CSE4020 - MACHINE LEARNING

FINAL REVIEW

Name:SR NAVYA SREE

Reg No: 16BCE0223

Slot: F1

Name: MARIA ALEX .K

Reg No: 16BCE02190

Slot: F1

TITLE – MALWARE DETECTION USING CNN ON EMBER DATASET

1. Title of the project (Problem Statement& method developed/used)

Problem Statement –

Malware is one of the most serious threats, with such a rapid increase in the malware attacks there is a need to demonstrate various mitigation and malware detection techniques. From the insights gained from this project, we have tried to detect malware by utilizing the CNN technique and feedforward neural network method.

Method Used –

Malware detection using CNN (Convolution Neural Networks) focusses on training machine learning models to efficiently detect the malicious Windows portable executable files. EMBER (Endgame Malware Benchmark for Research) is used for training and to create learning model for malware detection.

Scenario – We have performed the Feedforward Neural Network Concept for EMBER.

2. **Objectives of the project**

Objectives –

Objective-1 – Analysing the Malware dataset EMBER

Objective-2 – To perform Convolution Neural Network analysis on EMBER.

Objective-3 – To use Feedforward Neural Network analysis on Malware dataset.

Objective-4 – Comparing between the performance of Feedforward Neural Network and Convolution Neural network

3. Data Set used in your project

(Mention the source of your data (give url or reference of the data set). Describe the data and also mention if you have applied pre-processing techniques, then mention how you have pre-processed the data set).

Dataset Available in the Link - https://pubdata.endgame.com/ember/ember_dataset.tar.bz2

DESCRIPTION OF DATASET

EMBER data set is a collection of 1.1 million portable executable files sha256 hashes that was collected by VirusTotal. This data set is a combination of 900K training samples in which 300K is under malicious category, 300K belongs to benign category and remaining 300K is unlabelled. Test data set includes 200K test samples which is divided into malicious and benign with 100K of samples each.

The data set didn't require pre-processing it already in the JSON format with essential contents.

EMBER dataset is a collection of JSON lines –

- Unique identifier - SHA 256 hash of the original file
- Label – Benign / Malicious / Unlabelled
- Month Resolution (File was first seen)
- File size
- Headers which consist of characteristics, timestamps etc.

The data is already pre-processed and available in JSON format

5. Methodologies applied to your project

- ✓ ***Name of the Method-1 feed forward neural network***

A feedforward neural network is an artificial neural network where in connections between the nodes do not form a cycle .It is different from recurrent neural networks.

We are using feed forward neural network for binary malware classification that is trained on various features extracted from the static malware binary. All these features require further processing and are then fed into a four layer feed forward network for training.

- ✓ ***Name of the Method-2 convolution neural network***

Unlike neural networks in convolution neural network the input is a vector, here the input is a multi-channelled image

The first type of neural network we use is recurrent neural network that is trained for extracting behavioural features of PE file, and the second type is convolutional neural network that is applied to classify samples.

We will convert malware into binary 8 bit vector and then to 8 bit vector to grey scale image , then we will feed these images to malware machine learning classifiers that use CNN to train.

Three layers are created –

1) Convolution layer – Extracts features by multiplying original pixel values with filter values.
2) Pooling layer – Dimensionality of feature map is reduced in pooling operation.

6. Result (outcome of the project) and discussion

(Insert the editable table; don't paste the screenshot of image of the table). Follow the below table format (Table 1 & Table 2& so on…..)

Result of feed forward neural network

0EPOCH NUMBER	TEST DATA	PRECISION	RECALL	F1-score	support
0	0	0.97	0.97	0.97	100000
	1	0.97	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000

1	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
2	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
3	0	0.96	0.98	0.97	100000
	1	0.98	0.96	0.97	100000
Total/avg		0.97	0.97	0.97	200000
4	0	0.96	0.98	0.97	100000
	1	0.98	0.96	0.97	100000
Total/avg		0.97	0.97	0.97	200000
5	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
6	0	0.96	0.98	0.97	100000
	1	0.98	0.96	0.97	100000
Total/avg		0.97	0.97	0.97	200000
7	0	0.97	0.97	0.97	100000
	1	0.97	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
8	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
9	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
10	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
11	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
12	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
13	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
14	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
15	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
16	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000

17	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
18	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
19	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
20	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
21	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
total/avg		0.97	0.97	0.97	200000
22	0	0.97	0.98	0.98	100000
	1	0.98	0.97	0.98	100000
Total/avg		0.98	0.98	0.98	200000
23	0	0.97	0.98	0.98	100000
	1	0.98	0.97	0.98	100000
Total/avg		0.98	0.98	0.98	200000
24	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
25	0	0.97	0.98	0.97	100000
	1	0.98	0.97	0.97	100000
Total/avg		0.97	0.97	0.97	200000
26	0	0.96	0.98	0.97	100000
	1	0.98	0.96	0.97	100000
total/avg		0.97	0.97	0.97	200000
27	0	0.97	0.98	0.98	100000
	1	0.98	0.97	0.98	100000
Total/avg		0.98	0.98	0.98	200000
28	0	0.97	0.98	0.98	100000
	1	0.98	0.97	0.98	100000
Total/avg		0.98	0.98	0.98	200000
29	0	0.96	0.98	0.97	100000
	1	0.98	0.96	0.97	100000
Total/avg		0.97	0.97	0.97	200000

Result of Convolution neural network

EPOCH NUMBER	TEST DATA	PRECISION	RECALL	F1-score	support
0	0	0.95	0.93	0.94	100000
	1	0.93	0.95	0.94	100000
Total/avg		0.94	0.94	0.94	200000
1	0	0.95	0.93	0.94	100000

	1	0.93	0.96	0.94	100000
Total/avg		0.94	0.94	0.94	200000
2	0	0.96	0.94	0.95	100000
	1	0.94	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
3	0	0.93	0.96	0.95	100000
	1	0.96	0.93	0.95	100000
Total/avg		0.95	0.95	0.95	200000
4	0	0.96	0.93	0.95	100000
	1	0.94	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
5	0	0.95	0.94	0.95	100000
	1	0.94	0.95	0.95	100000
Total/avg		0.95	0.95	0.95	200000
6	0	0.95	0.95	0.95	100000
	1	0.95	0.95	0.95	100000
total/avg		0.95	0.95	0.95	200000
7	0	0.96	0.93	0.95	100000
	1	0.93	0.96	0.95	100000
total/avg		0.95	0.95	0.95	200000
8	0	0.95	0.94	0.95	100000
	1	0.94	0.95	0.95	100000
total/avg		0.95	0.95	0.95	200000
9	0	0.95	0.94	0.94	100000
	1	0.94	0.95	0.95	100000
Total/avg		0.95	0.95	0.95	200000
10	0	0.96	0.95	0.95	100000
	1	0.95	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
11	0	0.94	0.95	0.95	100000
	1	0.95	0.94	0.95	100000
Total/avg		0.95	0.95	0.95	200000
12	0	0.92	0.97	0.94	100000
	1	0.96	0.92	0.94	100000
Total/avg		0.94	0.94	0.94	200000
13	0	0.96	0.94	0.95	100000
	1	0.94	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
14	0	0.95	0.95	0.95	100000
	1	0.95	0.95	0.95	100000
Total/avg		0.95	0.95	0.95	200000
15	0	0.94	0.95	0.95	100000
	1	0.95	0.94	0.95	100000
Total/avg		0.95	0.95	0.95	200000
16	0	0.96	0.93	0.94	100000
	1	0.93	0.96	0.95	100000
total/avg		0.95	0.95	0.95	200000
17	0	0.94	0.96	0.95	100000

	1	0.96	0.94	0.95	100000
total/avg		0.95	0.95	0.95	200000
18	0	0.94	0.95	0.94	100000
	1	0.95	0.94	0.94	100000
total/avg		0.94	0.94	0.94	200000
19	0	0.96	0.93	0.95	100000
	1	0.94	0.96	0.95	100000
total/avg		0.95	0.95	0.95	200000
20	0	0.96	0.94	0.95	100000
	1	0.94	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
21	0	0.96	0.95	0.95	100000
	1	0.95	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
22	0	0.95	0.93	0.94	100000
	1	0.93	0.95	0.94	100000
Total/avg		0.94	0.94	0.94	200000
23	0	0.95	0.93	0.94	100000
	1	0.93	0.96	0.94	100000
Total/avg		0.94	0.94	0.94	200000
24	0	0.96	0.94	0.95	100000
	1	0.94	0.96	0.95	100000
Total/avg		0.95	0.95	0.95	200000
25	0	0.93	0.96	0.95	100000
	1	0.96	0.93	0.95	100000
total/avg		0.95	0.95	0.95	200000
26	0	0.96	0.93	0.95	100000
	1	0.94	0.96	0.95	100000
total/avg		0.95	0.95	0.95	200000
27	0	0.95	0.94	0.95	100000
	1	0.94	0.95	0.95	100000
total/avg		0.95	0.95	0.95	200000
28	0	0.95	0.95	0.95	100000
	1	0.95	0.95	0.95	100000
total/avg		0.95	0.95	0.95	200000
29	0	0.96	0.93	0.95	100000
	1	0.93	0.96	0.95	100000
total/avg		0.95	0.95	0.95	200000

7. Outcome in-terms of Graphs (result)

[Copy the image and paste it; don’t take screenshots. In python (Jupiter Note book you have the option of copying images) or R]

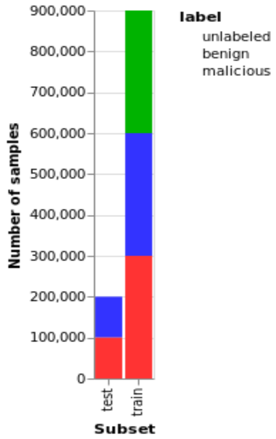


Figure-1: Distribution of malicious, benign and unlabeled samples in the training and test sets

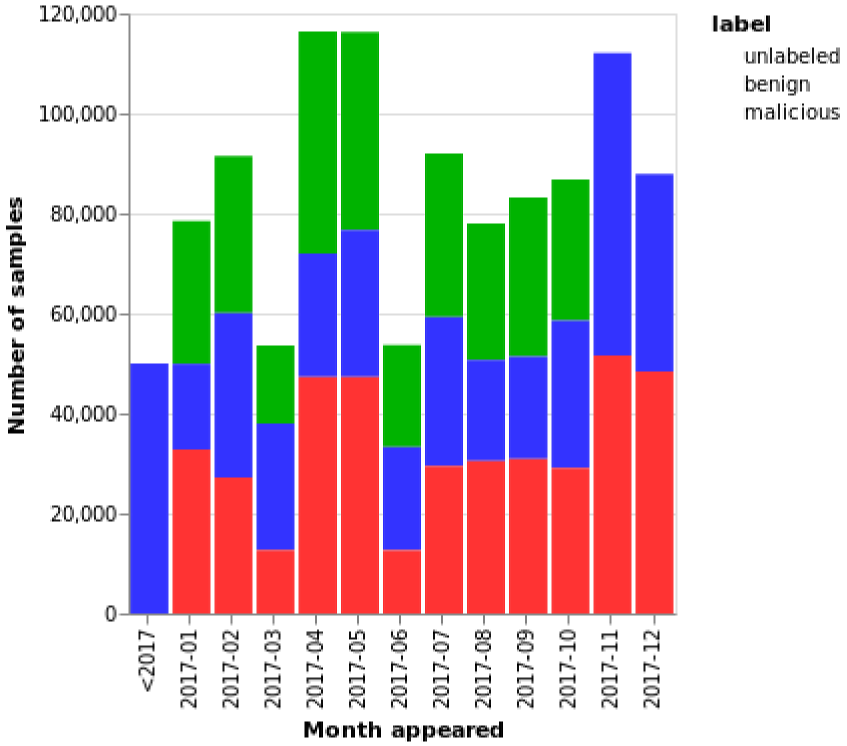


Figure-2: A temporal distribution of the dataset, available from chronology data available in the metadata, with 2017-11 and 2017-12 corresponding to the test set

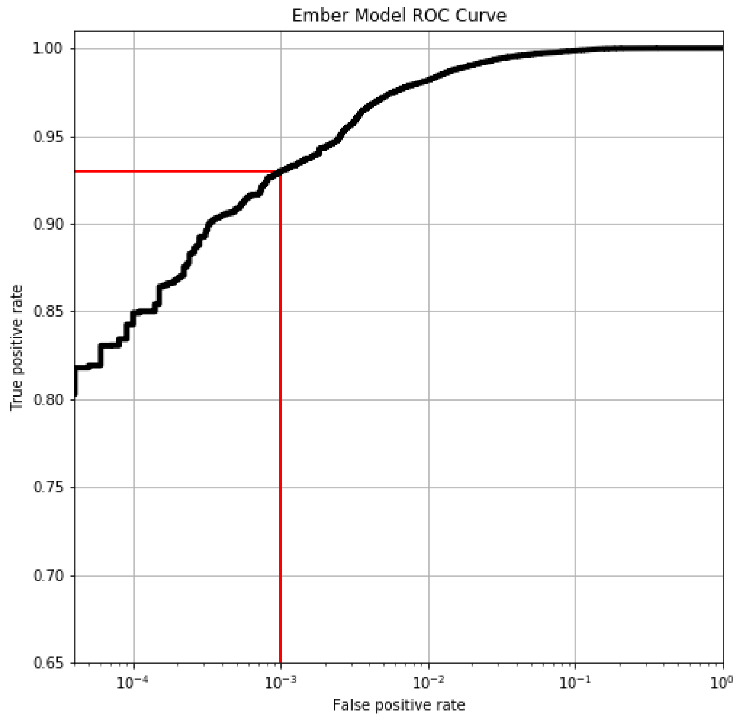


Figure-3: ROC curve with log scale for false positive rate (FPR). The threshold shown (red) corresponds to a 0.1% FPR and a detection rate about 93%. At 1% FPR the detection rate exceeds 98%.

