing in a patient's medical record, which was addressed using techniques of preprocessing and imputation of the data.The classifier's performance results were studied separately as accuracy and precision were key metrics in determining the success rates of all the classifiers. Among the four classifiers studied, the best accuracy and precision values were achieved with the Random Forest classifier model. Using the powerful services offered by Amazon's AWS platform, we examined the performances of offloading the computing processes onto cloud resources. Scaling-up instance hardware improved the speed of running our machine learning algorithms, especially with larger datasets.

I. Introduction

Diabetes mellitus (DM) is defined as a group of metabolic diseases characterized by hyperglycemia resulting from defects in insulin secretion, insulin action, or both. According to a World Health Organization report, more than 220 million people worldwide suffer from DM and total costs associated with DM exceed $218 billion nationally in the United States alone [6]. The vast majority of diabetes diagnoses fall into two categories classified as type 1 and type 2 diabetes. The cause of diabetes differs between the two in that type 1 is caused by an absolute deficiency of insulin secretion, whereas type 2 is caused by a combination of resistance to insulin action and an inadequate insulin secretory response [2]. However, it is difficult to have a clear diagnosis of diabetes due to the lack of standardized criteria for examining patients and a consensus on what classifies normal blood sugar levels.

The remarkable advances in health sciences and biotechnology has engendered a large production of data, such as clinical information, sourced from Electronic Health Records (EHRs) [4]. This current trend has made the application of data mining methods and machine learning in the field of bioscience a major step in transforming large sums of medical information into valuable knowledge. Given the increasing epidemic of diabetes, it is imperative to understand the symptoms of diabetes through data mining methods and machine learning in hopes of providing a preventative model to the onset of developing diabetes. The study done in this paper is largely concerned with understanding and extracting pertinent features of examination patient data for use in various supervised learning algorithms in predicting the onset of developing Type 2 diabetes.

II. Problem Definition

The average person may not be aware that they are on track towards developing a serious disease until they actually develop it. For example, 1 in 3 Americans have prediabetes but because there are often no symptoms, they can progress to Type 2 diabetes without knowing it [1]. By analyzing large data sets of medical records and lifestyle habits of people who have been diagnosed at distinct stages, we can offer predictive models for people who are at risk of developing the disease and offer health recommendations to improve their condition before it's too late. Using the AWS Cloud, we plan to process and analyze our data sets to develop a training set that can be used to highlight the most crucial factors that lead to developing distinct stages of diabetes. By using supervised learning algorithms to identify the features that cause patients to develop diabetes, predictions can be made on whether new patients are at risk of diabetes based on the factors inputted. Based on a study, [3], where a

prediction rule was measured to assess and distinguish conditions affecting Type 1 and Type 2 Diabetes, we identified parameters to investigate further: inpatient use of insulin, blood glucose levels, and age of diagnosis. These quantitative parameters will be considered in our machine learning algorithms, but there are additional factors such as eating habits, physical activity, and other lifestyle traits that need to be considered to ensure the accuracy of a more complex predictive model applied to a more general population.

III. Dataset

A. Study Population

The Pima Indian population located near Phoenix, Arizona was the source population used to assess onset diabetes diagnosis. The dataset is sourced from the National Institute of Diabetes and Digestive and Kidney Diseases database, and is of interest due to the high incident rate of diabetes within the Pima Indian population [5]. For the purposes of this dataset, diabetes was diagnosed according to the criteria set by the World Health Organization criteria, which identified patients as diabetic if the 2 hour post-load glucose level was at or above 200 mg/dl at any survey exam or if the hospital serving the community discovered a glucose concentration of at least 200 mg/dl during the duration of medical treatment [6].

B. Data Features

There were eight quantitative variables measured in the dataset along with a binary output variable taking values of either a 1 or 0 indicating if the patient was diagnosed with diabetes or not respectively. Data obtained from 768 female patients over the age of 21 is contained within the dataset.

The input variables of the dataset were:

1. Number of times pregnant
2. Plasma Glucose Concentration at 2 Hours in an Oral Glucose Tolerance Test (GTIT)
3. Diastolic Blood Pressure (mm Hg)
4. Triceps Skin Fold Thickness (mm)
5. 2-Hour Serum Insulin (mmU/ml)
6. Body Mass Index (Weight in kg / (Height in in))
7. Diabetes Pedigree Function
8. Age (years)

The significance of the variables above are self-explanatory except for the Diabetes Pedigree Function (DPF), which was developed to provide a synthesis of the diabetes mellitus history in relatives and the genetic relationship of those relatives to the subject under examination [8]. This function incorporates health information from parents, grandparents, full and half siblings, aunts, uncles and first cousins. It provides a calculated measure of the expected diabetes risk for a test subject based on the genetic influence from relatives with and without a diabetes diagnosis [8].

$$DPF = \frac{\sum_i K_i(88-ADM_i)+20}{\sum_j K_j(ACL_j-14)+50}$$

The parameters of the equation are defined as follows:

i – ranges over all relatives with a diabetes diagnosis
j – ranges over all relatives without a diabetes diagnosis
K_x – percent genes shared by a $relative_x$

- 0.5 if $relative_x$ is a parent or full sibling
- 0.25 if $relative_x$ is a grandparent, aunt, uncle or half sibling
- 0.125 if $relative_x$ is a grandparent, aunt, uncle or half sibling

ACL_i – age of $relative_i$ when diabetes was diagnosed
ADM_j – age of $relative_j$ at the last non-diabetic diagnosis prior to test subject's examination date
Constants

- 88 and 14 represent the max and min ages at which relatives of the subject's in this study developed diabetes
- 20 and 50 were chosen to show specific relationships between age and diabetes diagnosis

C. Dataset Analysis

Analysis of the dataset was done on a local machine using R to make sense of the features involved and perform preprocessing. By traversing the table, several values were missing for some of the input variables with a full description provided in Table I.

