yuf

by Kjfu Yufdtdt

Submission date: 23-May-2020 11:55PM (UTC+0530)

Submission ID: 1330564742

File name: Plag_check_major.pdf (387.82K)

Word count: 3950

Character count: 21431

INDEX

ABSTRACT

One of the most vital steps in building a good, supervised machine learning model is the process of model selection. The selection of the best learning algorithm from a set of available, applicable algorithms is of crucial importance. When dealing with the problem of data classification, there is a need to determine the model that provides the best accuracy for the given dataset.

Model selection by evaluation aims to determine which supervised learning algorithm performs the best on unseen data or testing data. Before shipping any machine learning model into production, there is a need to ensure that its performance does not degrade when working with new data. The trained models can be compared on one or more parameters. A single parameter to determine the best model can produce a skewed result, which begs the need for the use of various parameters like accuracy, precision, and some other parameters to determine the best algorithm for any given dataset.

Our project aims to apply various supervised machine learning algorithms on the available dataset and to determine the best algorithm for the same determined by comparing the value of several different parameters.

I. INTRODUCTION

One of the fastest fields in computer science is machine learning. While it can be used in various different fields one of the most important applications of it can be said to be in the field of medical science, such as finding patterns for the detection of diseases through common patterns from the available report of the patient. While humans have the capability to identify such symptoms themselves the presence of a model for detection provides a second opinion for better results.

While most available models use a specific supervised machine learning model, the availability of various classification models makes it important to determine the best applicable algorithm on the dataset of any particular disease.

Our project aims to determine the most suitable classification algorithm for diabetes prediction.

In this project, we have compared three commonly used classification algorithms namely,

i. Support Vector Machine (SVM),

ii. Naïve Bayes, and

iii. Random Forest.

The above mentioned three supervised machine learning classification algorithms are measured on parameters such as accuracy, recall, F-measure, and precision to determine the most suitable classification for the requirement.

II.MOTIVATION

Diabetes is one of the most chronic illnesses that affects a large population around the world and has been steadily on the rise in India. According to the National Diabetes and Diabetic Retinopathy Survey released by the health and family welfare, the prevalence of diabetes is at 11.8% in the last four years (2015 – 2019). According to the World Health Organisation (WHO), there are an estimated 72.96 million cases of diabetes in the adult population in India.

With no cure for diabetes, its early detection is considered to be the best available option to avoid the possible complication.

The high cases of diabetes in India make it a suitable disease to test various classification algorithms and determine the most suitable among them. The most commonly used classification has been compared to find the most suitable one.

III.LITERATURE SURVEY

III.1. Classification Mechanism of Support Vector Machine

SVM which refers to support Vector Machine, that is a supervised, machine learning classification and regression algorithm that works by producing an optimal hyperplane as an output, which helps in the classification of new examples into a suitable class. SVM is one of the preferred algorithms for the classification purpose due to its ability to achieve a good quality of classification generalisation through small data. SVM works on the principle of Structural Risk Minimisation (SRM). For example, in case of the two dimensional plane (i.e. classification on basis of two features), the hyperplane is of the form of a line dividing the plane into two parts such that one class lies either each side of the line.

Support vectors refer to the data points that are closest with respect to the hyperplane and influence its position and orientation. Our objective is to maximize the margin (which is the maximum Euclidean distance between the data points of the two of the classes for which the hyperplane is under consideration) of the classifier. When any of the support vector(s) is deleted, the position of the hyperplane gets changed.

SVMs for two-class classification belong to one of the three categories: linearly separable, linearly non-separable, and nonlinear.

1. When the dataset is linearly separable, the margin is maximised by minimising the regularization penalty.
2. In the case of a linearly non-separable dataset, the SVM is modified to add a penalty for violating the classification constraints. For this, we take the help of slack variables.
3. In the case of a non-linear dataset, SVM constructs an optimal separation hyperplane in the higher dimension.

SVMs also use the concept of Kernel function. By the use of kernel function, construction of mapping into higher dimensional feature space is created. The main purpose of Kernel function is to reduce the complexity of finding the mapping function.

III.2. An Empirical Study of Naïve Bayes Classifier

Naïve Bayes is a supervised machine learning algorithm that simplifies the learning process by the assumption that features of the dataset are independent of each other. And although independence among features is generally a poor choice, in practice naïve Bayes often competes well with other classifiers.

Although already have established some optimality conditions of Naïve Bayes, the aim of this study is to get a deeper understanding of the characteristics that affect the performance of the classifier.

The approach uses Monte Carlo simulations that allow for a systematic study of the classification accuracy of various different classes of randomly generated problems. The Naive Bayes classifier has been shown to work best in two cases: completely independent features (which is expected scenario) and functionally dependent features (which in fact is surmising). Naïve Bayes shows its worst performance in between the above mentioned two extremes.

The accuracy of the model is not directly related with the degree of features dependencies. The better measure of accuracy is the loss of information that features contain regarding the class when Naïve Bayes model is assumed. However, further theoretical and empirical study is necessary for better understanding of the relation between behaviour of Naïve Bayes and the information – theoretic metrics.

III.3. Random Forest Classifier

Random Forest is made up of two words Random and Forest, where Forest means a large collection of trees, and here it means decision trees and Random here signifies random selection of subsets of all features from the dataset.

Each decision tree contributes to the final classification.

A decision tree is a tree in which leaf nodes represent the classes. In our case, there are two classes: 0 represents the class of Non-diabetic patients, and 1 represents the class of Diabetic patients. Each internal node of the decision tree represents a feature, and on the basis of that feature decision is made (yes or no).

