A DISSERTATION

ON

Malware Detection using Machine Learning

Submitted in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY
In
SOFTWARE TECHNOLOGY

Submitted by
Vijay Kumar Gupta
University Roll No. 2K16/SWT/517

Under the Esteemed Guidance of
Dr. Kapil Sharma
(Professor - Department of Information Technology)

2016-2019

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY,
DELHI– 110042, INDIA

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

DECLARATION

I hereby declare that the thesis entitled “**Malware Detection using Machine Learning”** which is being submitted to the **Delhi Technological University**, in partial fulfilment of the requirements for the award of degree of **Master of Technology in Software Technology** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

DATE:

[illegible]

SIGNATURE:

Vijay Kumar Gupta
2K16/SWT/517

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

CERTIFICATE

This is to certify that thesis entitled “**Malware Detection using Machine Learning”,** is a bonafide work done by Mr. Vijay Kumar Gupta (Roll No: 2K16/SWT/517) in partial fulfilment of the requirements for the award of **Master of Technology Degree in Software Technology** at Delhi Technological University, Delhi, is an authentic work carried out by him under my supervision and guidance. The content embodied in this thesis has not been submitted by him earlier to any University or Institution for the award of any Degree or Diploma to the best of my knowledge and belief.

DATE:

[illegible]

SIGNATURE:

Dr.Kapil Sharma
Project Guide,
DEPARTMENT OF INFORMATION TECHNOLOGY.
DELHI TECHNOLOGICAL UNIVERSITY, DELHI 110042

ACKNOWLEDGEMENT

I am presenting my work on "**Malware Detection using Machine Learning"** with lot of pleasure and satisfaction. I take this opportunity to express my deep sense of gratitude and respect towards my guide **Dr. Kapil Sharma.** I am very much indebted to her for her generosity, expertise and guidance I have received from his while working on this project. Without her support and timely guidance the completion of the project would have seemed a far –fetched dream. In this respect I find myself lucky to have my guide. She have guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation. Besides my guides, I would like to thank entire teaching and non-teaching staff in the Department of Computer Science & Engineering, DTU for all their help during my tenure at DTU. Kudos to all my friends at DTU for thought provoking discussion and making stay very pleasant. I am also thankful to the SAMSUNG who has provided me opportunity to enroll in the M.Tech Programme and to gain knowledge through this programme. This curriculum provided me knowledge and opportunity to grow in various domains of computer science.

[illegible]

Vijay Kumar Gupta
2K16/SWT/517

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

DECLARATION

I hereby declare that the thesis entitled "**Malware Detection using Machine Learning"** which is being submitted to the **Delhi Technological University**, in partial fulfilment of the requirements for the award of degree of **Master of Technology in Software Technology** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

DATE:

Gupta.

SIGNATURE:

Vijay Kumar Gupta
2K16/SWT/517

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

CERTIFICATE

This is to certify that thesis entitled "**Malware Detection using Machine Learning",** is a bonafide work done by Mr. Vijay Kumar Gupta (Roll No: 2K16/SWT/517) in partial fulfilment of the requirements for the award of **Master of Technology Degree in Software Technology** at Delhi Technological University, Delhi, is an authentic work carried out by him under my supervision and guidance. The content embodied in this thesis has not been submitted by him earlier to any University or Institution for the award of any Degree or Diploma to the best of my knowledge and belief.

DATE:

SIGNATURE:

Dr.Kapil Sharma
Project Guide,
DEPARTMENT OF INFORMATION TECHNOLOGY.
DELHI TECHNOLOGICAL UNIVERSITY, DELHI 110042

ACKNOWLEDGEMENT

I am presenting my work on "**Malware Detection using Machine Learning"** with lot of pleasure and satisfaction. I take this opportunity to express my deep sense of gratitude and respect towards my guide **Dr. Kapil Sharma.** I am very much indebted to her for her generosity, expertise and guidance I have received from his while working on this project. Without her support and timely guidance the completion of the project would have seemed a far –fetched dream. In this respect I find myself lucky to have my guide. She have guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation. Besides my guides, I would like to thank entire teaching and non-teaching staff in the Department of Computer Science & Engineering, DTU for all their help during my tenure at DTU. Kudos to all my friends at DTU for thought provoking discussion and making stay very pleasant. I am also thankful to the SAMSUNG who has provided me opportunity to enroll in the M.Tech Programme and to gain knowledge through this programme. This curriculum provided me knowledge and opportunity to grow in various domains of computer science.

Gupta.

Vijay Kumar Gupta
2K16/SWT/517

Abstract

In this challenge, we are aiming to classify tens of thousands of malware files into families and using different machine learning models with some improvisation done in the modelling and training of data. The dataset used consisted of large proportions of .bytes and .asm files having around 11,000 malware files for test set and training set. Strategically doing the exploratory analysis, which included evaluating the distribution, feature extraction, multivariate analysis and data splitting, we have tried to compare the models and optimize the best one using new techniques to enhance the efficiency of the task. The task was both separately and conjunctively performed over the byte and asm files and effective analysis was made. Some new insights like visualization and effective feature engineering are also mentioned and appreciated that would further refine the accuracy of the process.

We have used a large dataset released by Microsoft in this work for both training and testing purposes. Training data set has 10868 sample malwares from 9 different classes of malware. Classes of malware are:

(a) Vundo (b) Lollipop, (c) Simda, (d) Ramnit, (e) Kelihos_ver3, (f) Obfuscator.ACY, (g) Kelihos_ver1, (h) Tracur, (i) Gatak.

The malware data set is almost half a tera-byte when it is un-compressed. The data-set consists of a set of known malware files representing a mixture of 9 different family. Each of the malware file has its own identifier , 20 character hash-valued uniquely identifying given file , & class label , that is integer represents the 1

of the nine family names to which the malwares might belonged to. For each of the given file, the raw-data contained the hexadecimal representation of the files binary (01) content , with-out the header (to ensure sterility). The given data set also includes a metadata manifest, that is the log containing various metadata information what is taken out from the binary (01), just like function calls, strings , etc. That was generated using the IDA disassembler tool.

The dataset is first loaded and then it is saved in memory for further transformations.

For each of the malware, we have 2 files - .asm file and .bytes file (and where given file binary content is represented in hexadecimal representation in raw data, but with out the P.E. header).The size of .bytes files are 10,868 and 10,868 asm files making a total of 21,736 files .

Introduction

Industry of Malwares has grown very quickly in the past few years that, to evade traditional protection the syndicates invest heavily in technologies, forcing the anti malware groups / communities to build more complex softwares to identify and terminate these attacks. Identifying whether a given content of any specific file or software is a malware or not is the first and important step of saving a computer system from a malware attack.

1.1 Background

The background to understand the need for M. L. methods in malware detection. First malware types are described then it is followed by the standard malware detection methods. After that, the need for ML is discussed, based on the knowledge gained, along with the relevant work performed in this field.

1.2 Malware types

It is useful to classify malwares to have a good understanding of the methods and logic behind it. Depending on its purpose malwares can be divided into several classes. The classes are as mentioned in below:

- Virus: -- This malware is a piece of code which modifies the attacked file in such a way that when original file/application is executed it also executed. Basically it insert itself in an application/file and execute when application runs.

  It is the simplest form of malware.

-Worm. This malware has a distinctive trait of self replication. Unlike Viruses it doesn’t require any end user to kick it off. It’s ability to spread without any end-user action makes it so devastating.

-Trojan. This malware class masquerade itself as a legitimate program but they contain harmful instructions. Due to that, the general spreading-vectors that utilized in this class called social engineering . Trojans require execution by it’s end user.
-Adware. This type of malware is generally used to display advertisements on the computer. Generally this could be seen as a sub--class of spy ware. Very unlikely it will lead to dramatic results.
-Spyware: --. As it can be implied from its name itself, it is the type of malware which performs espionage . Generally this type of malware are used to track search history and based on that send personalized advertisements, also selling data to the third parties.
-Rootkit. This type of malware are used to enable the attacker to access the data with higher permission compared to what is allowed. For example, it can be used in giving administrative access to an unauthorized user. Rootkits quite often are unnoticeable and always hide its existence on the system, thus it makes the detection and removal of them extremely hard.
- Backdoor: --.This is the type of malware which provide an additional undisclosed “entrance” to the computer for its attackers. Itself that is not harmful but it helps in providing broder surface to the attackers. So basically these are not used-independently, they are preceding of some other type of malware attack.
- Keylogge.: -- .This type of malware is used to track all of the keys that typed by any person, so, store all private data, thta includes passwords & other sensitive information .
-Ransomware : -- . As it’s name indicates these types of malwares are aims to encrypt all user data on his computer and then it asked that perosn to transfer huge money (Ransom) to find the unlock password. Usually, a machine get frozen infected by ransomware as all files are encrypted and user can not open any file , and on attacker’s demands desktop picture is used to provide required informations.

-Remote Administration Tools (RAT): --. These types are used to do possible modification as if they were accessed physically by allowing an attacker to gain the access to the given system. For example Team-Viewer, but with malicious intentions.

1.3 Detection methods

Every technique to detect the malware could be further expressed into 2 categories. Signature - based & behaviour - based method. This is very important to understand about the fundamental of these 2 malwares analysis approaches: i.e. static one and dynamic one malware analysis , before going into these methods. As it name implies, static analysis is performed with out execution of given file i.e. “statically”. On other hand, dynamic analysis is performed on the executing file for an e. g. in virtual machines.

Static analysis : would be seen as predicting the behavioral properties of the file by “reading the source code” of themalware. It can include various techniques:

File Format Inspection: Useful information can be provided by file matadata. For example, much information can be provided by Windows PE (portable executable) on compiled time, imported & exported functions etc.

String Extraction : this refers to inferring information about the malware operation by examinations of given SW output (example -status or any error message).

Fingerprinting : this techniques includes the cryptographic based hash computation, & find the environmental artifacts , like hard-coded user-name, file-name, or any registry strings.

AV scanning : if it is a case that inspected file is a quite known malware , mostly it can be detected by all anti-virus scanners. Although this may seem irrelevants, that way was generally used by A.V. sandboxes or vendors to “confirm their results”.

Disassembly : Disassembly belongs to inferring the software logic and intentions by reversing comletely the given machine code to ASM language & that is the very much common and useful method in static analysis.

Often, “Static analysis” id done using certain tool & beyond the very simple analysis, information 0n protection techniques used by malware can be provided by them. An ability to discover all feasible behavioural scenarios is of main importance in static analysis. Researching its code allowed all of researcher to find all possible methods of malware execution this is not only limited to this present situation. As the content of given file data could not executed and it could not result in bad consequences for entire given system, these types of analysis are safer than the dynamic analysis, also Static analysis are much more time eating. Due to all the reasons in real world dynamic environments it is generally not in use, such as anti virus systems, but it is often used for research purposes, for example in the zero day (0D) malware signature development procedure.

Dynamic Analysis (DA). In contrast to the static analysis (SA), in D.A the behaviour of the given file is measured while these are executing & the intentions and properties of the current file are inferred from extracted information. Generally, that file was being run in this virtual environment , for e.g. in the sand-box . In this type of analysis , this was possible to figure out all the behavioural attributes, like , created mutexes opened files, etc. Basically, it is quite fast than that of SA (static analysis). On the different side, behavirial scenario relevant to the current system properties could be seen by SA (static analysis). For example, the result might vary from the malware running in the environment of Windows 8.1 than to given VM (virtual machine) on which Windows 7 is installed,

Now, detection methods can be defined having the back-ground on malware analysis. The **signature based analysis** which relies on predefined signatures is a static method. This can be file fingerprint, example SHA1 hashes or MD5, file metadata, static strings. This scenario of detection, in this case, will be as follows:- the file arrived in this system, is

statically analyzed by the antivirus SW and an alert will be triggered, stating that this file is suspicious, if any of the signatures is matched. As in our case all the known samples of malware could be detected on basis of hash values, very often this kind of analysis is enough. However, attackers are now developing malware in a way which could change its signature. This feature of malware is referred as polymorphism. Truly, using only signature based detection techniques , such malware could never be found. Moreover, until the signatures are created, naval malware types could not be detected using the same signatures. Therefore, alternate way of detection required for A.V. Vendors :– behaviour based also referred to as **heuristics basedanalysis**. Real behaviour of any malware could be found during their execution in the method, looking for the malicious signs of behaviour, modifying givne registry keys, host files , establishing doubtful connections. By themselves, every one of the actions could not be a good symptom of malware , but their combination could increase the level of suspiciousness of a given file.
Any of the malware exceeding defined threshold level causes an alert.

The accuracy of heuristics based detection method are highly depends on its implementation. Virtual environment is utilized by the best ones, for example. the sand-box to execute the file and monitored given behaviour. Since before actual execution of the file, it is checked, it is much safer, Although this method is more time consuming . Behaviour based detection method could not identify only the known “malware types” but also ‘zero day attacks’ and ‘polymorphic viruses’ . However, in practice, such analysis could not be considered effective against new or polymorphic malware, considering the fast spreading rate of these malwares.

1.4 Need for machine learning

We had noted before, malware detectors which are primarily based on signatures may perform properly on ancient-types malware that were already found by any of antivirus vendors. By the way, it could not discover polymorphic malware , which has a great ability to vary its signatures , and the noval malware, for whom signatures1 has now not been created yet . In fact, the correctness of heuristics based detectors is not always adequate for correct detection, resulting in lots of false +ve and false -ves. There for need for the noval detection mechanism is dictated by the high spreading rate of polymorphic viruses. This problem can be solved by relying on the combination of heuristics based analysis and ML (machine learning) methods, that offers a good efficiency during the process of detection

When 'heuristics based approach' is relied on, a certain threshold for malware triggers has to be there, the amount of heuristics needed for the software is defined and is called malicious . For e. g. a set of suspicious features , such as permission changed, connection established, registry key changed etc. could be defined. Any SW that has triggered at least 5 features from the set could be called malicious . It is not always correct, although that method gives us some level of effectiveness & that was not accurate as some of the features could have more "weight" than other features, for an e. g., a permission changed usually ends with high severe impacts to the system rather than registry key changed. Moreover, feature combinations sometimes ismore suspicious than features. These correlations could be taken into consideration and for more accurate detection, ML 'machine learning' methods will be provided .

Objective

- Whether a given file/software is malware or not is detected

- A file may be detected as malware then it is classified into nine different categories
    - Lollipop
    - Vundo
    - Ramnit
    - Kelihos_ver3
    - Simda
    - Gatak
    - Tracur
    - Kelihos_ver1
    - Obfuscator.ACY
- Multiclass error are minimized
- Multiclass probability estimates for each malware is calculated
- Output is given by our model in few seconds or a minute

1.1 Outcome Expected

Input can be any file/software after converting it into .byte or .asm file. Project outcome is focused on finding the probability of that file belonging to given nine classes:

- Simda
- Ramnit
- Gatak
- Kelihos_ver3
- Kelihos_ver1

- Vundo
- Lollipop

- Obfuscator.ACY
- Tracur

Chapter Two: Review of Literature

The notion of strategies of computing device studying for detecting malware is no longer new however it is no longer extensively implemented. Studies of special sorts had been carried out in this field, which focused to parent the accuracy of exclusive methods

DragosGavrilut,in his paper 'Malware Detection Using Machine Learning' , which is aimed for creating a detection gadget based totally on quite a few modified perceptron algorithms . For one of a kind algorithms, the accuracy of 69.90%- 96.18% used to be achieved. Algorithm that produced the best range of false-positives additionally resulted in exceptional accuracy: the most correct one resulted in forty eight false positives. Algorithm with balanced accuracy and the low false +ve charge have the accuracy of 93.01% . ("Gavrilut" , et al. 2009).

The detection method is based on modified RF algorithm (Random Forest algorithm) along with "Information Gain" for better feature representation are discussed in this paper - 'Malware Detection Module using Machine Learning Algorithms to assist in Centralized Security in Enterprise Networks '. This data set is consists of portable executable files, for which feature extraction is generally easier, this point should be noted. The accuracy of 97% and 0.03 false positive rate is achieved as result ("Singhal and Raul" 2015).

Extraction methods proposed in "A Static Malware Detection System by using some Data Mining Methods" was based on DLLs, P.E. headers , methods and API functions that are based on Naive Bayes algorithms, Support Vector Machines and J-48 Decision Trees and. maximum overall accuracyof 99% with P.E. header feature type and hybrid P.E. header & API's function feature type, 99.1% with API's function feature type was achieved with the J48 algorithm (Baldangabo, Jambaljav and Horng 2013).

