Assignment 1 Report

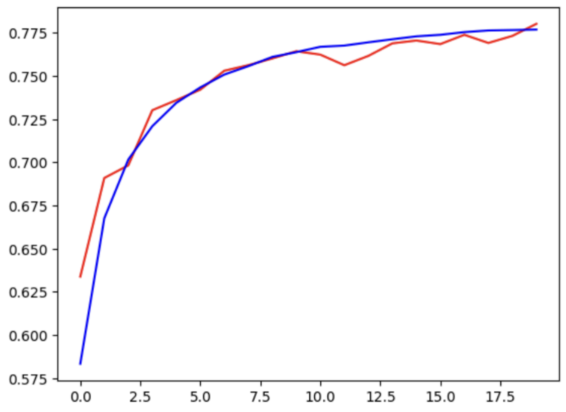
Milestone 1

MLP Classifier

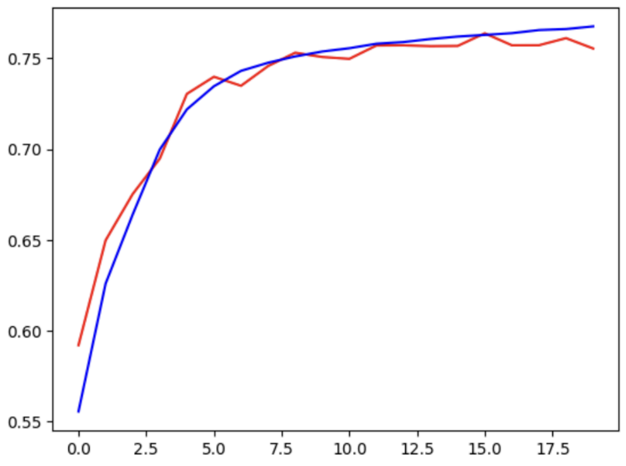
We test multiple simple MLP classifiers on the data in consideration and achieve the following performance results.

Low Noise (Target column)

A simple MLP classifier achieved a **77.46%** accuracy on the data. After using AdaBoost to remove some noise from the dataset the accuracy improved to **77.69%**.



However, to improve the performance we employed a technique called bagging. We use a Bagging Classifier with 100 estimators to achieve an accuracy of **76.68%**

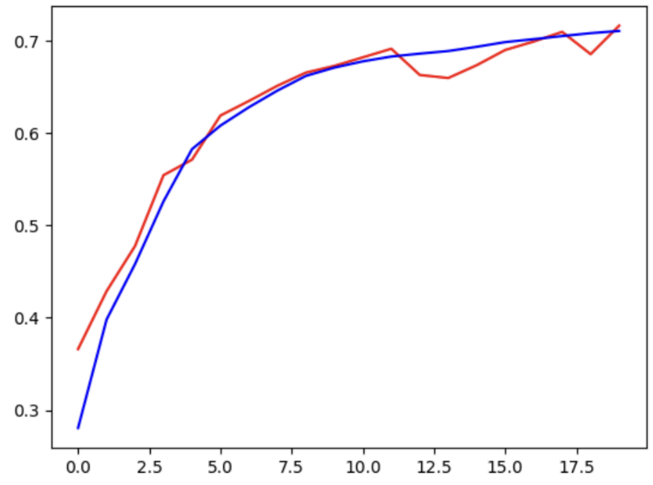


In an attempt to further tackle the noise in data we tried creating an ensemble of MLP classifiers and then use the multiple classifiers to make predictions on the validation data. With this technique we managed to achieve the highest accuracy of **78.50%**.

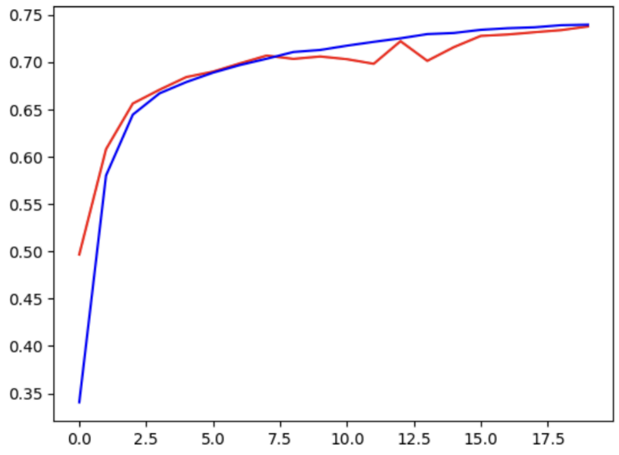
Low Noise (Era column)