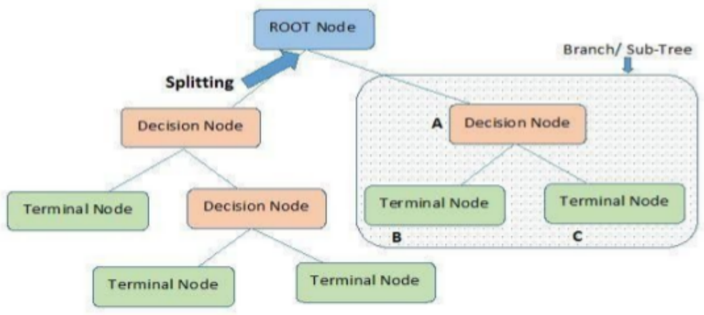


Figure 1. A single decision tree

Internal nodes are selected according to Gini Gain and Information Gain. The best possible split at each node is done and features whose Gini Gain is maximum will be an important feature and will do the best split.

Suppose z trees are created, then the data is passed to each tree, and each tree predicts a class to which the given data belongs. On the basis of the majority votes, the class of that data is decided. The following figure is demonstrating how to predict the class of given data in a random forest

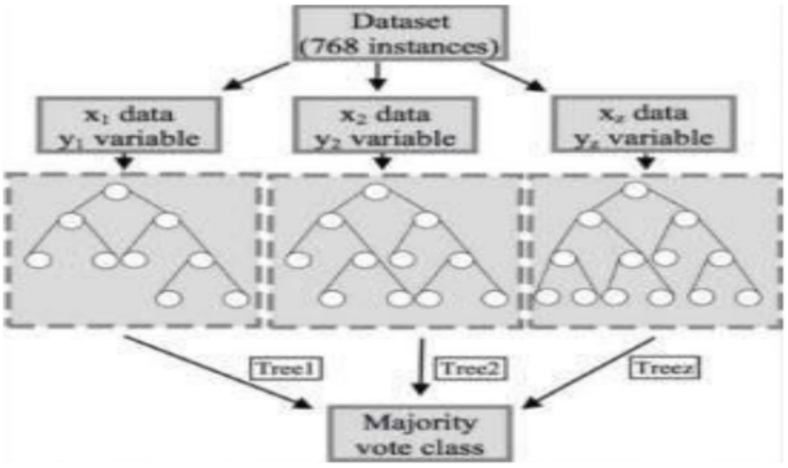


Figure 2. Various trees in Random Forest

IV.CONTRIBUTION TECHNICAL – COMPARISON OF CLASSIFICATION ALGORITHMS

The detailed approach which is used in this project is described below -

1.First of all, for any machine learning model, a large dataset is required. Data can be gathered from various resources. For diabetes prediction, we have used a dataset of PIMA Indians, which is a collection of 768 patients, and each row of the dataset represents a patient's various health parameters (8 parameters are used here).

Feature	Description	Unit
Pregnancies	Number of times Pregnant	Integer
Glucose	Plasma Glucose Concentration at 2 hours in an oral glucose tolerance test	Mmol/L
Blood Pressure	Diastolic blood pressure	mmHg
BMI	Body Mass Index	Kg/m^2
Diabetes Pedigree Function	Diabetes Pedigree Function	NA
Age	Age in years	Years, integer
Skin	Triceps skin fold thickness	mm
Insulin	2-Hour serum insulin	Mu U/ml
Outcome	Class Variable	0 or 1

Table 1: Description of the dataset

The dataset was stored in .csv format and read with the help of pandas library.

2. Now we have our data, but data that is gathered can be incomplete or inconsistent. Maybe values of any patient's parameter are not available, so we have to do data pre-processing to avoid these situations because this incompleteness of data can vary the accuracy of prediction in the end.

3. In any machine learning model, the importance of several features present usually is not the same. Because sometimes some features are having more impact on prediction in comparison to others. So feature selection is made in diabetes prediction. We selected the most important features and analysed the difference in the accuracies.