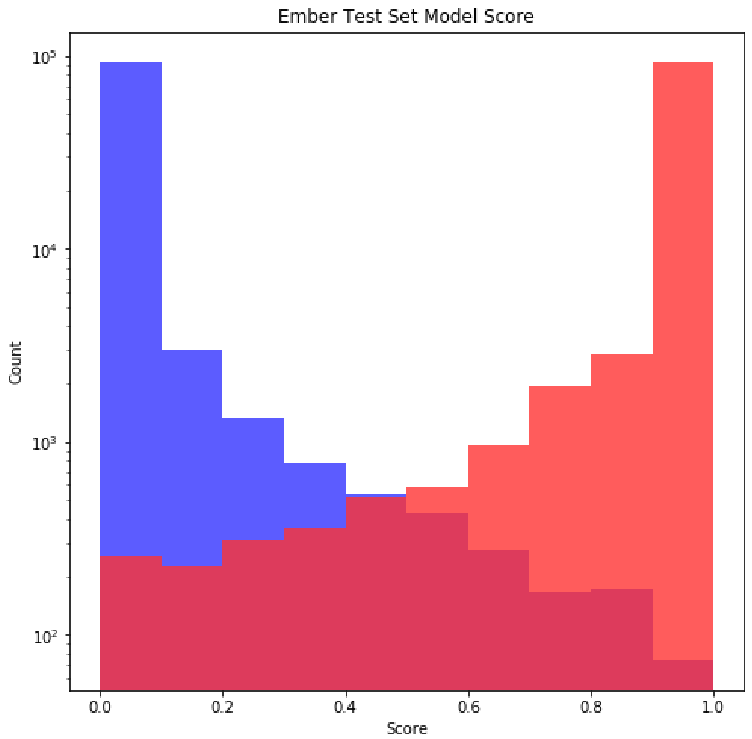


Figure-4: Distribution of model test scores on the test set (note the logarithmic scale)

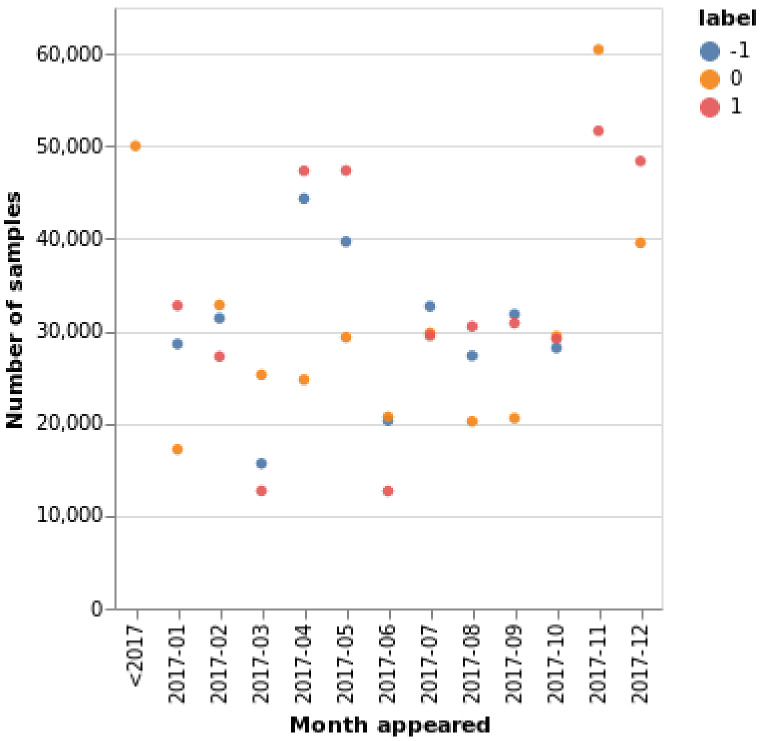


Figure-5: no of samples appeared month wise

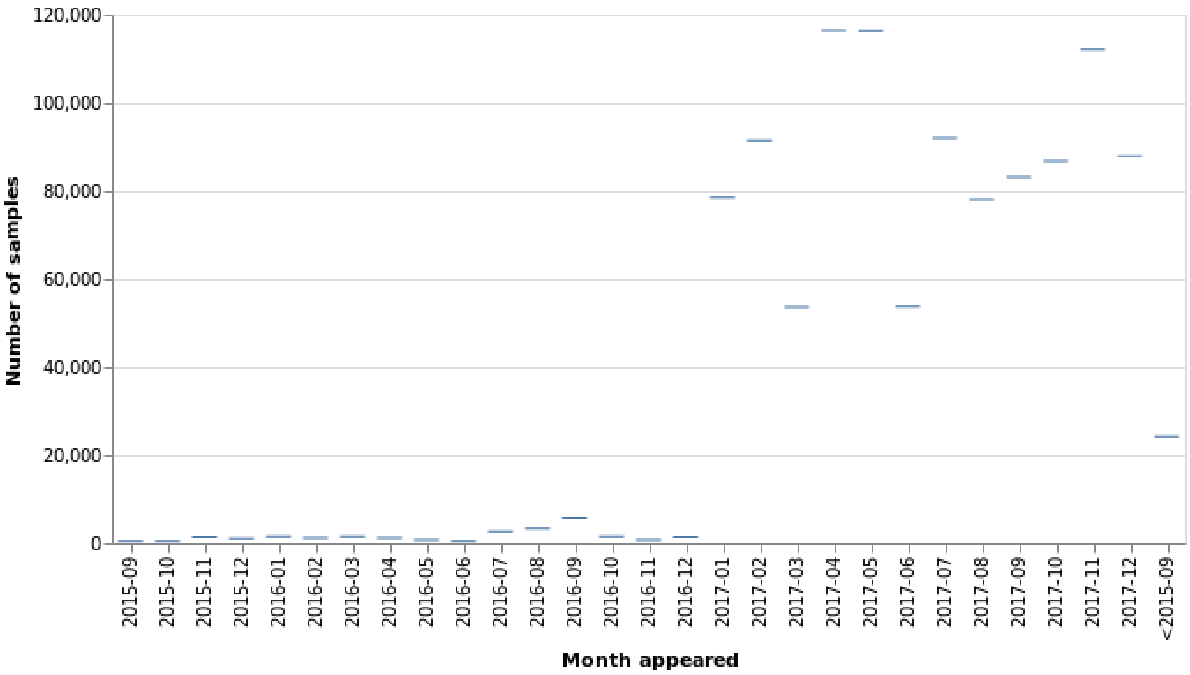


Figure-6: no of samples appeared month wise

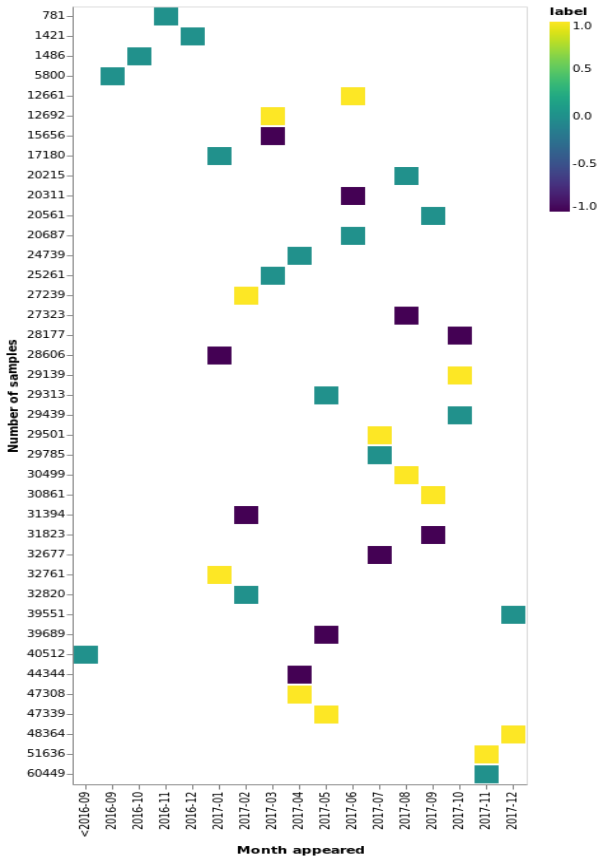


Figure-7: heat map representation

Conclusion

EMBER data is large public dataset which is used for malware detection using various using machine learning models that includes neural concept such as convolution neural network and feed forward neural network. These models classifies available EMBER data into malicious, unlabelled and benign. The performance measure is analysed by various concepts such as precision, recall, f-score and support.

Precision – Percentage of results that are relevant

Recall – refers to classifying the total relevant results that are classified into unlabelled and malicious.

F-score – it's a measure testing accuracy considering both precision and recall.

This measures are used to effectively identify how well the files has been classified as benign, unlabelled and malicious. And we have concluded Feedforward neural network is slightly better than cnn.

8. Reference papers(Give all references)

1. https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

2. https://dzone.com/articles/malware-detection-with-convolutional-neural-networ

3. http://www.covert.io/research-papers/deep-learning-security/Convolutional%20Neural%20Networks%20for%20Malware%20Classification.pdf

4. https://www.sciencedirect.com/science/article/pii/S1877050917316204

CODE

IMPORTING LIBRARIES

In [1]:

```
import os
from data.ember import ember
import numpy as np
import pandas as pd
import altair as alt
alt.renderers.enable('notebook')
import lightgbm as lgb
import matplotlib.pylab as plt
from vega_datasets import data
from sklearn.metrics import roc_auc_score, roc_curve
```

In [2]:

```
data_dir = "/home/hemanth/data/ember/ember"
```

In [3]:

```
ember.create_vectorized_features(data_dir)
_ = ember.create_metadata(data_dir)
```

```
Vectorizing training set
100%|██████████| 900000/900000 [13:37<00:00, 1101.56it/s]
Vectorizing test set
100%|██████████| 200000/200000 [02:54<00:00, 1143.48it/s]
```

In [4]:

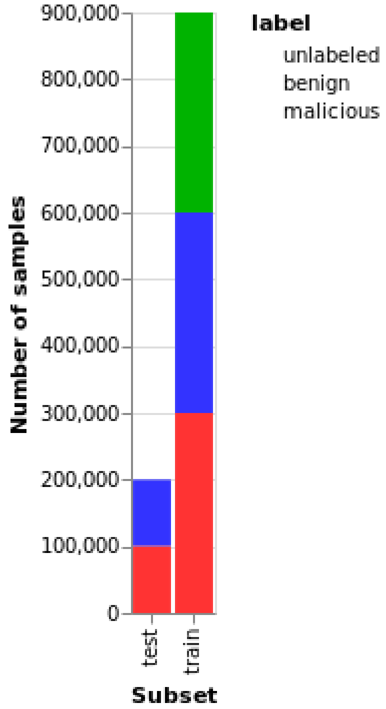
```
emberdf = ember.read_metadata(data_dir)
X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
lgbm_model = lgb.Booster(model_file=os.path.join(data_dir, "ember_model_2017.txt"))
```

```
/home/hemanth/.local/lib/python3.6/site-packages/numpy/lib/arraysetops.py:522: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```

In [5]:

```
plotdf = emberdf.copy()

gbdf = plotdf.groupby(["label", "subset"]).count().reset_index()

alt.Chart(gbdf).mark_bar().encode(

    alt.X('subset:O', axis=alt.Axis(title='Subset')),

    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),

    alt.Color('label:N', scale=alt.Scale(range=["#00b300", "#3333ff", "#ff3333"]),
legend=alt.Legend(values=["unlabeled", "benign", "malicious"]))

)
```

Out[5]:

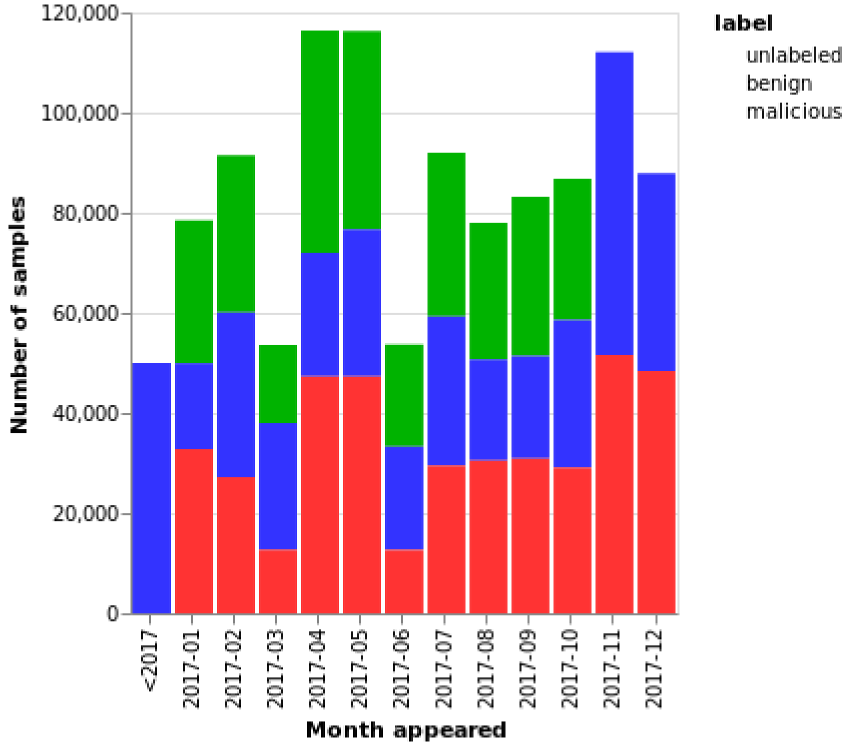


In [5]:

```
plotdf = emberdf.copy()

plotdf.loc[plotdf["appeared"] < "2017-01", "appeared"] = " <2017"

gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()

alt.Chart(gbdf).mark_bar().encode(

    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),

    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),
```

```
    alt.Color('label:N', scale=alt.Scale(range=["#00b300", "#3333ff", "#ff3333"]),
legend=alt.Legend(values=["unlabeled", "benign", "malicious"]))
)
```

Out[5]:



In [6]:

```
y_test_pred = lgbm_model.predict(X_test)
y_train_pred = lgbm_model.predict(X_train)
emberdf["y_pred"] = np.hstack((y_train_pred, y_test_pred))
```

In [7]:

```
def get_fpr(y_true, y_pred):
    nbenign = (y_true == 0).sum()
    nfalse = (y_pred[y_true == 0] == 1).sum()
    return nfalse / float(nbenign)



def find_threshold(y_true, y_pred, fpr_target):
    thresh = 0.0
```

```
    fpr = get_fpr(y_true, y_pred > thresh)
    while fpr > fpr_target and thresh < 1.0:
        thresh += 0.001
        fpr = get_fpr(y_true, y_pred > thresh)
    return thresh, fpr

testdf = emberdf[emberdf["subset"] == "test"]
print("ROC AUC:", roc_auc_score(testdf.label, testdf.y_pred))
print()

threshold, fpr = find_threshold(testdf.label, testdf.y_pred, 0.01)
fnr = (testdf.y_pred[testdf.label == 1] < threshold).sum() / float((testdf.label == 1).sum())
print("Ember Model Performance at 1% FPR:")
print("Threshold: {:.3f}".format(threshold))
print("False Positive Rate: {:.3f}%".format(fpr * 100))
print("False Negative Rate: {:.3f}%".format(fnr * 100))
print("Detection Rate: {}%".format(100 - fnr * 100))
print()

threshold, fpr = find_threshold(testdf.label, testdf.y_pred, 0.001)
fnr = (testdf.y_pred[testdf.label == 1] < threshold).sum() / float((testdf.label == 1).sum())
print("Ember Model Performance at 0.1% FPR:")
print("Threshold: {:.3f}".format(threshold))
print("False Positive Rate: {:.3f}%".format(fpr * 100))
print("False Negative Rate: {:.3f}%".format(fnr * 100))
print("Detection Rate: {}%".format(100 - fnr * 100))
ROC AUC: 0.9991123269999999