For the era column of the dataset we tried the same models as for the target column and achieved following performance:

- Simple MLP:**69.63%**
- Simple MLP (with AdaBoost to remove noise):**71.09%**



- MLP + Bagging:**74.06%**

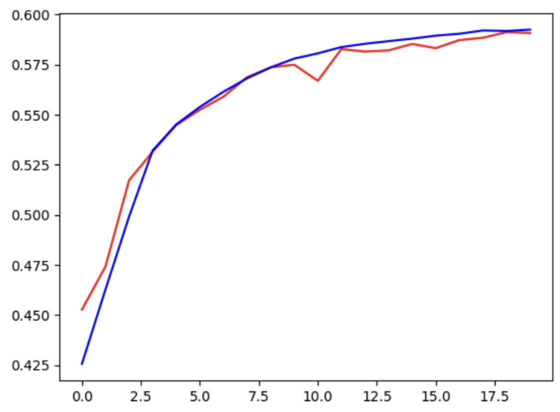


- MLP Ensemble + Bagging:**75.88%**

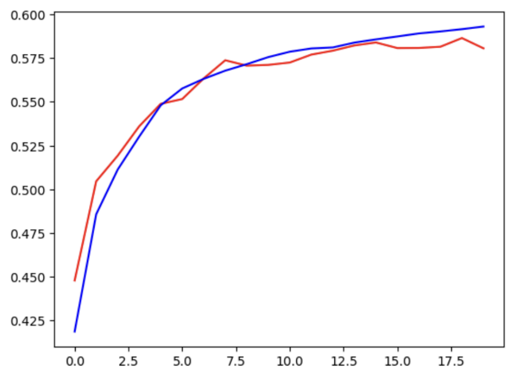
High Noise (target column)

For the target column of the dataset we tried the same models as for the target column and achieved following performance:

- Simple MLP:**59.39%**
- Simple MLP (and AdaBoost to remove noise):**59.25%**



- MLP + Bagging:**59.30%**

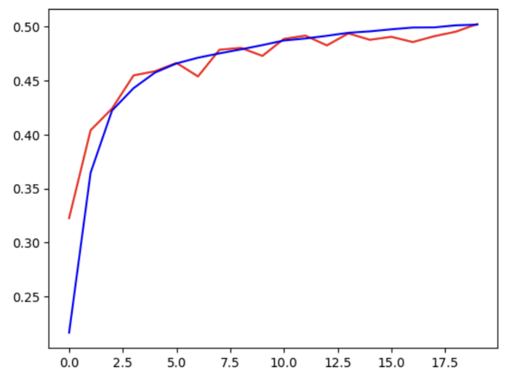


- MLP Ensemble + Bagging:**60.26%**

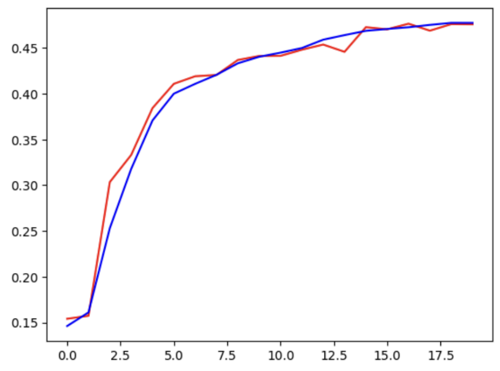
High Noise (era column)

For the era column of the dataset we tried the same models as for the target column and achieved following performance:

- Simple MLP:**47.69%**
- Simple MLP (and AdaBoost to remove noise):**50.17%**



- MLP + Bagging:**47.48%**



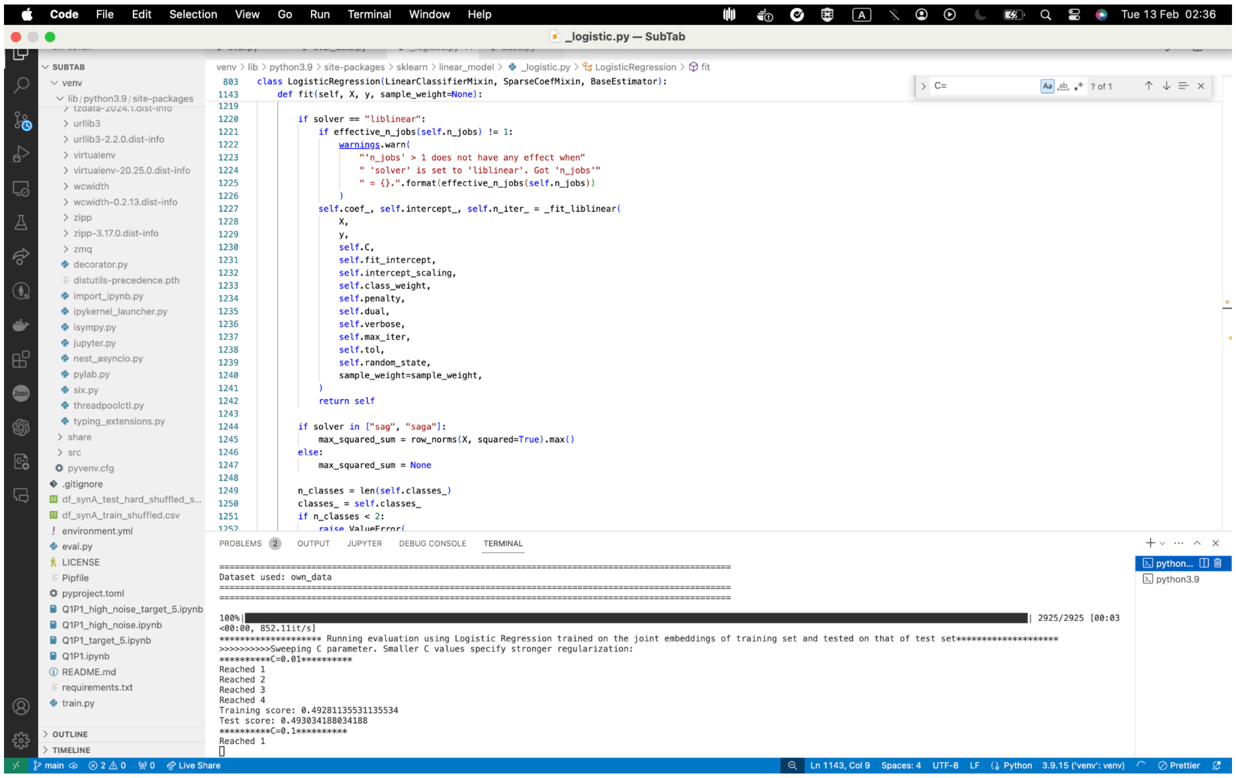
- MLP Ensemble + Bagging:**49.72%**

Methodology	**Low Noise (Target Column)**	**Low Noise (Era Column)**	**High Noise (Target Column)**	**High Noise (Era Column)**
Simple MLP	77.46%	69.63%	59.39%	47.69%
Simple MLP + AdaBoost	77.69%	71.09%	59.25%	50.17%
MLP + Bagging	76.68%	74.06%	59.30%	47.48%
MLP Ensemble + Bagging	**78.50%**	**75.88%**	**60.26%**	49.72%
SubTab	-	-	-	**61.03%**

SubTab Implementation

The SubTab paper works by dividing tabular data into multiple subsets and considering each subset as a different view. The model then learns representations by reconstructing the entire data from these subsets. This approach helps capture underlying patterns in data better than traditional methods, leading to improved performance in tasks like classification and clustering.

