

Guided Project Report

Human Activity Recognition

Name: Shruti Verma

Course: AI and ML

(Batch 4)

Duration: 10 months

Problem Statement: Perform activity recognition on the [dataset](#) using a hidden markov model.

Prerequisites

What things you need to install the software and how to install them:

Python 3.8 or higher versions This setup requires that your machine has latest version of python. The following url <https://www.python.org/downloads/> can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: <https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/>. Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic.

Second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url <https://www.anaconda.com/download/> You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.8 then run below commands in command prompt/terminal to install these packages pip install -U scikit-learn pip install numpy pip install scipy if you have chosen to install anaconda then run below commands in anaconda prompt to install these packages conda install -c scikit-learn conda install -c anaconda numpy conda install -c anaconda scipy .Install hmm learn.

Video Link

<https://drive.google.com/drive/folders/1nKjHKic6ZATLwmXNwcyb1Guxb1LNOrO>

Dataset used

The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. The objective is to classify activities into one of the six activities performed. Each person performed six activities (WALKING, WALKINGUPSTAIRS, WALKINGDOWNSTAIRS, SITTING, STANDING, LAYING). There are 563 columns and about 7500 records.

Method used for detection

- Reading the dataset, preprocessing the dataset
- Reducing dimensionality for redundant features
- Applying hmm model

Importing the libraries and capturing images:

Importing the necessary libraries and reading the Human activity recognition with smartphone train and test data

```

In [18]: 1 # Importing Necessary files
          2 import numpy as np
          3 import pandas as pd
          4 from sklearn.decomposition import PCA
          5 from sklearn.metrics import f1_score, accuracy_score
          6 import matplotlib.pyplot as plt

In [2]: 1 df = pd.read_csv('../archive/train.csv')
          2 df_test = pd.read_csv('../archive/test.csv')

In [3]: 1 df.head()

Out[3]:
tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X ...  fBodyBodyGyroJerkMag-kurtosis()  angle(tBodyAcc-mean()-X)
0  0.288585 -0.020294 -0.132905 -0.995279 -0.983111 -0.913526 -0.995112 -0.983185 -0.923527 -0.934724 ...
1  0.278419 -0.016411 -0.123520 -0.998245 -0.975300 -0.960322 -0.998807 -0.974914 -0.957686 -0.940688 ...
2  0.2798653 -0.019467 -0.113462 -0.995380 -0.967187 -0.978944 -0.996520 -0.963668 -0.977469 -0.938692 ...
3  0.279174 -0.026201 -0.123283 -0.996091 -0.983403 -0.990675 -0.997099 -0.982750 -0.989302 -0.938692 ...
4  0.276629 -0.016570 -0.115362 -0.998139 -0.980817 -0.990482 -0.998321 -0.979672 -0.990441 -0.942469 ...
5 rows × 563 columns

In [4]: 1 df.shape
Out[4]: (7352, 563)

In [5]: 1 # check for null values
          2 df.isnull().sum()

Out[5]:
tBodyAcc-mean()-X      0
tBodyAcc-mean()-Y      0
tBodyAcc-mean()-Z      0
tBodyAcc-std()-X       0
tBodyAcc-std()-Y       0

```

Data processing removing any null values and doing one hot encoder using dummy variable

Jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)

In [5]:

```
1 # check for null values
2 df.isnull().sum()
```

Out[5]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-madj()-X	tBodyAcc-madj()-Y	tBodyAcc-madj()-Z	tBodyAcc-max()-X	tBodyAcc-max()-Y	tBodyAcc-max()-Z	fBodyBodyGyroJ-ske
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.526907	-0.606150	-0.468604	-0.468604
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.485942	0.414122	0.544547	0.544547
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.978162	-0.980251	-0.936219	-0.936219
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.857328	-0.857143	-0.881637	-0.881637
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.066701	-0.265671	-0.017129	-0.017129
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.967664	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 562 columns

Jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)

In [6]:

```
1 # data preprocessing
2 # find the data types and do one hot encoding using dummy variable
3
4 df.describe()
```