Ember Model Performance at 1% FPR:
Threshold: 0.529
```

False Positive Rate: 0.998%

False Negative Rate: 1.838%

Detection Rate: 98.162%

Ember Model Performance at 0.1% FPR:

Threshold: 0.871

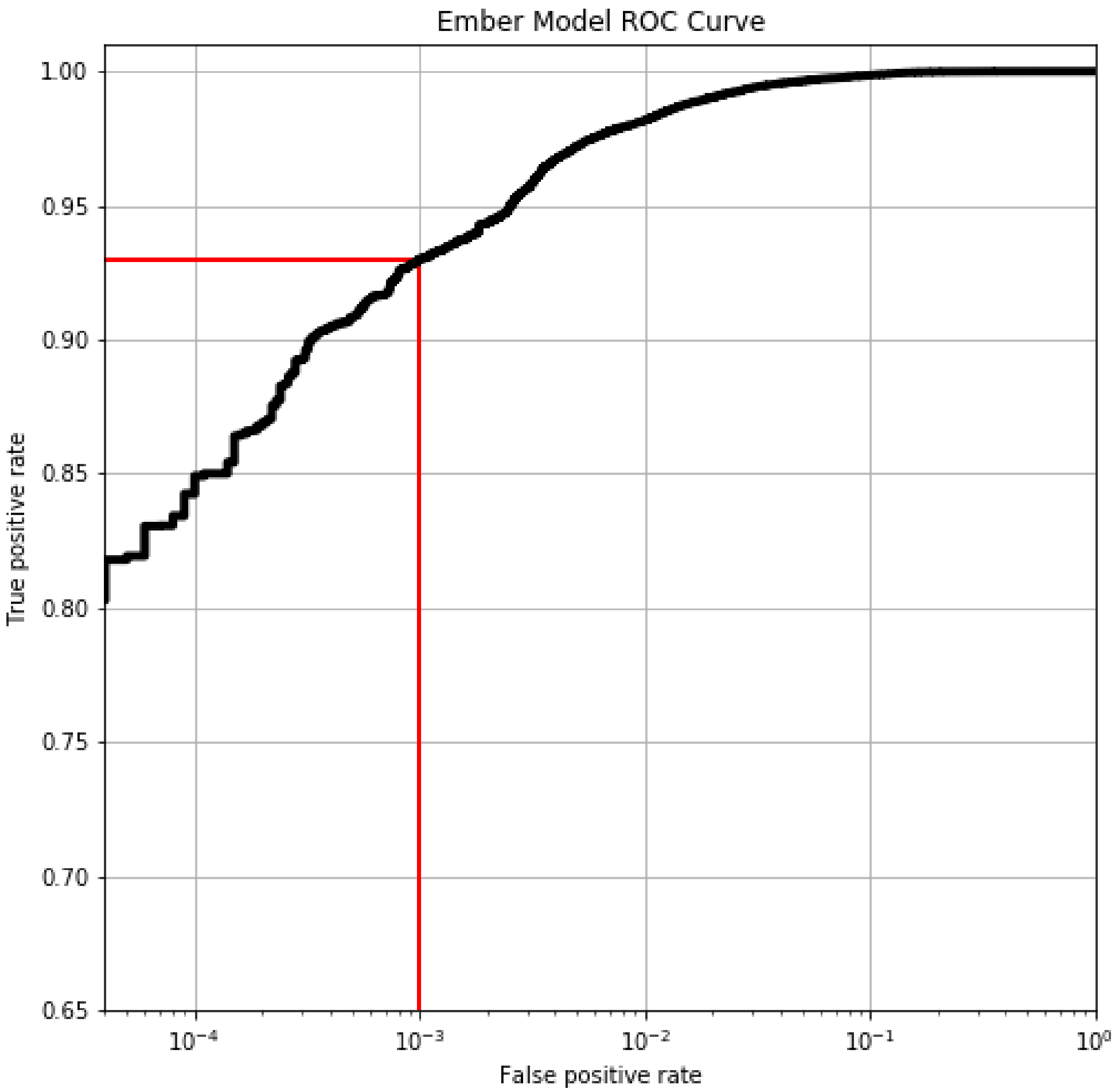
False Positive Rate: 0.099%

False Negative Rate: 7.009%

Detection Rate: 92.991%

In [8]:

```
plt.figure(figsize=(8, 8))
fpr_plot, tpr_plot, _ = roc_curve(testdf.label, testdf.y_pred)
plt.plot(fpr_plot, tpr_plot, lw=4, color='k')
plt.gca().set_xscale("log")
plt.yticks(np.arange(22) / 20.0)
plt.xlim([4e-5, 1.0])
plt.ylim([0.65, 1.01])
plt.gca().grid(True)
plt.vlines(fpr, 0, 1 - fnr, color="r", lw=2)
plt.hlines(1 - fnr, 0, fpr, color="r", lw=2)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
_ = plt.title("Ember Model ROC Curve")
```



In [9]:

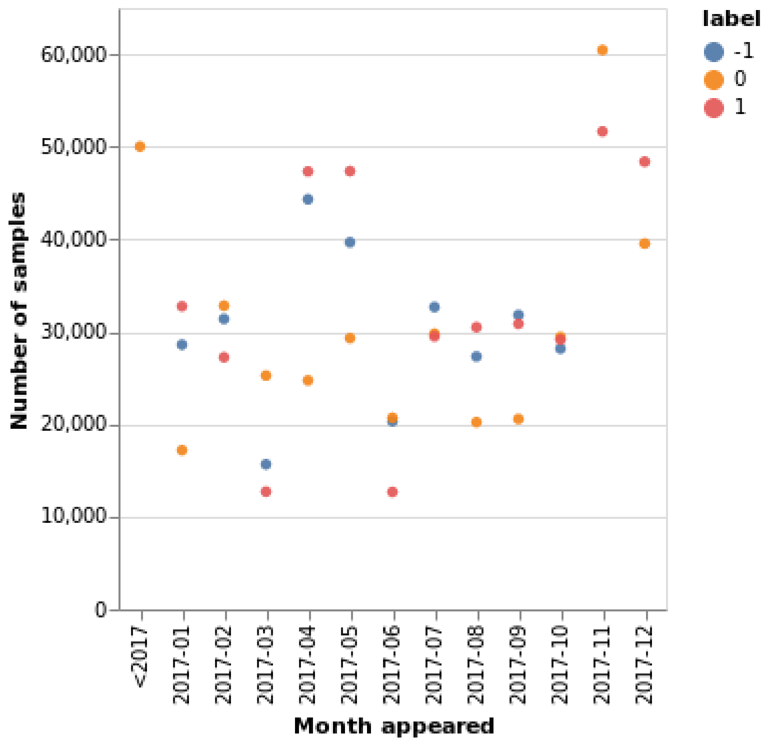
```
fig = plt.figure(figsize=(8, 8))

testdf[testdf["label"] == 0].y_pred.hist(range=[0, 1], bins=10, color="#3333ff", alpha=0.8, label="benign")

testdf[testdf["label"] == 1].y_pred.hist(range=[0, 1], bins=10, color="#ff3333", alpha=0.8, label="malicious")

plt.gca().set_yscale("log", nonposy="clip")

plt.gca().grid(False)

plt.xlabel("Score")

plt.ylabel("Count")
```

```
_ = plt.title("Ember Test Set Model Score")
```

In [10]:

```
alt.data_transformers.enable('default', max_rows=None)
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2017-01", "appeared"] = " <2017"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_circle(
    color='red',
    opacity=0.9
).encode(
    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),
    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),
    alt.Color('label:N')
)
```

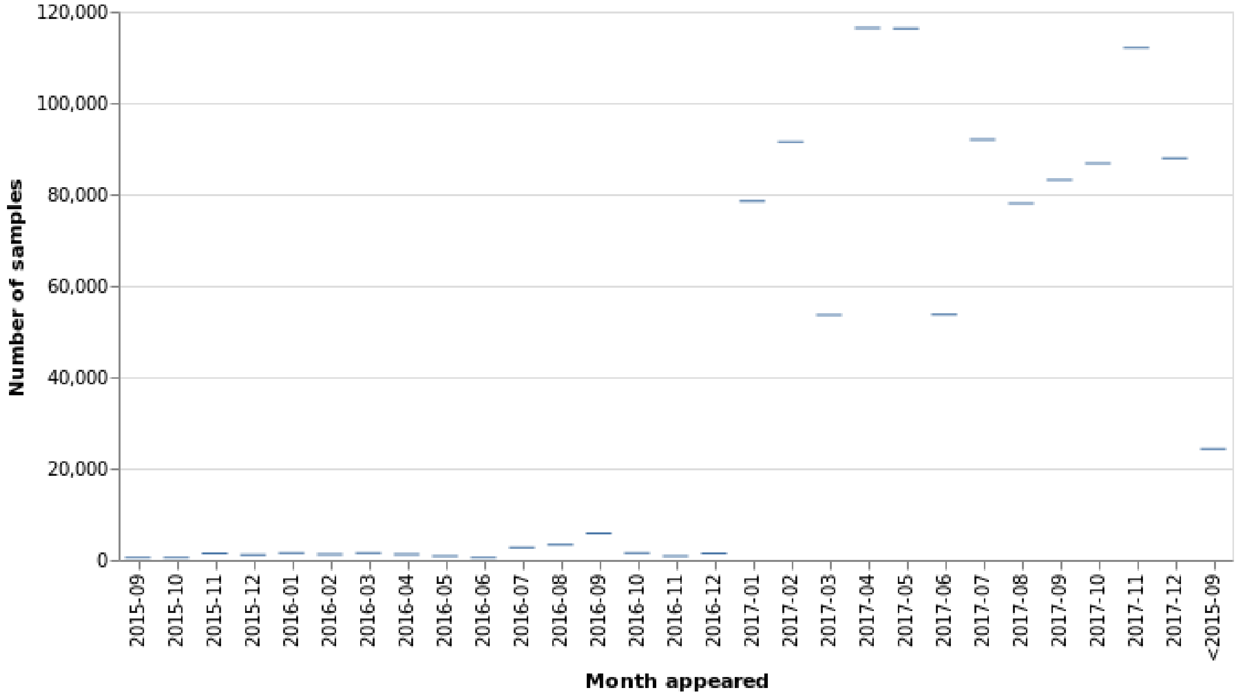
Out[10]:



In [11]:

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2015-09", "appeared"] = "<2015-09"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_tick().encode(
    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),
    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples'))
)
```

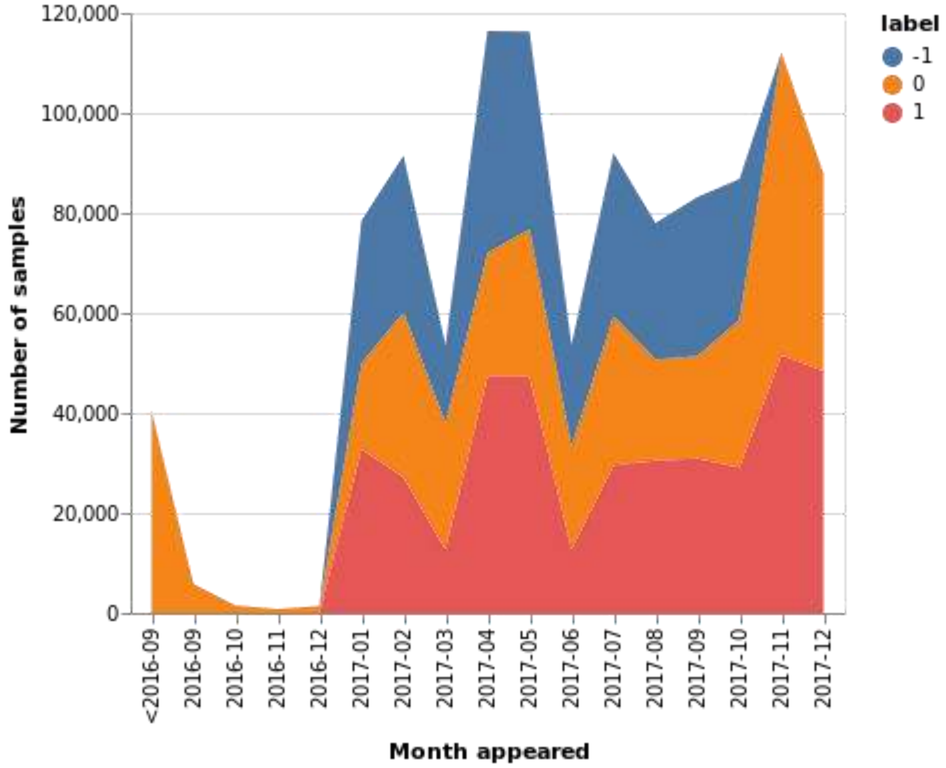
Out[11]:



In [12]:

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2016-09", "appeared"] = " <2016-09"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_area().encode(
    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),
    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),
    alt.Color('label:N')
)
```

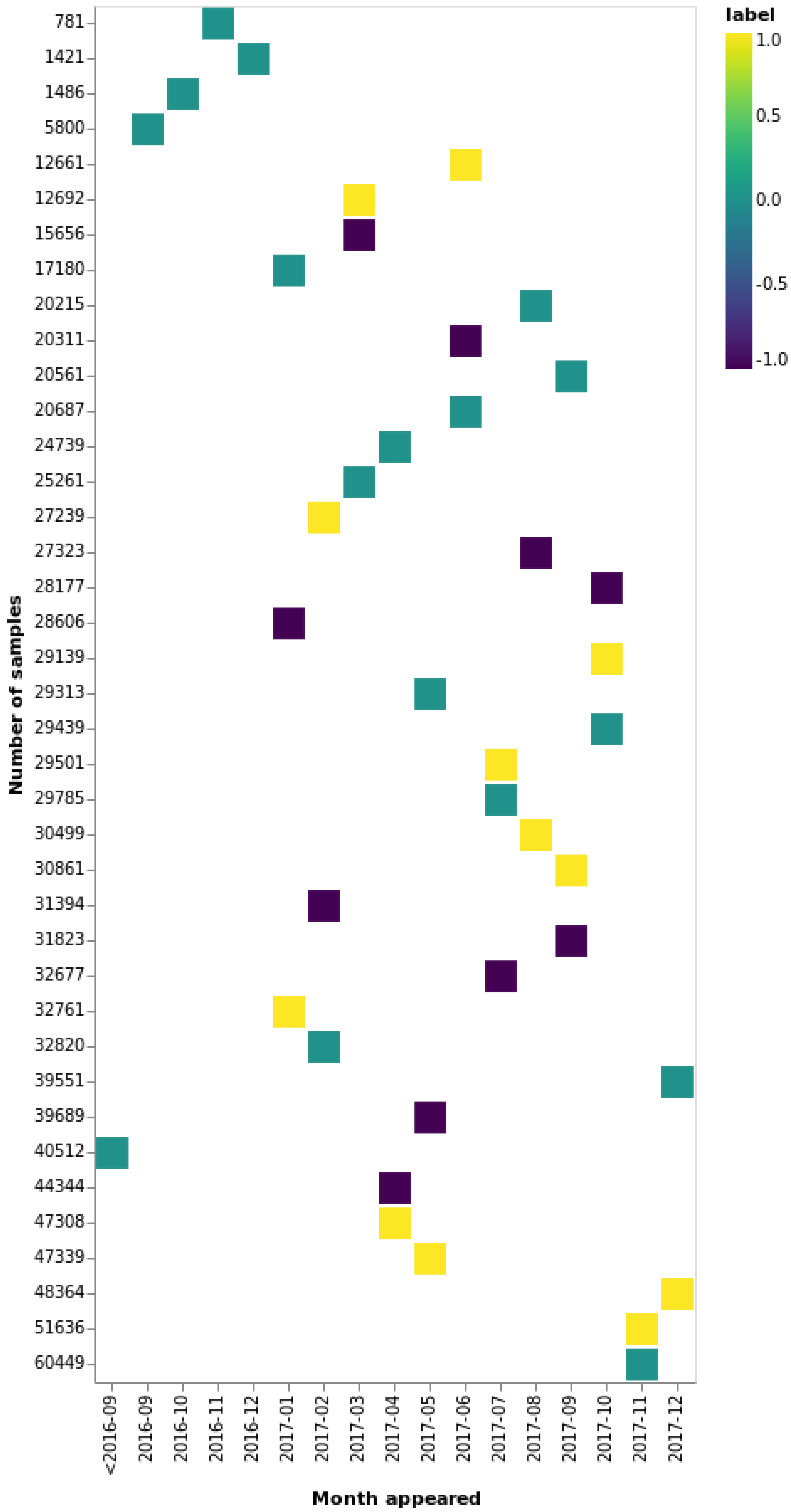
Out[12]:



In [13]:

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2016-09", "appeared"] = " <2016-09"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_rect().encode(
    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),
    alt.Y('sum(sha256):O', axis=alt.Axis(title='Number of samples')),
    alt.Color('label:Q')
)
```

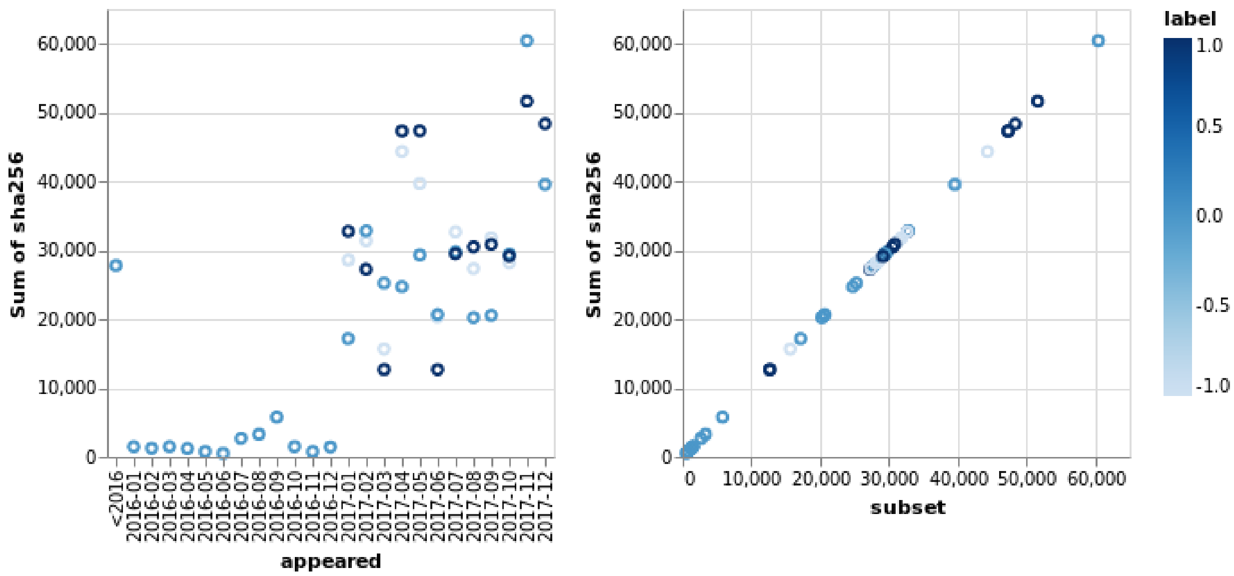
Out[13]:


label
1.0
0.5
0.0
-0.5
-1.0
Number of samples
781
1421
1486
5800
12661
12692
15656
17180
20215
20311
20561
20687
24739
25261
27239
27323
28177
28606
29139
29313
29439
29501
29785
30499
30861
31394
31823
32677
32761
32820
39551
39689
40512
44344
47308
47339
48364
51636
60449
<2016-09
2016-09
2016-10
2016-11
2016-12
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
Month appeared

In [14]:

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2016-01", "appeared"] = " <2016"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
brush = alt.selection(type='interval', resolve='global')
base = alt.Chart(gbdf).mark_point().encode(
    y='sum(sha256)',
    color=alt.condition(brush, 'label', alt.ColorValue('gray'))
).add_selection(
    brush
).properties(
    width=250,
    height=250
)

base.encode(x='appeared') | base.encode(x='subset')
```

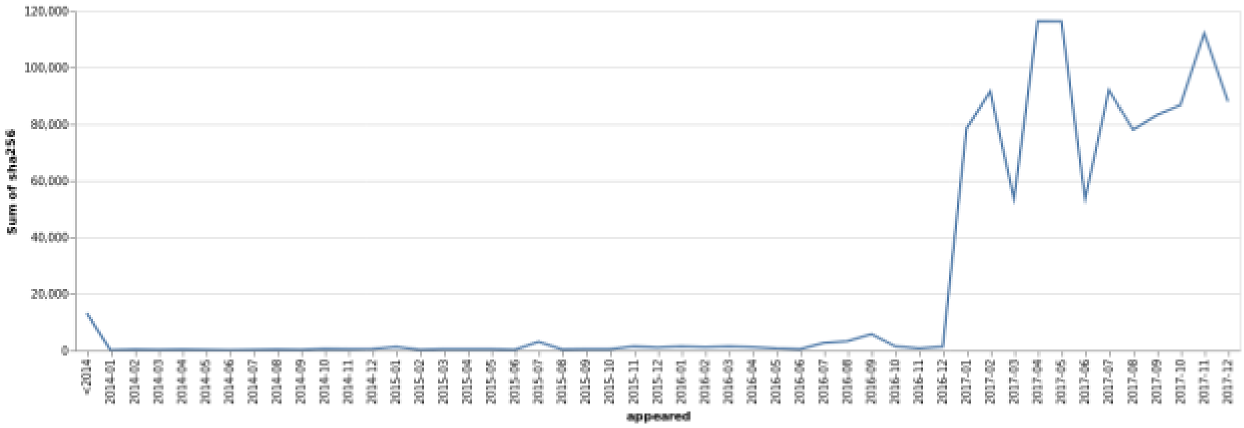
Out[14]:



In [15]:

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2014-01", "appeared"] = " <2014"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_line().encode(
    x='appeared',
    y='sum(sha256)'
)
```

Out[15]:



In [16]:

```
emberdf.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1100000 entries, 0 to 1099999
Data columns (total 5 columns):
sha256      1100000 non-null object
appeared    1100000 non-null object
subset      1100000 non-null object
label       1100000 non-null int64
y_pred      1100000 non-null float64
dtypes: float64(1), int64(1), object(3)
memory usage: 50.4+ MB
```

In [ ]:

```
import pandas as pd
import torch
import numpy as np
import sys
import gc

sys.path.append('../libs')
import ember

from tqdm import tqdm_notebook
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from torch import nn
from tensorboardX import SummaryWriter
from sklearn.metrics import classification_report, f1_score
from torch.utils.data import Dataset, DataLoader
```

. . .

In [2]:

```
data_dir='../data/ember'
logs='../logs'
batch_size=256
gpu_id=0
train_epochs=30
learning_rate=0.01
```

. . .

In [3]:

```
#Uncomment if data folder is newly created.
#ember.create_vectorized_features(data_dir)
#ember.create_metadata(data_dir)
```

. . .

In [4]:

```
X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
```

. . .

Filter malwares which are untagged

Filter malwares which are untagged

In [5]:

```
X_train=X_train[y_train!=-1]
y_train=y_train[y_train!=-1]
X_test=np.array(X_test)
y_test=np.array(y_test)
```

. . .

In [6]:

```
scaler=StandardScaler()
scaler.fit(X_train)
```

Out[6]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

. . .

In [7]:

```
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

. . .

In [8]:

```
class EmberDataset(Dataset):
    def __init__(self,X,y):
        self.X=X
        self.y=y
    def __len__(self):
        return self.X.shape[0]
    def __getitem__(self, idx):
```

```
        return self.X[idx].astype(np.float32),self.y[idx].astype(np.long)
```

. . .

Create train and test dataloaders

Create train and test dataloaders

In [9]:

```
train_dataset=EmberDataset(X_train,y_train)
test_dataset=EmberDataset(X_test,y_test)
train_dl=DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
test_dl=DataLoader(test_dataset,batch_size=batch_size)
```

. . .

In [10]:

```
class ConvDoc(nn.Module):
    def __init__(self):
        super(ConvDoc,self).__init__()
        input_dim=2351
        self.linear=nn.Linear(2351,128)
        self.linear2=nn.Linear(128,64)
        self.dropout=nn.Dropout(0.5)
        self.clf=nn.Linear(64,2)
```

```
    def forward(self,inputs):
        l_out=self.dropout(torch.tanh(self.linear(inputs)))
        l2_out=self.dropout(torch.tanh(self.linear2(l_out)))
        return nn.functional.log_softmax(self.clf(l2_out),dim=-1)
```

. . .

In [11]:

```
model=ConvDoc()
if gpu_id>=0:
    model=model.cuda(gpu_id)
criterion=nn.NLLLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate)
```

. . .

In [12]:

```
summary_writer= SummaryWriter(log_dir=logs)
global_step=0
for epoch in tqdm_notebook(range(train_epochs),desc='Epochs'):  # loop over the dataset
multiple times

    running_loss = 0.0
    print('-'*50)
    print('Epoch %d'%epoch)
    for i, data in enumerate(tqdm_notebook(train_dl), 0):
```

```
        # get the inputs
        inputs, labels = data
        if(gpu_id>=0):
            inputs=inputs.cuda(gpu_id)
            labels=labels.cuda(gpu_id)
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        summary_writer.add_scalar('FC/Batch loss',loss,global_step)
        global_step=global_step+1

    # Test evaluation
    targets=[]
    preds=[]
    model=model.eval()
    for i, data in enumerate(test_dl,0):
        inputs, labels = data
        if(gpu_id>=0):
            inputs=inputs.cuda(gpu_id)
            labels=labels.cuda(gpu_id)
        outputs=model(inputs)
        outputs=list(outputs.argmax(-1).cpu().detach().numpy())
        labels=list(labels.cpu().numpy())
        targets.extend(labels)
        preds.extend(outputs)
```

```
    model=model.train()
    print('Test data score:')
    print(classification_report(targets,preds))
    summary_writer.add_scalar('FC/F1 score',f1_score(targets,preds),global_step)
        #print('[%d, %5d] loss: %.3f' %
        #      (epoch + 1, i + 1, loss))
print('Finished Training')
HBox(children=(IntProgress(value=0, description='Epochs', max=30), HTML(value='')))
--------------------------------------------------
Epoch 0
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support


          0       0.97      0.97      0.97    100000
          1       0.97      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000


--------------------------------------------------
Epoch 1
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support


          0       0.96      0.98      0.97    100000
          1       0.98      0.96      0.97    100000

avg / total       0.97      0.97      0.97    200000
```

```
-------------------------------------------------
Epoch 2
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 3
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.98      0.97    100000
          1       0.98      0.96      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 4
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.98      0.97    100000
          1       0.98      0.96      0.97    100000
```

```
avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 5
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 6
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.98      0.97    100000
          1       0.98      0.96      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 7
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support
```

```
         0       0.97      0.97      0.97    100000
         1       0.97      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

--------------------------------------------------
Epoch 8
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.97      0.98      0.97    100000
         1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

--------------------------------------------------
Epoch 9
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.97      0.98      0.97    100000
         1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

--------------------------------------------------
Epoch 10
```

```
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

--------------------------------------------------
Epoch 11
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

--------------------------------------------------
Epoch 12
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000
```

Epoch 13

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	100000
1	0.98	0.97	0.97	100000
avg / total	0.97	0.97	0.97	200000

Epoch 14

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	100000
1	0.98	0.97	0.97	100000
avg / total	0.97	0.97	0.97	200000

Epoch 15

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	100000

```
         1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 16
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 17
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 18
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
```

```
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 19
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 20
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
```

Epoch 21

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	100000
1	0.98	0.97	0.97	100000
avg / total	0.97	0.97	0.97	200000

Epoch 22

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	100000
1	0.98	0.97	0.98	100000
avg / total	0.98	0.98	0.98	200000

Epoch 23

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	100000
1	0.98	0.97	0.98	100000

```
avg / total       0.98      0.98      0.98    200000


-------------------------------------------------
Epoch 24
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support


          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000


-------------------------------------------------
Epoch 25
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support


          0       0.97      0.98      0.97    100000
          1       0.98      0.97      0.97    100000

avg / total       0.97      0.97      0.97    200000


-------------------------------------------------
Epoch 26
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support
```

```
         0       0.96      0.98      0.97    100000
         1       0.98      0.96      0.97    100000

avg / total       0.97      0.97      0.97    200000

-------------------------------------------------
Epoch 27
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.97      0.98      0.98    100000
         1       0.98      0.97      0.98    100000

avg / total       0.98      0.98      0.98    200000

-------------------------------------------------
Epoch 28
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.97      0.98      0.98    100000
         1       0.98      0.97      0.98    100000

avg / total       0.98      0.98      0.98    200000

-------------------------------------------------
Epoch 29
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
```

```
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.98      0.97    100000
          1       0.98      0.96      0.97    100000

avg / total       0.97      0.97      0.97    200000



Finished Training
. . .
CloseExpandOpen in PagerClose
```

```python
import pandas as pd
import torch
import numpy as np
import pickle
import sys
import os
import gc

sys.path.append('../libs')
import ember

from tqdm import tqdm_notebook
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from torch import nn
from sklearn.externals import joblib
```

```
from tensorboardX import SummaryWriter
from sklearn.metrics import classification_report, f1_score
from torch.utils.data import Dataset, DataLoader
```

. . .

In [13]:

```
data_dir='../data/ember'
logs='../logs'
models_dir='../models/ConvDoc'
batch_size=256
gpu_id=0
train_epochs=30
learning_rate=0.01
```

. . .

In [3]:

```
#Uncomment if data folder is newly created.
#ember.create_vectorized_features(data_dir)
#ember.create_metadata(data_dir)
```

. . .

In [4]:

```
X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
```

. . .

Filter malwares which are untagged

Filter malwares which are untagged

In [5]:

```
X_train=X_train[y_train!=-1]
y_train=y_train[y_train!=-1]
X_test=np.array(X_test)
y_test=np.array(y_test)
```

. . .

In [6]:

```
scaler=StandardScaler()
scaler.fit(X_train)
```

Out[6]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

. . .

In [7]:

```
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

. . .

