1. **ABSTRACT**

The project focuses on a Hadoop based application which is used for the prediction of digits by using the position of pixels on a screen. The database used for the storage of the pixels is Hadoop based - HBase. Machine learning approach is used for the digit prediction where pixel position is given as the training data for the construction and learning of the machine learning model. After successful machine learning model construction, testing of the model is done using the testing data. The Random Forest algorithm is used by the application for the purpose of machine learning.

**Keywords:** Machine learning, digit, pixel, Random Forest, Hadoop, HBase.

1. **INTRODUCTION**

This project focuses on recognition of digits by considering the features, which are the positions of the various pixels on a computer screen. The phenomenon of digit recognition involves detection and recognition of digits from the input. The project predicts the digit present on the screen by making use of the pixel positions on the computer screen. It makes use of HBase database, which is Hadoop-oriented and machine learning for this prediction. The input parameters, which are the position of the pixels, are used for the Random Forest Algorithm for building the machine-learning model.

* 1. **Purpose**

• It helps greatly to the advancement of automation process and improving the interface between man and machine in many applications.

• It has an importance in several fields and it may probably be used in checks in banks or for recognizing numbers in cars plates, or many other applications.

• It provides a detailed understanding of Hadoop-oriented database- HBase and machine learning concepts regarding Random Forest Algorithm.

1. **SCOPE**

The project uses a database consisting of 709 features- position of the pixels and 1000 data rows. The scope can be extended by increasing the dataset size and by using other machine learning algorithms for obtaining the comparison results.

1. **REQUIREMENTS**

* Software Requirement:
  + Ubuntu 16.04 LTS
  + Java 8
  + Hadoop 2.7.3
  + HBase 1.2.4
  + NetBeans 8.2
  1. **NetBeans**

NetBeans is a [software development](https://en.wikipedia.org/wiki/Software_development) [platform](https://en.wikipedia.org/wiki/Platform_(computing)) written in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). The NetBeans Platform allows applications to be developed from a set of modular [software components](https://en.wikipedia.org/wiki/Software_component) called *modules*. Applications based on the NetBeans Platform, including the NetBeans [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE), can be extended by [third-party developers](https://en.wikipedia.org/wiki/Third_party_developer).

The NetBeans IDE is primarily intended for development in Java but also supports other languages, in particular, [PHP](https://en.wikipedia.org/wiki/PHP), [C](https://en.wikipedia.org/wiki/C_(programming_language))/[C++](https://en.wikipedia.org/wiki/C%2B%2B), and [HTML5](https://en.wikipedia.org/wiki/HTML5).

NetBeans is [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) and runs on [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), [Mac OS X](https://en.wikipedia.org/wiki/Mac_OS_X), [Linux](https://en.wikipedia.org/wiki/Linux), [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)) and other platforms supporting a compatible [JVM](https://en.wikipedia.org/wiki/Java_Virtual_Machine).

* 1. **Hadoop**

Apache Hadoop is an [open-source](https://en.wikipedia.org/wiki/Open_source) [software framework](https://en.wikipedia.org/wiki/Software_framework) used for [distributed](https://en.wikipedia.org/wiki/Clustered_file_system) and of [big data](https://en.wikipedia.org/wiki/Big_data) sets using the [MapReduce](https://en.wikipedia.org/wiki/MapReduce) [programming model](https://en.wikipedia.org/wiki/Programming_model). It consists of [computer clusters](https://en.wikipedia.org/wiki/Computer_cluster) built from [commodity hardware](https://en.wikipedia.org/wiki/Commodity_hardware). All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

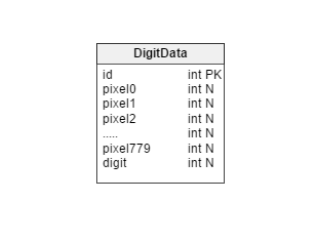
The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a [MapReduce](https://en.wikipedia.org/wiki/MapReduce) [programming model](https://en.wikipedia.org/wiki/Programming_model). Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers [packaged code](https://en.wikipedia.org/wiki/JAR_(file_format)) into nodes to process the data in parallel. This approach takes advantage of [data locality](https://en.wikipedia.org/wiki/Data_locality), where nodes manipulate the data they have access to. This allows the dataset to be [processed](https://en.wikipedia.org/wiki/Distributed_processing) faster and more efficiently than it would be in a more conventional [supercomputer architecture](https://en.wikipedia.org/wiki/Supercomputer_architecture) that relies on a [parallel file system](https://en.wikipedia.org/wiki/Parallel_file_system) where computation and data are distributed via high-speed networking.

