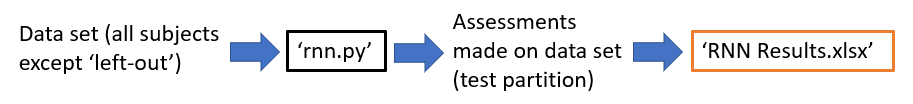
**Experiments and Results Discussion**

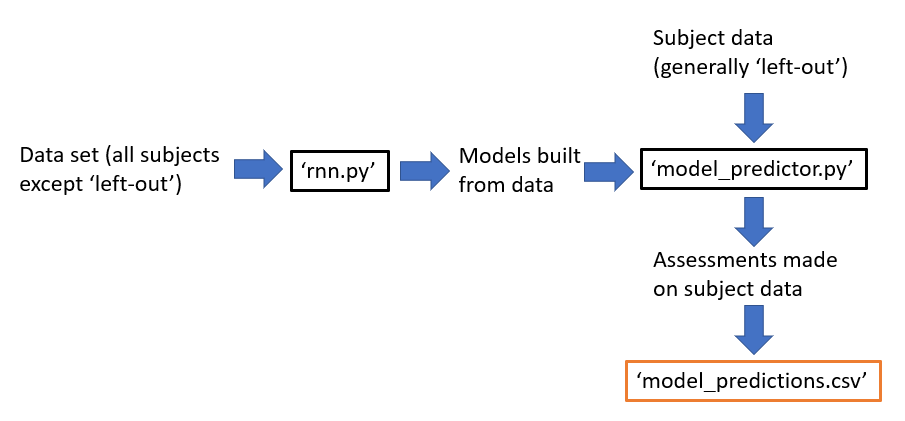
**Experiment Sets and Model Predictions Sets: Overviews and Differences**

Broadly speaking, there are two categories of experimentation that we carry out for the system. The first are what we call ‘experiment sets’. These are generally speaking carried out using only the ‘rnn.py’ script and are done to ascertain the optimal model setups to be used later on in ‘model predictions sets’, the second category of experimentation. Experiment sets include determining the best raw measurements to use to build a system, the number of features from computed statistical values to use, the sequence length and sequence overlap, among other things. All of the results we obtain from the experiment sets are obtained only by the use of ‘rnn.py’ and the results are obtained by the results of the testing portion of the input data set. Therefore, the results that we obtain here are from testing data from files that it has seen before but data from those files that it hasn’t. In other words, the testing data may reflect a more optimistic view of the real-world results on new subjects than would be the case in actuality, and therefore this is the cause of disparity between performance in many experiment sets and the results from many model predictions sets. However, even though the results might be less accurate than on left-out subjects, the experiment sets where we see improvements in the models by changing some aspect of the models’ setup can be regarded to also similarly work on left-out subjects.



The above diagram shows the relationship between the data that models are trained on, the script that builds the models, the final assessments made, and where we store the results.

The second category of experimentation are ‘model predictions sets’. Unlike the experiment sets, these generally involve the use of two scripts: ‘rnn.py’ and ‘model\_predictory.py’. Furthermore, unlike examining the results from the testing data from built models, we instead use a complete file from a subject left out of the data and do assessments on this subject using ‘model\_predictor.py’. These also make use of pre-built models, so many model predictions sets involve both the building of models with different settings set and also the loading of these recently-built models to assess a left-out subject on. This model predictions sets include testing the performance of different models built on different measurements, the impact of introducing noise to the data set in an effort to help generalisation, the leaving-out of anomalies from the data set, and so on. As these model prediction sets generally involve one or more subjects that have been left out of training completely, the results we obtain here are generally more indicative of real-world assessments of new subjects when the project has been completed and if the system is being used as an assessment tool.



The above diagram shows the relationship between data sets, the scripts utilized, and where the results of model predictions sets are stored.

**Experimentation Replication: How to Carry Out the Experiment Sets and Model Prediction Sets Outlined Below**

Prior to outlining all parts of the experimentation undertaken for this project, it’s preferable to give any prospective user the requisite steps needed to replicate the experimentation as best they can. This could be to validate the results that we claim to have achieved here, or to instead carry out experimentation of their own and would thus benefit a brief walk-through of the steps necessary to do so. Before we do any of this, however, we should note a few prerequisite steps that must be undertaken if they haven’t already been by the user. This includes downloading the project directory, all parts of the local directory, setting up Python and PyCharm, the running of the ‘setup.cmd’ batch script, and enabling TensorFlow to use any available GPU. Further details of the steps necessary can be found within the ‘System Setup’ of this report.

Once all the necessary setup steps have been done and with the project and local directories in place with the data sent through the data pipeline via the use of ‘setup.cmd’, we are ready to use the ‘rnn.py’ and ‘model\_predictor.py’ scripts in order to undertake experimentation.

To replicate any of the experiment sets below, one must do the following:

1. For a given experiment set, locate the necessary experiments within ‘RNN Results.xlsx’ that were used as part of the experiment set. For example, for experiment set 3 outlined below, we can see that experiments 13 to 39 were used for this set.
2. Locate the necessary experiments within ‘RNN Results.xlsx’ (located at **<project directory>\documentation \RNN Results.xlsx**). In the above case, one would find the rows within the table that correspond to the column ‘Experiment Number’ ranging from 13 to 39. Each row represents a single experiment undertaken (e.g. a model built from a certain data set, with certain data preprocessing hyperparameters, etc.) and the scripts and argument combinations needed to run it in turn to obtain the results found in that row’s ‘Results’ column. Also note that we store the hyperparameter settings to the right of the ‘Results’ columns so any ensures can ensure that they have the same hyperparameter settings as the results obtained below.
3. For each of experiments in the relevant rows, ensure that each of the scripts and argument combinations are run. Note that they must be run in the order in which they are presented (i.e. the cell to the right of ‘Experiment Number’ must be run first, following by the next right cell, and so on). This is because many of the scripts rely on the results of the previous one in order to work (e.g. ‘comp\_stat\_vals.py’ must be run before ‘ft\_sel\_red.py’). Also note that these are not required to be run before every experiment, but rather is a requirement to have been run beforehand. Hence, for experiments 13 to 39 we only need to run ‘python ext\_raw\_measures.py NSAA all all’ once, and not once per row (as this command just produces the raw measurement data and, while it is required to have been run for these experiments, it doesn’t need to be repeatedly called). However, for several of the other commands (e.g. ones concerning ‘rnn.py’), these must be run each time as they will very often differ at least slightly from the other experiments in the set (these differences are often what we are testing).
4. Once the user has ensured the requisite setup script combinations have been run followed by the ‘rnn.py’ variation we need, the user will be presented with a console output. Ensuring that the user has the hyperparameters setup that matches the hyperparameter settings outlined in the rightmost columns, the user can then directly compare the results for the model in question to the relevant row that has just been run and, specifically, compare the console outputs to the output displayed in the ‘Results’ column. Alternatively, if one wishes to instead carry out their own experiment results, simply disregard the final step about comparing to the previously obtained results for a given experiment and instead record the results of the model, the hyperparameter settings, etc., to the end of the table, with an adequate explanation of the purpose of the experiment in the ‘Description’ column.

To replicate any of the model predictions sets below, one must do the following:

1. As all model predictions sets have their ‘rnn.py’ and ‘model\_predictor.py’ steps all automated via batch scripts then, assuming the user has successfully run the entirety of the ‘setup.cmd’ script, one must simply launch and run the relevant batch script for the set. For example, if the user wishes to replicate model predictions set 3, one would simply run ‘model\_predictions\_set\_3.cmd’ which would run the required ‘rnn.py’ and ‘model\_predictor.py’ variants to setup the models and test them with complete files.
2. The results obtained by the running of the ‘model\_predictor.py’ scripts via the batch script will be outputted as rows to ‘model\_predictions.csv’, with one row per subject prediction; these rows are written at the end of the table. To establish which of these final rows we are concerned with, note how many times the batch script has called ‘model\_predictor.py’: this will correspond to the number of rows at the end of the table we are concerned with for this specific model predictions set.
3. Locate the previous rows within the table that signify the previous run of this experiment sets, the results of which we will discuss below. The user can do this by noting that the values in the ‘Short file name’ for the given rows will match those we just now obtained. Once the rows corresponding to the previously run iteration of the model predictions set has been found, we can directly compare the values held in each of the metrics columns (i.e. that include all columns except the first five). Doing so allows one to validate the results previously obtained and as discussed in detail below.

**Experiment Sets Table of Results (Drawn from ‘RNN Results.xlsx’)**

Note that the experiment number in the table below corresponds to the same experiment number in ‘RNN Results.xlsx’. Hence, for more information about a single row in the table below (e.g. RNN model parameters used, the scripts and exact arguments used to obtain the results, etc.), please see the row in ‘RNN Results’ corresponding to the same experiment number. Note that ‘Data Shape’ is the total shape of the ‘x’ data that is used to train the model and includes both the training and testing components. Also note the different parts of the shape: the first is the number of samples, the second number is the sequence length, and the third is the number of features of each sequence.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exper Number** | **Source Dir** | **Measurement** | **Output Type** | **Subject List** | **Data Shape** | **Results** |
| 1 | NSAA | Extracted statistical values | D/HC classify | 1 | (742, 10, 30) | Test Accuracy = 92.97% |
| 2 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (742, 10, 30) | MSE = 28.71, MAE = 2.90 |
| 3 | 6minutewalk-matfiles | Joint angles (from data cube) | D/HC classify | 2 | (8470, 60, 66) | Test Accuracy = 99.88% |
| 4 | 6minutewalk-matfiles | Joint angles (from data cube) | Overall NSAA regress | 2 | (8470, 60, 66) | MSE = 0.476, MAE = 0.403 |
| 5 | 6minutewalk-matfiles | Joint angles (from joint angle files) | D/HC classify | 3 | (2143, 60, 66) | Test Accuracy = 100.0% |
| 6 | 6minutewalk-matfiles | Joint angles (from joint angle files) | Overall NSAA regress | 3 | (2143, 60, 66) | MSE = 0.167, MAE = 0.292 |
| 7 | 6minutewalk-matfiles | Extracted statistical values | D/HC classify | 4 | (552, 10, 30) | Test Accuracy = 82.81% |
| 8 | 6minutewalk-matfiles | Extracted statistical values | Overall NSAA regress | 4 | (552, 10, 30) | MSE = 29.41, MAE = 3.56 |
| 9 | NSAA | Extracted statistical values | Single act classify | 1 | (742, 10, 30) | Ind Act Acc = 92.92%, All Act Acc = 79.69% |
| 10 | 6minutewalk-matfiles | Joint angles (from data cube) | Single act classify | 2 | (8470, 60, 66) | Ind Act Acc = 99.77%, All Act Acc = 98.5% |
| 11 | 6minutewalk-matfiles | Joint angles (from joint angle files) | Single act classify | 3 | (2143, 60, 66) | Ind Act Acc = 99.97%, All Act Acc = 99.74% |
| 12 | 6minutewalk-matfiles | Extracted statistical values | Single act classify | 4 | (552, 10, 30) | Ind Act Acc = 85.48%, All Act Acc = 73.44% |
| 13 | NSAA | Position | D/HC classify | 1 | (9379, 60, 69) | Test Accuracy = 95.04% |
| 14 | NSAA | Position | Overall NSAA regress | 1 | (8034, 60, 69) | MSE = 13.15, MAE = 2.22 |
| 15 | NSAA | Position | Single act classify | 1 | (8034, 60, 69) | Ind Act Acc = 94.4%, All Act Acc = 82.94% |
| 16 | NSAA | Velocity | D/HC classify | 1 | (9379, 60, 69) | Test Accuracy = 69.94% |
| 17 | NSAA | Velocity | Overall NSAA regress | 1 | (8034, 60, 69) | MSE = 38.11, MAE = 4.28, RMSE = 6.17, R^2 = -0.0004 |
| 18 | NSAA | Velocity | Single act classify | 1 | (8034, 60, 69) | Ind Act Acc = 71.51%, All Act Acc = 4.75% |
| 19 | NSAA | Acceleration | D/HC classify | 1 | (9379, 60, 69) | Test Accuracy = 77.48% |
| 20 | NSAA | Acceleration | Overall NSAA regress | 1 | (8034, 60, 69) | MSE = 48.36, MAE = 4.62, RMSE = 6.95, R^2 = -0.4 |
| 21 | NSAA | Acceleration | Single act classify | 1 | (8034, 60, 69) | Ind Act Acc = 74.98%, All Act Acc = 17.25% |
| 22 | NSAA | Angular Velocity | D/HC classify | 1 | (9379, 60, 69) | Test Accuracy = 71.01% |
| 23 | NSAA | Angular Velocity | Overall NSAA regress | 1 | (8034, 60, 69) | MSE = 37.64, MAE = 4.27, RMSE = 6.13, R^2 = -0.002 |
| 24 | NSAA | Angular Velocity | Single act classify | 1 | (8034, 60, 69) | Ind Act Acc = 72.82%, All Act Acc = 19.12% |
| 25 | NSAA | Angular Acceleration | D/HC classify | 1 | X shape = (9379, 60, 69) | Test Accuracy = 71.17% |
| 26 | NSAA | Angular Acceleration | Overall NSAA regress | 1 | (8034, 60, 69) | MSE = 37.65, MAE= 4.42, RMSE = 6.14, R^2 = 0.03 |
| 27 | NSAA | Angular Acceleration | Single act classify | 1 | (8034, 60, 69) | Ind Act Acc = 73.07%, All Act Acc = 21.81% |
| 28 | NSAA | Sensor Free Acceleration | D/HC classify | 1 | X shape = (9379, 60, 51) | Test Accuracy = 75.59% |
| 29 | NSAA | Sensor Free Acceleration | Overall NSAA regress | 1 | (8034, 60, 51) | MSE = 40.10, MAE = 4.38, RMSE = 6.33, R^2 = 0.002 |
| 30 | NSAA | Sensor Free Acceleration | Single act classify | 1 | (8034, 60, 51) | Ind Act Acc = 74.87%, All Act Acc = 14.94% |
| 31 | NSAA | Sensor Magnetic Field | D/HC classify | 1 | (9379, 60, 51) | Test Accuracy = 99.57% |
| 32 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (8034, 60, 51) | MSE = 4.49, MAE = 1.35, RMSE = 2.12, R^2 = 0.87 |
| 33 | NSAA | Sensor Magnetic Field | Single act classify | 1 | (8034, 60, 51) | Ind Act Acc = 99.32%, All Act Acc = 96.5% |
| 34 | NSAA | Joint Angle | D/HC classify | 1 | (9379, 60, 66) | Test Accuracy = 98.11% |
| 35 | NSAA | Joint Angle | Overall NSAA regress | 1 | (8034, 60, 66) | MSE = 3.95, MAE= 1.23, RMSE = 1.99, R^2 = 0.9 |
| 36 | NSAA | Joint Angle | Single act classify | 1 | (8034, 60, 66) | Ind Act Acc = 98.57%, All Act Acc = 93.38% |
| 37 | NSAA | Joint Angle XZY | D/HC classify | 1 | (9379, 60, 66) | Test Accuracy = 96.93% |
| 38 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (8034, 60, 66) | MSE = 4.98, MAE = 1.36, RMSE = 2.23, R^2 = 0.88 |
| 39 | NSAA | Joint Angle XZY | Single act classify | 1 | (8034, 60, 66) | Ind Act Acc = 96.98%, All Act Acc = 89% |
| 40 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (742, 10, 30) | MSE = 28.71, MAE = 2.9 |
| 41 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (920, 10, 30) | MSE = 28.34, MAE = 2.26, RMSE = 5.32, R^2 = 0.39 |
| 42 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (1227, 10, 30) | MSE = 13.22, MAE = 1.56, RMSE = 3.64, R^2 = 0.69 |
| 43 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (1840, 10, 30) | MSE = 9.06, MAE = 1.13, RMSE = 3.01, R^2 = 0.79 |
| 44 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (3710, 10, 30) | MSE = 1.19, MAE = 0.52, RMSE = 1.09, R^2 = 0.97 |
| 45 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (7420, 10, 30) | MSE = 0.03, MAE = 0.16, RMSE = 0.18, R^2 = 0.99 |
| 46 | NSAA | Position | Overall NSAA regress | 1 | (1013, 60, 69) | MSE = 0.13, MAE = 0.18, RMSE = 0.37, R^2 = 0.52 |
| 47 | NSAA | Velocity | Overall NSAA regress | 1 | (1013, 60, 69) | MSE = 0.45, MAE = 0.44, RMSE = 0.67, R^2 = -0.38 |
| 48 | NSAA | Acceleration | Overall NSAA regress | 1 | (1013, 60, 69) | MSE = 0.38, MAE = 0.42, RMSE = 0.62, R^2 = -0.43 |
| 49 | NSAA | Angular Velocity | Overall NSAA regress | 1 | (1013, 60, 69) | MSE = 0.38, MAE = 0.38, RMSE = 0.61, R^2 = -0.24 |
| 50 | NSAA | Angular Acceleration | Overall NSAA regress | 1 | (1013, 60, 69) | MSE = 0.27, MAE = 0.34, RMSE = 0.52, R^2 = 0.03 |
| 51 | NSAA | Sensor Free Acceleration | Overall NSAA regress | 1 | (1013, 60, 51) | MSE = 0.33, MAE = 0.4, RMSE = 0.57, R^2 = 0.04 |
| 52 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (1013, 60, 51) | MSE = 0.13, MAE = 0.17, RMSE = 0.37, R^2 = 0.54 |
| 53 | NSAA | Joint Angle | Overall NSAA regress | 1 | (1013, 60, 66) | MSE = 0.09, MAE = 0.12, RMSE = 0.3, R^2 = 0.65 |
| 54 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (1013, 60, 66) | MSE = 0.14, MAE = 0.18, RMSE = 0.37, R^2 = 0.55 |
| 55 | NSAA | Position | Overall NSAA regress | 1 | (13365, 60, 69) | MSE = 16.27, MAE = 2.76, RMSE = 4.03, R^2 = 0.64 |
| 56 | NSAA | Position | Overall NSAA regress | 1 | (13314, 90, 69) | MSE = 17.51, MAE = 2.6, RMSE = 4.18, R^2 = 0.6 |
| 57 | NSAA | Position | Overall NSAA regress | 1 | (13315, 120, 69) | MSE = 17.89, MAE = 2.81, RMSE = 4.23, R^2 = 0.59 |
| 58 | NSAA | Position | Overall NSAA regress | 1 | (13313, 180, 69) | MSE = 15.12, MAE = 2.44, RMSE = 3.89, R^2 = 0.67 |
| 59 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13365, 60, 66) | MSE = 4.46, MAE = 1.24, RMSE = 2.11, R^2 = 0.9 |
| 60 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13314, 90, 66) | MSE = 4.99, MAE = 1.27, RMSE = 2.23, R^2 = 0.89 |
| 61 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13315, 120, 66) | MSE = 4.21, MAE = 1.16, RMSE = 2.05, R^2 = 0.91 |
| 62 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13313, 180, 66) | MSE = 2.65, MAE = 0.99, RMSE = 1.63, R^2 = 0.94 |
| 63 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (13365, 60, 51) | MSE = 3.05, MAE = 1.13, RMSE = 1.75, R^2 = 0.93 |
| 64 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (13314, 90, 51) | MSE = 2.47, MAE = 0.89, RMSE = 1.57, R^2 = 0.94 |
| 65 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (13315, 120, 51) | MSE = 2.31, MAE = 0.84, RMSE = 1.52, R^2 = 0.95 |
| 66 | NSAA | Sensor Magnetic Field | Overall NSAA regress | 1 | (13313, 180, 51) | MSE = 7.07, MAE = 1.76, RMSE = 2.66, R^2 = 0.84 |
| 67 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (13365, 60, 66) | MSE = 5.99, MAE = 1.44, RMSE = 2.45, R^2 = 0.86 |
| 68 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (13314, 90, 66) | MSE = 4.29, MAE = 1.28, RMSE = 2.07, R^2 = 0.9 |
| 69 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (13315, 120, 66) | MSE = 4.72, MAE = 1.24, RMSE = 2.17, R^2 = 0.9 |
| 70 | NSAA | Joint Angle XZY | Overall NSAA regress | 1 | (13313, 180, 66) | MSE = 2.62, MAE = 1.01, RMSE = 1.62, R^2 = 0.94 |
| 71 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 10) | MSE = 0.03, MAE = 0.12, RMSE = 0.17, R^2 = 0.99 |
| 72 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 20) | MSE = 0.02, MAE = 0.09, RMSE = 0.14, R^2 = 0.99 |
| 73 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 30) | MSE = 0.01, MAE = 0.07, RMSE = 0.1, R^2 = 0.99 |
| 74 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 40) | MSE = 0.01, MAE = 0.09, RMSE= 0.12, R^2 = 0.99 |
| 75 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 50) | MSE = 0.02, MAE = 0.09, RMSE = 0.13, R^2 = 0.99 |
| 76 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 30) | MSE = 0.01, MAE = 0.07, RMSE = 0.1, R^2 = 0.99 |
| 77 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (6550, 20, 30) | MSE = 1.13, MAE = 0.5, RMSE = 1.06, R^2 = 0.98 |
| 78 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (3185, 40, 30) | MSE = 1.05, MAE = 0.67, RMSE = 1.02, R^2 = 0.98 |
| 79 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 30) | MSE = 0.01, MAE = 0.07, RMSE = 0.1, R^2 Score = 0.99 |
| 80 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (12794, 20, 30) | MSE = 0.26, MAE = 0.26, RMSE = 0.51, R^2 = 0.99 |
| 81 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (12195, 40, 30) | MSE = 0.53, MAE = 0.42, RMSE = 0.73, R^2 = 0.99 |
| 82 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13240, 10, 30) | MSE = 0.01, MAE = 0.07, RMSE = 0.1, R^2 = 0.99 |
| 83 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13265, 7, 30) | MSE = 0.05, MAE = 0.15, RME = 0.21, R^2 = 0.99 |
| 84 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13305, 5, 30) | MSE = 0.11, MAE = 0.19, RMSE = 0.33, R^2 = 0.99 |
| 85 | NSAA | Extracted statistical values | Overall NSAA regress | 1 | (13453, 3, 30) | MSE = 0.12, MAE = 0.19, RMSE = 0.35, R^2 = 0.99 |
| 86 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13365, 60, 66) | MSE = 4.46, MAE = 1.24, RMSE = 2.11, R^2 = 0.9 |
| 87 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13314, 90, 66) | MSE = 4.99, MAE = 1.27, RMSE = 2.23, R^2 = 0.89 |
| 88 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13315, 120, 66) | MSE = 4.21, MAE = 1.16, RMSE = 2.05, R^2 = 0.91 |
| 89 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13313, 180, 66) | MSE = 2.65, MAE = 0.99, RMSE = 1.63, R^2 = 0.94 |
| 90 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13314, 60, 66) | MSE = 3.36, MAE = 1.21, RMSE = 1.83, R^2 = 0.93 |
| 91 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13315, 60, 66) | MSE = 3.43, MAE = 1.13, RMSE = 1.85, R^2 = 0.93 |
| 92 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13313, 60, 66) | MSE = 3.13, MAE = 1.07, RMSE = 1.77, R^2 = 0.93 |
| 93 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13240, 60, 66) | MSE = 2.4, MAE = 0.88, RMSE = 1.55, R^2 = 0.95 |
| 94 | NSAA | Joint Angle | Overall NSAA regress | 1 | (13051, 60, 66) | MSE = 1.06, MAE = 0.6, RMSE = 1.03, R^2 = 0.98 |
| 95 | NSAA | Joint Angle | Overall NSAA regress | 1 | (12436, 60, 66) | MSE = 1.48, MAE = 0.66, RMSE = 1.22, R^2 = 0.97 |
| 96 | NSAA | Joint Angle | Overall NSAA regress | 1 | (11530, 60, 66) | MSE = 0.35, MAE = 0.41, RMSE = 0.59, R^2 = 0.99 |
| 97 | NSAA | Joint Angle | Overall NSAA regress | 1 | (10130, 60, 66) | MSE = 0.35, MAE = 0.46, RMSE = 0.59, R^2 = 0.99 |
| 98 | NSAA | Joint Angle | Overall NSAA regress | 1 | (7481, 60, 66) | MSE = 0.73, MAE = 0.7, RMSE = 0.86, R^2 = 0.98 |

= Experiment Set 1 = Experiment Set 6

= Experiment Set 2 = Experiment Set 7

= Experiment Set 3 = Experiment Set 8

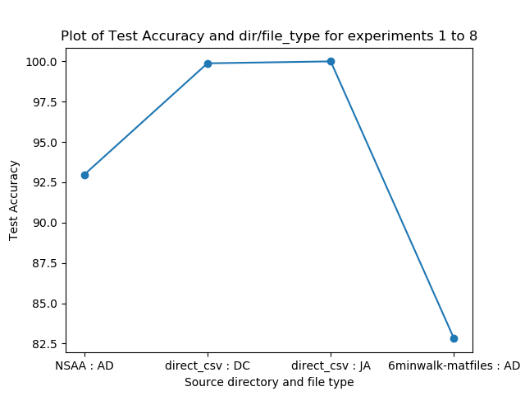
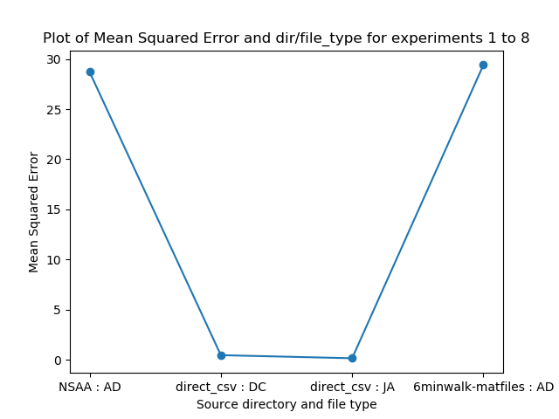
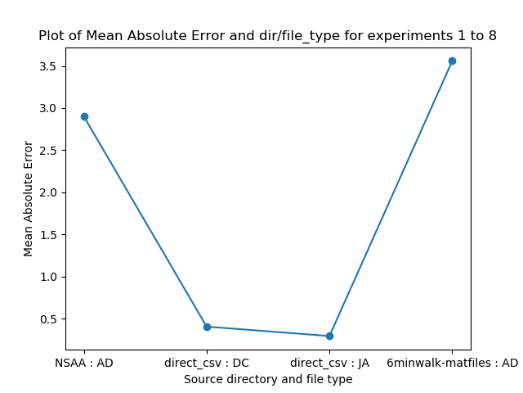
= Experiment Set 4 = Experiment Set 9

= Experiment Set 5 = Experiment Set 10

**Subjects Lists**

|  |  |
| --- | --- |
| **Subject List 1** | D2, D3, D4, D4V2, D5, D6, D7, D9, D10, D11, D12, D14, D15, D16, D17, D18, D19, HC3, HC4, HC5, HC6, HC7, HC8, HC9, HC10 |
| **Subject List 2** | D2, D3, D4, D5, D6, D7, D9, D10, D11, D12, D14, D15, D17, D18, D19, HC1, HC2, HC3, HC4, HC5, HC6, HC7, HC8, HC9, HC10 |
| **Subject List 3** | D2, D3, D4, HC2, HC5, HC6 |
| **Subject List 4** | D2, D3, D4, D5, D6, D7, HC1, HC2, HC3, HC4, HC5, HC7, HC8, HC9, HC10 |

**Experiment Set 1: Performance of RNNs on Different Source Data**

The purpose of this experiment set is to determine whether and how well RNN models regresses on different types of source data. By this, we mean data that is in the same format (i.e. ‘.mat’ files in the same organizational structure) but representing different measurements from different source directories. These are thus:

* **NSAA : AD** – statistical values extracted from the ‘all-data’ (‘AD’) files by the ‘comp\_stat\_vals.py’ script, whose files are sourced from the ‘NSAA’ directory; hence these are stat values of the subjects performing the NSAA activities, which are then used to train model(s).
* **direct\_csv : DC** – these are the joint angle values (i.e. raw measurements, not computed stat values) that are sourced from the data cube; this data cube contains the 6-minute walk data from various subjects, several of which aren’t included in the standard joint angle files of subjects doing the 6-minute walk.
* **direct\_csv : JA** – similar to ‘direct\_csv : DC’ as described above, using the same source directory type and raw measurement, but containing joint angle files that aren’t necessarily included within the data cube.
* **6minwalk-matfiles : AD** – again, uses the files corresponding to subjects’ 6-minute walk assessments, with the difference this time being that we aren’t using raw joint angle files either in the data cube format or as ‘loose’ files, but rather computing statistical values via ‘comp\_stat\_vals.py’ script; in this sense, it’s the same as ‘NSAA : AD’ models but using different assessment data (6-minute walk rather than NSAA).

