1) Pre-Processing: Extract the Data Set and Concatenation

1.a) Libraries

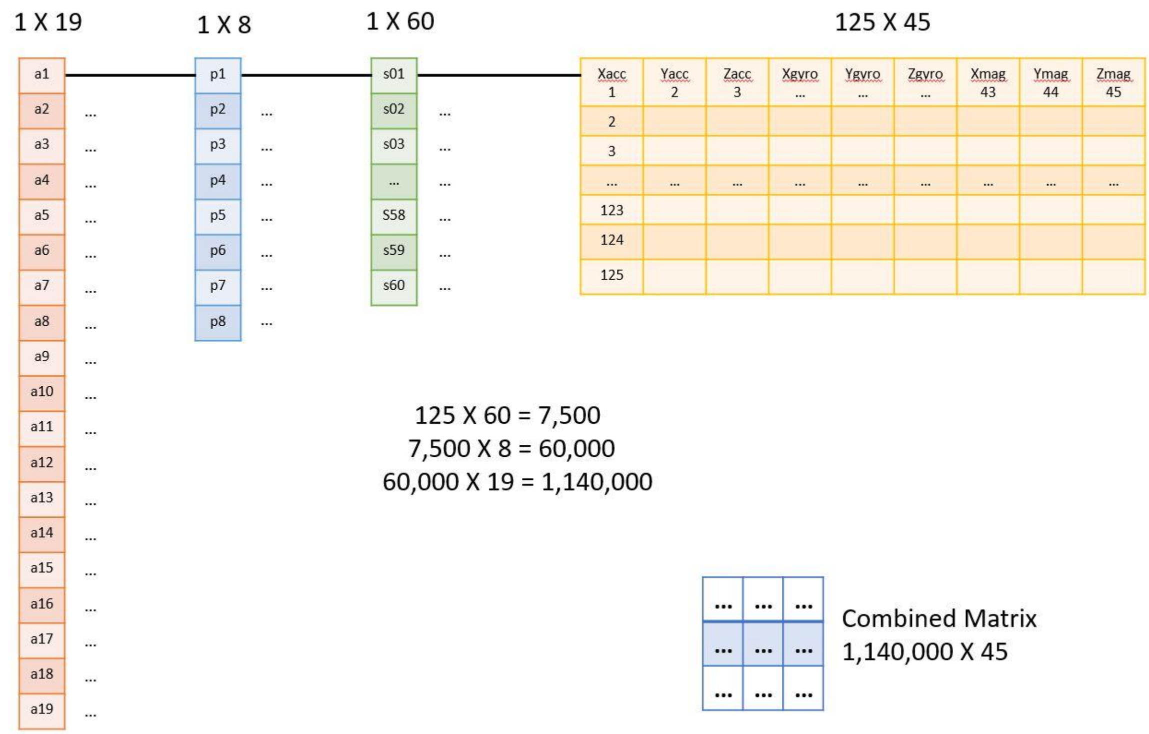
In [1]:
```
import pandas as pd
import numpy as np
import glob
import os
import time
from sklearn import linear_model

# plot feature and overall percent variance
%matplotlib inline
import matplotlib.pyplot as plt
from IPython import display
```

1.b) Data Set Original Structure

```
19 Activities (a)
8 subjects (p)
60 segments (s)

5 units: torso (T), right arm (RA), left arm (LA), right leg (RL), left leg
(LL)

9 sensors on each unit (x,y,z accelerometers, x,y,z gyroscopes, x,y,z
magnetometers)
Total of 125 x 45 readiing per segment (s)

The Data can be download directly from:
https://archive.ics.uci.edu/ml/datasets/daily+and+sports+activities
```

In [4]:
```python
from IPython import display
display.Image("./RawDataSet.png")
```

Out[4]:



- The 19 activities are:

```
 A1 = Sitting.
 A2 = Standing.
 A3 = Lying on back.
 A4 = Lying on right side.
 A5 = Ascending stairs.
 A6 = Descending stairs.
 A7 = Standing in an elevator still.
 A8 = Moving around in an elevator.
 A9 = Walking in a parking lot.
A10 = Walking on a treadmill with a speed of 4 km/h in flat.
```

```
A11 = Walking on a treadmill with a speed of 4 km/h in a 15 deg inclined
position.
A12 = Running on a treadmill with a speed of 8 km/h.
A13 = Exercising on a stepper.
A14 = Exercising on a cross trainer.
A15 = Cycling on an exercise bike in horizontal position.
A16 = Cycling on an exercise bike in vertical position.
A17 = Rowing.
A18 = Jumping.
A19 = Playing basketball.
```

- Features Identification:

- T = torso
- RA = Right arm
- LA = Left arm
- RL = Right leg
- LL = Left leg
- x, y, z = Axes
- acc = Accelerometer
- gyro = Gyroscope
- mag = Magnetometer

1.c) Loading data set from a01 to a19 for all subjects (p1 to p8) and all segments (s01 to s60) into a single Matrix

In [5]:
```
complete_data = pd.DataFrame()
start = time.time()
for i in range(1,20):

    if i < 10:
        activity_folder = os.listdir('./data/a0'+ str(i))
        a='a0'+str(i)
    else:
        activity_folder = os.listdir('./data/a'+ str(i))
        a='a'+str(i)

    for j in range(1, 9):
        person_folder = os.listdir('./data/'+ a +'/p'+str(j))
        p='p'+str(j)
        for file in person_folder:
            filepath = './data/'+a+'/'+p+'/'+ file

            data = pd.read_csv(filepath, header=None)

            #Adding Column 46 to Include the Activity Number corresponding to
            data[45]=i
            complete_data=complete_data.append(data)
    print('Matrix shape after combining: '+a ,complete_data.shape)
end = time.time()
duration_with_svd = end-start
print("Time taken to compile data set into a single matrix: %d seconds" %dura
```

```
Matrix shape after combining: a01 (60000, 46)
Matrix shape after combining: a02 (120000, 46)
Matrix shape after combining: a03 (180000, 46)
Matrix shape after combining: a04 (240000, 46)
Matrix shape after combining: a05 (300000, 46)
Matrix shape after combining: a06 (360000, 46)
Matrix shape after combining: a07 (420000, 46)
Matrix shape after combining: a08 (480000, 46)
Matrix shape after combining: a09 (540000, 46)
Matrix shape after combining: a10 (600000, 46)
Matrix shape after combining: a11 (660000, 46)
Matrix shape after combining: a12 (720000, 46)
Matrix shape after combining: a13 (780000, 46)
Matrix shape after combining: a14 (840000, 46)
Matrix shape after combining: a15 (900000, 46)
Matrix shape after combining: a16 (960000, 46)
Matrix shape after combining: a17 (1020000, 46)
Matrix shape after combining: a18 (1080000, 46)
Matrix shape after combining: a19 (1140000, 46)
Time taken to compile data set into a single matrix: 630 seconds
```

