Week 2 Report

Takeaways from Week 1 Meeting:

- GradientBoostingClassifier macro F1: 0.706
- MLPClassifier macro F1: 0.711
- The model is overfitting the data. Explore ways to mitigate this
- Investigate why particularly TWF's performance is bad

Possible ways to mitigate overfitting:

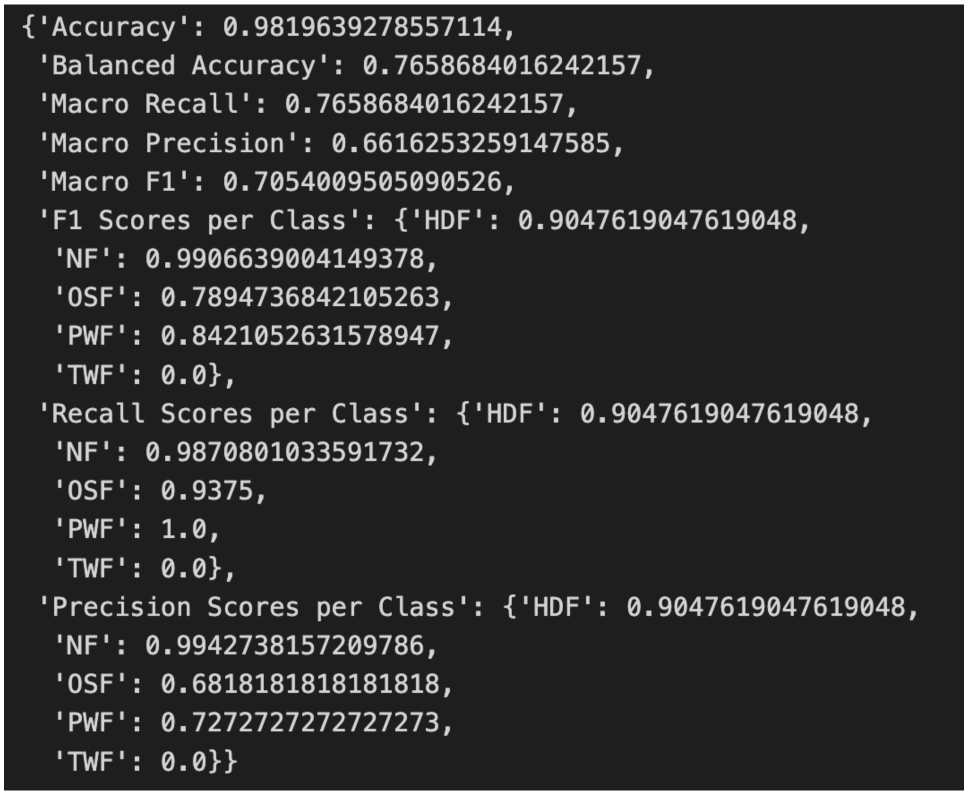
- L1, L2 regularisation, XGBoost, undersampling

My approach:

- Implement pipeline with XGBoost. It has parameters alpha and lambda for L1, L2 regularisation.
- MLPClassifier has alpha parameter for regularisation

XGBoost Implementation

- Had to encode target classes from strings to ints. Used a label encoder



```
{'Accuracy': 0.9819639278557114,
 'Balanced Accuracy': 0.7658684016242157,
 'Macro Recall': 0.7658684016242157,
 'Macro Precision': 0.6616253259147585,
 'Macro F1': 0.7054009505090526,
 'F1 Scores per Class': {'HDF': 0.9047619047619048,
  'NF': 0.9906639004149378,
  'OSF': 0.7894736842105263,
  'PWF': 0.8421052631578947,
  'TWF': 0.0},
 'Recall Scores per Class': {'HDF': 0.9047619047619048,
  'NF': 0.9870801033591732,
  'OSF': 0.9375,
  'PWF': 1.0,
  'TWF': 0.0},
 'Precision Scores per Class': {'HDF': 0.9047619047619048,
  'NF': 0.9942738157209786,
  'OSF': 0.6818181818181818,
  'PWF': 0.7272727272727273,
  'TWF': 0.0}}
```

- The above results show that the performance did not increase. This is because it used default values for alpha and lambda
  - alpha=0, lambda=1 (possible values: [0,inf])
- After fine tuning with the following candidates:

```
fine_tune_params = {
'model__n_estimators': [100],
'model__max_depth': [None, 2, 6, 20],
'model__lambda': [0, 0.001, 0.01, 0.1, 1, 10, 100],
'model__alpha': [0, 0.001, 0.01, 0.1, 1, 10, 100],
'model__eta': [0.2, 0.3, 0.4],
}
```