The API features had been used for function illustration in "Zero-day Malware Detection primarily based on Supervised Learning Algorithms of API name Signatures". Support Vector Machines algorithm with normalized polykernelachieved the first-rate result. This executed the precision of 97.6%, alongside with a falsepositive price of 0.025. (Alazab, et al. 2011).

All research ended up with special outcomes can be seen . No unified methodology used to be created but neither for detection nor characteristic illustration can be concluded. Each separate case accuracy relies upon on the specifics of malware households used and on the true implementation .

Chapter Three: Technical Approach

An in depth understanding can be achieved, by walking through the general workflow of the ML “machine learning” process as given in Diagram 1.

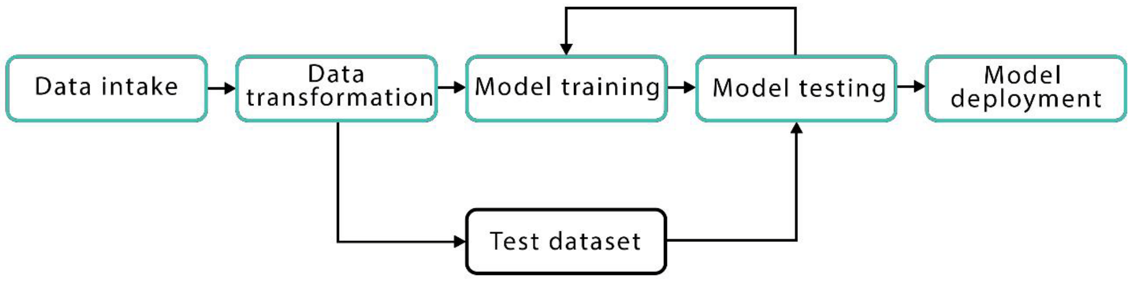


Figure 1 General workflow process

It is a 5 stage process as can be soon:

1. **Data intake** : First, we load the data-set from the fileand save it into a memory.
2. **Data transformation** . Now, after this step the data of step 1 has been cleared, transformed , and normalized to be suitable for the algorithm . Converted data lies in the given range, and have the exactly same format , etc. Feature extraction and selection, that are discussed further, are performed at this stage. Moreover, the data could be separated in-to the sets called “training set” & “testing set”. “Training set data” are used to make the model , which is later evaluated using the testing set .
3. **Normalization** For e.g. of normalization could be divide an image z, where z is the number of pixels with colour j, by the total number of count to “encode” the distribution and withdraw the dependency on the “Image size”. This translates in the given formula

4. **Standardization** Sometimes, even when comparable objects are refferd, features can have different scales. For instance, housing prices example can be considered. In this case, feature "room size" is a integer, that is not more than five and feature "house size" is calculated in sq meters . Although comparison can be done of both values and could be multiplied, added etc., the outcome would be unreasonable before normalization. The given scale value is mostly used: $x'_i = (x_i - \mu_i) / \sigma_i$, where μ_i and σ_i be the mean value & the std. deviation of feature x_i over training examples.
5. **Feature Extraction** Mentioned above instances, attributes from the provided data should be extracted, so could be fed to the given formula. For instance, for case of housing prices, multidimensional matrix could be used to represent that, in this each column can represent an attribute and rows tell us the numerical values for those attributes. Data asan RGB value of each pixel can be represented in this case

These attributes would be referred to the features , and the matrixes are termed as feature vector . Feature extraction is the process of extracting data from given set. A set of non-redundant and informative data could be obtained as a goal of feature extraction. To get knowledge, that features must represent the relevant and important information about given dataset is important because ac accurate prediction cannot be made without it. Non-obvious task of feature extraction requires a lot of research & testing . General methods apply here poorly as it is domain specific.

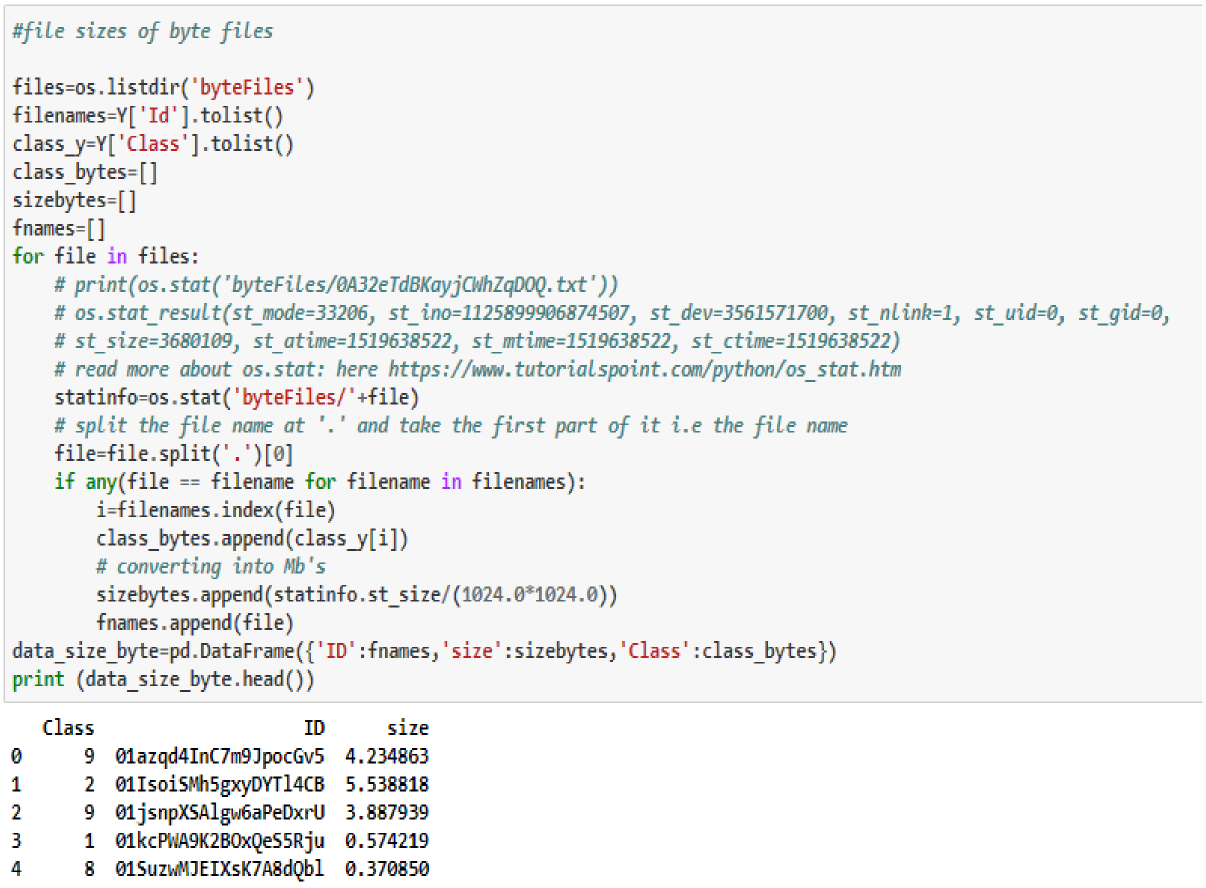
Non-redundancy is another urgent requirement for a decent feature set is. Algorithm can become biased because of features that are redundant that is features that outlined the redundant information , as well as the same information features, what are very closely inter-related on each other , hence, inaccurate result are provided

Moreover, for the input data that is quite large to be enter into our algorithm (which means it have large amount of features), must be transformed to a shrink feature vector fifteen (vector , having a fewer features number). Feature selection is the method where the vector dimensions are reduced. After this process, select the features to outline the relevant information from the initia set are expected so that instead of initial data it can be used without any loss of accuracy.

- **Featureextraction:**

3.2. Feature extraction

3.2.1 File size of byte files as a feature



```python
#file sizes of byte files

files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

```
   Class                    ID      size
0      9  01azqd4InC7m9JpocGv5  4.234863
1      2  01IsoiSMh5gxyDYTl4CB  5.538818
2      9  01jsnpXSAlgw6aPeDxrU  3.887939
3      1  01kcPWA9K2BOxQeS5Rju  0.574219
4      8  01SuzwMJEIXsK7A8dQbl  0.370850
```

- **Box plot of file size as feature.**

3.2.2 box plots of file size (.byte files) feature

```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

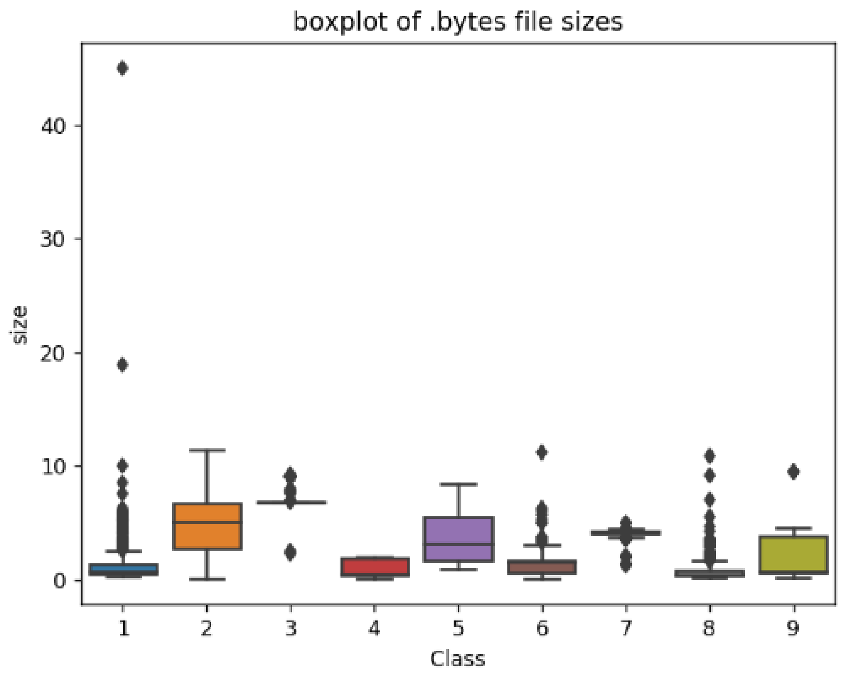


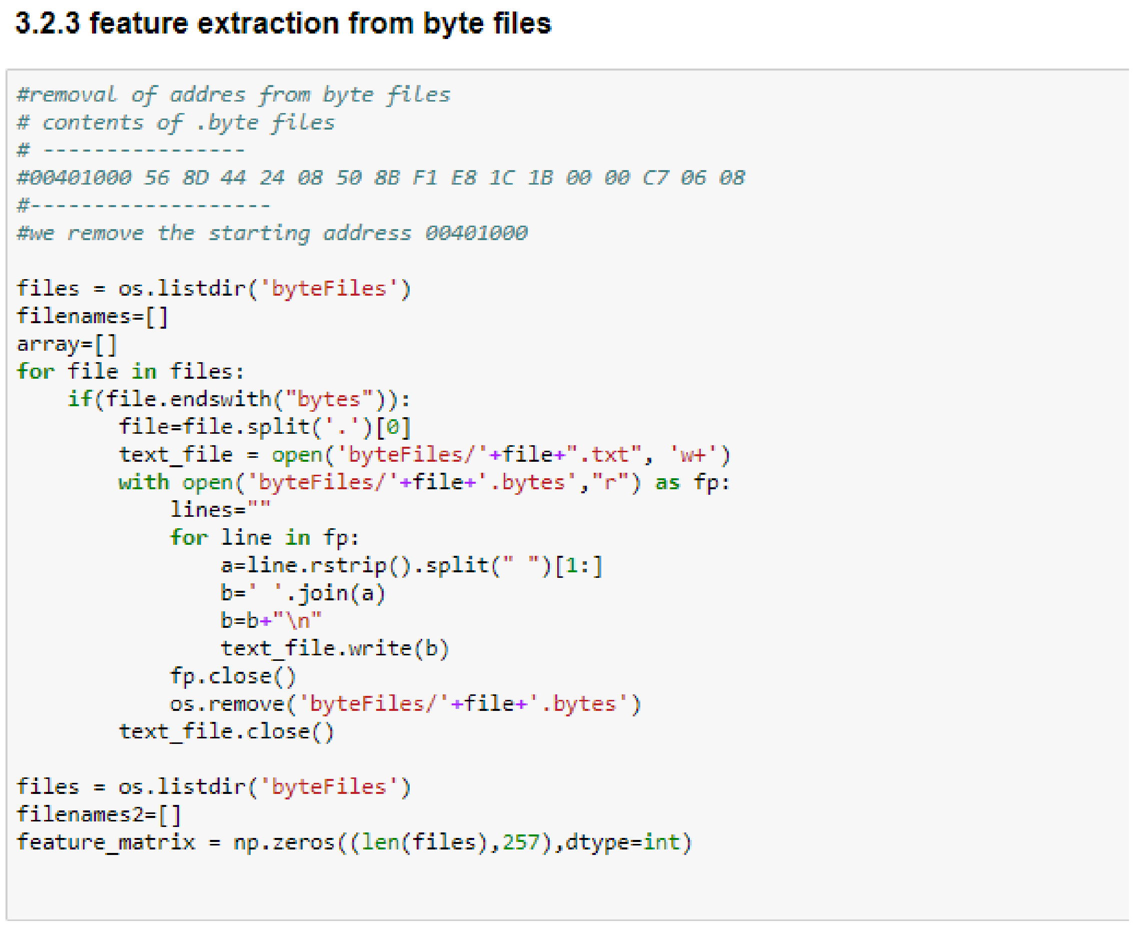
Figure 2

Extracting features from byte file:

In Feature extraction section, we will store all the hexadecimal code in a text file after removing line number form byte file.

After this we will keep track of count of hexadecimal code that is repeating in a particular file and store it in bag of words. This acts as feature of each file. After the feature extraction process is completed we will save it in a file named “result.csv”.

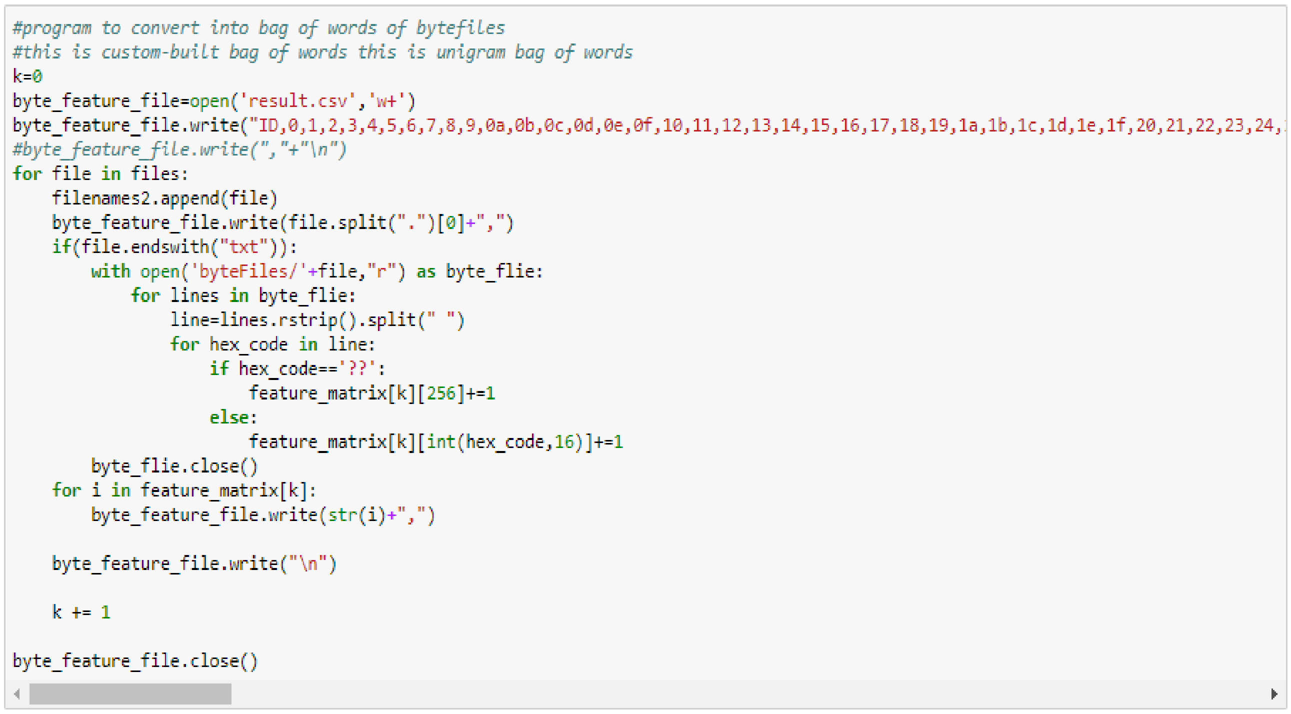
Part 1:



3.2.3 feature extraction from byte files

```python
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        with open('byteFiles/'+file+'.bytes',"r") as fp:
            lines=""
            for line in fp:
                a=line.rstrip().split(" ")[1:]
                b=' '.join(a)
                b=b+"\n"
                text_file.write(b)
            fp.close()
            os.remove('byteFiles/'+file+'.bytes')
        text_file.close()

files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((len(files),257),dtype=int)
```

Part 2:

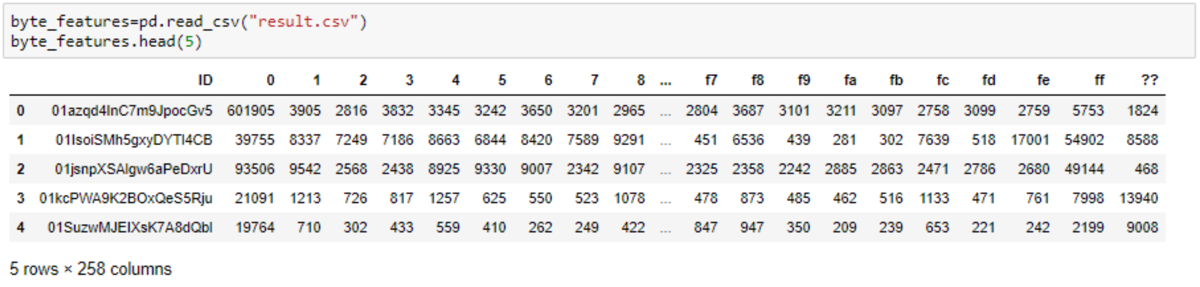


```python
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
k=0
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,
#byte_feature_file.write(","+"\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file.split(".")[0]+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")

    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

Output of extracted features:

The above code loads data from result.csv and prints it .

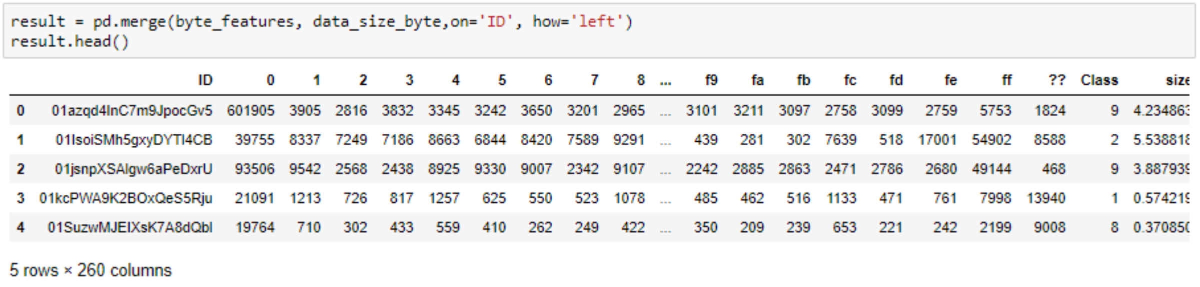
In the second snippet we also add file size as one of the features of the file.



```python
byte_features=pd.read_csv("result.csv")
byte_features.head(5)
```

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8	f9	fa	fb	fc	fd	fe	ff	??
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687	3101	3211	3097	2758	3099	2759	5753	1824
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536	439	281	302	7639	518	17001	54902	8588
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2325	2358	2242	2885	2863	2471	2786	2680	49144	468
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	478	873	485	462	516	1133	471	761	7998	13940
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	847	947	350	209	239	653	221	242	2199	9008

5 rows × 258 columns



```python
result = pd.merge(byte_features, data_size_byte,on='ID', how='left')
result.head()
```

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff	??	Class	size
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	5753	1824	9	4.234863
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001	54902	8588	2	5.538818
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2242	2885	2863	2471	2786	2680	49144	468	9	3.887939
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	485	462	516	1133	471	761	7998	13940	1	0.574219
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	350	209	239	653	221	242	2199	9008	8	0.370850

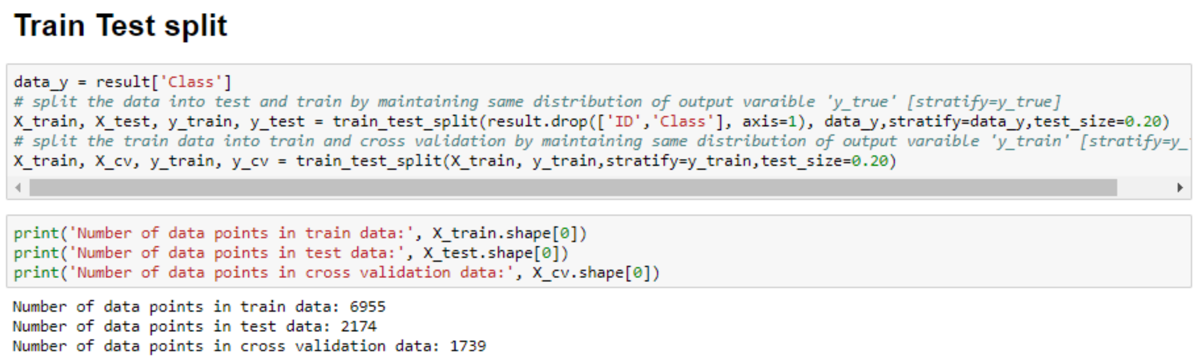
5 rows × 260 columns

3.1 ModelTraining

In this section we first split the data into two sets i.e. train and test dataset. Further we split train into two parts as actual train dataset and cross validation dataset. Cross validation dataset is used to tune the hyper parameters such as number of neighbours in KNN and depth of tree in random forest.

Below code snippet split dataset into train and test dataset.

Code:



Train Test split

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

3.2 Supervised Learning

A. **Regression** From previous observations, it predicts the next value i.e. values from training dataset. Simply, we could say that if the output is a number or in continuous form, this can be called a “regression problem”.

B. **Classification** On the basis of given set of label data , where each one of label represents a class, which this sample belongs to, we wanted to find out that class for the previously unknown sample . Given sets of possible outputs are finite in number and generally small. Usually, we could say that if given output is a categorical / discrete variable, then we can term it as a “classification problem”.

1. K- nearest neighbours

K-- Nearest Neighbours' (K--NN) is one of the most straightforward , however, exact ML machine learning algorithm. K--NN is a not a parametric algorithm, implying that it did not create any presumptions about the information structure. Actually In these real senarioes problems , provided data hardly obeys the general theoretical assumptions, that made this non parametric algorithms to find suitable solution for these problems. We could represent K--NN model in simple way as in this case data-set, requires no learning mechanism and the entire training set is stored . K--NN would be used for both types of classification and regression problems. Inside both of the problems, prediction is based on the 'K' training instances that is close to the input instance . In the K--NN problem of classification, the output was a class , to what the input instance belonged, predicted by the majority vote of the 'K' -- closest neighbours . In the problems of regression , the output is the property value , that is basically a mean value of the *'K'* -- nearest neighbours . E.g. isshowed in Diagram 2.

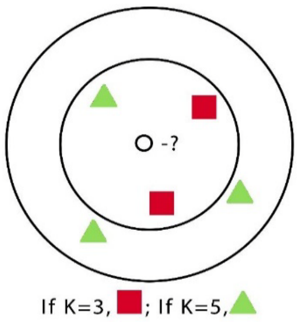


Figure - 3

The value of 'K has a important factor in accuracy of the algorithm for prediction. Moreever, selecting the 'K' value is a not a 'trivial task'. If data set has much noise then smaller values of k will give lower accuracy, since every instance of the training set now had a larger weight during the decision process . Large value of 'K' -- gives lower accuracy. In addition to this, if the value is quite high , the model could over fit, making the class boundaries less distinct and results in lower accuracy.

For classification problems with the even number of classes , We have to choose an odd k-- to avoid the possibility of a tie. The major drawback of given K--NN algorithm could be the poor performance on the unevenly distributed datasets . Therefore, if one of the classes vastly dominates the other class then this is more likely to had large neighbors of that class due to its large number, &, so, can make in-correct predictions .

2.RandomForest

Among all algorithms of machine learning, this one is very much popular. The results are usually accurate with very less preparation of data and modeling. Random Forests are based on the decision trees. Random forest produce more accurate prediction results, because this algorithm is collections of another algorithm called decision trees. This is the reason it's called "forest" – primarily it is a set of decision trees. Dividing dataset independently into subsets & growing several decision trees based on them, is the fundamental idea. At every node, some(n)variables from the feature set. This selection of variables is random. Now, on these selected variables, algorithm finds best split. In simplerterms, the algorithm can be described as follows:

-On approximately two-third of the training data is (62.3%), algorithm builds multiple trees. Data selection is random in this process.

-From all predictor variables, multiple predictor variable selection is done randomly.After that, to split each node on these picked variables, best split is used. By default, the number of the predictor variables selected, is the square root of the total number of all predictors for classification, and it is constant for all trees.

-Calculation of rate of misclassification is done by using remaining data. Calculation of the total error rate is done as the rate of overall out-of-bag error.

-Classification result is given by each tree that is trained, giving its own

“Vote”. Result is chosen by selecting the class which has received most “votes”.

Just like decision trees, in Random Forest algorithm, irrelevant features won’t be accountable in any case, Random Forest removes feature selection need for it. Sometimes feature selection is needed in Random Forest algorithm, this is in case where dimensionality reduction is needed. Random Forest’s own cross-validation method is considered to be the out-of-bag error rate, which we mentioned earlier. Difficult cross-validation measures are removed due to this, which would have to be taken otherwise. Many advantages are inherited by Random forest, of the decision tree algorithms. Both problems:classification and regression, are applicable to them; they can be easily computed and fitted quickly. They also usually result in the better accuracy. One cannot interpret results very easily unlike decision tress. In decision trees, valuable information can be extracted of important variables and their affection in the result. This is not possible with random forests. Decision tress might also be descried as less stable than Random Forests – by modifying data a little bit, accuracy could be reduced due to change in decision trees. This will never occur in the random forest algorithms – it remains stable since it is the combination of many decision trees.

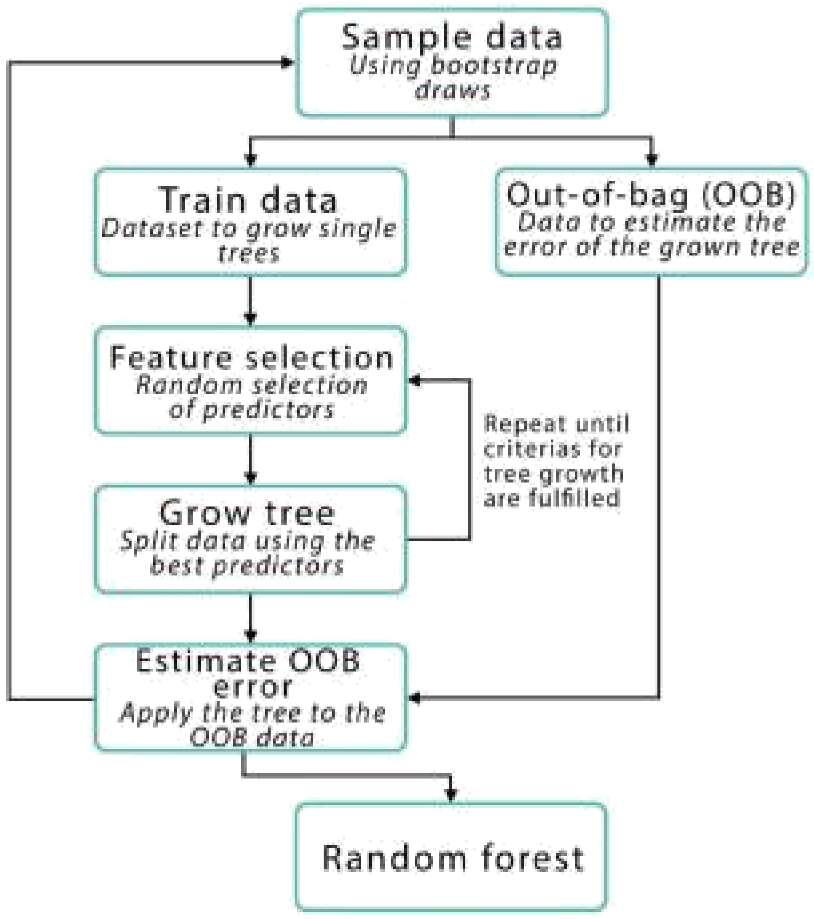

Sample data
Using bootstrap draws
Train data
Dataset to grow single trees
Out-of-bag (OOB)
Data to estimate the error of the grown tree
Feature selection
Random selection of predictors
Repeat until criterias for tree growth are fulfilled
Grow tree
Split data using the best predictors
Estimate OOB error
Apply the tree to the OOB data
Random forest

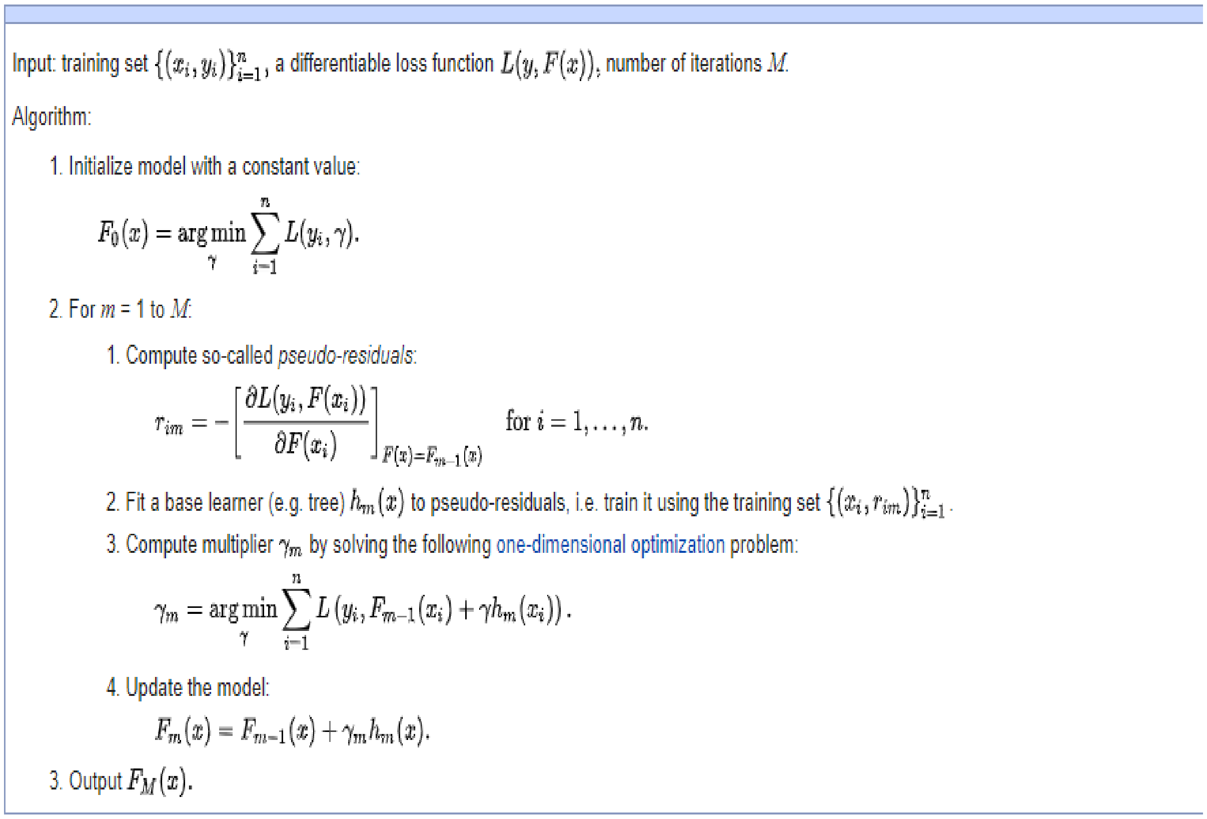
Figure 4

3.XGBoost

In recent times, this an algorithm which dominates applied ML machine learning and Kaggle competitions over others for tabular and structural data. XGBoostalgorithm is implementedby gradient boosted decision trees designed for speed and performance.In XGBoost we use extremely randomized decision-trees . All the trees have high bias and very low variance.This technique basically use Gradient Boosting. **Gradient boosting** is a technique of ML 'machine learning' for regression and classification problems , that generates a prediction model in the form of an ensemble of dull prediction models , typically in decision trees . It prepared this model in a “stage wise” fashion as other boosting methods do, and it conclude them by allowing the arbitrary differentiable loss function optimization.

Leo Breiman had observed and thought ofgradient boosting, that boosting techniques could be considered as some optimization algorithms on some appropriate cost function . Explicit regression “gradient boosting algorithm” was subsequently developed by Mr. Jerome H Friedman all together with the more general functional gradient boosting perspective of Llew Mason , Peter Bartlett, Marcus Frean and Jonathan Baxter. From the later two papers were introduced in the view of “boosting algorithms” as an iterative functional gradient descent algorithms . This is algorithms which optimized a cost function over function space by iteratively selecting a function (“weak hypothesis”) that would points to the –ve “gradient direction” . This ‘functional gradient’ view of boosting has given direction for the development of various ‘boosting algorithms’ in many areas of ML “ machine learning” and ‘statistics’ beyond the regression and classification .

Algorithm:



