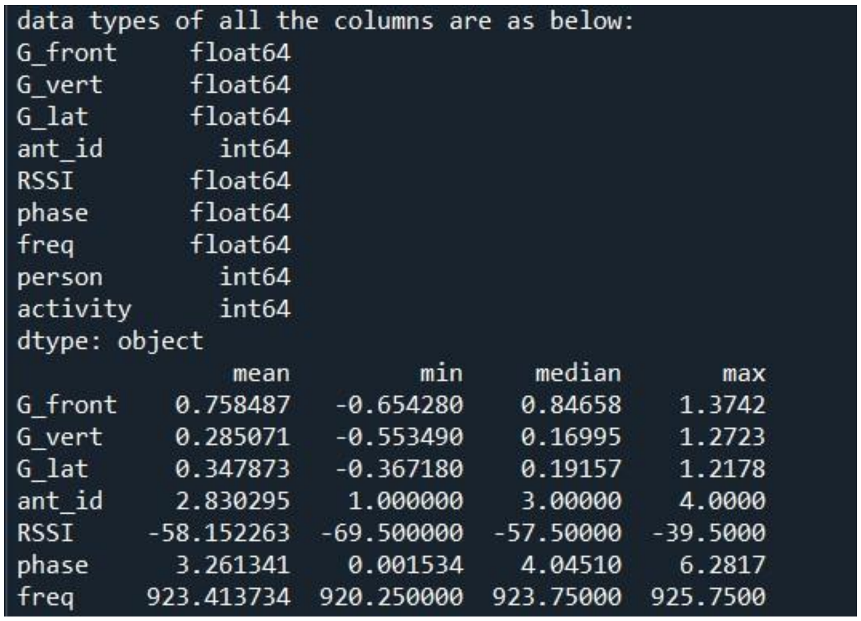
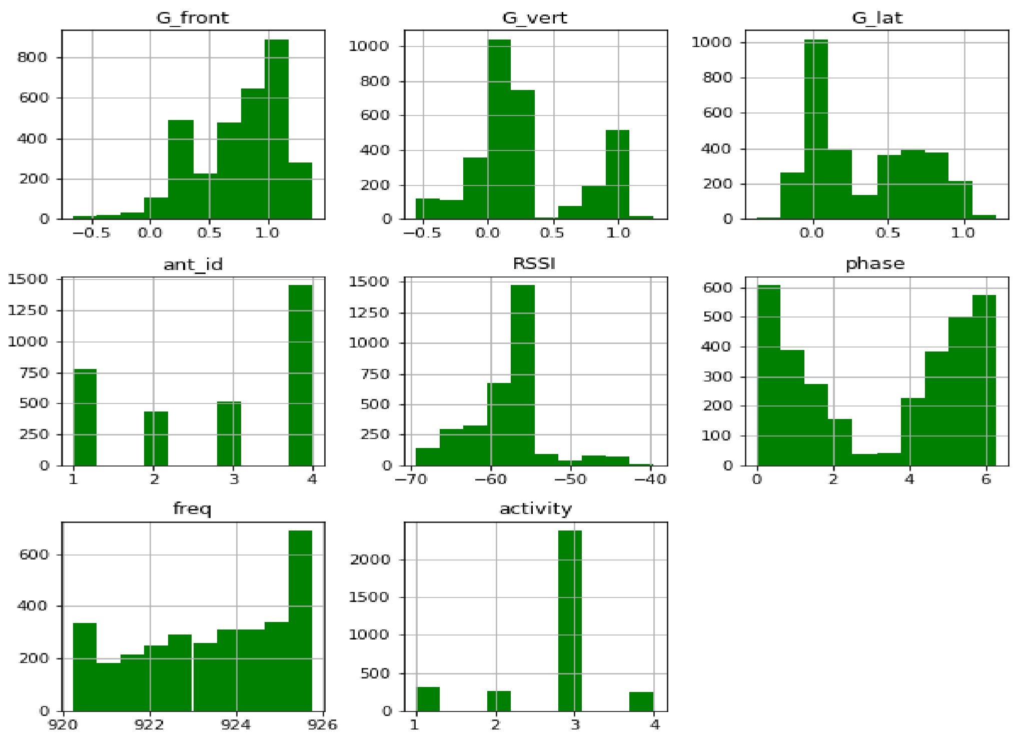
- ✓ “You may benefit from generating a data profile!” – Question paper: point 2.
    - o <u>Data types and Data profile:</u>



```
data types of all the columns are as below:
G_front      float64
G_vert       float64
G_lat        float64
ant_id         int64
RSSI         float64
phase        float64
freq         float64
person         int64
activity       int64
dtype: object
                mean         min      median        max
G_front     0.758487   -0.654280    0.84658     1.3742
G_vert      0.285071   -0.553490    0.16995     1.2723
G_lat       0.347873   -0.367180    0.19157     1.2178
ant_id      2.830295    1.000000    3.00000     4.0000
RSSI      -58.152263  -69.500000  -57.50000   -39.5000
phase       3.261341    0.001534    4.04510     6.2817
freq      923.413734  920.250000  923.75000   925.7500
```