For the second and third diagrams above, with regards to the output type, all these file types and directory sources outlined above are used to train models trained to regress on the overall NSAA score for a given sequence from a file. This value is able to range between 0 and 34 (though based on how the assessment is done, it generally ranges between 15 and 34). Hence, for a given type of source data, if it has a MAE = 0.5, that means that, on test data of sequences from files of the given type of source data, the model predicts for each of them a score of between 0 and 34 on average ‘0.5’ away from the true value for that sequence (the true value for a sequence being the overall NSAA score of the file that the sequence comes from). However, for the first diagram above, it tests the different files types on its ability to classify whether a sequence that come from a file are from a ‘D’ or ‘HC’ subject.

**Results Discussion and Conclusions**

In using just the raw joint angle values from ‘DC’ or ‘JA’ files, we achieve an approximate **99%** accuracy (i.e. for each sequence of 60 rows of 66 joint angle values, the model can predict with 99% accuracy whether it came from a ‘D’ file or an ‘HC’ file); however, looking at more measurements (e.g. position, accelerometer values, etc.), performing manual feature extraction via computing of the statistical values and then reducing the dimensionality, and then training the model provides a much worse classification accuracy of **82.81%** for data that comes from the same assessment (6-minute walk). This can also be seen when the same data sources are then used to train the RNN to perform regression for the overall NSAA score: the raw joint angle data gives a much smaller **MAE = 0.4037** (meaning that it predicts a score of between 0 and 34 which on average is 0.4037 away from the true value in either direction), compared with a much worse **MAE = 3.56** from 6-minute walk ‘AD’ statistical value files. A further observation can be made about the experiments concerning the raw joint angle files in that they were performing much better than we were expecting them to be: by simply considering only the joint angle measurement of a subjects suit data, given 1 second’s worth of an input sequence to the RNN, it can correctly classify whether the frame comes from a healthy control subject or one with DMD to a very high accuracy of 99% and predict the overall NSAA score to within 0.4037 of the true value of between 0 and 34. This is extraordinarily high, much better than the ability of medical professionals and, most notably, this is only the first iteration of the experiments with raw measurement files.

This large difference between raw measurements and computed statistical features may seem counterintuitive at first glance, as the former is just using data direct from the provided ‘.mat’ files, while with the latter we actually process it further to ideally extract more important features from the data. One possible reason for raw joint angle models performing so well could be that there aren’t that many subjects with these files that are provided to us in comparison with ‘AD’ files, so not as diverse a training set is used; this means that there would be a narrower range of overall NSAA scores to regress towards, making it easier for the model to approximate the true score with a smaller margin of error. Another possible reason could be that the features that the RNN models extracts from its inputted data are more useful for it to train on and approximate an overall score: a prominent benefit of using neural networks is that they are traditionally noted to perform better with raw data rather than manually extracted features. Furthermore, when it trains on the stat value data from ‘comp\_stat\_values.py’ for the ‘AD’ file types, it still computes its own features within the RNN, and so this ‘features from features’ behaviour might have proved to be problematic for the network.

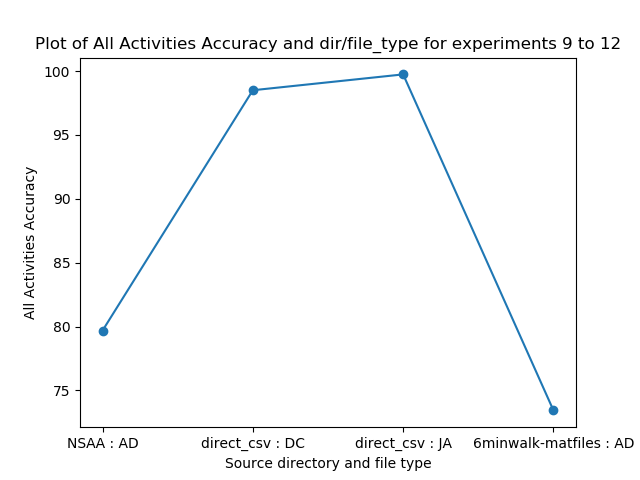
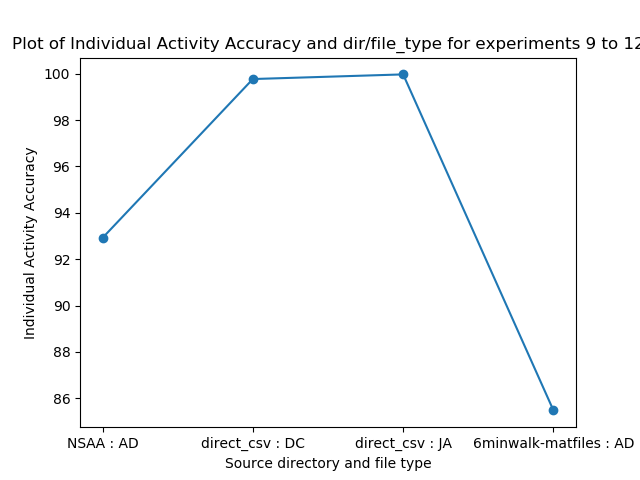
It should also be noted that training the RNN with raw joint angle data requires far fewer training epochs (~20) than for statistical values (~100). This will primarily be due to the larger amount of raw joint angle data that is fed through the network; in the case of training on all files within the data cube, the ‘x’ input shape is (8470, 60, 66) while for the corresponding AD files we only have (552, 10, 30) samples due to how computing statistical values dramatically reduces the raw amount of available data. This decrease in amount of available data also might help to account for the reduction in accuracy when using data sourced from AD files.

Overall, given our limited data and without any way to currently increase it, we come to several conclusions:

1. **RNN models can successfully regress and classify on data 6-minute walk and NSAA assessment data**
2. **Not only this, but raw joint angles perform phenomenally well in both classification and regression**
3. **The raw data measurements might be better put through RNNs than extracted statistical features**
4. **This is probably due to extracted statistical features being a lot smaller of a data set, RNNs better utilizing raw measurements over pre-computed features, and there being a small range of target values with files sourced as ‘JA’ or ‘DC’**

We shall shortly be examining ways to improve the performance of models trained on computed statistical values by increasing sequence overlap, modifying sequence length, and so on.

**Experiment Set 2: Performance of RNNs for Single Activity Classification**



This experiment utilizes the same source data types as experiment set 1. Hence, we won’t discuss what each of these data types represent in the x-axes of the graphs above, as these are exactly the same files as used previously. The difference with this experiment set, however, is that it is looking at a different output type: while experiment set 1 looked at performance for classification of D/HC labels and overall NSAA scores for given test sequences, this experiment set looks at predicting multiple classification values for a single sequence. Here, the models are trained to predict a sequence of 17 values of numbers of either 0, 1, or 2. These represent the single activity scores for a single sequence that correspond to the single activity scores of the file that the sequence is sourced from. It’s worth noting that, as overall NSAA score is the sum of these, a sequence will have an overall NSAA score that is equal to the sum of the same sequence’s 17 single act scores. Furthermore, when predicting single act scores, the RNN architecture will be different: for D/HC classification or overall NSAA score regression, there will be only 1 output neuron (though predicting different ranges of values for the two tasks), while for single acts there are 17 output neurons, 1 for each single act it is predicting. The overall aim of this experiment is thus to see if a model is able to predict, given a sequence of values from a file (that corresponds to a subject, e.g. ‘D4’), what that file’s single-act scores are. Note that the two metrics that we are using (that are computed by each RNN model over all its testing data) is individual activity accuracy and all activities accuracy: the former is the percentage of activities in the testing data set that were correctly predicted to be a 0, 1, or 2, while the latter is the percentage of whole activity sets (i.e. single RNN output of 17 values for a single test sequence) that were correctly predicted.

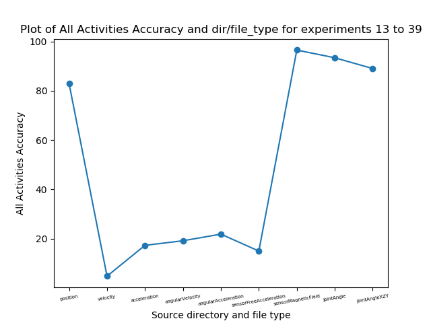
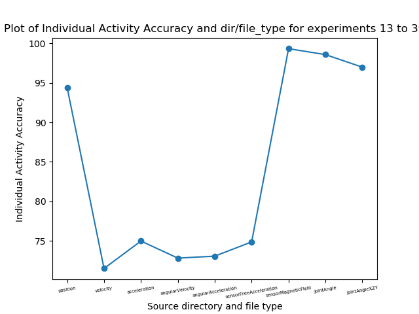
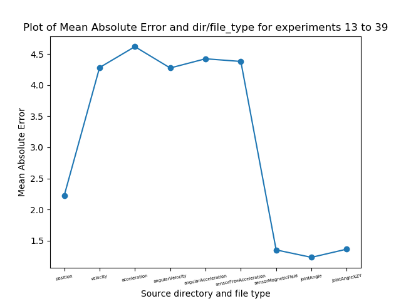
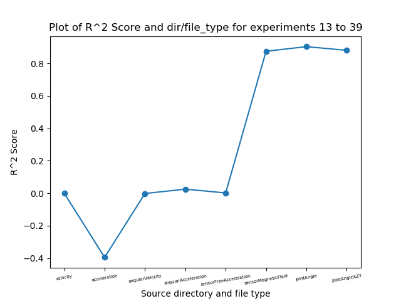
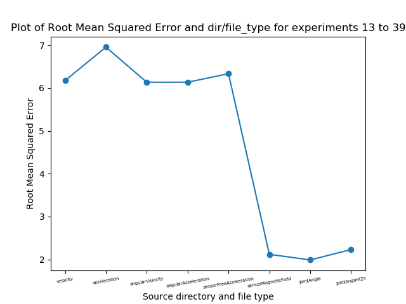
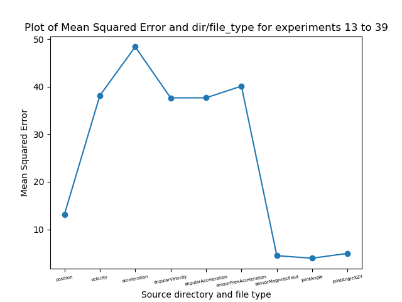
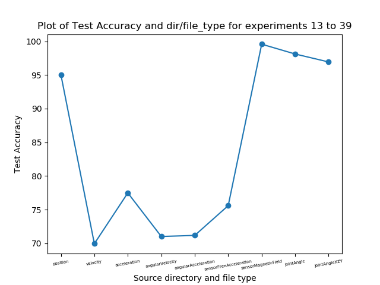
**Results Discussion and Conclusions**

In line with the results from experiment set 1, we can see here that raw joint angle data is much more useful in predicting the single act scores for test data of sequences than stat values from ‘AD’ files. This is most likely down to the same reasons as for the other output types; namely, comparatively small about of ‘AD’ data, neural networks better utilizing raw measurements, etc. Given that it is consistent with the previous experiment set results in determining the comparative performance for different source data types, the results of the experiment set is therefore more about whether it is possible for these data types to train a RNN to predict sequences of values that correspond to activities that the sequence being tested might not be from (i.e. the sequence will be from part of a file that, at most, corresponds to one activity the subject is performing, and therefore can only try to assess what its likely other single activity scores are). The result is that it performs well, especially with joint angle data, predicting approximately **99%** of the individual activity scores and **98%** of the all activity sequence scores.

From this, we can draw several conclusions:

1. **The different types of source data can be used to train a model to accurately predict single act values**
2. **Like the previous two output types, raw joint angle measurements do this better than ‘AD’ stat values**

**Experiment Set 3: Raw Measurements for All Output Types**



With this experiment set, we now turn our attention to just looking exclusively at raw measurements. This is primarily due to the much better performance of models trained on raw measurements compared with ones trained on stat values from ‘AD’ files, as seen in experiment sets 1 and 2. The question we thus ask is: can we show similarly high performance when we train models on types of raw measurements from the suit data other than just joint angles? To this end, we look at 9 raw measurements in total that are contained within the ‘AD’ files and that are recorded by the suit during use: position, velocity, acceleration, angularAcceleration, angularVelocity, sensorFreeAcceleration, sensorMagneticField, jointAngle, and jointAngleXZY. Unlike the joint angle measurements, we don’t have the other raw measurements in separate, unique files. Therefore, we make extensive use of the ‘ext\_raw\_measures.py’ script in order to extract, for every file and for every raw measurement, the measurement data and store them in separate files as ‘.csv’ files. From here, these are then able to train and test an RNN model in the same way.

For this experiment set, we are exclusively concerning ourselves now with NSAA assessment files rather than 6-minute walk files. This is more a choice based on the intended direction of the project to be more concerned with assessing and making predictions concerning the NSAA assessments and how that is connected to natural movement data (more on this later) than the 6-minute walk data. Thus, each of the entries along the x-axis of the graphs represents a single model trained on that particular raw measurement data for every NSAA file we have available. Currently, we are just looking at how the models performs on testing data from subjects it has already seen before, rather than complete subjects being left out of the training set, though this shall be explored later on with ‘left out’ subjects in trained models. Furthermore, though each of the models are trained with the same number of files and each with the same sequence length (due to there at every time step the suit records all raw measurements), the feature size will vary; in other words, if the training shape to the RNN models are of shape (x, y, z), ‘x’ and ‘y’ will always be the same between raw measurements but ‘z’ will vary: for raw measurements based on segment measurements of the suit ‘z’ will be 69, while for angle-based measurements it will be 66 and for sensor readings it will be 51.

**Results Discussion and Conclusions**

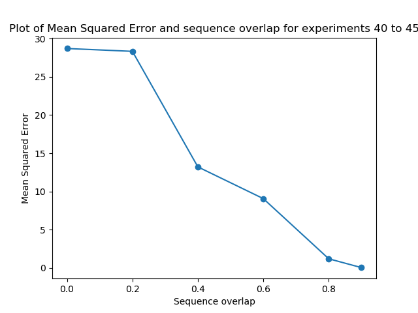
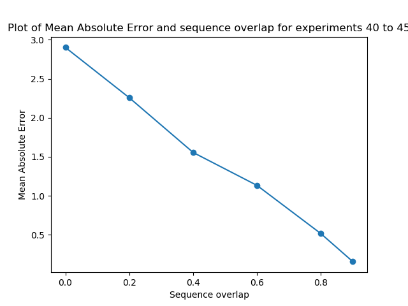
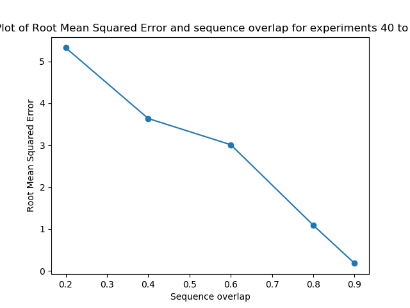
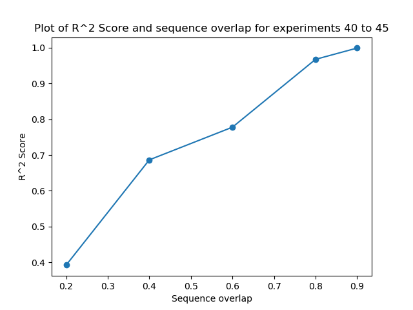
A consistent finding in all of the above graphs is that there are three raw measurements that are far and away better than the others: jointAngle, jointAngleXZY, and sensorMagneticField. These measurements can be seen to perform better for all three output types we are training towards: D/HC classification (graph 1), overall NSAA score (graphs 2 – 5), and individual activities classification(graphs 6 – 7). Both jointAngle and jointAngleXZY raw measurements were to be expected to perform this well, as this is consistent with results obtained in experiment sets 1 and 2 (note that this isn’t using exactly the same files, as experiment sets 1 and 2 use joint angle exclusive files that were pre-extracted before this project’s inception, while the models trained on ‘jointAngle’ here were instead measurements extracted from the ‘AD’ files and represent somewhat different subjects than the ‘JA’ or ‘DC’ files).

Position is another measurement that performs particularly well compared with the other 5 measurements, though for some metrics (e.g. MAE for overall NSAA score) it performs noticeably worse than jointAngle, jointAngleXZY, and sensorMagneticField. We can observe that these 4 ‘useful’ measurements predict **~97%** of test sequences accurately for D/HC classification, compared with only **~73%** for the other 5. Furthermore, when we look at models trained for overall NSAA score regression, we note an average MAE of **~1.5** for the useful 4 measurements, while we see an average MAE of **~4.3** for the other 5; and for single-activity classification for individual activity accuracy, we see **~97%** for the useful 4 and **~74%** for the other 5. With respect to the performance of the useful 4 measurements, while we expect to see performance for jointAngle and jointAngleXZY, the usefulness of sensorMagneticField and position is slightly more surprising. For position, we can speculate this to be as a result of subjects with more severe Duchenne being more inclined to have their limbs and torso in positions which are very indicative of their condition, compared with the healthy-control patients. The sensorMagneticField measurement, however, is slightly more mystifying and does not have an obvious explanation at this point. What’s more, our initial speculation was that the velocity and acceleration measurements would be fairly useful to distinguish between D and HC subjects and, moreover, help predict their overall NSAA scores, as we believed that, as these are key measurements of a subject’s potential for movement, they would be useful for training an RNN model. We found these, however, to perform comparatively badly on test sequences, which leads us to believe that either velocities and acceleration of movement between subjects don’t vary that much or that they aren’t as indicative of movement ability as we thought.

Therefore, the conclusions we draw are as follows:

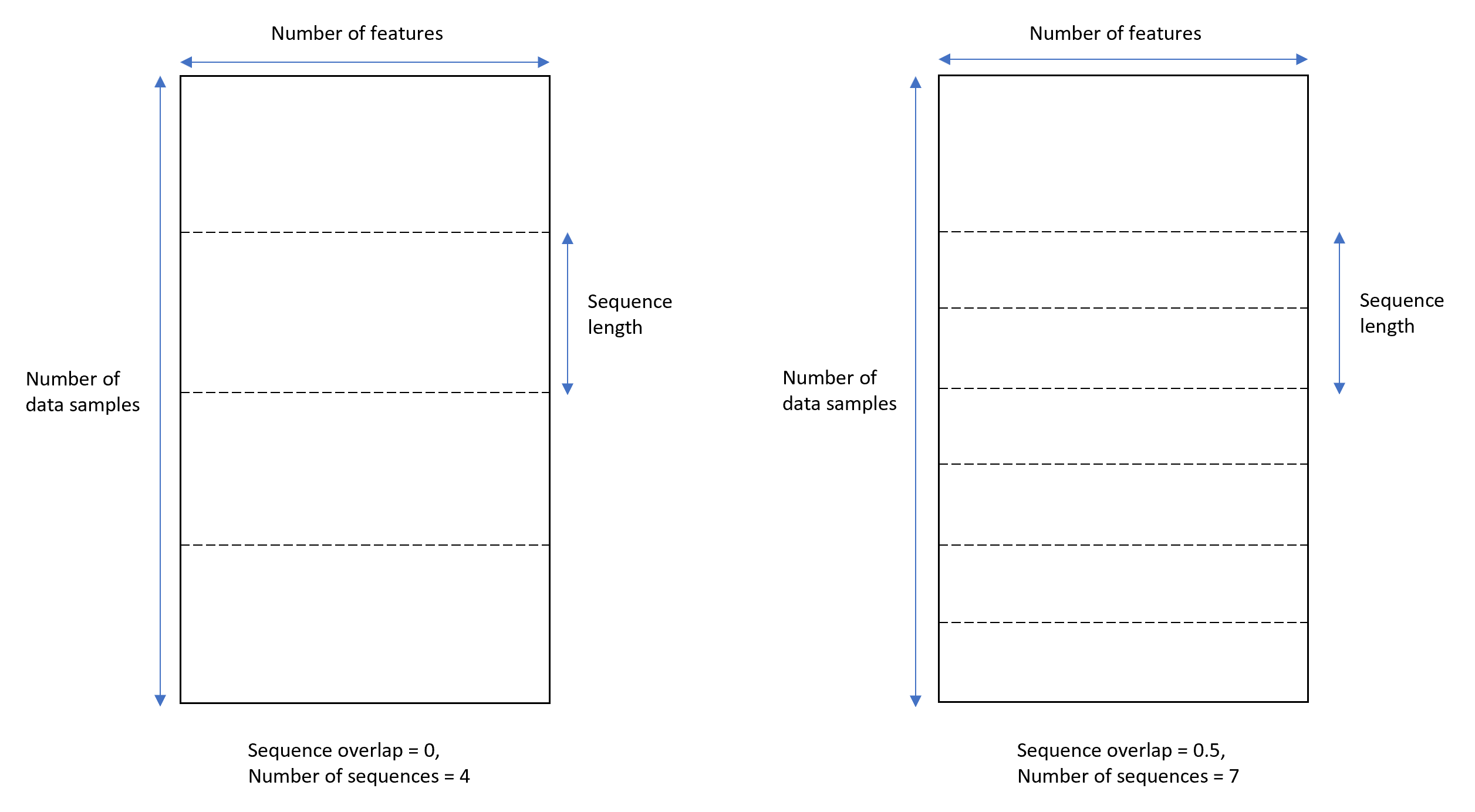
1. **jointAngle, jointAngleXZY, position, and sensorMagneticField are useful raw measurements for building RNN models on, while velocity, acceleration, angularVelocity, angularAcceleration, and sensorFreeeAcceleration are not**
2. **sensorMagneticField is surprisingly well performing (more or less just as good as the joint angle measurements), while position is still strong but slightly worse, and velocity and acceleration aren’t as useful as initially thought**

**Experiment Set 4: Sequence Overlap for Stat Values from AD Files**

One of the problems with stat values extracted from ‘AD’ files via the ‘comp\_stat\_vals.py’ script was that it dramatically reduced the amount of available data we have for training: when we are computing the stat values over intervals of 1 second (the standard measurement used for this project thus far), we are essentially doing calculations over 60 rows of raw measurement data (due to the sampling rate of the suit being 60Hz and we chose 1 second’s worth of data) and producing as an output 1 row of data. While this theoretically contains much of the useful information of the 60 rows simply condensed into 1 row, it doesn’t change the fact that we have now reduced the amount of raw data that we feed into the RNN 60-fold. This is most likely a large factor in the comparatively-weak results of stat values from ‘AD’ files seen in experiment sets 1 and 2. However, a way we chosen to get around this is by using a sequence overlap.

Sequence overlap is essentially used here as follows: consider a 2D block of data that we have available that is produced by the ‘comp\_stat\_vals.py’ script and ‘ft\_sel\_red.py’ to reduce to dimensionality of the data. This block has a number of rows equal to the number of produced rows of stat value data over all files that we are using (e.g. all NSAA files available in the directory) and a number of columns equal to the reduced dimensionality produced by ‘ft\_sel\_red.py’ (e.g. from ~4000 from ‘comp\_stat\_vals.py’ to ~30 to be read in by ‘rnn.py’). Normally, we will take ‘slices’ of this block to produce smaller blocks with a number of rows now equal to ‘sequence length’ before moving on to the next block below it until all data is consumed (ignoring leftover data at the end of the block that won’t fit into a smaller block). This is the case with the diagram below on the left.



