HiggsBoson_DNN_models

November 17, 2023

1 Classification of a Signal that Produces Higgs Boson Particles and background signals

2 Convolutional Neural Network

2.0.1 Matthew Boyer and Jonah Goldfine

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import classification_report, accuracy_score, roc_curve,␣
  ↪auc, confusion_matrix
from sklearn.base import BaseEstimator,ClassifierMixin
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm import tqdm
from skorch import NeuralNetBinaryClassifier
from skorch import callbacks as cb
import pickle
```

Loading best model from pickle file from the Optuna hyperparameter tuning.

[ ]:
```
studies_file='study.pkl'
best_model_file='best_model.pkl'
with open(studies_file,'rb') as file:
    studies=pickle.load(file)
```

```
c:\Python39\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found.
Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
[ ]: studies_df=studies.trials_dataframe().sort_values(by='value',ascending=False).
     ↪head(1)
     best_trial = studies_df.iloc[0]
     best_trial
```

```
[ ]: number                                           4
     value                                     0.759064
     datetime_start          2023-11-13 08:06:15.778856
     datetime_complete       2023-11-13 10:42:27.524985
     duration                    0 days 02:36:11.746129
     params_activation                        LeakyReLU
     params_batch_size                              200
     params_layer_sizes                112_56_28_14_7
     params_lr                                   0.0001
     params_max_epochs                              250
     state                                     COMPLETE
     Name: 4, dtype: object
```

```
[ ]: name_dtype=np.array([['class_label', np.float32], ['jet_1_b-tag', np.float64],
                 ['jet_1_eta', np.float64], ['jet_1_phi', np.float64],
                 ['jet_1_pt', np.float64], ['jet_2_b-tag', np.float64],
                 ['jet_2_eta', np.float64], ['jet_2_phi', np.float64],
                 ['jet_2_pt', np.float64], ['jet_3_b-tag', np.float64],
                 ['jet_3_eta', np.float64], ['jet_3_phi', np.float64],
                 ['jet_3_pt', np.float64], ['jet_4_b-tag', np.float64],
                 ['jet_4_eta', np.float64], ['jet_4_phi', np.float64],
                 ['jet_4_pt', np.float64], ['lepton_eta', np.float64],
                 ['lepton_pT', np.float64], ['lepton_phi', np.float64],
                 ['m_bb', np.float64], ['m_jj', np.float64],
                 ['m_jjj', np.float64], ['m_jlv', np.float64],
                 ['m_lv', np.float64], ['m_wbb', np.float64],
                 ['m_wwbb', np.float64], ['missing_energy_magnitude', np.float64],
                 ['missing_energy_phi', np.float64]])
     fullData=pd.read_csv('HIGGS.csv',header=None,names=name_dtype[:,0])
     unscaled_X=fullData.drop(['class_label'],axis=1)
```

```
[ ]: categ_features = ['jet_3_b-tag', 'jet_4_b-tag', 'lepton_eta', 'm_jj']
     num_features = unscaled_X.columns[~unscaled_X.columns.isin(categ_features)]
     unscaled_X[['jet_3_b-tag', 'jet_4_b-tag', 'lepton_eta', 'm_jj']]
```

[ ]:

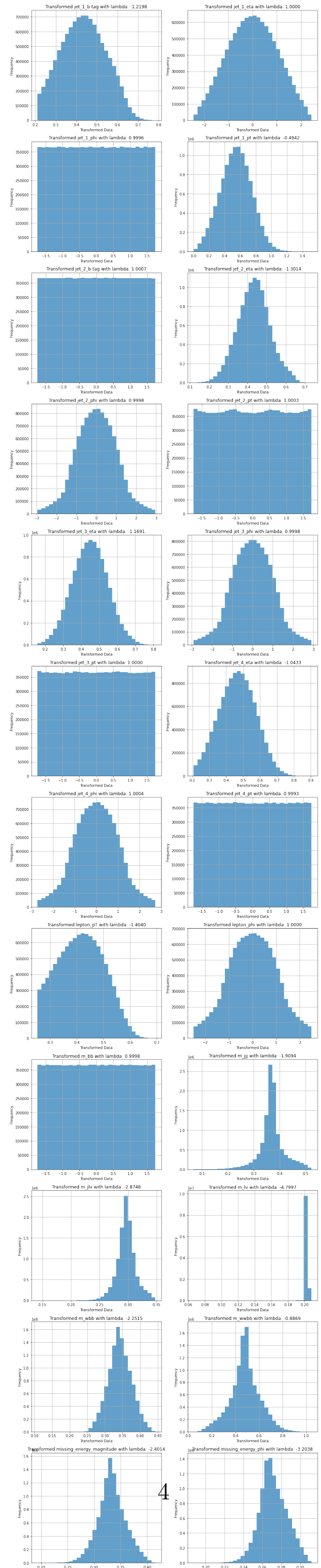
	jet_3_b-tag	jet_4_b-tag	lepton_eta	m_jj
0	0.000000	1.107436	0.000000	3.101961
1	2.173076	2.214872	0.000000	0.000000
2	0.000000	2.214872	2.548224	0.000000
3	0.000000	2.214872	0.000000	0.000000
4	0.000000	2.214872	0.000000	0.000000
…	…	…	…	…

