To detect whether a person is running or walking using machine learning in python

By Shrinivas Kallol

Objective:

my aim is to detect Whether a person is running or walking based on deep neural network. The sensor data was collected from IOS device. I took this data from Kaggle competition the data is broadly described below.

Data description:

Currently, the dataset contains a single file which represents 88588 sensor data samples collected from accelerometer and gyroscope from iPhone 5c in 10 seconds interval and ~5.4/second frequency. This data is represented by following columns (each column contains sensor data for one of the sensor's axes):

- acceleration_x
- acceleration_y
- acceleration_z
- gyro_x
- gyro_y
- gyro_z

There is an activity type represented by "activity" column which acts as label and reflects following activities:

- "0": walking
- "1": running

Apart of that, the dataset contains "wrist" column which represents the wrist where the device was placed to collect a sample on:

- "0": left wrist
- "1": right wrist

Additionally, the dataset contains "date", "time" and "username" columns.

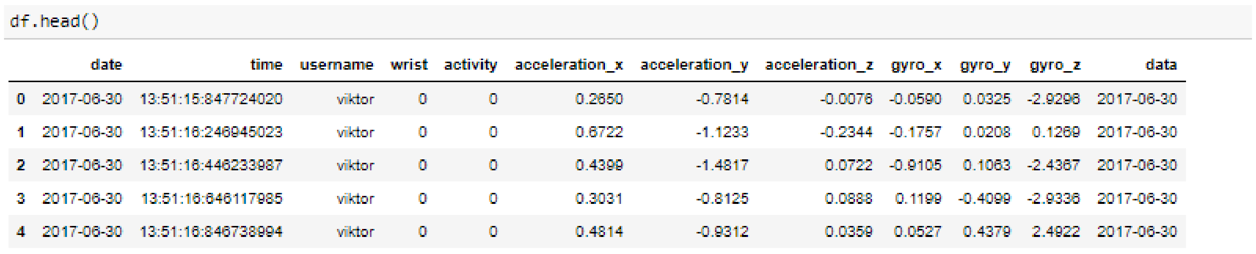
Introduction

Smartphones which are almost available to everybody are now offering many other features such as multitasking and deployment of variety of sensors due to which it has become easy to keep a track of our activities, learn them and help in making better decision regarding future actions.

In this project the Data produced by accelerometer and gyroscope will be used to detect human activity(walking/running) for this project I have used two approaches one is machine learning approach and another is deep learning approach(LSTM).

Loading the data

- Dataset was available in csv and was loaded with the help of pandas the following fig shows the data set