**4.3 HBase**

HBase is an [open source](https://en.wikipedia.org/wiki/Open_source), [non-relational](https://en.wikipedia.org/wiki/Non-relational_database), [distributed database](https://en.wikipedia.org/wiki/Distributed_database) modeled [Google's](https://en.wikipedia.org/wiki/Google) [Big table](https://en.wikipedia.org/wiki/Bigtable) and is written in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). It is developed as part of [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation)'s [Apache Hadoop](https://en.wikipedia.org/wiki/Hadoop) project and runs on top of [HDFS (Hadoop Distributed File System)](https://en.wikipedia.org/wiki/Hadoop_Distributed_File_System), providing Big table-like capabilities for Hadoop. That is, it provides a [fault-tolerant](https://en.wikipedia.org/wiki/Fault-tolerant) way of storing large quantities of [sparse](https://en.wikipedia.org/wiki/Sparse_file) data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

HBase features compression, in-memory operation, and [Bloom filters](https://en.wikipedia.org/wiki/Bloom_filter) on a per-column basis as outlined in the original Big table paper. Tables in HBase can serve as the input and output for [MapReduce](https://en.wikipedia.org/wiki/Mapreduce) jobs run in Hadoop and may be accessed through the [Java API](http://hbase.apache.org/apidocs/index.html) but also through [REST](https://en.wikipedia.org/wiki/REST), [Avro](https://en.wikipedia.org/wiki/Avro_(serialization_system)) or [Thrift](https://en.wikipedia.org/wiki/Thrift_(protocol)) gateway APIs. HBase is a column-oriented key-value data store and has been idolized widely because of its lineage with Hadoop and HDFS. HBase runs on top of HDFS and is well-suited for faster read and write operations on large datasets with high throughput and low input/output latency.