```
10999995     0.000000     0.000000     0.000000  3.101961
10999996     2.173076     0.000000     0.000000  3.101961
10999997     2.173076     2.214872     0.000000  0.000000
10999998     0.000000     2.214872     0.000000  0.000000
10999999     2.173076     2.214872     0.000000  0.000000

[11000000 rows x 4 columns]
```

These column seem to be categorical as they only have 3 different numbers for each column across the entire dataset. Then using Yeo-Johnson transformation, as it can normalize positive and negative numbers.

[ ]:
```python
import scipy.stats as stats

for column in num_features:
    unscaled_X[column], fitted_lambda = stats.yeojohnson(unscaled_X[column])
```

[ ]:
```python
num_columns = len(num_features)
num_rows = (num_columns + 1) // 2

fig, axes = plt.subplots(num_rows, 2, figsize=(12, 5*num_rows))

for i, column in enumerate(num_features):
    row = i // 2
    col = i % 2

    axes[row, col].hist(unscaled_X[column], bins=30, alpha=0.7)
    axes[row, col].set_title(f'Transformed {column} with lambda: {fitted_lambda:
 ↪.4f}')
    axes[row, col].set_xlabel('Transformed Data')
    axes[row, col].set_ylabel('Frequency')
    axes[row, col].grid(True)

plt.tight_layout()
plt.show()
```


Transformed jet_1_b-tag with lambda: -1.2198
Transformed jet_1_eta with lambda: 1.0000
Transformed jet_1_phi with lambda: 0.9996
Transformed jet_1_pt with lambda: -0.4942
Transformed jet_2_b-tag with lambda: 1.0007
Transformed jet_2_eta with lambda: 1.3014
Transformed jet_2_phi with lambda: 0.9998
Transformed jet_2_pt with lambda: 1.0003
Transformed jet_3_eta with lambda: -1.1691
Transformed jet_3_phi with lambda: 0.9998
Transformed jet_3_pt with lambda: 1.0000
Transformed jet_4_eta with lambda: -1.0433
Transformed jet_4_phi with lambda: 1.0004
Transformed jet_4_pt with lambda: 0.9993
Transformed lepton_pT with lambda: -1.4040
Transformed lepton_phi with lambda: 1.0000
Transformed m_bb with lambda: 0.9998
Transformed m_jj with lambda: -1.9094
Transformed m_jlv with lambda: -2.8748
Transformed m_lv with lambda: -4.7997
Transformed m_wbb with lambda: -2.2515
Transformed m_wwbb with lambda: -0.8869
Transformed missing_energy_magnitude with lambda: -2.4014
Transformed missing_energy_phi with lambda: -3.2038
Transformed Data
Frequency

```
[ ]: scaler=StandardScaler()
     norm_col = scaler.fit_transform(unscaled_X[num_features])
     one_hot = pd.get_dummies(unscaled_X[categ_features].astype(str))
     df = pd.DataFrame(norm_col, columns=num_features)
     df = pd.concat([one_hot, df], axis=1)
     df.head(5)
```

```
[ ]:    jet_3_b-tag_0.0  jet_3_b-tag_1.0865380764007568  \
     0                1                               0
     1                0                               0
     2                1                               0
     3                1                               0
     4                1                               0

        jet_3_b-tag_2.1730761528015137  jet_4_b-tag_0.0  \
     0                               0                0
     1                               1                0
     2                               0                0
     3                               0                0
     4                               0                0

        jet_4_b-tag_1.1074360609054563  jet_4_b-tag_2.2148721218109126  \
     0                               1                               0
     1                               0                               1
     2                               0                               1
     3                               0                               1
     4                               0                               1

        lepton_eta_0.0  lepton_eta_1.2741122245788574  \
     0               1                              0
     1               1                              0
     2               0                              0
     3               1                              0
     4               1                              0

        lepton_eta_2.548224449157715  m_jj_0.0  …  lepton_pT  lepton_phi  \
     0                             0         0  …  -0.634425   -0.010358
     1                             0         1  …  -1.743290   -1.130232
     2                             1         1  …   0.811105    1.120230
     3                             0         1  …  -0.348277   -0.673188
     4                             0         1  …  -1.345102   -0.370698

            m_bb     m_jjj     m_jlv      m_lv     m_wbb    m_wwbb  \
     0 -0.045404  1.064561  0.126241 -0.534207 -0.013868 -0.402035
     1 -0.000740 -3.402916 -0.611616 -0.446915  0.190051 -0.240768
```