Out[6]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-madj()-X	tBodyAcc-madj()-Y	tBodyAcc-madj()-Z	tBodyAcc-max()-X	tBodyAcc-max()-Y	tBodyAcc-max()-Z	fBodyBodyGyroJ-ske
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.526907	-0.606150	-0.468604	-0.468604
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.485942	0.414122	0.544547	0.544547
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.978162	-0.980251	-0.936219	-0.936219
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.857328	-0.857143	-0.881637	-0.881637
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.066701	-0.265671	-0.017129	-0.017129
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.967664	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 562 columns

In [8]:

```
1 data = df.iloc[:, :-1].values
2 type(data)
3 data.shape
```

Out[8]: (7352, 562)

In [9]:

```
1 cov = np.cov(data, rowvar=False)
```

In [10]:

```
1 cov.shape
```

Out[10]: (562, 562)

In [19]:

```
1 # check the determinant
2 np.linalg.det(cov)
```

Out[19]: -0.0

Grouping by each of six activity

Jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```

Out[10]: (562, 562)

In [19]:
1 # check the determinant
2 np.linalg.det(cov)

Out[19]: -0.0

In [20]:
1 df["Activity"].unique()

Out[20]: array(['STANDING', 'SITTING', 'LAYING', 'WALKING', 'WALKING_DOWNSTAIRS',
       'WALKING_UPSTAIRS'], dtype=object)

In [12]:
1 df.groupby("Activity").groups

Out[12]:
{'LAYING': [51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, ...], 'SITTING': [27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 716, ...], 'STANDING': [0, 1, 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, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, ...], 'WALKING': [78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 434, 435, 436, 437, 438, ...], 'WALKING_DOWNSTAIRS': [125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 296, 297, 298, 299, 300, 301, 302, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 797, 798, ...], 'WALKING_UPSTAIRS': [150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, ...]}


```

Checking the distribution for each class

Jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```

In [15]:
1 # data points for all the standing activities
2 df.iloc[df.groupby("Activity").groups["STANDING"],:]

Out[15]:
tBodyAcc-mean0-X tBodyAcc-mean0-Y tBodyAcc-mean0-Z tBodyAcc-std0-X tBodyAcc-std0-Y tBodyAcc-std0-Z tBodyAcc-mad0-X tBodyAcc-mad0-Y tBodyAcc-mad0-Z tBodyAcc-max0-X ... fBodyBodyGyroJerkMag-kurtosis angle(tf)
0 0.288585 -0.020294 -0.132905 -0.995279 -0.983111 -0.913526 -0.995112 -0.983185 -0.923527 -0.934724 ...
1 0.278419 -0.016411 -0.123526 -0.998245 -0.975300 -0.960322 -0.998807 -0.974914 -0.957686 -0.943068 ...
2 0.279653 -0.019467 -0.113462 -0.995380 -0.967187 -0.978944 -0.996520 -0.963666 -0.977469 -0.938692 ...
3 0.279714 -0.026201 -0.123283 -0.996091 -0.983403 -0.996075 -0.997099 -0.982750 -0.989302 -0.938692 ...
4 0.276629 -0.016570 -0.115362 -0.998139 -0.980817 -0.990482 -0.998321 -0.979672 -0.990441 -0.942469 ...
...
7195 0.274527 -0.032259 -0.110678 -0.990631 -0.687065 -0.885434 -0.991251 -0.676243 -0.864352 -0.935640 ...
7196 0.270246 -0.067724 -0.147258 -0.990453 -0.774084 -0.926984 -0.991050 -0.793255 -0.922003 -0.937346 ...
7197 0.274422 -0.021115 -0.096432 -0.993005 -0.907217 -0.937185 -0.994032 -0.909919 -0.929252 -0.938918 ...
7198 0.274834 -0.014149 -0.080437 -0.978700 -0.871865 -0.903318 -0.984797 -0.906035 -0.908251 -0.907795 ...
7199 0.277676 0.011021 -0.062292 -0.958620 -0.815830 -0.849154 -0.971655 -0.820806 -0.851430 -0.809207 ...

1374 rows x 563 columns

In [16]:
1 # Distribution for each class
2 df.groupby("Activity").count()

Out[16]:
Activity tBodyAcc-mean0-X tBodyAcc-mean0-Y tBodyAcc-mean0-Z tBodyAcc-std0-X tBodyAcc-std0-Y tBodyAcc-std0-Z tBodyAcc-mad0-X tBodyAcc-mad0-Y tBodyAcc-mad0-Z tBodyAcc-max0-X ...
LAYING 1407 1407 1407 1407 1407 1407 1407 1407 1407 1407 ...
SITTING 1286 1286 1286 1286 1286 1286 1286 1286 1286 1286 ...
STANDING 1374 1374 1374 1374 1374 1374 1374 1374 1374 1374 ...
WALKING 1226 1226 1226 1226 1226 1226 1226 1226 1226 1226 ...
WALKING_DOWNSTAIRS 986 986 986 986 986 986 986 986 986 986 ...