```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.
Algorithm:
  1. Initialize model with a constant value:
       $F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$
  2. For $m$ = 1 to $M$:
       1. Compute so-called pseudo-residuals:
            $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$   for $i = 1, \ldots, n.$
       2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
       3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
            $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
       4. Update the model:
            $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
  3. Output $F_M(x)$.
```

3.2.1 Model Testing

In step 3 we trained/built a model, it will be tested now using 'test' set of data. New model will be built by the result produced from this, that might considers the predefined models , i.e. "learn" from them

3.2.2 Cross-validation

The major setback in the methods of evaluation of accuracy present in methods of machine learning is that on the new data, model performance cannot be predicted. Cross-validation approach is used to overcome the drawback.Raw data set is split initially. Firstly largest part of split set of data is trained on model and subsequently smaller sets are tested. Three separate classes are there present for cross-validation.

Holdout method: – here, separate the data-set in 2 parts: a test set and atraining set. Fit the model using training set.Test this trained model using test set, model has not seen data in this set before. The errors resulting would be then used to find out the “mean absolute test error” , which is used for model evaluation . Higher speed is plus point of this method. On the other side, the evaluations results highly depend on how we select the test set because the variance is usually high . So one can observe significant difference in the evaluation results between different test sets .

The K - fold method: could be viewed as the improvements over the hold out method . Here , given ‘K’-- subsets has been selected randomly, and single holdout method is repeated ‘K’- times , where the each time one of the ‘K’- subset is used in the training set , and the (K - 1) subsets are used as the testing set . Then average error has been calculated all over ‘K’- runs of the hold-out methods. With the increase of ‘K’, the variance is also reduced , ensuring that the accuracy would not changed with different data-sets. Running time and complexity are than the holdout method, which is a disadvantage of this method.

Chapter Four: Experiment Data

4.1 Distribution of data over various class:

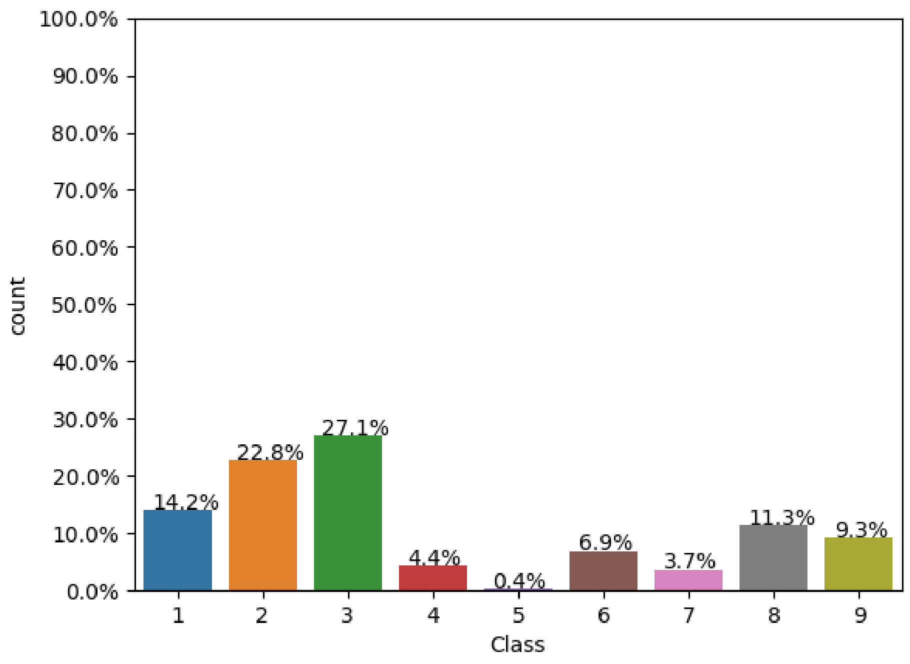
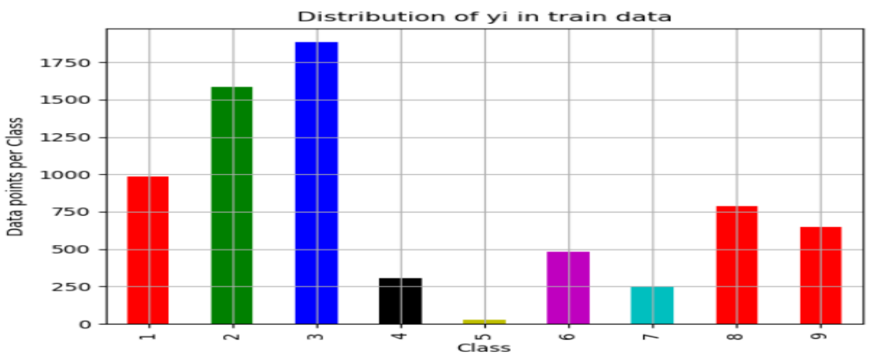


Figure 5

Distribution of datapoints :

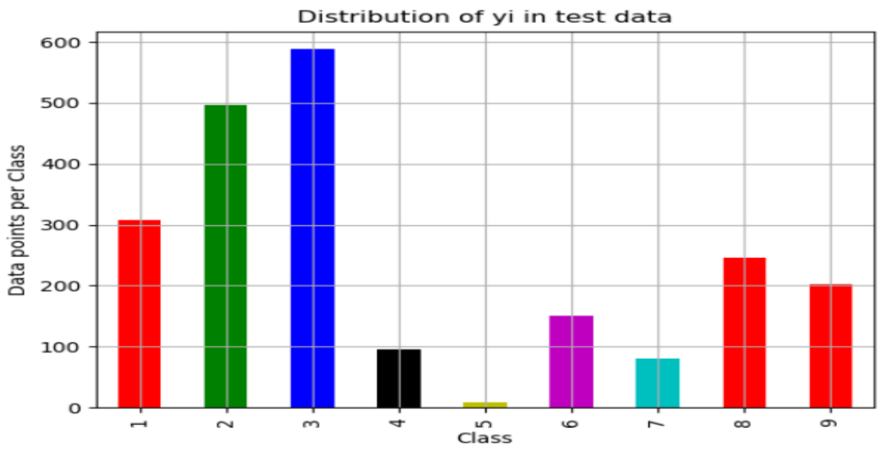
1.Train Dataset:



```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
---------------------------------------------------------
```

Figure 6

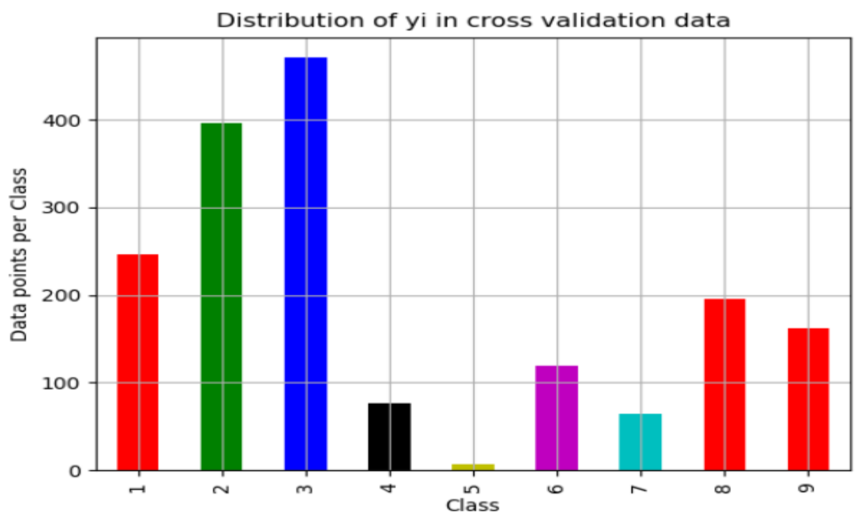
1. **Test Dataset:**



```
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
-----------------------------------------------------------
```

Figure 7

2. **Cross Validation dataset:**



```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

Chapter Five: Multivariate Analysis

Here we plot all the datapoints using TSNE by reducing the dimension of the datapoints from 260 to just 2.

Code:

```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axi
s=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("j
et", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

TSNE plots:

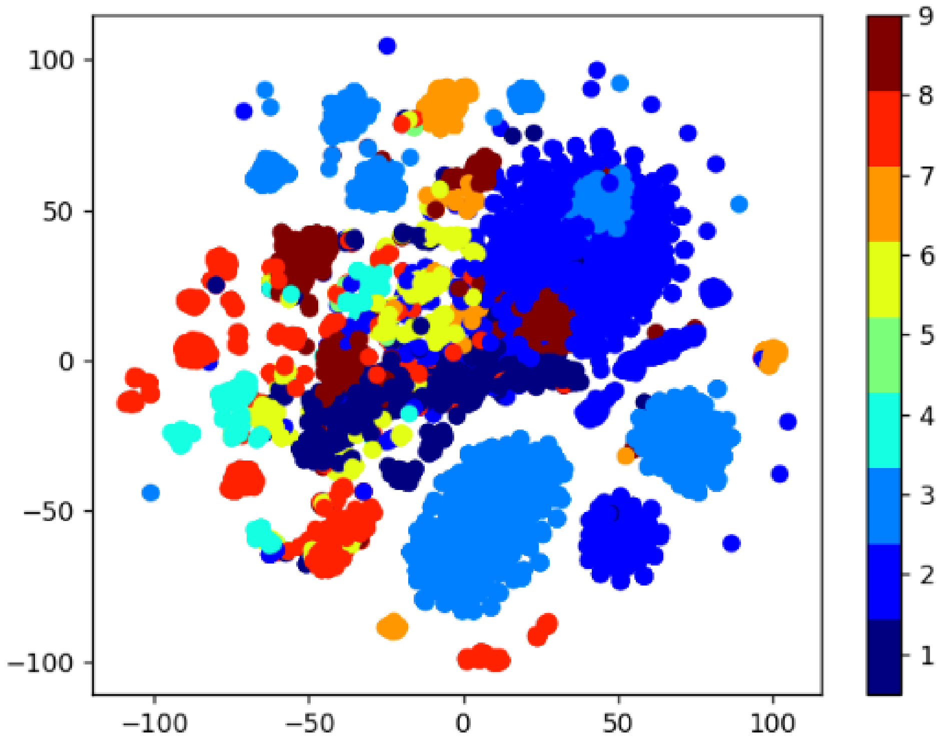


Figure 9

Code:

```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axi
s=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("j
et", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

TSNE plots:

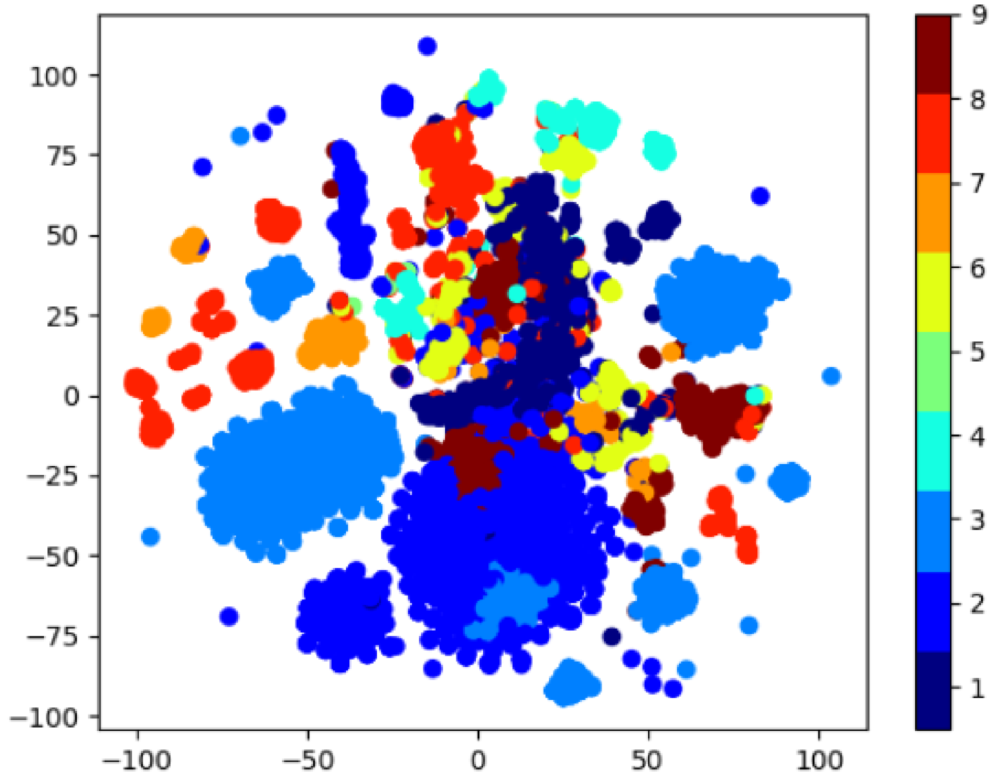
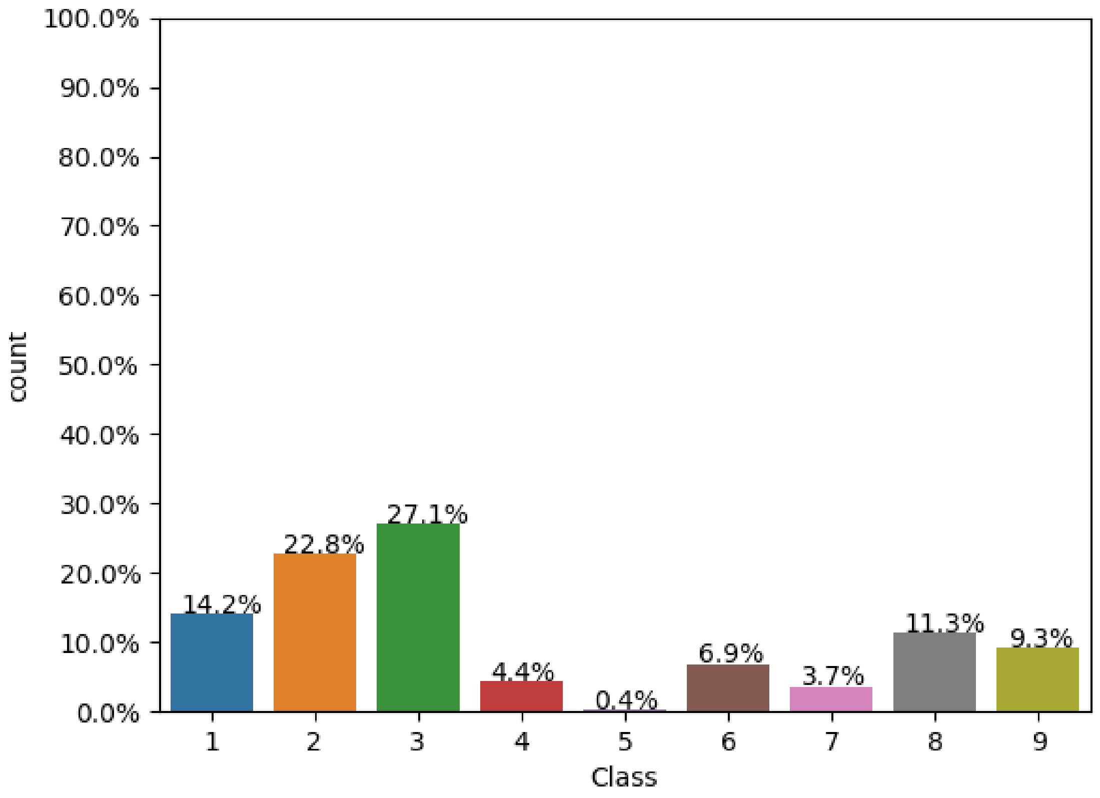