```
2  0.894760  0.019234  0.620455 -0.446999  0.098544 -0.178249
3 -1.351842  0.133202  0.329243 -0.302941 -0.853484 -0.010873
4  0.112398 -0.533463  1.311710 -0.436624 -0.337931  0.561024

   missing_energy_magnitude  missing_energy_phi
0                  0.122774           -0.061638
1                  0.136117           -0.482431
2                 -0.385904           -0.591754
3                  0.259354            0.303283
4                 -0.356921           -0.423574

[5 rows x 36 columns]
```

[ ]:
```python
full_y=fullData['class_label']
X_train_df,X_test_df,y_train_df,y_test_df=train_test_split(df,full_y,test_size=0.
↪8,random_state=0)
```

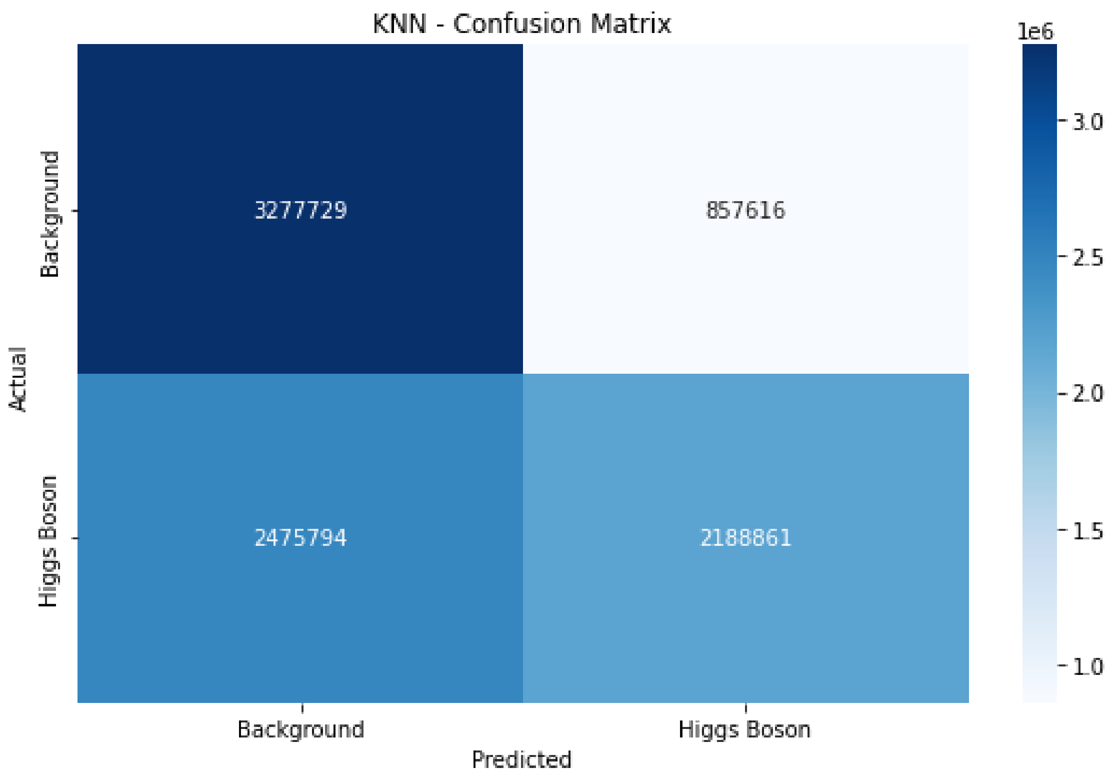
[ ]:
```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train_df, y_train_df)

knn_predict = knn.predict(X_test_df)
knn_predict_train = knn.predict(X_train_df)

knn_score_train = accuracy_score(y_train_df, knn_predict_train)
print(f'Train KNN accuracy: {knn_score_train:.3f}')
knn_score = accuracy_score(y_test_df, knn_predict)
print(f'Test KNN accuracy: {knn_score:.3f}')
```

```
c:\Python39\lib\site-packages\sklearn\neighbors\_classification.py:237:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy
1.11.0, this behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be eliminated, and the
value None will no longer be accepted. Set `keepdims` to True or False to avoid
this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
c:\Python39\lib\site-packages\sklearn\neighbors\_classification.py:237:
FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy
1.11.0, this behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be eliminated, and the
value None will no longer be accepted. Set `keepdims` to True or False to avoid
this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