In [8]:

```
class EmberDataset(Dataset):
    def __init__(self,X,y):
        self.X=X
        self.y=y
    def __len__(self):
        return self.X.shape[0]
    def __getitem__(self, idx):
        return self.X[idx].astype(np.float32),self.y[idx].astype(np.long)
```

. . .

Create train and test dataloaders

Create train and test dataloaders

In [9]:

```
train_dataset=EmberDataset(X_train,y_train)
test_dataset=EmberDataset(X_test,y_test)
train_dl=DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
test_dl=DataLoader(test_dataset,batch_size=batch_size)
```

. . .

In [10]:

```
class ConvDoc(nn.Module):
    def __init__(self):
        super(ConvDoc,self).__init__()
        input_dim=2351
        self.conv1=nn.Conv1d(1,8,3,stride=2)
        self.conv2=nn.Conv1d(8,16,3,stride=4)
        self.conv3=nn.Conv1d(16,4,15,stride=5,dilation=10)
        self.dropout=nn.Dropout(0.2)

        self.clf=nn.Linear(124,2)
    def forward(self,inputs):
        batch_size=inputs.shape[0]
        inputs=inputs.unsqueeze(1)
        conv1=self.dropout(torch.relu(self.conv1(inputs)))
        conv2=self.dropout(torch.relu(self.conv2(conv1)))
        conv3=torch.relu(self.conv3(conv2))
        latent=conv3.view([batch_size,-1])
        return nn.functional.log_softmax(self.clf(latent),dim=-1)
```

. . .

In [11]:

```
model=ConvDoc()
if gpu_id>=0:
    model=model.cuda(gpu_id)
criterion=nn.NLLLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=learning_rate)
```

. . .

In [12]:

```
summary_writer= SummaryWriter(log_dir=logs)
global_step=0
for epoch in tqdm_notebook(range(train_epochs),desc='Epochs'):  # loop over the dataset multiple times

    running_loss = 0.0
    print('-'*50)
    print('Epoch %d'%epoch)
    for i, data in enumerate(tqdm_notebook(train_dl), 0):

        # get the inputs
        inputs, labels = data
        if(gpu_id>=0):
            inputs=inputs.cuda(gpu_id)
            labels=labels.cuda(gpu_id)
```

```
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    summary_writer.add_scalar('ConvDoc/Batch loss',loss,global_step)
    global_step=global_step+1

# Test evaluation
targets=[]
preds=[]
model=model.eval()
for i, data in enumerate(test_dl,0):
    inputs, labels = data
    if(gpu_id>=0):
        inputs=inputs.cuda(gpu_id)
        labels=labels.cuda(gpu_id)
    outputs=model(inputs)
    outputs=list(outputs.argmax(-1).cpu().detach().numpy())
    labels=list(labels.cpu().numpy())
    targets.extend(labels)
    preds.extend(outputs)
model=model.train()
print('Test data score:')
print(classification_report(targets,preds))
summary_writer.add_scalar('ConvDoc/F1 score',f1_score(targets,preds),global_step)
    #print('[%d, %5d] loss: %.3f' %
```

```
#     (epoch + 1, i + 1, loss))
print('Finished Training')
HBox(children=(IntProgress(value=0, description='Epochs', max=30), HTML(value='')))
--------------------------------------------------
Epoch 0
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.95      0.93      0.94    100000
          1       0.93      0.95      0.94    100000

avg / total       0.94      0.94      0.94    200000

--------------------------------------------------
Epoch 1
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.95      0.93      0.94    100000
          1       0.93      0.96      0.94    100000

avg / total       0.94      0.94      0.94    200000

--------------------------------------------------
Epoch 2
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support
```

```
         0       0.96      0.94      0.95    100000
         1       0.94      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 3
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.93      0.96      0.95    100000
         1       0.96      0.93      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 4
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

         0       0.96      0.93      0.95    100000
         1       0.94      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 5
```

```
HBox(children=(IntProgress(value=0, max=2344), HTML(value="")))
Test data score:
             precision    recall  f1-score   support

          0       0.95      0.94      0.95    100000
          1       0.94      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 6
HBox(children=(IntProgress(value=0, max=2344), HTML(value="")))
Test data score:
             precision    recall  f1-score   support

          0       0.95      0.95      0.95    100000
          1       0.95      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 7
HBox(children=(IntProgress(value=0, max=2344), HTML(value="")))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.93      0.95    100000
          1       0.93      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000
```

Epoch 8

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.95	0.94	0.95	100000
1	0.94	0.95	0.95	100000
avg / total	0.95	0.95	0.95	200000

Epoch 9

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.95	0.94	0.94	100000
1	0.94	0.95	0.95	100000
avg / total	0.95	0.95	0.95	200000

Epoch 10

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.96	0.95	0.95	100000

```
          1       0.95      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 11
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.94      0.95      0.95    100000
          1       0.95      0.94      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 12
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.92      0.97      0.94    100000
          1       0.96      0.92      0.94    100000

avg / total       0.94      0.94      0.94    200000

--------------------------------------------------
Epoch 13
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
```

```
             precision    recall  f1-score   support

          0       0.96      0.94      0.95    100000
          1       0.94      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

------------------------------------------------
Epoch 14
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.95      0.95      0.95    100000
          1       0.95      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000

------------------------------------------------
Epoch 15
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.94      0.95      0.95    100000
          1       0.95      0.94      0.95    100000

avg / total       0.95      0.95      0.95    200000

------------------------------------------------
```

Epoch 16

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.96	0.93	0.94	100000
1	0.93	0.96	0.95	100000
avg / total	0.95	0.95	0.95	200000

Epoch 17

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.94	0.96	0.95	100000
1	0.96	0.94	0.95	100000
avg / total	0.95	0.95	0.95	200000

Epoch 18

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.94	0.95	0.94	100000
1	0.95	0.94	0.94	100000

```
avg / total       0.94      0.94      0.94    200000

--------------------------------------------------
Epoch 19
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.93      0.95    100000
          1       0.93      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 20
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

          0       0.96      0.94      0.95    100000
          1       0.94      0.96      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 21
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support
```

```
           0       0.95      0.95      0.95    100000
           1       0.95      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 22
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

           0       0.93      0.97      0.95    100000
           1       0.97      0.92      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 23
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
Test data score:
             precision    recall  f1-score   support

           0       0.93      0.97      0.95    100000
           1       0.96      0.93      0.95    100000

avg / total       0.95      0.95      0.95    200000

-------------------------------------------------
Epoch 24
HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))
```

```
Test data score:
             precision    recall  f1-score   support


          0       0.95      0.95      0.95    100000
          1       0.95      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 25
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support


          0       0.94      0.96      0.95    100000
          1       0.96      0.94      0.95    100000

avg / total       0.95      0.95      0.95    200000

--------------------------------------------------
Epoch 26
HBox(children=(IntProgress(value=0, max=2344), HTML(value=")))
Test data score:
             precision    recall  f1-score   support


          0       0.95      0.95      0.95    100000
          1       0.95      0.95      0.95    100000

avg / total       0.95      0.95      0.95    200000
```

Epoch 27

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.95	0.96	0.95	100000
1	0.95	0.95	0.95	100000
avg / total	0.95	0.95	0.95	200000

Epoch 28

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.94	0.95	0.94	100000
1	0.95	0.94	0.94	100000
avg / total	0.94	0.94	0.94	200000

Epoch 29

HBox(children=(IntProgress(value=0, max=2344), HTML(value='')))

Test data score:

	precision	recall	f1-score	support
0	0.96	0.94	0.95	100000
1	0.94	0.96	0.95	100000

avg / total 0.95 0.95 0.95 200000

Finished Training

. . .

05 Save Trained model

05 Save Trained model¶

In [18]:

```
torch.save(model.state_dict(), os.path.join(models_dir,'model.pth'))
joblib.dump(scaler,os.path.join(models_dir,'scaler.pkl'))
```

Out[18]:

['../models/ConvDoc/scaler.pkl']

. . .

CloseExpandOpen in PagerClose

MACHINE LEARNING LAB ASSIGNMENT – 4

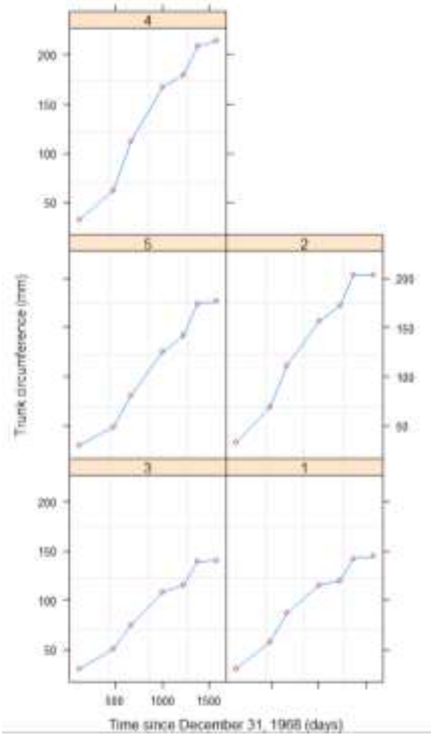
NAME – MARIA ALEX KUZHIPPALLIL

REG – 16BCE2190

DATA SET – ORANGE

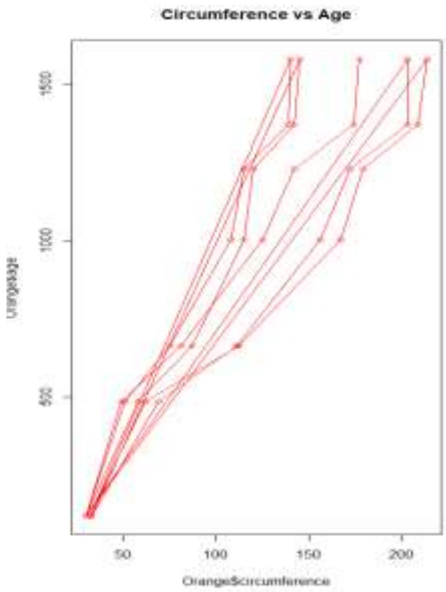
```
Grouped Data: circumference ~ age | Tree
   Tree  age circumference
1     1  118            30
2     1  484            58
3     1  664            87
4     1 1004           115
5     1 1231           120
6     1 1372           142
7     1 1582           145
8     2  118            33
9     2  484            69
10    2  664           111
11    2 1004           156
12    2 1231           172
13    2 1372           203
14    2 1582           203
15    3  118            30
16    3  484            51
17    3  664            75
18    3 1004           108
19    3 1231           115
20    3 1372           139
21    3 1582           140
22    4  118            32
23    4  484            62
24    4  664           112
25    4 1004           167
26    4 1231           179
27    4 1372           209
28    4 1582           214
29    5  118            30
30    5  484            49
31    5  664            81
32    5 1004           125
33    5 1231           142
34    5 1372           174
35    5 1582           177
```

```
> plot(Orange,type="o",col="red")
```



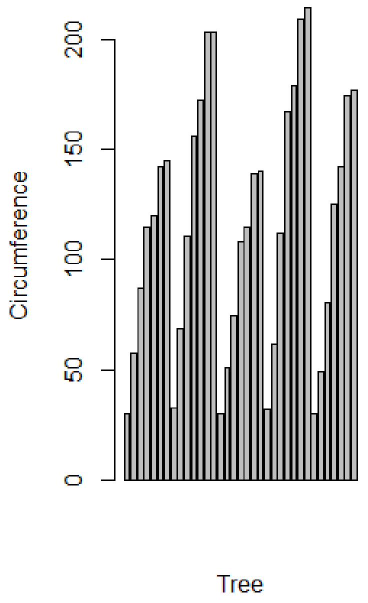
This linear plot describes about various group of trees and their circumference between the range 0 – 200. Totally Orange has 5 groups hence 5 linear plots.

```
> plot(Orange$age~Orange$circumference,type="o",col="red",main="Circumfere
nce vs Age")
```



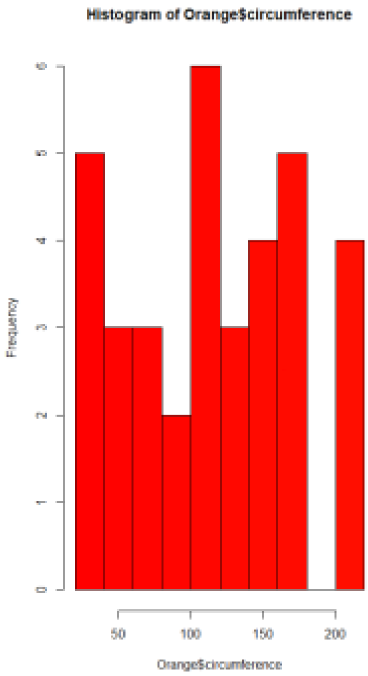
This is linear plot depicts the relationship between the orange trees age and orange trees circumference.

```
> barplot(Orange$circumference,ylab="Circumference",xlab="Tree")
```