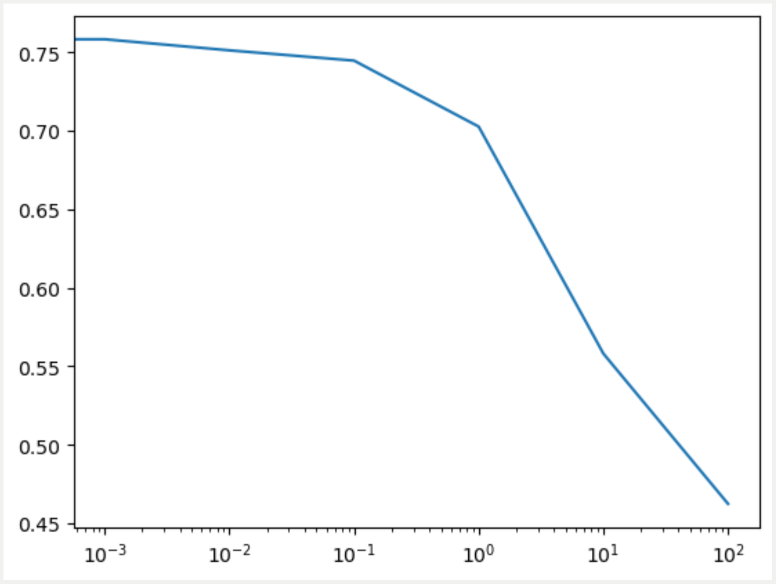
  - F1 score increased to 0.706 (very marginal increase)
    - Parameters:
      - {'model__alpha': 0.001, 'model__eta': 0.2, 'model__lambda': 0, 'model__max_depth': None, 'model__n_estimators': 100}

Checking if model is still overfitting:

F1 score:

    - training data: 0.998
    - test data: 0.706
- Conclusion: Model is clearly still overfitting. Training accuracy much higher than test accuracy.

💡 One possible reason for this is that the fine tuning pipeline found low values of alpha and lambda as the best parameters. These low values of regularisation parameters does not mitigate the overfitting effect. So why is the fine tuning pipeline selecting these values. Maybe higher values of regularisation will perform better.
No, the graph below shows that this is not the vase. X-axis is regularisation parameter, y-axis is F1 score. F1 score reduces as regularisation increases.
Therefore regularisation is not the solution



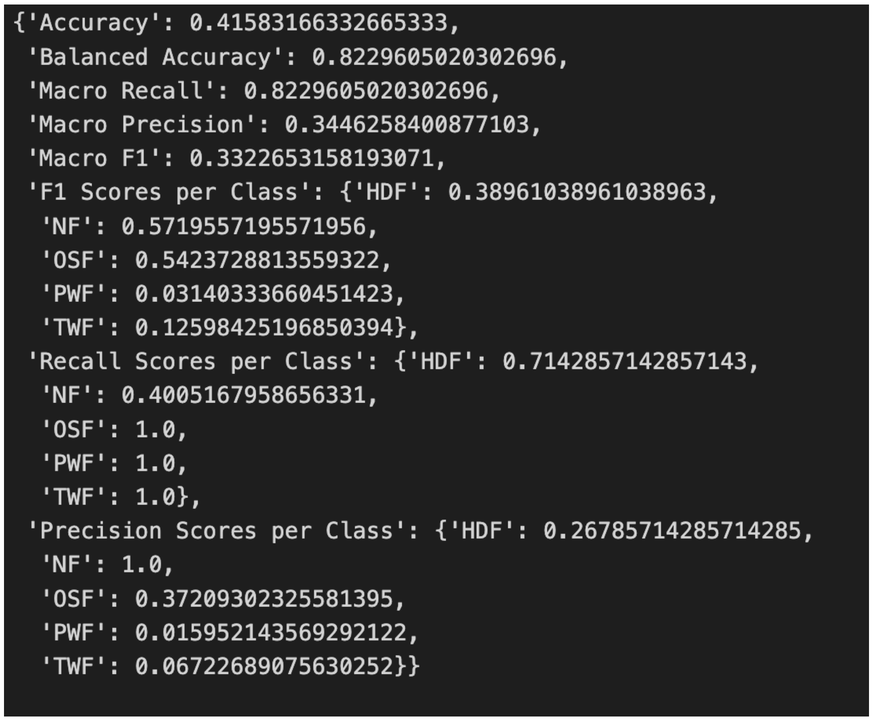
Adding regularisation to Neural Net

- Improved F1 score 0.717 with alpha=0.01 (again marginal improvement only)