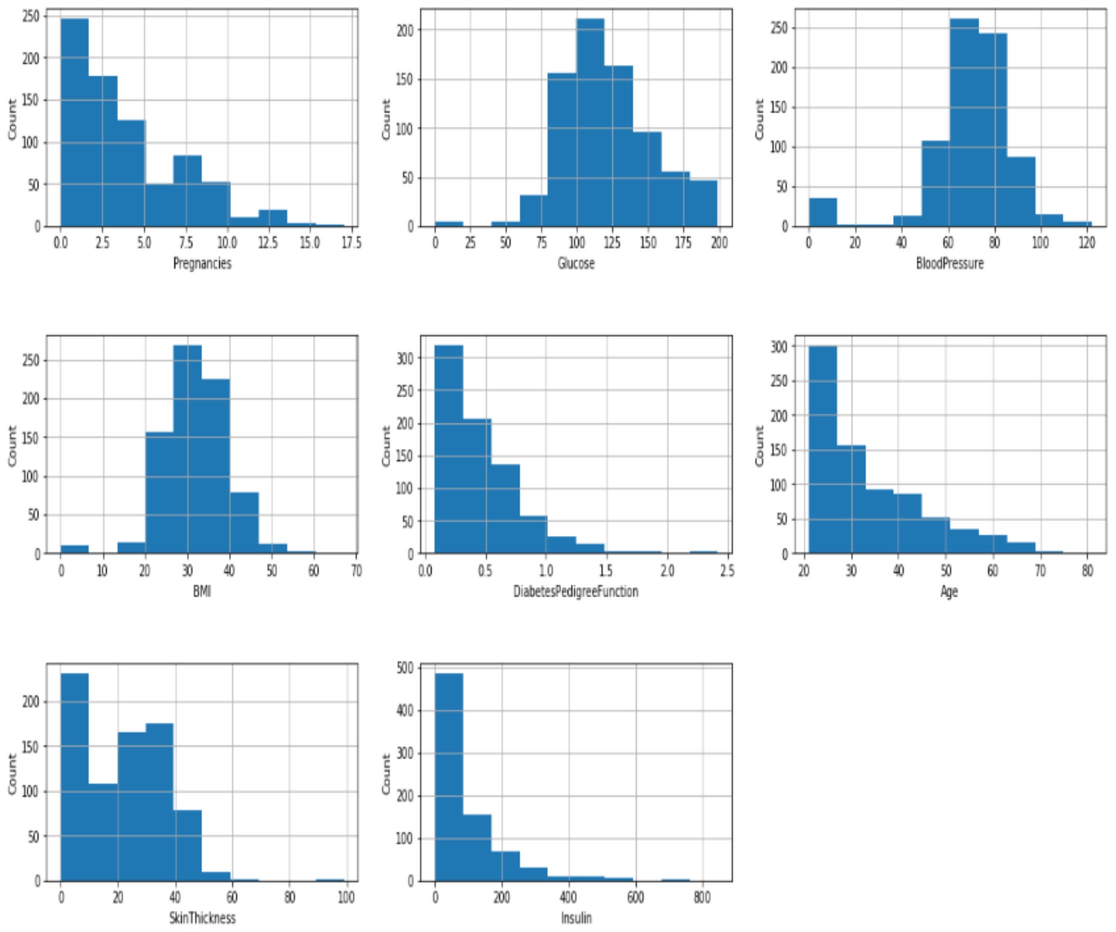


Figure 3: Histogram representation of features and their respective count

4. After feature selection, the dataset is divided into two parts: training set and testing set. We analysed our model with a testing set size of 20% and 30%. We can basically tune testing set size accordingly to achieve higher accuracy.

5. Then three classifiers, Naive Bayes, SVM, and Random Forest, are trained on the training set. These classifiers are further used to predict the testing set.

6. After training of classifiers, testing data is predicted. Through comparison of the predicted values of testing data and actual values of testing data the confusion matrix is created. The confusion matrix helps to find the number of true positive, false positive, true negative, and false negative. True positive means a person has diabetes and is predicted as diabetic. True negative means a person does not have diabetes and predicted as notdiabetic.

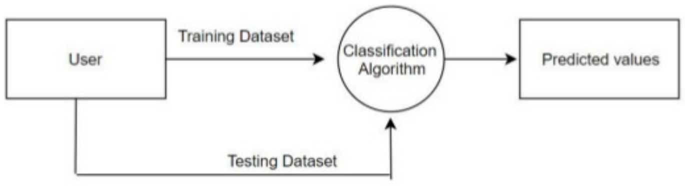
False-positive means a person does not have diabetes but predicted as diabetic. False-negative means a person has diabetes but predicted as not diabetic.

7. We used four measures to compare the performance of the above-mentioned models. These measures are – Precision, Accuracy, Recall, and F_Measure.

8. After that, the above mentioned four measures are visualized, and a comparison of each model's accuracy is shown.

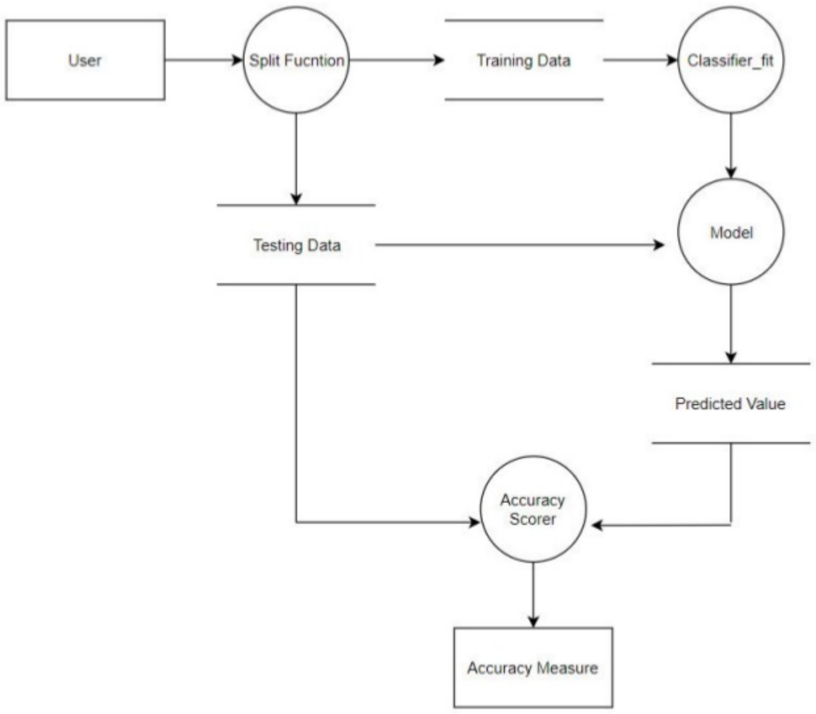
9. The accuracy value for each test set (20% and 30%) was calculated 10 times to better generalise the result.