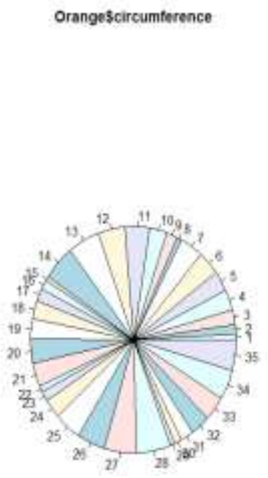
This bar plot depicts different values of circumference exhibited by orange trees and the circumfence values ranging with a minimum value of 30 and 214 holding maximum circumference value.

```
> hist(Orange$circumference,col=heat.colors(max(Orange$circumference)))
```



This histogram represents frequency count for various range of Orange trees circumference.

```
> pie(Orange$circumference,main="Orange$circumference")
```

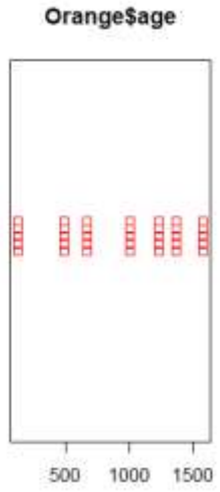


Pie chart represents various circumference value that orange circumference available in Orange data.

```
> dotchart(t(Orange$age),col="red",cex=0.8,main="Orang$age")
```

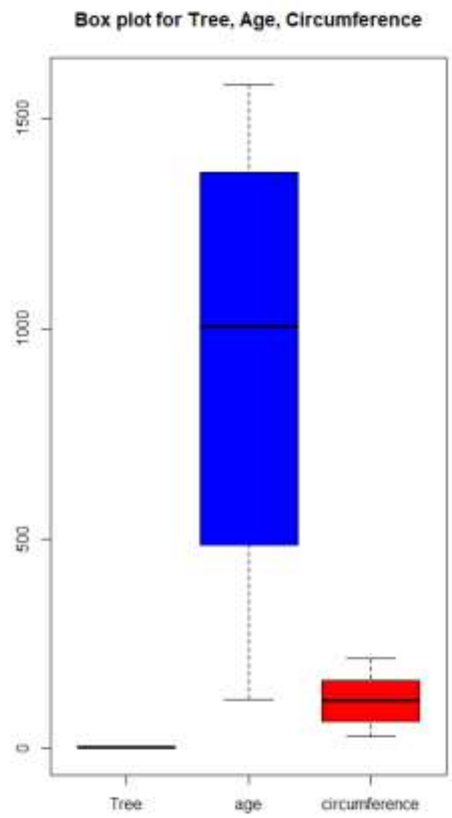
Dot chart values representing various values orange tree age values.

```
> stripchart(Orange$age,method="stack",main="Orange$age",col="red")
```



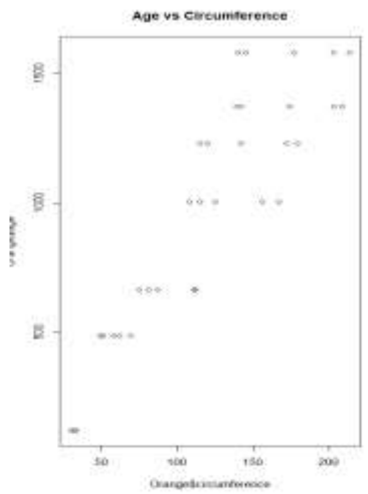
Strip chart which represents various value for age in stack order. Other methods available includes jitter.

```
> boxplot(Orange,main="Box plot for Tree, Age, Circumference",col=c("black","blue","red"))
```



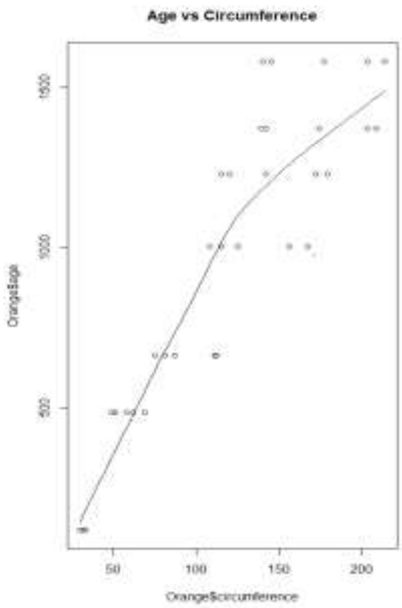
Box plot of 3 different parameters such as Group tree, Orange tree age and circumference of the tree.

```
> plot(Orange$age~Orange$circumference,data=Orange,main="Age vs Circumference")
```



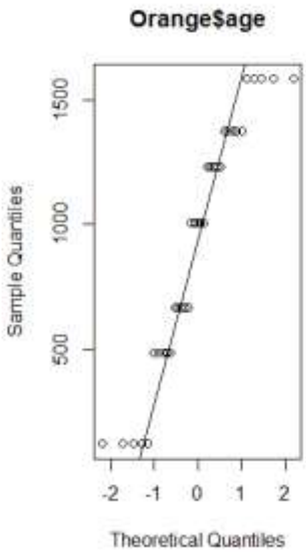
Plot representing the data Orange's Age vs Circumference.

```
> cor(Orange$age,Orange$circumference)
[1] 0.9135189

> scatter.smooth(Orange$age~Orange$circumference,data=Orange,main="Age vs
Circumference")
```



Based on the correlation value it can be observed that the correlation between Age and circumference values are highly correlated and is represented by this scatter plot.

```
> qqnorm(Orange$age,main='Orange$age')
> qqline(Orange$age,main='Orange$age')
```

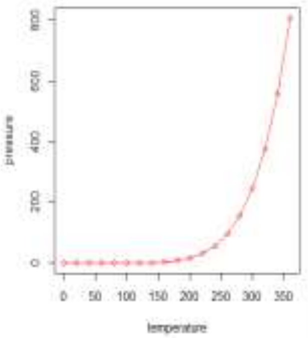


Qnnorm values represents the which shows sample quartiles vs rheoritical quartile values.

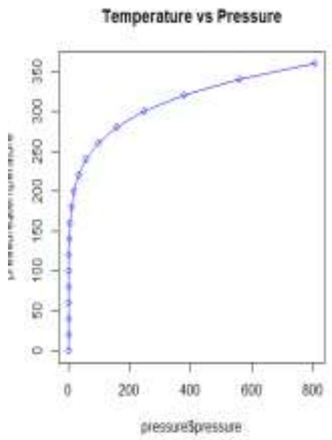
DATA SET – PRESSURE

```
> pressure
   temperature pressure
1            0   0.0002
2           20   0.0012
3           40   0.0060
4           60   0.0300
5           80   0.0900
6          100   0.2700
7          120   0.7500
8          140   1.8500
9          160   4.2000
10         180   8.8000
11         200  17.3000
12         220  32.1000
13         240  57.0000
14         260  96.0000
15         280 157.0000
16         300 247.0000
17         320 376.0000
18         340 558.0000
19         360 806.0000
```

```
> plot(pressure,type="o",col="red")
```

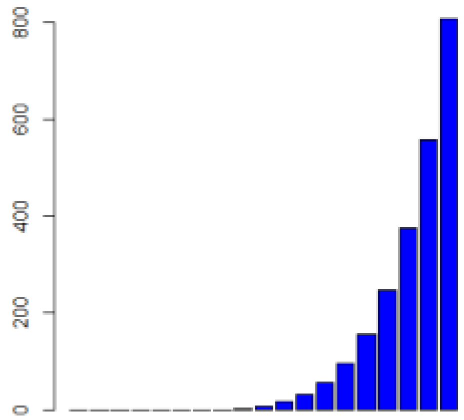



```
> plot(pressure$temperature~pressure$pressure,type="o",col="blue",main="Te
mperature vs Pressure")
```



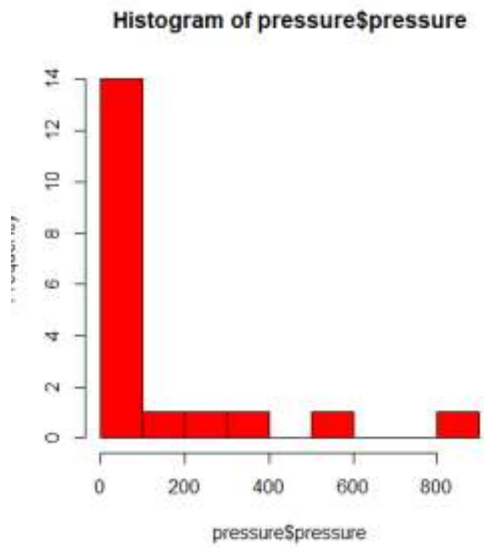
Linear plot depicting temperature vs pressure in data Pressure

```
> barplot(pressure$pressure,col="blue")
```



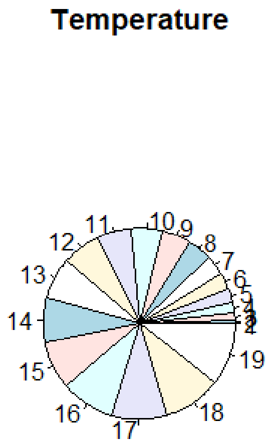
Bar plot representing the pressure

```
> hist(pressure$pressure,col=heat.colors(max(pressure$pressure)))
```



Histogram which represents the frequency count value for various pressure value in Pre ssure data

```
> pie(pressure$temperature,main="Temperature")
```

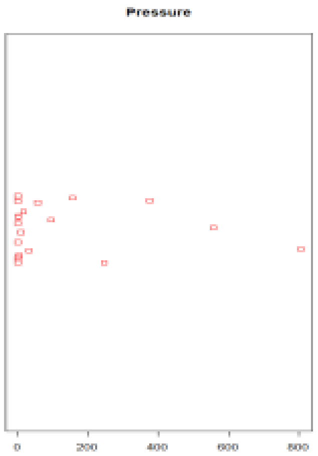


Pie chart representing various temperature values

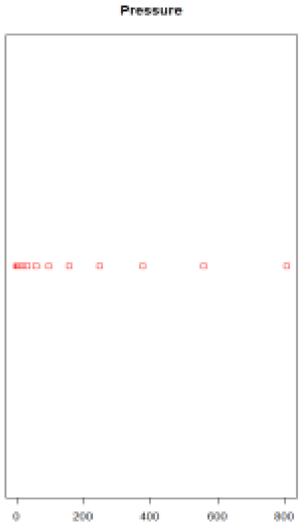
```
> dotchart(t(pressure$pressure),col="red",cex=0.8,main="Pressure")
```

This is a dot chart depicting various pressure values for different with its count value.

```
> stripchart(pressure$pressure,method="jitter",main="Pressure",col="red")
```

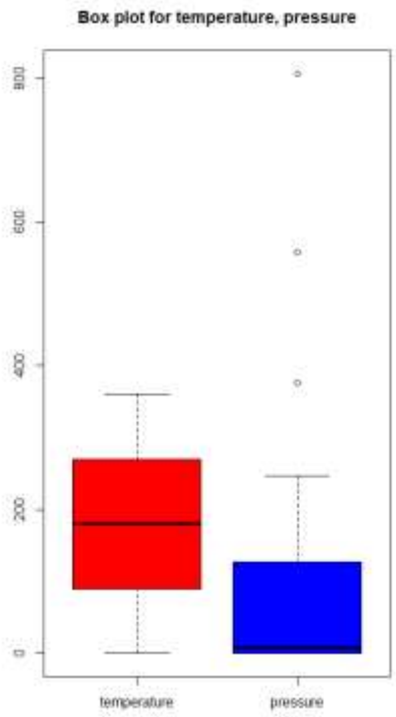



```
> stripchart(pressure$pressure,method="stack",main="Pressure",col="red")
```



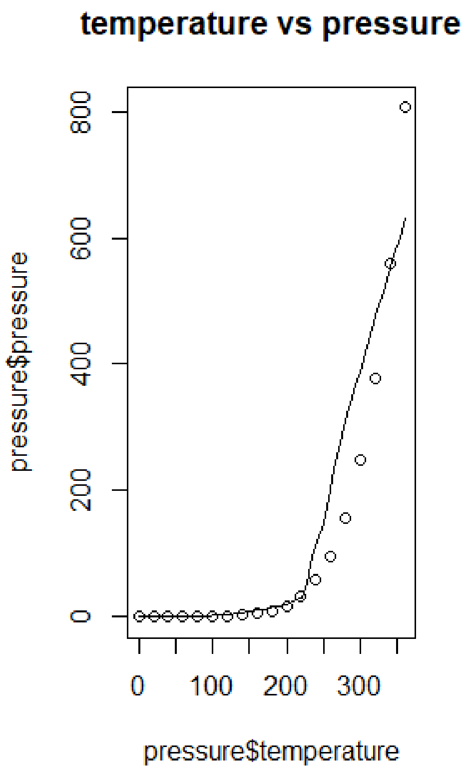
Both the plots are strip charts which provides pressure values for different ranges in two forms which includes jitter plot and stack plot.

```
> boxplot(pressure,main="Box plot for temperature, pressure",col=c("red","
blue"))
```

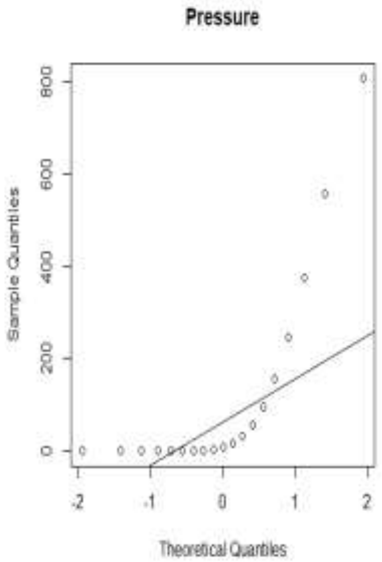


Various box plot diagram for data pressure which is represented for various parameter s such as temperature, pressure

```
> scatter.smooth(pressure$pressure~pressure$temperature,data=pressure,main
="temperature vs pressure")
```



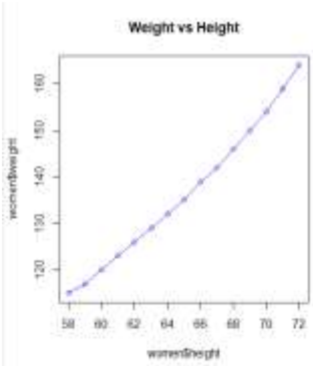

```
> qqnorm(pressure$pressure,main='Pressure')
```



Qnnorm values represents the which shows sample quartiles vs rheoritical quartile values.