However, if we consider a sequence overlap of 50% (i.e. overlap proportion of 0.5), we are instead able to capture 7 blocks of data rather than 4. Scaled up to smaller sequence lengths relative to the number of data samples, this is approximately 50% more data; if we scale up to an overlap of 0.9, we have 900% more data. Though we end up with a fair bit of redundancy with this approach (as the same vector of a data sample is used in numerous sequences), there are two primary benefits of doing this:

1. Much more available data, which is useful to train the models to become much more accurate.
2. With sequences that have no overlap, there is a high chance that one or more activities that occur in the NSAA assessment might be ‘cut’ along these lines (see the dashed lines in the above left image); this would mean the activity isn’t included in an entire sequence, while with a sufficient sequence length and sequence overlap, it’s more likely to be captured in its complete form in at least one of the sequences.

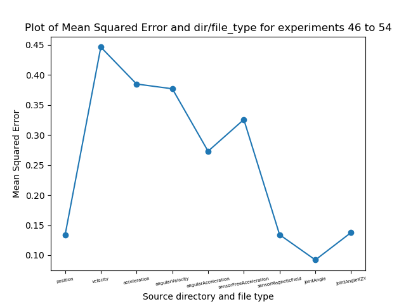
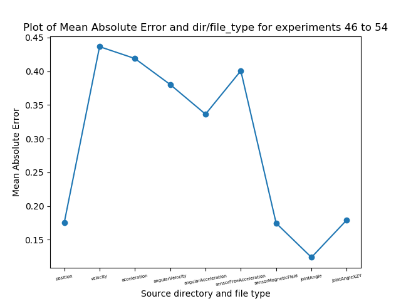
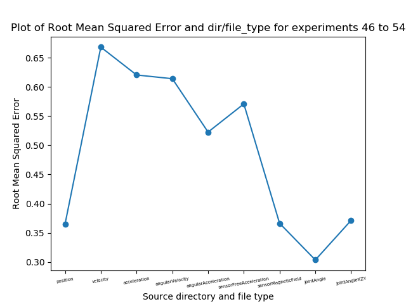
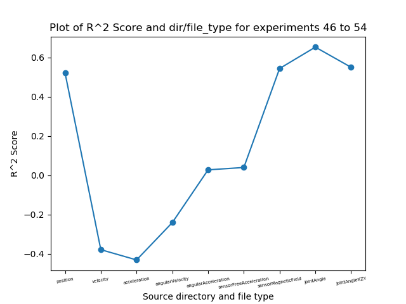
**Results Discussion and Conclusions**

Note that for this experiment set, we are only concerned with the overall NSAA score regression output type, as previous results show the performance of models on this output type are generally very consistent with results on other output types, hence there isn’t expected to be any need to repeat these experiments with the other output types here. Unsurprisingly, we see a massive increase in the performance with a larger sequence overlap. This will be in part due to the massive increase in available data: while the total data for NSAA files from stat values produced by ‘comp\_stat\_vals.py’ for a sequence over of 0 is (742, 10, 30) and shows a **MAE = ~2.9**, when the sequence overlap increases to 0.9, we have total data of shape (7420, 10, 30), ten times as many available sequences, which results in **MAE = ~0.1**. Note that these results are still just on test data, so it shows that the models can generalize much better to new, unseen sequences the overlap is much higher.

We can therefore conclude two things:

1. **A large sequence overlap leads to better performance for stat values from ‘AD’ files for NSAA files.**
2. **This is most likely due to the larger amount of available data and the increased window to capture complete activities.**

**Experiment Set 5: Sequence Overlap for Stat Values from AD Files**

This is the first experiment set that makes use of ‘mat\_act\_div.py’ to use the source ‘.mat’ files and the annotated Google sheet to experiment with what we call ‘single-act files’. These are files that contain only a single NSAA assessment within it and no other activities. For example, if we take the ‘D4’ source ‘.mat’ file, we then create 17 new files containing the 17 NSAA assessments, with each new file being a name like ‘D4\_act1.mat’, ‘D4\_act2.mat’, etc. This process obviously contains files that are a lot smaller than the source file, as well as cutting out a lot of the bits in between the assessments in the original file. For example, the original file will contain a complete recording of the suit data for a subject; this includes the data where a subject isn’t doing anything particularly relevant (e.g. standing around before beginning the next activity of the assessment or trying but failing to do one of the assessments). For more details on how this is done, see the section within the script ecosystem overview on ‘mat\_act\_div.py’.

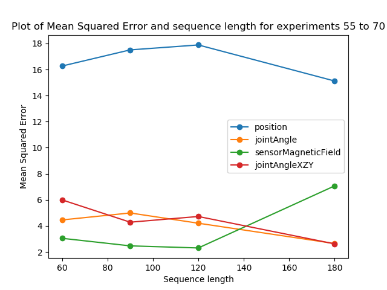
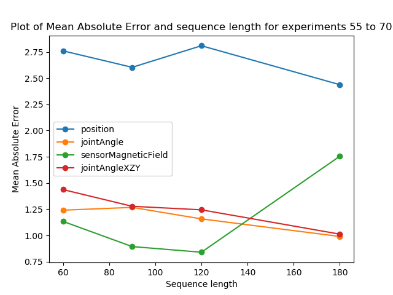
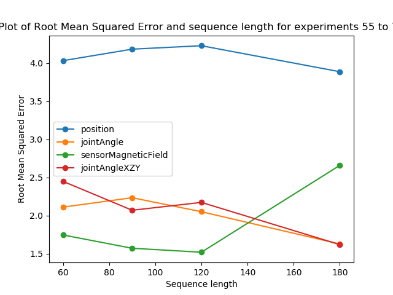
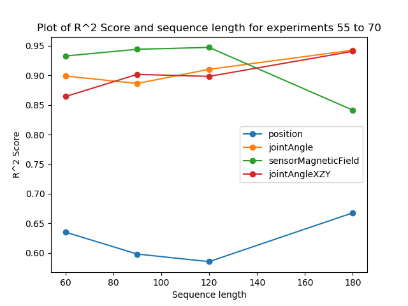
The aim of this experiment was to build models that are trained on one type of raw measurement; this is similar to experiment set 3 but with two differences: we’re now assessing on the single activity score for the file in question (i.e. a value of 0, 1, or 2), and we’re now using files that contain a single activity from a subject rather than all of the assessments (in most cases, though some smaller source files exist that contain only a handful of activities). We wanted to see whether or not the same raw measurements were as comparatively useful for single act source files as they were for the original files that contained all of the data; in a sense, we wanted to see if the measurements were as consistent when we removed a lot of the ‘non-assessment’ data from the source files (that is, data within the source files that don’t have an activity being undertaken). If this was the case, then we can conclude fairly strongly that these raw measurements are informative for NSAA assessments. Also, it’s worth noting that the ‘y labels’ assigned to each sequence coming from a single act file is now based on not only the file name but also what activity it represents; these two features are used to lookup the relevant value from the annotated Google sheet to get the assessed value of the activity.

**Results Discussion and Conclusions**

As expected, the raw measurements that proved to be the ‘best’ in experiment set 3 (jointAngle, jointAngleXZY, sensorMagneticField, and position) proved to be the best here as well, with the other 5 raw measurements heavily underperformed in comparison. This can be seen by the ‘useful’ 4 raw measurements obtaining a **MAE = ~0.15** while the others obtaining a **MAE = ~0.37**. Hence, this performs as we would expect, and the removal of a lot of the data between assessments from the source files does not affect the relative importance of some of the measurements. We can therefore conclude the following:

1. **The relative importance of jointAngle, jointAngleXZY, sensorMagneticField, and position as raw measurements compared to the other 5 is consistent with using just single act files rather than the complete source files.**

**Experiment Set 6: Different sequence lengths w/ overlaps for raw measurements**

For raw measurements, we have been (up until now) using a sequence length of ‘60’ as default for all of the experiments sets thus far. This represents a time window for the sequence of 1 second, as the suit is sampling at 60Hz, which therefore means that there are 60 rows of data stored in the source ‘.mat’ files every 1 second. Therefore, we wish to know whether or not this is a good time window to create each sequence with; however, with a given amount of source data, if we increase the sequence length, then we decrease the number of overall samples we can taken from this finite block. Therefore, it was decided to use a scaling sequence overlap; that is, we increase the sequence overlap in proportion to the sequence length to keep the number of samples more-or-less constant (accounting for differing numbers of left-over data at the ends of every file that can’t be made into a sequence). This is done by the following formula:

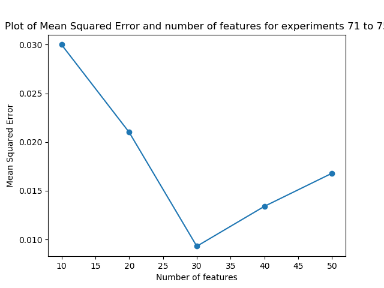
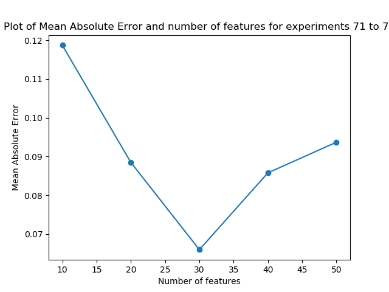
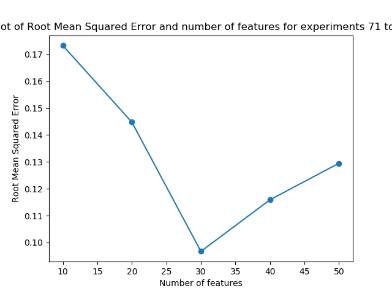
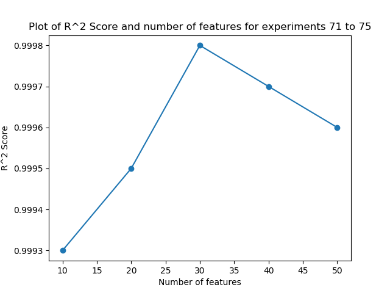
…where ‘’ and ’. For example, given our original sequence length of ‘60’, if we wish to experiment with ‘’, then (as ) and therefore (that is to say, we must set the sequence overlap proportion to 0.6667 to maintain a constant amount of samples. The reason we do this is to hold the dataset set size as a constant, and therefore any changes in performance of differing sequence lengths would be due to sequence length alone. Also note that we again only assess the differing sequence length on the overall NSAA score output type (for reasons previously mentioned in the experiment set 5 discussion. Finally, we wish to explore differing sequence lengths on multiple different types of raw measurements and compared them side by side (hence why we plot them all together on the same graph); however, we exclude 5 of the less ‘useful’ measurements, as they’ve been determined in experiment sets 3 and 5 to be not as useful as the other 4; hence to same time and graph readability, we don’t consider them here.

**Results Discussion and Conclusions**

What we were looking for in the above graphs is a consistent pattern for a given graph among the four plotting lines of an increase or decrease in performance when we change the sequence length; in other words, whether performance increased or decrease in the same way for all raw measurements for a given sequence length. This would give us a strong indication of how sequence length alters the performance of a certain metric. However, not only can we not any inter-raw-measurements patterns of change but we can’t see any discernible intra-raw-measurements pattern (i.e. can’t see single lines changing particularly strongly with respect to the output metric). For example, looking at the graph for the MAE metric, none of the lines for any of the raw measurements show any consistent improvement or worsening with increasing sequence length; each of them improves at some point on increasing the sequence length and also worsens at some point on increasing the sequence length. Additionally, the lines aren’t consistent with each other: the improving or worsening performance with increasing sequence length is often exclusive to one measurement and doesn’t necessarily translate to the others. From this, we can conclude:

1. **Increasing sequence length for data from raw measurements beyond 60 doesn’t show a consistent increase or decrease of performance amongst different useful raw measurement types; this is most likely due to the capturing window of 1 second being long enough to correctly assess for a given output type**
2. **To save on computational cost of training and testing on unnecessarily-longer sequences, we decide to keep the sequence length of 60**

**Experiment Set 7: Number of features needed for stat value data**

We now turn our attention back to stat value data from ‘AD’ files (i.e. the output of ‘comp\_stat\_vals.py’ as opposed to raw measurements from files obtained by ‘ext\_raw\_measures.py’). As a requirement of the pipeline, the data produced by ‘comp\_stat\_vals.py’ needs to have its dimensions dramatically reduced while keeping as much of the inherent variation of the data set still present. In the context of the data block diagram as seen in the section for experiment set 4, this is the ‘width’ of the block. As sequences are extracted from the source data ‘block’ by moving downwards, the number of features we chose to set is independent of the sequence length, sequence overlap, and discard proportion (more on this later), and so doesn’t affect the number of samples used in any of the experiment sets, including this one.

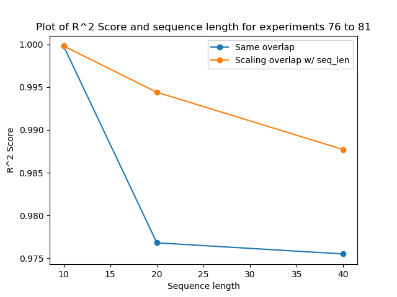
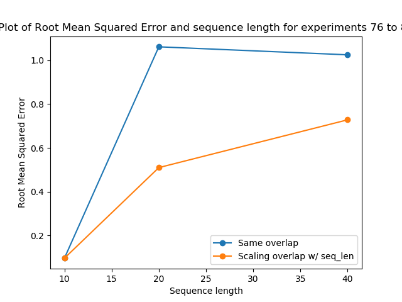
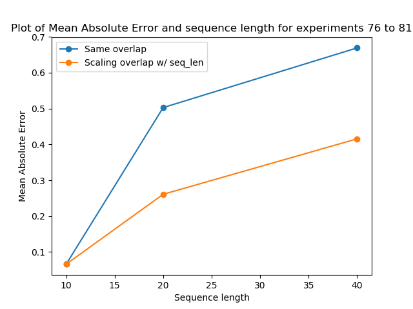
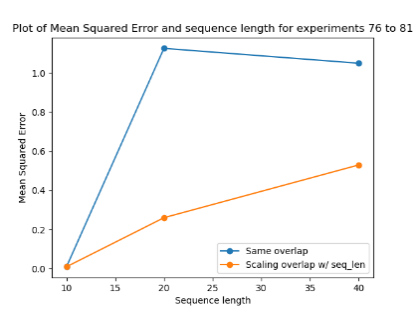
We have, up until now, been using ‘30’ as the default number of features. This is due to 30 features being the number of features that included over 99% of the variance when using the ‘variance threshold’; hence, this was a good default number of features to use for stat value files for any of the choices of feature reduction techniques. Also, it should be noted that the above results are for when the choice of feature reduction technique for ‘ft\_sel\_red.py’ was ‘pca’; that is, every result above (and also for every previous experiment conducted) has used PCA to reduce the dimensions of outputs of ‘comp\_stat\_vals.py’ to a smaller value. Further exploration of different feature selection/reduction techniques are explored at a later point, though PCA was chosen here as the default feature selection/reduction technique for other studies.

**Results Discussion and Conclusions**

Note that the above graphs are just for a single output type (the overall NSAA score) for reasons previously outlined and we are only interested in feature reduction for the statistical values from ‘comp\_stat\_vals.py’. From observing the above graphs, we can see a dramatic reduction in performance when we reduce the number of features from 30 to 10: we go from **MAE = ~0.1** to **MAE = ~0.17**, an increase in error of approximately 70%. This is most likely due to 10 features being not enough for PCA to capture the vast amount of the variance within the data it is given; hence, when we feed this data into RNN models, it is not as able to make accurate predictions of overall NSAA scores because it doesn’t have as complete a picture as if it’s using 30 features. Alternatively, if we increase the number of features from 30 to 50, we can see a notable worsening of performance (from **MAE = ~0.1** to **MAE = ~0.13**); hence, even though by using 50 features and therefore capturing more of the inherent variance and therefore ‘characteristics’ of the original data from ‘comp\_stat\_vals.py’, this is evidently not enough to overcome the consequences of using a higher-dimensional data for sequences to overcome the effects of the ‘curse of dimensionality’. Therefore, the following conclusions can be drawn:

1. **We shall continue with number of features to reduce to from data produced by ‘comp\_stat\_vals.py’ to 30.**
2. **This is the ideal ‘middle ground’ between sequences that don’t capture enough of the variance (e.g. number of features = 10) and sequences that have too many features to effectively train on (e.g. number of features = 50).**

**Experiment Set 8: Larger sequence lengths w/ overlaps for stat values from ‘AD’ files**



In the same way that we looked at increasing the sequence lengths used for raw measurements in experiment set 6, we now look at the same thing but from computed statistical values of source files by ‘comp\_stat\_vals.py’ and subsequently reduced to ‘30’ features. This time, however, we look at the effects of scaling the sequence overlap with sequence length (as done in experiment set 6) compared with not scaling the sequence overlap. We start with a sequence overlap of ‘0.9’ as standard (the value of which was determined by experiment set 4) for the initial default sequence length of 10. Note that the initial sequence length of 10 was chosen due to the initial lack of data available to use before the use sequence overlaps. Also, it’s worth pointing out that this length of 10 corresponds to a sequence capture window of 10 seconds (this is due to every ‘row’ of stat value data representing 1 seconds worth of data due to them being calculated over 60 rows of raw data sampled at 60Hz from the suit), unlike the raw data whose sequence length of 60 corresponds to a sequence capture window of 1 second. We also again assess only on the overall NSAA score output type for reasons previously discussed.

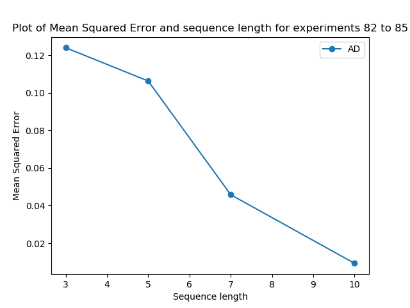
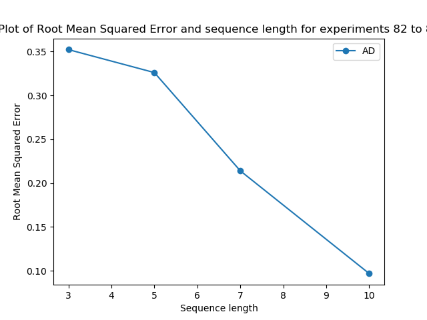
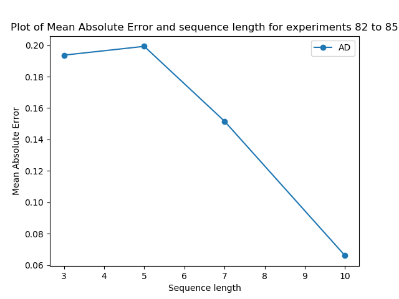
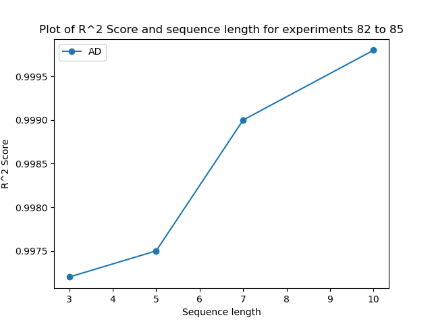
The primarily purpose of this experiment is to see if the performance of the models improved with a longer sequence length for computed statistical values of files. However, the secondary purpose is to determine if a scaling overlap improves the performance of the model, as was previously assumed for experiment set 6. This scaling overlap keeps the number of samples as a constant, as opposed to it decreasing due to the longer sequence lengths capturing more of the data per sequence. The experiments that have a scaling sequence overlap to keep the number of samples more-or-less constant are represented by the orange lines in the above graphs, while the ones with no scaling sequence overlap (and thus a reducing number of overall samples) are represented by the blue lines.

**Results Discussion and Conclusions**

We can see that in both scenarios where we either have a scaling sequence overlap or don’t, the performance of the model decreases with increasing sequence length. This is also significant because, for raw measurements, we can see that sequences are quite able to use longer sequence lengths and that ‘60’ seems to be an ideal point for these measurements. Therefore, from a smaller sequence length being more ideal for computed statistical values, we can conclude that this is a result of each row of data for computed statistical values containing more contextual data than a single row of raw measurement data, as opposed to it being down to the ideal data shape for an RNN to learn from (which is not the case as an RNN can learn quite well for a sequence length of ‘60’, as seen in experiment set 6). This is most likely due to the data contextual window growing too large. For example, when we increase the sequence length to 50 here, that means that each sequence takes in 50 seconds worth of source data, whch most likely isn’t able to account for the minutia of details that differentiates sequences of different classifications, overall NSAA scores, etc. Additionally, our reasoning of using a scaling sequence overlap in here and in experiment set 6 is justified, as for every metric above, the increased sequence length performs better with a scaling sequence overlap, as opposed to having a static sequence overlap of ‘0.9’, though in both cases it is still not as good as keeping the sequence length to 10. Therefore, we can conclude the following:

1. **Increasing the sequence length beyond 10 is not ideal for computed statistical values, which is shows that a contextual window of beyond 10 seconds makes learning for an RNN increasingly difficult**
2. **The use of a scaling sequence overlap shows better results than not using a scaling sequence overlap**

**Experiment Set 9: Smaller sequence lengths w/ overlaps for stat values from ‘AD’ files**

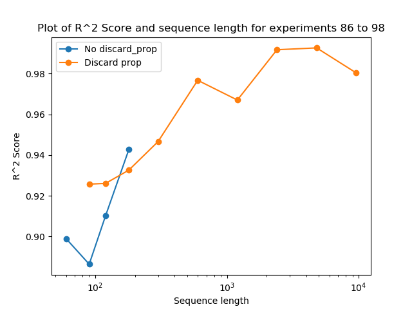
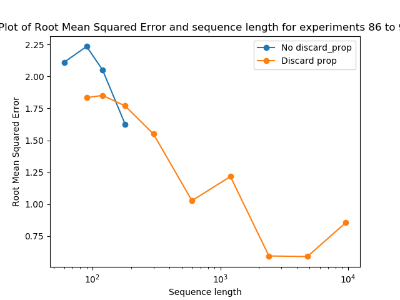
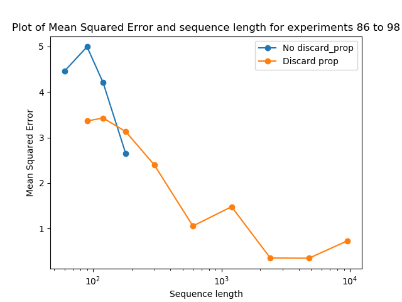
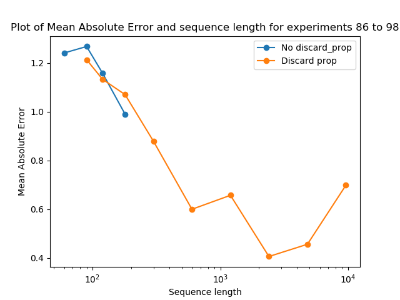
Before moving onto examining of larger sequence overlaps for raw measurements as opposed to just for extracted statistical values (as in experiment set 8), it’s necessary to ensure that a sequence length of 10 is indeed ideal for our setup, as the previous experiment set only concurred that a sequence length of 10 or smaller could be the ideal length. Here, we look at these smaller sequence lengths, going down to a sequence length of 3 (i.e. three sequences of vectors of 30 numbers are fed into the RNN which represents a context windows of 3 seconds). The expectation prior to undertaking these experiments was that a sequence length of 3 would be too short to draw time-contextual inferences from by the RNN when learning, while it was felt that 3 seconds might be too short to make accurate predictions for D/HC classifications, overall NSAA scores, etc., for many sequences. Note that everything else remains as it was in experiment set 8 (i.e. same source directory, same output type for the models, etc.), with the above graphs showing the results when we are using a scaling sequence overlap to keep the number of samples as a relative constant (i.e. decreasing the overlap as we decrease the sequence length), which corresponds to the orange lines in the graphs in experiment set 8.

**Results Discussion and Conclusions**

As we can see in the above graphs, the results evidently concur with our predictions: the performance for the overall NSAA score output type for models trained on extracted statistical values peak at a sequence length of around 10, with decreasing performance shown when we lower this value while keeping the number of samples relatively constant. As mentioned previously, this is most likely due to an RNN generally thriving on sequences of longer than 3 to get time-contextual information along with a context window of 3 seconds being not as useful as a 10 second window for NSAA scores. Furthermore, this knowledge that 10 seconds of context window for NSAA activity sequences is generally most useful will carry us over into the following experiment, where we shall attempt to use a similar context window to see if we get comparable results. Therefore, we can conclude the following:

1. **Smaller sequence lengths for models built on extracted statistical values showed worse performance, and therefore a sequence length of 10 for these sequences are ideal**

**Experiment Set 10: Very Large Sequence Lengths for Raw Measures w/ Discard Proportion**



Having established that a 10 second contextual window was ideal for extracted statistical values, we now wish to see if this is something that is exclusive to statistical values or whether it exists for raw measurements as well; if it does, then the 10 second context window is more likely to be an inherent characteristic of sequences made from NSAA assessment files, regardless of what measurement types are drawn from these files. For raw measurements, each row of data that is going into the RNN represents 1/60th of a second, as was previously established due to the sampling rate of the suit producing the data being 60Hz. Therefore, to get 10 seconds worth of data from raw measurements, we would need 600 rows of data (i.e. a sequence length of 600) while computed statistical values only need a sequence length of 10. However, setting sequence length to 60 for raw measurements presents us with two problems:

1. The increasing of sequence length for raw measurement data from 60 (as we have used as the default up until this point) to 600 will reduce the data 10-fold due to there being far more of the data needed for each of the sequences.
2. A sequence of several hundred has been previously established to be difficult for an RNN model to train on, along with much more computational demanding and containing a lot of possibly redundant data.