V. Data Flow Diagram



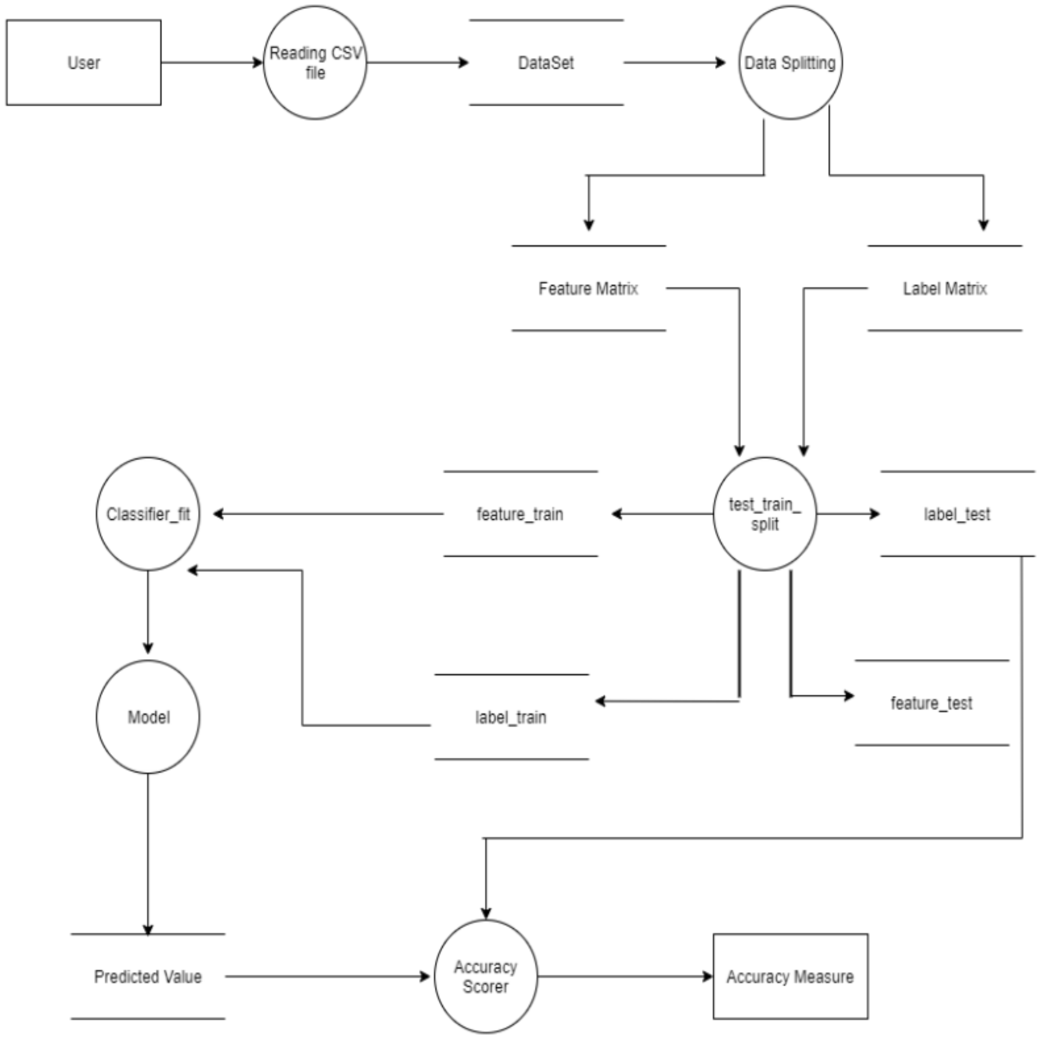
Level 0 DFD for Data Classification

Fig 4. Level 0 DFD



Level 1 DFD for the Data classification

Fig 5. Level 1 DFD



Level 2 DFD for Data classification

Fig 6. Level 2 DFD

VI. OBSERVATION

Naïve Bayes	SVM	Random Forest
71.42857	74.67532	71.42857
70.12987	70.77922	68.83117
77.27273	77.92208	72.72727
75.324468	77.92208	75.32468
75.32468	79.22078	75.32468
77.92208	80.51947	76.62338
77.27373	81.16883	77.27273
77.72727	76.62338	74.67532
79.87013	79.87013	80.51948
76.62338	79.22078	77.27273