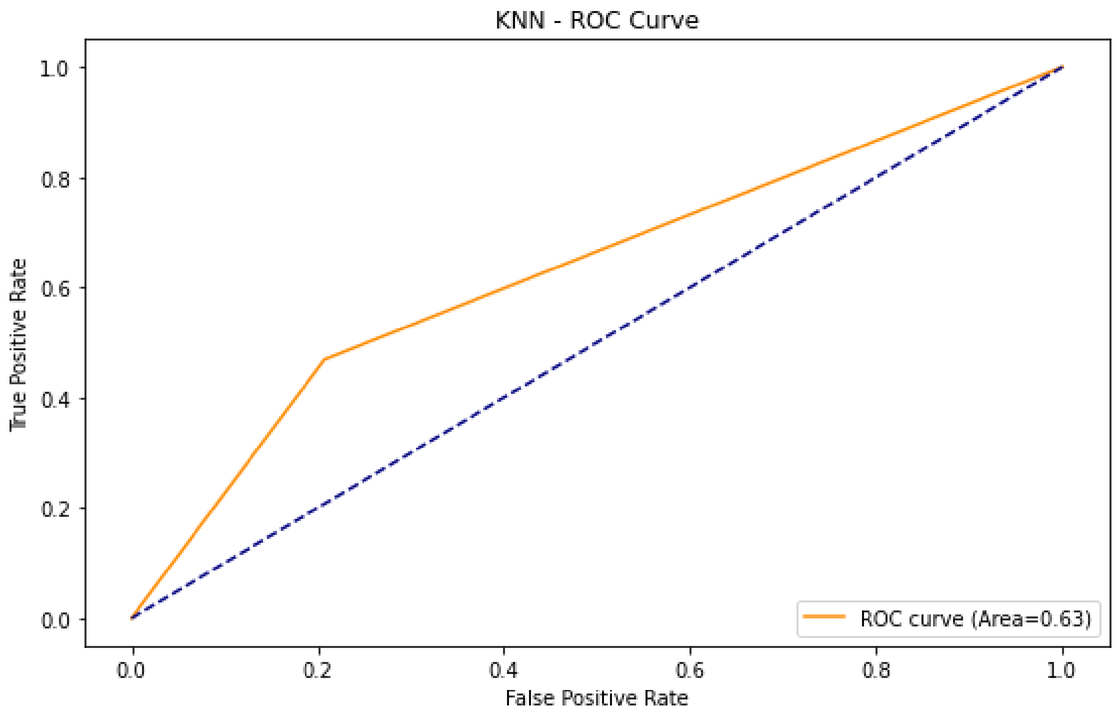
```
Train KNN accuracy: 0.823
```

Test KNN accuracy: 0.621

[ ]:
```
cm=confusion_matrix(y_test_df,knn_predict)
plt.figure(figsize=(9.26,5.62))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues',xticklabels=['Background','Higgs␣
↪Boson'],yticklabels=['Background','Higgs Boson'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('KNN - Confusion Matrix')
plt.show()
```



[ ]:
```
fpr,tpr,_=roc_curve(y_test_df,knn_predict)
auc_sc=auc(fpr,tpr)

plt.figure(figsize=(9.26,5.62))
plt.plot(fpr,tpr,color='darkorange',label=f'ROC curve (Area={auc_sc:.2f})')
plt.plot([0,1],[0,1],color='navy',linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('KNN - ROC Curve')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
X_train=torch.tensor(X_train_df.values).float()
X_test=torch.tensor(X_test_df.values).float()
y_train=torch.tensor(y_train_df.values).float()
y_test=torch.tensor(y_test_df.values).float()
```

[ ]:
```
nn.ModuleList()
```

[ ]:
```
ModuleList()
```

2.1 DNNs

Training 3 different DNN models. One with Dropout and LeakyReLu activation, one without Dropout and LeakyReLu activation, and one with Dropout and Tanh activation.

[ ]:
```
class DNN_Drop(nn.Module):
    def __init__(self, layer_sizes):
        super(DNN_Drop, self).__init__()
        self.layers=nn.ModuleList()
        activation = nn.LeakyReLU()
        # This is here because in Optuna you can only send lists of strings,␣
 ↪not lists of lists. So we split the string to get the layer sizes.
        if layer_sizes == 'empty':
            layer_sizes = []
        else:
```

```
            layer_sizes = [int(size) for size in layer_sizes.split('_')]

        if len(layer_sizes)==0:
            self.layers.append(nn.Linear(36,1))
            self.layers.append(nn.Sigmoid())
        else:
            for i, hidden_size in enumerate(layer_sizes):
                if i==0:
                    self.layers.append(nn.Linear(36,hidden_size))
                    self.layers.append(nn.Dropout(p=0.25))
                    self.layers.append(activation)
                    input_size=hidden_size
                else:
                    self.layers.append(nn.Linear(input_size,hidden_size))
                    self.layers.append(activation)
                    input_size=hidden_size
            self.layers.append(nn.Linear(input_size,1))
            self.layers.append(nn.Sigmoid())

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x.squeeze()
```