In [6]:
```
#Labels for Columns:
complete_data.columns=['T_xacc', 'T_yacc', 'T_zacc', 'T_xgyro','T_ygyro','T_z
'RA_xacc', 'RA_yacc', 'RA_zacc', 'RA_xgyro', 'RA_ygyro','RA_zgyro','RA_xmag',
'LA_xacc', 'LA_yacc', 'LA_zacc', 'LA_xgyro', 'LA_ygyro','LA_zgyro','LA_xmag',
'RL_xacc', 'RL_yacc', 'RL_zacc', 'RL_xgyro', 'RL_ygyro','RL_zgyro','RL_xmag',
'LL_xacc', 'LL_yacc', 'LL_zacc', 'LL_xgyro','LL_ygyro','LL_zgyro','LL_xmag',
```

In [7]:
```
# Data collected by the 45 sensors for the 19 Activities done by the 8 people
complete_data
```

Out[7]:

	T_xacc	T_yacc	T_zacc	T_xgyro	T_ygyro	T_zgyro	T_xmag	T_ymag	T_zmag
0	8.13050	1.03490	5.42170	-0.009461	0.001915	-0.003424	-0.78712	-0.069654	0.157300
1	8.13050	1.02020	5.38430	-0.009368	0.023485	0.001953	-0.78717	-0.068275	0.158900
2	8.16040	1.02010	5.36220	0.015046	0.014330	0.000204	-0.78664	-0.068277	0.158790
3	8.16030	1.00520	5.37700	0.006892	0.018045	0.005649	-0.78529	-0.069849	0.159120
4	8.16050	1.02750	5.34730	0.008811	0.030433	-0.005346	-0.78742	-0.068796	0.159160
...	...	...	...	...	...	...	...	...	...
120	16.00800	-2.01660	-0.58220	2.027100	1.656800	0.584410	-0.73195	-0.476070	-0.013494
121	8.28230	-0.69936	0.48698	2.887900	1.603900	-0.020417	-0.73055	-0.472470	-0.012385
122	2.71210	0.49967	0.84053	1.996400	1.465800	-0.072605	-0.72533	-0.478630	-0.012810
123	2.03080	-0.71349	-0.11264	1.766100	1.010300	-0.102120	-0.71933	-0.482240	-0.011469
124	-0.04915	0.76302	-0.19343	2.590200	0.179090	0.011850	-0.71592	-0.483020	0.022000

1140000 rows × 46 columns

In [8]:
```
#Separate the Data from the Classes (Targets).

X_raw = complete_data.iloc[:,:45]     # X_raw is a df 1140000 rows x 45 colum
y_raw = complete_data.iloc[:,-1]      # y_raw is a df 1140000 rows x 1 column
```

1.d) Saving the X_raw and Y_raw as pickle files into local directory

In [9]:
```
# Data Saved to local drive to speed up iterations of preprocessing. (To avoi

X_raw.to_pickle('X_raw.pkl')
y_raw.to_pickle('y_raw.pkl')
```

--

2) Preprocessing

2.a) Loading libraries

In [10]:
```
from sklearn import preprocessing
from sklearn.preprocessing import normalize
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

2.b) Data Standardization and Variance Analysis

In [11]:
```
# Reload X_raw and y_raw from Local Drive

X_raw = pd.read_pickle('X_raw.pkl')
y_raw = pd.read_pickle('y_raw.pkl')
```

Using StandardScaler to standarize the dataset into unit scales (mean = 0 and variance = 1)

In [12]:
```
X_std = StandardScaler().fit_transform(X_raw)
X_std.shape
```

Out[12]:
```
(1140000, 45)
```

Alternate way to standardize and normalize

#the indices of the rows keep repeating every 125 rows. Reset the indices Alt_X_raw = X_raw.reset_index(drop=True) Alt_Y_raw = y_raw.resent_index(drop=True)

#Convert to Numpy data X = df3.to_numpy()

#Standardize and normalize the data in sections of 7500 rows starting_points = range(0,len(X)+1,7500) #added the +1 so you get the last index Alt_X_std = np.empty(X.shape) count = 0 for i in range(len(starting_points)-1): X_a = X[starting_points[i]:starting_points[i+1]] X_ax = StandardScaler().fit_transform(X_a) X_axn = preprocessing.normalize(X_ax, norm = 'l2', axis = 0) Alt_X_std[starting_points[i]:starting_points[i+1], :] = X_axn

#rename the variable for the next steps of analysis X_std = Alt_X_std

In [13]:
```
# Computing the Covariance Matrix
X_sm = X_std
X_cov = X_sm.T.dot(X_sm) / (X_sm.shape[0] - 1)

# Perform the eigendecomposition of the covariance matrix
eig_vals, eig_vecs = np.linalg.eig(X_cov)
```

In [14]:
```
def percvar(v):
    """Transform eigen/singular values into percents.
    Return: vector of percents, prefix vector of percents
    """
    # sort values
    s = np.sort(np.abs(v))
    # reverse sorting order
    s = s[::-1]
    # normalize
    s = s/np.sum(s)
    return s, np.cumsum(s)
print("eigenvalues:    ", eig_vals)
pct, pv = percvar(eig_vals)
print("percent values: ", pct)
print("prefix vector:  ", pv)
```