Table 1. Accuracy measure at 20% testing data with all 8 features

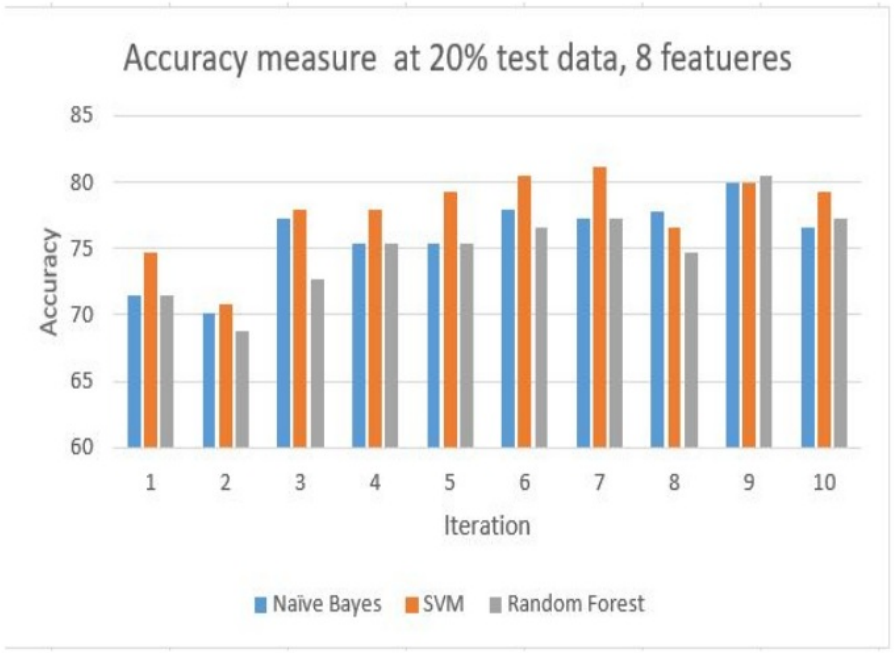


Fig 7. Bar Graph for accuracy measure at 20% testing data with all 8 feature

Naïve Bayes	SVM	Random Forest
75.32468	76.19048	67.96537
79.22078	80.95238	77.05628
78.78788	77.92208	77.48918
73.59307	75.75758	75.32468
75.75758	76.19048	75.32468
77.48918	78.78788	72.29437
77.48918	77.9208	75.75758
74.89177	77.05628	75.32468
75.32468	78.35498	79.22078
73.59307	76.62238	74.45887

Table 2. Accuracy measure at 30% testing data with all 8 features

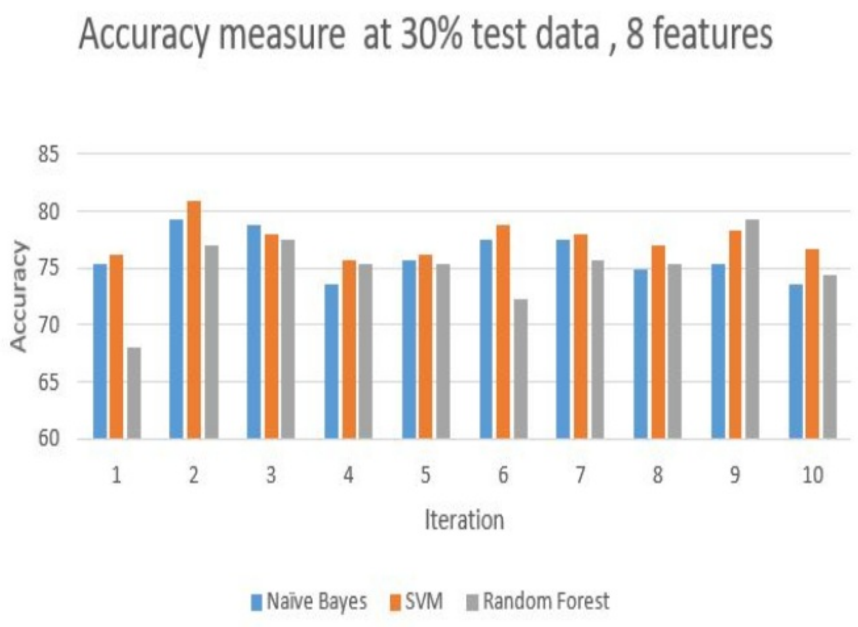


Fig 8. Bar Graph for accuracy measured at 30% testing data with all 8 features.

Naïve Bayes	SVM	Random Forest
79.87013	80.51948	73.37662
79.22078	80.51948	76.62338
75.97403	76.62338	76.62338
79.87013	77.92208	75.97403
74.67532	80.51948	79.22078
74.67532	72.07792	76.62338
77.92208	78.57143	72.07792
75.97403	76.62338	68.83117
75.3468	74.67532	73.37662
77.27273	79.87013	75.97403

Table 3. Accuracy measure at 20% testing data with only top 6 features

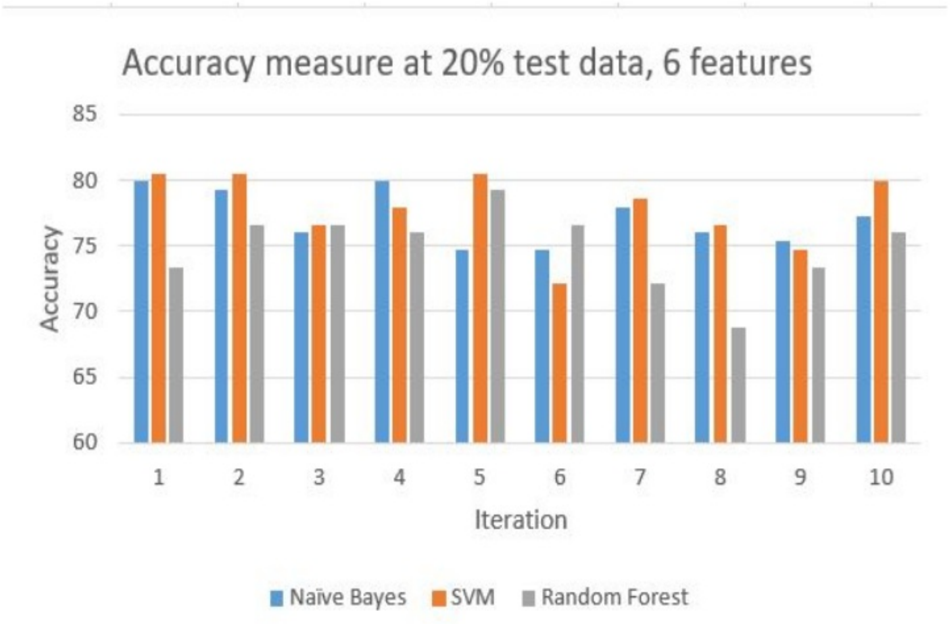


Fig 9. Bar Graph for accuracy measured at 20% testing data with 6 features.