[ ]:
```
DNN_Drop_model=NeuralNetBinaryClassifier(
    DNN_Drop,
    module__layer_sizes=best_trial['params_layer_sizes'],
    criterion=nn.BCELoss,
    optimizer=optim.Adam,
    optimizer__lr=best_trial['params_lr'],
    max_epochs=int(best_trial['params_max_epochs']),
    batch_size= int(best_trial['params_batch_size']),
    callbacks=[('early_stopping',cb.EarlyStopping(patience=10)),
               ('train_acc',cb.EpochScoring(scoring='accuracy',
                                             lower_is_better=False,
                                             on_train=True))],
    verbose=5,
    threshold=0.525
)
```

[ ]:
```
DNN_Drop_model.fit(X_train, y_train)
```

```
  epoch    accuracy    train_loss    valid_acc    valid_loss      dur
-------  ----------  ------------  -----------  ------------  -------
      1      0.6728        0.6006       0.7100
0.5603  48.4227
      2      0.7061        0.5643       0.7214
```

```
0.5438  48.4777
     3        0.7157        0.5504        0.7278
0.5329  48.1945
     4        0.7218        0.5408        0.7324
0.5253  48.3131
     5        0.7257        0.5349        0.7353
0.5206  48.2053
     6        0.7285        0.5306        0.7377
0.5168  48.2286
     7        0.7304        0.5278        0.7392
0.5142  48.0651
     8        0.7321        0.5253        0.7411
0.5121  48.2584
     9        0.7328        0.5237        0.7423
0.5104  47.9619
    10        0.7346        0.5219        0.7432
0.5088  48.0726
    11        0.7353        0.5202        0.7440
0.5073  48.5125
    12        0.7361        0.5191        0.7446
0.5064  48.3082
    13        0.7370        0.5181        0.7451
0.5054  48.0660
    14        0.7375        0.5170        0.7462
0.5041  48.3366
    15        0.7383        0.5160        0.7467
0.5033  47.9059
    16        0.7389        0.5153        0.7470
0.5026  48.1920
    17        0.7395        0.5144        0.7476
0.5017  48.1372
    18        0.7400        0.5138        0.7481
0.5009  47.9737
    19        0.7404        0.5132        0.7482
0.5002  48.0771
    20        0.7409        0.5126        0.7490
0.4996  47.9229
    21        0.7412        0.5120        0.7491
0.4993  48.1849
    22        0.7414        0.5115        0.7499
0.4984  48.2667
    23        0.7422        0.5109        0.7499
0.4980  48.0748
    24        0.7421        0.5106        0.7503
0.4975  48.3967
    25        0.7427        0.5101        0.7508
0.4971  48.2489
    26        0.7428        0.5096        0.7509
```

0.4970 48.2112
27 0.7430 0.5093 0.7512
0.4964 48.4004
28 0.7435 0.5089 0.7514
0.4961 48.0681
29 0.7437 0.5086 0.7518
0.4959 48.0919
30 0.7439 0.5083 0.7521
0.4954 48.2233
31 0.7439 0.5080 0.7521
0.4954 48.2627
32 0.7441 0.5077 0.7528
0.4949 48.0408
33 0.7445 0.5075 0.7524
0.4947 48.4197
34 0.7448 0.5071 0.7531
0.4943 48.0764
35 0.7449 0.5070 0.7530
0.4942 49.9575
36 0.7451 0.5067 0.7532
0.4940 49.9159
37 0.7452 0.5065 0.7534
0.4938 48.4978
38 0.7450 0.5066 0.7535 0.4937
48.2909
39 0.7456 0.5062 0.7534
0.4936 48.1046
40 0.7455 0.5062 0.7536
0.4932 47.9791
41 0.7457 0.5057 0.7537
0.4930 48.9385
42 0.7457 0.5056 0.7542
0.4930 48.3790
43 0.7459 0.5054 0.7542
0.4927 47.7411
44 0.7460 0.5053 0.7539 0.4927
48.0944
45 0.7465 0.5050 0.7544
0.4927 48.4828
46 0.7463 0.5048 0.7542 0.4924
47.7073
47 0.7466 0.5048 0.7544
0.4923 47.8189
48 0.7463 0.5047 0.7543 0.4922
48.6789
49 0.7464 0.5046 0.7546 0.4922
47.6177
50 0.7465 0.5044 0.7547