TABLE I
MISSING VALUES OF DATASET BY VARIABLE

Variable	**Type**	**Num. Missing Values**
Number of Pregnancies	Continuous	0
Plasma Glucose Concentration	Continuous	5
Diastolic Blood Pressure	Continuous	35
Triceps Skin-Fold Thickness	Continuous	227
2-Hour Serum Insulin	Continuous	374
Body Mass Index	Continuous	11
Diabetes Pedigree Function	Continuous	0
Age	Continuous	0
Diabetes Diagnosis	Binary	0

Since the data size is relatively small, we did not want to omit the instances that had values missing, which would have been nearly 50% of the dataset. Therefore, we used two methods of imputation to obtain the most information from the given data. For the variables that had a small amount of missing values, we replaced the missing data with sensible median values given the distribution of that respective variable. Missing values for the variables that had a large amount of data values missing were predicted based on the statistical technique of multiple imputation.

Median Imputation Variables:

1. Glucose Test (GTIT)
2. Body Mass Index (BMI)
3. Diastolic Blood Pressure

Multiple Imputation Variables:

1. Serum Insulin
2. Triceps skinfold thickness

The method of multiple imputation presents a new sample bias to the dataset due the assumption that the predicted values will be consistent with the relative distribution of the original dataset. After applying the MICE function, package in R, to the dataset, we plotted the distribution of the two variables under imputation to verify that the distribution of the data didn't deviate drastically before and after the function was applied. The plots can be seen in Figure 1 below.

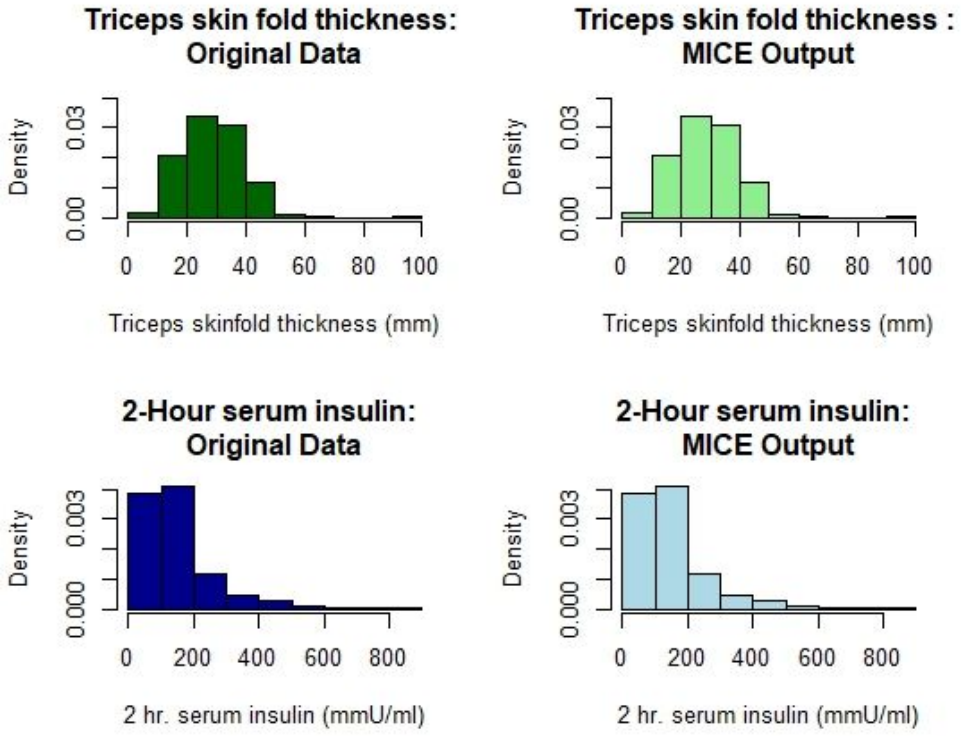


Figure 1: Distribution of the two variables under imputation before and after MICE function was applied

Compared with the original distributions, the two complete distributions after applying the MICE function shown above for Triceps skin-fold thickness and serum insulin are not significantly changed, indicating the predictions are statistically valid [7]. After this verification, the dataset is now considered complete and ready to be tested on our learning algorithms.

D. Preprocessing

After performing imputation on missing values, the original dataset was divided into two sets with the (80-20) composition as the Training Set and the Testing Set respectively.

1. Splitting into Training and Testing Set

The most important library for Machine learning i.e sklearn was installed in Python. By using its inbuilt train_test_split() function, the Pima-Diabetes Dataset was divided into Training Set(data_train, data_test) and correspondingly Testing Set (target_train and target_test) [10]. The testing dataset will be kept aside until the classifier is trained and then that classifier will be applied to the testing data set [9]. The Dataset (read from the .csv file) was converted to a Pandas Data frame named as the Utility_ Matrix1 by installing the pandas library in python. Following techniques outlined below were used to perform the pre-processing on the dataset.[10]

2. Feature Recasting/Conversion

After extracting the contents of a csv file to a Pandas Data frame, the foremost important task is to convert the entire dataset into numeric data type, whereas already numerical feature were left unchanged. Since all our features in the dataset were in a numerical format already, no recasting was necessary.

3. Feature Rescaling

We passed the data to the StandardScaler() function imported from the sklearn.preprocessing package of the sklearn library in python. Rescaled values are between (0,1) by default.