The first problem is solved by using a sequence overlap of 0.9 (for increasing from a sequence length of 60 to 600); this keeps the number of samples at a relative constant as we increase the sequence size. This ensures that any changes that occur in performance will be down to just sequence length and not the number of samples used; see experiment set 4 for a more in-depth discussion of this. However, this does not solve the other problem of too-long sequences to feasibly train the RNN model on. What we want is a way for an RNN model to gain a longer contextual window (e.g. of 10 seconds, corresponding traditionally to a sequence length of 600) without lengthening the sequence length itself. In this sense, any increase in performance while increasing the context window will be entirely down to increasing the RNN’s contextual information for a given sequence and not the actual sequence length. This is done by the ‘--discard\_prop’ optional argument supplied to ‘rnn.py’.

What this does is fairly simple: for each sequence, once it has been extracted from the source data block (also accounting for sequence overlap if appropriate), we keep only every ‘nth’ line of the sequence. For example, assume we have a sequence of length 600 and have set the ‘--discard\_prop’ argument to 0.9. This means that 90% of the data of the sequence is discarded, which results in keeping every 10th line in the sequence. This achieves two things:

1. Discards 540 rows and we are left with 60 rows of data which, as a result of taking the ‘kept’ rows at even increments along the sequence, means that we still have context covering the 600 rows.
2. By having even sampling of ‘kept’ rows of data, we also limit the possibility of missing any ‘important’ pieces of data that existing within the original 600 rows, as it’s likely that 1 or more of the kept lines would have captured some of this information.

We therefore use this technique to allow us to experiment with much larger sequence lengths (up to 9600) with corresponding discard proportions; these are shown by the orange lines in the graphs above. We also look at not using the discard proportions for increasing sequence lengths, though we can only go up to a certain sequence length before computational limits restrict us from continuing; for example, training models with data of a sequence length of 9600 with no discard proportion was expected to take between 3 and 4 days to train the model with the same number of epochs as the previous ones.

It’s worth noting that the sequence length is scaling for all experiments done here so the number of samples is always kept relatively constant. This is true for both the blue and orange lines above, as in both scenarios with a sequence length of 120, the discard proportion is set to 0.5. Finally, we only use one raw measurement here (joint angles) to experiment with longer sequence lengths. This is mainly due to previous experiment sets showing us that the performance of the useful 4 raw measurements are almost always completely in-line with each other with respect to increases or decreases in performances with changing independent variables. Therefore, to save on expensive experimentation time, we chose to evaluate the performance of models on only one raw measurement, safe in the knowledge that other raw measurements will most likely see the same change in performance.

**Results Discussion and Conclusions**

One thing we can see from the graphs above is the noticeable increase in performance (e.g. lowering of MAE on the test set) up to approximately a sequence length of 600. This is the point on the graph of the MAE metric where the discard proportion line shows an upwards inflection afterwards (i.e. when going up to a sequence length of 1200, the MAE gets higher); furthermore, past this point in the graph, the improvements are comparatively minimal for the extra time it takes to pre-processing the data. With very large sequence lengths and corresponding discard proportions, even though the amount of data that is input into the RNN models, it takes longer and longer to prepare the data. Therefore, a compromise was reached in the **sequence length of 600** being the best choice given the proportional discard proportion for how long it takes the model to compute.

This is also a good length to have as it means the ‘ideal’ (in the sense of model performance and time needed to compute) context window for raw measurements with a high discard proportion and scaling sequence overlap is 10 seconds, which is the same as for extracted statistical values. Another thing to note is that, for sequence lengths where we have results for both the blue and orange lines in the graphs, we see that the orange line (representing using a discard proportion) shows better results. This shows that, given the same sequence overlaps, when using a discard proportion to reduce the sequence length that is used to train the models, we see better model performance; this is most likely due to RNN models not performing for sequences of several hundred or longer. We can therefore conclude the following:

1. **Performance of models increases with higher sequence length and scaling discard proportion, with a good compromise of performance and computing cost being a sequence length of 600.**
2. **The ideal context window of 10 seconds for extracted statistical values is more-or-less replicated here, with the 10 second context window (sequence length of 600) being a high-performance parameter setting, and therefore 10 seconds is a good window for data from NSAA files in any measurement form.**
3. **Using discard proportion over not using it for identical sequence lengths shows better performance, which indicates that RNN models struggle to perform on sequences of too-high length.**

**Model Predictions Set 1: Natural movement files on models built on NSAA and walk files**

MAE between true and predicted NSAA scores over files (NSAA) = 4.66,

MAE between true and predicted NSAA scores over files (6minwalk-matfiles) = 5.03

Percentage of correct predicted file D/HC label (NSAA) = 75.92%

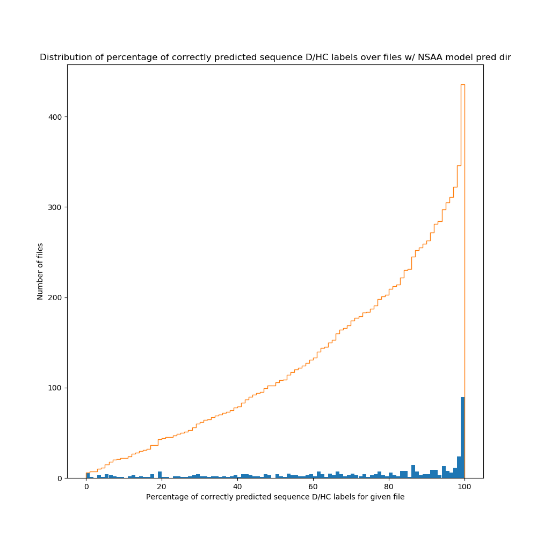
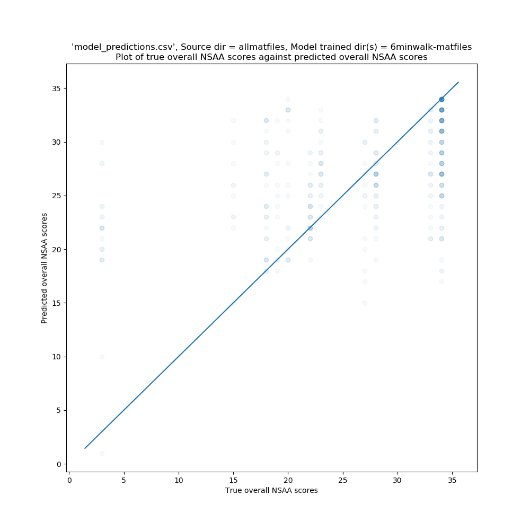
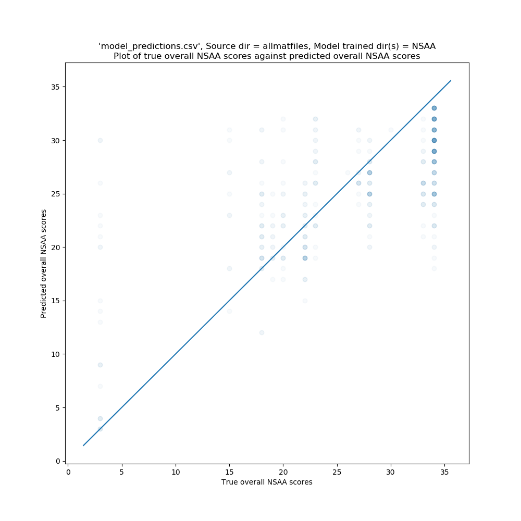
Percentage of correct predicted file D/HC label (6minwalk-matfiles) = 74.54%

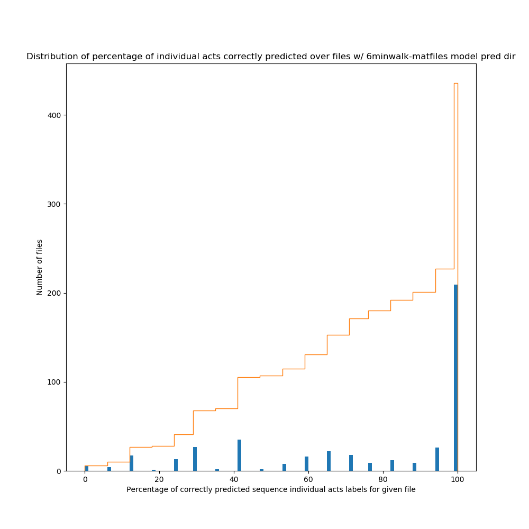
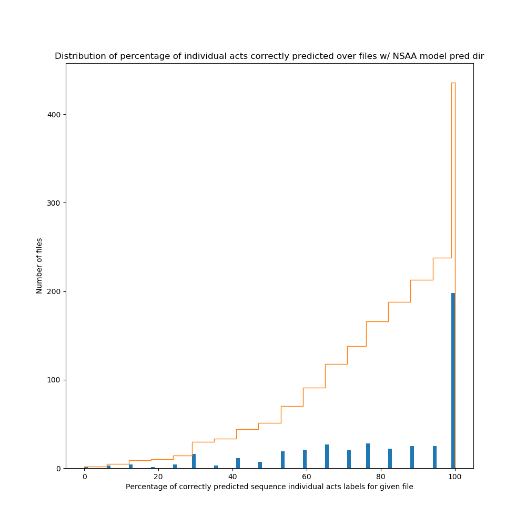
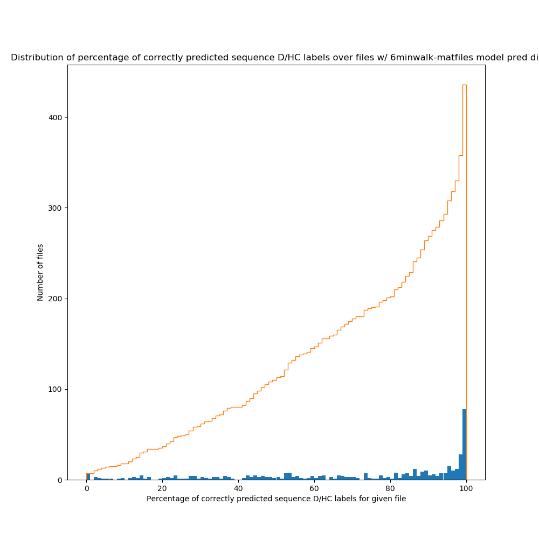
MAE of percentage predicted wrong sequence D/HC classification over files (NSAA) = 28.72

MAE of percentage predicted wrong sequence D/HC classification over files (6minwalk-matfiles) = 29.28

Average percentage of single acts correctly predicted over files (NSAA) = 80.84%

Average percentage of single acts correctly predicted over files (6minwalk-matfiles) = 75.28%





With the models now built that we will most likely be sticking with for the duration of the project (i.e. with an ideal sequence length, overlap, discard proportion, raw measurements used, etc.) ascertained by the previous experiment sets, we now move onto using these along with ‘model\_predictor.py’ to predict on whole files. Previously, in all experiment sets, the data that was reported was testing sequences from a mixture of the source files used to build the data set. This meant that assessment was done on a sequence-by-sequence basis, where the assessment of each sequence was to be independent of each other. With ‘model\_predictor.py’, this is quite different: we instead provide a specific name for the file, which will use all of the data to do an assessment on (the difference being that for the previous experiment sets, there will be a mixture of target D/HC classifications, overall NSAA scores, etc. for the test sequences, whereas with ‘model\_predictor.py’ there is a D/HC classification, overall NSAA score, etc. that is common among all of the sequences within the source file.

For making predictions on whole files via ‘model\_predictor.py’, the broad sequence of steps is as follows:

1. Split the source file into sequences, each with the same ‘y’ labels.
2. Assess each of these sequences on each of the models that we wish to be using, with a model for each measurement type and each output type (e.g. if we are assessing on 4 measurement types and 3 output types, there will be 12 models that each sequence is assessed on).
3. For each of the output types, average the response over all sequences for a given measurement and then average these over all measurements to get a single prediction for the given output type.

Each assessment of a file made by ‘model\_predictor.py’ is then stored as a single row of results within ‘model\_predictions.csv’. This is what’s used for the following model predictions sets.

For the first set, we are concerned with every file that we have of subjects involved in what we call ‘natural movement behaviour’; that is, data captured by the suit of the subject doing activities that aren’t NSAA or 6-minute walk assessments, such as playing or eating. This amounts to over 400 files, with as much as 30 files for a single subject captured. Hence, for each of these files, we run it through ‘model\_predictor.py’ to make various predictions for different output types. This is helped by the use of ‘test\_altdirs.py’, which automates a lot of the process and doesn’t require the user to manually run ‘model\_predictor.py’ >400 times.

While we shall get to assessing NSAA files on models built on NSAA files in the next section, here we wish to look at how well models that are built and tuned on NSAA files perform when presented with natural movement behaviour files. Obviously, the models have never seen any part of the files before, and the data it is now assessing on does not necessarily have characteristics of the original NSAA assessment files; for instance, the models will have been trained on sequences from files that do specific activities (i.e. one of the 17 NSAA assessment activities) or contain the subject walking. However, the natural movement behaviour will most likely be of movement characteristics that it’s never seen, and so the models are required to generalize their knowledge to other types of movement.

**Results Discussion and Conclusions**

**Console Output**

The following covers the model predictions of rows 1 to 436 of ‘model\_predictions.csv; for information on a prediction-by-prediction basis, see these rows. Instead of showing the results contained in these rows individually, however, we instead computed several statistics from these rows for certain columns with the aim to provide an insight into how well models build on NSAA and 6-minute walk files predict on natural behaviour files. It’s also worth noting that, at present, the natural movement behaviour files only exist as raw joint angle files. Hence, we the only raw measurement type of model we can use is the ones built on joint angle data (unlike predictions on NSAA files we shall discuss later, which uses 5 total data types for models).

The first two console outputs as seen above takes the average of the difference between each file’s true value for overall NSAA score and predicted value. This is done by, for each relevant row of ‘model\_predictions.csv’, finding the absolute difference between the true and predicted overall NSAA value columns; this gives a measure of how well the model predicted the overall NSAA score for that file, with this difference being 0 if it predicted the correct score; this is then averaged over all the relevant rows. What we can see is that models build on NSAA files that predict natural behaviour files predict a file on average within **4.66** of its true value, whereas models built on 6-minute walk files predict within **5.03** of its true value. These are reasonably strong initial results, though are possibly slightly skewed given the prevalence of files within the 25 to 34 score range. This is also including every single natural behaviour file available, and so the scores might be negatively impacted by natural movement files that are more-or-less not possible to infer an exact score (e.g. a file of a subject sitting very still for long periods).

The second two console outputs are the percentage of files that have the correct D/HC predicted classification. For each of the files, ‘model\_predictor.py’ takes the most common ‘D’ or ‘HC’ label for the sequences for the D/HC output type and makes that the prediction for the file; for example, for a file that contains 100 sequences, if 60 of them are predicted as being from a ‘D’ file and 40 are predicted as being from an ‘HC’ file, then the overall prediction for the file is a ‘D’ label. We also have a ‘true’ D/HC classification for the file; this is just based on which of the two the file’s name begins with. Hence, the percentage of correctly predicted D/HC labels is the percentage of rows with a predicted D/HC label that matches the true D/HC label. The results of **75.92%** and **74.54%** shown that in a majority of cases, natural movement files assessed on models built on either NSAA or 6-minute walk files get a correct classification; this is a notable good result that indicates, with further refinement and tuning, that these models would identify quite accurately what type of classification the subject is by simply observing natural movement data.

With the third set of console outputs, we are still concerned with analysing the D/HC classification predictions. Unlike the second set, for each file we aren’t concerned with the single estimated classification of the file by the models; rather, we look at it on a sequence by sequence basis in that we want to see the percentage of correctly classified sequences for the file being tested on. We then compute the ‘percentage of predicted wrong sequences’; this is simply the difference of the percentage of correctly classified sequences and 100%. For example, for ‘allmatfiles\D10-001’, we have 77.81% sequences predicted as being of ‘D’ label and 22.19% predicted as being of a ‘HC’ label. Since they are all supposed to be ‘D’ sequences (since they all came from a ‘D’ file), it therefore got 22.19% of the sequences wrong: this is the ‘percentage predicted wrong sequences’. We then repeat this over all other files from the natural movement behaviour data set and find the MAE of this set. This results in an error percentage of **28.72%** on models built on NSAA files and **29.28%** on models built on 6-minute walk files. This is fairly similar to the second set of console outputs in that it shows the models predict correct classes the majority of the time, though ideally this will approach 0% for both model types (NSAA and 6-minute walk) as we continue to refine and improve the system.

The final set of console outputs are more straightforward. For every predicted file (i.e. row we are concerned with in ‘model\_predictions.csv’), we get the value contained within the ‘Percent of acts correctly predicted’. This looks at the single act predictions for that file (i.e. a list of 17 values between 0 and 2 that the model believes are the correct single act scores for that file) and sees how many of those it got correct with respect to their true values; this then manifests as this ‘percent of acts correctly predicted’ score. We then repeat this over all the natural movement behaviour files and average this value to find the scores that appear on the fourth set of console outputs. The values of **80.84%** and **75.28%** are quite impressive: this means that on average the models that are trained on NSAA files predict ~13.7 of the 17 activities with the correct score, while the models trained on 6-minute walk files predict ~12.8 of the 17 activities. It’s also noted that particularly accurate scores for certain rows generally correspond with an overall NSAA prediction that is quite close to it’s true value, which makes sense as they are both involved with ascertaining NSAA scores; it’s just that the latter is predicting cumulative scores.

**Graph Output**

Along with the console output that is produced by ‘graph\_creator.py’, as discussed above, we also have several graphs that have been produced. However, in comparison to previous experiment sets, it is somewhat harder to ascertain specific results from these graphs, as these contain ~400 points and therefore must rely on the more evident ‘trends’ shown within the graphs. The first two show the true overall NSAA scores for each natural movement behaviour file along the x-axis and the files’ corresponding predicted overall NSAA scores along the y-axis by the NSAA models and 6-minute walk models. The closer these points are to the ‘y = x’ line projected through the middle of the plot the better, as this means they are closer to their true predicted values. From these graphs, we can see a tendency for points to hover around this line; however, this is still a great deal of variation around these areas. We can also see the model particularly struggles when presented with files that have an overall score of ‘3’; this is most likely due to these files being from an ‘outlier’ subject (due to the subject not completing many of the activities, which resulted in a lower score than they most likely should have). Additionally, we can see that files that have an overall score of ‘34’ (i.e. being from an ‘HC’ subject) are often assigned a score a fair bit lower of between 25 and 30. These observations from the two graphs for both types of models therefore give us good guidance of where to focus on improving the models.

The next two graphs show the distribution of the percentage of correctly predicted sequence D/HC labels for every natural movement behaviour file we tested on. This is essentially looking at either the ‘Percentage of predicted ‘D’ sequences’ or ‘Percentage of predicted ‘HC’ sequences’ for each file (depending on whether the file is a ‘D’ or ‘HC’ file). We then plot the distribution of these percentages for each of the files for both types of models, along with plotting the cumulative distribution lines. We can see from these graphs that there is high number of files where 100% of its sequences were predicted with the correct label, which is doubly impressive given the models these files are testing on have never seen natural movement behaviour before. It also shows us that, in both model types cases, there is a wide distribution of scores, with a not insignificant number of files having fewer than 50% of their sequences correctly classified (~100 files in both cases out of ~400 total). These graphs highlight that, while we have seen fairly positive results for the classification as shown on the second set of console outputs, there is still a lot of room for improvement.

The final two graphs again show a distribution of files. This time, however, we are looking at the distribution of percentages of correctly predicted sequence individual acts labels (which corresponds to the distribution of scores used to compute the fourth set of console outputs). Here, we can see a noticeable difference between the two types of models: while the models built on NSAA files have ~50 files that have <50% of the correctly predicted sequence individual acts labels, models build on 6-minute walk files have ~100 files that have <50% of the correctly predicted sequence individual acts labels. This is particularly impressive with regards to the NSAA models and shows the models proficiency to learn these individual scores. The disparity between the two model types also makes sense: the 6-minute walks don’t contain any NSAA activities within them (only the walk) and so we are asking the model to make predictions for the individual activities for a subject when the model itself never sees any of these files; it’s only presented with sequences of subjects with corresponding individual activities. Because it can’t as easily draw any inferences about walk data to correspond to other activity scores, it makes sense that it doesn’t perform as well compared with a model that does see these activities.

**Model Predictions Set 2: Model performance on left-out vs non-left-out files**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 (already seen) | AD, Joint Angle, Joint Angle XZY, Position, Sensor Magnetic Field | 100% | D | 100% | 15 | 18 |
| D3 | AD, Joint Angle, Joint Angle XZY, Position, Sensor Magnetic Field | 23.53% | D | 80% | 15 | 26 |
| D3 | AD | 5.88% | D | 54.69% | 15 | 28 |
| D3 | Joint Angle | 29.41% | D | 97.1% | 15 | 27 |
| D3 | Joint Angle XZY | 23.53% | D | 81.7% | 15 | 26 |
| D3 | Position | 41.18% | D | 85.94% | 15 | 25 |
| D3 | Sensor Magnetic Field | 47.06% | HC | 35.49% | 15 | 22 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D11 (already seen) | AD, Joint Angle, Joint Angle XZY, Position, Sensor Magnetic Field | 100% | D | 100% | 27 | 27 |
| D11 | AD, Joint Angle, Joint Angle XZY, Position, Sensor Magnetic Field | 76.47% | D | 100% | 27 | 27 |
| D11 | AD | 70.59% | D | 71.88% | 27 | 28 |
| D11 | Joint Angle | 82.35% | D | 65.78% | 27 | 26 |
| D11 | Joint Angle XZY | 82.35% | D | 74.53% | 27 | 25 |
| D11 | Position | 52.94% | D | 84.84% | 27 | 28 |
| D11 | Sensor Magnetic Field | 47.06% | D | 99.69% | 27 | 26 |

In this model predictions set, we look at 2 subjects (‘D3’ and ‘D11’) and look at how well models perform on predicting upon them when they have seen the files in training vs when they haven’t. These two subjects were chosen as their overall NSAA scores were mid-range (i.e. weren’t close to a perfect ‘34’ and weren’t outliers like ‘3’), though we intend to repeat this for numerous other subjects in the near-future, and each table represents the results concerning a single one of these two subjects. It should be noted that the models these files are tested upon are models built on NSAA files, along with the testing files being NSAA files themselves, so the models should be familiar with this sort of data.

We’ll first examine the results of the ‘D3’ subject. The first row is based on models that have already seen the ‘D3’ file in training and not only is it familiar with the subject, but it is also familiar in the particular data it is being tested on. This is only useful to us as a ‘baseline’ of how well the model could theoretically perform, as this is using some of the data that has been used for training for testing purposes. This therefore holds no practical use as a metric, as files that we would present to the models in a real-world setting would obviously not have been seen by the models before. Note that this uses all the measurements that we have decided upon using (extracted statistical values and the 4 ‘useful’ raw measurements) to make its estimations upon. Unsurprisingly, it predicts all of its sequences with the correct D/HC classification, along with all of all of the single act labels. However, it is still out on its estimation of the overall NSAA score by 3, which is somewhat surprising as not only should it be closer because the models are familiar with the file’s data, but the single-act scores are completely accurate; if these single act scores are correct, then should also have a overall score of a correct value as this is simply the accumulation of these values. This discrepancy of performance of different output types is possibly due to the single-act models being better trained to deal with sequences from this file; the cause of this is prompt for further investigation.

For the second line of the table, we then observe how well models predict on files that have been left-out completely of the training and testing process. This more accurately simulates what it would be like for the model in ‘production’; that is, assessing on new files in its intended capacity. We can see a noticeable steep drop in performance of the models: it only gets 23.53% of the single act predictions correctly, the number of correct sequence classifications drops 20% (though it still correctly determines the file to be a ‘D’ file) and the predicted overall NSAA score drops to being 11 away from the real score, down from 3. This is particularly poor generalization performance for this newly-seen file for the models and is heavy motivation to continue with generalization techniques to generalise the models’ learning inferences to new files. It’s worth noting that this was a particularly low score for ‘D’ files (with most falling in the +20 range), so this might contribute somewhat to its poor performance, though further investigation into other ‘left out’ files will be conducted later to confirm this.

The next 5 lines of the table again look at models that have had the ‘D3’ subject ‘left out’ of the model training (done by setting the ‘rnn.py’ optional argument to ‘--leave\_out=D3’). However, instead of using all 5 measurements for the three output types and aggregating their predictions, we instead look at individual measurements in turn; that is, a measurement is extracted from ‘D3’ (e.g. joint angle) and is tested on the three that correspond to this measurement (with ‘D3’ being left out of all of them) for each output type. The aim with this is to determine whether or not any particular measurements are more useful for generalizing to new, unseen files. At this point, the results from these rows are fairly inconsistent: while models built from the position and sensor magnetic field measurements perform better with respect to the single act scores output, the joint angle and joint angle XZY measurements perform better with respect to the D/HC classification output type. Additionally, there is not a noticeable disparity of improvement for the overall NSAA score among the measurement (with the exception of sensor magnetic field). Hence, at this stage, this is not enough information to draw any conclusions from and more of these ‘left out’ models for other subjects are needed.

Much of the same conclusions can be drawn for the ‘D11’ subject table. For the models that are trained on all measurements when seeing ‘D11’ as a left-out file (i.e. the 2nd row of the table), however, the models are a lot better at predicting more accurate outputs: there is a much higher percent of acts correctly predicted, higher percent of correct predicted sequences, and a closer true overall NSAA score compared with that of ‘D3’; here, the model predicts the ‘D11’ overall NSAA score correctly, rather than in the case of ‘D3’ where it is off by 11. This is particularly impressive for ‘D11’, as these models obviously have never seen this subject before. A likely cause for this increase in accuracy, however, is that ‘D11’ is closer to the mean overall NSAA score amongst the subject groups: ‘D11’ is more representative of the ‘average’ subject with Duchenne. Again, however, when we look at the left-out models but only single-measurements (rows 3 to 7 of the table), we again cannot drawn any particular conclusions with respect to one measurement or another being more useful to model generalization of unseen data; again, more of these left out subjects (rather than just ‘D3’ and ‘D11’) are most likely needed to draw any conclusions in this regard.