1. **DATA MODELING**



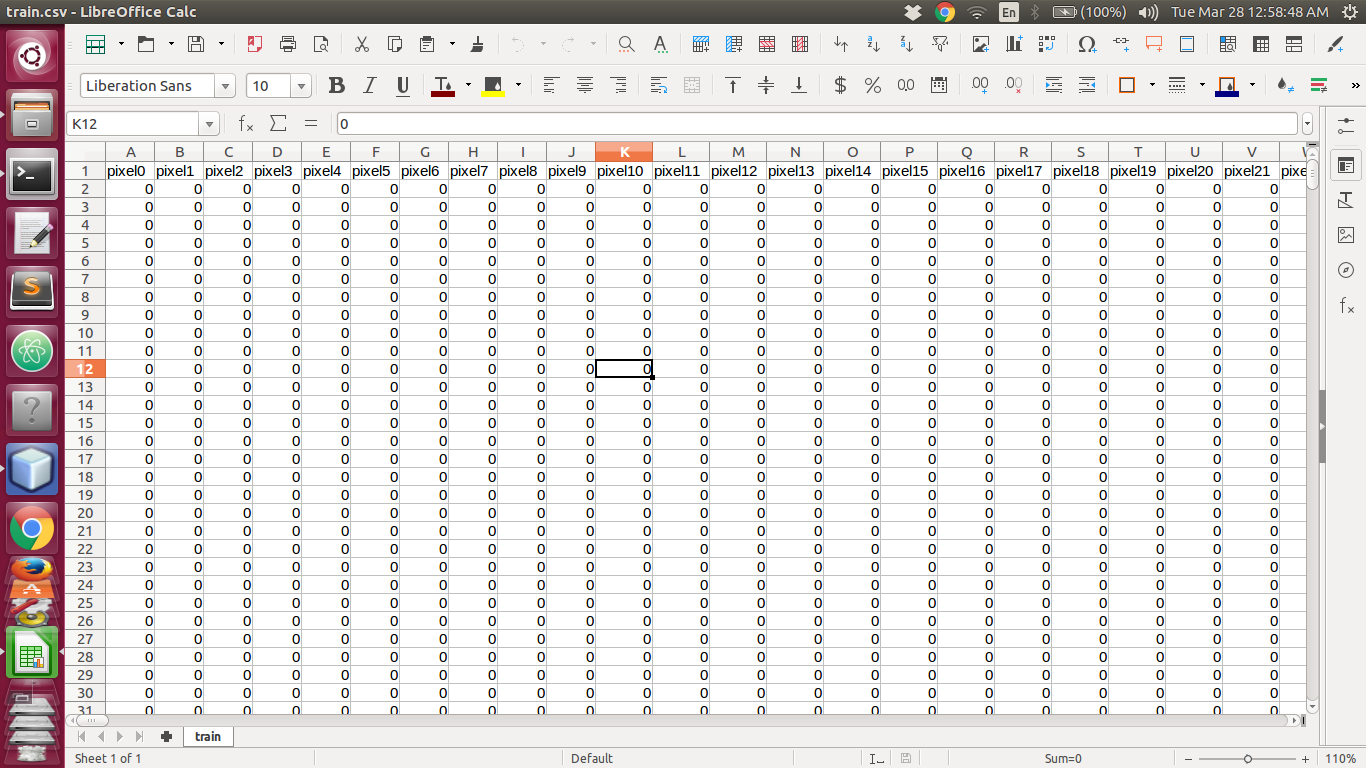
**Fig 5.1: Data Model**

Initially, the pixel position data as a .csv file is loaded into Hadoop-based database HBase. Once the data has been loaded, a retrieval of data row wise is done using an HBase based Java API. Row- wise data is then applied to the machine learning process, in which the data is divided into training and testing data. Once all the training data is loaded for the ML algorithm- Random Forest a model is generated, on which the testing data is applied. Once the model processing is done the data to be predicted is applied. The results are then represented in a command line format.

1. **DATABASE DESIGN**

**6.1 Dataset:**

The dataset for our project has been downloaded from **Kaggle.com**. The dataset is in the form of a CSV (Comma Separated Value) file. The dataset contains information about the position of various pixels from which the recognition of digits is done. The dataset has 1000 entries, which helps us to predict the digit.



**Fig 6.1: Dataset**

**6.2 Column Family**

Column families are the base storage mechanism in HBase.   An HBase table is comprised of one or more column families, each of which is stored in a separate set of region files sharing a common key. To express it in terms of an RDBMS, a column family is roughly analogous to an RDBMS table with the row key as a clustered primary key index. The column family contains all the columns of the dataset which we are going to use to classify the digit.

1. **ALGORITHM**

**7.1 Random Forest Algorithm:**

The general method of random decision forests was first proposed by Ho in 1995, who established that forests of trees splitting with oblique hyperplanes if randomly restricted to be sensitive to only selected feature dimensions, can gain accuracy as they grow without suffering from overtraining. A subsequent work along the same lines concluded that other splitting methods, as long as they are randomly forced to be insensitive to some feature dimensions, behave similarly. Note that this observation of a more complex classifier (a larger forest) getting more accurate nearly monotonically is in sharp contrast to the common belief that the complexity of a classifier can only grow to a certain level of accuracy before being hurt by overfitting. The explanation of the forest method's resistance to overtraining can be found in Kleinberg's theory of stochastic discrimination.

The early development of Berriman’s notion of random forests was influenced by the work of Amit and German who introduced the idea of searching over a random subset of the available decisions when splitting a node, in the context of growing a single tree. The idea of random subspace selection from Ho was also influential in the design of random forests. In this method, a forest of trees is grown, and variation among the trees is introduced by projecting the training data into a randomly chosen subspace before fitting each tree or each node. Finally, the idea of randomized node optimization, where the decision at each node is selected by a randomized procedure, rather than a deterministic optimization was first introduced by Dietterich.