4. Dimensionality Reduction

There were only eight feature inputs and one output feature in the Pima Diabetes Dataset, each significant enough and contained relevant information to be considered in the training of the classifier. We achieved this by performing Principal Component Analysis and SVD (Singular-Valued Decomposition). In PCA, first of all the dataset was passed through a StandardScaler() function then the corresponding values for the Covariance Matrix via numpy.cov() function are calculated [12]. Also, we used the np.linalg.eig(cov_mat) function to calculate eigen_values and eigen_vectors [13]. These were then sorted in a descending order, where it was observed that all the eigenvalues were nearly equal, each of which had a significant impact on training the classifiers.

IV. Algorithms Used

There were several Classifier Algorithms used in this project, such as “Logistic Regression”, “Ada Boosting”, “Random Forest”, “Naive Bayes Classifier” and “Linear Regression”. All of these models were trained based on the training dataset provided, and then evaluated against the data testset. On the basis of the Classification_report() parameters like the precision, recall, F-1 score and support along with the accuracy of the predictions, Random Forest model gave the best results. We also described the theory behind Naive Bayes and Random Forest Classifiers briefly below, along with its experimental setup.

1. Logistic Regression Model

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. We implemented this classifier model through the inbuilt function in the sklearn.linear_model package in Python called the LogisticRegression(). The function logreg.fit(data_train,target_train) is used to fit the model on the training data and logreg.predict(data_test) gave the predicted values for our testing dataset. Because it is logistic, the dependent variable (DV) in the regression model is categorical. It usually covers the case of a binary dependent variable—that is, where it can take only two values, "0" and "1," representing outcomes that correspond to a response of either YES or NO. For our project it means either Having Diabetes / Not Having Diabetes represented by 1/0 respectively.

2. Ada Boosting

Ada Boosting can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. The benefits of applying AdaBoost is its adaptiveness, in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. Another important aspect is its sensitivity to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (e.g., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner. AdaBoost was implemented

in a similar manner as all the other classifiers by importing the sklearn.ensemble package in python.

3. Naive Bayes Model

In machine learning, *naive Bayes classifiers* are a family of simple probabilistic classifiers based on Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers. We implemented this classifier model with the help of an inbuilt function of sklearn.naive_bayes package in python called the GaussianNB(). Also, gnb.fit(data_train,target_train) is used to fit the model on the training data and with the help of gnb.predict(data_test) we get the predicted values for our testing dataset. From the description, we know that Naive bayes is not a distribution free classifier, is based on the concept of PDF(Probability density Function) and it requires the class priors. Several distribution methods such as "Normal/Gaussian", "Kernel", and "Multinomial" methods were tried and we got the best results in the case of a Normal Distribution. Best accuracy and F-1 scores were recorded for the Normal distribution amongst the other distributions used for the Naive Bayes Classifier.

4. Random Forest Model

We implemented this classifier model by using a function from sklearn.ensemble package called the RandomForestClassifier(). rnd_fc.fit(data_train,target_train), which is used to fit the model on the training data and with the help of rnd_fc.predict(data_test) we get the predicted values for our testing dataset. From the description of Random forests or random decision forests, we know they are an ensemble learning method for classification, regression and other tasks. It operates by constructing a multitude of decision trees at training time and outputting the class, which is either the mode of the classes (classification) or mean prediction (regression) of the individual trees.

5. Software Tools and Libraries-

(A) *Pycharm* - Installed from Jetbrains as our IDE, Pycharm was used to perform machine learning techniques on our machine. It provided support for all the libraries required for our project.

(B) *sklearn* - Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

(C) *pandas* - Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

(D) numpy - NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- useful linear algebra, Fourier transform, and random number capabilities

(E) matplotlib - Matplotlib is a plotting library for the Python programming language and its

numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

V. AWS Cloud Platform

A. AWS Machine Learning Service

On Amazon's AWS platform, there is an established, integrated Machine Learning service that provides thorough analysis of large datasets using cloud resources. This was our first approach to processing and modeling the Pima Diabetes Dataset. However, AWS's Machine Learning application only processes the data with logistic regression, which turns out to be a less accurate representation of the dataset in comparison to other classification algorithms [11]. Instead, this software tool became a supplement to the main Python sklearn tests run on EC2. Nevertheless, it still provides an efficient analysis of the data that assists us in making adjustments to the predictive model by comparing these results with other algorithms.

In order to utilize this application, the first step is to upload the CSV file of preprocessed data into an S3 bucket. This way the AWS can directly link to the data over the cloud, scan through its contents, and create a datasource out of it that can be used to train and evaluate a model. Within the AWS interface, each column of data is labeled with a feature and the outcome is selected for the model to be trained around. By randomly splitting up the data set, where 70% is used to train the model and the remaining 30% will be used to evaluate it,

Displayed below are the results generated by the AWS Machine Learning service after training and evaluating the model on the Pima Diabetes data set:

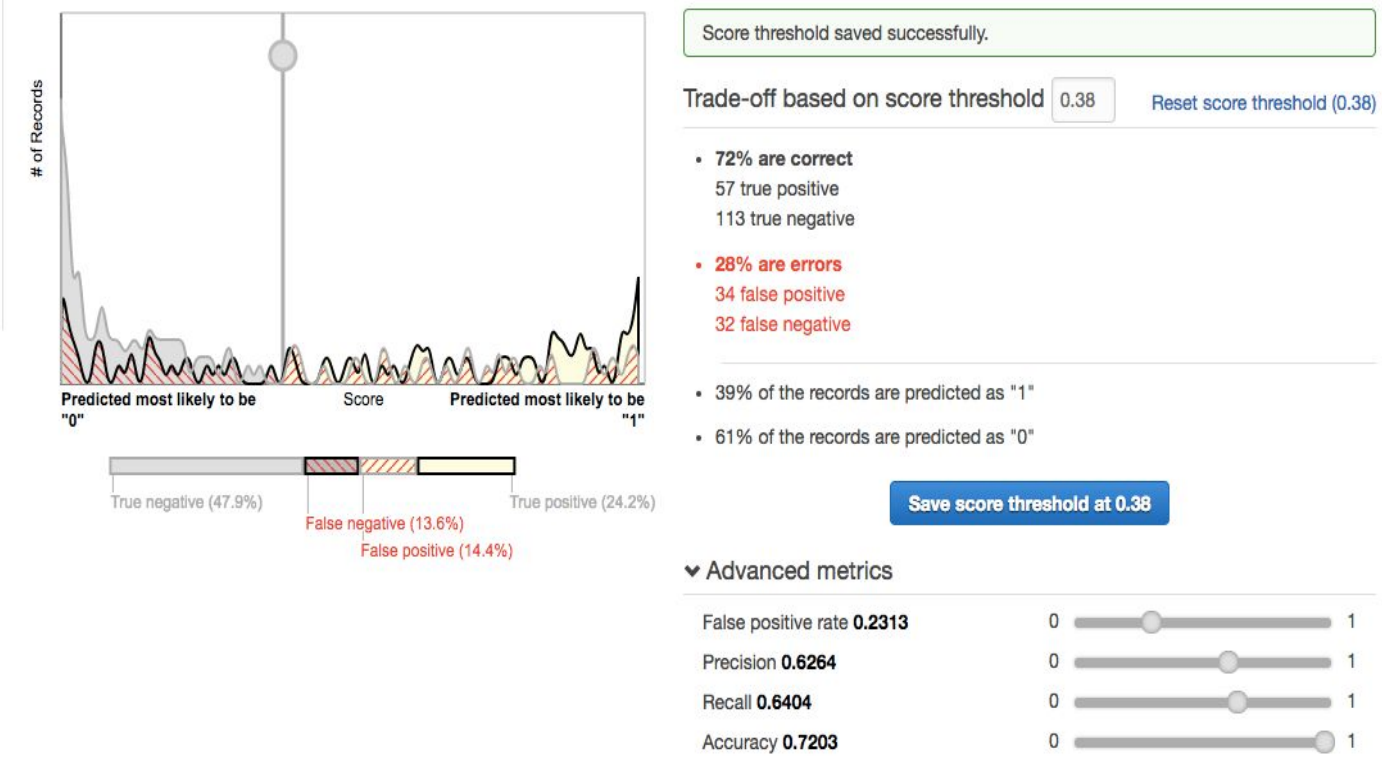


Figure 2: Visualization of Pima Indians Diabetes model, trained and evaluated on AWS Machine Learning.

As shown by the above results, there are still a fair amount of errors in classification. While our objective is to achieve the most accurate diabetes predictive model, it is more harmful to have false negatives

than false positives. A false positive would be diagnosing a patient with diabetes as healthy, which could be detrimental to their immediate health if left unregulated or untreated. On the other hand, a patient falsely diagnosed with diabetes might attempt to improve their lifestyle in a healthier manner, which would not have as many negative consequences besides taking unnecessary medication. Therefore, we tried to limit the amount of false negatives without increasing the total error significantly. This was accomplished by adjusting the score threshold, which changes the desired precision, accuracy, and recall values to find the optimal ratio of false positives and negatives deemed acceptable.