```
eigenvalues:     [5.94925544 5.08965484 3.49066243 2.62163963 1.98432653 1.6
0852254
 1.55318668 1.52652034 1.48942255 1.24562937 1.17867867 1.16164134
 1.08064068 1.04611417 0.96530666 0.90531748 0.86429856 0.84133456
 0.810782   0.78435298 0.71010623 0.67925309 0.64631549 0.61210055
 0.60157498 0.54459872 0.08079    0.09542987 0.11111812 0.14111989
 0.12400694 0.12696877 0.51456235 0.49733088 0.45635433 0.41493837
 0.39744966 0.18482402 0.19691699 0.21394258 0.24799887 0.26872089
 0.29788336 0.31318666 0.32526039]
percent values:  [0.13220556 0.11310334 0.07757021 0.05825861 0.04409611 0.0
3574491
 0.03451523 0.03392264 0.03309825 0.02768063 0.02619284 0.02581423
 0.02401422 0.02324696 0.02145124 0.02011815 0.01920662 0.01869631
 0.01801736 0.01743005 0.01578012 0.0150945  0.01436255 0.01360222
 0.01336832 0.01210218 0.01143471 0.01105179 0.0101412  0.00922084
 0.00883221 0.007228   0.0069597  0.00661962 0.00597157 0.00551108
 0.00475428 0.00437593 0.0041072  0.00313599 0.00282153 0.00275571
 0.00246929 0.00212066 0.00179533]
prefix vector:   [0.13220556 0.2453089  0.32287911 0.38113772 0.42523382 0.4
6097874
 0.49549397 0.52941661 0.56251486 0.59019549 0.61638833 0.64220256
 0.66621677 0.68946373 0.71091497 0.73103312 0.75023974 0.76893605
 0.78695341 0.80438346 0.82016358 0.83525808 0.84962064 0.86322286
 0.87659118 0.88869337 0.90012807 0.91117986 0.92132106 0.93054191
 0.93937411 0.94660211 0.95356181 0.96018144 0.96615301 0.97166409
 0.97641836 0.98079429 0.98490149 0.98803748 0.99085901 0.99361472
 0.99608401 0.99820467 1.        ]
```

In [15]:
```
def perck(s, p):
    s = [x for x in s if x <= p]
    return len(s)

for p in [40, 60, 80, 85, 90, 95, 99, 100]:
    print("Number of dimensions to account for %d%% of the variance: %d" % (p
```

```
Number of dimensions to account for 40% of the variance: 4
Number of dimensions to account for 60% of the variance: 10
Number of dimensions to account for 80% of the variance: 19
Number of dimensions to account for 85% of the variance: 23
Number of dimensions to account for 90% of the variance: 26
Number of dimensions to account for 95% of the variance: 32
Number of dimensions to account for 99% of the variance: 40
Number of dimensions to account for 100% of the variance: 44
```

- It seems that 32 dimensions capture 95% of the variance in the original data set.

2.c) Logistic Regression (Baseline)

Checking the accuracy of the data before implementing Dimensionality Reduction. This extra step helps us better understand how the different methods of dimensionality reduction can affect the performance and accuracy of the model.

This model has parameters: n_plist=5 , n_repeats=2. Scoring for accucary gives us a mean Accuracy of 85.61% with a very small Std (0.00045).

In [17]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [18]:
```
# model_1 = LogisticRegression(class_weight='balanced')

# cv_1 = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=0)
# scores_1 = cross_val_score(model_1, X_std, y_raw, scoring='accuracy', cv=cv_

# print(f'Accuracy: {np.mean(scores_1): .5f}(std: {np.std(scores_1): .5f})')
```

```
Accuracy:  0.85613(std:  0.00045)
```

2.c) Computing PCA for Dimesionality Reduction

- PCA is an unsupervised linear dimensionality reduction technique that helps us to denty patterns in the data based of the correlation between features.
- Based on results from the perck function, we selected 32 dimensions to capture 95% of the variance of the original data set.

In [19]:
```
pca = PCA(n_components = 32)
X_PCA = pca.fit_transform(X_std)
```

In [20]:
```
X_PCA.shape
```

Out[20]:
```
(1140000, 32)
```

* Checking the Accuracy of the model after PCA was implented and n_components were down to 32.

In [ ]:
```
# model_check_PCA = LogisticRegression(class_weight='balanced')

# cv_2 = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=0)
# scores_2 = cross_val_score(model_check_PCA, X_PCA, y_raw, scoring='accuracy

# print(f'Accuracy: {np.mean(scores_2): .5f}(std: {np.std(scores_2): .5f})')
```

2.d) Splitting the Data into Training (90%) and Test (10%)

In [22]:
```
# Split the data into train and test
# random_state=10 TO KEEP THE SET selection Constant.
X_train, X_test, y_train, y_test = train_test_split(X_PCA,y_raw, test_size=0.
len(X_test), len(y_test), len(X_train), len(y_train),
```

Out[22]:
```
(114000, 114000, 1026000, 1026000)
```

In [23]:
```
# Saving the four files on Local Directory as txt
np.savetxt('X_train.txt', X_train, delimiter=",", newline="\n")
np.savetxt('y_train.txt', y_train, delimiter=",", newline="\n")
np.savetxt('X_test.txt', X_test, delimiter=",", newline="\n")
np.savetxt('y_test.txt', y_test, delimiter=",", newline="\n")
```

In [25]:
```
# Split the data further into even smaller train and test datasets for the pr
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X_test,y_test, te