    - o <u>Histogram:</u>



- ✓ “Describe your four classifiers (including all hyperparameter settings) in your Findings document.” – Question paper: point 3.

  - <u>Decision Tree:</u>
    - Used “DecisionTreeClassifier” from scikit-learn. It has the capability to perform multi-class classification on a dataset.
    - Parameters:
      - **Criterion:** When the split happens, this parameter measures the quality of the split. I have used the default value “gini”. Although “entropy” and “gini” work internally in similar fashion for measuring the quality of the split, I understand that “gini” has better computing power than “entropy”. So, for this reason I have used the default value “gini” provided by the decision tree classifier.
      - **Random State:** To generate consistent results when the algorithm is running multiple times, we should not have any shuffling of the data. Hence, I have set the random state as “10”, “40”, “80”, “120”, “150”, “200” (multiple evaluations methodologies – so used specific random state for each method) and used the same random state when running the experiments multiple times.

  - <u>Random Forest:</u>
    - Used “RandomForestClassifier” from scikit-learn. It is basically a set of decision trees from a randomly selected subset of the training set.
    - Parameters:
      - **N estimators:** It is the number of trees in the forest. I used the default value 100, didn’t change it.
      - **Criterion:** When the split happens, this parameter measures the quality of the split. I have used the default value “gini”. Although “entropy” and “gini” work internally in similar fashion for measuring the quality of the split, I understand that “gini” has better computing power than “entropy”. So, for this reason I have used the default value “gini” provided by the random forest classifier.
      - **Random State:** To generate consistent results when the algorithm is running multiple times, we should not have any shuffling of the data. Hence, I have set the random state as “20”, “50”, “90”, “130”, “160”, “210” (multiple evaluations methodologies – so used specific random state for each method) and used the same random state when running the experiments multiple times.

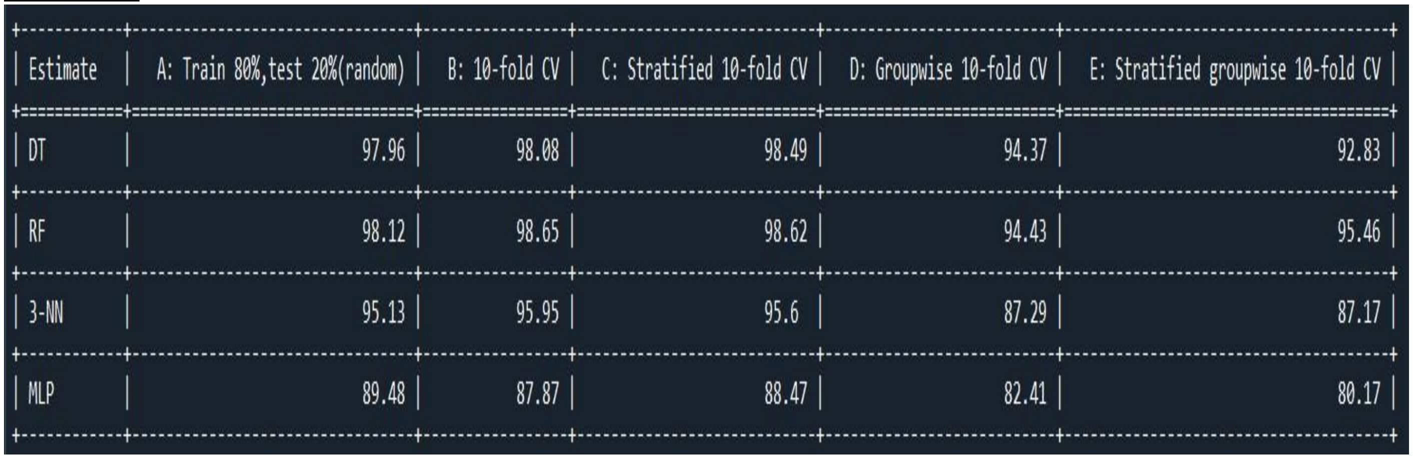
- K Neighbors:
    - Used “KNeighborsClassifier” from scikit-learn. KNN algorithm assumes that similar things are close to each other. It captures the idea of similarity (proximity).
    - Parameters:
        - **N neighbors:** It is the number of neighbors to use for the kneighbors queries. It’s default value is 5, but in the question paper, it talks about 3-NN, where k = 3 which is the number of neighbors, so I have set the value of n neighbors as “3”.
        - **Weights:** With reference to scikit-learn documentation, I didn’t change this parameter, used the default value “uniform” to have equal weights for all the points in the neighborhood.