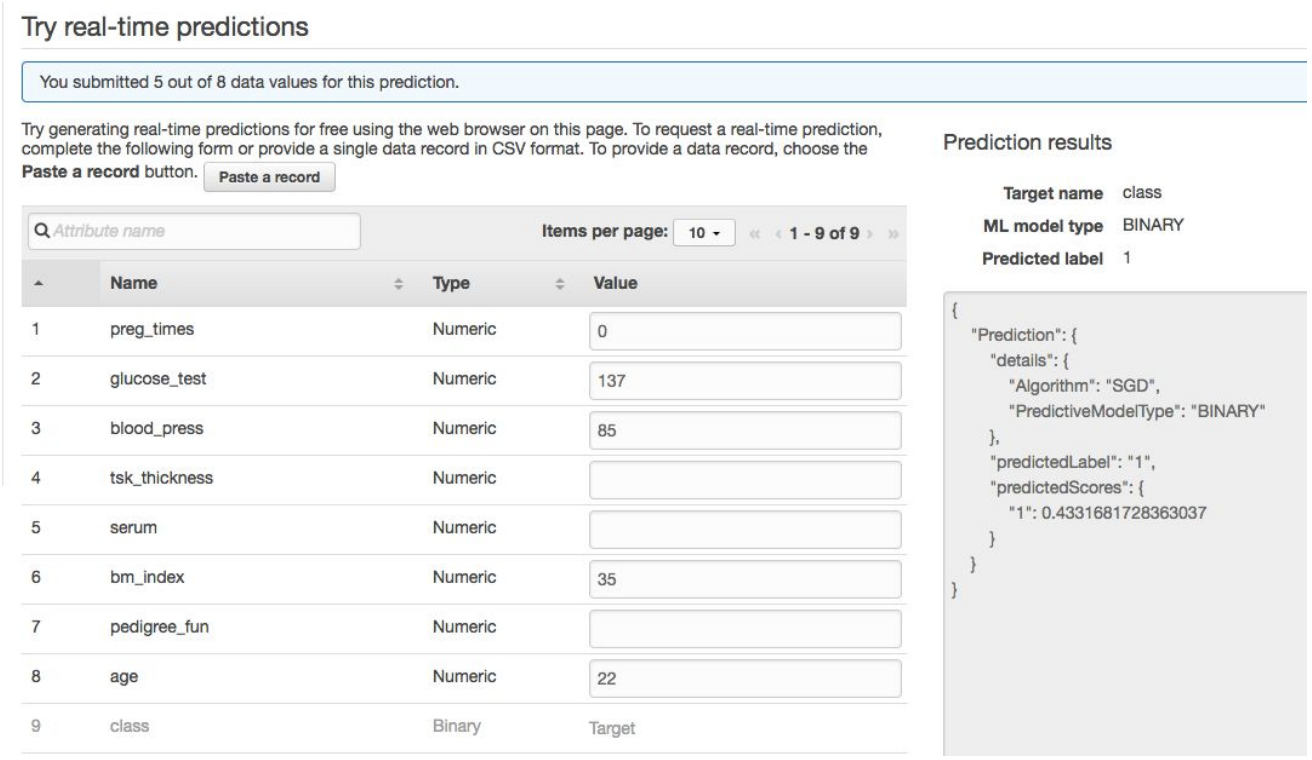


Figure 3: Example of predicting diabetes diagnosis with specific input parameters on AWS Machine Learning.

After evaluating the model in AWS, we can perform real-time predictions using their interface. In this example, we input a sample person to test some of the parameters with these model. By inserting higher values for blood pressure and BMI that a typical diabetic might have, we expected the model to report an outcome of having diabetes, which in fact this model does, designated by the 1 listed under prediction results in the figure above.

B. Step-by-Step Setup of EC2 Environment

To develop a more accurate predictive model for diabetes, other classification algorithms needed to be run. Fortunately, the scikit-learn library in Python supports a convenient method of performing machine learning. While this can be done on any local machine, running the application on the cloud has many advantages when it comes to speed, efficiency, and scalability. In this example, we used the AWS cloud with the Elastic Compute Cloud (EC2) service to perform supervised learning to obtain a predictive model on Diabetes.

The first step necessary before launching an instance is to create and download a key-pair to securely log in to the instance remotely. Then through the AWS EC2 dashboard, an instance can be launched directly. There are a variety of options to choose from when it comes to the operating system, hardware, and performance desired. For this experiment, we started with a t2.micro instance, set up with the default configuration for simplicity purposes. Then using a local machine, 'SSH' into the instance's address using the key-pair that was downloaded in the first step. In a separate terminal window, use the 'scp' (secure copy) command with the '–i'

flag and the public address of the EC2 instance to upload all the necessary files, most notably the python script and the 2 datasets to the instance.

While an EC2 instance is quite versatile, it is missing many of the required libraries needed to perform machine learning processes. The pip install method is an incredibly simple way to obtain all the libraries required for running the code (listed in a previous section). Finally, to monitor the instances' performance metrics, the detailed monitoring option from CloudWatch must be turned on in order to examine the featured parameters. When finished running programs and applications on the EC2 instance, it is important to terminate the instance manually to avoid excess charges (automatic termination on idle can be set up).

C. Cloud Performance Metrics

The performance metrics that have been selected for checking the cloud performance for the machine learning processes in this project are given below:

1. Scalability:

The formal definition is the ability to expand or contract computing resources, according to the needs of the user [14]. It is driven by the productivity and quality of the cloud system. For this project, we will be scaling up instance sizes to take advantage of more powerful hardware, such as more processors and RAM, to improve the speed of the machine learning program. This metric evaluates the economy of scale from one configuration to another. The higher this ratio, the more opportunity there exists to target the desired scaling scheme. Mathematically it is defined as [14]:

$$Scale - Up(N) = \frac{Time\ to\ Run\ on\ 1\ Processor}{Time\ to\ Run\ on\ N\ Processors}$$

2. Efficiency:

This is defined as the percentage of peak performance achieved. Using the previous values for Scale-Up we can obtain efficiency as [14]:

$$Efficiency\ (\Lambda) = Scale - Up\ (\Lambda) / N(\Lambda)$$

Where, N is the number of cores as opposed to the number of instances.

3. Productivity:

Cloud productivity for a configuration is defined as [14]:

$$P(\Lambda) = \frac{p(\Lambda) \times \omega(\Lambda)}{C(\Lambda)}$$

Where:

$p(\lambda)$ = Performance metric used (in our case, execution time is being used)

$C(\lambda)$ = Cost of implementing configuration

w(λ) = Quality of Service (QoS) of the cloud configuration

QoS: For AWS, we shall assume that the Quality of Service is 99% in accordance with the CloudHarmony report on cloud websites.

Cost: This is the relative cost of resources available and used versus the monetary cost of implementing a particular type of instance [14].

D. Scale-Up Experiments on EC2

In this project, we conducted the Scale-Up experiment, executing the program on various types of instance hardware to check the differences in performance for each of them. The main parameters that we used to compare the performances are CPU utilization and execution times. Below are the results for each type of instance.

1. CPU Utilization

The Pima Diabetes dataset may be fairly small with only 768 data entries, but we thoroughly ran and evaluated several different machine learning algorithms which accounts for the CPU usage shown.From the table and graph below, it is obvious that the peak consumption of the CPU resources decreases as we increase the number of processors (by using larger instance types). Supplying more cores imply more processing power in our system for the same load capacity, therefore decreasing the overall utilization percentage rate. This is fairly intuitive as the processor has fewer calculations to make, using fewer resources of the CPU.