"""Save these smaller files for professor"""

np.savetxt('X_train_p.txt', X_train_p, delimiter=",", newline="\n")
np.savetxt('y_train_p.txt', y_train_p, delimiter=",", newline="\n")
np.savetxt('X_test_p.txt', X_test_p, delimiter=",", newline="\n")
np.savetxt('y_test_p.txt', y_test_p, delimiter=",", newline="\n")

len(X_test_p), len(y_test_p), len(X_train_p), len(y_train_p),
```

Out[25]:
```
(1140, 1140, 112860, 112860)
```

--

3) CLASSIFICATION METHODS

3.a) First Classification Model: "ANN"

In [26]:
```
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import classification_report
```

2-Layer ANN: No hidden layer

In [27]:
```
model1 = keras.Sequential([
    keras.layers.Dense(20, input_shape=(32,),activation= 'sigmoid')
])
model1.compile(
    optimizer='adam',
    loss= 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model1.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
32063/32063 [==============================] - 21s 630us/step - loss: 0.7824
- accuracy: 0.7563
Epoch 2/5
32063/32063 [==============================] - 20s 633us/step - loss: 0.6947
- accuracy: 0.7738
Epoch 3/5
32063/32063 [==============================] - 20s 632us/step - loss: 0.6899
- accuracy: 0.7742
Epoch 4/5
32063/32063 [==============================] - 20s 628us/step - loss: 0.6879
- accuracy: 0.7743
Epoch 5/5
32063/32063 [==============================] - 21s 644us/step - loss: 0.6869
- accuracy: 0.7743
```

Out[27]:
```
<tensorflow.python.keras.callbacks.History at 0x2e192bdbe20>
```

In [28]:
```
model1.evaluate(X_test, y_test)
```

```
3563/3563 [==============================] - 2s 497us/step - loss: 0.6803 -
accuracy: 0.7735
```

Out[28]:
```
[0.6803039312362671, 0.7734736800193787]
```

In [29]:
```
# This will predict all activitiyes and output and array of scores
y_predicted1 = model1.predict(X_test)

# To select item # 5 from the list and look ate the array of scores for A1 - /
y_predicted1[5]
```

Out[29]:
```
array([1.4699018e-06, 6.8650037e-02, 6.0890782e-01, 1.3457964e-07,
       6.0516960e-08, 8.8590151e-01, 2.7256906e-03, 9.5041096e-01,
       2.6585782e-01, 8.0482912e-01, 9.0572208e-01, 9.9402159e-01,
       5.1160395e-01, 9.8213899e-01, 1.0989115e-06, 3.4124976e-06,
       5.4173827e-02, 1.7443299e-04, 7.5451887e-01, 9.8059201e-01],
      dtype=float32)
```

In [30]:
```
# Pick the maximun Score from the prdicted array of scores.
np.argmax(y_predicted1[5])
```

Out[30]:
```
11
```

In [31]:
```
y_predicted_labels1 = [np.argmax(i) for i in y_predicted1]
y_predicted_labels1[:5]
```

Out[31]:
```
[5, 6, 5, 2, 14]
```

In [32]:
```
y_test[0:5]
```

Out[32]:
```
4      5
13     6
9      5
64    13
26    14
Name: Activity, dtype: int64
```

In [33]:
```
print("Classification Report for 2 Layes Network: \n", classification_report(
```

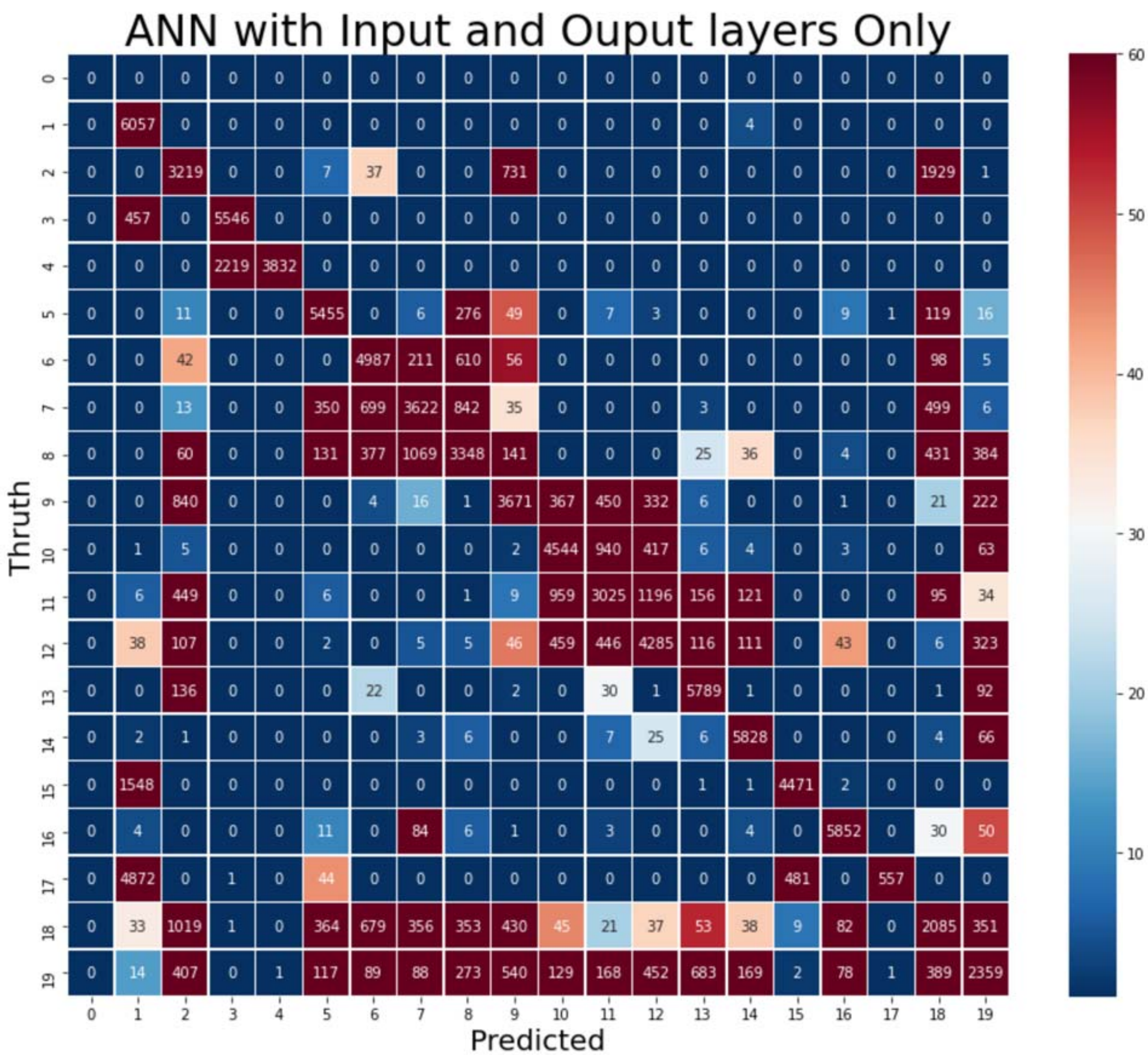
```
Classification Report for 2 Layes Network: 
               precision    recall  f1-score   support