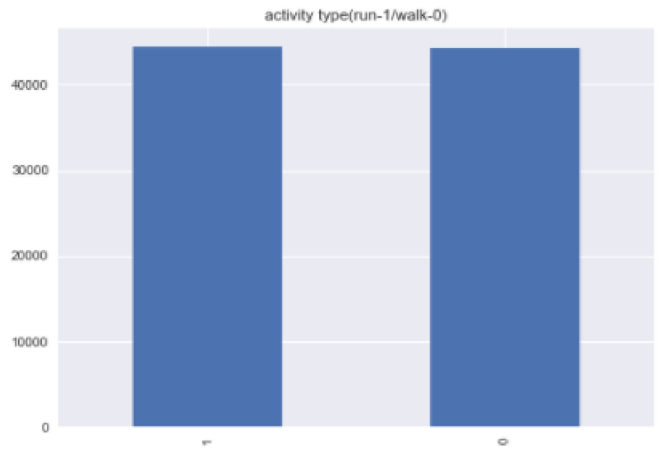
```
df.head()
```

	date	time	username	wrist	activity	acceleration_x	acceleration_y	acceleration_z	gyro_x	gyro_y	gyro_z	data
0	2017-06-30	13:51:15:847724020	viktor	0	0	0.2650	-0.7814	-0.0076	-0.0590	0.0325	-2.9296	2017-06-30
1	2017-06-30	13:51:16:246945023	viktor	0	0	0.6722	-1.1233	-0.2344	-0.1757	0.0208	0.1269	2017-06-30
2	2017-06-30	13:51:16:446233987	viktor	0	0	0.4399	-1.4817	0.0722	-0.9105	0.1063	-2.4367	2017-06-30
3	2017-06-30	13:51:16:646117985	viktor	0	0	0.3031	-0.8125	0.0888	0.1199	-0.4099	-2.9336	2017-06-30
4	2017-06-30	13:51:16:846738994	viktor	0	0	0.4814	-0.9312	0.0359	0.0527	0.4379	2.4922	2017-06-30

Preprocessing

- Since the range of values of raw data does not varies widely scaling is not required
- There were no NAN values present in the data set
- 0 is walk and 1 is run below we can see that the count of both run and walk is nearly equal so there is no need of sampling

```
df['activity'].value_counts().plot(kind='bar', title='activity type(run-1/walk-0)');
```

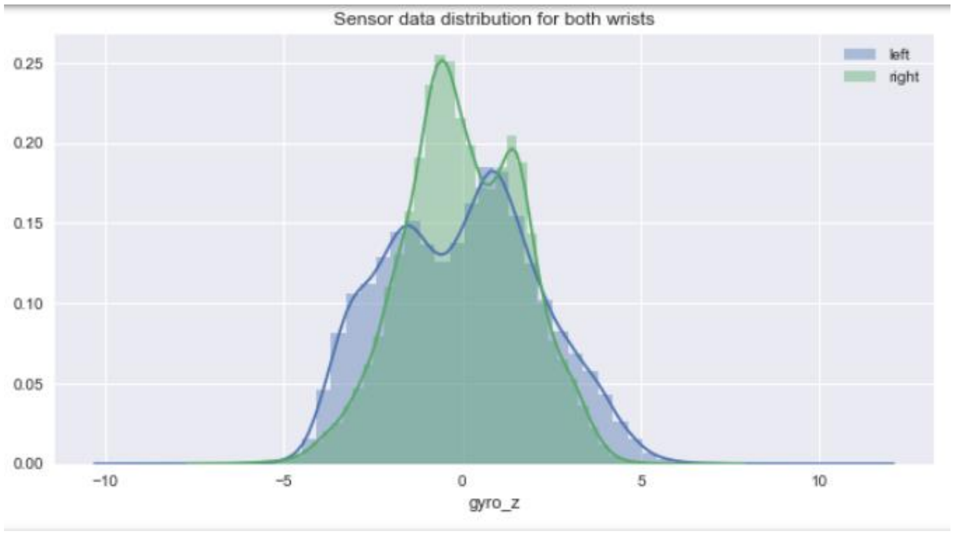


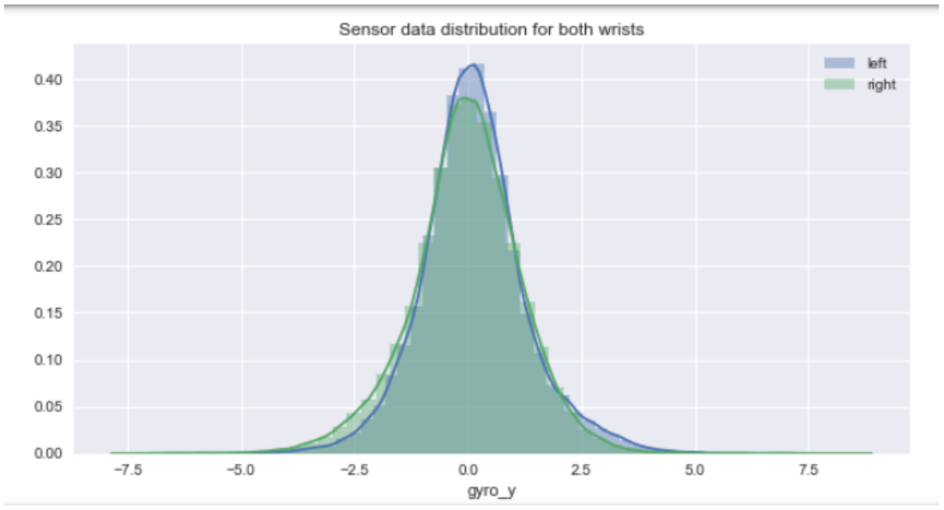
- the walk has 44223 values in that 18622 are left wrist and 25601 are right wrist
- the run has 44365 values in that 23708 left wrist and 20657 right wrist
- The only issue is that the number of walk samples recorded on the right wrist is significantly more than the samples recorded on the left wrist.