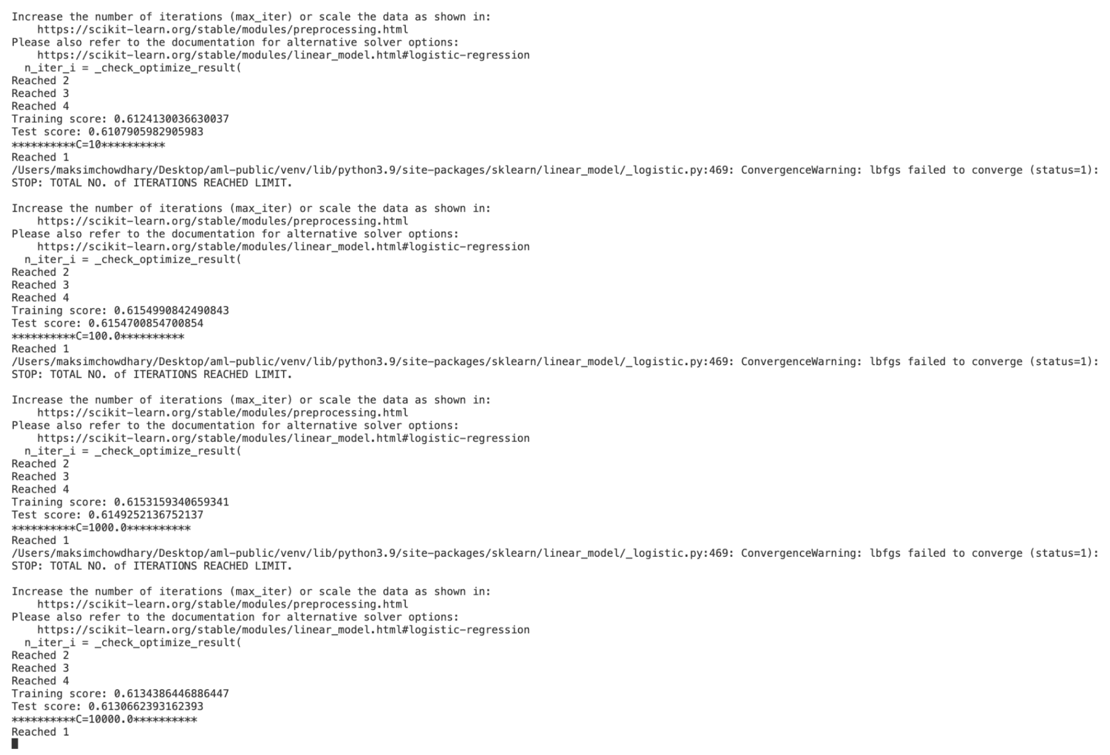
I tried using SubTab implementation on our data and achieved a higher performance. We had to make certain modifications to the code and after trying different hyperparameters we managed to achieve a highest performance of 61.3% on the high noise data for the era column. For differnign value of hyperparameter C the performance varied.



```python
class LogisticRegression(LinearClassifierMixin, SparseCoefMixin, BaseEstimator):
    def fit(self, X, y, sample_weight=None):

        if solver == "liblinear":
            if effective_n_jobs(self.n_jobs) != 1:
                warnings.warn(
                    "'n_jobs' > 1 does not have any effect when"
                    " 'solver' is set to 'liblinear'. Got 'n_jobs'"
                    " = {}.".format(effective_n_jobs(self.n_jobs))
                )
            self.coef_, self.intercept_, self.n_iter_ = _fit_liblinear(
                X,
                y,
                self.C,
                self.fit_intercept,
                self.intercept_scaling,
                self.class_weight,
                self.penalty,
                self.dual,
                self.verbose,
                self.max_iter,
                self.tol,
                self.random_state,
                sample_weight=sample_weight,
            )
            return self

        if solver in ["sag", "saga"]:
            max_squared_sum = row_norms(X, squared=True).max()
        else:
            max_squared_sum = None

        n_classes = len(self.classes_)
        classes_ = self.classes_
        if n_classes < 2:
            raise ValueError(
```

```
==========================================================================================
Dataset used: own_data
==========================================================================================
==========================================================================================

100%|████████████████████████████████████████████████████████████| 2925/2925 [00:03
<00:00, 852.11it/s]
******************** Running evaluation using Logistic Regression trained on the joint embeddings of training set and tested on that of test set********************
>>>>>>>>>>Sweeping C parameter. Smaller C values specify stronger regularization:
**********C=0.01**********
Reached 1
Reached 2
Reached 3
Reached 4
Training score: 0.49281135531135534
Test score: 0.493034188034188
**********C=0.1**********
Reached 1
```

When C == 0.01, an accuracy of 49.30% was achieved.



```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Reached 2
Reached 3
Reached 4
Training score: 0.6124130036630037
Test score: 0.6107905982905983
**********C=10**********
Reached 1
/Users/maksimchowdhary/Desktop/aml-public/venv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Reached 2
Reached 3
Reached 4
Training score: 0.6154990842490843
Test score: 0.6154700854700854
**********C=100.0**********
Reached 1
/Users/maksimchowdhary/Desktop/aml-public/venv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Reached 2
Reached 3
Reached 4
Training score: 0.6153159340659341
Test score: 0.6149252136752137
**********C=1000.0**********
Reached 1
/Users/maksimchowdhary/Desktop/aml-public/venv/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Reached 2
Reached 3
Reached 4
Training score: 0.6134386446886447
Test score: 0.6130662393162393
**********C=10000.0**********
Reached 1
█
```

When C == 10000, an accuracy of 61.30% was achieved.

I wasn’t able to run SubTab for all other combinations of data due to insufficient computation resources on the local PC. This shows that in order to achieve better performance we sometimes need to trade off the speed and cost of training.