           1       0.46      1.00      0.63      6061
           2       0.51      0.54      0.53      5924
           3       0.71      0.92      0.81      6003
           4       1.00      0.63      0.78      6051
           5       0.84      0.92      0.88      5952
           6       0.72      0.83      0.77      6009
           7       0.66      0.60      0.63      6069
           8       0.59      0.56      0.57      6006
           9       0.64      0.62      0.63      5931
          10       0.70      0.76      0.73      5985
          11       0.59      0.50      0.54      6057
          12       0.64      0.72      0.67      5992
          13       0.85      0.95      0.90      6074
          14       0.92      0.98      0.95      5948
          15       0.90      0.74      0.81      6023
          16       0.96      0.97      0.97      6045
          17       1.00      0.09      0.17      5955
          18       0.37      0.35      0.36      5956
          19       0.59      0.40      0.48      5959

    accuracy                           0.69    114000
   macro avg       0.72      0.69      0.67    114000
weighted avg       0.72      0.69      0.67    114000
```

In [34]:
```
cm1 = tf.math.confusion_matrix(labels=y_test, predictions=y_predicted_labels1
cm1
```

Out[34]:
```
<tf.Tensor: shape=(20, 20), dtype=int32, numpy=
array([[   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0, 6057,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    4,    0,    0,    0,    0,    0],
       [   0,    0, 3219,    0,    0,    7,   37,    0,    0,  731,    0,
           0,    0,    0,    0,    0,    0,    0, 1929,    1],
       [   0,  457,    0, 5546,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0, 2219, 3832,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,   11,    0,    0, 5455,    0,    6,  276,   49,    0,
           7,    3,    0,    0,    0,    9,    1,  119,   16],
       [   0,    0,   42,    0,    0,    0, 4987,  211,  610,   56,    0,
           0,    0,    0,    0,    0,    0,    0,   98,    5],
       [   0,    0,   13,    0,    0,  350,  699, 3622,  842,   35,    0,
           0,    0,    3,    0,    0,    0,    0,  499,    6],
       [   0,    0,   60,    0,    0,  131,  377, 1069, 3348,  141,    0,
           0,    0,   25,   36,    0,    4,    0,  431,  384],
       [   0,    0,  840,    0,    0,    0,    4,   16,    1, 3671,  367,
         450,  332,    6,    0,    0,    1,    0,   21,  222],
       [   0,    1,    5,    0,    0,    0,    0,    0,    0,    2, 4544,
         940,  417,    6,    4,    0,    3,    0,    0,   63],
       [   0,    6,  449,    0,    0,    6,    0,    0,    1,    9,  959,
        3025, 1196,  156,  121,    0,    0,    0,   95,   34],
       [   0,   38,  107,    0,    0,    2,    0,    5,    5,   46,  459,
         446, 4285,  116,  111,    0,   43,    0,    6,  323],
       [   0,    0,  136,    0,    0,    0,   22,    0,    0,    2,    0,
          30,    1, 5789,    1,    0,    0,    0,    1,   92],
       [   0,    2,    1,    0,    0,    0,    0,    3,    6,    0,    0,
           7,   25,    6, 5828,    0,    0,    0,    4,   66],
       [   0, 1548,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    1,    1, 4471,    2,    0,    0,    0],
       [   0,    4,    0,    0,    0,   11,    0,   84,    6,    1,    0,
           3,    0,    0,    4,    0, 5852,    0,   30,   50],
       [   0, 4872,    0,    1,    0,   44,    0,    0,    0,    0,    0,
           0,    0,    0,    0,  481,    0,  557,    0,    0],
       [   0,   33, 1019,    1,    0,  364,  679,  356,  353,  430,   45,
          21,   37,   53,   38,    9,   82,    0, 2085,  351],
       [   0,   14,  407,    0,    1,  117,   89,   88,  273,  540,  129,
         168,  452,  683,  169,    2,   78,    1,  389, 2359]])>
```

In [35]:
```python
import seaborn as sn
plt.figure(figsize = (15,12))
b = sn.heatmap(cm1, annot=True, fmt='d', linewidths=.5, square=True, cmap='Rd
b.axes.set_title("ANN with Input and Ouput layers Only",fontsize=30)
b.set_xlabel("Predicted",fontsize=20)
b.set_ylabel("Thruth",fontsize=20)
```

Out[35]:
```
Text(131.28000000000003, 0.5, 'Thruth')
```



3-Layer ANN: 1 hidden layer

In [36]:
```
model2 = keras.Sequential([
    keras.layers.Dense(100, input_shape=(32,),activation= 'relu'),
    keras.layers.Dense(20,activation= 'sigmoid')
])
model2.compile(
    optimizer='adam',
    loss= 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model2.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
32063/32063 [==============================] - 22s 667us/step - loss: 0.1054
- accuracy: 0.9688
Epoch 2/5
32063/32063 [==============================] - 22s 686us/step - loss: 0.0445
- accuracy: 0.9853
Epoch 3/5
32063/32063 [==============================] - 22s 684us/step - loss: 0.0377
- accuracy: 0.9871
Epoch 4/5
32063/32063 [==============================] - 22s 675us/step - loss: 0.0344
- accuracy: 0.9880
Epoch 5/5
32063/32063 [==============================] - 22s 679us/step - loss: 0.0324
- accuracy: 0.9887
```

Out[36]:
```
<tensorflow.python.keras.callbacks.History at 0x2e1d85628e0>
```

In [37]:
```
r"""Printing the Accuracy and the losss values"""
model2.evaluate(X_test, y_test)
```