Naïve Bayes	SVM	Random Forest
74.89177	77.48918	77.48918
75.32468	75.75758	75.19048
76.62338	76.19048	71.86147
73.16017	73.16017	70.99567
79.65368	78.78788	73.16017
78.35498	80.08558	79.22078
70.56277	74.02597	71.86147
78.35498	79.22078	73.59307
75.75758	77.05628	70.12987
75.32468	74.89177	77.05628

Table 4. Accuracy measure at 30% testing data with only top 6 features

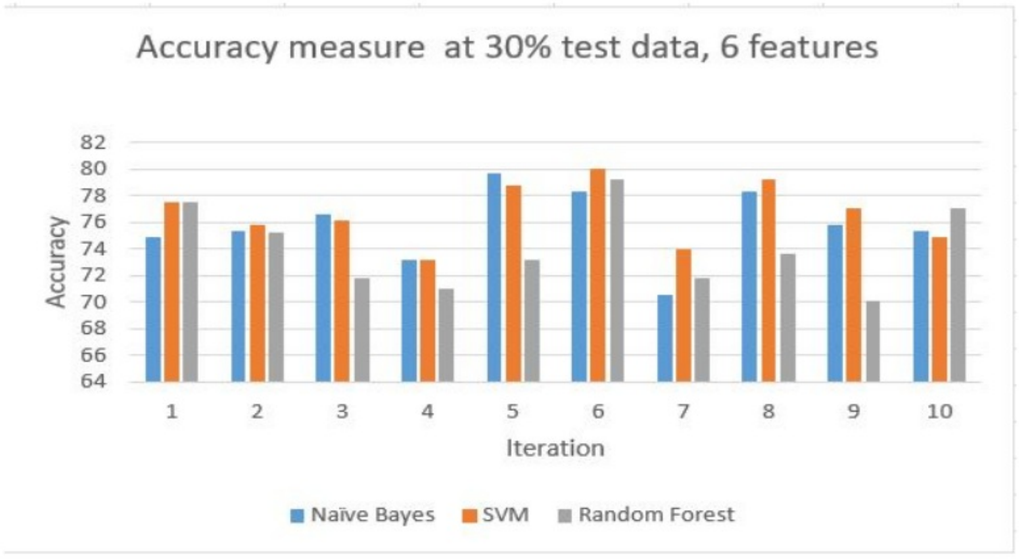


Fig 10. Bar Graph for accuracy measure at 30% testing data with 6 features.

VII. RESULT

From the above tables and graphs it can easily be observed that Support Vector machines (SVM) have the highest and most consistent accuracy among all three compared supervised classification algorithms. The better result obtained for SVM can be attributed to the fact that it works quite well in the presence of small dataset due to better ability to determine the required hyperplane. **Support Vector Machines** algorithm is the better algorithm compared to **Random Forest** and **Naïve Bayes** in prediction of Diabetes.

VIII. REFERENCES

[1] C.Junli, and J.Licheng, “Classification Mechansim of Support Vector Machines”, 5th International Conference on Signal Processing Proceedings, 6th World Computer Congress, 2000

[2] I.Rish, “An Empirical Study of Naïve Bayes Classifier”, IJCAI 2001 workshop on empirical methods in artificial intelligence, Vol 3, Issue 22, Pg 41 – 46, 2001

[3] A.Pretorius, S.Bierman, and S.J.Steel,“A Meta-Analysis of Research in Random Forests for Classification”, Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), Stellenbosch, 2016

[4] D.Shetty, K.Rit, S.Shaikh, and N.Patil, “Diabetes Disease Prediction Using Data Mining”, International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS), 2017

[5] Jin X., Xu A., Bie R., Guo P. (2006) Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles. In: Li J., Yang Q., Tan AH. (eds) Data Mining for Biomedical Applications. BioDM 2006. Lecture Notes in Computer Science, vol 3916. Springer, Berlin, Heidelberg

[6] P. Domingos, and M. Pazzani, “On the Optimality of the simple Bayesian Classifier under Zero-One Loss”, Machine Learning, Volume 29, Number 2 – 3, Page 103, 1997

[7] D.Sisodia,D.S.Sisodia, “Prediction of Diabetes using Classification Algorithms”, International Conference on Computational Intelligence and Data Science (ICCIDS 2018)

[8] Orabi, K.M., Kamal, Y.M., Rabah, T.M., “Early Predictive System for Diabetes Mellitus Disease”, Industrial Conference on Data Mining, Springer. pp. 420–427, 2016

[9] Pradhan, P.M.A., Bamnote, G.R., Tribhuvan, V., Jadhav, K., Chabukswar, V., Dhobale, V., “A Genetic Programming Approach for Detection of Diabetes. International Journal Of Computational Engineering Research 2”, 91–94, 2012

[10] Tarik A. Rashid, S.M.A., Abdullah, R.M., Abstract, “An Intelligent Approach for Diabetes Classification, Prediction and Description”. Advances in Intelligent Systems and Computing 424, 323–335, 2016

[11] Nai-Arun, N., Moungmai, R., “Comparison of Classifiers for the Risk of Diabetes Prediction”. Procedia Computer Science 69, 132–142, 2015