Figure 10

Chapter Six: Result Analysis

Plot showing distribution of data belonging to various class :

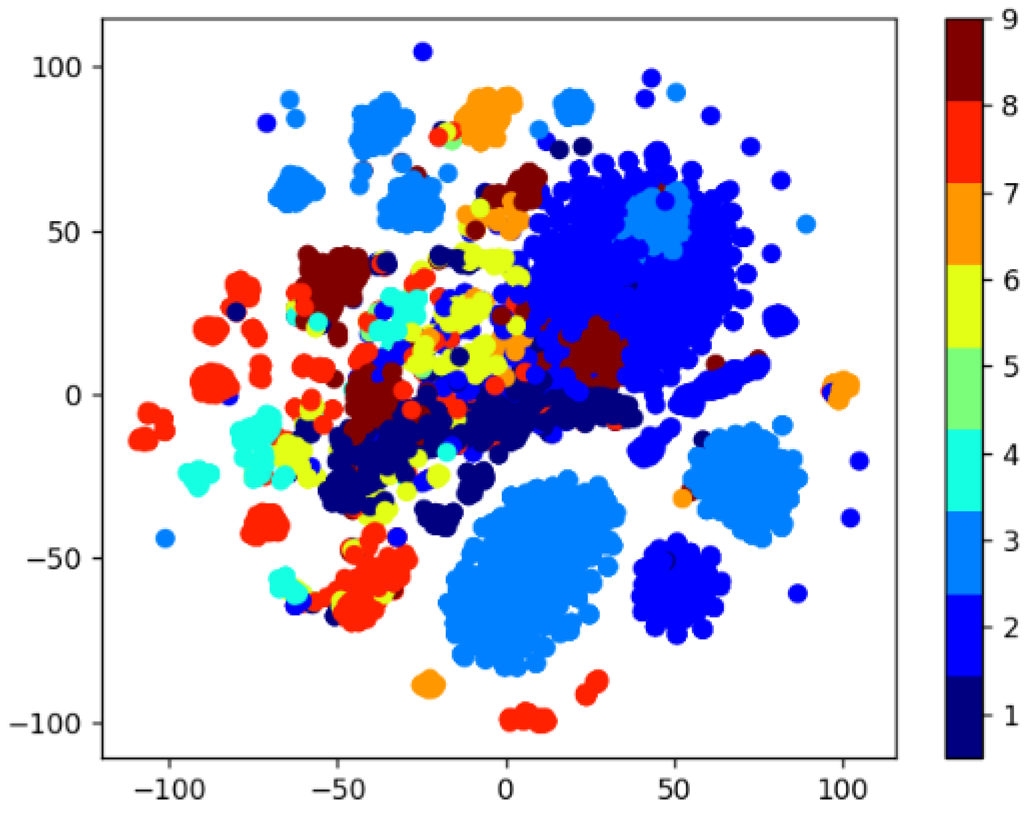


Plot of training data:

The plot below has been plotted using TSNE dimensenality reduction technique. TSNE maintains the neighbors when the dimension is drastically reduced to 2-D.

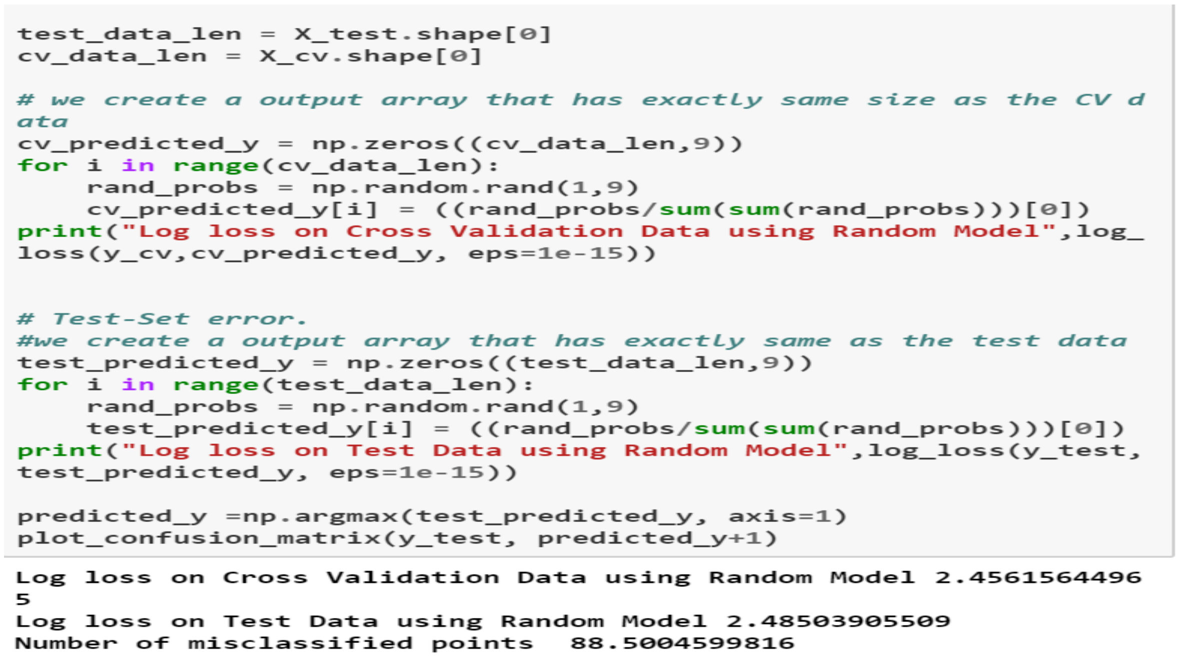
This plot has been plotted to see whether data points belonging to same category are clustered together or not . This helps us in understanding whether the data points can actually be classified or not.

I the plot below we can see that class 3 , 2 and 8 are pretty seperate from other class while other class are mingled .



Random Model:

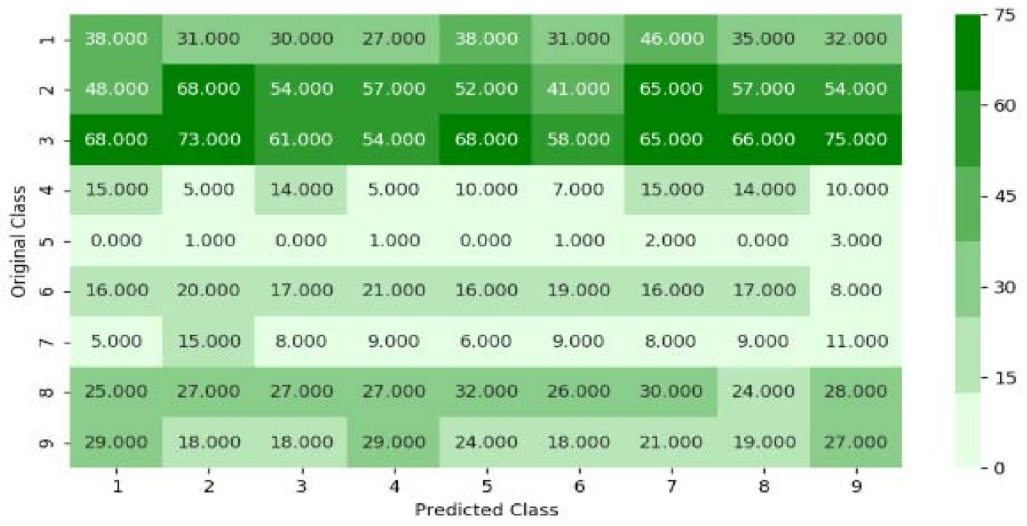
Code:



```
test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV d
ata
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_
loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same size as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,
test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

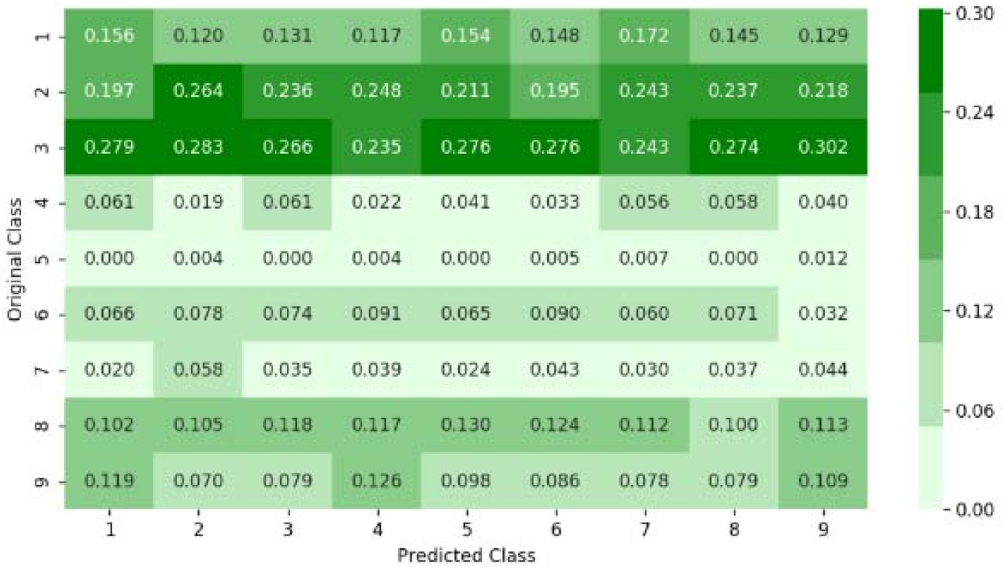
```
Log loss on Cross Validation Data using Random Model 2.4561564496
5
Log loss on Test Data using Random Model 2.48503905509
Number of misclassified points  88.5004599816
```

Confusion matrix:



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	38.000	31.000	30.000	27.000	38.000	31.000	46.000	35.000	32.000
2	48.000	68.000	54.000	57.000	52.000	41.000	65.000	57.000	54.000
3	68.000	73.000	61.000	54.000	68.000	58.000	65.000	66.000	75.000
4	15.000	5.000	14.000	5.000	10.000	7.000	15.000	14.000	10.000
5	0.000	1.000	0.000	1.000	0.000	1.000	2.000	0.000	3.000
6	16.000	20.000	17.000	21.000	16.000	19.000	16.000	17.000	8.000
7	5.000	15.000	8.000	9.000	6.000	9.000	8.000	9.000	11.000
8	25.000	27.000	27.000	27.000	32.000	26.000	30.000	24.000	28.000
9	29.000	18.000	18.000	29.000	24.000	18.000	21.000	19.000	27.000

Precision matrix:



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.156	0.120	0.131	0.117	0.154	0.148	0.172	0.145	0.129
2	0.197	0.264	0.236	0.248	0.211	0.195	0.243	0.237	0.218
3	0.279	0.283	0.266	0.235	0.276	0.276	0.243	0.274	0.302
4	0.061	0.019	0.061	0.022	0.041	0.033	0.056	0.058	0.040
5	0.000	0.004	0.000	0.004	0.000	0.005	0.007	0.000	0.012
6	0.066	0.078	0.074	0.091	0.065	0.090	0.060	0.071	0.032
7	0.020	0.058	0.035	0.039	0.024	0.043	0.030	0.037	0.044
8	0.102	0.105	0.118	0.117	0.130	0.124	0.112	0.100	0.113
9	0.119	0.070	0.079	0.126	0.098	0.086	0.078	0.079	0.109

Figure 12

Recall matrix:

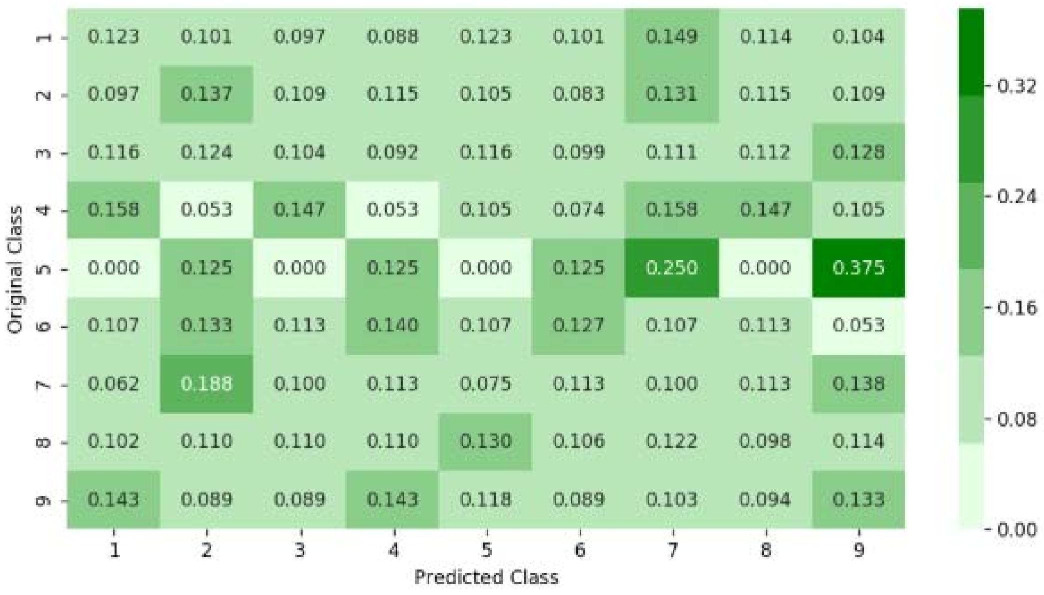
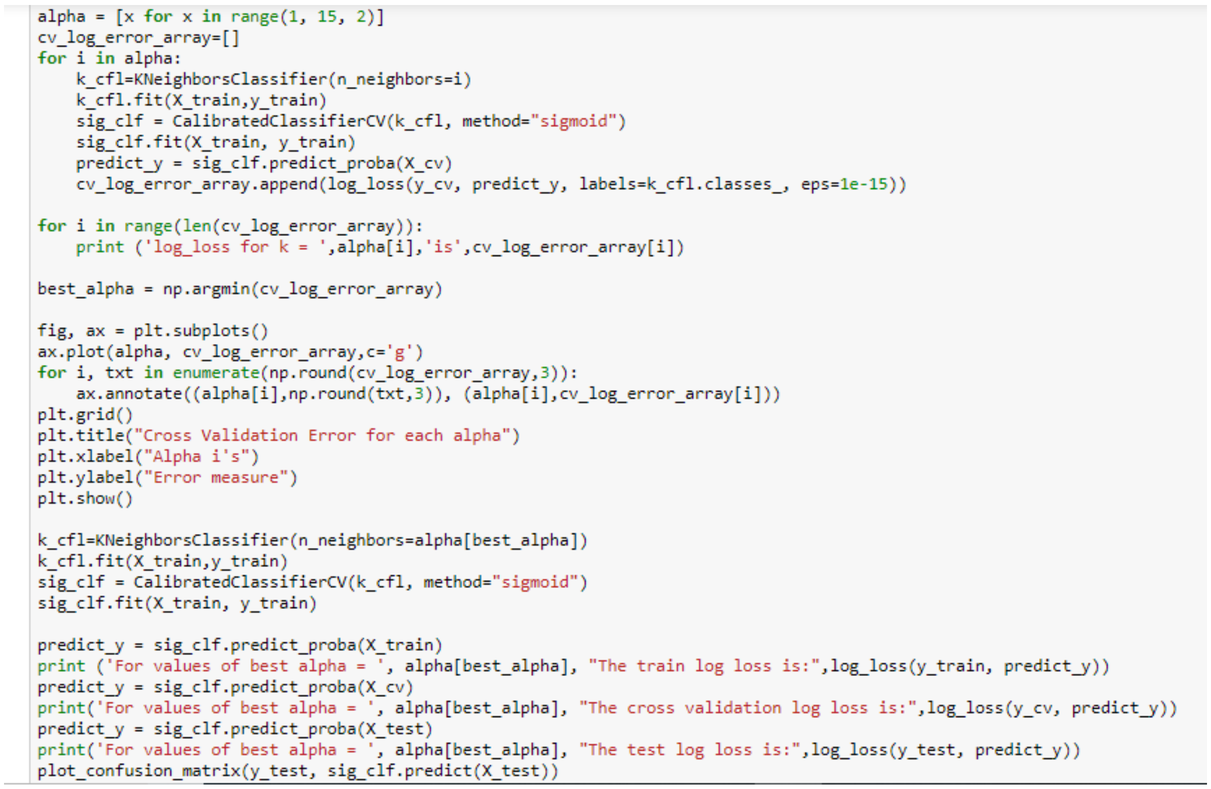


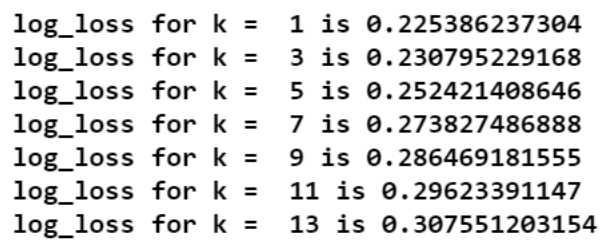
Figure 13

KNN Model:

Code:

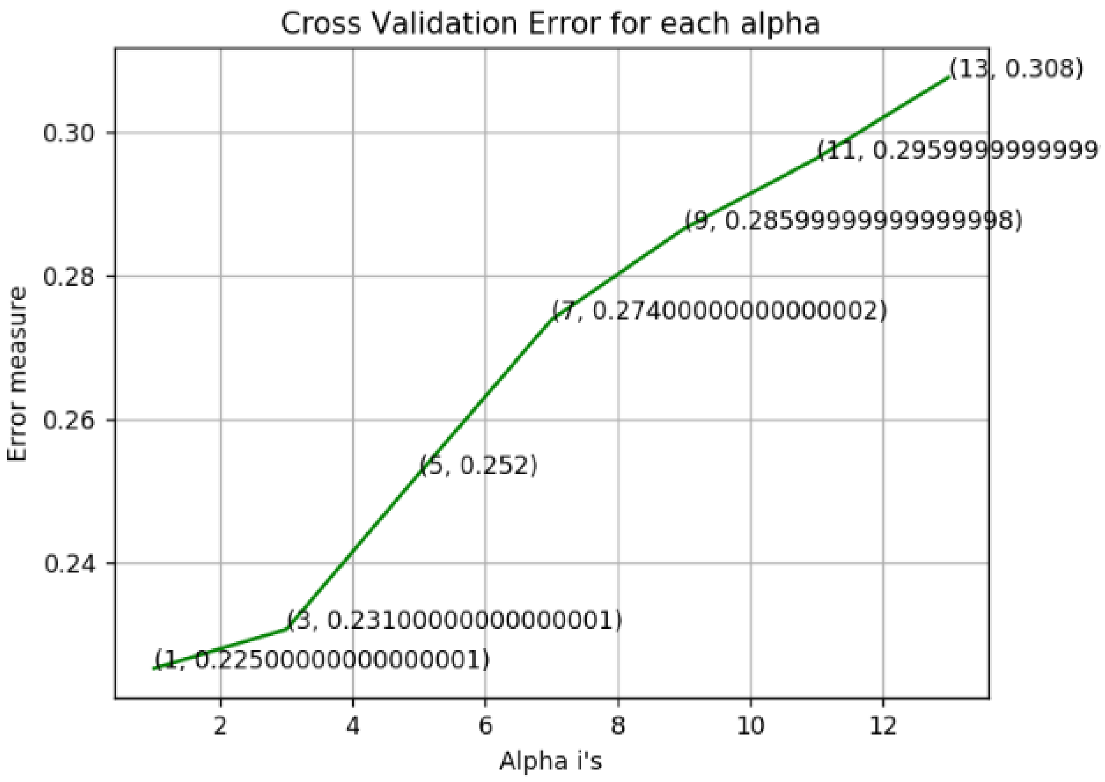