0.4921 48.4262
51 0.7466 0.5045 0.7547
0.4920 48.2150
52 0.7469 0.5041 0.7546
0.4917 48.1870
53 0.7471 0.5040 0.7550
0.4919 47.9419
54 0.7471 0.5040 0.7549 0.4917 49.1582
55 0.7472 0.5038 0.7549
0.4915 48.5975
56 0.7472 0.5038 0.7551
0.4914 48.2914
57 0.7469 0.5037 0.7550 0.4915 48.7191
58 0.7474 0.5035 0.7550 0.4915
48.4158
59 0.7476 0.5035 0.7552
0.4914 49.7014
60 0.7476 0.5032 0.7554
0.4912 48.1276
61 0.7477 0.5029 0.7553
0.4912 48.6496
62 0.7473 0.5030 0.7555 0.4909
47.8554
63 0.7478 0.5030 0.7557 0.4912
48.4411
64 0.7476 0.5030 0.7555 0.4908 48.2693
65 0.7476 0.5029 0.7556 0.4909 48.2603
66 0.7479 0.5029 0.7559
0.4905 47.9031
67 0.7479 0.5024 0.7557 0.4906 48.7829
68 0.7480 0.5025 0.7554 0.4907 48.6833
69 0.7478 0.5025 0.7558 0.4904 48.5834
70 0.7479 0.5024 0.7560 0.4905
48.3614
71 0.7482 0.5022 0.7561
0.4903 48.3955
72 0.7478 0.5025 0.7562 0.4902
48.4094
73 0.7480 0.5021 0.7558 0.4902 48.4374
74 0.7483 0.5021 0.7564
0.4902 47.7880
75 0.7481 0.5020 0.7562 0.4901
48.9583
76 0.7487 0.5018 0.7562 0.4903
48.2743
77 0.7486 0.5018 0.7561 0.4899
49.6647
78 0.7487 0.5017 0.7561 0.4901

48.0292					
79	0.7483	0.5020	0.7563	0.4899	48.3542
80	0.7483	0.5016	0.7560	0.4899	
48.2573					
81	0.7484	0.5018	0.7562	0.4896	48.6606
82	0.7487	0.5017	0.7563	0.4898	47.6241
83	0.7488	0.5013	0.7562	0.4898	
48.1400					
84	0.7487	0.5014	0.7562	0.4898	48.0016
85	0.7486	0.5013	0.7562	0.4897	47.7484
86	0.7487	0.5013	0.7564	0.4897	46.5854
87	0.7487	0.5012	0.7564	0.4896	46.2540
88	0.7489	0.5014	0.7567		
0.4894	47.0573				
89	0.7485	0.5012	0.7565	0.4895	49.4235
90	0.7487	0.5010	0.7560	0.4895	50.2959
91	0.7490	0.5008	0.7562	0.4894	
50.4395					
92	0.7487	0.5010	0.7565	0.4892	50.1285
93	0.7489	0.5010	0.7564	0.4894	50.5362
94	0.7491	0.5010	0.7566	0.4891	
51.1843					
95	0.7494	0.5006	0.7567		
0.4891	50.6748				
96	0.7490	0.5009	0.7565	0.4893	49.9694
97	0.7493	0.5004	0.7563	0.4894	49.8709
98	0.7491	0.5006	0.7566	0.4890	50.3997
99	0.7491	0.5006	0.7567	0.4890	
49.9507					
100	0.7491	0.5006	0.7570	0.4886	
50.0739					
101	0.7494	0.5005	0.7566	0.4891	49.5839
102	0.7491	0.5006	0.7568	0.4889	49.6666
103	0.7495	0.5005	0.7568	0.4887	49.4509
104	0.7496	0.5002	0.7567	0.4890	
50.5542					
105	0.7496	0.5001	0.7569	0.4889	
50.1863					
106	0.7494	0.5000	0.7567	0.4886	
49.5485					
107	0.7494	0.5000	0.7569	0.4886	49.6887
108	0.7496	0.5001	0.7571	0.4883	
49.2847					
109	0.7495	0.5001	0.7568	0.4886	49.6883
110	0.7494	0.5001	0.7573	0.4884	49.4078
111	0.7495	0.5000	0.7573	0.4883	49.3279
112	0.7490	0.5001	0.7573	0.4885	49.4928
113	0.7493	0.5001	0.7571	0.4886	49.3026

```
    114      0.7496       0.5000       0.7570       0.4884  49.1140
    115      0.7497       0.4998       0.7570       0.4884
49.4470
    116      0.7495       0.4999       0.7573       0.4884  48.9390
    117      0.7497       0.5000       0.7571       0.4883
49.0322
    118      0.7498       0.4998       0.7572
0.4882  49.1637
    119      0.7498       0.4996       0.7572       0.4884  49.3495
    120      0.7497       0.4997       0.7574       0.4882  49.3651
    121      0.7497       0.4999       0.7571       0.4883  49.7677
    122      0.7498       0.4995       0.7571       0.4882  49.2925
    123      0.7498       0.4995       0.7571       0.4885  49.5195
    124      0.7499       0.4995       0.7574       0.4881
49.3702
    125      0.7499       0.4994       0.7573       0.4881
49.6248
    126      0.7501       0.4994       0.7574
0.4881  49.4429
    127      0.7501       0.4994       0.7575
0.4880  49.6613
    128      0.7499       0.4994       0.7573       0.4878  49.3856
    129      0.7501       0.4993       0.7575       0.4879  49.2393
    130      0.7500       0.4991       0.7573       0.4880  49.2137
    131      0.7497       0.4991       0.7574       0.4882  48.9471
    132      0.7501       0.4992       0.7574       0.4881  49.0937
    133      0.7502       0.4991       0.7573       0.4878  49.0062
    134      0.7501       0.4992       0.7575       0.4877  49.2667
    135      0.7502       0.4989       0.7574       0.4879  49.0341
    136      0.7502       0.4990       0.7573       0.4881  49.1147
    137      0.7501       0.4989       0.7575       0.4876  49.2338
    138      0.7503       0.4989       0.7574       0.4881  48.8492
    139      0.7506       0.4988       0.7575       0.4878
49.7324
    140      0.7501       0.4990       0.7575       0.4880  49.1874
    141      0.7507       0.4987       0.7576
0.4879  49.2416
    142      0.7503       0.4987       0.7574       0.4880  49.1171
    143      0.7505       0.4988       0.7572       0.4880  49.2611
    144      0.7506       0.4986       0.7573       0.4878  49.0985
    145      0.7503       0.4988       0.7576       0.4881  49.4248
    146      0.7506       0.4986       0.7577       0.4878
49.4869
Stopping since valid_loss has not improved in the last 10 epochs.
```