- MLP:
    - It is multi-layer perceptron classifier. Used “MLPClassifier” from neural networks library of scikit-learn. It is a feed forward artificial neural network model which converts input datasets into a collection of useful outputs.
    - Parameters:
        - **Activation:** It is the activation function for the hidden layer. Used the default value “relu” which is rectified linear unit function.
        - **Solver:** The solver for weight optimization. The default value is “adam” but I read that for smaller datasets “lbfgs”, an optimizer in the family of quasi-newton methods will converge faster and perform better. So, I used “lbfgs” as the solver parameter.
        - **Random State:** To generate consistent results when the algorithm is running multiple times, we should not have any shuffling of the data. Hence, I have set the random state as “30”, “70”, “110”, “140”, “180”, “220” (multiple evaluations methodologies – so used specific random state for each method) and used the same random state when running the experiments multiple times.

- Baseline Classifier:
    - Used “DummyClassifier” from scikit-learn. Randomly predicting the outcome from class probabilities of the dev set.
    - Parameters:
        - **Strategy:** Default strategy is “prior” but used “stratified” as given in the question which concentrates on the class probabilities for randomly predicting the outcome.
        - **Random State:** For the above-mentioned reason(previous classifiers), used random state as “190” for generating consistent results.

A. Using the development data only (Step 3), for each evaluation methodology (A-E), which classifier was expected to perform best in the future?

Answer:

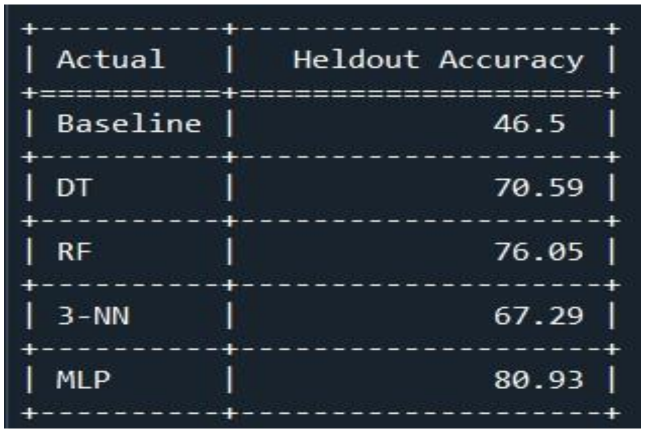


```
+----------+------------------------------+---------------+-------------------------+-------------------------+---------------------------------+
| Estimate | A: Train 80%,test 20%(random) | B: 10-fold CV | C: Stratified 10-fold CV | D: Groupwise 10-fold CV | E: Stratified groupwise 10-fold CV |
+==========+==============================+===============+=========================+=========================+=================================+
| DT       |                        97.96 |         98.08 |                   98.49 |                   94.37 |                           92.83 |
+----------+------------------------------+---------------+-------------------------+-------------------------+---------------------------------+
| RF       |                        98.12 |         98.65 |                   98.62 |                   94.43 |                           95.46 |
+----------+------------------------------+---------------+-------------------------+-------------------------+---------------------------------+
| 3-NN     |                        95.13 |         95.95 |                   95.6  |                   87.29 |                           87.17 |
+----------+------------------------------+---------------+-------------------------+-------------------------+---------------------------------+
| MLP      |                        89.48 |         87.87 |                   88.47 |                   82.41 |                           80.17 |
+----------+------------------------------+---------------+-------------------------+-------------------------+---------------------------------+
```

- ✓ As per development data when worked with the A to E evaluation methodologies, the accuracy results are as above.
- ✓ Overall, for each methodology (A-E), Random Forest performed better than the other classifiers.
- ✓ From the above table, we can conclude that in comparison, “Random Forest” seems performed better here (slightly ahead of Decision Tree) and so random forest classifier was expected to perform best in the future.