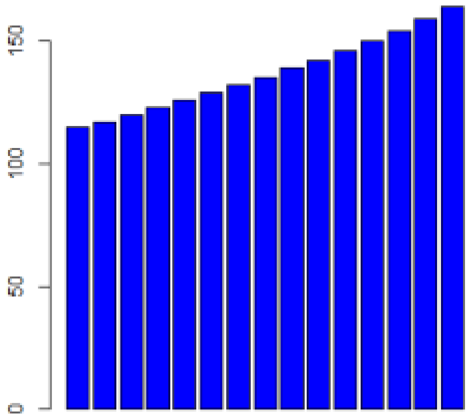
DATA SET 3 – WOMEN

```
> women
   height weight
1      58    115
2      59    117
3      60    120
4      61    123
5      62    126
6      63    129
7      64    132
8      65    135
9      66    139
10     67    142
11     68    146
12     69    150
13     70    154
14     71    159
15     72    164

> plot(women$weight~women$height,type="o",col="blue",main="Weight vs Heigh
t")
```



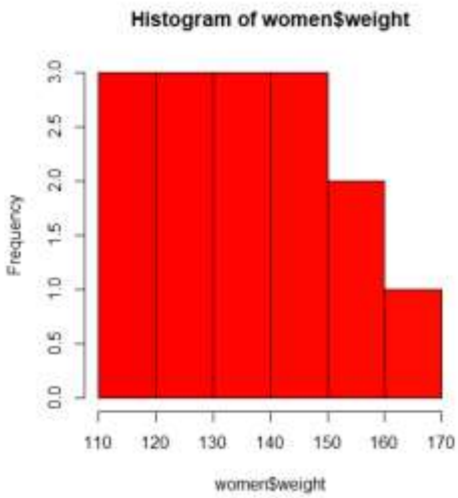
Linear plot depicting weight vs height in data women

```
> barplot(women$weight,col="blue")
```



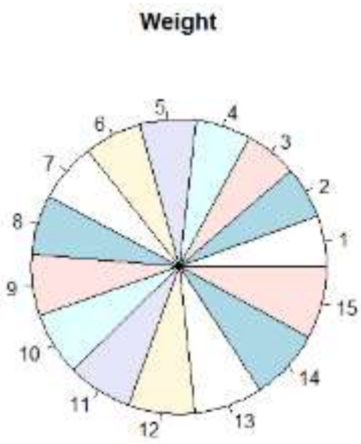
Bar plot representing the women's weight

```
hist(women$weight,col=heat.colors(max(women$weight)))
```



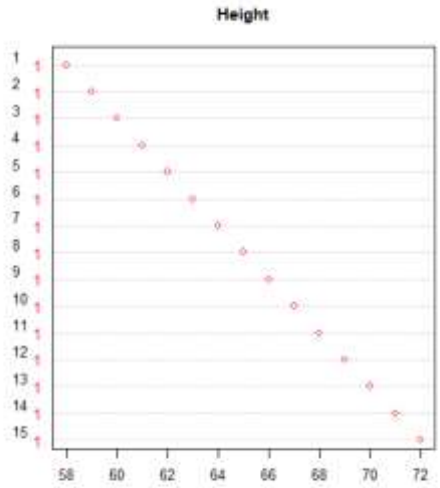
Histogram which represents the frequency count value for various pressure value in wo men's weight data

```
> pie(women$weight,main="Weight")
```



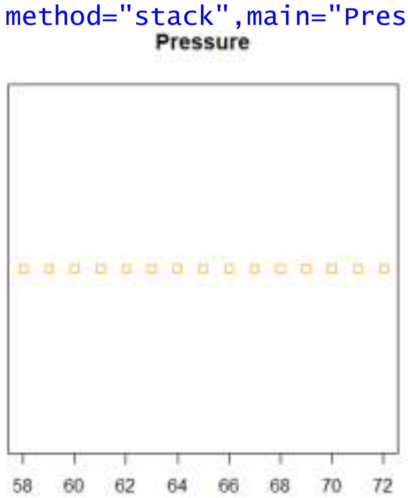
Pie chart representing various weight values of women

```
> dotchart(t(women$height),col="red",cex=0.8,main="Height")
```



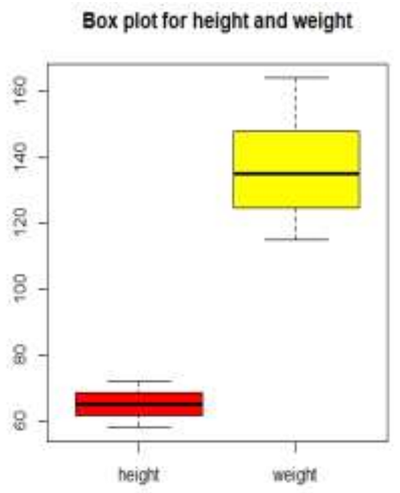
This is a dot chart depicting various women values for different with its count value.

```
> stripchart(women$height,method="stack",main="Pressure",col="orange")
```



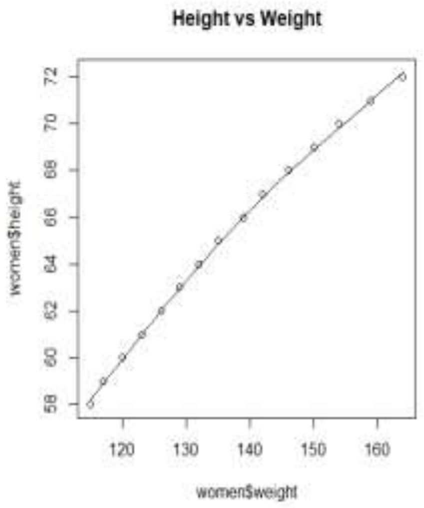

```
> boxplot(women,main="Box plot for height and weight",col=c("red","yellow"
))
```

Both the plots are strip charts which provides women values for different ranges in two forms which inclu des jitter plot and stack plot.

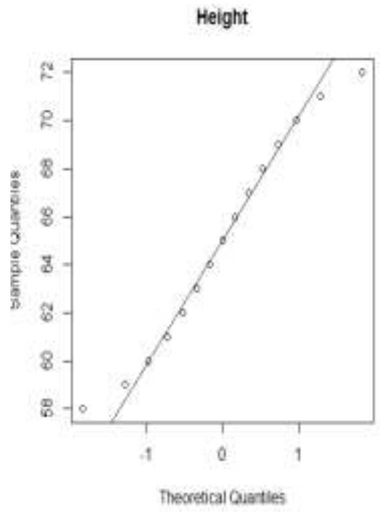


Various box plot diagram for data women which is represented for various parameters such as height and weight

```
> scatter.smooth(women$height~women$weight,data=women,main="Height vs Weig
ht")
```




```
> qqnorm(women$height,main='Height')
> qqline(women$height)
```



Qnnorm values represents the which shows sample quartiles vs rheoritical quartile values.

MACHINE LEARNING LAB ASSIGNMENT – 5

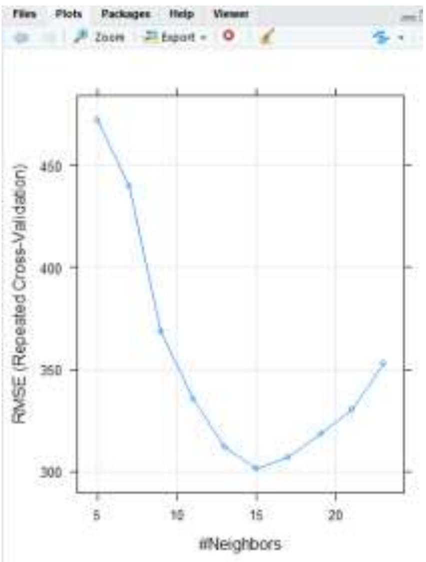
NAME – MARIA ALEX

REG – 16BCE2190

1. KNN TRAINING

```
> library(caret)

> library(e1071)

> set.seed(101)

> dt=sample(nrow(Orange),nrow(Orange)*0.8)

> train<-Orange[dt,]

> validation<-Orange[-dt,]

> dim(train)
[1] 28  3

> dim(validation)
[1] 7 3

> head(train)
Grouped Data: circumference ~ age | Tree
   Tree  age circumference
14    2 1582           203
2     1  484            58
24    4  664           112
22    4  118            32
8     2  118            33
10    2  664           111

> head(validation)
Grouped Data: circumference ~ age | Tree
   Tree  age circumference
6     1 1372           142
9     2  484            69
13    2 1372           203
18    3 1004           108
23    4  484            62
25    4 1004           167

> trctrl <- trainControl(method = "repeatedcv", number = 10
, repeats = 3)

> set.seed(3333)

> knn_fit <- train(age~., data=Orange, method = "knn",
+                   trControl=trctrl,
+                   preProcess = c("center", "scale"),
+    .... [TRUNCATED]
```

```
> knn_fit
k-Nearest Neighbors

35 samples
 2 predictor

Pre-processing: centered (5), scaled (5)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 32, 32, 31, 31, 31, 32, ...
Resampling results across tuning parameters:

  k   RMSE      Rsquared   MAE
   5  472.0962  0.5442437  421.4124
   7  439.6430  0.4971248  396.3222
   9  368.6787  0.6800141  327.0581
  11  335.3505  0.7575582  296.1266
  13  311.9519  0.7834273  271.9059
  15  301.3648  0.8006230  259.5211
  17  306.7305  0.7908583  262.8601
  19  318.3190  0.7854731  272.7433
  21  330.1665  0.8103771  286.7449
  23  352.6668  0.8232134  309.6047

RMSE was used to select the optimal model using
 the smallest value.
The final value used for the model was k = 15.
> plot(knn_fit)
```



3. K MEANS clustering

```
> data("USArrests")      # Loading the data set
> df <- scale(USArrests) # Scaling the data
>
> # View the firt 3 rows of the data
> head(df, n = 3)
            Murder  Assault   UrbanPop         Rape
Alabama 1.24256408 0.7828393 -0.5209066 -0.003416473
Alaska  0.50786248 1.1068225 -1.2117642  2.484202941
Arizona 0.07163341 1.4788032  0.9989801  1.042878388
>
```

```
> set.seed(123)
> km.res <- kmeans(df, 4, nstart = 25)
>
> print(km.res)
K-means clustering with 4 clusters of sizes 13, 16, 13, 8

Cluster means:
      Murder    Assault   UrbanPop         Rape
1 -0.9615407 -1.1066010 -0.9301069 -0.96676331
2 -0.4894375 -0.3826001  0.5758298 -0.26165379
3  0.6950701  1.0394414  0.7226370  1.27693964
4  1.4118898  0.8743346 -0.8145211  0.01927104

Clustering vector:
       Alabama         Alaska        Arizona
             4              3              3
      Arkansas     California       Colorado
             4              3              3
   Connecticut       Delaware        Florida
             2              2              3
       Georgia         Hawaii          Idaho
             4              2              1
      Illinois        Indiana           Iowa
             3              2              1
        Kansas       Kentucky      Louisiana
             2              1              4
         Maine       Maryland  Massachusetts
             1              3              2
      Michigan      Minnesota    Mississippi
             3              1              4
      Missouri        Montana       Nebraska
             3              1              1
        Nevada  New Hampshire     New Jersey
             3              1              2
    New Mexico       New York North Carolina
             3              3              4
  North Dakota           Ohio       Oklahoma
             1              2              2
        Oregon   Pennsylvania   Rhode Island
             2              2              2
South Carolina   South Dakota      Tennessee
             4              1              4
         Texas           Utah        Vermont
             3              2              1
      Virginia     Washington  West Virginia
             2              2              1
     Wisconsin        Wyoming
             1              2

Within cluster sum of squares by cluster:
[1] 11.952463 16.212213 19.922437  8.316061
 (between_SS / total_SS =  71.2 %)

Available components:

[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

4. ACCURACY, ROC

```
> data(PimaIndiansDiabetes)
> # prepare resampling method
> control <- trainControl(method="cv", number=5)
> set.seed(7)
> fit <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", metric=
"ROC", trControl=control)
> # display results
> print(fit)
Generalized Linear Model

768 samples
  8 predictor
  2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 615, 615, 614, 614, 614
Resampling results:

  Accuracy   Kappa
  0.7774043  0.4851161

> library(mlbench)
> # load the dataset
> data(PimaIndiansDiabetes)
> # prepare resampling method
> control <- trainControl(method="cv", number=5, classProbs=TRUE, summaryF
unction=twoClassSummary)
> set.seed(7)
> fit <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", metric=
"ROC", trControl=control)
> # display results
> print(fit)
Generalized Linear Model

768 samples
  8 predictor
  2 classes: 'neg', 'pos'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 615, 615, 614, 614, 614
Resampling results:

  ROC        Sens   Spec
  0.8344745  0.886  0.5747729
```

5. POLYNOMIAL REGRESSION

```
> linearmod<-lm(formula=Murder~Assault+UrbanPop+Rape,data=U
SArrests)
> linearmod

Call:
lm(formula = Murder ~ Assault + UrbanPop + Rape, data = USA
rrests)

Coefficients:
(Intercept)      Assault     UrbanPop         Rape
```

```
    3.27664      0.03978     -0.05469      0.06140

> summary(linearmod)

Call:
lm(formula = Murder ~ Assault + UrbanPop + Rape, data = USA
rrests)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.3990 -1.9127 -0.3444  1.2557  7.4279 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.276639   1.737997   1.885   0.0657 .  
Assault      0.039777   0.005912   6.729 2.33e-08 ***
UrbanPop    -0.054694   0.027880  -1.962   0.0559 .  
Rape         0.061399   0.055740   1.102   0.2764    
---
Signif. codes:  
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.574 on 46 degrees of freedom
Multiple R-squared:  0.6721,	Adjusted R-squared:  0.6507 
F-statistic: 31.42 on 3 and 46 DF,  p-value: 3.322e-11



>
> predmurder<-predict(formula=Murder~Assault+UrbanPop+Rape,linearmod)
> predmurder
       Alabama         Alaska        Arizona 
     10.793487      13.845014      12.499018 
      Arkansas     California       Colorado 
      9.296908      11.770833       9.501235 
   Connecticut       Delaware        Florida 
      4.122251       9.775774      14.185141 
       Georgia         Hawaii          Idaho 
      9.972108       1.807086       5.968315 
      Illinois        Indiana           Iowa 
     10.115168       5.505761       3.080437 
        Kansas       Kentucky      Louisiana 
      5.346423       5.769092      10.934441 
         Maine       Maryland  Massachusetts 
      4.267684      13.252220       5.555289 
      Michigan      Minnesota    Mississippi 
     11.527607       3.445667      12.222335 
      Missouri        Montana       Nebraska 
      8.259884       5.720538       4.955995 
        Nevada  New Hampshire     New Jersey 
     11.694674       3.064389       5.887785 
    New Mexico       New York North Carolina 
     12.755499      10.278912      15.208861 
  North Dakota           Ohio       Oklahoma 
      3.108308       5.261824       6.791813 
        Oregon   Pennsylvania   Rhode Island 
      7.735738       4.469929       5.949135 
South Carolina   South Dakota      Tennessee 
     13.130661       5.022175       9.179467 
         Texas           Utah        Vermont 
      8.462044       5.080455       4.123421 
      Virginia     Washington  Wes
```