[ ]:
```
<class 'skorch.classifier.NeuralNetBinaryClassifier'>[initialized](
  module_=DNN_Drop(
```

```
    (layers): ModuleList(
      (0): Linear(in_features=36, out_features=112, bias=True)
      (1): Dropout(p=0.25, inplace=False)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Linear(in_features=112, out_features=56, bias=True)
      (4): LeakyReLU(negative_slope=0.01)
      (5): Linear(in_features=56, out_features=28, bias=True)
      (6): LeakyReLU(negative_slope=0.01)
      (7): Linear(in_features=28, out_features=14, bias=True)
      (8): LeakyReLU(negative_slope=0.01)
      (9): Linear(in_features=14, out_features=7, bias=True)
      (10): LeakyReLU(negative_slope=0.01)
      (11): Linear(in_features=7, out_features=1, bias=True)
      (12): Sigmoid()
    )
  ),
)
```

[ ]:
```
class DNN_No_Drop(nn.Module):
    def __init__(self, layer_sizes):
        super(DNN_No_Drop, self).__init__()
        self.layers=nn.ModuleList()
        activation = nn.LeakyReLU()
        if layer_sizes == 'empty':
            layer_sizes = []
        else:
            layer_sizes = [int(size) for size in layer_sizes.split('_')]

        if len(layer_sizes)==0:
            self.layers.append(nn.Linear(36,1))
            self.layers.append(nn.Sigmoid())
        else:
            for i, hidden_size in enumerate(layer_sizes):
                if i==0:
                    self.layers.append(nn.Linear(36,hidden_size))
                    self.layers.append(activation)
                    input_size=hidden_size
                else:
                    self.layers.append(nn.Linear(input_size,hidden_size))
                    self.layers.append(activation)
                    input_size=hidden_size
            self.layers.append(nn.Linear(input_size,1))
            self.layers.append(nn.Sigmoid())

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
```

```
        return x.squeeze()
```

[ ]:
```
DNN_No_Drop_model=NeuralNetBinaryClassifier(
    DNN_No_Drop,
    module__layer_sizes=best_trial['params_layer_sizes'],
    criterion=nn.BCELoss,
    optimizer=optim.Adam,
    optimizer__lr=best_trial['params_lr'],
    max_epochs=int(best_trial['params_max_epochs']),
    batch_size= int(best_trial['params_batch_size']),
    callbacks=[('early_stopping',cb.EarlyStopping(patience=10)),
               ('train_acc',cb.EpochScoring(scoring='accuracy',
                                             lower_is_better=False,
                                             on_train=True))],
    verbose=5,
    threshold=0.525
)
```