- So is the model still overfitting?
    - Suprisingly not as much
        - Training F1 score: 0.82
        - Testing F1 score: 0.717

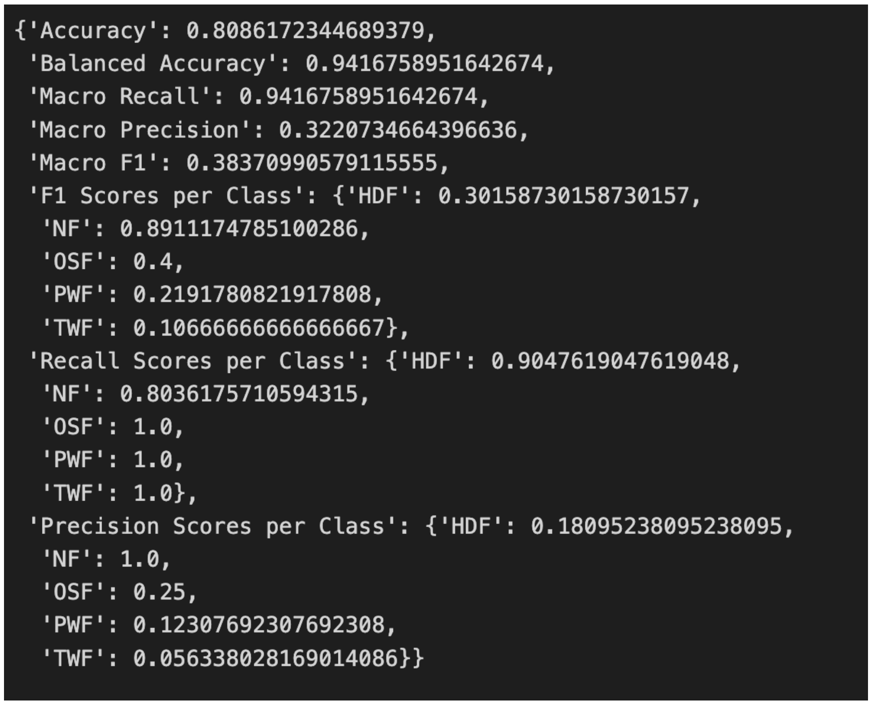
Undersampling:

Near Miss



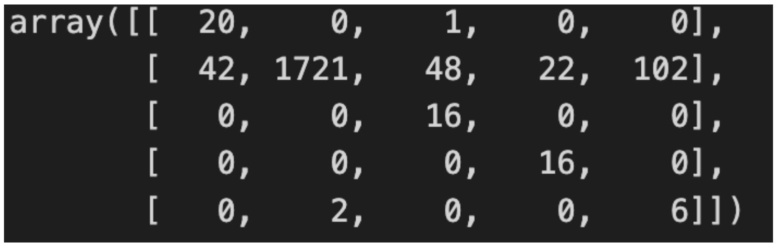
```
{'Accuracy': 0.41583166332665333,
 'Balanced Accuracy': 0.8229605020302696,
 'Macro Recall': 0.8229605020302696,
 'Macro Precision': 0.3446258400877103,
 'Macro F1': 0.3322653158193071,
 'F1 Scores per Class': {'HDF': 0.38961038961038963,
  'NF': 0.5719557195571956,
  'OSF': 0.5423728813559322,
  'PWF': 0.03140333660451423,
  'TWF': 0.12598425196850394},
 'Recall Scores per Class': {'HDF': 0.7142857142857143,
  'NF': 0.4005167958656331,
  'OSF': 1.0,
  'PWF': 1.0,
  'TWF': 1.0},
 'Precision Scores per Class': {'HDF': 0.26785714285714285,
  'NF': 1.0,
  'OSF': 0.37209302325581395,
  'PWF': 0.015952143569292122,
  'TWF': 0.06722689075630252}}
```

Random



```
{'Accuracy': 0.8086172344689379,
 'Balanced Accuracy': 0.9416758951642674,
 'Macro Recall': 0.9416758951642674,
 'Macro Precision': 0.3220734664396636,
 'Macro F1': 0.3837099057911555,
 'F1 Scores per Class': {'HDF': 0.30158730158730157,
  'NF': 0.8911174785100286,
  'OSF': 0.4,
  'PWF': 0.2191780821917808,
  'TWF': 0.10666666666666667},
 'Recall Scores per Class': {'HDF': 0.9047619047619048,
  'NF': 0.8036175710594315,
  'OSF': 1.0,
  'PWF': 1.0,
  'TWF': 1.0},
 'Precision Scores per Class': {'HDF': 0.18095238095238095,
  'NF': 1.0,
  'OSF': 0.25,
  'PWF': 0.12307692307692308,
  'TWF': 0.056338028169014086}}
```