```
alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```



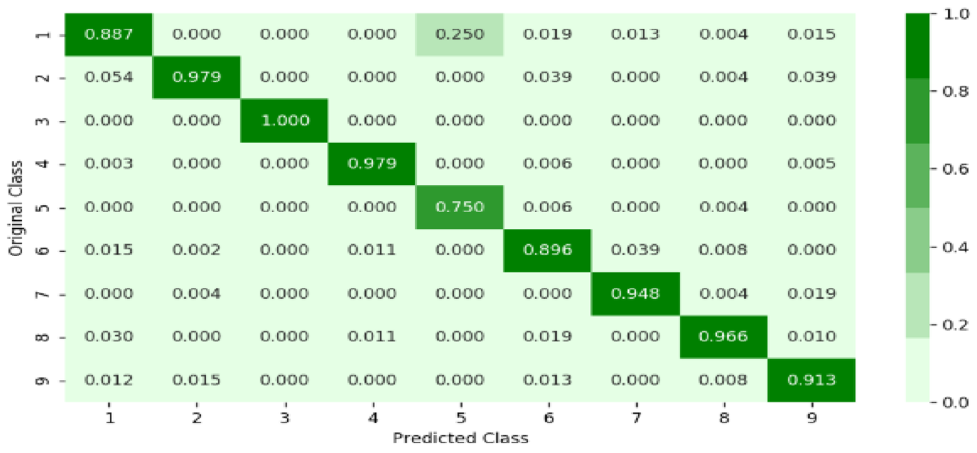
```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154
```

Cross Validation Error:



```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.225386237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
```

Confusion matrix:



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.887	0.000	0.000	0.000	0.250	0.019	0.013	0.004	0.015
2	0.054	0.979	0.000	0.000	0.000	0.039	0.000	0.004	0.039
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.003	0.000	0.000	0.979	0.000	0.006	0.000	0.000	0.005
5	0.000	0.000	0.000	0.000	0.750	0.006	0.000	0.004	0.000
6	0.015	0.002	0.000	0.011	0.000	0.896	0.039	0.008	0.000
7	0.000	0.004	0.000	0.000	0.000	0.000	0.948	0.004	0.019
8	0.030	0.000	0.000	0.011	0.000	0.019	0.000	0.966	0.010
9	0.012	0.015	0.000	0.000	0.000	0.013	0.000	0.008	0.913

Precision matrix:

```
-------------------------------------------------- Precision matrix --------------------------------------------------
<IPython.core.display.Javascript object>
```

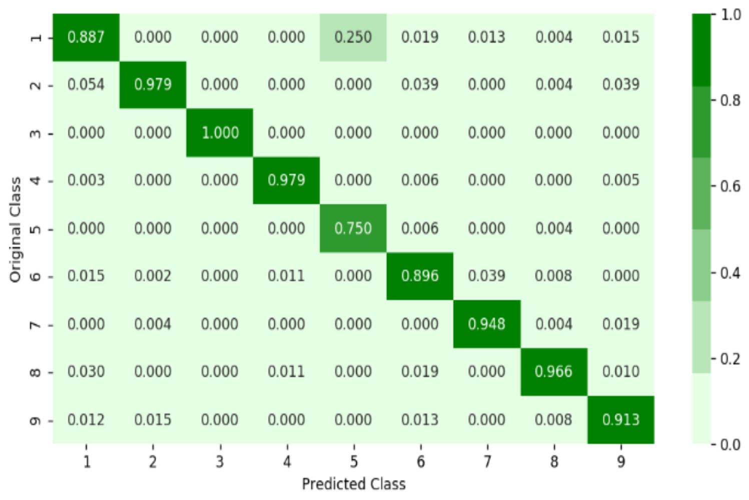


Figure 15

Using KNN and plotting precision matrix we can see that we have good score i.e. in most of the cells in diagonal element we have value greater than 0.85 or 85%.

Precision matrix shows that of all the points predicted to belong to a particular class how many actually belongs to that class.

Recall matrix:

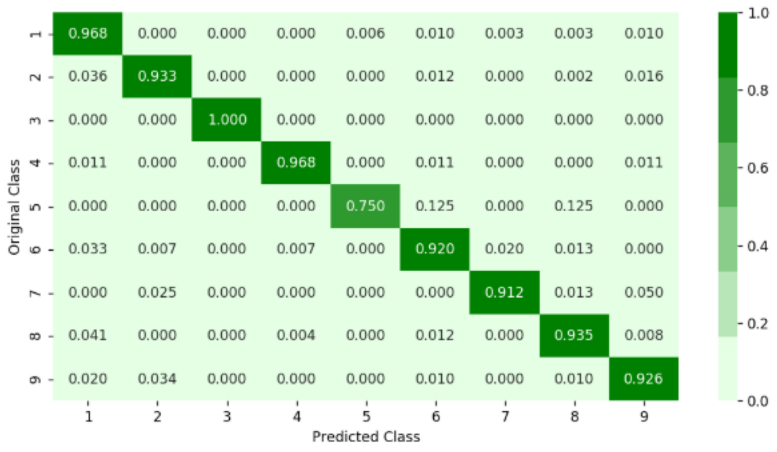


Figure 16

Accuracy:

1. Using KNN we got a logloss of 0.244.

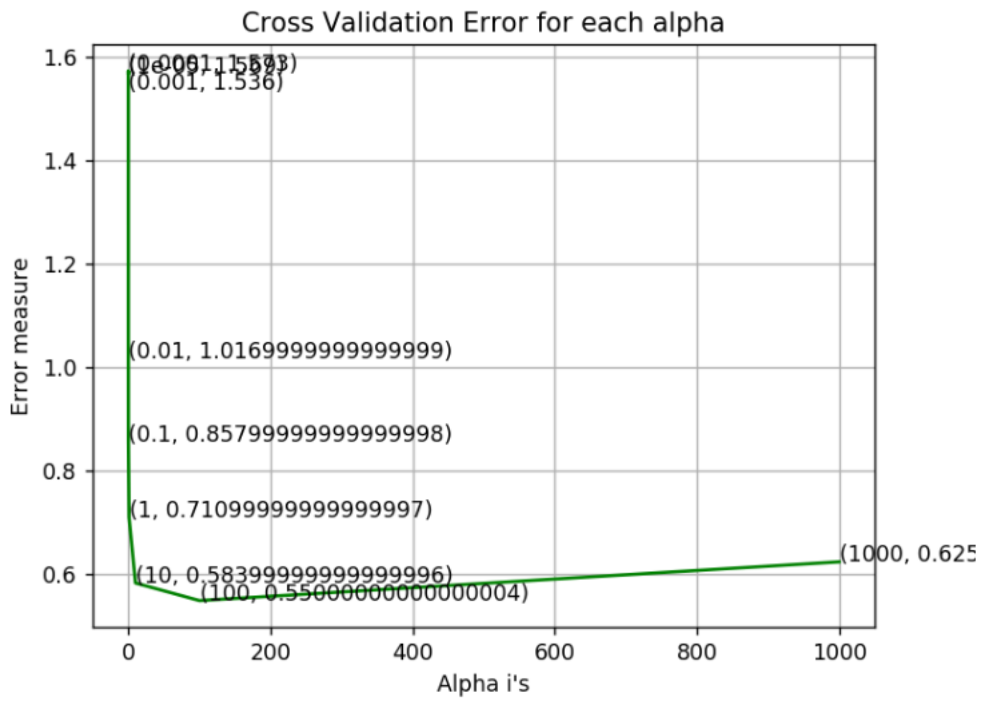
2. Using KNN of all the 2174 points in test dataset approximaely 5 points were misclassified.

Logistic Regression

Code:

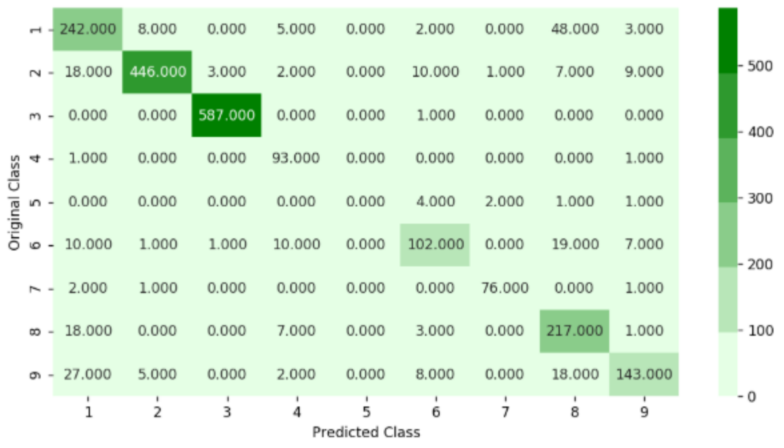
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =  100 is 0.549929846589
log_loss for c =  1000 is 0.624746769121
```