```

Reduce Dimensionality and convert back to data frame

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Home Page - Select or create a notebook, Documents/Python practice/Assignments/Guided Proj..., Project14 Human Activity Recognition - Jupyter Note..., Human Activity Recognition with Smartphones | Kag...
jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Code Cell 1:** # reduce dimentionality to avoid redundant features
Applying PCA
pca = PCA(n_components = 100)
cov_pca = pca.fit(df.iloc[:, :-1].values)
- Code Cell 2:** data_train = cov_pca.transform(df.iloc[:, :-1].values)
data_test = cov_pca.transform(df_test.iloc[:, :-1].values)
- Code Cell 3:** # covert back to panda df
df_train_red = pd.DataFrame(data_train)
df_train_red["Activity"] = df["Activity"]
- Code Cell 4:** df_train_red.head()
- Output 31:** A table showing the first 5 rows of the reduced dataset. The columns are labeled 0 through 94.
- Code Cell 5:** df_train_red_STAND = df_train_red[df_train_red["Activity"]=="STANDING"]
df_train_red_SIT = df_train_red[df_train_red["Activity"]=="SITTING"]
df_train_red_WALK = df_train_red[df_train_red["Activity"]=="WALKING"]
- Output 5:** 6 rows x 562 columns

Calculating true labels and Installing HMM learn

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** "localhost", "Project14 Human Activity Recognition - Jupyter Note...", "Human Activity Recognition with Smartphones | Kag...".
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Cell 38:** Code cell containing Python code for calculating true labels from a dataset. The code uses a for loop to iterate through the 'Activity' column of 'df_test_red' and appends values 0, 1, 2, 3, 4, or 5 to 'labels_true' based on the activity type. It also handles 'WALKING_UPSTAIRS' and 'WALKING_DOWNSTAIRS' cases. The final output is an array of labels.
- Cell 38 Output:** "Out[38]: array([0, 0, 0, ..., 5, 5, 5])"
- Cell 40:** Code cell containing the command "pip install hmmlearn".
- Cell 40 Output:** A log of the pip installation process for 'hmmlearn' version 0.2.6, showing dependencies like numpy, scikit-learn, scipy, joblib, threadpoolctl, and building wheels for the package.

Applying HMM and fitting for each activity

The screenshot shows a Jupyter Notebook interface on a Mac OS X desktop. The title bar reads "localhost" and "Project14 Human Activity Recognition - Jupyter Note...". The notebook has two cells:

In [41]:

```
1 # Implementing hmm model
2 from hmmlearn import hmm
```

In [59]:

```
1 # Fitting for each activity
2 def HMM_Flscore(N,M,labels_true):
3     hmm_stand=hmm.GMMHMM(n_components=N,n_mix = M)
4     hmm_sit=hmm.GMMHMM(n_components=N,n_mix = M)
5     hmm_walk=hmm.GMMHMM(n_components=N,n_mix = M)
6     hmm_lay=hmm.GMMHMM(n_components=N,n_mix = M)
7     hmm_walk_d=hmm.GMMHMM(n_components=N,n_mix = M)
8     hmm_walk_u=hmm.GMMHMM(n_components=N,n_mix = M)
9
10    hmm_stand.fit(df_train_red_STAND.iloc[:,0:100].values)
11    hmm_sit.fit(df_train_red_SIT.iloc[:,0:100].values)
12    hmm_walk.fit(df_train_red_WALK.iloc[:,0:100].values)
13    hmm_lay.fit(df_train_red_LAY.iloc[:,0:100].values)
14    hmm_walk_d.fit(df_train_red_WALKD.iloc[:,0:100].values)
15    hmm_walk_u.fit(df_train_red_WALKU.iloc[:,0:100].values)
16
17    #calculating F1 score
18    labels_predict = []
19    for i in range(len(df_test_red)):
20        log_likelihood_value = np.array([hmm_stand.score(df_test_red.iloc[i,0:100].values.reshape((1,100))),
21                                         hmm_sit.score(df_test_red.iloc[i,0:100].values.reshape((1,100))),
22                                         hmm_walk.score(df_test_red.iloc[i,0:100].values.reshape((1,100))),
23                                         hmm_lay.score(df_test_red.iloc[i,0:100].values.reshape((1,100))),
24                                         hmm_walk_d.score(df_test_red.iloc[i,0:100].values.reshape((1,100))),
25                                         hmm_walk_u.score(df_test_red.iloc[i,0:100].values.reshape((1,100)))]))
26
27        labels_predict.append(np.argmax(log_likelihood_value))
28
29    F1 = f1_score(labels_true,labels_predict,average = 'micro')
30    acc = accuracy_score(labels_true,labels_predict)
31    return F1,acc
```

Training separately and combining all six activity

The screenshot shows a Jupyter Notebook interface on a Mac OS X desktop. The title bar reads "localhost" and "Project14 Human Activity Recognition - Jupyter Note...". The notebook has two cells:

In [60]:

```
1 states =np.arange(1,36,1)
2 states
```

Out[60]:

```
array([ 1,  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, 31, 32, 33, 34,
       35])
```

In [61]:

```
1 # training separately and than combining
2 F1_value_states = []
3 acc_value_states = []
4 for i in states:
5     print("HMM is trained for states ={}".format(i))
6     f1,acc = HMM_Flscore(i,labels_true)
7     F1_value_states.append(f1)
8     acc_value_states.append(acc)
9 fig,ax = plt.subplots(2,1)
10
11 ax[0].plot(F1_value_states)
12 ax[1].plot(acc_value_states)
13
14 plt.show()
```

The output of the second cell shows the following printed text:

```
HMM is trained for states =1
HMM is trained for states =2
HMM is trained for states =3
HMM is trained for states =4
HMM is trained for states =5
HMM is trained for states =6
HMM is trained for states =7
HMM is trained for states =8
HMM is trained for states =9
HMM is trained for states =10
HMM is trained for states =11
HMM is trained for states =12
HMM is trained for states =13
HMM is trained for states =14
HMM is trained for states =15
HMM is trained for states =16
HMM is trained for states =17
HMM is trained for states =18
HMM is trained for states =19
```

Plotting to show the f1 value and accuracy

Jupyter Project14 Human Activity Recognition Last Checkpoint: Yesterday at 12:45 AM (autosaved)

In [64]:

```
1 f1_test =[]
2 acc_test =[]
3
4 for i in range(1,6):
5     f1,acc1 =HMM_Flscore(3,i,labels_true)
6     f1_test.append(f1)
7     acc_test.append(acc)
8
9 fig,ax = plt.subplots(2,1)
10
11 ax[0].plot(f1_test)
12 ax[1].plot(acc_test)
13
14 plt.show()
```

In [65]:

```
1 f1_val,acc_val = HMM_Flscore(3,8,labels_true)
2 print(f1_val)
3 print(acc_val)
```

0.5256192738378012
0.5256192738378012

The Jupyter Notebook interface displays two plots. The first plot, generated by In [64], shows the F1 score (blue line) and accuracy (orange line) across 35 iterations. The F1 score starts around 0.45 and fluctuates between 0.4 and 0.45. The accuracy starts around 0.45 and fluctuates between 0.4 and 0.45. The second plot, generated by In [65], shows the F1 score (blue line) and accuracy (orange line) across 4 iterations. The F1 score starts at approximately 0.7 and decreases to about 0.6. The accuracy remains constant at approximately 0.38.