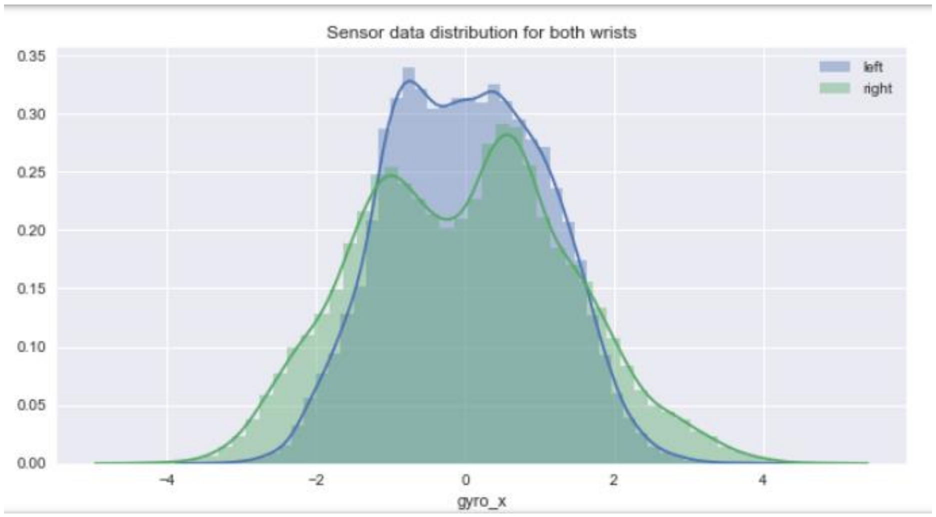
- However, the most important here is that the total number of walk and run samples is almost the same.