```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
```

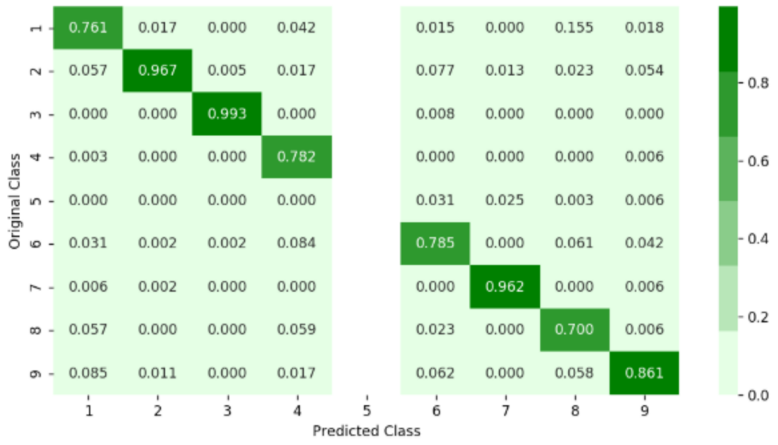
Confusion matrix:



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	242.000	8.000	0.000	5.000	0.000	2.000	0.000	48.000	3.000
2	18.000	446.000	3.000	2.000	0.000	10.000	1.000	7.000	9.000
3	0.000	0.000	587.000	0.000	0.000	1.000	0.000	0.000	0.000
4	1.000	0.000	0.000	93.000	0.000	0.000	0.000	0.000	1.000
5	0.000	0.000	0.000	0.000	0.000	4.000	2.000	1.000	1.000
6	10.000	1.000	1.000	10.000	0.000	102.000	0.000	19.000	7.000
7	2.000	1.000	0.000	0.000	0.000	0.000	76.000	0.000	1.000
8	18.000	0.000	0.000	7.000	0.000	3.000	0.000	217.000	1.000
9	27.000	5.000	0.000	2.000	0.000	8.000	0.000	18.000	143.000

Figure 17

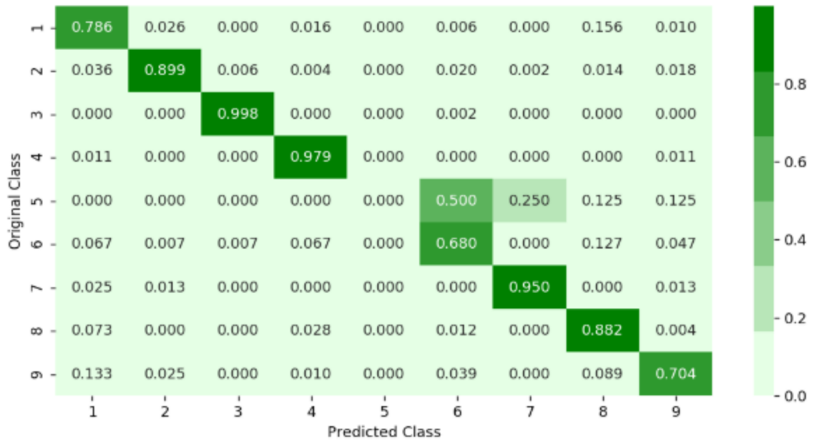
Precision matrix:



	1	2	3	4	5	6	7	8	9
1	0.761	0.017	0.000	0.042		0.015	0.000	0.155	0.018
2	0.057	0.967	0.005	0.017		0.077	0.013	0.023	0.054
3	0.000	0.000	0.993	0.000		0.008	0.000	0.000	0.000
4	0.003	0.000	0.000	0.782		0.000	0.000	0.000	0.006
5	0.000	0.000	0.000	0.000		0.031	0.025	0.003	0.006
6	0.031	0.002	0.002	0.084		0.785	0.000	0.061	0.042
7	0.006	0.002	0.000	0.000		0.000	0.962	0.000	0.006
8	0.057	0.000	0.000	0.059		0.023	0.000	0.700	0.006
9	0.085	0.011	0.000	0.017		0.062	0.000	0.058	0.861

Figure 18

Recall matrix:

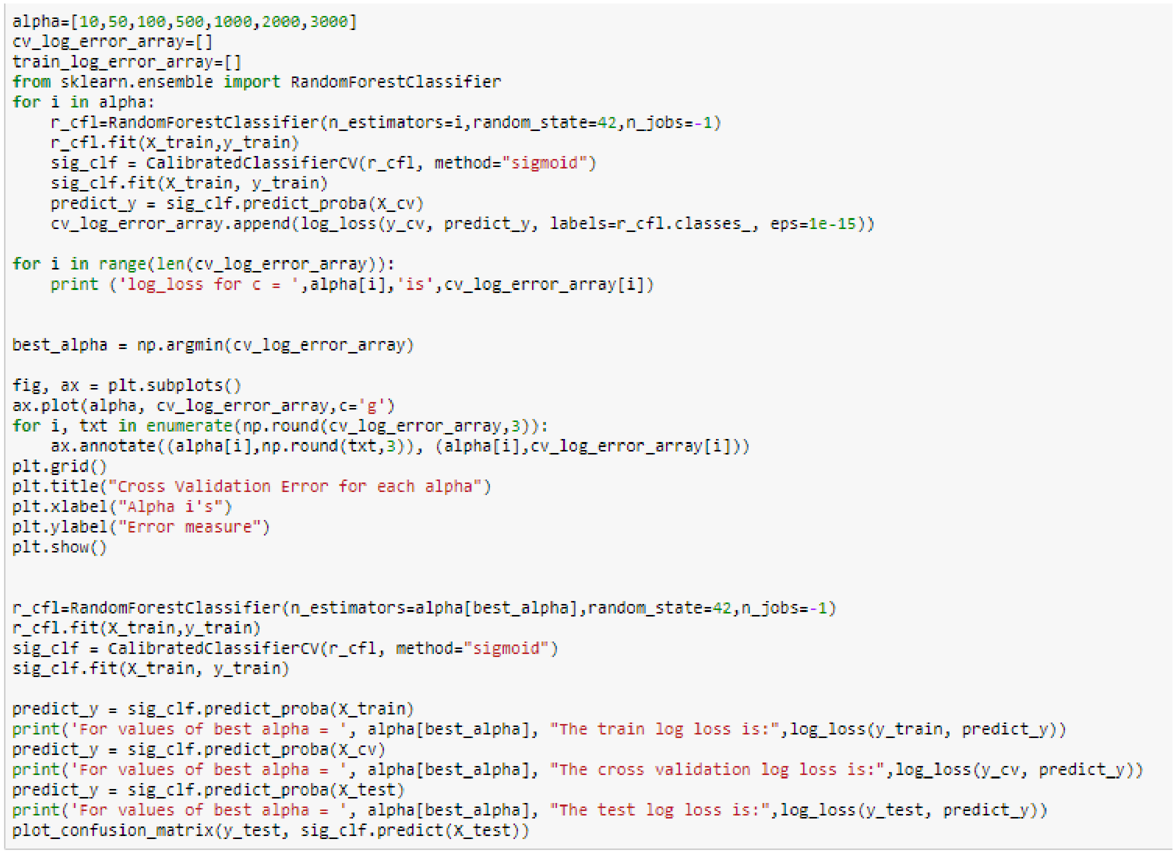


	1	2	3	4	5	6	7	8	9
1	0.786	0.026	0.000	0.016	0.000	0.006	0.000	0.156	0.010
2	0.036	0.899	0.006	0.004	0.000	0.020	0.002	0.014	0.018
3	0.000	0.000	0.998	0.000	0.000	0.002	0.000	0.000	0.000
4	0.011	0.000	0.000	0.979	0.000	0.000	0.000	0.000	0.011
5	0.000	0.000	0.000	0.000	0.000	0.500	0.250	0.125	0.125
6	0.067	0.007	0.007	0.067	0.000	0.680	0.000	0.127	0.047
7	0.025	0.013	0.000	0.000	0.000	0.000	0.950	0.000	0.013
8	0.073	0.000	0.000	0.028	0.000	0.012	0.000	0.882	0.004
9	0.133	0.025	0.000	0.010	0.000	0.039	0.000	0.089	0.704

Figure 19

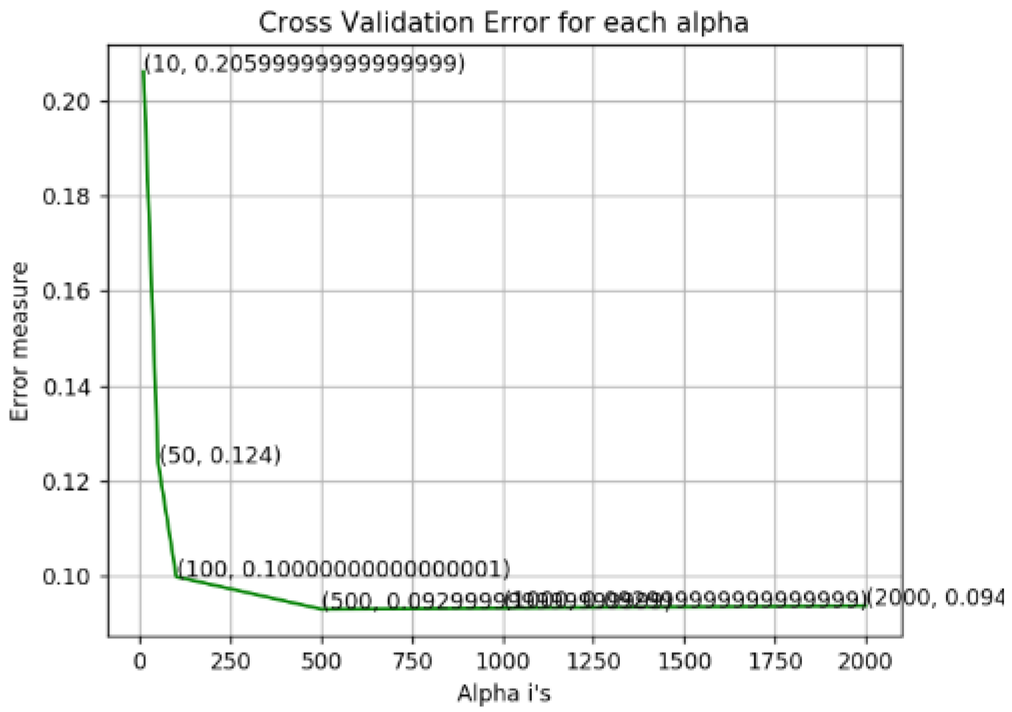
Random Forest Model:

Code:



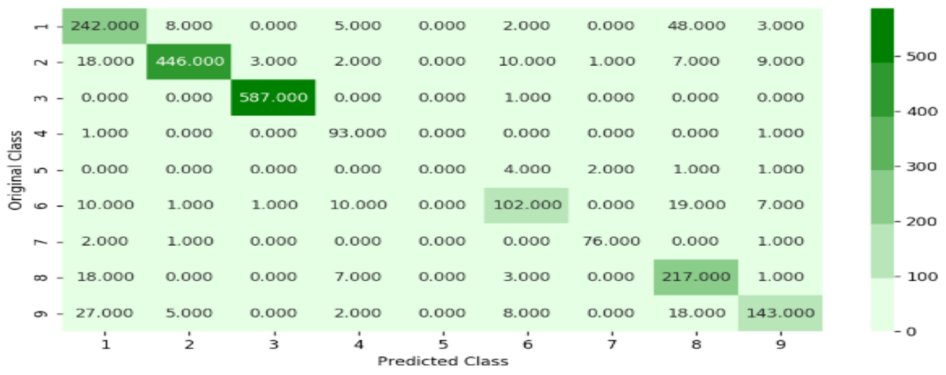
```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

Cross Validation Plot:



```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.0879849524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
```

Confusion Matrix:

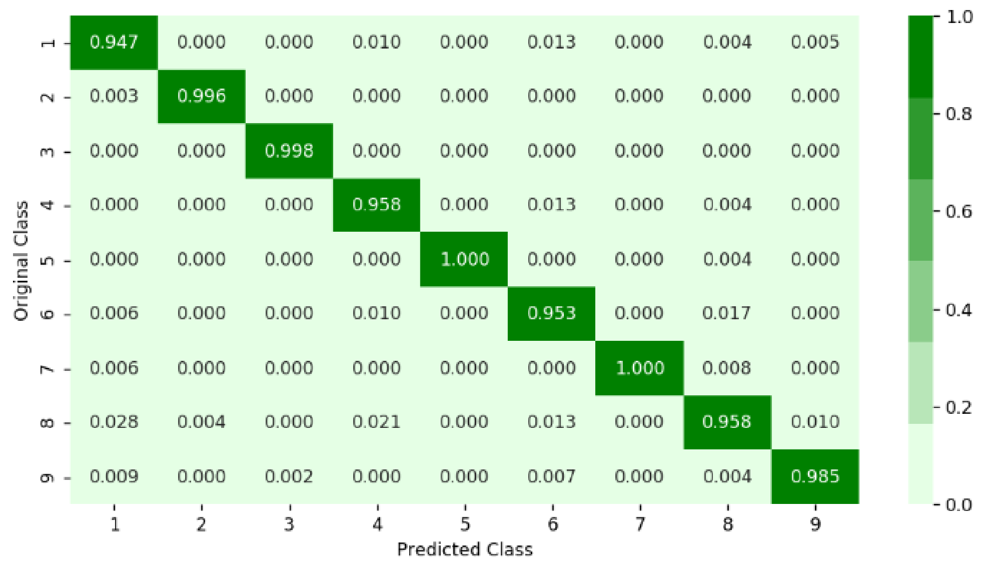


	1	2	3	4	5	6	7	8	9
1	242.000	8.000	0.000	5.000	0.000	2.000	0.000	48.000	3.000
2	18.000	446.000	3.000	2.000	0.000	10.000	1.000	7.000	9.000
3	0.000	0.000	587.000	0.000	0.000	1.000	0.000	0.000	0.000
4	1.000	0.000	0.000	93.000	0.000	0.000	0.000	0.000	1.000
5	0.000	0.000	0.000	0.000	0.000	4.000	2.000	1.000	1.000
6	10.000	1.000	1.000	10.000	0.000	102.000	0.000	19.000	7.000
7	2.000	1.000	0.000	0.000	0.000	0.000	76.000	0.000	1.000
8	18.000	0.000	0.000	7.000	0.000	3.000	0.000	217.000	1.000
9	27.000	5.000	0.000	2.000	0.000	8.000	0.000	18.000	143.000

Figure 20

Precision matrix:

```
-------------------------------------------------- Precision matrix --------------------------------------------------

<IPython.core.display.Javascript object>
```

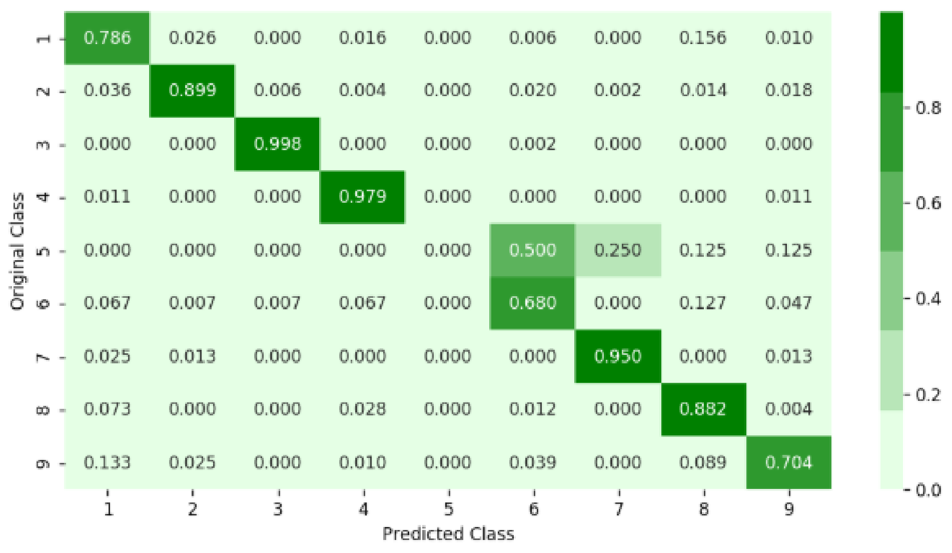


	1	2	3	4	5	6	7	8	9
1	0.947	0.000	0.000	0.010	0.000	0.013	0.000	0.004	0.005
2	0.003	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	0.998	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.958	0.000	0.013	0.000	0.004	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.004	0.000
6	0.006	0.000	0.000	0.010	0.000	0.953	0.000	0.017	0.000
7	0.006	0.000	0.000	0.000	0.000	0.000	1.000	0.008	0.000
8	0.028	0.004	0.000	0.021	0.000	0.013	0.000	0.958	0.010
9	0.009	0.000	0.002	0.000	0.000	0.007	0.000	0.004	0.985

Figure 21

Here we clearly see that the value we got in the diagonal cells is far better than KNN. The value in these diagonal cells has an average of 0.95 or 95%.

Recall matrix:



	1	2	3	4	5	6	7	8	9
1	0.786	0.026	0.000	0.016	0.000	0.006	0.000	0.156	0.010
2	0.036	0.899	0.006	0.004	0.000	0.020	0.002	0.014	0.018
3	0.000	0.000	0.998	0.000	0.000	0.002	0.000	0.000	0.000
4	0.011	0.000	0.000	0.979	0.000	0.000	0.000	0.000	0.011
5	0.000	0.000	0.000	0.000	0.000	0.500	0.250	0.125	0.125
6	0.067	0.007	0.007	0.067	0.000	0.680	0.000	0.127	0.047
7	0.025	0.013	0.000	0.000	0.000	0.000	0.950	0.000	0.013
8	0.073	0.000	0.000	0.028	0.000	0.012	0.000	0.882	0.004
9	0.133	0.025	0.000	0.010	0.000	0.039	0.000	0.089	0.704

Figure 22

Accuracy:

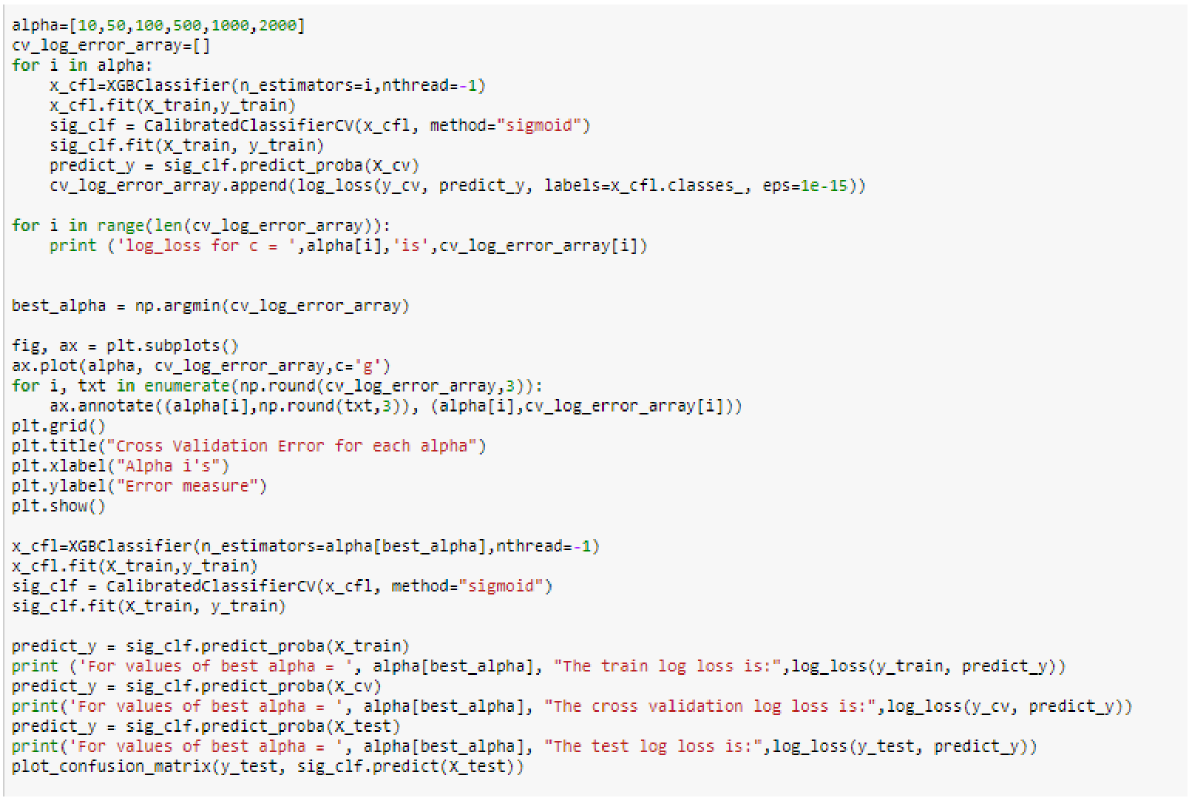
1. Using Random Forest we got an improved logloss of 0.08 which is far better than .