```
3563/3563 [==============================] - 2s 522us/step - loss: 0.0312 -
accuracy: 0.9898
```

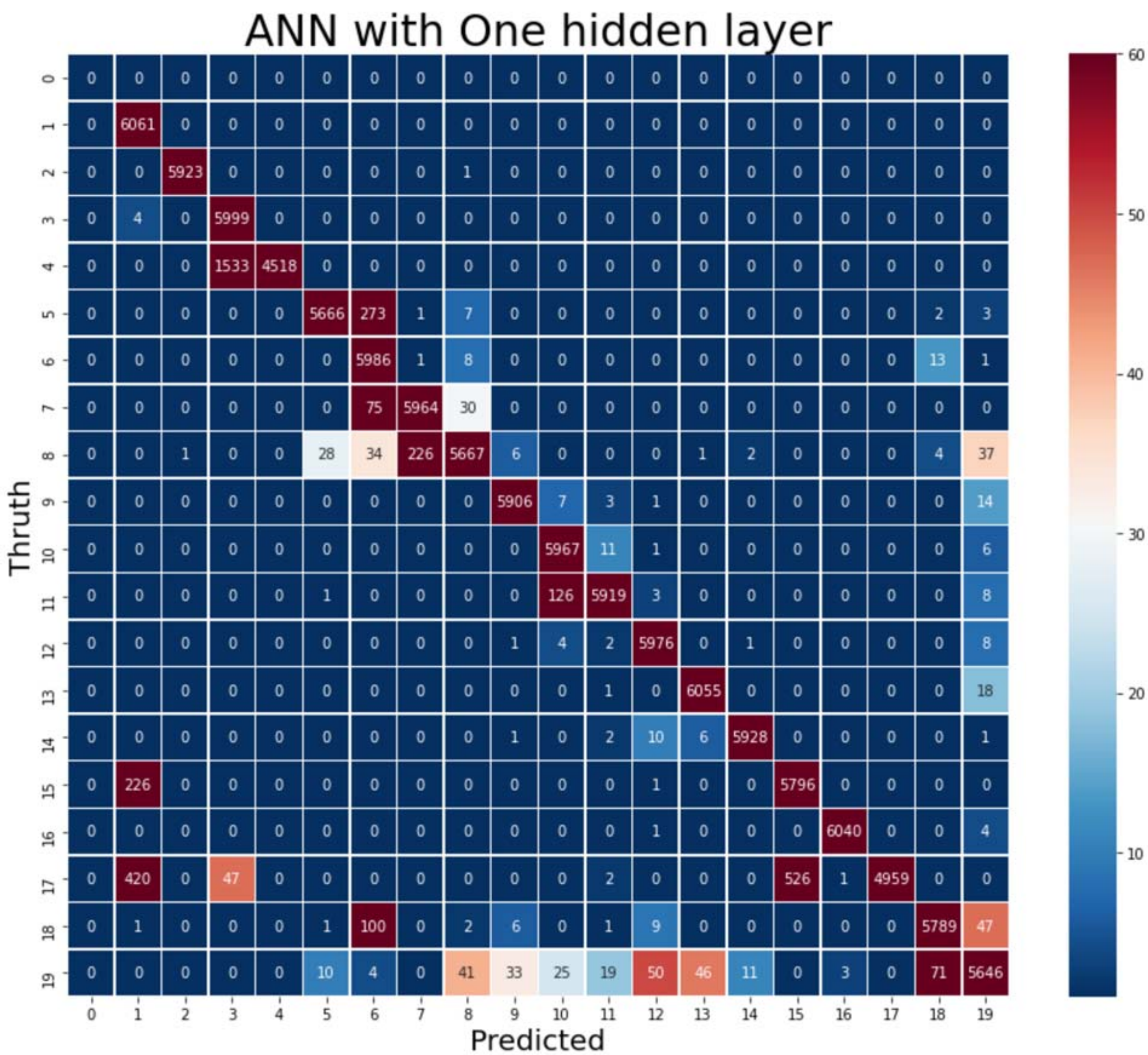
Out[37]:
```
[0.031220970675349236, 0.9897631406784058]
```

In [38]:
```
# y_predicted for teh second Model (With a hidden layer)
y_predicted2 = model2.predict(X_test)
y_predicted_labels2 = [np.argmax(i) for i in y_predicted2]
print("Classification Report for 3 Layes Network: \n", classification_report(
```

```
Classification Report for 3 Layes Network:
               precision    recall  f1-score   support

           1       0.90      1.00      0.95      6061
           2       1.00      1.00      1.00      5924
           3       0.79      1.00      0.88      6003
           4       1.00      0.75      0.85      6051
           5       0.99      0.95      0.97      5952
           6       0.92      1.00      0.96      6009
           7       0.96      0.98      0.97      6069
           8       0.98      0.94      0.96      6006
           9       0.99      1.00      0.99      5931
          10       0.97      1.00      0.99      5985
          11       0.99      0.98      0.99      6057
          12       0.99      1.00      0.99      5992
          13       0.99      1.00      0.99      6074
          14       1.00      1.00      1.00      5948
          15       0.92      0.96      0.94      6023
          16       1.00      1.00      1.00      6045
          17       1.00      0.83      0.91      5955
          18       0.98      0.97      0.98      5956
          19       0.97      0.95      0.96      5959

    accuracy                           0.96    114000
   macro avg       0.97      0.96      0.96    114000
weighted avg       0.97      0.96      0.96    114000
```

In [39]:
```
cm2 = tf.math.confusion_matrix(labels=y_test, predictions=y_predicted_labels2
plt.figure(figsize = (15,12))
b = sn.heatmap(cm2, annot=True, fmt='d', linewidths=.5, square=True, cmap='Rd
b.axes.set_title('ANN with One hidden layer',fontsize=30)
b.set_xlabel('Predicted',fontsize=20)
b.set_ylabel('Thruth',fontsize=20)
```

Out[39]:
```
Text(131.28000000000003, 0.5, 'Thruth')
```



4-Layer ANN: 2 hidden layers

In [40]:
```
model3 = keras.Sequential([
    keras.layers.Dense(100, input_shape=(32,),activation= 'relu'),
    keras.layers.Dense(100,activation= 'relu'),
    keras.layers.Dense(20,activation= 'sigmoid')
])
model3.compile(
    optimizer='adam',
    loss= 'sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model3.fit(X_train, y_train, epochs=5)
```