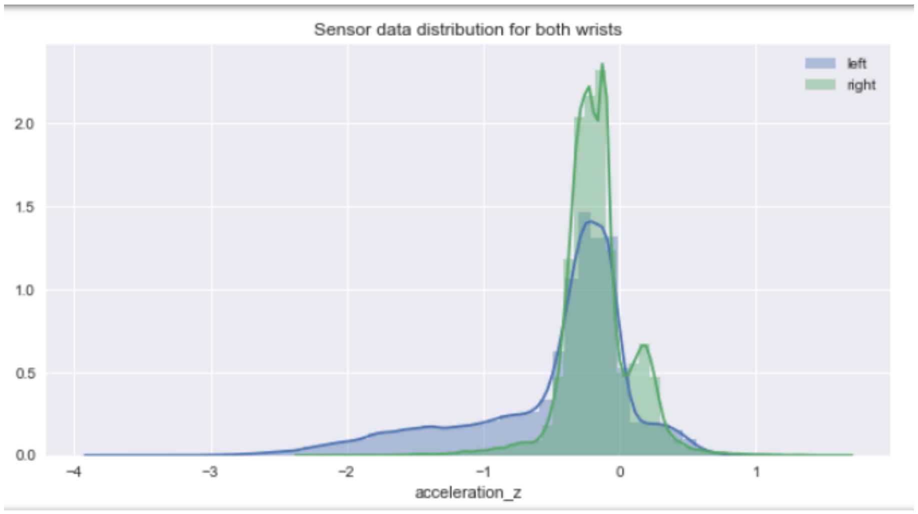
Exploratory analysis

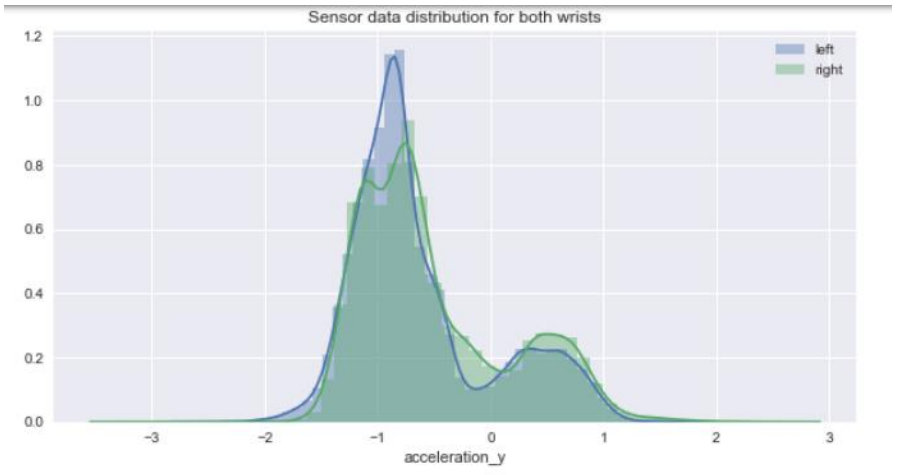
- Since we have samples from both left and wright wrists, it makes sense to explore distribution of numerical data separately for both wrists.

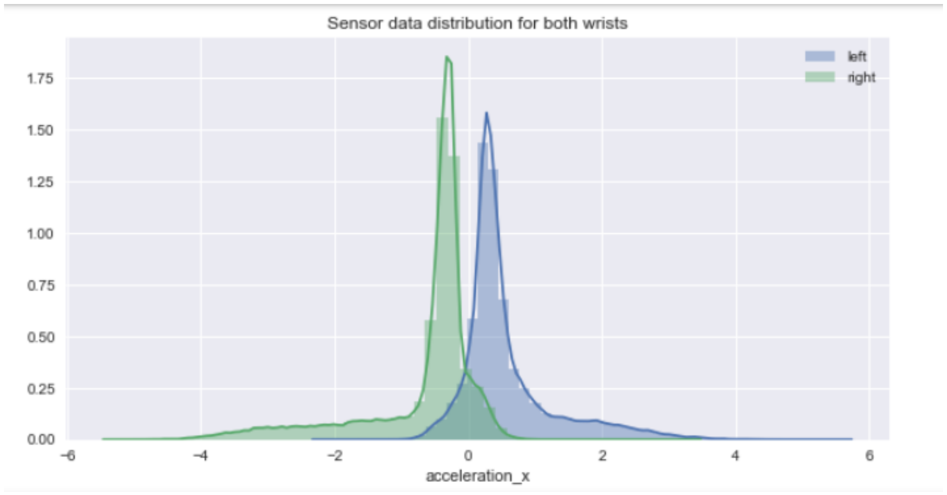





Sensor data distribution for both wrists
left
right
gyro_x
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4
−2
0
2
4


Sensor data distribution for both wrists
left
right
acceleration_z
2.0
1.5
1.0
0.5
0.0
−4
−3
−2
−1
0
1





The data set analyzed is properly formatted and has no NAN values and has clear values

Regarding the sensor data presented in the dataset, we can conclude that the data presented doesn't suffer from skewing and normally distributed.