B. Which classifier actually performed best on the held-out test set (part 4)?

Answer:



```
+----------+--------------------+
| Actual   |  Heldout Accuracy  |
+==========+====================+
| Baseline |              46.5  |
+----------+--------------------+
| DT       |              70.59 |
+----------+--------------------+
| RF       |              76.05 |
+----------+--------------------+
| 3-NN     |              67.29 |
+----------+--------------------+
| MLP      |              80.93 |
+----------+--------------------+
```

- ✓ From the above held-out accuracies table we can say that the classifier “Multi-Layer Perceptron” performed well in comparison to other methods.

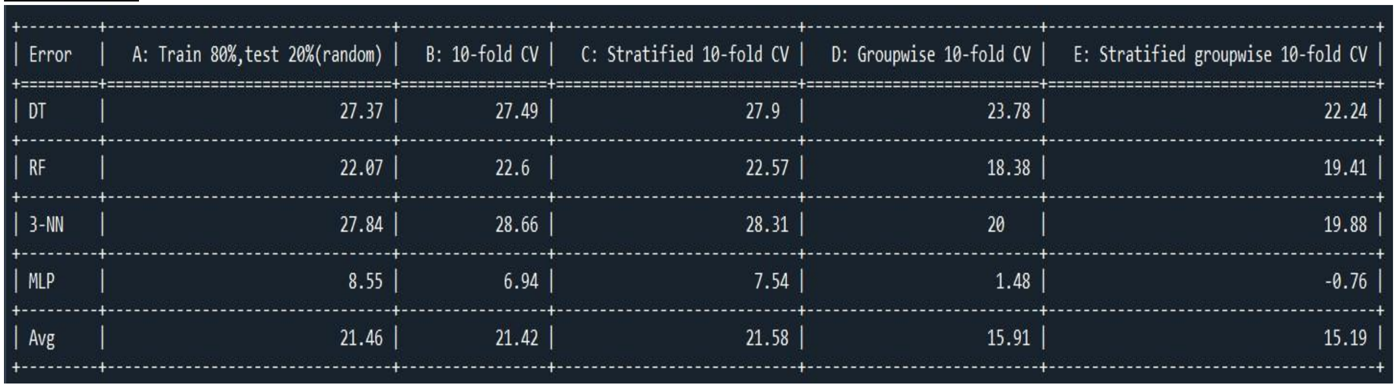
C. Which classifier(s) performed better than the baseline on the held-out set?

Answer:

- ✓ From the above table, we can confirm that all the classifiers (DT, RF, 3-NN, MLP) performed better than the baseline classifier. So, it is an indication that working on this dataset with a machine learning model with a proper classifier is productive.

D. Which evaluation method from step 3 had the smallest error (averaged across classifiers) between estimated test performance (from the development data) and actual held-out performance?

Answer:



```
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
| Error   |   A: Train 80%,test 20%(random) |   B: 10-fold CV |   C: Stratified 10-fold CV |   D: Groupwise 10-fold CV |   E: Stratified groupwise 10-fold CV |
+=========+===============================+================+===========================+=========================+===================================+
| DT      |                         27.37 |          27.49 |                      27.9 |                   23.78 |                             22.24 |
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
| RF      |                         22.07 |          22.6  |                     22.57 |                   18.38 |                             19.41 |
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
| 3-NN    |                         27.84 |          28.66 |                     28.31 |                   20    |                             19.88 |
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
| MLP     |                          8.55 |           6.94 |                      7.54 |                    1.48 |                             -0.76 |
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
| Avg     |                         21.46 |          21.42 |                     21.58 |                   15.91 |                             15.19 |
+---------+-------------------------------+----------------+---------------------------+-------------------------+-----------------------------------+
```

- ✓ From the above table, we can conclude that Method E (Stratified groupwise 10-fold cross validation) evaluation method had the smallest error (15.19%) between the estimated test performance and actual held-out performance.
- ✓ In comparison, we can see that Method D (Groupwise 10-fold cross validation) also performed very well and it is slightly less than the method E.

E. Why do you think there is a difference in the generalization estimates made by methods B, C, and D on this data set?

Answer:

- ✓ Method B:
  - o KFold focuses on partitioning the given dataset into equal folds. In each iteration, it basically selects one-fold as the validation set and the remaining folds as training set. The iterations continue till each fold is treated as validation set.
  - o But as we can see in the data profile of this dataset, we see that based on the class label "activity" we observe that this dataset is imbalanced. So, when we apply KFold for such imbalance data, there are chances that the model will not give good performance results and also there is a chance of the training and test data getting mixed up with the Kfold approach.
- ✓ Method C:
  - o StratifiedKFold fixes the issue of imbalance in the dataset which KFold cannot handle. In this method, we concentrate on stratified folds where we will make sure that each fold has roughly the same amount of target class in comparison with the full dataset.
- ✓ Method D:
  - o GroupKFold works in a better way when the data is present in multiple groups. Like in our dataset it is about data related to "different persons" and the model can be trained specific to these persons as group. The model takes out the individual groups for the training set.
- ✓ Hence, from the above observations for each method, I think there will be difference in the generalization estimates as the given dataset is imbalanced (activity).