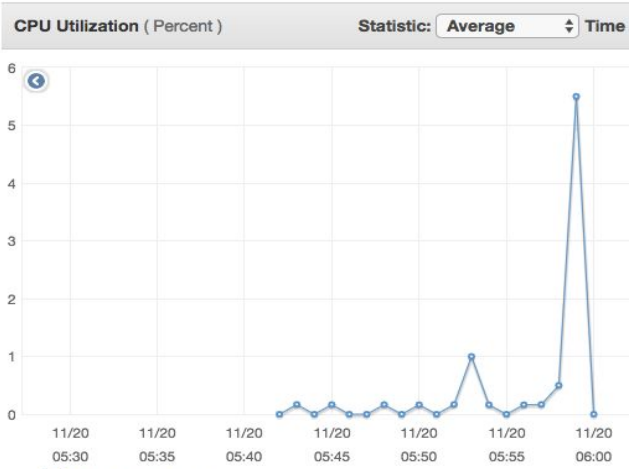


Figure 4: CloudWatch monitoring of CPU Utilization of t2.micro instance

TABLE II

PEAK UTILIZATION TIMES OF SCALE-UP EXPERIMENTS

Instance Type	Peak Utilization Percent Rate (%)
t2.micro	5.5
t2.large	2.58
m4.large	2.54
m4.xxlarge	1.995

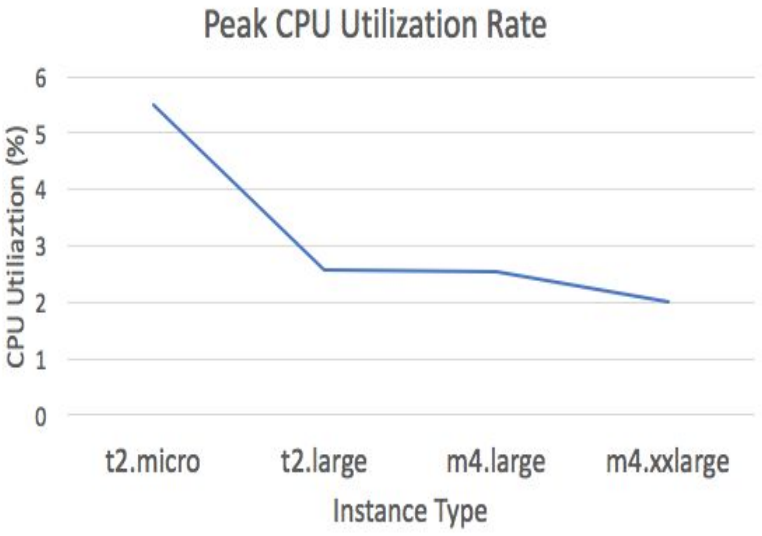


Figure 5: Scale-Up graph comparing CPU Utilization of each instance type

2. Execution times

The execution times of the code on the instances were recorded and are presented in Table III. To put this into perspective, the local machine on which the code was run ran the code in 1.1066 seconds.

TABLE III

EXECUTION TIMES OF SCALE-UP EXPERIMENTS

Instance Type	Execution Times (secs)
t2.micro	0.88080
t2.large	0.84988
m4.large	0.80758
m4.xxlarge	0.76405

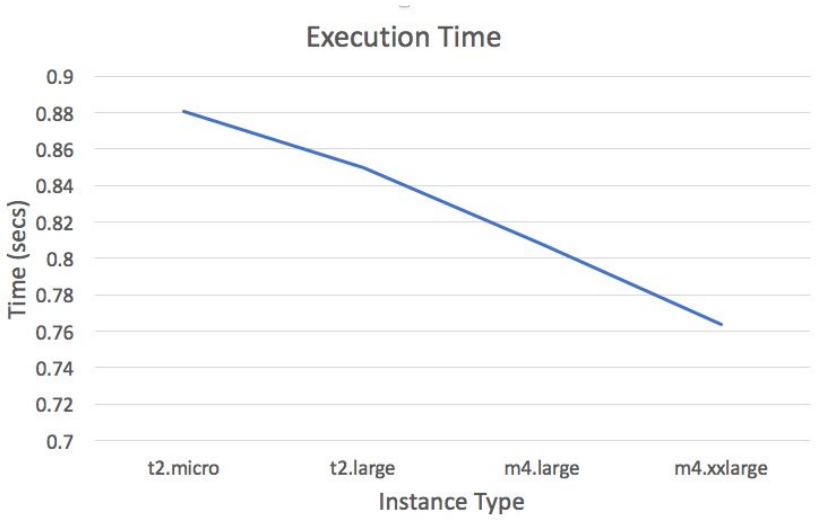


Figure 6: Scale-Up graph comparing execution times of each instance type

VI. Classification Results

A. Performance Evaluation Techniques

We calculated the performance of our various classifiers by passing the Testing dataset through the exact same process as the training Set. The first and most important step was preprocessing the testing dataset, similar to what was done on the training set. Then this preprocessed test set was allowed to pass through the various Classifiers trained earlier. By using an inbuilt function of sklearn.metrics library in python we used the classification_report() function to get the various resulting parameters for comparison. Also using the same library we used another function to calculate the Confusion_Matrix for the testing dataset in all the three classifiers to compare with each other. it gave the performance of each Model on testing set whose true value is known. Results were plotted graphically as well to make it look more comprehensive and legible by using the matplotlib.pyplot library in python.

B. Classification Results

The results of various Classifiers used during the Analysis of the Pima Diabetes Dataset are shown on the next page. These results are calculated by using the classification_report() from sklearn.metrics library in Python [5][6].