Machine learning approach

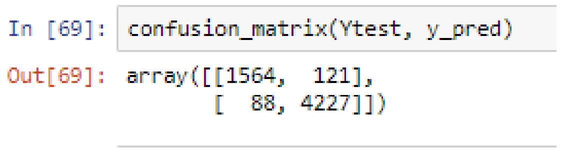
- Initially I used PCA for future reduction of our input variables since after retaining 99% of variance it gave 6 components therefore PCA is not useful in this case
- Data was split into train and validation set using sklearn.cross_validation.train_test_split(test_size=0.3)
- Since it is binary classification I used different machine learning algorithms like **SVM,KNN,Logistic regression, Naïve bayes, Random forest** and due to page limit constraint I have documented only Support vector classifier

Parameter tuning of SVC

- I used gridsearchcv method for this approach Using GridSearchCV is easy. You just need to import GridSearchCV from sklearn.grid_search (1)
  By using this the Best C: 10. Best Kernel: rbf. Best Gamma: 0.01

Training and evaluation of SVC

- Using these parameters I trained the model by importing svc from sklearn since the data was huge it was sliced to 20000 observation
- Validation accuracy was measured using metrics.accuracy_score the the validation accuracy was 0.9651666666666666
- The confusion matrix is



```
In [69]: confusion_matrix(Ytest, y_pred)

Out[69]: array([[1564,  121],
                [  88, 4227]])
```

-
- Precision is 0.98
- Recall is 0.95

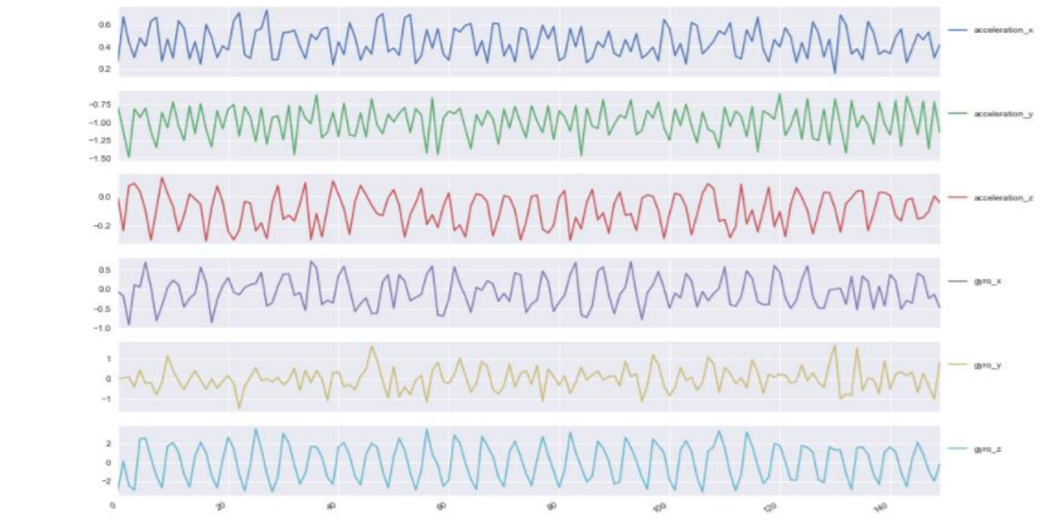
Deep learning approach

Exploration

A plot was plotted which displays observations on the y-axis against equally spaced time intervals on the x-axis

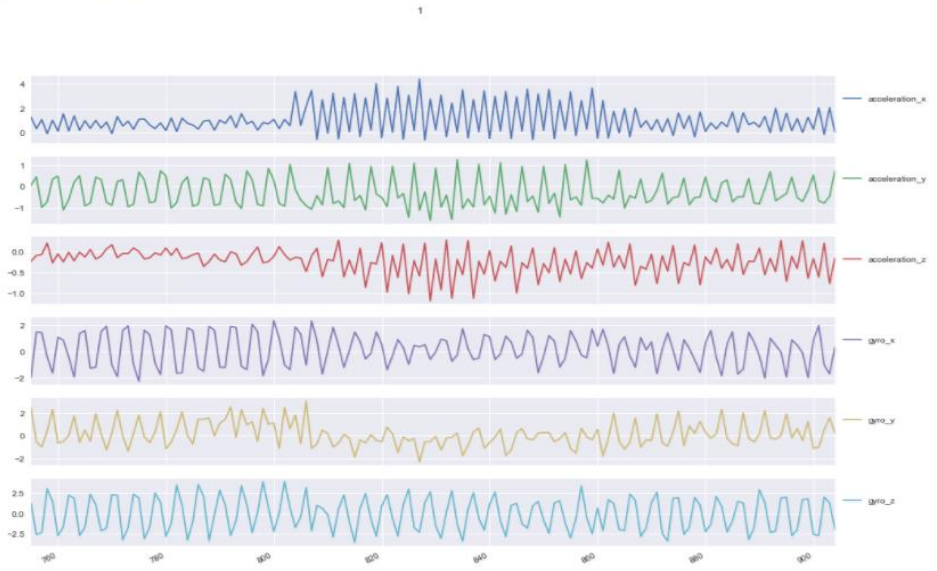
Walking

```
In [18]: plot_activity(0, df)
```