```
> error<-USArrests$Murder-predmurder
> error
       Alabama         Alaska        Arizona 
     2.4065125     -3.8450138     -4.3990175 
      Arkansas     California       Colorado 
    -0.4969079     -2.7708331     -1.6012355 
   Connecticut       Delaware        Florida 
    -0.8222515     -3.8757741      1.2148592 
       Georgia         Hawaii          Idaho 
     7.4278916      3.4929141     -3.3683148 
      Illinois        Indiana           Iowa 
     0.2848317      1.6942392     -0.8804370 
        Kansas       Kentucky      Louisiana 
     0.6535767      3.9309079      4.4655590 
         Maine       Maryland  Massachusetts 
    -2.1676843     -1.9522196     -1.1552889 
      Michigan      Minnesota    Mississippi 
     0.5723925     -0.7456669      3.8776645 
      Missouri        Montana       Nebraska 
     0.7401157      0.2794616     -0.6559955 
        Nevada  New Hampshire     New Jersey 
     0.5053257     -0.9643889      1.5122154 
    New Mexico       New York North Carolina 
    -1.3554987      0.8210880     -2.2088613 
  North Dakota           Ohio       Oklahoma 
    -2.3083077      2.0381756     -0.1918128 
        Oregon   Pennsylvania   Rhode Island 
    -2.8357384      1.8300712     -2.5491354 
South Carolina   South Dakota      Tennessee 
     1.2693389     -1.2221747      4.0205334 
         Texas           Utah        Vermont 
     4.2379557     -1.8804554     -1.9234205 
      Virginia     Washington  West Virginia 
     1.1928536     -2.6603581      0.7634473 
     Wisconsin        Wyomin
> write.csv(m, "table.csv",
+           row.names = TRUE)
> RMSE<-sqrt(mean(error^2))
> RMSE
[1] 2.469139
```

	A	B	C	D	E
1		Actual	Predicted	Error	
2	Alabama	13.2	10.79349	2.406513	
3	Alaska	10	13.84501	-3.84501	
4	Arizona	8.1	12.49902	-4.39902	
5	Arkansas	8.8	9.296908	-0.49691	
6	California	9	11.77083	-2.77083	
7	Colorado	7.9	9.501235	-1.60124	
8	Connecticu	3.3	4.122251	-0.82225	
9	Delaware	5.9	9.775774	-3.87577	
10	Florida	15.4	14.18514	1.214859	
11	Georgia	17.4	9.972108	7.427892	
12	Hawaii	5.3	1.807086	3.492914	
13	Idaho	2.6	5.968315	-3.36831	
14	Illinois	10.4	10.11517	0.284832	
15	Indiana	7.2	5.505761	1.694239	
16	Iowa	2.2	3.080437	-0.88044	
17	Kansas	6	5.346423	0.653577	
18	Kentucky	9.7	5.769092	3.930908	
19	Louisiana	15.4	10.93444	4.465559	
20	Maine	2.1	4.267684	-2.16768	
21	Maryland	11.3	13.25222	-1.95222	
22	Massachus	4.4	5.555289	-1.15529	
23	Michigan	12.1	11.52761	0.572393	
24	Minnesota	2.7	3.445667	-0.74567	
25	Mississippi	16.1	12.22234	3.877665	
26	Missouri	9	8.259884	0.740116	
27	Montana	6	5.720538	0.279462	
28	Nebraska	4.3	4.955995	-0.656	

24	Minnesota	2.7	3.445667	-0.74567
25	Mississippi	16.1	12.22234	3.877665
26	Missouri	9	8.259884	0.740116
27	Montana	6	5.720538	0.279462
28	Nebraska	4.3	4.955995	-0.656
29	Nevada	12.2	11.69467	0.505326
30	New Hamp	2.1	3.064389	-0.96439
31	New Jerse	7.4	5.887785	1.512215
32	New Mexi	11.4	12.7555	-1.3555
33	New York	11.1	10.27891	0.821088
34	North Carc	13	15.20886	-2.20886
35	North Dak	0.8	3.108308	-2.30831
36	Ohio	7.3	5.261824	2.038176
37	Oklahoma	6.6	6.791813	-0.19181
38	Oregon	4.9	7.735738	-2.83574
39	Pennsylvar	6.3	4.469929	1.830071
40	Rhode Isla	3.4	5.949135	-2.54914
41	South Carc	14.4	13.13066	1.269339
42	South Dak	3.8	5.022175	-1.22217
43	Tennessee	13.2	9.179467	4.020533
44	Texas	12.7	8.462044	4.237956
45	Utah	3.2	5.080455	-1.88046
46	Vermont	2.2	4.123421	-1.92342
47	Virginia	8.5	7.307146	1.192854
48	Washingto	4	6.660358	-2.66036
49	West Virgi	5.7	4.936553	0.763447
50	Wisconsin	2.6	2.438163	0.161837
51	Wyoming	6.8	7.356976	-0.55698

6. PCA – mtcars

```
> mtcars.pca <- prcomp(mtcars[,c(1:7,10,11)], center = TRUE,scale. = TRUE)
>
> summary(mtcars.pca)
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     2.3782 1.4429 0.71008 0.51481
Proportion of Variance 0.6284 0.2313 0.05602 0.02945
Cumulative Proportion  0.6284 0.8598 0.91581 0.94525
                           PC5     PC6     PC7    PC8
Standard deviation     0.42797 0.35184 0.32413 0.2419
Proportion of Variance 0.02035 0.01375 0.01167 0.0065
Cumulative Proportion  0.96560 0.97936 0.99103 0.9975
```

```
                           PC9
Standard deviation     0.14896
Proportion of Variance 0.00247
Cumulative Proportion  1.00000
```

7. > dist_mat <- dist(Orange, method = 'euclidean')

```
> hclust_avg <- hclust(dist_mat, method = 'average')
> plot(hclust_avg)
> dist_mat
             1           2           3           4
2   367.069476
3   548.967212  182.321145
4   890.067975  523.114710  341.150993
5  1116.632885  749.568543  567.959506  227.055059
6  1258.991660  891.964125  710.133086  368.989160
7  1468.509789 1101.441328  919.830419  578.778023
8     3.162278  366.854194  548.664743  889.787053
9   368.073362   11.045361  180.900525  522.031608
10  551.976449  187.643279   24.020824  340.024999
11  894.915080  529.154987  346.932270   41.012193
12 1122.022281  755.649390  573.336725  234.049140
13 1265.877561  899.761079  717.440590  378.376796
14 1474.186555 1107.533295  925.300492  584.661440
15    2.000000  367.074924  548.970855  890.070222
16  366.607419    7.280110  183.575598  523.927476
17  547.854908  180.812057   12.165525  342.350697
18  889.429030  522.402144  340.653783    7.280110
19 1116.242805  749.174212  567.694460  227.008810
20 1258.729915  891.688847  709.909853  368.787202
21 1468.128060 1101.059490  919.530859  578.543862
22    3.605551  366.934599  548.771355  889.884262
23  367.408492    5.000000  181.752579  522.702592
24  552.131325  187.949461   25.179357  340.026470
25  896.534439  531.309703  349.297867   52.086467
26 1122.933213  756.742360  574.423189  235.868607
27 1266.714648  900.751908  718.440673  379.827592
28 1475.520586 1109.030658  926.748078  586.424761
29    4.000000  367.091269  548.981785  890.076963
30  366.514665    9.848858  184.010869  524.186990
             5           6           7           8
2
3
4
5
6   142.705991
7   351.889187  210.021427
8  1116.395539 1258.728724 1468.278243
9   748.739608  890.996072 1100.627548  367.766230
10  567.072306  708.679053  918.629958  551.543289
11  229.839074  368.267566  578.105527  894.497065
12   52.009614  144.159634  352.038350 1121.646112
13  163.618459   61.008196  217.864637 1265.470663
14  360.681300  218.682418   58.008620 1473.837169
15 1116.634676 1258.993249 1468.511151    3.162278
16  750.182644  892.652788 1102.018149  366.443720
17  568.786427  711.165944  920.667149  547.613915
18  227.325757  369.572726  579.186498  889.169275
19    5.385165  143.575764  352.285396 1116.017025
20  142.288439    3.605551  210.095217 1258.472487
21  351.575028  210.019047    5.385165 1467.905310
22 1116.477496 1258.818891 1468.357586    2.236068
```

23	749.254296	891.601368	1101.136685	367.152557
24	567.064370	708.641658	918.597845	551.689224
25	231.833992	368.860407	578.426313	896.078122
26	59.076222	145.804664	352.655639	1122.536859
27	166.766304	67.067131	219.556371	1266.292225
28	363.381342	222.020269	69.065187	1475.147789
29	1116.640049	1258.998014	1468.515237	4.242641
30	750.377238	892.865611	1102.195990	366.361843

	9	10	11	12
2				
3				
4				
5				
6				
7				
8				
9				
10	184.835062			
11	527.227655	342.965013		
12	754.067636	570.271865	227.563178	
13	898.053451	713.952379	370.989218	144.367586
14	1106.146464	922.598504	579.907751	352.366287
15	368.073362	551.976449	894.915080	1122.022281
16	18.027756	189.739295	530.495994	756.737075
17	180.102748	36.013886	349.516809	575.238212
18	521.461408	340.014706	48.010416	235.851648
19	748.415660	567.014991	230.675096	57.008771
20	890.755297	708.554162	368.393811	144.813673
21	1100.293597	918.458491	578.222276	352.457090
22	367.870901	551.689224	894.637357	1121.772259
23	7.280110	186.560982	528.431642	755.058276
24	185.075660	2.236068	342.841071	570.169273
25	529.157821	344.586709	11.180340	227.063868
26	755.058276	571.066546	228.170989	7.280110
27	898.970522	714.753104	371.802367	145.787517
28	1107.534650	923.762415	580.906189	353.509547
29	368.084229	551.983695	894.919549	1122.025846
30	20.223748	190.402206	530.903004	757.064726

	13	14	15	16
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14	210.000000			
15	1265.877561	1474.186555		
16	900.915645	1108.471470	366.601964	
17	719.478283	926.881330	547.851257	181.592951
18	380.065784	585.755922	889.426782	523.114710
19	166.210710	361.864616	1116.241013	749.736620
20	64.007812	219.538152	1258.728327	892.349707
21	219.248717	63.007936	1468.126698	1101.601107
22	1265.606969	1473.954206	2.236068	366.494202
23	899.126799	1107.018067	367.397605	11.045361
24	713.827010	922.501491	552.124080	190.057886

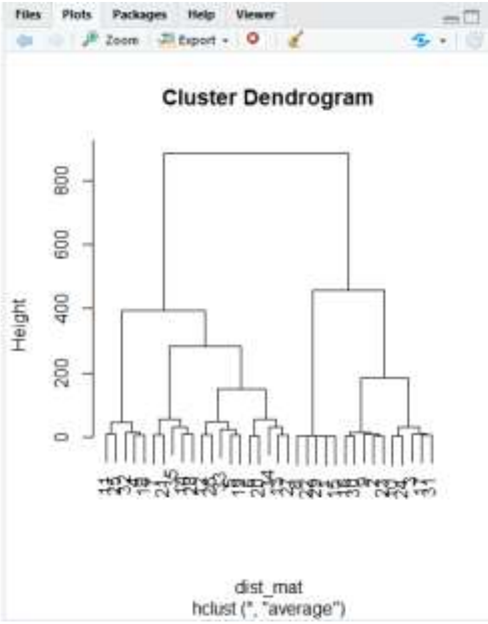
```
25  369.762086  579.123476  896.529977  532.782320
26  143.041952  351.825241 1122.929651  757.887855
27    6.324555  210.095217 1266.711490  901.947338
28  210.297408   11.180340 1475.517875 1110.033333
29 1265.880721 1474.189269    2.000000  366.607419
30  901.259674 1108.751099  366.498295    2.828427
            17          18          19          20
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18  341.597717
19  568.409184  227.107904
20  710.886770  369.303398  143.027969
21  920.298321  578.885135  351.889187  210.002381
22  547.691519  889.254182 1116.090946 1258.557110
23  180.471604  522.031608  748.878495  891.332710
24   37.013511  340.024999  567.008818  708.515349
25  352.228619   59.008474  232.881944  369.065035
26  576.459886  237.846589   64.007812  146.567391
27  720.569913  381.609748  169.463860   70.007142
28  928.464323  587.640196  364.695764  222.993273
29  547.854908  889.429030 1116.242805 1258.729915
30  181.879081  523.340234  749.912662  892.551399
            21          22          23          24
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 1467.978542
23 1100.767460  367.227450
24  918.427460  551.829684  186.815417
25  578.631143  896.225976  530.495052  344.419802
26  353.161436 1122.665578  756.107135  570.944831
```

27	221.047506	1266.430022	900.084996	714.613882
28	74.006756	1475.269467	1108.471019	923.649284
29	1468.128060	2.236068	367.397605	552.124080
30	1101.766309	366.395961	13.038405	190.709203

	25	26	27	28
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26	227.316959			
27	370.388985	144.156165		
28	579.907751	352.740698	210.059515	
29	896.529977	1122.929651	1266.711490	1475.517875
30	533.221342	758.228198	902.299839	1110.328780

	29	30	31	32
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				

```
29
30  366.492838
            33          34
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
 [ reached getOption("max.print") -- omitted 5 rows ]
> hclust_avg

Call:
hclust(d = dist_mat, method = "average")

Cluster method   : average
Distance         : euclidean
Number of objects: 35
```



8. **Medical Diagnosis using Machine Learning**

Machine learning techniques that uses pattern recognition can be efficiently used to assist diagnosis. Health risk predictions can be performed but understanding and evaluating existing data available from similar cases. Using machine learning models predict risk of various diseases with around 90% accuracy. And another important analysis that can be done based on medical diagnosis is estimating the admission patient rates on a daily basis using the internal and external data available of the medical institution and based on this forecast rate human resources can be allocated and improve the service for the patient and increase patient outcomes.

Using regression model can estimate t heh the time taken to discharge the patient from hospital suffering from a chronic condition, cost to treat the patient, time for insurance provider to provide compensation and how far the demographic has led to the disease.