The introduction of random forests proper was first made in a paper by Leo Breiman. This paper describes a method of building a forest of uncorrelated trees using a CART-like procedure, combined with randomized node optimization and bagging. In addition, this paper combines several ingredients, some previously known and some novel, which form the basis of the modern practice of random forests, in particular: Using out-of-bag error as an estimate of the generalization error. Measuring variable importance through permutation. Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie et al., because it is invariant under scaling and various other transformations of feature values, is robust to the inclusion of irrelevant features and produces inspectable models. However, they are seldom accurate. In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have a low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model. Each tree is grown as follows:

If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.

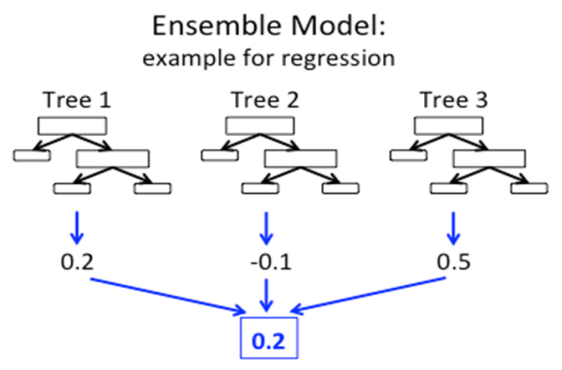
If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on this m is used to split the node. The value of m is held constant during the forest growing. Each tree is grown to the largest extent possible. There is no pruning.

**7.2 Features of Random Forest Algorithm:**

* It is unexcelled in accuracy among current algorithms.
* It runs efficiently on large databases.
* It can handle thousands of input variables without variable deletion.
* It gives estimates of what variables are important in the classification.
* It generates an internal unbiased estimate of the generalization error as the forest building progresses. **Steps:**

Draw n bootstrap samples from the original data.

1. For each of the bootstrap samples, grow a regression tree. Randomly sample the m features and then choose the best split among these attributes.
2. Make the predictions on new data by aggregating the predictions of all the n trees.