Running

```
In [23]: plot_activity(1, df)
```



By the above figures we can easily distinguish between the activities(walking/running) so this data can be used to train LSTM model

Data preprocessing

- LSTM model expects fixed length sequences of data so each generated sequence contains 200 training examples

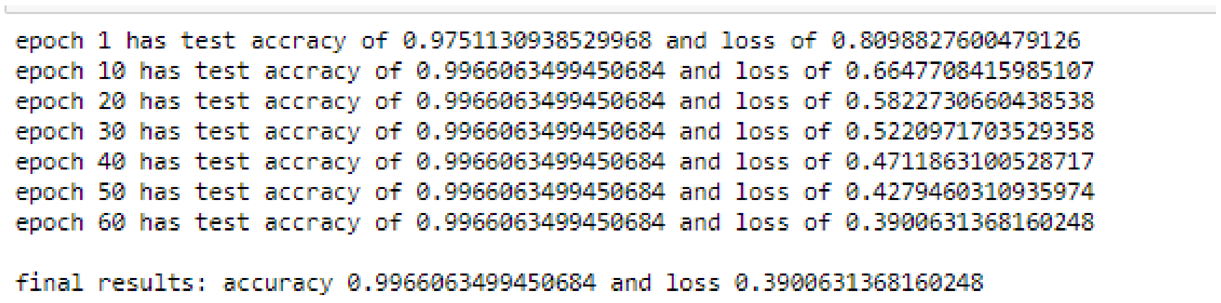
- The new shape of after transformation using stats.mode is (4420, 6, 200) which is far more reduced from original shape (88588, 6)
- This transformed shape is again reshaped into sequence of 200 rows the new shape is (4420, 200, 6)
- Applied 1 hot encoding for labels using pd.get_dummies function and the data was split into training and testing using test_train_split (20%test)

<u>Modelling</u>

- This model contains 2 fully-connected and 2 LSTM layers (stacked on each other) with 64 units each. This was modelled in python using tensorflow (3)
- Placeholder were created A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. In TensorFlow terminology, we then feed data into the graph through these placeholders.(2)
- Tenor was named by which I will obtain predictions and l2 regularization is used which can be noted in loss
- LEARNING_RATE = 0.0025 and adamoptimiser and relu was used
- trained the model for 60 epochs and keep track of accuracy and error