**Model Predictions Set 3: Model Performance over the Chosen 5 Subjects**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100% | 19 | 27 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100% | 22 | 25 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 27 | 28 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50% | 33 | 28 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75% | 34 | 31 |

With a majority of the programming work and options added to each of the scripts, we now move into among the final stages of experimentation: testing on left out subjects to assess and improve upon generalization performance of models to assess new subjects. To this end, we chose 5 subjects (‘D3’, ‘D9’, ‘D11’, ‘D17’, and ‘HC6’). This is because these subjects show a great variety of overall NSAA scores and cover most of the spectrum of various scores we see in real-world subjects.

For this set, we built a total of 60 models. This is due to each model corresponding to each output type (D/HC classification, overall NSAA score, individual act scores), for each measurement we are concerned with (position, sensor magnetic field, joint angle, and computed statistical values), and each set of these built for a different subject that has been left out of training (5 subjects, as outlined above). Hence, we have models for this experiment set.

It should be noted that there are a few changes made between the previous model predictions set, as evidenced by the differing values for ‘D3’ and ‘D11’. The first change is that the true overall score for subjects ‘D2’ and ‘D3’ were found to be mistakenly labelled by the assessors, and so we’ve been working with slightly-incorrect data since then for these two subjects; the reference file which the scripts use to draw the NSAA labels has now been changed to reflect these new values. The second change is that we no longer use the ‘Joint Angle XZY’ measurement. This is due to learning that this is simply a different way of displaying the joint angle data within the suit data files and provides no more useful information to the user. What’s more, by including it within the measurements, it’s essentially ‘diluting’ the impact of the other measurements when they combine their assessments.

From the results seen above, we can see a propensity for the overall NSAA score predictions to tend towards the median overall value among all the subjects (~28); however, we can still see that the lower true value overall NSAA scores for patients tend to have predicted scores lower than this median value. This implies that there is still some generalisation going on amongst the models, though not to the degree for which we are aiming. This will be attempted to be rectified by various methods in the coming model predictions sets. It should also be noted that we will be paying special attention to generalising to outside of the median overall NSAA value; as we can see above for a subject with a true value of 19, the model struggles to generalise to this degree, which is something that needs to be looked into.

**Model Predictions Set 4: Model Performance for 5 Subjects for Specific Measurements**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100% | 19 | 27 |
| D3 | Position | 58.82% | D | 85.27% | 19 | 25 |
| D3 | Sensor Magnetic Field | 52.94% | D | 63.84% | 19 | 24 |
| D3 | Joint Angle | 47.06% | D | 90.4% | 19 | 27 |
| D3 | AD | 23.53% | D | 51.56% | 19 | 32 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100% | 22 | 25 |
| D9 | Position | 52.94% | D | 96.56% | 22 | 24 |
| D9 | Sensor Magnetic Field | 52.94% | D | 99.06% | 22 | 26 |
| D9 | Joint Angle | 58.82% | D | 85.62% | 22 | 25 |
| D9 | AD | 47.06% | D | 87.5% | 22 | 24 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 27 | 28 |
| D11 | Position | 41.18% | D | 85.62% | 27 | 31 |
| D11 | Sensor Magnetic Field | 70.59% | D | 99.06% | 27 | 24 |
| D11 | Joint Angle | 58.82% | D | 72.97% | 27 | 27 |
| D11 | AD | 58.82% | D | 65.62% | 27 | 29 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50% | 33 | 28 |
| D17 | Position | 94.12% | HC | 12.5% | 33 | 28 |
| D17 | Sensor Magnetic Field | 94.12% | HC | 12.5% | 33 | 28 |
| D17 | Joint Angle | 76.47% | D | 89.84% | 33 | 27 |
| D17 | AD | 94.12% | D | 93.75% | 33 | 28 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75% | 34 | 31 |
| HC6 | Position | 100.0% | HC | 64.45% | 34 | 32 |
| HC6 | Sensor Magnetic Field | 100.0% | HC | 100.0% | 34 | 33 |
| HC6 | Joint Angle | 100.0% | HC | 61.72% | 34 | 29 |
| HC6 | AD | 100.0% | D | 40.62% | 34 | 30 |

The model predictions set is an extension of the work that was done in the previous set. It looks at the predictions for each of the subjects for their associated ‘left-out’ models but, along with using the aggregate predictions made by the models trained on the four measurements, we now look at how each of the measurements predict in turn. That is to say, each subject has one measurement type taken from it and loads the appropriate three model (one for each output type) for that measurement type and that ‘left-out’ subject, as opposed to using all measurements as the basis of the models. The aim therefore is to see if there is any particular relationship between the measurement type and its prediction ability.

One benefit from this measurement set is that it doesn’t require us to build any more models than have already been build, as it simply uses all the models built for experiment set one, but we simply use fewer of the models per each of the rows in the above table (disregarding the included rows from the table in model prediction set 3). The rows copied over from model prediction set 3 also serve to show how well the aggregation ability of those rows work with respect to the ideal predictions for each given subject.

**Results Discussion and Conclusions**

The main takeaway from this experiment set is that **there is no obvious measurement type that is ‘dragging down’ the aggregate predictions seen in model predictions set 3** at this stage. For the overall NSAA score predictions, each measurement generally predicts within 1 or 2 of each other and no measurement type is generally closer to the true overall NSAA value than any of the others. The same can be said for the percentage of correct predicted sequences (for output type ‘dhc’): there are no measurements that consistently perform better than the others to predict sequence D/HC labels. We can note two things from this metric, however: the aggregation affect for the metric for output type ‘dhc’ works in our favor, as for 4 of the 5 subjects it increases the average percentage from its predictions using single measurements to something higher with the aggregate prediction (for example, for subject D11, the average of the 4 single-measurement predictions was (85.62% + 99.06% + 72.97% + 65.62%) / 4 = 80.82%, while the aggregate prediction over all 4 measurements was 100%). This suggests to us that **the aggregation effect of all measurements to cover outlying predictions is useful at least for output type ‘dhc’**. This, however, is not observed in the same way for the percentage of acts correctly predicted (for output type ‘single-act’), thought it also does not show an average decrease when compared to predictions made by single measurements. Furthermore, for this metric there is again no particular measurement types that stand out as being notably and consistently better or worse than the others in being useful to estimate a higher percentage of correct acts.

Hence, at this point in the process with the models struggling to generalize to subjects that it hasn’t previously seen, **we see no improvements by simplifying predictions by only considering one measurement at a time for a given subject over using an aggregation of the measurements**. Rather, for the percent of correctly predicted sequences output metric, we see an improvement in performance for 4 out of the 5 subjects when using this aggregation. Therefore, for upcoming model prediction sets, **we will continue to use measurement aggregations to make predictions for subjects left-out of the training process**.

**Model Predictions Set 5: Comparable Performance of ‘FR\_’ vs. ‘FRC\_’ Files for ‘AD’ Models**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | AD | 23.53% | D | 51.56% | 19 | 32 |
| D3 (FRC) | AD | 23.53% | D | 100.0% | 19 | 12 |
| D9 | AD | 47.06% | D | 87.5% | 22 | 24 |
| D9 (FRC) | AD | 58.82% | D | 100.0% | 22 | 27 |
| D11 | AD | 58.82% | D | 65.62% | 27 | 29 |
| D11 (FRC) | AD | 35.29% | D | 100.0% | 27 | 12 |
| D17 | AD | 94.12% | D | 93.75% | 33 | 28 |
| D17 (FRC) | AD | 70.59% | D | 100.0% | 33 | 26 |
| HC6 | AD | 100.0% | D | 40.62% | 34 | 30 |
| HC6 (FRC) | AD | 64.71% | D | 0.0% | 34 | 26 |

One of the oversights of the feature reduction we perform on the computed statistical values (i.e. the ‘ft\_sel\_red.py’ script operating on the ‘AD’ output of ‘comp\_stat\_vals.py’) is that we have been reducing the file’s dimensionality on a file-by-file basis: as single files of computed statistical values come into the ‘ft\_sel\_red.py’ script, they are projected to a lower-dimensional space one file at a time. This mainly due to a programming oversight and the fact that we computed statistical values one file at a time, so it felt natural to reduce the dimensionality on a file-by-file basis as well. However, this does not guarantee that each file will be projected to the same dimensioned subspace via PCA (which is the feature reduction technique that we have been using thus far in ‘ft\_sel\_red.py’) and, moreover, it is extremely unlikely that each file will be on exactly the same feature subspace as any other; hence, they can be considered to contain different features than each other, even if they each contain the same number of features. Hence, we wanted to investigate whether or not this will have been an issue for the models by building models from computed statistical values with files having been reduced to *the same* lower-dimensional space; if these performed better than the previous ones, then the models’ learning potentials benefit from having computed statistical values sharing the same feature space.

To carry out this prediction set, we first ran ‘ft\_sel\_red.py’ with the ‘--combine\_files’ optional argument set. This combines all the files vertically (i.e. stacks all the rows of computed statistical values) on top of each other before performing dimensionality reduction with PCA before reducing separating them back into their original files (i.e. with the same number of rows of data as before but with far fewer columns). These take the exact same form as their normally-reduced ‘.csv’ output counterparts in terms of shape, but each of these ‘newly’ reduced files share the same reduced dimensionality space. These files are placed in the same directory as the other files but with a ‘FRC\_’ (for feature-reduced via concatenation) at the beginning of the file name as opposed to ‘FR\_’ of the others. To get models specifically trained on these ‘FRC\_’ files, we build them in ‘rnn.py’ with the ‘--use\_frc’ optional argument to select these files (note: this is only done for models build on ‘AD’ rather than raw measurements as raw measurements don’t have their features reduced) and the ‘model\_predictor.py’ uses a similar ‘--use\_frc’ optional argument to select these models.

**Results Discussion and Conclusion**

As we can see above, each of the 5 left out subjects are tested on two groups models or the ‘AD’ (computed statistical values) measurement only for each of the three output tiles: one group contains models trained on ‘normally’ reduced computed statistical values (the ‘FR\_’ files), while the other contains the ‘concatenated’ version of the files (the ‘FRC\_’ files) which we described above. For the percent of acts correctly predicted metric (for the ‘single-acts’ output type), we see that the ‘FR\_’ files perform better for 3 of the subjects, while the ‘FRC\_’ performs better for 1 subject and they predict the same percentage for 1 subject. However, we see an improvement for 4 of the 5 subjects for the percentage of correctly predicted sequences (for the ‘dhc’ output type), suggesting that generally **models trained on computed statistical values to assess D/HC classification perform better if the stat values share the same feature subspace over all files**.

However, we see that for the predicted overall score metric, we see that, much like for the metric for the ‘single-acts’ output type, the original version of the files (i.e. as ‘FR\_’ files) are consistently better to use to train the models to accurately assess for 4 out of the 5 subjects. As a result of these findings, even though the new ‘FRC\_’ files build models that perform well on left-out subjects for D/HC classification, this isn’t the main focus of these models, which places the predictions of accurate single-act scores and overall NSAA scores ahead of that of basic classification. As a result of these priorities, we therefore decide to **continue to use the traditional way of reducing the dimensionality of files (i.e. on a file-by-file basis) and hence use the ‘FR\_’ files with which to build models**.

**Model Predictions Set 6: Chosen Subjects on Familiar vs Non-Familiar Models**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100% | 19 | 27 |
| D3 (already seen) | Position, Sensor Magnetic Field, Joint Angle, AD | 100% | D | 100% | 19 | 22 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100% | 22 | 25 |
| D9 (already seen) | Position, Sensor Magnetic Field, Joint Angle, AD | 100% | D | 100% | 22 | 22 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 27 | 28 |
| D11 (already seen) | Position, Sensor Magnetic Field, Joint Angle, AD | 100% | D | 100% | 27 | 27 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50% | 33 | 28 |
| D17 (already seen) | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | D | 100% | 33 | 31 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75% | 34 | 31 |
| HC6 (already seen) | Position, Sensor Magnetic Field, Joint Angle, AD | 100% | HC | 75% | 34 | 32 |

It’s worth clarifying at this point what we mean by ‘familiar’ vs non-familiar models. We consider the models that we have been using up to this point (i.e. where one subject was left out of training for any given model and we then use this model to assess using ‘model\_predictor.py’ the subject that was left out of training) to be ‘non-familiar’ with respect to the left-out subject, as the model is non-familiar with the subject when it comes to assessing it. However, we also want a reference point with respect to how well the models could potentially do when it is said to have ‘understood’ the subject; at this point, this is done by seeing a lot of the subject’s data during the training process. Note that on average, the models that are familiar with the subject will have been trained on ~80% of its data, as on average 20% per subject will have been placed in the testing set and so won’t have been used for testing.

An important thing to note about these ‘familiar’ models that we will have built: while they show much better results than the standard non-familiar models, it is not a good indicator of the ability of the models to ‘generalize’ to new subjects (i.e. properly function in ‘production’ where we wish to assess new subjects, which is what the ‘non-familiar’ models aim to replicate). Thus, the ‘familiar’ models must be taken for what they are: an assessment of the ability of RNNs to learn from the subject data from computed statistical values and raw measurements, and not necessarily its ability to generalize to new patients. This also provides us with essentially a ‘gold standard’: a target the models should aim for in their generalization capabilities.

**Results Discussion**

It should be noted that, in the table above, the rows with a file name containing ‘(already seen)’ were the subject files tested on models that were trained on data that included the subject in question, while the ones not containing it were the same files tested on models that have not seen data from those subjects before (as done in model predictions set 3). Hence, these pairs of rows are exactly the same data, just assessed on a different set of 15 models (15 models being all combinations of each of the 5 measurements and for each of the 3 output types). Additionally, all rows that are ‘(already seen)’ file assessments are in fact trained on the same 15 models, as these 15 are just the models but with no subjects left out, so there is no reason (or way) to create models that both trained on all subjects and also exclusive to certain subjects during the assessment via ‘model\_predictor.py’.

As we expected, we see an improvement for each of the subjects when assessed on models that have already seen some of the data of that subject before (again, due to the train/test split ratio of the models being 0.2, it’s likely that these models will have seen ~80% of the subject files’ data during training). Notably, for 3 of the 5 subjects, there is a significant increase in the percentage of acts correctly predicted when assessed on models familiar with the subjects, while for the other 2 it is already very high and there being little to improve upon. The increase in D/HC classification potential is also highlighted in this set, with the ‘D17’ subject (which was previously misclassified when assessed by models not familiar with ‘D17’) now correctly classified as ‘D’. Finally, for all 5 of the subjects, the overall NSAA scores much more closely approximate the true overall score when using models that have seen the subjects before: in particular, ‘D3’ (which has a very low score relative to the average of all subjects) is much more closely approximated when using a model familiar with the subject, while for 2 of the 5 subjects they now achieve an exact match in scores when using ‘familiar’ models.

Again, it’s necessary to temper the magnitude of the takeaways of this experiment set, as this is looking at models that we won’t really be using further along in the project; rather, these just help serve as a baseline for the kind of results we should be aiming to achieve with these left-out subjects when we further look at generalization techniques in upcoming model prediction sets. However, one thing that we can definitely take away is that **the model architecture shows an ability to learn D/HC classifications and individual/overall NSAA scores for a variety of severities of subjects with Duchenne, even if at this point it struggles to generalize to unseen patients**.

**Model Predictions Set 7: Assessing Subjects Using Single-Act Files**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 91.74% | 19 | 26 |
| D3 (act 1) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 79.69% | 19 | 27 |
| D3 (act 2) | Position, Sensor Magnetic Field, Joint Angle | 35.29% | D | 100.0% | 19 | 26 |
| D3 (act 3) | Position, Sensor Magnetic Field, Joint Angle | 29.41% | D | 100.0% | 19 | 20 |
| D3 (act 4) | Position, Sensor Magnetic Field, Joint Angle | 35.29% | D | 100.0% | 19 | 22 |
| D3 (act 5) | Position, Sensor Magnetic Field, Joint Angle | 35.29% | D | 100.0% | 19 | 26 |
| D3 (act 6) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | D | 100.0% | 19 | 24 |
| D3 (act 7) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 100.0% | 19 | 27 |
| D3 (act 8) | Position, Sensor Magnetic Field, Joint Angle | 41.18% | D | 100.0% | 19 | 24 |
| D3 (act 9) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 19 | 28 |
| D3 (act 10) | Position, Sensor Magnetic Field, Joint Angle | 41.18% | D | 100.0% | 19 | 27 |
| D3 (act 11) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 81.25% | 19 | 27 |
| D3 (act 12) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 19 | 27 |
| D3 (act 13) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 19 | 26 |
| D3 (act 14) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 19 | 25 |
| D3 (act 15) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 19 | 24 |
| D3 (act 16) | Position, Sensor Magnetic Field, Joint Angle | 35.29% | D | 100.0% | 19 | 25 |
| D3 (act 17) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | HC | 42.19% | 19 | 23 |
| D9 | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 22 | 26 |
| D9 (act 1) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 22 | 24 |
| D9 (act 2) | Position, Sensor Magnetic Field, Joint Angle | 82.35% | D | 100.0% | 22 | 25 |
| D9 (act 4) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | D | 100.0% | 22 | 26 |
| D9 (act 5) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 22 | 25 |
| D9 (act 6) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 22 | 25 |
| D9 (act 7) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 27 |
| D9 (act 8) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 100.0% | 22 | 24 |
| D9 (act 9) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 100.0% | 22 | 24 |
| D9 (act 10) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 22 | 24 |
| D9 (act 11) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 22 | 26 |
| D9 (act 12) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 26 |
| D9 (act 13) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 22 | 29 |
| D9 (act 14) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 30 |
| D9 (act 15) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 22 |
| D9 (act 16) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 20 |
| D9 (act 17) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 22 | 28 |
| D11 | Position, Sensor Magnetic Field, Joint Angle | 82.35% | D | 94.84% | 27 | 27 |
| D11 (act 1) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 27 | 29 |
| D11 (act 3) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | D | 85.94% | 27 | 27 |
| D11 (act 4) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 27 | 28 |
| D11 (act 5) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 27 | 30 |
| D11 (act 6) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 27 | 22 |
| D11 (act 7) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 100.0% | 27 | 27 |
| D11 (act 8) | Position, Sensor Magnetic Field, Joint Angle | 88.24% | D | 100.0% | 27 | 22 |
| D11 (act 9) | Position, Sensor Magnetic Field, Joint Angle | 76.47% | D | 100.0% | 27 | 24 |
| D11 (act 10) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 27 | 22 |
| D11 (act 11) | Position, Sensor Magnetic Field, Joint Angle | 76.47% | D | 90.62% | 27 | 26 |
| D11 (act 12) | Position, Sensor Magnetic Field, Joint Angle | 35.29% | D | 84.38% | 27 | 27 |
| D11 (act 13) | Position, Sensor Magnetic Field, Joint Angle | 58.82% | D | 100.0% | 27 | 26 |
| D11 (act 14) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 100.0% | 27 | 23 |
| D11 (act 15) | Position, Sensor Magnetic Field, Joint Angle | 47.06% | D | 100.0% | 27 | 23 |
| D11 (act 16) | Position, Sensor Magnetic Field, Joint Angle | 82.35% | D | 100.0% | 27 | 29 |
| D11 (act 17) | Position, Sensor Magnetic Field, Joint Angle | 76.47% | D | 100.0% | 27 | 29 |
| D17 | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 22.66% | 33 | 28 |
| D17 (act 1) | Position, Sensor Magnetic Field, Joint Angle | 76.47% | D | 100.0% | 33 | 29 |
| D17 (act 2) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 90.62% | 33 | 28 |
| D17 (act 3) | Position, Sensor Magnetic Field, Joint Angle | 88.24% | HC | 42.19% | 33 | 22 |
| D17 (act 4) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 30 |
| D17 (act 5) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 20 |
| D17 (act 6) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 33 |
| D17 (act 7) | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 33 | 30 |
| D17 (act 8) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 31 |
| D17 (act 9) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | D | 100.0% | 33 | 31 |
| D17 (act 10) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 29 |
| D17 (act 11) | Position, Sensor Magnetic Field, Joint Angle | 100.0% | HC | 25.0% | 33 | 31 |
| D17 (act 12) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 20.31% | 33 | 33 |
| D17 (act 13) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 29 |
| D17 (act 14) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 26 |
| D17 (act 15) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 26 |
| D17 (act 16) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 0.0% | 33 | 27 |
| D17 (act 17) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | HC | 50.0% | 33 | 28 |

One of the things we were curious to know is how the performance on certain NSAA activities were reflected in the overall performance of the subject: we wanted to know if there were certain single activities of the 17 undertaken by each subject that showed to be more ‘impactful’ to the overall assessment of the subject. Therefore, we want to whether specific activities undertaken by subjects are indicative of their overall NSAA score, more so than other activities. This leads us to a hypothesis that motivates this model prediction set: **single-activity data that is consistent across multiple subjects at being able to approximate overall NSAA scores and D/HC classification would show these to be the more useful activities at assessing a subject**. The idea is that, if we found several activities of the 17 that are far better at predicting overall scores and classification than others, this could motivate specialists to pay special attention to the scoring of these activities during assessment or, alternatively or perhaps in addition, could result in the discarding of some or many of the 17 activities if they prove to have little or no correlation to the overall assessment.

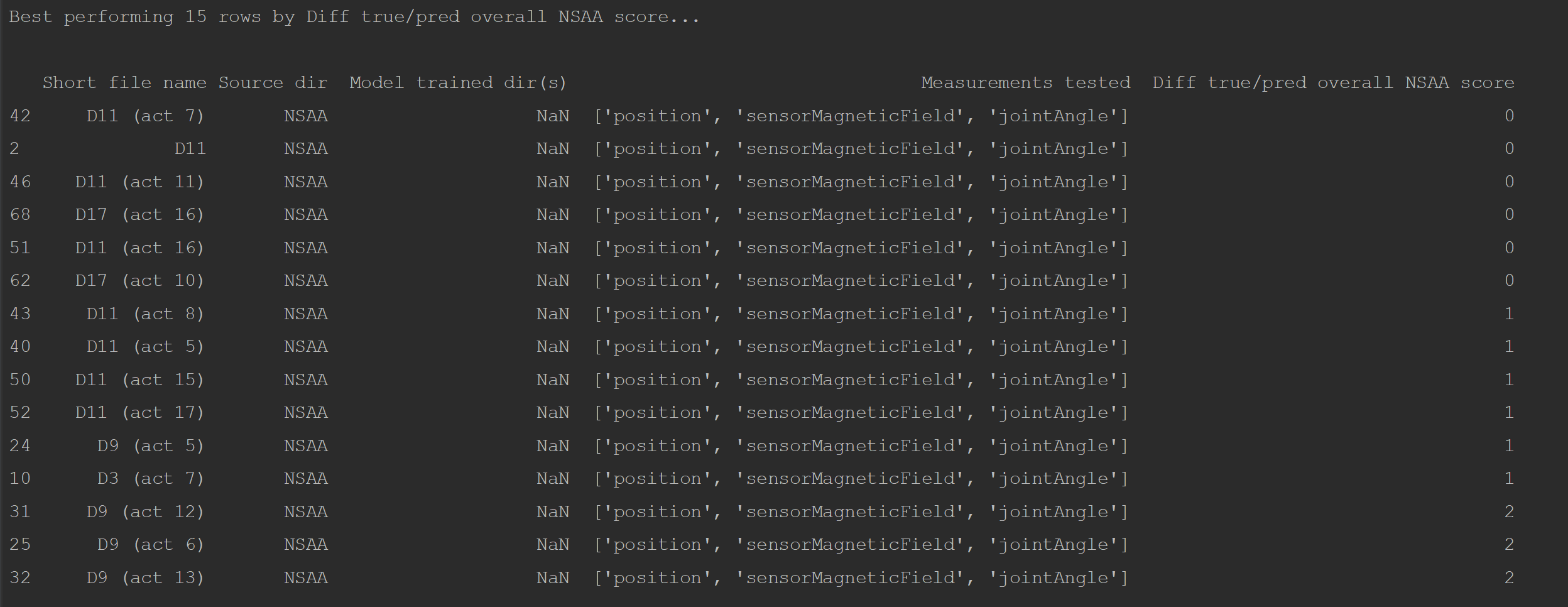
Conveniently, we have the single acts data files already in place and in the same ‘format’ as the complete files for each subject (i.e. the files containing all 17 activities with ‘filler’ in between). For each of the 5 subjects, we then compare the results of these 17 activities against the assessment of the subjects done with their complete files (as done in model predictions set 3). If certain specific activities prove to be better at assessing the subject for all, or most, of the 5 left-out subjects, then we know that these activities are more useful to the assessment. It should be noted that we are testing these single-act files for each of the subjects (17 activities x 5 subjects = 85 file assessments) on the same models as was built for model predictions set 3; in other words, the ‘single-activity’ files are being tested on the same models used to predict the ‘all-activities’ files, as in both cases the subject was seen by the models before while the data is still in the measurements it is expecting to see.