```
Epoch 1/5
32063/32063 [==============================] - 23s 721us/step - loss: 0.0807
- accuracy: 0.9741
Epoch 2/5
32063/32063 [==============================] - 23s 723us/step - loss: 0.0345
- accuracy: 0.9876
Epoch 3/5
32063/32063 [==============================] - 23s 726us/step - loss: 0.0283
- accuracy: 0.9896
Epoch 4/5
32063/32063 [==============================] - 23s 731us/step - loss: 0.0250
- accuracy: 0.9906
Epoch 5/5
32063/32063 [==============================] - 23s 730us/step - loss: 0.0233
- accuracy: 0.9911
```

Out[40]:
```
<tensorflow.python.keras.callbacks.History at 0x2e18afdb790>
```

In [41]:
```
model3.evaluate(X_test, y_test)
```

```
3563/3563 [==============================] - 2s 536us/step - loss: 0.0265 -
accuracy: 0.9911
```

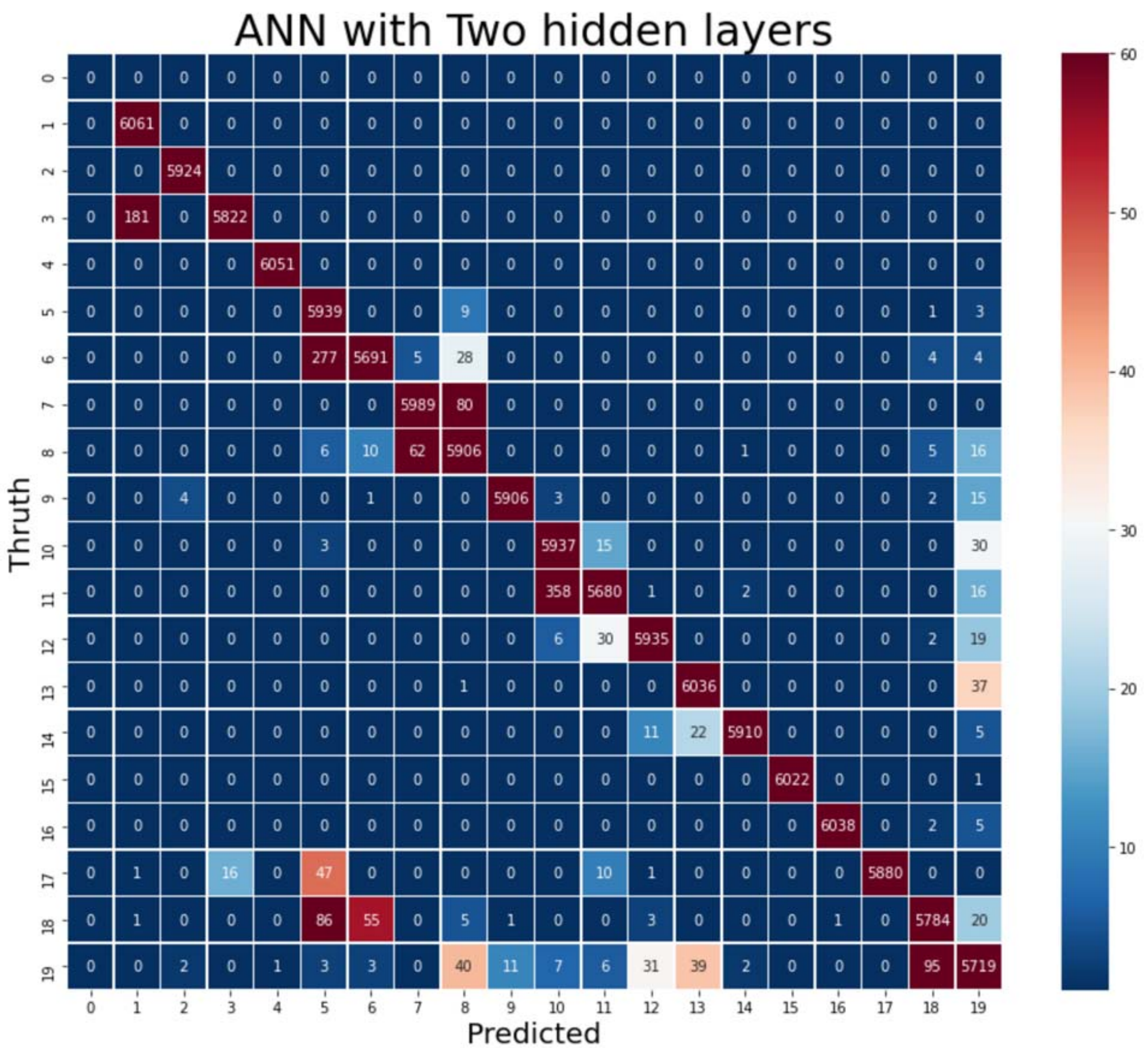
Out[41]:
```
[0.026452023535966873, 0.9910877346992493]
```

In [42]:
```
# y_predicted for the third Model (With Two hidden Layers)
y_predicted3 = model3.predict(X_test)
y_predicted_labels3 = [np.argmax(i) for i in y_predicted3]
print("ANN Class Report for 2 Hidden Layers: \n", classification_report(y_tes
```