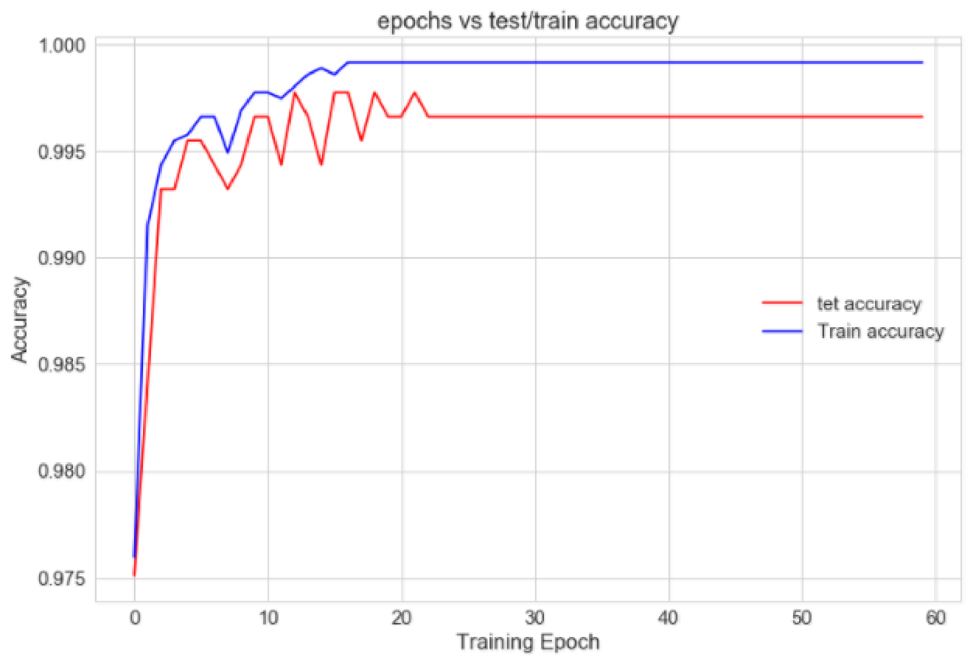
Evaluation

The final test accuracy is 0.996 and loss is 0.39



```
epoch 1 has test accracy of 0.9751130938529968 and loss of 0.8098827600479126
epoch 10 has test accracy of 0.9966063499450684 and loss of 0.6647708415985107
epoch 20 has test accracy of 0.9966063499450684 and loss of 0.5822730660438538
epoch 30 has test accracy of 0.9966063499450684 and loss of 0.5220971703529358
epoch 40 has test accracy of 0.9966063499450684 and loss of 0.4711863100528717
epoch 50 has test accracy of 0.9966063499450684 and loss of 0.4279460310935974
epoch 60 has test accracy of 0.9966063499450684 and loss of 0.3900631368160248

final results: accuracy 0.9966063499450684 and loss 0.3900631368160248
```

```
Out[60]: <matplotlib.text.Text at 0x1ea3fbb8cc0>
```



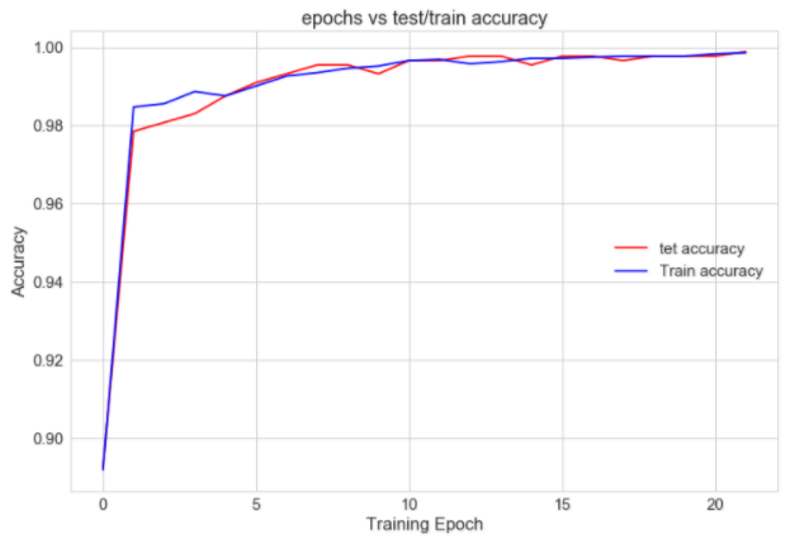
- As seen I the above graph as the epoch increases the accuracy increases and decreases but the test epoch at 22 gives max test accuracy and then model overfits and gives constant accuracy with increasing epoch

Self evaluation

Using epohs ==22 we get more accuracy(test accuracy==99.88%)