Conclusion

The MLP classifiers seem to work best in an ensemble since the ensemble of classifiers may be able to distinguish noise from substance to a better degree. However, in the case of highly noisy data the ensemble’s performance is not as significantly better than a single classifier. This must be noted and considered when employing models in real life since the training time and cost for an ensemble may eb multiple times higher than that for a single classifier. The cost-benefit analysis of the model to be employed would be useful in such a scenario.

However, SubTab was able to boost up the accuracy significantly hence proving its power for training on noisy data. We were able to jump from an accuracy of~49% to an accuracy of ~61% by using SubTab instead of an ensemble of MLP classifiers with AdaBoost.

Milestone 2

TabPFN Implementation

Low Noise (Target column)

The TabPFN paper has a limitation that for large datasets it starts requiring memory in quadratic space complexity. For this reason, training TabPFN on our data becomes a challenge. For our data, what I did was train the data on a smaller sample of the dataset in order to not run out of memory on the local PC. After trying with different sample sizes, the optimal sample size came out to be **approximately 1000 samples** (70-30 train-test split).

The TabPFN model in this case gave an **accuracy of 73.3% for the Target columns**

Low Noise (Era column)

The Era column in our dataset has 12 classes. Resultantly, on top of the sample size restriction for TabPFN I also are challenged with the max. number of classes restriction for TabPFN. The way TabPFN is designed it allows training to occur for classification tasks with a **maximum number of 10 distinct classes**.

In order to deal with it I tested two separate strategies: grouped rare classes and created separate classifiers for **two groups of 6 classes.**

In grouping rare classes, I grouped the 4 of rarest classes into two and as a result got a classification task with 10 classes (since the original “era” column has 12 classes). By doing this , and training TabPFN on the new data, I achieved an accuracy of 64.66%.

In the other case, I made two separate classifiers: one that was trained on classes 0-5 and another that trained on 6-11. Two separate TabPFN classifiers Ire trained on corresponding data and during prediction the prediction of both Ire used. For combining results of both, I use a one-vs-rest **approach** of ensemble methods (treat each group as a binary classification problem (the group vs the rest). For a given input, if a classifier predicts 'rest', you pass the input to the next classifier. If it predicts one of its classes, that's your final prediction).

The classification accuracy thus achieved comes out to be **79.33 %.**

Lastly, since TabPFN **does not use the complete data** provided to us (due to its space complexity constraints) I train an ensemble of 40 classifiers each trained on a sample of 2000 rows (taken out using bootstrapping method). The ensemble is trained on corresponding datasets and the outputs from the classifiers are combined during testing using a simple voting system to give predictions.

Surprisingly, this approach led to the **lolst accuracy of all - 38.66%.** A possible reason for this is **Diversity vs. Accuracy Trade-off in Ensembles**: Ensembles work best when individual models are both accurate and diverse. If the models are too similar (highly correlated errors), they will all make the same mistakes, reducing the effectiveness of the ensemble. Conversely, if they are too diverse but individually inaccurate, their predictions may not improve overall accuracy when combined.

High Noise (target column)

The reported results for the corresponding model on the high noise data is 49.3%.

High Noise (era column)

The reported results for the corresponding models on the high noise data are:

- Rarest Class grouping: **44.66%**
- Paired classifier setting: **58.3%**
- Ensemble of paired classifiers: **20.6%**

Methodology	**Low Noise - Target Column**	**Low Noise - Era Column**	**High Noise - Target Column**	**High Noise - Era Column**
Original Sample Size	**73.3%**		**49.3%**	
Grouped Rare Classes		64.66%		44.66%
Paired Classifiers		**79.33%**		**58.3%**
Ensemble of 40 Classifiers		38.66%		20.6%

Conclusion

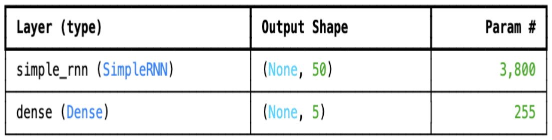
From the above it is fair to conclude that the TabPFN model works best when employed singularly (outside of an ensemble setting). This most probably is caused by the Diversity vs Accuracy tradeoff of ensembles. Other than that for tackling tasks which have more than 10 classes, a division of classification tasks into multiple classifiers each focusing on certain classifiers may be an efficient technique to overcome the “max class” limit of the TabPFN model. Last but not the least, the training data sample size constraint of the TabPFN model may become a hurdle as the training data becomes larger and larger. Since the memory needed scales quadratically with the training data size, for larger datasets using TabPFN may not be a convenient and efficient approach.

Sequential Implementations

Since I now treat the data as sequential, there is no need to reshuffle the data (as done previously). In order to train on this sequential data I test arious RNNs and LSTMs with certain modifications to be trained on the noisy data.