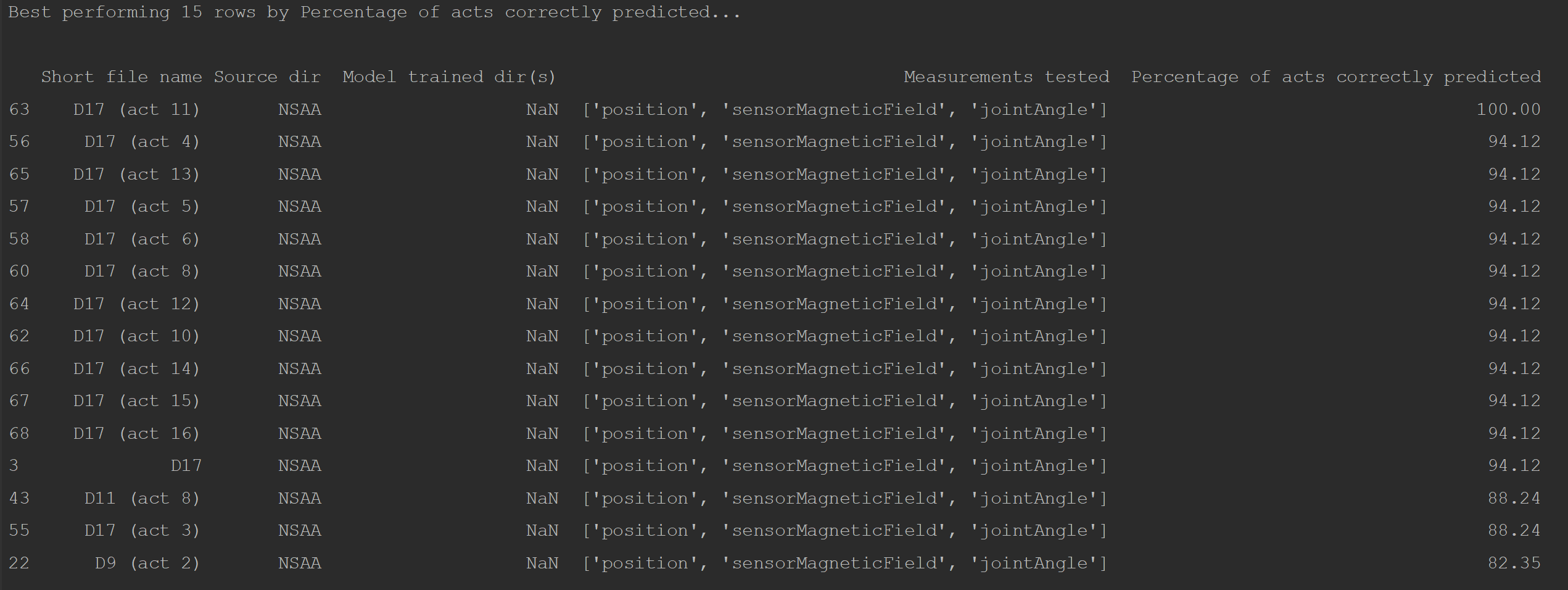
It should be noted that we are not including ‘HC6’ in this model predictions set. This is more out necessity than anything else: at the present time, the reference sheet that we have for subjects does not contain entries for most of the activities for subject ‘HC6’ for their activities. Hence, rather than including only several of these activities, we omitted this subject from this model predictions set. Conveniently, having an ‘HC’ subject for this set isn’t as necessary, as we are only comparing the performance of single act files to their corresponding all-act files, rather than seeing in general how the models perform in classification and regression statistics. It should also be noted that there are several activities missing from a few of the subjects; this is mainly due to the subjects having not performed those specific activities in their assessments. Rather than excluding these subjects completely, as it is only a few activities we instead decide to keep the subjects in this model predictions set and work around these missing values.

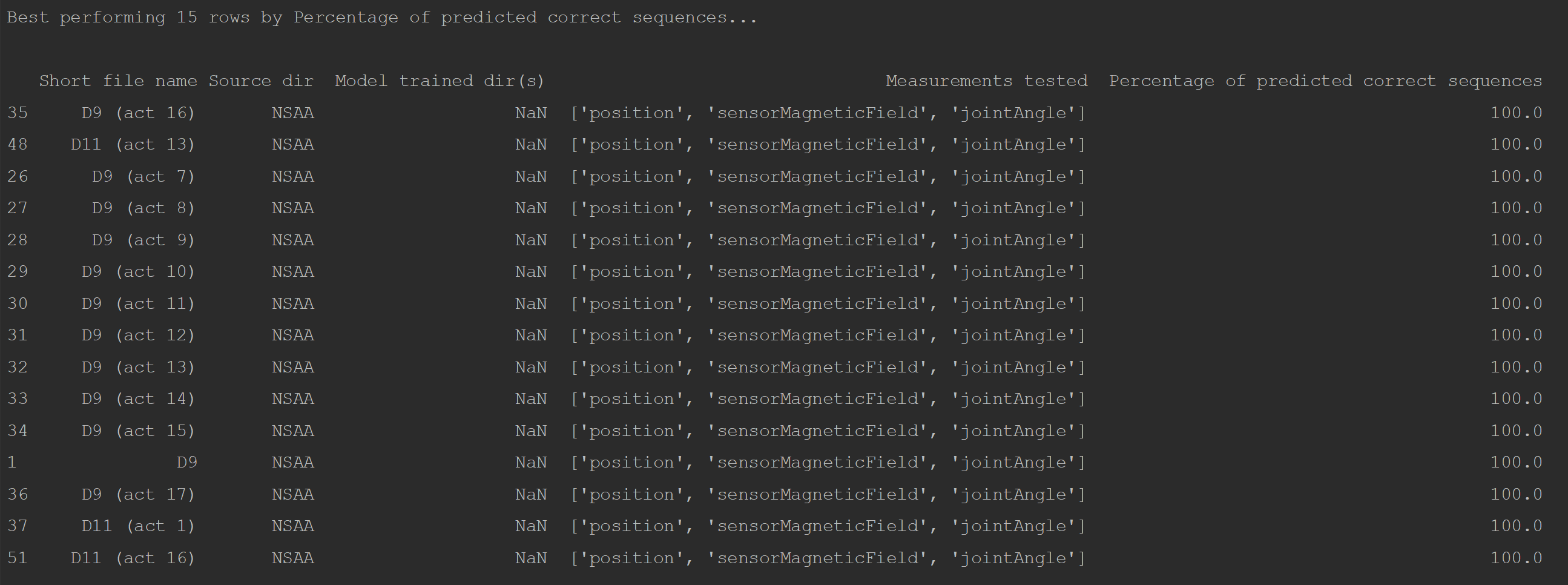
Finally, an important thing to note here is the absence of using the ‘AD’ measurement (i.e. the computed statistical values). This is due to the inability to use ‘ft\_sel\_red.py’ for single-act ‘AD’ files. This is because many single activities that are extracted from their base files are very short in length (often around 1 second for activities such as ‘step up R’). Consequently, when we compute the statistical values over 1 second intervals, the result is that the file containing the computed statistical values for a given subject and a certain activity (e.g. ‘AD\_D3\_act7\_stats\_features.csv’) may contain only several rows of data; hence, these might have a shape of something like (1, 4000). As in many cases the number of rows are fewer than the target number of reduced columns (e.g. target of 30 columns), we can’t reduce the dimensions of these files, so we can’t use them in our RNN models (as we can’t viably setup a model with ~4000 columns). Therefore, as we are limited in the number of possible single-act files for the computed stat value measurement for activity files of >30 seconds in length (which doesn’t cover many of the activities), we decide to discard this measurement and instead compute models just based on position, joint angle, and sensor magnetic field.

**Results Discussion**

As this table is fairly challenging to interpret at a glance, considering it has ~70 entries total with data contain, it was felt necessary to make use of the ‘predictions\_selector.py’ script in order to filter the lines based on certain metrics. We therefore ran the script three times: once to find the best 15 rows according to metric measuring the difference between the true and predicted overall NSAA values (for the ‘overall’ output type), the percentage of acts correctly predicted (for the ‘acts’ output type), and the percentage of predicted correct sequences (for the ‘dhc’ output type). The results can be seen below:







As we are primarily focused on the regression capabilities of the model, our focus mainly lies with the metric shown in the first and third images. However, for the third image seen above (for the ‘acts’ output type), there are numerous lines that perform very well according to that metric; hence, we decide to at this point focus exclusively on the first image that shows the best activities for each subject ranked according to how closely they predict the overall NSAA score (with perfectly predicting resulting in a ‘0’ value). We saw a greater number of acts between 5 and 7 in this better performing region (i.e. files of act 5, act 6, or act 7 of each subject tended to better approximate the true overall NSAA score on models that hadn’t seen the subject before). However, we wanted to view this in a bit more detail and find out which activities tended to perform better or worse and see if there any trends.

The graph above shows, for each activity number, the average difference between the true and predicted overall NSAA scores over the 4 subjects as plotted by the blue line, while the orange line is the average over both all subjects and all activities. Therefore, activities significantly below this line would indicate that they are more likely to be useful in predicting the overall NSAA score (and thus be a more useful activity to the assessment), and vice versa. One thing we can note is that activities 1, 2, 3, and 8 all perform significantly worse than the average; these are the ‘stand’, ‘walk’, ‘stand up from chair’, and ‘descent box right’. These are the activities that **show much lower correlation with the overall NSAA assessment for the subject, and hence are less likely to be useful in the assessment**. Conversely, we can see particularly good performance from activities 7, 9, and 10, which are the ‘climb box left’, ‘descend box left’ and ‘get to sitting’; these are **more likely to be activities that the assessor should pay special attention to according to the above results**.

It should be noted before making any conclusions that these are far from complete recommendations for specialists due to:

* Only considering the chosen 5 subjects (reduced to 4 due to not having the single-act data for ‘HC6’)
* Models struggling to generalize at this point, giving worse scores than if we were using models that generalized better
* Cross-confirmation of single-act times entered into the reference table; as multiple users entered certain times into the table, it’s advisable for everyone involved in creating it to confer and agree upon exact times for each subject’s individual activities in order to ensure accurate activity times from the files

While the above results give us an indication of what activities are the most and least useful, this does prove to potentially be fertile ground to draw conclusions about the NSAA assessment at a later point with more improved models and techniques.

**Model Predictions Set 8: Model Performance over the 5 Subjects w/ Outlier Excluded**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100% | 19 | 27 |
| D3 (other\_lo=D10) | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | HC | 50% | 19 | 28 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100% | 22 | 25 |
| D9 (other\_lo=D10) | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 22 | 26 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 27 | 28 |
| D11 (other\_lo=D10) | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100% | 27 | 27 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50% | 33 | 28 |
| D17 (other\_lo=D10) | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50% | 33 | 29 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75% | 34 | 31 |
| HC6 (other\_lo=D10) | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75% | 34 | 28 |

One of the features of this data set is that it contains one subject (‘D10’) that is an outlier; not for medical reasons (i.e. not because he suffers much worse with Duchenne in comparison with the other subjects), but rather as a consequence of him not undertaking the complete assessment. He instead only undertook 3 of the 17 activities as part of the NSAA assessment for what are believed to be personal reasons. Hence, a score of 3 for the overall NSAA score was recorded for that subject. This is significantly lower than the next lowest score of ‘15’ for subject ‘D2’. We believed that this may affect the models’ ability to generalise to new subjects, as it is forced to learn a wider range of scores for patients: with the ‘D10’ included, the model has to learn scores of between ‘3’ and ‘34’, while with it not included it only has to learn how to predict overall scores of between ‘15’ and ‘34’. The idea here was that, as these particularly-low scores aren’t a true reflection of the subject in question (as the subject only achieved this score due to real-world circumstances), by discarding this subject from the training set, it will make it easier for the model to learn scores within this narrower range and hopefully better generalise.

To achieve this, we simply modified the ‘rnn.py’ script to take in several subject names for the ‘--leave\_out’ argument and pass in the ‘D10’ subject as the second part of this argument. For example, for the ‘D3’ row above, we built it with models from the ‘--leave\_out=D3’ argument, while for the ‘D3 (other\_lo=D10)’, we built it with the ‘--leave\_out=D3,D10’ argument. We then specified the ‘model\_predictor.py’ script to search for these models with multiple subjects left-out with the ‘other\_lo’ argument.

**Results Discussion**

The findings from this set of model predictions weren’t particularly promising. Focusing on the performance for the ‘overall’ output type, the predicted overall NSAA scores did not seem to improve with leaving the ‘D10’ subject out of training completely; rather, for 3 of the 5 subjects they got worse with the ‘HC6’ subject getting worse by being 3 further away from the true score. And while 2 of the 5 subjects showed an improvement in approximating the overall score better, we saw an **increase in cumulative difference between the true and predicted scores from 20 to 23**. And while we saw performance increase for the percent of acts correctly predicted metric (for the ‘acts’ output type), we also saw a diminishing performance with respect to the ‘dhc’ output type: **when having ‘D10’ left out of the training set, only 3 of the 5 subjects have the correct label predicted, while including this subject we see 4 of the 5 subjects with the correct label**.

Rather than helping the model to more easily learn the scores within what we would consider the ‘normal’ range (i.e. not including particularly low scores like ‘3’) and help with the generalization ability, in removing the ‘D10’ subject the models tend to perform worse. This can be most likely attributed to the ‘D10’ subject adding some much-needed noise to the data set. It’s been well documented that machine learning models often need noise in the data to better perform on unseen data and thus to be said to be truly ‘learning’. Hence, this subject with a particularly low score helps the models in predicting other subjects within the ‘normal’ range. This also presents a good argument to be experimenting with adding noise to the data set, which we shall explore further in another model prediction set.

**Model Predictions Set 9: Model Performance for 5 Subjects on Single-Act-Concat Models**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100.0% | 19 | 27 |
| D3 (src sac) | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 75.0% | 19 | 29 |
| D3 (src normal) | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 75.0% | 19 | 28 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100.0% | 22 | 25 |
| D9 (src sac) | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 84.38% | 22 | 28 |
| D9 (src normal) | Position, Sensor Magnetic Field, Joint Angle, AD | 47.06% | D | 75.0% | 22 | 28 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100.0% | 27 | 28 |
| D11 (src sac) | Position, Sensor Magnetic Field, Joint Angle, AD | 64.71% | D | 75.0% | 27 | 24 |
| D11 (src normal) | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 75.0% | 27 | 24 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (src sac) | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 39.06% | 33 | 27 |
| D17 (src normal) | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50.0% | 33 | 28 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (src normal) | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 50.0% | 34 | 29 |

A feature of the NSAA assessment data files that we get as source data is that it contains the full recording of the assessment of the subject’s NSAA assessment, though these are sometimes broken down into 2 separate files. While these usually contain all 17 of the activities performed by the subject, the files often contain a data that is not quite as useful; for example, the subjects standing around before they undertake the next activity (possibly listening to instructions from the assessor about what activity to do at the time). Additionally, the start and end times of the activities that were recorded in the Google annotation sheet only include the start and end times for the activities that were completed and that could only include one of each activity within the time window. Hence, all data that was captured by the suit where the subject either failed to complete the activity or did the activity twice was also captured in these files. However, when ‘mat\_act\_div.py’ is used on these source files to divide up the files, we selected only the parts of the file where the activities were completed; hence, much of this data was discarded.

An option that was created as part of the ‘mat\_act\_div.py’ script was the ability to ‘recombine’ the single-act ‘.mat’ files into a single file per subject again. Much like the source file, this recombined file (which we refer to as a ‘single-act-concat’ file), contains all the activities for the assessment done by the subject in question. However, where it differs from the original source file is that it has a lot of data cut out of it, data that includes the subject just standing around or trying but failing to complete activities once or multiple times. A lot of this data was therefore not considered to be particularly ‘useful’ for training the model, at least in comparison to the completed activities. Moreover, it was thought that some of this ‘less-useful’ data (e.g. subjects standing around for extended periods of time) would possibly bias the model away from learning important insights from the ‘real’ activities within the data, as the models would have to account for both the useful and less-useful data in training. To this end, the ‘mat\_act\_div.py’ script was run to divide up and recombine the files, with the ‘comp\_stat\_vals.py’ and ‘ext\_raw\_measures.py’ scripts being used to obtain the computed stat values and raw measurements, respectively, followed by ‘rnn.py’ being trained with the ‘--use\_sac’ argument to use build models from these ‘single-act-concat’ versions of the computed stat values and raw measurements files, and finally ‘model\_predictor.py’ being run on the left-out subjects assessed on the models built with ‘--use\_sac’.

**Results Discussion**

There are three types of ‘left-out’ subjects that we consider in ‘model\_predictions.csv’ for each subject. The first of the three is the ‘baseline’; i.e. the results obtained in model predictions set 3 for subjects that were assessed on models that weren’t built on ‘single-act-concat’ files, but rather files that didn’t have single-activities extracted from them. The second type were for ‘single-act-concat’ subject files that were assessed using models trained on these ‘single-act-concat’ files; hence, to predict on these subjects, one would first need to run ‘mat\_act\_div.py’ along with ‘comp\_stat\_vals.py’ and ‘ext\_raw\_measures.py’ before assessing these subjects. The third type was for standard subject files (i.e. the same as those used in the first time as done for model predictions set 3 with no single activities extracted from them) but assessing on models built on ‘single-act-concat’ files; the idea of looking at both varieties of assessing file on models built on ‘single-act-concat’ files came from the desire to see whether models were any better or worse at assessing the subject when presented with files that contained this ‘less-useful’ data.

However, we can very clearly see from the above tables that **both varieties of assessed files (single-act-concat files or normal) assessed on models built on single-act-concat files performed worse than their counterparts which were assessed on models built from the normal data files**. This was the case over all 3 metrics with no improvements anywhere, which was unlike any other model prediction set we’ve seen thus far where even with changes that produced overall worse models, they still showed some signs of improvement for some metric(s) and some subject(s). While disappointing that this new approach didn’t lead to any improvement in results, it isn’t entirely surprising. For one, we are excluding a lot of the data: in the process of extracting small pieces of the source files and knitting them back together, we reduce the amount of raw data we have available by a factor of between 5 and 10, depending on the subject (generally higher if the subject has a lower overall NSAA score the subject generally takes more attempts at completing each activity). Even though we consider a lot of this data that we excluded ‘less-useful’, **the removal of not-as-useful data is not enough to overcome to loss in performance as a consequence of much less overall data**. Furthermore, in the same way as in model predictions set 8 that removing a subject most likely showed worse results due to a comparable lack of ‘noise’ now present in the data set, there is a good chance that the same thing happened here, where the data of the subjects ‘standing around’ and other minor activities (that weren’t necessarily a part of the NSAA assessment such as raising one’s arm) may have added some much-needed ‘robustness’ to the data set.

Additionally, another theory as to why models trained on ‘single-act-concat’ files performed not-as-well as models built on ‘normal’ files could be due to the capturing of activities that weren’t properly completed by the subject (which aren’t included as part of the ‘single-act-concat’ files) gave a fair bit of information about the subjects to the models. This is evidenced by the fact that for the lowest-scoring 2 of the 5 subjects, the models trained on ‘single-act-concat’ files predict a higher overall NSAA score than the models trained on the ‘normal’ files, where the **inclusion of the ‘failed’ activities most likely gave the models a better sense for the lower-scoring subjects that they should have a corresponding lower score**. Therefore, the main takeaway we have of this model predictions set is that **it is worthwhile using all of the available data captured by the bodysuit as opposed to only using the data showing the completed activities**.

**Model Predictions Set 10: Model Performance 5 Subjects on Downsampled Files**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100.0% | 19 | 27 |
| D3 (downsampled) | Position, Sensor Magnetic Field, Joint Angle, AD | 41.18% | D | 100.0% | 19 | 25 |
| D9 | Position, Sensor Magnetic Field, Joint Angle, AD | 52.94% | D | 100.0% | 22 | 25 |
| D9 (downsampled) | Position, Sensor Magnetic Field, Joint Angle, AD | 47.06% | D | 75.0% | 22 | 23 |
| D11 | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | D | 100.0% | 27 | 28 |
| D11 (downsampled) | Position, Sensor Magnetic Field, Joint Angle, AD | 58.82% | D | 100.0% | 27 | 24 |
| D17 | Position, Sensor Magnetic Field, Joint Angle, AD | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (downsampled) | Position, Sensor Magnetic Field, Joint Angle, AD | 70.59% | HC | 50.0% | 33 | 25 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle, AD | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (downsampled) | Position, Sensor Magnetic Field, Joint Angle, AD | 58.82% | D | 25.0% | 34 | 23 |

Another feature of the data set that makes it less-than-ideal for machine learning model training is the distribution of data with different overall and individual NSAA scores and D/HC classifications. In particular, with regards to predicted overall score, we have a lot of data with an overall score of 34 (as this covers every ‘HC’ file), while only one subject with a score of ‘3’ and one with ‘15’. Hence, we have a very uneven distribution of classification and regression scores that we are expected to train one which, while it reflects the real-world nature of the dataset and the real distribution of subjects with Duchenne, it does not help with the ability of models to learn generalized knowledge of a variety of classifications and regression scores that is able to generalize to new subjects. To overcome, we decided to experiment with a rebalancing approach to the data so that we have the exact same number of sequences for every possible overall NSAA score that we are presented which are used to train models.

To do this, we consider a down sampling approach. This is primarily due to time and computing restrictions at this point: with the up sampling approach, each model would take approximately 30-35 minutes to built which, given we have 12 models per left-out subject to build (3 output types x 4 measurement types), it would take between 30-35 hours to build the requisite models; meanwhile, down sampling the data and building models from this data takes a fraction of the time as each training epoch takes a much shorter length of time in comparison. To down sample the data, the ‘rnn.py’ is specified with the ‘--balance=down’ argument; this then results in the requisite functions from the ‘data\_balancer.py’ script to be called. From here, for any overall NSAA score for sequences that isn’t the ‘least popular’ overall score, the sequences are instead randomly sampled from the pool of sequences all corresponding to a specific overall NSAA score. This random sampling is repeated until we have a number of samples equal to that of the ‘least popular’ overall score (note that this is sampling with replacement as the same sequence can be selected multiple times by the random sampler). This is then repeated for all overall NSAA scores that appear in the data set: this results in a much smaller data set but with an equal number of sequences for each overall score.

**Results Discussion**

With regards to the two types of output lines in the table above, the first is the standard ‘left-out’ subject results as seen in model predictions set 3, while the second is the same subject file but assessed on models built on a balanced data set by way of down sampling. While we can see an improvement for 2 of the 5 subjects with respect to the difference between predicted and true overall NSAA scores for the ‘overall’ output type, we can see a significant worsening for the other 3 subjects. Indeed, **by using down sampling the overall accumulative difference error over the 5 subjects for the overall NSAA score increases from 20 to 29, an almost 50% increase**. Additionally, we see no consistent ‘pattern’ of the predicted scores: we would expect them to increase vaguely in line with the true overall value, even if they aren’t that close to the true overall NSAA values, as we can somewhat see with the standard ‘left-out’ subject results. Hence, the down sampled models show no sense of predicting when one sequence is higher or lower than the other with respect to overall NSAA score, which the standard models can somewhat do.

A clue for what might be motivating this behaviour can be found in the distribution of these overall scores: the predicted values are limited to between 23 and 25. This is near the median value for the data set **which implies that the models are trying to minimize its own loss by simply guessing new sequences from unseen files to be somewhat close to the middle of the expected range of overall values**; in this sense, the models would be said not to be truly learning from the data but rather adopting a strategy to try to minimize its own loss by other means. A possible reason for this poor performance might be the massive reduction in data samples: while the standard ‘left-out’ subjects setup has a data set of roughly 13000 sequences, the models build from down sampled data only has approximately 2700 sequences to work with. This is due to there being so few sequences for the overall NSAA score of ‘3’ that we have to discard many of the most common sequences with corresponding scores (such as ‘30’ or ‘34’) so that we have the same number of each overall NSAA scores for the sequences. In a similar vein to model predictions set 8 and 9 where some of the data is discarded, we can conclude here that **the rebalancing of the data set to have an even distribution of scores is not enough to overcome to loss in performance due to comparable lack of data**.

**Model Predictions Set 11: Model Performance for 5 Subjects on Feature Concat Models**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 91.74% | 19 | 25 |
| D3 (feature concat = 60) | Position, Sensor Magnetic Field, Joint Angle | 23.53% | D | 77.23% | 19 | 28 |
| D3 (feature concat = all) | Position, Sensor Magnetic Field, Joint Angle | 41.18% | D | 87.95% | 19 | 29 |
| D9 | Position, Sensor Magnetic Field, Joint Angle | 64.71% | D | 100.0% | 22 | 25 |
| D9 (feature concat = 60) | Position, Sensor Magnetic Field, Joint Angle | 41.18% | D | 80.94% | 22 | 28 |
| D9 (feature concat = all) | Position, Sensor Magnetic Field, Joint Angle | 41.18% | D | 73.75% | 22 | 25 |
| D11 | Position, Sensor Magnetic Field, Joint Angle | 82.35% | D | 94.84% | 27 | 27 |
| D11 (feature concat = 60) | Position, Sensor Magnetic Field, Joint Angle | 82.35% | D | 69.53% | 27 | 26 |
| D11 (feature concat = all) | Position, Sensor Magnetic Field, Joint Angle | 70.59% | D | 63.12% | 27 | 27 |
| D17 | Position, Sensor Magnetic Field, Joint Angle | 94.12% | HC | 22.66% | 33 | 27 |
| D17 (feature concat = 60) | Position, Sensor Magnetic Field, Joint Angle | 94.12% | D | 79.69% | 33 | 26 |
| D17 (feature concat = all) | Position, Sensor Magnetic Field, Joint Angle | 88.24% | D | 81.25% | 33 | 26 |
| HC6 | Position, Sensor Magnetic Field, Joint Angle | 100.0% | HC | 83.2% | 34 | 31 |
| HC6 (feature concat = 60) | Position, Sensor Magnetic Field, Joint Angle | 52.94% | D | 14.06% | 34 | 26 |
| HC6 (feature concat = all) | Position, Sensor Magnetic Field, Joint Angle | 100.0% | HC | 86.72% | 34 | 32 |

Up until this point, we have been building separate models for each of the data types that are used to build models. For example, for a given subject to be left out of the training set and for a given output type that the models are to produce (e.g. ‘overall’), if we are using 4 measurement types total to assess the subject (e.g. position, sensor magnetic field, joint angle, and AD), we need to build 4 separate models, each built on a different measurement type that is from the same source data from the subject files. However, we wanted to look at whether it was possible to combine this data prior to feeding it through a model; that way, we would need only 1 model per subject file and output type combination. This would take the total numbers needed for a typical model predictions set (i.e. for 5 different left-out subjects) from 60 down to 15. This would hopefully hold two advantages:

* Reduce overall training time, as while each model will take longer to train the dramatic reduction in numbers of models to create and train would hopefully offset this to be an overall time save.
* The requirement of models to train and assess on multiple measurement types for each model may encourage it to learn more general trends from the data and hopefully would generalise better.