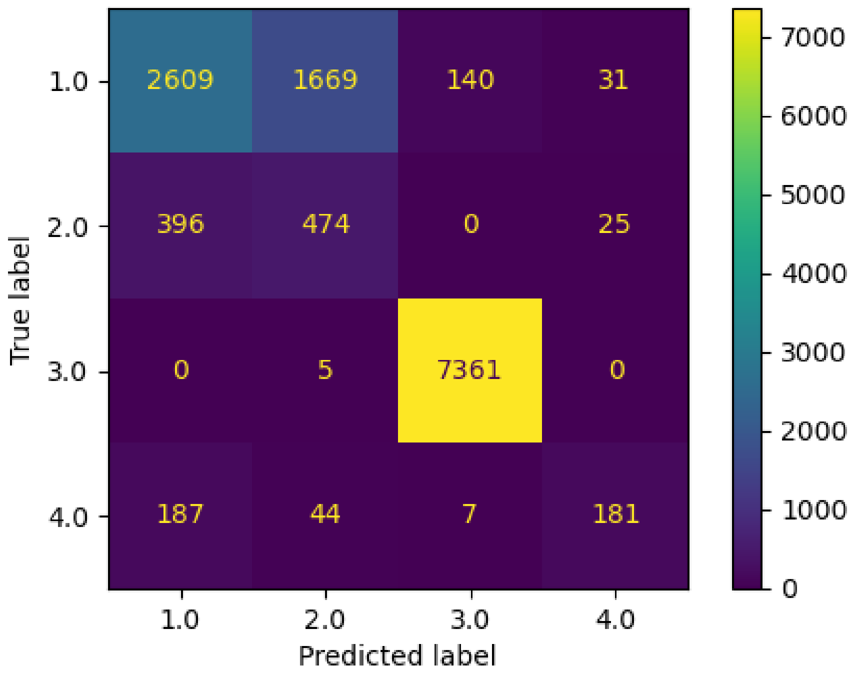
F. The held-out set contained observations from 10 new people. If instead we have a test set that contains new observations from the same people from the training set, which evaluation methodology (A-E) would give the best generalization estimate (smallest gap between predicted and actual accuracy), and why?

<u>Answer:</u>

- ✓ If we have a test set that contains new observations from the "same" people from the training set, then I think Method "E" would give the best generalization estimate.
- ✓ This is because the StratifiedGroupKFold has the class-stratified for handling the imbalances in the dataset, so it won't be impacted even if our new dataset creates more imbalance to the existing training set.
- ✓ Next, the StratifiedGroupKFold method has the group-wise strategy where we already trained our training set based on "groups" which is actually the persons. So, when we add more data for the "same" people, the new dataset will produce accurate results.
- ✓ Hence, I think method E will give the best generalization estimate. I also feel method "D"(GroupKFold) can be a close choice as the best generalization estimate as it has similar features to the method E.

G. Show the 4x4 confusion matrix for the classifier that performed best on the held-out set. Which kind of error do you think is the most important to avoid/penalize?

<u>Answer:</u>



Label	True Positive (TP)	True Negative (TN)	False Positive (FP)	False Negative (FN)
1	2609	474+25+5+7361+44+7+81 = 7997	2609+396 = **3005**	1669+140+31 = **1840**
2	474	2609+140+31+7361+187+7+81 = 10,416	1669+5+44 = **1718**	396+25 = **421**
3	7361	2609+1669+31+396+474+25+187+44+181 = 5616	140+7 = **147**	**5**
4	181	2609+1669+140+396+474+5+7361 = 12, 654	25+31 = **56**	187+44+7 = **238**

False Positive: Predicted value is positive but the actual value is negative.

False Negative: Predicted value is negative but the actual value is positive.