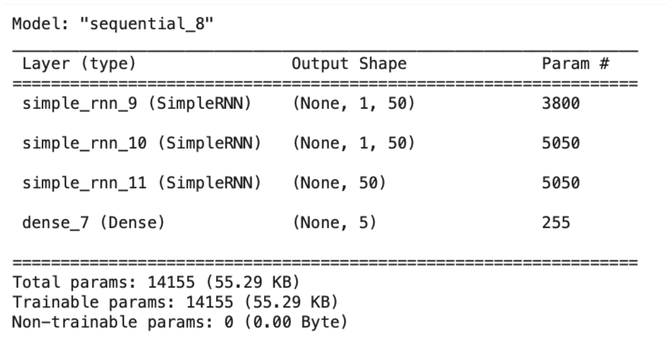
Low Noise (Target column)

For the target column I first train a simple RNN with a single RNN layer and achieve a validation accuracy of **70.82%.**

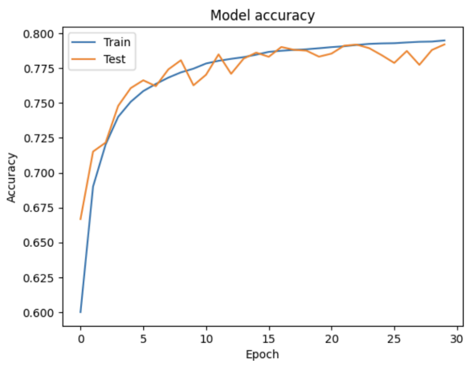


Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	3,800
dense (Dense)	(None, 5)	255

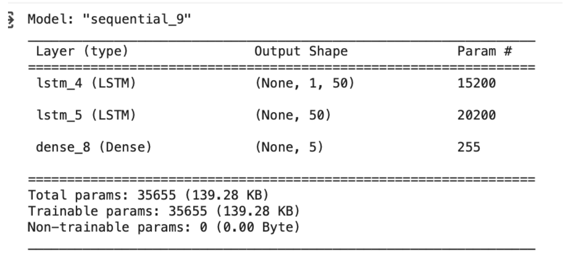
I then increase the complexity of the network and achieve an accuracy of **79.50%**



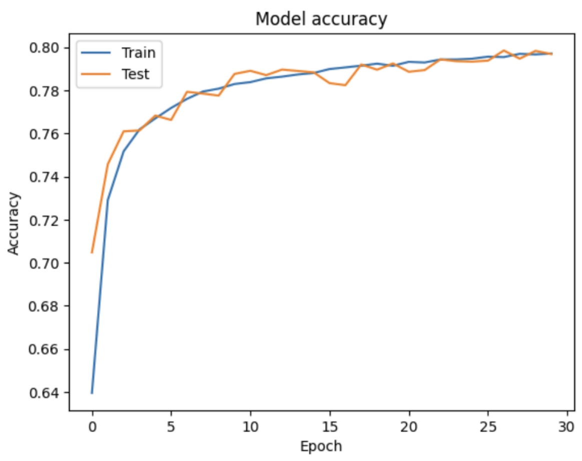
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_9 (SimpleRNN)    (None, 1, 50)             3800

 simple_rnn_10 (SimpleRNN)   (None, 1, 50)             5050

 simple_rnn_11 (SimpleRNN)   (None, 50)                5050

 dense_7 (Dense)             (None, 5)                 255

=================================================================
Total params: 14155 (55.29 KB)
Trainable params: 14155 (55.29 KB)
Non-trainable params: 0 (0.00 Byte)
```



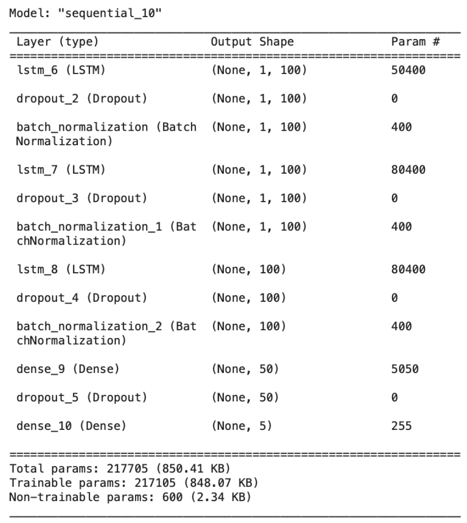
For a simple LSTM with two layers, the achieved accuracy is **79.86%** (a bit higher than a complex RNN)