**Fig 7.1: Ensemble Model**

1. **GRAPHICAL USER INTERFACE**

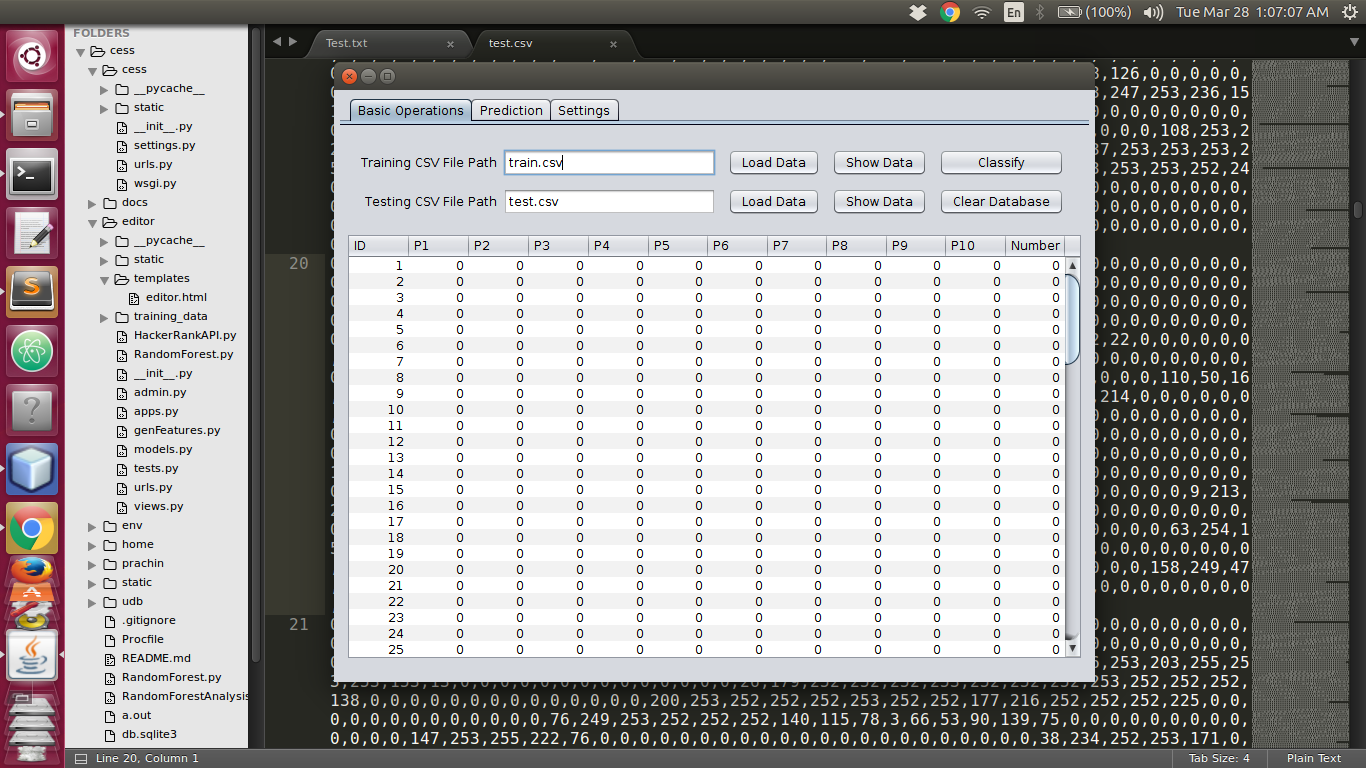


Fig 8.1: Training Data

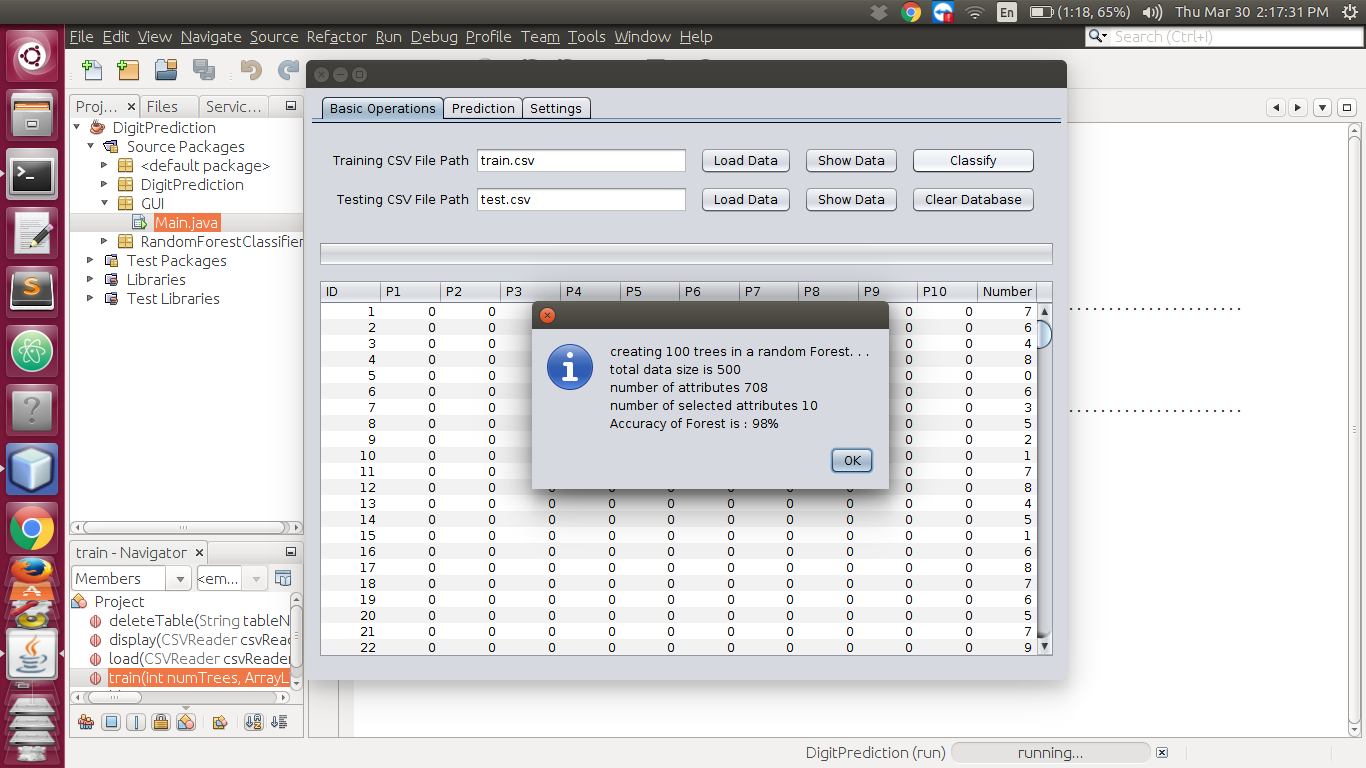


Fig 8.2: Random Forest Output

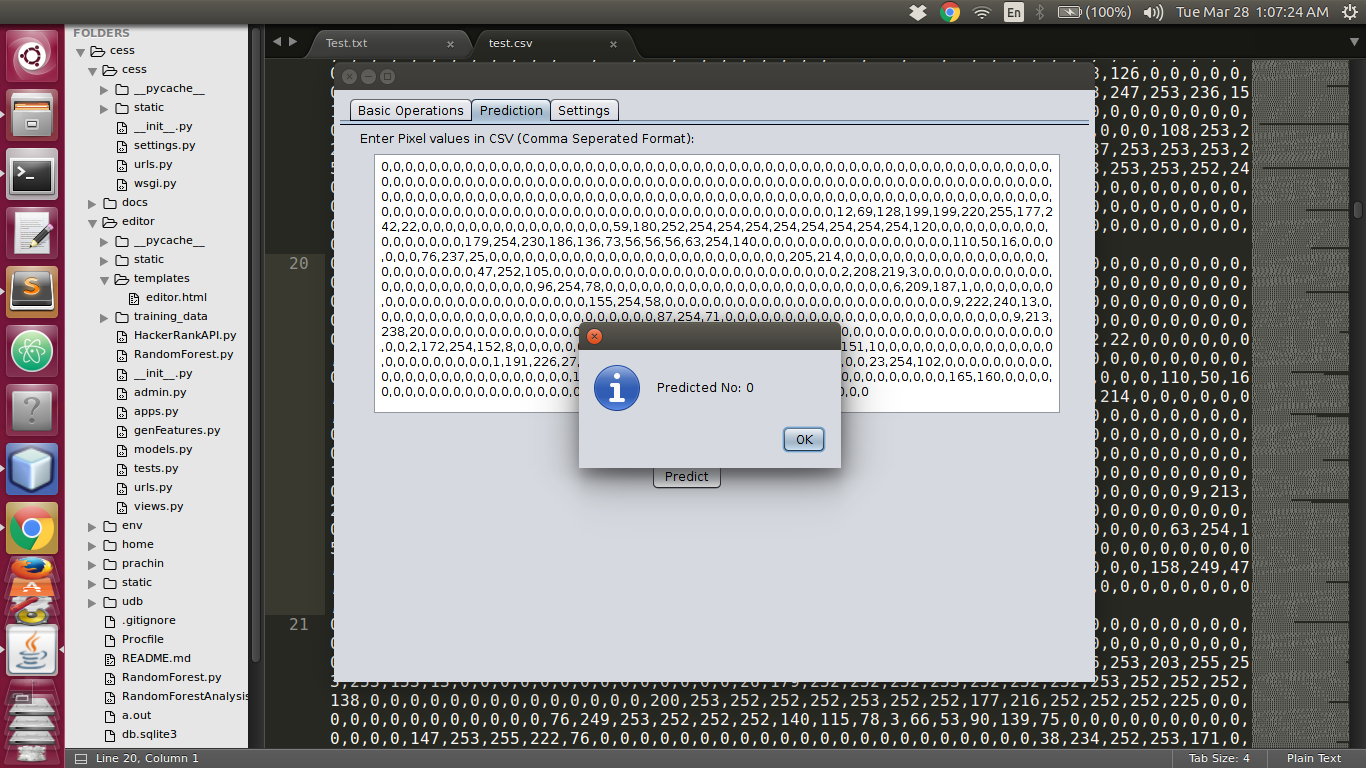


Fig 8.3: Prediction Result

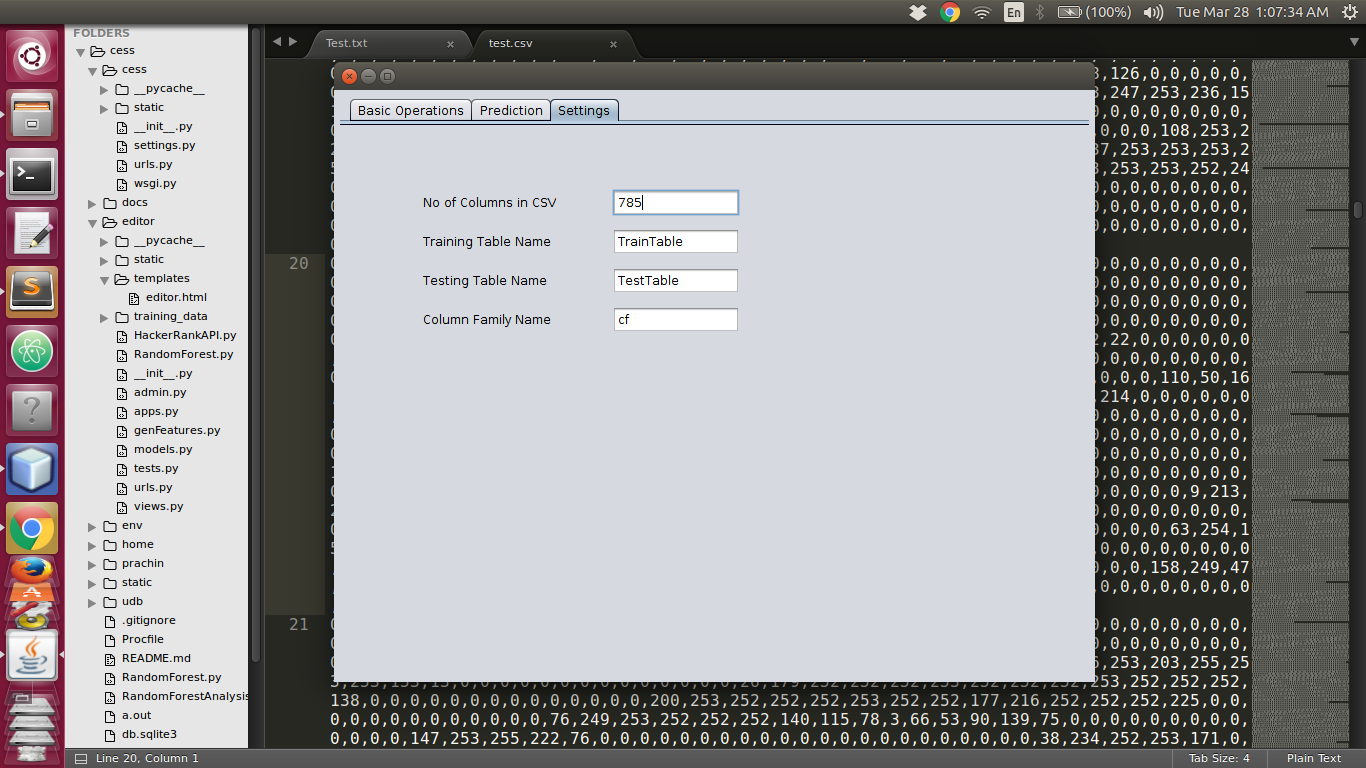


Fig 8.4: Settings

**9. SOURCE CODE**

-HBase Operations

package DigitPrediction;

import java.awt.HeadlessException;

import java.io.IOException;

import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.HColumnDescriptor;

import org.apache.hadoop.hbase.HTableDescriptor;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.HBaseAdmin;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.ResultScanner;

import org.apache.hadoop.hbase.client.Scan;

import org.apache.hadoop.hbase.util.Bytes;

public class HBase {

// Instantiating configuration class

private Configuration hconfig = HBaseConfiguration.create();