From the above table, we can calculate the below:

Total "False Positive" = 3005 + 1718 + 147 + 56 = 4926

Total "False Negative" = 1840 + 421 + 5 + 238 = 2504

Hence we can see that False Positive (FP) > False Negative (FN). So, it is most important to penalize "FP" in this dataset. In general, for most of the practical applications False Negative pose more risk than False Positive but in this dataset it is up to the actual users to decide which one is important to penalize. According to the numeric value as above, I think false positive have to be penalized (assuming FP and FN have same impact on the overall outcome)

H. Given what you know of how the data was collected, what kinds of bias may be present in this data set? (Recall Mehrabi et al. (2021), "A Survey on Bias and Fairness in Machine Learning", https://dl.acm.org/doi/abs/10.1145/3457607)

Answer:

- ✓ Omitted Variable Bias: The main goal of this dataset is to recognize the activities in clinical environments. Some of the activities that are involved in this are sitting on the chair, lying on bed. Since the sensor is attached to the persons, there can be movements like sitting on bed. During this also the sensor records the data as lying on bed or getting of the bed. So there is no clarity on such variables which are important. Hence, I think there are some important variables like movements on bed, movements on chair are missed and such information can give more accurate results for the model. Gender can also be considered as an important feature.
- ✓ Sampling Bias: I think the dataset doesn't talk about the gender of this dataset. A healthy older aged between 66 and 86 years old will have different walking speed, movement and flexibility based on the gender. But the data specifically talks about the person but not gender. Hence, the overall findings and results cannot be generalized as there is issue in the measurement of the details where it doesn't distinguish between gender. So, I think the results of such model are created from a sampling bias as there can be chances that this data is collected predominantly from male or female and ignored the other gender.
- ✓ Representation Bias: The dataset talks about the healthy people aged between 66 and 86 years but never talks about the people and their diversities. Because as the data is collected from 66 to 86 years old with no specification of the diversity of the people, there can be a chance that this experiment was conducted with people from America and the result of this model cannot be applied to the same age group of another continent as the health habits differ and the health of the age group differs.
- ✓ Measurement Bias: The experiment is conducted on the people where the sensors were used on top of their clothing but the sensor activity may record different reading based

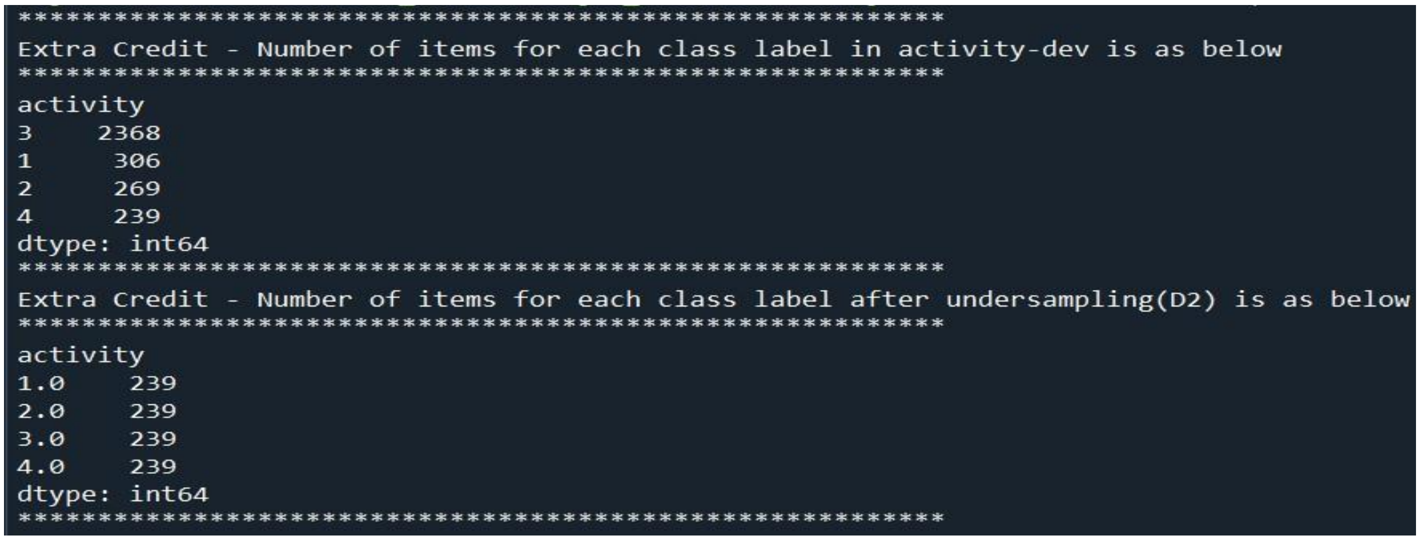
on the thickness of the clothing or even when the sensor is directly attached to the body. This results in measurement bias.