- In both the undersampling methods, the recall scores look great but precision has decreased
- Confusion Matrix:



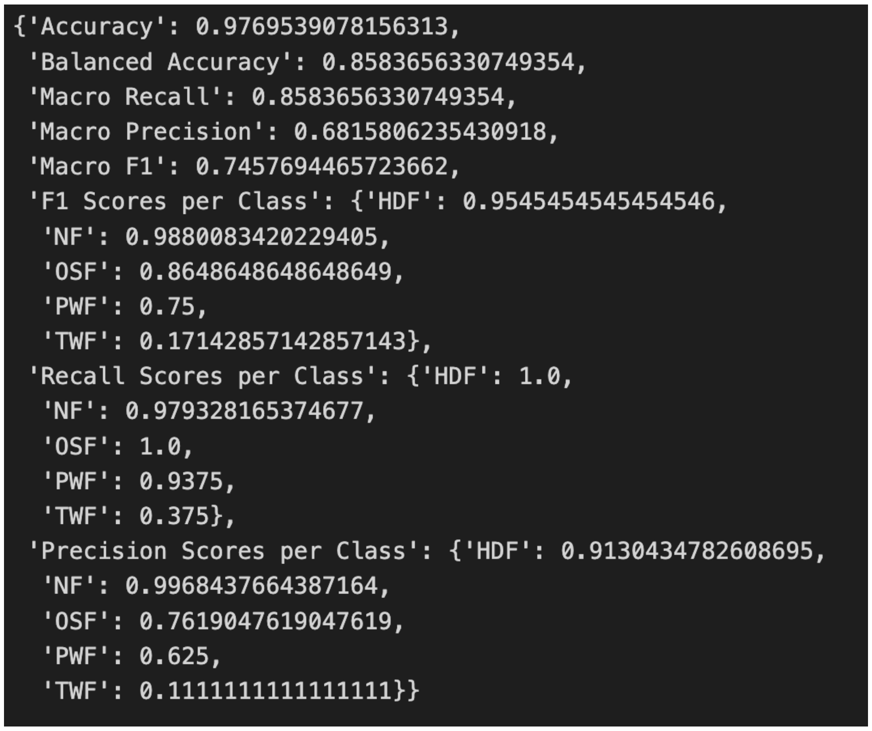
```
array([[  20,    0,    1,    0,    0],
       [  42, 1721,   48,   22,  102],
       [   0,    0,   16,    0,    0],
       [   0,    0,    0,   16,    0],
       [   0,    2,    0,    0,    6]])
```

Combining OverSampling and Undersampling :SMOTETOMEK

Tomek Links are pairs of instances (one from the minority class and one from the majority class) that are nearest neighbors of each other but belong to different classes. Removing these pairs can help in clarifying the decision boundaries between classes, particularly the minority class.

Used a custom sampling strategy:

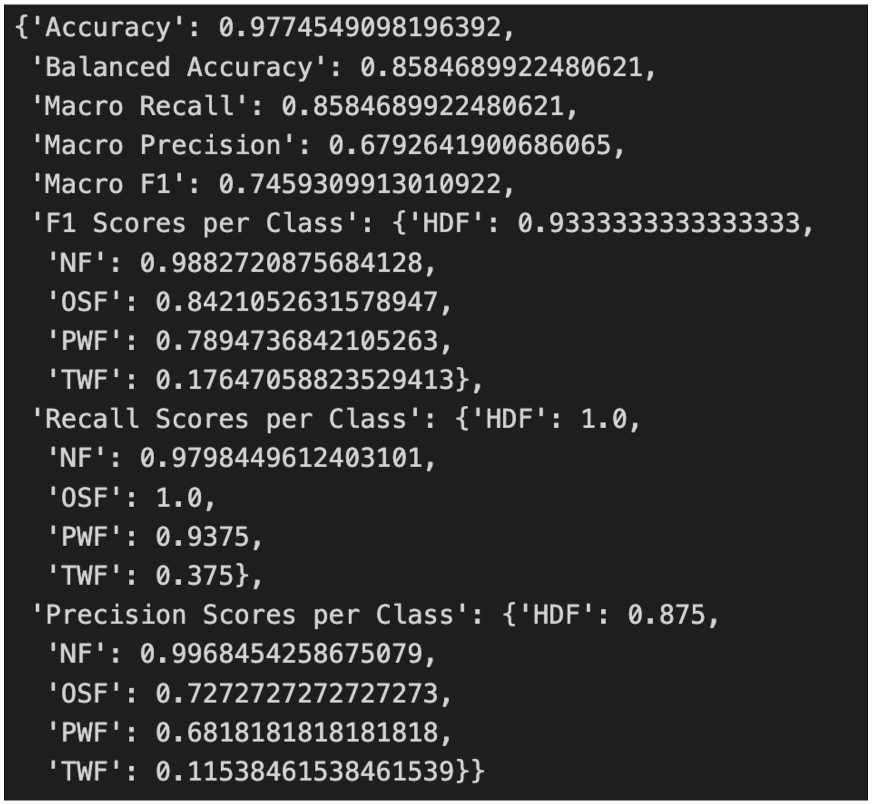
sampling_strategy = {
0: 7735,   # HDF
1: 7735,   # NF
2: 7735,   # OSF
3: 7735,   # PWF
4: 100000   # TWF
}



```
{'Accuracy': 0.9769539078156313,
 'Balanced Accuracy': 0.8583656330749354,
 'Macro Recall': 0.8583656330749354,
 'Macro Precision': 0.6815806235430918,
 'Macro F1': 0.7457694465723662,
 'F1 Scores per Class': {'HDF': 0.9545454545454546,
  'NF': 0.9880083420229405,
  'OSF': 0.8648648648648649,
  'PWF': 0.75,
  'TWF': 0.17142857142857143},
 'Recall Scores per Class': {'HDF': 1.0,
  'NF': 0.979328165374677,
  'OSF': 1.0,
  'PWF': 0.9375,
  'TWF': 0.375},
 'Precision Scores per Class': {'HDF': 0.9130434782608695,
  'NF': 0.9968437664387164,
  'OSF': 0.7619047619047619,
  'PWF': 0.625,
  'TWF': 0.1111111111111111}}
```