// Instantiating HbaseAdmin class

private HBaseAdmin hbase\_admin = null;

public HBase()

{

System.out.println("Connecting...");

try

{

hbase\_admin=new HBaseAdmin(hconfig);

}

catch(Exception e){ }

}

public boolean checkTableExists(String tableName)

{

try{

return hbase\_admin.isTableAvailable(tableName);

}

catch(IOException e){

return false;

}

}

public HTable createTable(String tableName,String columnDescriptors)

{

try

{

HTableDescriptor htable = new HTableDescriptor(TableName.valueOf(tableName));

System.out.println("Table: " + htable.getNameAsString() + " not found " + "now creating..");

htable.addFamily(new HColumnDescriptor(columnDescriptors));

System.out.println("Connecting...");

System.out.println("Creating Table...");

hbase\_admin.createTable(htable);

System.out.println(tableName+" created successfully!!!");

Configuration config = HBaseConfiguration.create();

return new HTable(config, tableName);

}

catch(IOException e){

return null;

}

}

public void put(HTable table,String columnFamily, String columns[], int recID, String record[])

{

try

{

for(String s:record)

if(s==null||s.equals(""))

return;

Put p = new Put(Bytes.toBytes(Long.toString(recID)));

for(int i=0;i<columns.length;i++)

p.add(Bytes.toBytes(columnFamily), Bytes.toBytes(columns[i]), Bytes.toBytes(record[i].toUpperCase().replaceAll("\"", "")));

table.put(p);

}

catch(IOException e){ }

}

public ArrayList<int[]> getAll(String tableName,String columnFamily, String columns[],DefaultTableModel tbl)

{

try

{

HBaseConfiguration config = new HBaseConfiguration(new Configuration());

HTable table=new HTable(config, tableName);

ArrayList<int[]> data=new ArrayList<>();

int row[]=new int[columns.length];

int recID=1;

Scan scan = new Scan();

scan.setCaching(50);

scan.addFamily(Bytes.toBytes(columnFamily));

ResultScanner resScan = table.getScanner(scan);

for (Result res = resScan.next(); (res != null); res = resScan.next())

{

for(int i=0;i<columns.length;i++)

{

row[i] = Integer.parseInt(Bytes.toString(res.getValue(Bytes.toBytes(columnFamily), Bytes.toBytes(columns[i]))));

}

if(tbl!=null)

{

Integer tblRow[]=new Integer[tbl.getColumnCount()];

tblRow[0]=recID++;

int i;

for(i=1;i<tbl.getColumnCount()-1;i++)

{

tblRow[i]=row[i];

}

tblRow[i]=row[tbl.getColumnCount()-1];

tbl.addRow(tblRow);

}

data.add(row);

}

return data;

}

catch(IOException e)

{

System.err.println("error");

}

return null;

}

public void deleteTable(String tableName)

{

try

{

HBaseConfiguration hconfig = new HBaseConfiguration(new Configuration());

HTableDescriptor htable = new HTableDescriptor(tableName);

System.out.println("Connecting...");

HBaseAdmin hbase\_admin = new HBaseAdmin(hconfig);

System.out.println("Dropping Table...");

if (hbase\_admin.isTableAvailable(htable.getName()))

{

System.out.println("table " + htable.getNameAsString() + " found " + "now deleting..");

hbase\_admin.disableTable(htable.getName());

hbase\_admin.deleteTable(htable.getName());

System.out.println("Deleted!");

}

else

{

System.out.println("table " + htable.getNameAsString() + "not found");

}

}

catch (IOException | HeadlessException e) { }

}

}

1. **TESTING DOCUMENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Description** | **Expected Output** | **Output** | **Remarks** |
| TC-1 | Check if table exists in HBase | True | True | Pass |
| TC-2 | Table creation test | TestTable | TestTable | Pass |
| TC-3 | Table deletion test | False | False | Pass |
| TC-4 | Test dataset column count | 709 | 709 | Pass |
| TC-5 | Test dataset data column count | 709 | 709 | Pass |

10.1 Testing Document

1. **CONCLUSION**

We have implemented the random forest algorithm for applying machine learning approach to the pixel position data for recognizing the digits present on the screen.