```
ANN Class Report for 2 Hidden Layers:
               precision    recall  f1-score   support

           1       0.97      1.00      0.99      6061
           2       1.00      1.00      1.00      5924
           3       1.00      0.97      0.98      6003
           4       1.00      1.00      1.00      6051
           5       0.93      1.00      0.96      5952
           6       0.99      0.95      0.97      6009
           7       0.99      0.99      0.99      6069
           8       0.97      0.98      0.98      6006
           9       1.00      1.00      1.00      5931
          10       0.94      0.99      0.97      5985
          11       0.99      0.94      0.96      6057
          12       0.99      0.99      0.99      5992
          13       0.99      0.99      0.99      6074
          14       1.00      0.99      1.00      5948
          15       1.00      1.00      1.00      6023
          16       1.00      1.00      1.00      6045
          17       1.00      0.99      0.99      5955
          18       0.98      0.97      0.98      5956
          19       0.97      0.96      0.97      5959

    accuracy                           0.98    114000
   macro avg       0.98      0.98      0.98    114000
weighted avg       0.98      0.98      0.98    114000
```

In [43]:
```
cm3 = tf.math.confusion_matrix(labels=y_test, predictions=y_predicted_labels3
plt.figure(figsize = (15,12))
b = sn.heatmap(cm3, annot=True, fmt='d', linewidths=.5, square=True, cmap='Rd
b.axes.set_title('ANN with Two hidden layers',fontsize=30)
b.set_xlabel('Predicted',fontsize=20)
b.set_ylabel('Thruth',fontsize=20)
```

Out[43]:
```
Text(131.28000000000003, 0.5, 'Thruth')
```



3.b) Second Classification Model: "Random Forest"

Libraries

In [44]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
```

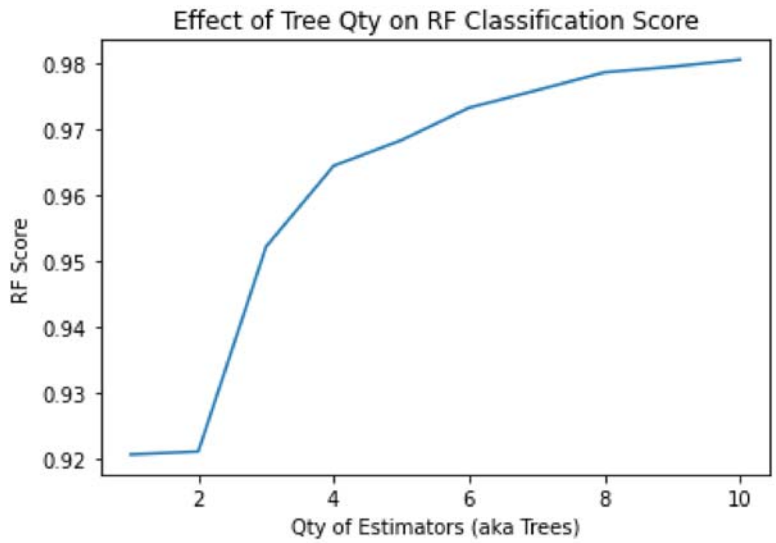
Random Forest Classifier

In [45]:
```
###X_train, X_test, y_train, y_test

##Code to find if there is a trend where score improves with estimator qty
e_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
RFscore_list = []

for e in e_list:
    clfRF = RandomForestClassifier(n_estimators = e)
    clfRF.fit(X_train, y_train)
    y_pred = clfRF.predict(X_test)
    RFscore = clfRF.score(X_test, y_test)
    print("estimator qty", e, "score:", RFscore)
    RFscore_list.append(RFscore)
```

```
estimator qty 1 score: 0.9205350877192983
estimator qty 2 score: 0.9209824561403509
estimator qty 3 score: 0.9520614035087719
estimator qty 4 score: 0.964359649122807
estimator qty 5 score: 0.9682456140350877
estimator qty 6 score: 0.9731666666666666
estimator qty 7 score: 0.9758245614035088
estimator qty 8 score: 0.9785438596491228
estimator qty 9 score: 0.9793947368421053
estimator qty 10 score: 0.9804649122807018
```

In [46]:
```
plt.figure(1)#, figsize=(20,30))
plt.plot(e_list, RFscore_list)
plt.ylabel('RF Score')
plt.xlabel('Qty of Estimators (aka Trees)')
plt.title('Effect of Tree Qty on RF Classification Score')
```

Out[46]:
```
Text(0.5, 1.0, 'Effect of Tree Qty on RF Classification Score')
```



In [47]:
```
target_names = ['a01', 'a02','a03','a04','a05','a06','a07','a08','a09','a10',
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

         a01       1.00      1.00      1.00      6061
         a02       1.00      1.00      1.00      5924
         a03       1.00      1.00      1.00      6003
         a04       1.00      1.00      1.00      6051
         a05       0.92      1.00      0.96      5952
         a06       0.96      0.95      0.95      6009
         a07       1.00      0.99      1.00      6069
         a08       0.96      0.94      0.95      6006
         a09       0.99      0.99      0.99      5931
         a10       0.98      0.99      0.98      5985
         a11       0.99      0.98      0.98      6057
         a12       0.98      1.00      0.99      5992
         a13       0.98      0.99      0.98      6074
         a14       0.99      0.99      0.99      5948
         a15       1.00      1.00      1.00      6023
         a16       1.00      1.00      1.00      6045
         a17       1.00      1.00      1.00      5955
         a18       0.97      0.94      0.95      5956
         a19       0.93      0.88      0.90      5959

    accuracy                           0.98    114000
   macro avg       0.98      0.98      0.98    114000
weighted avg       0.98      0.98      0.98    114000
```

In [48]:
```
#confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(20, 20))
plot_confusion_matrix(clfRF, X_test, y_test, normalize = "all", ax = ax)
```

Out[48]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2e18f372
3d0>
```