```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 1, 50)             15200

 lstm_5 (LSTM)               (None, 50)                20200

 dense_8 (Dense)             (None, 5)                 255

=================================================================
Total params: 35655 (139.28 KB)
Trainable params: 35655 (139.28 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



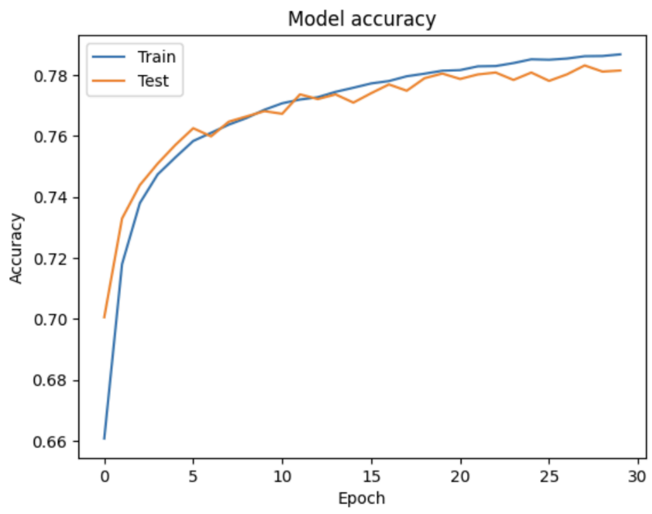
Since the data is noisy and unclean, in order to tackle the noise, I tried adding a couple of Dropout layers and regularization. HoIver, the output of the same was not any better than a simple LSTM. The achieved accuracy was **53.77%**



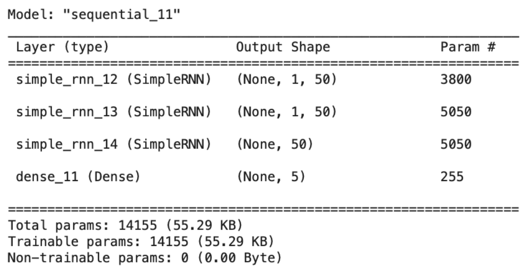
```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 1, 100)            50400

 dropout_2 (Dropout)         (None, 1, 100)            0

 batch_normalization (Batch  (None, 1, 100)            400
 Normalization)

 lstm_7 (LSTM)               (None, 1, 100)            80400

 dropout_3 (Dropout)         (None, 1, 100)            0

 batch_normalization_1 (Bat  (None, 1, 100)            400
 chNormalization)

 lstm_8 (LSTM)               (None, 100)               80400

 dropout_4 (Dropout)         (None, 100)               0

 batch_normalization_2 (Bat  (None, 100)               400
 chNormalization)

 dense_9 (Dense)             (None, 50)                5050

 dropout_5 (Dropout)         (None, 50)                0

 dense_10 (Dense)            (None, 5)                 255

=================================================================
Total params: 217705 (850.41 KB)
Trainable params: 217105 (848.07 KB)
Non-trainable params: 600 (2.34 KB)
_________________________________________________________________
```

In order to further tackle the noise in the data I tried employing feature selection with a Random Forest classifier to only use the columns which would be most relevant for the classification task on hand. I ran the RFC to select only some of the most important features and ran the data through an LSTM network with dropout layers to achieve an accuracy of **77.46%.**

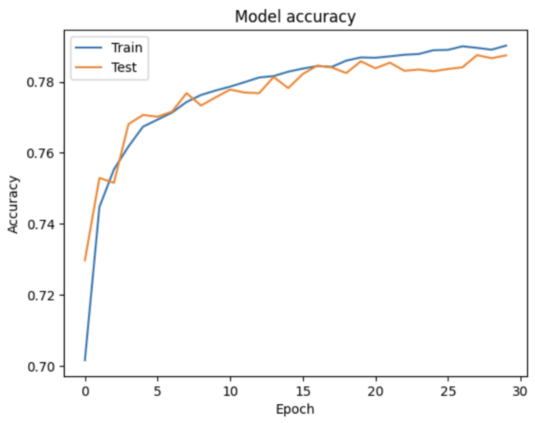
For an LSTM with no Dropout layers rose to **78.15**%



Lastly, we employed feature selection with an RNN network to achieve the accuracy of **78.74%**