TABLE IV

ACCURACY RESULTS LOGISTIC REGRESSION

Diabetes Diagnoses	Precision	Recall	F1-Score	Support
YES	0.75	0.89	0.82	95
NO	0.76	0.53	0.62	59
Average/Total	0.75	0.75	0.74	154

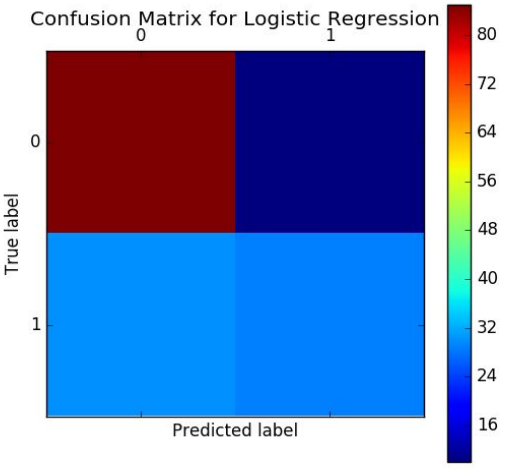


Figure 7: Confusion Matrix for Logistic Regression

TABLE V

ACCURACY RESULTS ADA BOOSTING

Diabetes Diagnoses	Precision	Recall	F1-Score	Support
YES	0.81	0.93	0.86	95
NO	0.84	0.64	0.73	59
Average/Total	0.82	0.82	0.81	154

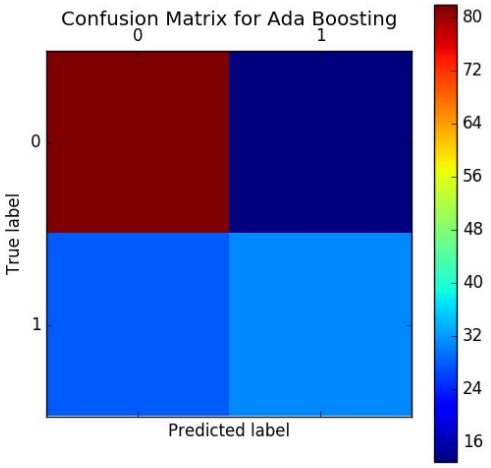


Figure 8: Confusion Matrix for Ada Boosting

TABLE VI

ACCURACY RESULTS BAYES MINIMUM CLASSIFIER

Diabetes Diagnoses	Precision	Recall	F1-Score	Support
YES	0.76	0.78	0.77	95
NO	0.63	0.61	0.62	59
Average/Total	0.71	0.71	0.71	154

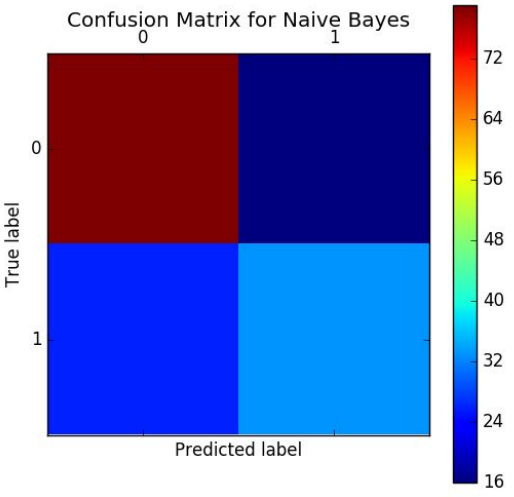


Figure 9: Confusion Matrix for Naive Bayes

TABLE VII
ACCURACY RESULTS RANDOM FOREST

Diabetes Diagnoses	Precision	Recall	F1-Score	Support
YES	0.92	0.95	0.93	95
NO	0.50	0.19	0.28	59
Average/Total	0.92	0.92	0.92	154

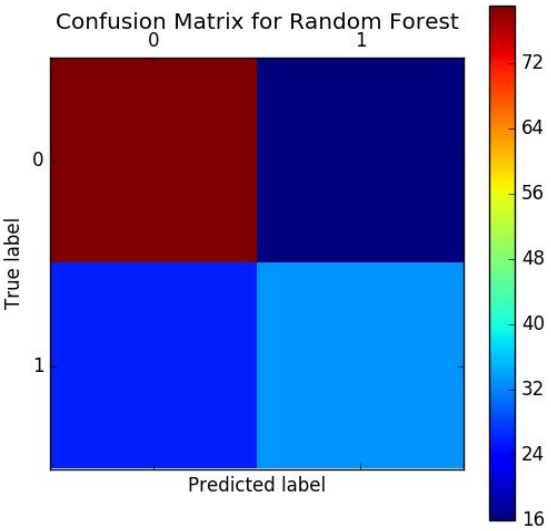
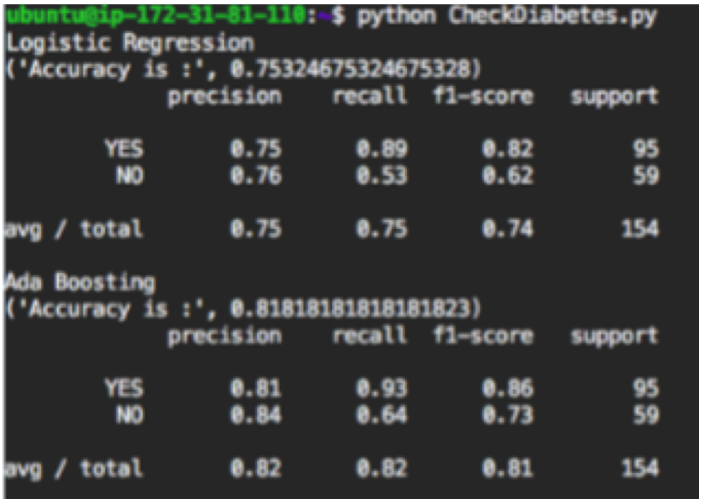


Figure 10: Confusion Matrix for Random Forest

TABLE VII
COMPARISON OF OVERALL CLASSIFIER ACCURACY

Classifier Used	Accuracy	Standard Error
Logistic Regression	0.75	+/- 0.04
Ada Boosting	0.81	+/- 0.04
Bayes Minimum	0.71	+/- 0.04
Random Forest	0.91	+/- 0.04

After comparing the results of each machine learning algorithm, it is clear that the Pima diabetes set is best classified using the Random Forest classifier. With an average accuracy of about 91.5% and a precision score of 92%, we can use this model to predict future patients chances of being diagnosed with diabetes if supplied with their individual data.