APPENDIX

A. COMPLETE CONTRIBUTORY SOURCE CODE

```
#initial importing some of the required libraries
import pandas as pd #used to manipulate the .csv file
import matplotlib.pyplot as numb_plt # for visualization of features present in dataset
import numpy as num_p #for mathematical operation
pat1_data = pd.read_csv("diabetes.csv")
#print(pat1_data.head(5))
from sklearn.model_selection import train_test_split
# feature matrix: x
xp = pat1_data.iloc[:,:-1].values
# label matrix: y
yp = pat1_data.iloc[:,-1].values
# print(xp)
# print(yp)
#random splitting of dataset into two dataset: training and testing dataset
xp_train, xp_test, yp_train, yp_test = train_test_split(x, y, test_size  = 0.20)
import itertools

cols = pat1_data.columns[:8]
numb_plt.subplots(figsize = (20,15))
col_length = len(cols)

for var_x, var_y in itertools.zip_longest(cols, range(col_length)):
  numb_plt.subplot((col_length/2), 3, var_y + 1)
```

```
    numb_plt.subplots_adjust(wspace = 0.2,hspace = 0.5)
    pat1_data[var_x].hist(bins = 10)
    numb_plt.xlabel(var_x)
    numb_plt.ylabel('Count')
numb_plt.show()
xp_train.shape
6
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.metrics import 1confusion_matrix, accuracy1_score, make_score

naive_classifier_model = GaussianNB()
naive_classifier_model.fit(xp_train, yp_train)

train1_predict1 = naive_classifier_model.predict(xp_train)

#getting accuracy for training dataset
from sklearn import metrics
print("Accuracy = ", accuracy1_score(yp_train, train1_predict1))
train_1accuracy_value = accuracy1_score(yp_train, train1_predict1)

#Getting the accuracy value for testing dataset
test1_predict1 = naive_classifier_model.predict(xp_test)
print("Accuracy = ",accuracy1_score(yp_test, test1_predict1))
test_1accuracy_value = accuracy1_score(yp_test, test1_predict1)

accuracy1 = test_1accuracy_value
```

```
confusion_mat = 1confusion_matrix(yp_test, test1_predict1)
true1_pos = 1confusion_matrix(yp_test, test1_predict1)[1, 1]
true_neg = 1confusion_matrix(yp_test, test1_predict1)[0, 0]
false1_pos = 1confusion_matrix(yp_test, test1_predict1)[0, 1]
false1_neg = 1confusion_matrix(yp_test, test1_predict1)[1, 0]

print(confusion_mat)
print(true1_pos)
print(true_neg)
print(false1_pos)
print(false1_neg)

1accuracy_val = (true1_pos+true_neg) / (true1_pos+true_neg+false1_pos+false1_neg)
1precision_val = (true1_pos) / (true1_pos+false1_pos)
1recall_val = (true1_pos) / (true1_pos+false1_neg)
f_1measure_val = (2*1recall_val*1precision_val) / (1recall_val + 1precision_val)

print(1accuracy_val)
print(1precision_val)
print(1recall_val)
print(f_1measure_val)
import plotly.graph_objects as go
measures = ['Accuracy', 'Precision', 'Recall', 'f_measure']
values = [1accuracy_val*100, 1precision_val*100, 1recall_val*100, f_1measure_val*100]
bar_graph = go.Figure(data=[
    go.Bar(name='Total', x=measures, y=values),
])
bar_graph.update_layout(
    title = 'Measures'
    , xaxis = dict(title = 'Measure')
    , yaxis = dict(title = 'Percentage')
    , barmode='group'
    , bargap = 0.2)
bar_graph.show()
import plotly.graph_objects as go
test_train = ['Training data', 'Testing data']
test_train_accuracy1 = [xp_train.shape[0], xp_test.shape[0]]
test_train_accuracy2 = [train_1accuracy_val*xp_train.shape[0],
test_1accuracy_val*xp_test.shape[0]]


bar_graph = go.Figure(data=[
    go.Bar(name='Total', x=test_train, y=test_train_accuracy1),
    go.Bar(name='Accurate', x=test_train, y=test_train_accuracy2)
])
bar_graph.update_layout(
    title = 'Accuracy of training and testing data'
    , xaxis = dict(title = 'Type of data')
    , yaxis = dict(title = 'Number of patients')
    , barmode='group'
    , bargap = 0.2)
bar_graph.show()
```