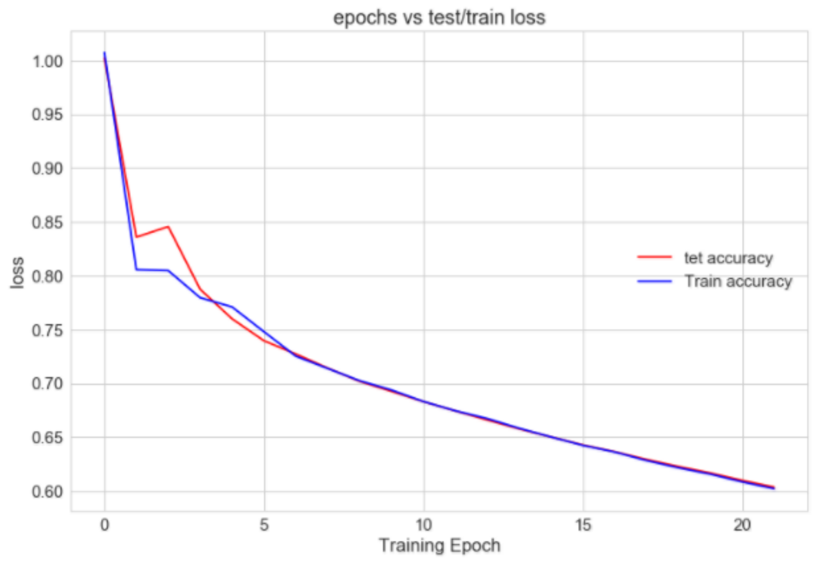
```
epoch 1 has test accracy 0.8925339579582214 and loss test1.0019768476486206
epoch 5 has test accracy 0.9875565767288208 and loss test0.7600255608558655
epoch 10 has test accracy 0.9932126402854919 and loss test0.6926394701004028
epoch 15 has test accracy 0.9954751133918762 and loss test0.6502702236175537
epoch 20 has test accracy 0.9977375268936157 and loss test0.6167940497398376

final results: accuracy 0.9988687634468079 and loss 0.6036503314971924
```

```
Out[63]: <matplotlib.text.Text at 0x1ea3894fcc0>
```



The test accuracy is max at 22nd epoch

```
Out[64]: <matplotlib.text.Text at 0x1ea37448e10>
```



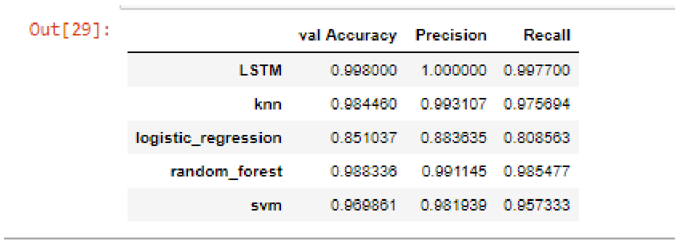
- The loss gradually decreases as number of epoch increases
- Precision, recall and Confusion matrix

```
          precision =  1.0
           recall =  0.9977827051

Out[73]: array([[433,   0],
                [  1, 450]])
```

- Which gives one only one misclassification and max precision-1.0

Conclusion

- The LSTM model(Test accuracy==99.988%) will be my final model with epoch=22. because the test accuracy is more compared to all other models



Out[29]:

	val Accuracy	Precision	Recall
LSTM	0.998000	1.000000	0.997700
knn	0.984460	0.993107	0.975694
logistic_regression	0.851037	0.883635	0.808563
random_forest	0.988336	0.991145	0.985477
svm	0.969861	0.981939	0.957333

References

1. https://medium.com/@aneesha/svm-parameter-tuning-in-scikit-learn-using-gridsearchcv-2413c02125a0
2. https://learningtensorflow.com/lesson4/
3. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
4. https://pdfs.semanticscholar.org/f3c8/e586cee14ff1ee33b33d8ca7604d2f5ca31a.pdf