2. Number of misclassified points out of 2174 has been lowered to just 2 compared

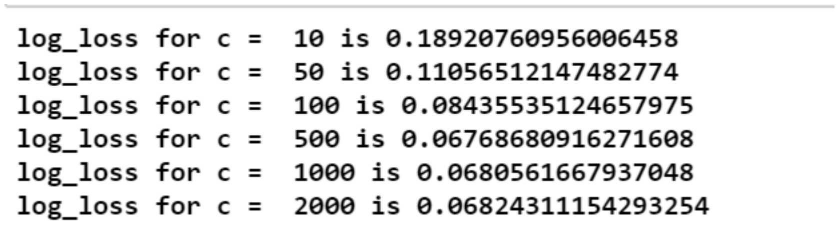
to 5 of KNN.

XGBoost Model:

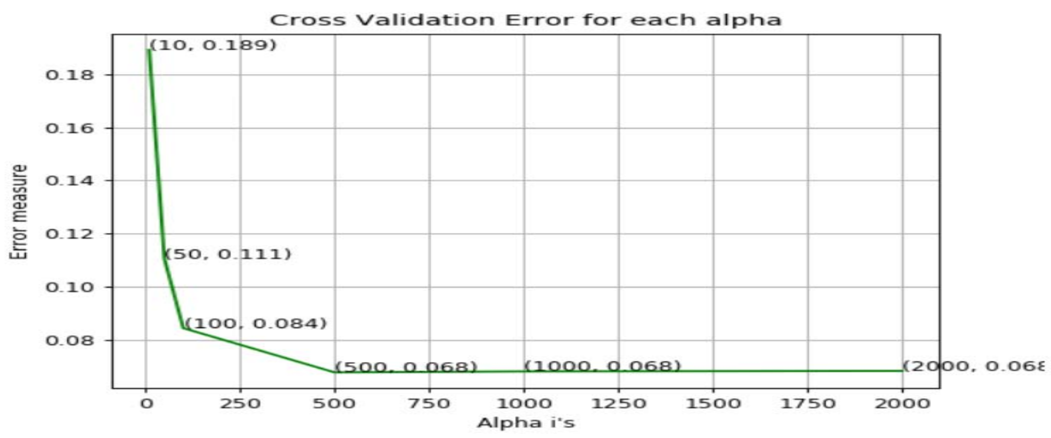
Code:



```
alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```



```
log_loss for c =  10 is 0.18920760956006458
log_loss for c =  50 is 0.11056512147482774
log_loss for c =  100 is 0.08435535124657975
log_loss for c =  500 is 0.06768680916271608
log_loss for c =  1000 is 0.0680561667937048
log_loss for c =  2000 is 0.06824311154293254
```



XgBoost Classification with best hyper parameters using Random Search

Code:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,ve
rbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)
```

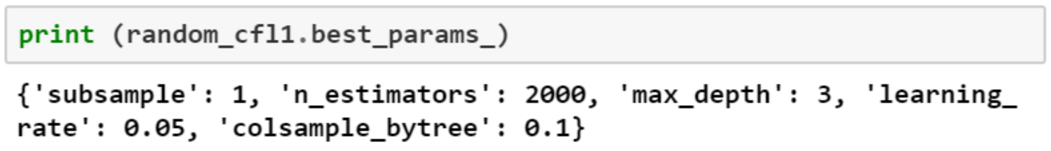
```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 19.7min rem
aining:  2.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 20.5min fin
ished
```

Output:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, booster='gbtre
e', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_
step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimator
s=100,
       n_jobs=1, nthread=None, objective='binary:logistic', rando
m_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.0
5, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000],
'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1],
'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

Best Values of Hyper parameters:

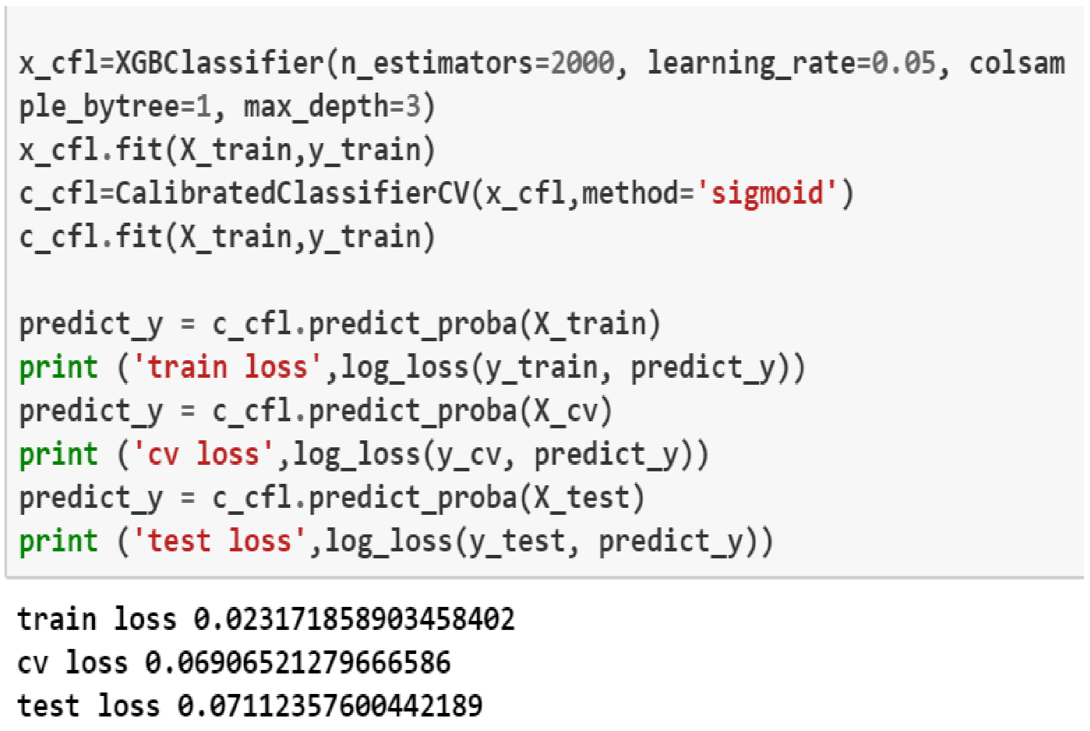


```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 3, 'learning_
rate': 0.05, 'colsample_bytree': 0.1}
```

Training a hyper-parameter tuned Xg-Boost regressor on our train data:

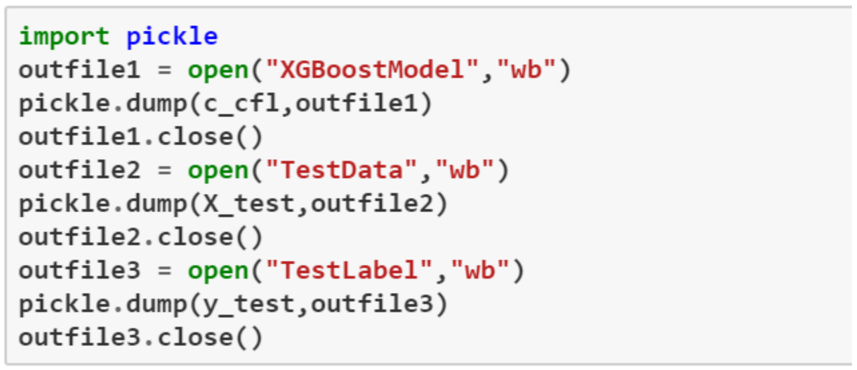
Code:



```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsam
ple_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.023171858903458402
cv loss 0.06906521279666586
test loss 0.07112357600442189
```

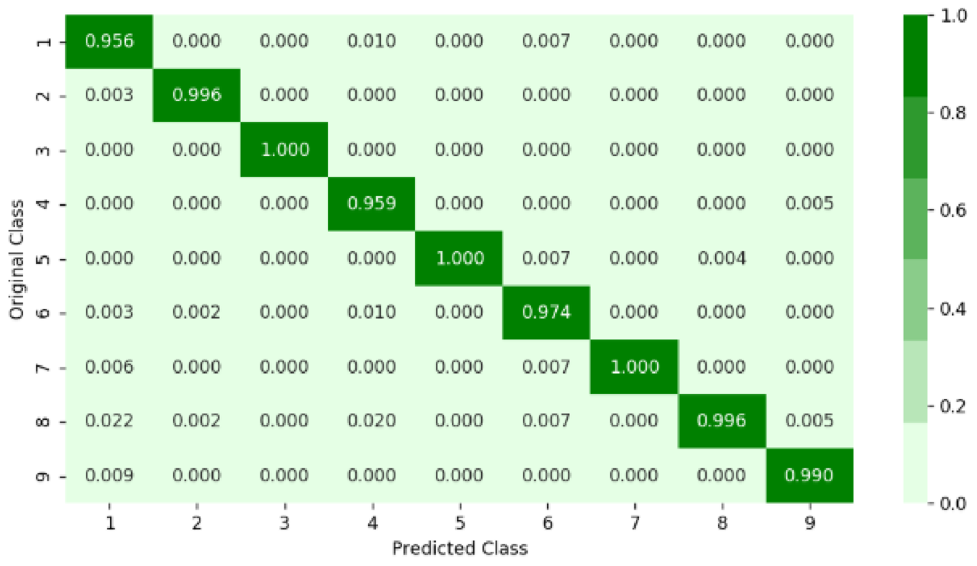
Storing the model :



```
import pickle
outfile1 = open("XGBoostModel","wb")
pickle.dump(c_cfl,outfile1)
outfile1.close()
outfile2 = open("TestData","wb")
pickle.dump(X_test,outfile2)
outfile2.close()
outfile3 = open("TestLabel","wb")
pickle.dump(y_test,outfile3)
outfile3.close()
```

Confusion Matrix:

```
------------------------------------------------- Precision matrix -------------------------------------------------
<IPython.core.display.Javascript object>
```



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.956	0.000	0.000	0.010	0.000	0.007	0.000	0.000	0.000
2	0.003	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.959	0.000	0.000	0.000	0.000	0.005
5	0.000	0.000	0.000	0.000	1.000	0.007	0.000	0.004	0.000
6	0.003	0.002	0.000	0.010	0.000	0.974	0.000	0.000	0.000
7	0.006	0.000	0.000	0.000	0.000	0.007	1.000	0.000	0.000
8	0.022	0.002	0.000	0.020	0.000	0.007	0.000	0.996	0.005
9	0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.990

Figure 23

Here the average value of the diagonal cells have improved to 0.97 or 97%.

Accuracy:

1. The logloss we achieved using XGBoost is 0.07 which is better than random forest.

2. Number of misclassified points out of 2174 point is approximately 1. Thus we have been able to reduce the error further down.

Test Result Analysis

```python
import pickle
infile1 = open("XGBoostModel","rb")
XG_model = pickle.load(infile1)
infile2 = open("TestData","rb")
Test_data = pickle.load(infile2)
infile3 = open("TestLabel","rb")
Test_label = pickle.load(infile3)
infile1.close()
infile2.close()
infile3.close()
```

```python
from sklearn.metrics import confusion_matrix
import numpy as np

predict_label = XG_model.predict(Test_data)
```

```python
# Converting pandas series to list
Test_label = Test_label.tolist()
```

```python
# Printing Percentage of files whose class is correctly predicted.

count = 0
for i in range(len(Test_label)):
    if(Test_label[i]==predict_label[i]):
        count += 1

percentage = (count/len(Test_label))*100
print("Accuracy of the model is:{:.3f}%".format(percentage))
```

```
Accuracy of the model is:98.620%
```

Conclusion and Future Scope

In this challenge, we have compared the different models used in machine learning and have chosen the best model through extensive cross validation and hyper-parameter tuning. This way we have achieved a good accuracy by modelling the data files. We observe that such efficiency of the task is best met by the using Xgboost algorithm for the provided data and using algorithms like random forest search for tuning purposes. Even this accuracy can further be improved by effectively calibrating the models built separately over byte and asm files as well as those trained after their conjunction.

For future work, dataset will also be extended to include all classes of malware in the wild and visualization , better feature engineering and ensembling to enhance the log loss will be the research directions as well.

References

[1] I. Santos, Y. K. Penya, J. Devesa, and P. G. Garcia, "N-grams-based file signatures for malware detection," pp. 317-320, 2009.

[2] T. H. C. W. P. D. a. P. L. K. Rieck, "Learning and classification of malware behavior," in *DIMVA ’08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment.*, Berlin, Heidelberg: Springer-Verlag, 2008.

[3] E. Konstantinou, "Metamorphic virus: Analysis and detection," *Technical Report RHUL-MA-2008-2, Search Security Award M.Sc. thesis,* p. 93 pages, 2008.

[4] P. K. C. a. R. Lippmann, "Machine learning for computer security," *Journal of Machine Learning Research*, vol. 6, p. 2669–2672, 2006.

[5] J. Z. K. a. M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research,* vol. 7, no. special Issue on Machine Learning in omputer Security, p. 2721–2744, December 2006.

[6] M. C. D. A. L. C. Dragos¸ Gavrilut¸, "Malware Detection Using Machine Learning," in *Proceedings of the International Multiconference on Computer Science and Information Technology*, 2009.

[7] N. R. Priyank Singhal, "Malware Detection Module using Machine Learning Algorithms to Assist in Centralized Security in Enterprise Networks," *International Journal of Network Security & Its Applications (IJNSA),*, vol. 4, no. 1, 2012.

[8] N. J.-J. H. Usukhbayar Baldangombo, "A STATIC MALWARE DETECTION SYSTEM USING A STATIC MALWARE DETECTION SYSTEM USING," *International Journal of Artificial Intelligence & Applications (IJAIA),* vol. 4, no. 4, July, 2013..

[9] S. V. P. W. A. Mamoun Alazab, "Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures," in *Proceedings of the 9-th Australasian Data Mining Conference (AusDM'11)*, Ballarat, Australia, 2011.

[10] T. L. D. A. a. S. S. I. Yanfang Ye, "A survey on malware detection using data mining

techniques," *ACM Computing Surveys (CSUR),* p. 41, 2017.

[11] J. X. P. C. a. S. M. A. H. Sung, "Static Analyzer of Vicious utables (SAVE).".

[12] S. J. Mihai Christodorescu, "Static Analysis of Executables to Detect Malicious Patterns," in *Proceedings of the 12th Conference on USENIX,* USENIX Association, Berkeley, CA,USA., 2003.

[13] S. M. Tabish, M. Z. Shafiq and M. Farooq, "Malware Detection Using Statistical Analysis of Byte-level File Content," in *In Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics(CSI-KDD'09),* ACM, New York, NY, USA, 2009.

[14] I. Santos, F. Brezo, X. Ugarte-Pedrero and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Information Sciences 231,* p. 64 – 82, 2013.

[15] B. Yadegari, B. Johannesmeyer, B. Whitely and S. Debray, "A Generic Approach to Automatic Deobfuscation of Executable Code," *IEEE Symposium on Security and Privacy,* p. 674–691, 2015.

[16] J. Drew, T. Moore and M. Hahsler, "Polymorphic malware detection using sequence classification methods," *In Security and Privacy Workshops(SPW),* pp. 81-87, 2016.

[17] L. Nataraj, V. Yegneswaran, P. Porras and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Workshop on Artificial Intelligence and Security (AISec),* Oct 2011.

[18] L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath, "Malware images Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security, VizSec '11,* New York, NY, USA, 2011.

[19] U. Bayer, P. M. Comparett, C. Hlauschek, C. Kruegel and E. Kirda, "Scalable, behavior-based malware clustering".

[20] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian and J. Nazario, "Automated classification and analysis of internet malware," *Lecture Notes in Computer Science,* vol. 4637, p. 178–197, 2007.