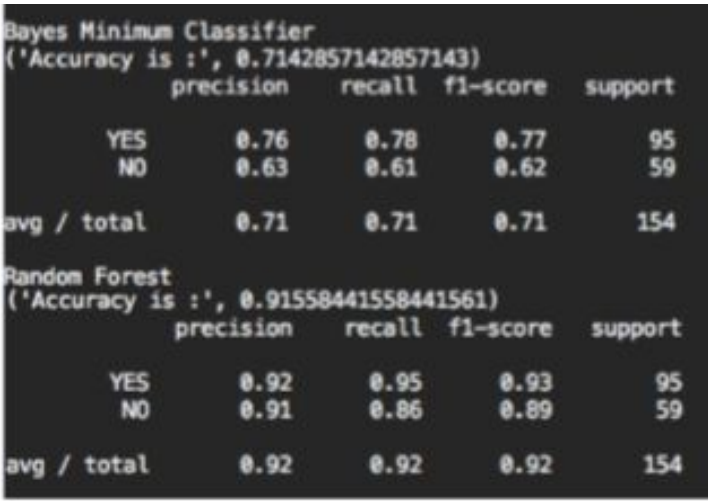


Figure 13: Output of Evaluating Machine Learning Classifiers on EC2 Instance

VII. Conclusions

A. Predicting diagnoses of diabetes

From the results compiled, we can conclude that the random forest classification algorithm most accurately models the diabetes dataset and provides the best predictive model. Especially for a health condition as widespread and variable as diabetes, it is extremely difficult to precisely model the disease from a restricted number of characteristics. However, by evaluating crucial health factors such as glucose, insulin, and BMI that can be measured for any person, the model can effectively identify patterns and symptoms of a pool of patients. When inputting and adjusting sample data into our predictive models, we can clearly see the increased conditional probabilities of being diagnosed with diabetes with higher levels of glucose, blood pressure, and BMI, health measures that are strongly correlated with diabetics. Diabetes prevention and treatment primarily revolves around regulating blood sugar levels, limiting excess intake, and maintaining a healthy lifestyle. This model should still primarily serve as an initial assistive measure for diagnosis but is another useful perspective of analyzing one study population as opposed to generalizing an entire global population.

B. Cloud Performance Analysis

Although our dataset was relatively small, the effects of using various EC2 instance types on computing efficiency, and productivity were seen in our study. From our scale-out experiments on peak utilization rate of CPU's, we can observe that the utilization rate is higher in the t2.micro instance types compared to the m4.xxlarge instance types due to the less amount of CPU's available on the t2.micro instance compared to the m4.xxlarge instance type. Additionally, we were able to observe the speedup effects of using instance types with more CPU's as shown in Figure 6 representing the trend of decreasing execution time as the number of CPU's increases based on the instance type used. The scale-out effects presented in this study would be more noticeable if more data was present, but the initial findings from a small population of data is enough to engender further exploration into using cloud resources for big data and machine learning applications in the health industry.

VIII. REFERENCES

[1] American Diabetes Association, Diagnosing Diabetes and Learning about Prediabetes. Diabetes Basics: Diagnosis 2016.

[2] American Diabetes Association, Diagnosis and Classification of Diabetes Mellitus. Diabetes Care 2010.

[3] W. Lo-Ciganic, J.C. Zgibor, K. Rupert, V.C. Arena, R.A. Stone. Identifying Type 1 and Type 2 Diabetic Cases Using Administrative Data: A Tree-Structured Model. J Diabetes Sci Technol 2011. 5(3):486-93.

[4] I. Kavakiotis, O. Tsave, A. Salifoglou, N. Maglaveras, I. Vlahavas, I. Chouvarda. Machine Learning and Data mining methods in Diabetes Research. Computational and Structural Biotechnology Journal 2017. 15:104-116.

[5] W.C. Knowler, P.H. Bennett, RF. Hamman, and M. Milier. Diabetes incidence and prevalence in Pima Indians: a 19-fold greater incidence than in Rochester, Minnesota. Am J Epidemiol 1978. 108:497-505.

[6] World Health Organization, Report of a Study Group: Diabetes Mellitus. World Health Organization Technical Report Series. Geneva, 727 1985.

[7] X. Zhou. Case Study on Pima Indian Diabetes 2016.

[8] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, R.S. Johannes. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. National Institute of Diabetes Digestive and Kidney Diseases: Epidemiology and Data Systems Program 1988.

[9] SUPERVISED LEARNING — SCIKIT- LEARN 0.18.1 DOCUMENT -Supervised learning — scikit- learn 0.18.1 documentation", Scikit-learn.org, 2017. [Online]. Available: http://scikit-learn.org/ stable/ supervised_learning.html#supervised-learning.

[10] PREPROCESSING DATA — SCIKIT- LEARN 0.18.1 :- Preprocessing data — scikit-learn 0.18.1 documentation", Scikit- learn.org, 2017. [Online]. Available: http://scikit-learn.org/ stable/modules/ preprocessing.html#preprocessing.

[11] Tutorial: Using Amazon ML to Predict Responses to a Marketing Offer. [Online]. Available: http://docs.aws.amazon.com/machine-learning/latest/dg/tutorial.html.

[12] Principal Component Analysis [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition-PCA.html

[13] Single Valued Decomposition [Online]. Available: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.svd.html

[14] Hwang, Kai, et al. Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies. *IEEE Transactions on Parallel and Distributed Systems*.