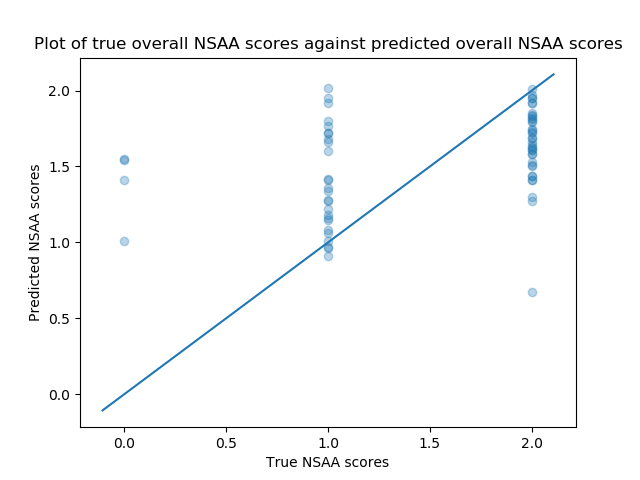
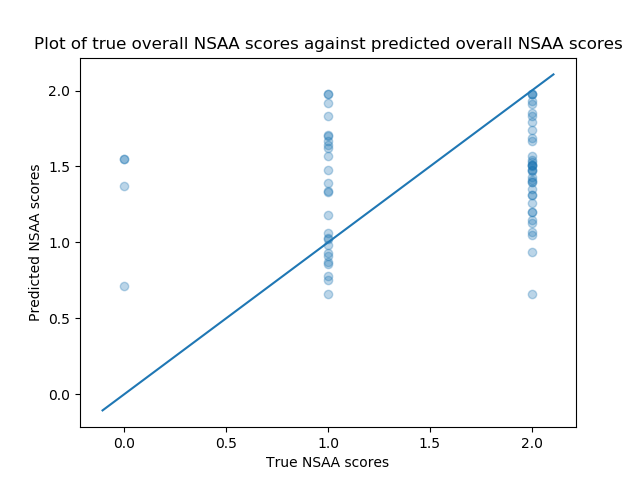
To do this, it was necessary to add the options for ‘rnn.py’ and ‘model\_predictor.py’ to concatenate the data together. Specifically, the data was the be added together along the ‘features’ dimension. For example, for a given left-out subject and output type, let’s say that we had (13026, 60, 66) as the shape for the ‘jointAngle’ data, (13026, 60, 51) as the shape for ‘sensorMagneticField’ data, and (13026, 60, 69) as the shape for the ‘position’ data. Concatenating all these together prior to feeding the data into the model gives a shape of (13026, 60, 186). Hence, all the raw measurements are fed into the same model rather than three separate models and then concatenating predictions made on the three separate models. It should be noted that we are excluding the computed statistical values from this model prediction set. This is due to that data having a shape that differs along more than the 3rd dimension in comparison to the other raw measurement data: the ‘AD’ data will generally have fewer rows and a shorter sequence length (which is at this time set to ‘10’ as opposed to ‘60’ for the raw measurement types). Hence, this data can’t be concatenated together with the raw measurement data and so we don’t consider this measurement here. Therefore, to have a fair comparison with the traditional method of creating models (one model per measurement type), we also don’t consider the ‘AD’ type in these rows of the above table. In other words, they are similar to how we predicted measurements in model predictions set 3, only not including ‘AD’ as part of the assessment process (hence why the corresponding lines in the above table are different to those in MPS 3).

**Results Discussion**

We consider two different types of concatenation of features (along with the standard method which is shown by the first line in each group in the table). The first uses PCA to reduce the dimensionality from 186 features down to 60. The reason this is done is so investigate whether feature reduction would be effective for raw measurements once they have been concatenated together (as we have up until this point only done so for computed statistical values). The reason that ‘60’ was chosen to be the number of features to be reduced to was that we have effectively seen that models can be built from sequences with between 51-69 features with them, so ‘60’ was chosen to be an ideal size that also held a majority of the inherent variance within the original 186 features. The other type of model that was built and assessed on was the using all of the concatenated features (i.e. not reducing the dimensionality of the concatenated raw measurements data); this is shown by the ‘feature concat = all’ lines. These are the lines that shown the results of left-out subjects built on models that are built in line with the explanation given above (i.e. having a data shape of (13026, 60, 186)). Also note that that the data for the subject we are using to test each of the models are transformed into the way the models are expecting: for example, for the models build with ‘feature concat = all’, the subject file that we are assessing on has all of its raw measurements concatenated together in the same way as was the data used to train the model.

Considering the difference between the true and predicted overall scores for various left-out subjects (for the ‘overall’ output type), we can see that in every case the feature reduction of concatenated raw measurement features to 60 shows a worse performance with respect to the standard method. We can thus infer from this that **the concatenation and subsequent feature reduction of raw measurements is inferior to using the separate models for each measurement type**. This is most likely due to the comparative loss of information: why we keep the majority of the variance when we reduce the dimensions down to 60, it’s still a significant loss when compared with using different models trained on individual measurement types. Additionally, while we don’t see such extreme results when we use the non-feature-reduced data in ‘feature concat = all’, we still do see slightly worse results: when looking at the total accumulation of the difference between true and predicted overall NSAA scores for all subjects, the standard approach yields a total of ‘18’ while the new approach has a total of ’22’. This is notably worse in comparison, particularly when looking at the ‘D3’ subject which has a particularly low score. This difference in performance is more pronounced when looking at the percentage of acts correctly predicted, where the standard approach to model building is still superior. We can predict that the reason for this loss of performance is the comparative difficulty of learning from sequences with ‘186’ dimensions as opposed to ‘60’ due to the curse of dimensionality, as we otherwise have all the same information available as if we had instead built three separate models per left-out subject and output type instead of the one with all three measurements used to train it. Hence, we can say that **concatenating the raw measurement features to train models is inferior to using training separate models with each raw measurement type used to train a different model**.

**Model Predictions Set 12: Model Performance for 5 Subjects (Single-act files) for the ‘indiv’ Output Type**



Up until this point, we have not really considered the fourth ‘output type’ that the RNN models can be built to target. This is the ‘indiv’ output type that ensures that the RNN models regress on a single continuous value on the single output neuron. In this sense, it is much like for the ‘overall’ output type except, while that output type regressed for the overall NSAA score for the given subject data, this output type regresses for the single-act score for a given subject **and a given activity number**; hence, while the ‘overall’ output will range between 0 and 34, the ‘indiv’ output type will range between 0 and 2. However, while models trained for the ‘overall’ output type can receive any type of file, be it a complete source file or a single-act source file, models trained for the ‘indiv’ output type are only expected to receive files representing single-activities. This makes intuitive sense, as it makes no real sense to predict the single-act score of between 0 and 2 for a file that would contain up to 17 activities, as the models wouldn’t know which activity of which the user wants the score. Hence, to assess the performance for the ‘indiv’ output type, we only use ‘single-act’ files.

These single-act files are the extracted single-activities of the source ‘.mat’ files for a given subject that are divided up based on the annotated Google sheet and by using the ‘mat\_act\_div.py’ script. From here, the raw measurements are extracted as normal, so for a given subject that we divide up into 17 activities and extract ‘position’, ‘sensor magnetic field’, and ‘joint angle’ raw measurements from, we would have 51 total ‘.csv’ source files. It’s important to note that we don’t compute the statistical values (i.e. the ‘AD’ measurement type) for single-act files. This is because, for single-activities, there might be comparatively very few rows in the source ‘.mat’ file’s data table and so, when we compute the statistical values for these, we may end up with files that we need to reduce the dimensions of with a shape of something like ’10 x 4000’ (as computing the statistical values reduces the number of rows 60-fold in our current setup); hence reducing the dimensions to ‘30’ becomes impossible with that few rows, and therefore it is simpler just to exclude the computed values when we work with single-act files.

We again look at the 5 subjects that are left-out of training in their corresponding models. The models themselves will have been trained on single-activity files to only target the ‘indiv’ output type, though for each left-out subject we would compute 3 models, one for each raw measurement type we’re interested in. However, instead of assessing the subjects’ ‘complete’ files on the models via ‘model\_predictor.py’ that have been trained on single-act files for the ‘indiv’ output type, we instead assess each of the left-out subject’s single-act files in turn on the aforementioned models; hence, we end up with ~17 assessments for each of the 5 left-out subjects in total, or ~85 assessments total. In fact, we don’t have quite this many assessments made (i.e. lines in ‘model\_predictions.csv’), as the Google annotation sheet doesn’t contain every single activity start and end times for the 5 subjects we assess on, due to some of the activities not appearing in the corresponding source file. Therefore, some of the activities for some of the subjects are missing.

Each of these assessments (e.g. act 13 for subject ‘D9’) has a ‘true’ score for the corresponding subject and activity number that’s determined by the ‘nsaa\_6mw\_info’ reference sheet. For each of the ~85 assessments made within this model predictions set, this true value is written to the column in the table (i.e. being one of 0, 1, or 2) and the aggregated predicted value made by the three models (one for each of the measurement types) is written as a value between 0 and 2. Note that this predicted value is a continuous value rather than a discrete value, as it was felt that a continuous value would be more useful for the above plots at showing how close or far away predictions were, due to the comparatively smaller range of true values when compared to plotting the true and predicted values for overall NSAA scores. The above left plot therefore shows each of the ~85 assessments on a 2D plot, with its x-position representing it’s true single-act NSAA value of either 0, 1, or 2 and its y-position representing it’s predicted value of between 0 and 2.

Finally, it’s worth noting that the right image is the similar to the left image in that it shows the distribution of true and predicted single-act scores for single-activity files of the 5 left-out subjects. However, the right image shows the same exact files that are assessed except that these are assessed on models that are already familiar with the subjects from training. Therefore, we would expect these to be more accurate (i.e. the points should be closer to the ‘y=x’ line); in this sense, this is analogous to model prediction set 6 in that we are trying to establish a baseline with which to compare the generalisation ability of the models.

**Results Discussion**

As we can see in the above left graph, the results for this setup are overall not particularly promising. For assessing single-act files scores for left-out subjects on models trained on single-act files to predict between 0 and 2, the predicted values are not particularly indicative of the true scores for the files. If they were highly correlated, then a large number of the points would follow the ‘y=x’ line in the graph closely; however, this is not the case as we can see in the aforementioned graph and, for activities and subjects with true scores of either 1 or 2, the predicted scores for those subject/activity combinations are not particularly indicative of the true scores and the predictions often have a large distribution of predicted values, ranging between 0.5 and 2 and, while for subject/activity combinations with large true values have somewhat higher predicted scores than their smaller true valued counterparts (evidenced by there the median being higher for subject/activity combinations of true value equalling 2 than those of true value equalling 1), this is not large enough to conclude that the models can differentiate between subject/activity combinations with different true single-act scores.

What’s also interesting is that, even with the subject being familiar to the models (i.e. the 5 left-out subjects instead being used to train the model along with the other subjects), as seen in the above right graph, the models are still not particularly good at assessing approximating the true single-act NSAA score. Hence, **the problem with this output type is less to do with its generalisation ability and more to do with the models not being able to learn the differentiating factors between subject/activity combinations of different scores**. This might be down to several factors, the most notable of which might be the drastic reduction in the amount of data when using only single-activity files to train the model (as this process involves ‘cutting out’ a lot of the data between the activities performed in the source file; see conclusions drawn in model predictions set 7). The results, or lack thereof, of this model predictions set might seem even more perplexing when compared with the results of previous model predictions sets when assessing for the ‘acts’ output type (i.e. 17 output nodes in the model predicting the single-activity scores for complete files). However, this could be down to regressing towards a value between 0 and 2 in the case of ‘indiv’, while for ‘acts’ we are classifying for 17 output nodes as being either a 0, 1, or 2. This might be due to **regression possibly not be as reliable for regressing towards 1 of 3 values (i.e. between 0 or 2) in comparison with classification as the limited range of true values for single-act files don’t easily lend themselves to regression**. Hence, an alternative method could examine the same ‘indiv’ output type but instead retraining the models as a classification model for three possible classes (0, 1, or 2) that might achieve better results than we see here.

**Model Predictions Set 13: Model Performance for 5 Subjects w/ Include 6-min Walk Data**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 41.18% | D | 100.0% | 19 | 27 |
| D3 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 35.29% | D | 75.0% | 19 | 27 |
| D3 | ['jointAngle'] | 47.06% | D | 90.4% | 19 | 27 |
| D3 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['jointAngle'] | 47.06% | D | 92.63% | 19 | 26 |
| D9 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 52.94% | D | 100.0% | 22 | 25 |
| D9 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 82.35% | D | 75.0% | 22 | 24 |
| D9 | ['jointAngle'] | 58.82% | D | 85.62% | 22 | 25 |
| D9 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['jointAngle'] | 70.59% | D | 82.19% | 22 | 25 |
| D11 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 27 | 28 |
| D11 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 27 | 23 |
| D11 | ['jointAngle'] | 58.82% | D | 72.97% | 27 | 27 |
| D11 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['jointAngle'] | 76.47% | D | 80.16% | 27 | 26 |
| D17 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 88.24% | HC | 50.0% | 33 | 22 |
| D17 | ['jointAngle'] | 76.47% | D | 89.84% | 33 | 27 |
| D17 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['jointAngle'] | 88.24% | D | 85.16% | 33 | 25 |
| HC6 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 100.0% | 34 | 29 |
| HC6 | ['jointAngle'] | 100.0% | HC | 61.72% | 34 | 29 |
| HC6 (add dir = 6minwalk-matfiles,6MW-matFiles) | ['jointAngle'] | 100.0% | HC | 73.44% | 34 | 30 |

Up until this point, we have been working exclusively with files taken from the ‘NSAA’ data set. This is primarily due to our main focus being on the analysis of NSAA files to make assessments upon the NSAA data from new subjects. However, this is not the only directory we can use to train models; specifically, we have both the ‘6minwalk-matfiles’ and ‘6MW-matFiles’ (both containing the 6-minute walk assessments of the subjects) and ‘allmatfiles’ (which contains the natural movement behaviour of the subjects). It should be noted that these are more-or-less the same subjects as in the NSAA directory, so when a 6-minute walk file references subject ‘D4’, it refers to the same ‘D4’ subject as appears in the NSAA directory. What we want to look at here, however, is whether or not our models’ generalisation ability improves with the addition of new files from other directories containing different types of data. When comparing to previous model prediction sets, performance has always dropped when we reduced the amount of data in the set (e.g. through the removal of outliers, down sampling the most popular classes, concatenating raw measurement features and then reducing the dimensions, etc.) so we wanted to investigate whether increasing the amount of available data (albeit walking data rather than NSAA assessment data) would help the model generalise to new subjects’ NSAA assessments, as in the previous cases. On the one hand, we are providing more data to the models, which should help it learn better and generalise easier, while on the other hand the new data is it is being used to train with might not be very useful in helping a model to generalise to NSAA activities due to this new data predominantly showing different movements than appears in the assessed NSAA data.

To enable this to happen, we modify the ‘rnn.py’ script so that the ‘dir’ argument can now take more than one directory argument. In the cases where we wish to use more than one directory as the source of the data files, we provide these to the same ‘dir’ argument but separated by commas. This causes the preprocessing step to repeat over all the directories and, with the data from each directory name from the ‘preprocessing()’ function, concatenates this ‘vertically’ (i.e. concatenating each sample, not concatenating the features as done previously), and then splits the total data into training and testing components. Once these models have been built, we then modify ‘model\_predictor.py’ in a way such that, if the ‘add\_dir’ is provided with additional directories to have been used within the model directory name, ensures that the models that are loaded are those trained on all the directories provided. For example, if ‘dir=NSAA’ and ‘--add\_dir=6minwalk-matfiles,6MW-matFiles’ are both provided to ‘model\_predictor.py’, it ensures that only directories containing ‘NSAA,6minwalk-matfiles,6MW-matFiles’ in the model directory names are used. This ensures that we can use ‘model\_predictor.py’ in conjunction with ‘rnn.py’ when it has built models from multiple source directories.

**Results Discussion**

There are several things to note before continuing to an in-depth results discussion. For one, the only rows that we are focused on here are the rows with all the rows reflecting subjects assessed using 4 measurements rather than just joint angles. This is so that the results we see here can be more easily compared with results from previous model predictions sets, which predominantly make use of all 4 measurements. The reason we include the predictions made with just using the joint angles is so we can potentially compare the results with models trained on NSAA and natural movement behaviour data sets: the natural movement behaviour data set (‘allmatfiles’) currently only exists as joint angles, and so we would only be able to build models and assess models built on these data sets using the joint angle measurement; hence, it’s necessary to have models here that also have only been trained on joint angles to make a fair comparison between using additional 6-minute walk or natural movement files.

Another thing to note (that should be fairly evident) is the difference between the two types of lines. For each subject, we have either than ‘standard’ setup (i.e. as used in model predictions set 3, where the subject is not seen by the models before, uses the 4 measurement types to do the assessment, but is other not modified), while the second type of line (i.e. with the subject name and ‘(add dir =…)’) are assessing the subject on models that have been trained on files in the NSAA source directory and the directories described by ‘add dir=’. However, the subject that is being assessed, however, is still source from the NSAA directory. This is because we still want to see how well the model generalises to left-out subjects from the NSAA directory, just also want to see the effect of using additional files from other directories.

From the table above, the results are not particularly promising. When comparing the performance of models on left-out subjects that have been trained on either just NSAA files or NSAA files along with files from ‘6minwalk-matfiles’ and ‘6MW-matFiles’, **we see an overall drop in performance when using these additional directories: the cumulative difference between true and predicted NSAA scores rises from 20 to 30**. This is a dramatic loss in performance, though the performance for the ‘percent of acts correctly predicted’ and ‘percent of correct predicted sequence’ metrics remain relatively stable (however, these aren’t as significant to assessing the models as the diff true/predicted overall score metric). This loss in performance is most likely due to the 6-minute walks between subjects being not nearly as distinctive between subjects as their corresponding NSAA assessments, as while the NSAA assessments test for many activities that could help models distinguish between subjects, the walk assessments aren’t as comparatively distinctive. Hence, in forcing the models to train on the walk data as well as the NSAA assessments, we are essentially ‘blurring the lines’ between subjects for the models due to them having to accommodate this additional not-as-useful walk data. This therefore reduces its ability to generalise on new subjects, as certain bands of overall NSAA scores are less associated with specific patterns of joint angles, positions, and so on than if it had only used the NSAA files. Therefore, **to assess left-out subjects from the NSAA directory we should not also train the corresponding models on the 6-minute walking data**.

**Model Predictions Set 14: Model Performance for 5 Subjects w/ Measurement Feature Reduction via PCA**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 41.18% | D | 100.0% | 19 | 27 |
| D3 (feature reduced = 20) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 41.18% | D | 100.0% | 19 | 27 |
| D9 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 52.94% | D | 100.0% | 22 | 25 |
| D9 (feature reduced = 20) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 75.0% | 22 | 27 |
| D11 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 27 | 28 |
| D11 (feature reduced = 20) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 76.47% | D | 100.0% | 27 | 27 |
| D17 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (feature reduced = 20) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 88.24% | D | 75.0% | 33 | 27 |
| HC6 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (feature reduced = 20) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 58.82% | D | 0.0% | 34 | 26 |

Referring back to model predictions set 11, we can see the results of concatenating the different measurement types that are used to train the models (excluding the ‘AD’ type) and subsequently reducing the dimensions down from 186 to 60 (as shown in the lines of the table containing ‘feature concat = 60’). This model prediction set therefore makes use of feature concatenation as well as subsequent feature reduction capabilities of the scripts. However, we now wish to observe the results of excluding the feature concatenation step of the above process; in other words, looking at reducing the dimensionality of each of the measurement types (69, 66, 51, and 30 for position, joint angle, sensor magnetic field, and AD, respectively) going into the models. The difference between this and model prediction set 11 (apart from reducing to a different dimensionality) is that here we are only building models from one measurement type at a time (as is the standard way currently that’s been used thus far everywhere except model prediction set 11) but still reducing the dimensions of the measurements.

The aim here is therefore to see if the models are able to generalise to the 5 chosen left-out subjects in turn when the models have been trained on feature reduced measurements and the subjects that are being assessed via ‘model\_predictor.py’ are also reduced to the same dimensionality for their features. The hope is that, with choosing a lower dimensionality for the data, the models may be able to keep a majority of the inherent variance of the training data whilst also generalizing better to new subjects due to the comparative ease of models learning from lower dimensioned data as opposed to higher dimensioned data. For this model prediction set, we have chosen **20** as the target lower dimensionality of each of the measurements; that is, models that are trained on position data will be reduced from a shape of ‘(x, y, 69)’ to ‘(x, y, 20)’, models trained on joint angles will be reduced from ‘(x, y, 66)’, to ‘(x, y, 20)’, and so on. This was chosen due to it being sufficiently small to hopefully make it easier to train the models whilst also capturing most of the inherent variance of the training data sets: generally, it captured between 80-95% of the variance of the data. It should also be noted that we apply the same dimensionality reduction technique to the ‘AD’ data while will have been noted to already have been reduced in dimensionality from ~4000 to 30 via ‘ft\_sel\_red.py’. This additional application of feature reduction is more to keep the process homogenous with respect to all measurement types so we could say that all models would always expect a feature dimensionality of 20, irrespective of the input measurement data.

**Results Discussion**

As we can see from the above table, we again see no real improvement in modifying away from the standard approach set up in model predictions set 3. For the percent of acts correctly predicted metric (corresponding to the ‘acts’ output type), we see an improvement for 2 of the subjects when assessing on models that have been trained on feature reduced data (with the subjects in question also being reduced to the same dimensionality as the models) but a worsening for 2 subjects. Likewise, for the percent of correct predicted sequences (for the ‘dhc’ output type), we see that while reducing the feature dimensionality to 20 improves for subject ‘D17’, it worsens for subjects ‘D9’ and ‘HC6’. Most significantly, however, is the performance of the model with respect to the difference between the true and predicted overall NSAA scores (for the ‘overall’ output type): **the sum of the differences between the true and predicted overall NSAA scores for the 5 subjects goes up from 20 to 27**, which is noticeably much worse than that of the ‘standard’ approach of model prediction set 3 (i.e. not reducing the dimensionality of each of the measurements any further within ‘rnn.py’ or ‘model\_predictor.py’). We can therefore say, as reducing the dimensionality to 20 shows no improvement over the three output types, that **reducing the dimensionality of each measurement, including raw measurements and further reducing computed statistical values, has a negative effect on the ability of the models to generalise to unseen subjects**.

While these models would most likely have found it easier to train with this lower-dimensioned data, it is shown from the results above that this is not enough to offset the results of losing a portion of the inherent variance of the data. Even though this is generally not much variance that is lost when compressing the data to a lower dimensioned space via PCA, we can see that it is enough to reduce the performance of the models on new subject data as the comparative lack of information from the data is enough to make distinguishing between sequences of different overall NSAA scores, individual NSAA scores, and D/HC labels that much more difficult. As a result, **to help the models generalise towards new subjects it’s necessary to keep all the dimensions of the measurement data that comes into the ‘rnn.py’ and ‘model\_predictor.py’ scripts**.

**Model Predictions Set 15: Model Performance for 5 Subjects w/ Added Gaussian Noise**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 41.18% | D | 100.0% | 19 | 27 |
| D3 (noise = 0.0001) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 47.06% | D | 75.0% | 19 | 26 |
| D9 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 52.94% | D | 100.0% | 22 | 25 |
| D9 (noise = 0.0001) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 58.82% | D | 100.0% | 22 | 27 |
| D11 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 27 | 28 |
| D11 (noise = 0.0001) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 64.71% | D | 100.0% | 27 | 26 |
| D17 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (noise = 0.0001) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 50.0% | 33 | 28 |
| HC6 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (noise = 0.0001) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 75.0% | 34 | 30 |

A common technique that has been used in order to help with overfitting (and therefore potentially with the generalization problems we have encountered) is adding noise to the data that the model is trained on. In doing this, it generally forces the model to learn general features of the data in order to approximate a more general function that models the training set, as opposed to be more easily able to learn specific minutia of the training samples that wouldn’t carry over onto the left-out set. Therefore, the theory in going into this model prediction set was that in adding noise, it might force our models to learn general features about movement data from position, joint angle, sensor magnetic field, and computed statistical value measurements and how they are connected with the true overall NSAA score, the D/HC classification, and the single-act NSAA scores; in other words, it was hoped that in adding noise to the training set it would help it better approximate to new, unseen subjects.

To carry this out, we first needed to compute the distribution of which we wished to draw from in order to add noise to our data set. With the data nearly fully preprocessed in ‘rnn.py’ (just prior to splitting into training and testing components), we add a certain amount of Gaussian-distributed noise to each sample. The reason we chose Gaussian for our noise distribution is that it’s generally the standard distribution used for noise in most machine learning models due to it generally modelling most real-world distributions the best. We first flatten the data from it’s form as separated sequences to these ‘stacked’ sequences: we go from a ‘x’ data shape of ‘(a, b, c)’ to ‘(a\*b, c)’. We then calculate the standard deviations for each of these columns so we have ‘c’ standard deviations total. Then, for each sample of this ‘stack’ of sequences (i.e. a vector of length ‘c’) and for each feature within this, we draw a random sample from the Gaussian distribution of ‘N(0, std(j)\*k)’ where ‘std(j)’ is the standard deviation of the column that corresponds to the current feature and ‘k’ is a constant factor. Hence, each value within the ‘(a\*b, c)’ matrix has a Gaussian distributed value added to it that is proportional to the constant factor ‘k’ and the standard deviation of the column the value is contained within.

Regarding the constant factor, this is essentially a ‘scaling’ factor of how much noise to add to the data set. If this is set to 0, then no noise is added, while if we set it to 1 then each value within the data set has an amount added to it that is drawn directly from the Gaussian distribution with mean 0 and standard deviation set to the standard deviation of each column of the data set, which in most cases is far too large. Indeed, in trying to find a good value for the constant factor ‘k’ we experimented with ‘1’ first; however, this outright prevented the models from learning for a variety of different measurements as input types and for different output types, as we wouldn’t see any drop in the training loss in training epochs. Indeed, we continued decreasing this factor and ‘0.0001’ became the first value we encountered that enabled the training of a model to actually decrease in the value of loss during training (i.e. showing that the models are able to now learn from the data). Therefore, the aim for this model predictions set is to directly compare the results of the ‘standard’ model setup to that of models with a small amount of added noise that is proportional to the standard deviations of each of the data’s features.