4

```
from sklearn import svm

svm_classifier_model = svm.SVC(kernel = 'linear')
svm_classifier_model.fit(xp_train,yp_train)

train1_predict1_mat = svm_classifier_model.predict(xp_train)

#getting accuracy for training dataset
print("Accuracy = ", accuracy1_score(yp_train, train1_predict1_mat))
train_1accuracy_val = accuracy1_score(yp_train, train1_predict1)

#Getting accuracy value for testing dataset
test1_predict1 = svm_classifier_model.predict(xp_test)
print("Accuracy = ",accuracy1_score(yp_test, test1_predict1))
test_1accuracy_val = accuracy1_score(yp_test, test1_predict1)

accuracy2 = test_1accuracy_val
confusion_mat = 1confusion_matrix(yp_test, test1_predict1)
true1_pos = 1confusion_matrix(yp_test, test1_predict1)[1, 1]
true_neg = 1confusion_matrix(yp_test, test1_predict1)[0, 0]
false1_pos = 1confusion_matrix(yp_test, test1_predict1)[0, 1]
false1_neg = 1confusion_matrix(yp_test, test1_predict1)[1, 0]

print(confusion_mat)
print(true1_pos)
print(true_neg)
print(false1_pos)
print(false1_neg)

1accuracy_val = (true1_pos+true_neg) / (true1_pos+true_neg+false1_pos+false1_neg)
1precision_val = (true1_pos) / (true1_pos+false1_pos)
1recall_val = (true1_pos) / (true1_pos+false1_neg)
f_1measure_val = (2*1recall_val*1precision_val) / (1recall_val + 1precision_val)

print(1accuracy_val)
print(1precision_val)
print(1recall_val)
print(f_1measure_val)

measures = ['Accuracy', 'Precision', 'Recall', 'f_measure']
values = [1accuracy_val*100, 1precision_val*100, 1recall_val*100, f_1measure_val*100]
bar_graph = go.Figure(data=[
    go.Bar(name='Total', x=measures, y=values),
])
bar_graph.update_layout(
    title = 'Measures'
    , xaxis = dict(title = 'Measure')
    , yaxis = dict(title = 'Percentage')
    , barmode='group'
    , bargap = 0.2)
bar_graph.show()
test_train = ['Training data', 'Testing data']
```

```
test_train_accuracy1 = [xp_train.shape[0], xp_test.shape[0]]

test_train_accuracy2 = [train_1accuracy_val*xp_train.shape[0],
test_1accuracy_val*xp_test.shape[0]]


bar_graph = go.Figure(data=[
    go.Bar(name='Total', x=test_train, y=test_train_accuracy1),
    go.Bar(name='Accurate', x=test_train, y=test_train_accuracy2)
])
bar_graph.update_layout(
    title = 'Accuracy of training and testing data'
    , xaxis = dict(title = 'Type of data')
    , yaxis = dict(title = 'Number of patients')
    , barmode='group'
    , bargap = 0.2)
bar_graph.show()
from sklearn.ensemble import RandomForestClassifier


rf_classifier_model = RandomForestClassifier()
rf_classifier_model.fit(xp_train,yp_train)
train1_predict1_mat= rf_classifier_model.predict(xp_train)


#getting accuracy for training dataset
print("Accuracy = ", accuracy1_score(yp_train, train1_predict1))
train_1accuracy_val = accuracy1_score(yp_train, train1_predict1)


#Getting accuracy value for testing dataset
```

```
test1_predict1 = rf_classifier_model.predict(xp_test)

print("Accuracy = ",accuracy1_score(yp_test, test1_predict1))

test_1accuracy_val = accuracy1_score(yp_test, test1_predict1)


accuracy3 = test_1accuracy_val

confusion_mat = 1confusion_matrix(yp_test, test1_predict1)

true1_pos = 1confusion_matrix(yp_test, test1_predict1)[1, 1]

true_neg = 1confusion_matrix(yp_test, test1_predict1)[0, 0]

false1_pos = 1confusion_matrix(yp_test, test1_predict1)[0, 1]

false1_neg = 1confusion_matrix(yp_test, test1_predict1)[1, 0]


print(confusion_mat)

print(true1_pos)

print(true_neg)

print(false1_pos)

print(false1_neg)


1accuracy_val = (true1_pos+true_neg) / (true1_pos+true_neg+false1_pos+false1_neg)

1precision_val = (true1_pos) / (true1_pos+false1_pos)

1recall_val = (true1_pos) / (true1_pos+false1_neg)

f_1measure_val = (2*1recall_val*1precision_val) / (1recall_val + 1precision_val)


print(1accuracy_val)

print(1precision_val)

print(1recall_val)

print(f_1measure_val)
```

```
measures = ['Accuracy', 'Precision', 'Recall', 'f_measure']

values = [1accuracy_val*100, 1precision_val*100, 1recall_val*100, f_1measure_val*100]

bar_graph = go.Figure(data=[

    go.Bar(name='Total', x=measures, y=values),

])

bar_graph.update_layout(

    title = 'Measures'

    , xaxis = dict(title = 'Measure')

    , yaxis = dict(title = 'Percentage')

    , barmode='group'

    , bargap = 0.2)

bar_graph.show()


test_train = ['Training data', 'Testing data']

test_train_accuracy1 = [xp_train.shape[0], xp_test.shape[0]]

test_train_accuracy2 = [train_1accuracy_val*xp_train.shape[0],
test_1accuracy_val*xp_test.shape[0]]


bar_graph = go.Figure(data=[

    go.Bar(name='Total', x=test_train, y=test_train_accuracy1),

    go.Bar(name='Accurate', x=test_train, y=test_train_accuracy2)

])

bar_graph.update_layout(

    title = 'Accuracy of training and testing data'

    , xaxis = dict(title = 'Type of data')

    , yaxis = dict(title = 'Number of patients')
    , barmode='group'
    , bargap = 0.2)
bar_graph.show()
classifier = ['Naive Bayes', 'SVM', 'Random Forest']
values = [accuracy1*100, accuracy2*100, accuracy3*100]
bar_graph = go.Figure(data=[
    go.Bar(name='Accuracy', x=classifier, y=values, width = 0.5),
])
bar_graph.update_layout(
    title = 'Accuracy comparison'
    , xaxis = dict(title = 'Accuracy')
    , yaxis = dict(title = 'Percentage')
    , bargap = 0.5
    )
bar_graph.show()
```

yuf

ORIGINALITY REPORT

12%	4%	3%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Birkbeck College Student Paper	3%
2	Submitted to University of Greenwich Student Paper	3%
3	gitlab.fdmci.hva.nl Internet Source	1%
4	Submitted to University of Warwick Student Paper	1%
5	Submitted to University of Southern California Student Paper	1%
6	www.coursehero.com Internet Source	1%
7	Submitted to National Institute of Technology, Kurukshetra Student Paper	1%
8	aplunket.com Internet Source	1%