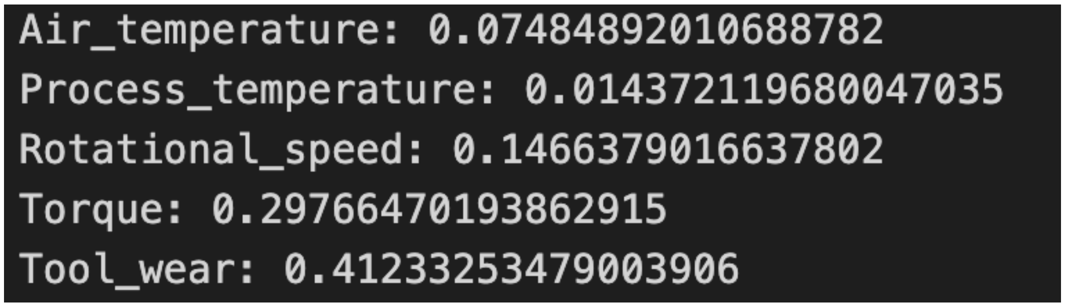
After Fine Tuning:

{'model__alpha': 0.001, 'model__eta': 0.2, 'model__lambda': 0.01, 'model__max_depth': None, 'model__n_estimators': 100}



```
{'Accuracy': 0.9774549098196392,
 'Balanced Accuracy': 0.8584689922480621,
 'Macro Recall': 0.8584689922480621,
 'Macro Precision': 0.6792641900686065,
 'Macro F1': 0.7459309913010922,
 'F1 Scores per Class': {'HDF': 0.9333333333333333,
  'NF': 0.9882720875684128,
  'OSF': 0.8421052631578947,
  'PWF': 0.7894736842105263,
  'TWF': 0.17647058823529413},
 'Recall Scores per Class': {'HDF': 1.0,
  'NF': 0.9798449612403101,
  'OSF': 1.0,
  'PWF': 0.9375,
  'TWF': 0.375},
 'Precision Scores per Class': {'HDF': 0.875,
  'NF': 0.9968454258675079,
  'OSF': 0.7272727272727273,
  'PWF': 0.6818181818181818,
  'TWF': 0.11538461538461539}}
```

- Was also able to use this model to calculate feature importances:



```
Air_temperature: 0.07484892010688782
Process_temperature: 0.014372119680047035
Rotational_speed: 0.1466379016637802
Torque: 0.29766470193862915
Tool_wear: 0.41233253479003906
```

Summary:

- Adding L1, L2 regularisation through XGboost only had marginal improvements. The model is still overfitting.
- Adding regularisation through Neural Net also has only marginal improvements. But it is not overfitting (as much i think)
- Conclusion: The issue may not be just an issue of overfitting. There is an unusual drop in peformance for TWF particularly

Next Steps:

- XGBoost regularisation has not worked. Is it a problem with my implementation or something do with the data? Have to examine this
- ~~Explore the undersampling approach~~