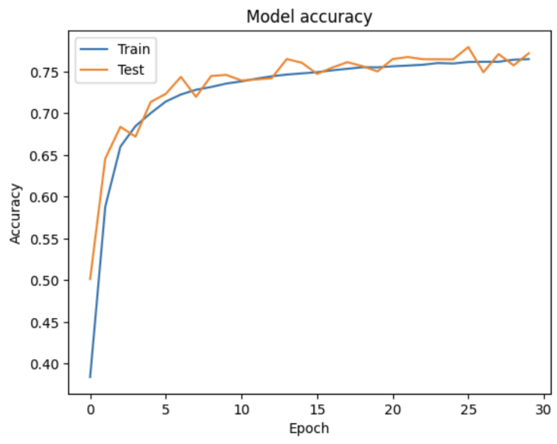


```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_12 (SimpleRNN)   (None, 1, 50)             3800

 simple_rnn_13 (SimpleRNN)   (None, 1, 50)             5050

 simple_rnn_14 (SimpleRNN)   (None, 50)                5050

 dense_11 (Dense)            (None, 5)                 255

=================================================================
Total params: 14155 (55.29 KB)
Trainable params: 14155 (55.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

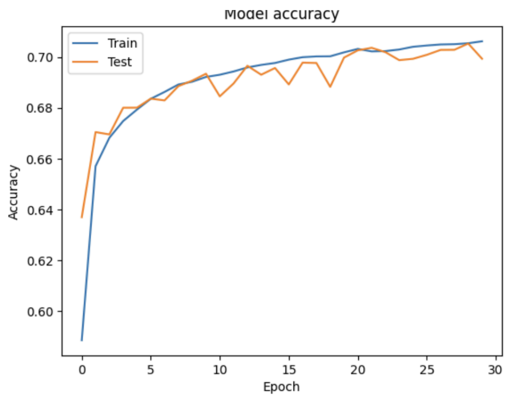


Low Noise (Era column)

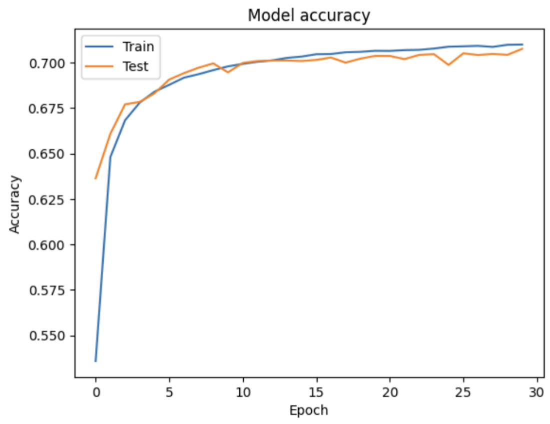
For running tests on the era column we tested the strategies that worked best for the target column on the dataset in consideration. We first run a simple RNN to achieve a validation accuracy of **77.18%**.



RNN with feature selection achieved an accuracy of **69.93%.**



The LSTM network with Feature selection with RF Classifier (No Dropout) achieved accuracy of 70.76%



High Noise (Target column)

On the high noise data for the target column the reported accuracies are as following:

- Simple RNN(1 layer): **54.10%**
- Simple RNN (3 layers): **60.97%**
- Simple LSTM: **60.19%**
- LSTM with Dropout and Regularization: **24.13%**
- LSTM with RFC Feature Selection and Dropout:**59.60%**
- **LSTM with RFC Feature Selection and no Dropout:60.43%**
- RNN with RFC Feature selection and no Dropout:**60.25%**

High Noise (Era column)

On the high noise data for the era column the reported accuracies are as following:

- Simple RNN (3 layers): **51.13%**
- LSTM with RFC Feature Selection and no Dropout:**45.62%**
- RNN with RFC Feature selection and no Dropout:**45.41%**
- **Simple LSTM: 60.69%**

Methodology	Low Noise - Target Column (%)	Low Noise - Era Column (%)	High Noise - Target Column (%)	High Noise - Era Column (%)
Simple RNN (1 Layer)	70.82	**77.18**	54.10	51.13
RNN (Increased Complexity)	79.50	-	-	-
Simple LSTM (2 Layers)	79.86	-	60.19	**60.69%**
LSTM (Dropout & Regularization)	53.77	-	24.13	-
LSTM (RFC Feature Selection & Dropout)	77.46	-	59.60	-
LSTM (RFC Feature Selection, No Dropout)	78.15	70.76	**60.43**	45.62
RNN (RFC Feature Selection, No Dropout)	**78.74**	69.93	60.25	45.41

Conclusion

From above we can conclude that simple models (like LSTMs) tend to work better with high noise data (since they do not overfit on the training data) while for low noise data more complex models tend to do better. Regularization and Dropout layers do not seem to significantly increase the accuracy of the model.