**Results Discussion**

One thing we can say immediately about the addition of Gaussian noise to the training set is that **only a small amount of Gaussian noise is able to be added; otherwise the models won’t train**. Therefore, the data can be said as being quite sensitive to the addition of noise, which implies that the data doesn’t contain any easily-recognizable patterns or trends that the models would be able to pick up on as, if there were, the models would probably be more resilient to noise than they are. An alternative to this could be due to the interdependency of many of the features on each other. For example, the position values for several features (e.g. those representing the ‘x’, ‘y’, and ‘z’ coordinates of the left-ankle) would be proportion to the position values of other features (e.g. those representing the coordinates of the left-leg) and so added random noise to each of these features, of which each addition of noise is independent of the others, would possibly distort this relationship. Therefore, this could possibly be avoided by instead adding noise per sample (i.e. over a vector of ‘c’ values) rather than per value so as to not distort this relationship too much and therefore possibly allow us to work with higher levels of noise.

As for the results themselves of this addition of noise to the data set, alternative models are rebuilt with this addition of noise with ‘k=0.0001’ and used to evaluate left-out subjects (which don’t have added noise to them as adding noise to the evaluation set in machine learning is generally avoided). These are then compared with the ‘standard’ setup results which have still proved to be the best at generalization to left-out subjects. We can see in the table above that difference between the true and predicted overall NSAA scores only worsened when using models built using this ‘noise-added’ data, as the accumulation of these differences between the overall NSAA scores rises from 20 to 22. While performance with respect to the metrics relevant to the ‘dhc’ output type (i.e. the percent of correct predicted sequences and the predicted D/HC label) remains the same and the performance with respect to the ‘acts’ metric (i.e. percent of acts correctly predicted) marginally improves for 2 subjects while decreasing for only 1, as a result of the worsened results for the ‘overall’ output type **we consider these noise-added models marginally worse than the standard setup at generalizing to left-out subjects**. This is most likely due to there not being enough noise added to the data set to help it generalize properly, while adding the requisite noise in practice distorts the data enough to make it impossible to train on.

**Model Predictions Set 16: Model Performance for 5 Subjects w/ Included Joint Angle Data of Natural Movement Behaviour**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | ['jointAngle'] | 47.06% | D | 90.4% | 19 | 27 |
| D3 (additional dir = allmatfiles) | ['jointAngle'] | 47.06% | HC | 48.44% | 19 | 27 |
| D9 | ['jointAngle'] | 58.82% | D | 85.62% | 22 | 25 |
| D9 (additional dir = allmatfiles) | ['jointAngle'] | 58.82% | HC | 34.38% | 22 | 24 |
| D11 | ['jointAngle'] | 58.82% | D | 72.97% | 27 | 27 |
| D11 (additional dir = allmatfiles) | ['jointAngle'] | 82.35% | D | 52.66% | 27 | 27 |
| D17 | ['jointAngle'] | 76.47% | D | 89.84% | 33 | 27 |
| D17 (additional dir = allmatfiles) | ['jointAngle'] | 82.35% | D | 67.19% | 33 | 27 |
| HC6 | ['jointAngle'] | 100.0% | HC | 61.72% | 34 | 29 |
| HC6 (additional dir = allmatfiles) | ['jointAngle'] | 100.0% | HC | 84.77% | 34 | 31 |

From all the model prediction sets we have undertaken thus far, there is a common theme running through many of them: the more data that we have, the better results we get on left-out subjects. This includes including any outliers, not down sampling the data, not reducing the dimensionality of the raw measurements, and so on. However, this doesn’t necessarily apply to including data from the 6-minute walk assessments: as we see in MPS 13, add 6-minute walk data to the data set along with NSAA files doesn’t result in any better assessments made on new subjects’ NSAA files via ‘model\_predictor.py’. It was noted, though, that the type of movement behaviour expressed in these 6-minute walk files are predominantly different than the behaviour expressed in the NSAA files, as these files only contain a fraction of the amount of movement that’s similar to walking while the majority of the other data within the file includes the subject climbing boxes, getting off the floor, hopping on one leg, etc. This variety of movement expressed in the NSAA files and thus what we required the model to learn so we can assess a new subject’s NSAA file is not helped by adding in walking data. What might help, however, is adding in NMB data of the subjects: the larger variety of movement of the subjects in NMB files is more likely to show a variety of movement contortions that would be more useful to models in order to learn generalized correlations between NSAA scores and subjects’ movements than adding in extra 6-minute walk data that doesn’t provide any additional insight to the data that we already have.

To this end, we intend to repeat MPS 13 but with using the NMB directory as the additional directory (along with the ‘NSAA’ directory) instead of ‘6minwalk-matfiles’ and ‘6MW-matFiles’. However, there is a problem with this direct approach: there exists far more NMB data than in the NSAA or either of the 6-minute walk directories. It was felt that with including the entirety of the data set would cause the models that are training from both NSAA and NMB files to prioritize learning from the NMB files more so than the NSAA files as there it would be far more conducive in the training process to focus on optimizing on the NMB data to reduce overall training loss. By way of comparison, each RNN model is trained on approximately 13,000 sequences from the NSAA data set, while we have >100,000 sequences to draw from in NMB. Furthermore, the data from natural movement is less likely to be as useful to models that needs to make inferences about new subjects’ NSAA scores than data from NSAA files due to the natural movement files not containing actual NSAA activities.

To reduce the size of the data set of the NMB directory, we add an optional argument, ‘--balance\_allmatfiles’, to ‘rnn.py’ that takes in an integer value. This value specifies the maximum number of files per subject that can be drawn from the ‘allmatfiles’ directory when we’re loading files into ‘rnn.py’. For example, setting ‘--balance\_allmatfiles=3’ results in a maximum of 3 files per short subject name to be drawn from the data set. To maintain a variety of different type of movement files used, these 3 files per subject are chosen at random from the possible subjects. As a result, instead of drawing data from 452 files we instead use 62 and, therefore, we go from using >100,000 sequences from NMB to just using ~14,000, which is much more comparable to the size of the NSAA data set. We hope that with using only this amount of NMB data that the data will complement the use of data from the NSAA data set for the models learning the how to assess new subjects’ NSAA files, as opposed to overwhelming the model with NMB and possibly impacting its ability to generalize to these new subjects.

**Results Discussion**

One thing to be reminded of at this point is that we currently only have the NMB in joint angle form. In other words, we have source ‘.mat’ files for the NMB data but these source files currently only contain the joint angle values, as opposed to also containing data for the other >10 raw measurements. Note that this is down to, at the time of writing, all the data directly from the suits in ‘.mvnx’ format having not been converted to ‘.mat’ format and is something we will hope to obtain in the near future. Hence, to make a fair comparison, we compare models that have been built solely on the NSAA directory from the joint angle measurement to those built on files from NSAA and ‘allmatfiles’, also only on joint angles. This is also why the results why the rows of the table corresponding to the ‘standard’ setup (i.e. that don’t contain ‘additional dir’ in the name) differ in results to those seen in MPS 3.

An immediate observation that we can make from the above table is the apparent disconnect between the performance for the output types concerning NSAA scores (‘acts’ and ‘overall’) and the performance for the D/HC classification output type (‘dhc’). In using the additional data from the NMB data set with ‘--balance\_allmatfiles=3’, we see that models build using this argument end up predicting two subjects as the incorrect classification (assessing both ‘D3’ and ‘D9’ as ‘HC’). Additionally, the two ‘D’ subjects that it gets correct (‘D11’ and ‘D17’) have a much lower percentage of correctly predicted sequences with the correct classification when using these models as opposed to models with the ‘standard’ setup (i.e. only using the ‘NSAA’ directory). Therefore, this implies that **using additional data in the form of NMB data results in models predicting new subjects with less accurate D/HC classifications than if we only used the ‘NSAA’ directory**. One reason for this might be due to natural movement behaviour blurring the lines between what is considered ‘healthy’ movement ability, which may come from using NMB files where the subjects aren’t performing tasks that don’t discriminate particularly well between subjects with DMD (‘D’) and those without (‘HC’); for example, NMB files where the subject is sitting, talking, or eating won’t be as useful for models to help classify the subject compared with activities that are specifically designed for the task (i.e. that can be found in files from the ‘NSAA’ directory).

However, this is not consistent with the results we see with respect to the metrics concerned with the NSAA scores: here, we see that the percentage of acts correct predicted increases for two subjects will decreasing for none when using models trained on NSAA and NMB files compared to just having trained on NSAA files. We can also see that the accumulation of the differences between the true and predicted overall NSAA scores decreases from 22 when using models trained only on NSAA to 19 when using models trained on both NSAA and NMB files. Hence, **the fact that we see improvement for the metrics concerning the NSAA scores implies that using additional NMB files along with NSAA files helps with assessing the NSAA scores (both single and overall) of new patients even though D/HC classification is worse**. One reason for this could be that, while the NMB data blurs the line between what’s considered healthy or DMD behaviour due to having more ‘unclassifiable’ activities (e.g. sitting and eating), it also would provide more activities that are in NMB that contain characteristics specific to certain overall scores or sequences of single-act scores. So while this requires further experimentation when we have all measurements available (as opposed to just using models built from joint angles as done here), we can tentatively state that **the use of supplementary data from the NMB directory ‘allmatfiles’ is preferable to only using data from the ‘NSAA’ directory when we are more concerned with performance for the ‘overall’ and ‘acts’ output type than ‘dhc’**.

**Model Predictions Set 17: Model Performance for 5 Subjects using a Single Sequence from Each Subject**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** |
| D3 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 41.18% | D | 100.0% | 19 | 27 |
| D3 (single sequence) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 19 | 28 |
| D9 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 52.94% | D | 100.0% | 22 | 25 |
| D9 (single sequence) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 52.94% | D | 100.0% | 22 | 27 |
| D11 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 70.59% | D | 100.0% | 27 | 28 |
| D11 (single sequence) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 58.82% | D | 100.0% | 27 | 28 |
| D17 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 50.0% | 33 | 28 |
| D17 (single sequence) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 94.12% | HC | 0.0% | 33 | 30 |
| HC6 | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | HC | 75.0% | 34 | 31 |
| HC6 (single sequence) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 100.0% | D | 0.0% | 34 | 32 |

As each of the models are trained on sequences rather that complete files, it stands to reason that we are only able to test subjects on prebuilt models by using sequences from the assessing file(s). To do this, the assessed file is split into sequences that are each used as input to the models that are fed to the models in batches. The results from each of these sequences of the assessed file by ‘model\_predictor.py’ are then averaged together to provide assessment for the subject for a given input type and output type; in other words, the assessments for various metrics for a single model (e.g. joint angles of the subject used to predict the overall NSAA score) are in fact the average of the results of each of the sequences from the subject fed through the model. This process repeats for each measurement we are using and each output type we are targeting to then make an accumulated prediction over all input and output types.

Here, however, we wish to look at the possibility of drawing a single sequence from the assessed file rather than dividing the source file into numerous sequences and finding the average. For example, let’s say we are assessing subject ‘D9’ on models that haven’t seen ‘D9’ during training, and let’s say we are specifically concerned with the model trained on joint angle data for the ‘overall’ output type. Traditionally, we would take corresponding subject file(s) for ‘D9’ and divide it up into sequences: if the joint angle information for subject ‘D9’ (as the file ‘D9\_jointAngle.csv’) is of shape (3000, 66), we would divide this into 50 sequences so the new data shape would be (50, 60, 66) and would feed these 50 sequences into the relevant model and average the 50 predictions for the overall NSAA score for that subject for the joint angle input type and ‘overall’ output type to get a single prediction. For this model prediction set, however, we take a different approach: from an input data shape of (3000, 66), we instead draw a single sequence that is sampled throughout the data so that every 50 rows of data we sample a row (‘frame’) of data, repeating throughout the file. Hence, from data of a shape (3000, 66) we get (60, 66).

The idea behind this was to investigate whether or not there are obvious issues with our initial assumption of being able to assess a complete file by assessing on its individual sequences and combining the results on these sequences. Therefore, if the ‘single sequence’ method is evidently superior, then we can infer that we cannot realistically assess on a complete subject file in ‘model\_predictor.py’ by splitting it into sequences. However, we do not expect this to be the case due to a single reason: the creation of a single sequence where each frame is temporally separated by 59 other frames would most likely result in each frame being temporally unrelated and so would impact the RNN’s ability to make use of its ‘memory’ within the LSTM units, as for a given raw measurements the states of the hidden units would modify the next frame of a sequence going through the network based on the assumption that the previously seen frame haven’t existed ‘1 frame in time’ prior to the current one, while in actually it would exists ’59 frames in time’ prior to the current one. Hence this expected incorrect assumption that results in the model’s LSTM units would most likely lead it to draw incorrect conclusions, so we would expect this approach to not work as well.

**Results Discussion**

It should be noted first and foremost that the assessments made for a given subject are using the exact same models for a given ‘left-out’ subject for both using a single sequence and using the standard approach of multiple sequences. For example, for the two rows concerning the ‘D3’ left-out subject based on the ‘standard’ and ‘single-sequence’ approach, they both use the same models to predict on and the only difference between them is how the ‘D3’ files for the different measurements are preprocessed by ‘model\_predictor.py’ (i.e. divided into a single sequence or not).

The results concerning the ‘overall’ and ‘acts’ output types are surprisingly similar to that of the ‘standard’ setup as used in MPS 3: in using the ‘single-act’ approach over the standard approach, we see a rise of the accumulation of the difference between the true and predicted overall NSAA scores decrease from 20 to 19. However, as this is a very small decrease and where the models improve on are for the higher-scored subjects like ‘D17’ and ‘HC6’ (where improvements aren’t as import as lower-scored subjects like ‘D3’ due to requiring our models to improve mainly in this area due to them particularly underperforming for low-scored subjects), we can’t see this as an overall improvement. However, For the percent of acts correctly predicted metric (for the ‘acts’ output type), we see a similar case as the percent improves for one subject but worsens for another (‘D3’ and ‘D11’, respectively). Therefore, **these results surprisingly indicate that the models do not predict any worse in using a single-sequence from the assessing file than using all available sequences**.

However, while the results for the ‘overall’ and ‘acts’ metrics are comparable, the results for the D/HC classification are notably worse: using single-sequences results in 2/5 of the subjects being misclassified, as opposed to only 1/5 when using the standard approach. Additionally, for the assessments made with single sequences it ‘completely’ misclassifies them: 0% of the sequences from ‘D17’ and ‘HC6’ are correctly classified as being ‘D’ or ‘HC’ sequences, respectively. This is far worse than their equivalents in using the standard approach, where even in the case of incorrectly classifying ‘D17’ as being ‘HC’ it still managed to get half of the sequences classified as correct (note that ‘model\_predictor.py’ was coded in a way that requires more than 50% of the sequences to be classified as ‘HC’ to get a overall file label of ‘HC’). Hence, **the use of the single-sequence assessments of subjects show that it is an inferior approach to using all the sequences available from the subject for D/HC classification**. Therefore, we can conclude that as a result of not improving with respect to two of the output types and noticeably worsening for the third, **we shall remain with the approach of drawing the full amount of sequences from the given assessed subject’s file(s)**.

**Model Predictions Set 18: Model Performance for 5 Subjects w/ Aggregated Assessments for Overall NSAA Score**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **True Overall Score** | **Predicted Overall Score** | **Aggregated Predicted Overall Score** |
| D3 (aggregate overall) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 19 | 27 | 27 |
| D9 (aggregate overall) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 22 | 25 | 24 |
| D11 (aggregate overall) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 27 | 28 | 28 |
| D17 (aggregate overall) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 33 | 28 | 32 |
| HC6 (aggregate overall) | ['position', 'sensorMagneticField', 'jointAngle', 'AD'] | 34 | 31 | 33 |

One thing that we have emphasized in many of the model predictions sets thus far is the relative importance of the ‘overall’ NSAA score compared to the other output types. This is primarily due to two reasons: it is the easiest of the three to evaluate generalization performance of left-out subjects, and it will most likely be the most important output of the system to specialists in order to give us information about the subject. Therefore, we wanted to add an option to ‘model\_predictor.py’ to be able to somehow combine the assessments made for ‘acts’, ‘dhc’, and ‘overall’ into a more accurate ‘overall’ for the given subject. The idea behind this is similar to the reasoning behind the combining of models trained on different measurements: we are expecting to get more accurate results from combining different models than models on their own, as we can see having been done regarding input types for MPS 4. We have seen numerous cases where, for example, subject ‘HC6’ has been correctly classified as an ‘HC’ subject with 100% of its acts correctly predicted but not having its overall NSAA score being predicted as 34. In other words, two out of the three output types have been correctly determined by the model. In this case, the overall score can be inferred by the other two: if two other output types believe the subject is a ‘HC’ subject, then it stands to reason that they would believe it has an overall NSAA score of 34, which is not what the models for the ‘overall’ output type predicts. Therefore, in cases like these it would behove us to use all three types of model output to correctly assess the overall NSAA score.

To that end, an additional function was added, ‘combine\_preds()’, to ‘model\_predictor.py’. Thus function takes in the strings that are to be outputted to console and written to ‘model\_predictions.csv’ and does the following:

1. From the relevant strings (i.e. the strings corresponding to predictions made by the models, not the true corresponding values), extract the predictions made for the predicted ‘acts’ value (i.e. list of 17 values of 0, 1, or 2), the predicted ‘D’ or ‘HC’ classification, the percentage of sequences correctly predicted, and the predicted overall NSAA scores.
2. For each of the three output types, compute an approximated overall NSAA score. For the ‘overall’ output type, this is easy: it is the value predicted by the models. For the ‘acts’ output type, again this is fairly simple: the sum of the 17 values would be the overall NSAA score predicted by models trained for the ‘acts’ output type. For the ‘dhc’ output type, however, this is slightly more complex: if the predicted D/HC label is ‘HC’, then the corresponding overall NSAA score by ‘dhc’ models would be 34. However, if it predicted ‘D’, then we find the ‘median’ value of all overall scores for the ‘D’ subjects in the ‘nsaa\_17subtasks\_matfiles.csv’ table and assign that value: therefore, the “correct” overall NSAA score for models built for the ‘dhc’ output type would be this median value.
3. Take the average of the overall NSAA scores predicted by the three types of model output to get the aggregated overall NSAA score over the three output types. It’s worth noting, however, that the value for the ‘dhc’ output type is averaged with the others in proportion to the percentage of correct predicted sequences. This is because the D/HC classification has a much narrow band of values to be added aggregated (i.e. either the median of ‘D’ subjects or 34, depending on the predicted D/HC label). For example, if we were assessing for the left-out subject ‘HC6’ and the models predicted a sequence of all 2’s for the ‘acts’ models, an overall NSAA score of 30, and a D/HC label of ‘HC’ with 75% of sequences correctly predicted, the aggregated prediction would be computed as:

This is then rounded up to an aggregated score of ‘33’, which is closer to the true overall NSAA value of 34 (as it’s an ‘HC’ subject) than by just using the ‘overall’ models, which would have predicted a ‘30’, due to taking advantage of all three output types.

**Results Discussion**

As we are only concerned with aggregating predictions made here, for the table above we run ‘model\_predictor.py’ only once per subject, as opposed to most of MPS’s where it is run once with the ‘standard’ setup and another time with some other option(s) set. Instead, we are here comparing two types of model output to each other directly: when the ‘--combine\_preds’ argument is set, both the predicted overall NSAA score that is predicted by ‘overall’ models and the aggregated predicted overall NSAA score (as discussed above) is written to file. We can see that this approach sees a noticeable improvement: the accumulative difference between the true and predicted overall NSAA score decreases from 20 when using the predicted values from the ‘Predicted Overall Score’ column to 13 when using the ‘Aggregated Predicted Overall Score’. Therefore, in the case of the 5 left-out subjects **there is a noticeable improvement of the predictable ability of the models with respect to the overall NSAA score predictions when aggregation the predictions made by all three output types**.

In particular, we don’t see any real improvement for the subjects with a low or ‘medium’ overall NSAA score. However, when we apply this to subjects that have a higher score (e.g. ‘D17’ or ‘HC6’), the results show much more accurate overall NSAA scores. This is due to the performance of the ‘acts’ and ‘dhc’ models for these subjects being particularly good, and so helps compensate and ‘bring up’ the scores predicted by ‘overall’ models that are lower than they should be. Additionally, another important point to draw is that we don’t see any worsening of the predictions made for the overall NSAA scores for lower scoring subjects; rather, simply a non-improved value. Therefore we can say that based on the above results that **the aggregation of the output types to estimate the overall NSAA score is a good approach to use for subjects with a variety of degrees of severity of their condition**.

**Model Predictions Set 19: Model Performance for 5 Subjects w/ All Data Used for Training**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **File Name** | **Measurements tested** | **Percent of acts corrected predicted** | **Predicted D/HC Label** | **Percent of correct predicted sequences** | **True Overall Score** | **Predicted Overall Score** | **Aggregated Predicted Overall Score** |
| D3 (additional dir = allmatfiles) (aggregate overall) | ['jointAngle'] | 47.06% | HC | 48.44% | 19 | 27 | 30 |
| D3 (additional dir = allmatfiles) (aggregate overall) (no testset) | ['jointAngle'] | 47.06% | D | 55.13% | 19 | 27 | 28 |
| D9 (additional dir = allmatfiles) (aggregate overall) | ['jointAngle'] | 58.82% | HC | 34.38% | 22 | 24 | 29 |
| D9 (additional dir = allmatfiles) (aggregate overall) (no testset) | ['jointAngle'] | 82.35% | HC | 50.0% | 22 | 26 | 27 |
| D11 (additional dir = allmatfiles) (aggregate overall) | ['jointAngle'] | 82.35% | D | 52.66% | 27 | 27 | 28 |
| D11 (additional dir = allmatfiles) (aggregate overall) (no testset) | ['jointAngle'] | 82.35% | D | 51.25% | 27 | 26 | 27 |
| D17 (additional dir = allmatfiles) (aggregate overall) | ['jointAngle'] | 82.35% | D | 67.19% | 33 | 27 | 27 |
| D17 (additional dir = allmatfiles) (aggregate overall) (no testset) | ['jointAngle'] | 94.12% | D | 68.75% | 33 | 28 | 29 |
| HC6 (additional dir = allmatfiles) (aggregate overall) | ['jointAngle'] | 100.0% | HC | 84.77% | 34 | 31 | 33 |
| HC6 (additional dir = allmatfiles) (aggregate overall) (no testset) | ['jointAngle'] | 100.0% | HC | 83.59% | 34 | 32 | 33 |

Having established that provided additional data to the models trained on NSAA in the form of a sample of files from the natural movement behaviour data set ‘allmatfiles’ along with the preference to use the ‘aggregation’ option to make an aggregate overall NSAA score prediction, we now turn our attention to what has been an oversight amongst most of the model predictions sets undertaken thus far. The default ‘train-test-split’ ratio for training the models up until now has been 0.2, which means that 80% of the data that is provided to the models as training data and 20% is used as test data. This was what was used during the experiment sets to assess the models on performance for various measurement types, sequence lengths, and so on. However, for many of the model predictions set we also leave out certain subjects’ data for certain models, and thus is the data that we assess on, not the 20% of the data that is left out of the data set after the ‘left-out’ subject’s data is removed from the data set. Therefore, all that leaving out this 20% from the training set does is deprive us of 20% of the data that could be used to help train the model, which would hopefully achieve better generalization performance while not impacting our ability to test on ‘left-out’ subjects, as these subjects would still not be familiar to the models they are assessed on even with the inclusion of the models’ test sets.

To achieve this, we simply add an argument to ‘rnn.py’, ‘--no\_testset’, that sets the train-test ratio used by the ‘train\_test\_split()’ function to 0. Upon setting this, all of the data available (that is not covering the ‘left-out’ subject in question) is used when the RNN object calls its ‘train()’ method. As a result of having no test data, the ‘predict()’ method is not called, along with no results being written to the ‘RNN outputs’ directory (as there is no test data there can be no predictions made of which to compare the data): when the ‘train()’ method is complete, the program that created this model finishes. Additionally, the ‘model\_predictor.py’ script is modified to be able to call models that have had the ‘--no\_testset’ optional argument set.

**Results Discussion**

Having noted in model predictions set 18 previously that the aggregate overall NSAA score that can be determined is the most useful output metric to us (as it is the target we’re most interested in optimizing, along with being a good reflection of the performance from all three output types), here we exclusively focus on the performance with respect to this metric. It’s worth noting that both types of models for a given ‘left-out’ subject are trained on both ‘NSAA’ and the ‘allmatfiles’ directories; hence, we can only use the joint angle measurement as this is the only form that ‘allmatfiles’ currently takes (natural movement behaviour data with more than just joint angles will be discussed later when the data is obtained). The only difference between the two types of rows within the table is that the second row in each group of two are trained on models that have the full 100% of the data used, rather than just the 80% partitioned for the training set. It should also be noted that, in both cases, the subject is being assessed by ‘model\_predictor.py’ in exactly the same way, only using either models trained on 80% of the available data or models trained on 100% of the available data.

Unsurprisingly, **we see more promising results with respect to the aggregated predicted overall score for models trained on 100% of the data** (i.e. option ‘--no\_testset’ is used). There is an overall decrease in the cumulative difference between the true and aggregate predicted overall NSAA scores from 26 to 19, which is a fairly significant boost to performance. This is most likely down to the usefulness of this additional 20% of the data use, where we see significant improvements with respect to the ‘acts’ and ‘overall’ output types through the percent of acts correctly predicted and percent of correctly predicted sequences metrics. This additional data therefore most likely helps the models to generalize to new, unseen subjects better due to being exposed to new sequences with their corresponding labels. Hence, **we therefore shall set the models to use all available data when training models to assess on unseen subjects via ‘model\_predictor.py’**.