7. Examine the development set carefully. Generate a new development data D2 set. that is class-balanced, using a method of your choice that addresses class imbalance. Describe the method you used and the number of items in each class in D2. Run your activity_eval.py program using D2 to make generalization estimates, but with no change to the held-out set. In this case, which evaluation method from step 3 has the smallest error (again averaged across the four classifiers) in its generalization estimate (from D2 to held-out)? Are your conclusions about the best evaluation methodology for this problem different? Include your new development data set in your .zip file.

Answer:

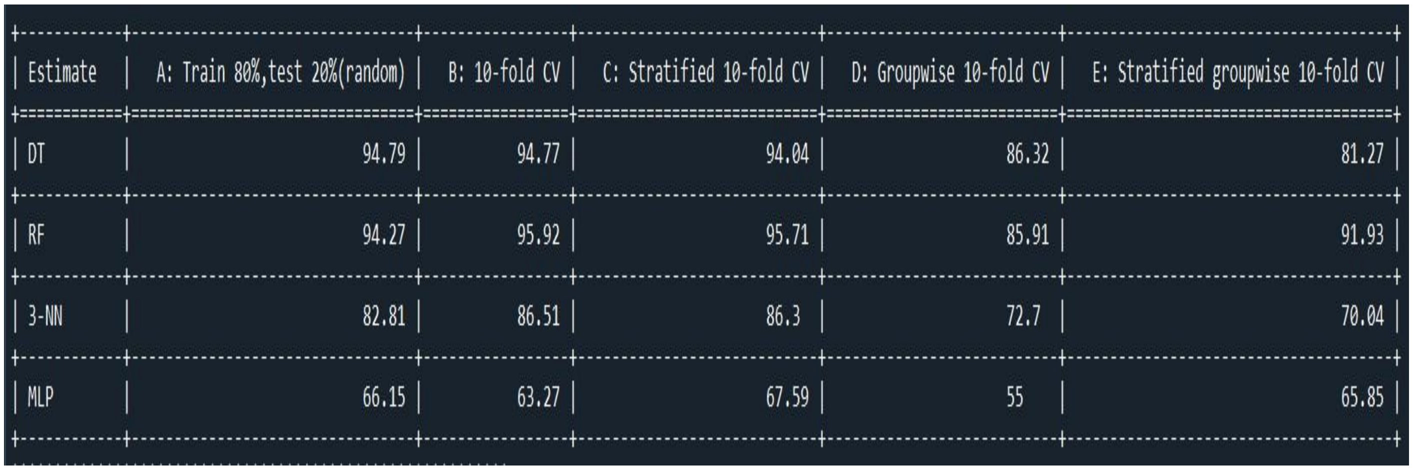
- ✓ I have submitted the extra-credit code in another python file "extra_credit.py" and mentioned the instructions in the README file. I tried to merge both the normal work and extra credit but ended up with some issues, so I have separated the work. But the "extra_credit.py" has the complete logic in-line with the requirements of the extra credit. Please accept my submission for extra credit.
- ✓ Method used:
  - I have used "RandomUnderSampler" from imblearn class which is useful for random under sampling of the dataset.
  - I used the parameter "random state" to generate the consistent results for repeated runs of the code. The other parameters I have used the default values.
  - Sampling_strategy as "auto", replacement as "False".
  - Here the count of the "minority class" (239 – label 4) is taken and applied for all the other class labels (labels – 1, 2, 3) as the under-sampling technique.

- ✓ Number of items per each class label after under-sampling:



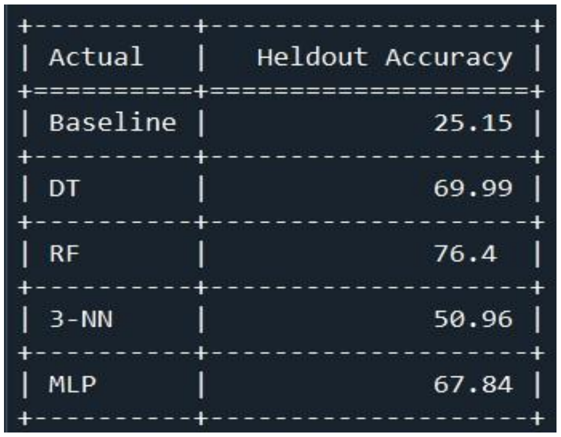
```
**********************************************************
Extra Credit - Number of items for each class label in activity-dev is as below
**********************************************************
activity
3    2368
1     306
2     269
4     239
dtype: int64
**********************************************************
Extra Credit - Number of items for each class label after undersampling(D2) is as below
**********************************************************
activity
1.0    239
2.0    239
3.0    239
4.0    239
dtype: int64
**********************************************************
```

- ✓ Generalization estimate:



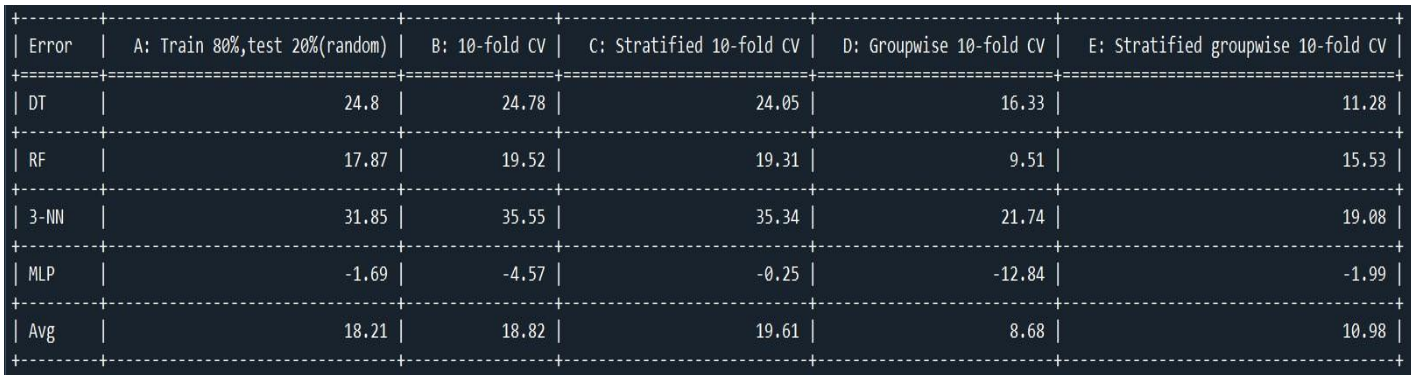
```
+----------+-----------------------------+--------------+-------------------------+-------------------------+----------------------------------+
| Estimate |  A: Train 80%,test 20%(random) | B: 10-fold CV | C: Stratified 10-fold CV | D: Groupwise 10-fold CV | E: Stratified groupwise 10-fold CV |
+==========+=============================+==============+=========================+=========================+==================================+
| DT       |                       94.79 |        94.77 |                   94.04 |                   86.32 |                            81.27 |
+----------+-----------------------------+--------------+-------------------------+-------------------------+----------------------------------+
| RF       |                       94.27 |        95.92 |                   95.71 |                   85.91 |                            91.93 |
+----------+-----------------------------+--------------+-------------------------+-------------------------+----------------------------------+
| 3-NN     |                       82.81 |        86.51 |                   86.3  |                   72.7  |                            70.04 |
+----------+-----------------------------+--------------+-------------------------+-------------------------+----------------------------------+
| MLP      |                       66.15 |        63.27 |                   67.59 |                   55    |                            65.85 |
+----------+-----------------------------+--------------+-------------------------+-------------------------+----------------------------------+
```

- ✓ <u>Held-out Accuracy:</u>



```
+----------+--------------------+
| Actual   |   Heldout Accuracy |
+==========+====================+
| Baseline |              25.15 |
+----------+--------------------+
| DT       |              69.99 |
+----------+--------------------+
| RF       |              76.4  |
+----------+--------------------+
| 3-NN     |              50.96 |
+----------+--------------------+
| MLP      |              67.84 |
+----------+--------------------+
```

- ✓ <u>Signed Error:</u>



```
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
| Error  |   A: Train 80%,test 20%(random) |   B: 10-fold CV |   C: Stratified 10-fold CV |   D: Groupwise 10-fold CV |   E: Stratified groupwise 10-fold CV |
+========+================================+===============+==========================+=========================+====================================+
| DT     |                           24.8  |         24.78 |                    24.05 |                   16.33 |                              11.28 |
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
| RF     |                           17.87 |         19.52 |                    19.31 |                    9.51 |                              15.53 |
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
| 3-NN   |                           31.85 |         35.55 |                    35.34 |                   21.74 |                              19.08 |
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
| MLP    |                           -1.69 |         -4.57 |                    -0.25 |                  -12.84 |                              -1.99 |
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
| Avg    |                           18.21 |         18.82 |                    19.61 |                    8.68 |                              10.98 |
+--------+--------------------------------+---------------+--------------------------+-------------------------+------------------------------------+
```

- ✓ Method D has the smallest error in its generalization estimate.
- ✓ It is almost the same the methodology for this problem as well. For the actual problem, I mentioned method E as the best with slight better values than method D. But there as well, method D did get small error in the generalization estimate.