[ ]:
```
DNN_No_Drop_model.fit(X_train, y_train)
```

```
  epoch    accuracy    train_loss    valid_acc    valid_loss      dur
-------  ----------  ------------  -----------  ------------  -------
      1      0.6889        0.5825       0.7131
0.5540  45.9063
      2      0.7182        0.5468       0.7230
0.5385  46.3394
      3      0.7259        0.5343       0.7288
0.5288  46.3844
      4      0.7312        0.5259       0.7329
0.5221  46.2294
      5      0.7350        0.5204       0.7359
0.5178  46.1934
      6      0.7374        0.5166       0.7382
0.5147  46.1024
      7      0.7395        0.5138       0.7394
0.5122  46.4174
      8      0.7411        0.5113       0.7408
0.5100  46.4164
      9      0.7426        0.5091       0.7421
0.5081  46.1504
     10      0.7439        0.5072       0.7432
0.5065  46.6415
     11      0.7450        0.5056       0.7443
0.5051  46.0674
     12      0.7459        0.5042       0.7451
0.5039  46.4184
     13      0.7469        0.5030       0.7460
```

0.5027 46.6985

14 0.7476 0.5019 0.7469

0.5017 46.2984

15 0.7483 0.5008 0.7477

0.5008 46.4094

16 0.7490 0.4998 0.7481

0.5001 46.5595

17 0.7498 0.4989 0.7490

0.4993 46.4714

18 0.7505 0.4980 0.7495

0.4986 46.7955

19 0.7511 0.4972 0.7502

0.4979 46.5595

20 0.7516 0.4965 0.7504

0.4973 46.6305

21 0.7521 0.4958 0.7507

0.4968 46.8215

22 0.7524 0.4951 0.7511

0.4962 46.7295

23 0.7528 0.4945 0.7512

0.4957 46.9756

24 0.7531 0.4940 0.7513

0.4953 46.9681

25 0.7536 0.4934 0.7517

0.4948 46.8605

26 0.7539 0.4929 0.7517

0.4945 46.8335

27 0.7541 0.4925 0.7519

0.4940 46.8815

28 0.7544 0.4920 0.7522

0.4936 46.9296

29 0.7547 0.4916 0.7523

0.4934 47.1586

30 0.7549 0.4912 0.7526

0.4930 47.0926

31 0.7552 0.4908 0.7528

0.4927 47.1066

32 0.7554 0.4904 0.7530

0.4923 47.2886

33 0.7557 0.4901 0.7531

0.4921 47.1676

34 0.7559 0.4897 0.7532

0.4919 47.0256

35 0.7562 0.4894 0.7533

0.4917 46.9666

36 0.7564 0.4891 0.7537

0.4915 47.1516

37 0.7566 0.4888 0.7537

0.4912 47.0086
38 0.7567 0.4885 0.7536
0.4910 47.0446
39 0.7569 0.4883 0.7537
0.4908 47.4397
40 0.7570 0.4880 0.7540
0.4906 45.5163
41 0.7572 0.4877 0.7540
0.4905 45.0421
42 0.7574 0.4875 0.7541
0.4903 45.3262
43 0.7576 0.4872 0.7540
0.4901 44.9361
44 0.7577 0.4870 0.7542
0.4899 44.9451
45 0.7579 0.4867 0.7539
0.4897 45.1401
46 0.7582 0.4865 0.7540
0.4896 45.0831
47 0.7583 0.4863 0.7542
0.4894 45.1452
48 0.7585 0.4860 0.7543
0.4892 45.4512
49 0.7587 0.4858 0.7545
0.4890 45.2442
50 0.7588 0.4856 0.7546
0.4888 45.2892
51 0.7589 0.4854 0.7544
0.4887 45.1497
52 0.7590 0.4852 0.7546
0.4886 45.1782
53 0.7591 0.4850 0.7545
0.4884 45.0441
54 0.7592 0.4847 0.7548
0.4882 45.1351
55 0.7594 0.4845 0.7551
0.4880 45.1091
56 0.7595 0.4844 0.7552
0.4879 44.9951
57 0.7597 0.4842 0.7555
0.4877 45.0781
58 0.7598 0.4840 0.7556
0.4876 45.0501
59 0.7599 0.4838 0.7559
0.4875 45.3322
60 0.7600 0.4836 0.7559
0.4874 45.1141
61 0.7601 0.4835 0.7560

0.4872 45.0901
62 0.7602 0.4833 0.7563
0.4870 45.1186
63 0.7603 0.4831 0.7564
0.4869 45.2142
64 0.7604 0.4830 0.7565
0.4868 45.1622
65 0.7605 0.4828 0.7566
0.4867 44.9431
66 0.7605 0.4827 0.7568
0.4866 45.4112
67 0.7607 0.4826 0.7568
0.4865 45.2932
68 0.7608 0.4824 0.7569
0.4864 45.3932
69 0.7608 0.4823 0.7570
0.4863 45.5332
70 0.7609 0.4822 0.7569
0.4862 45.2102
71 0.7609 0.4820 0.7569
0.4862 45.3812
72 0.7610 0.4819 0.7571
0.4860 45.4062
73 0.7611 0.4818 0.7570
0.4859 45.2532
74 0.7613 0.4817 0.7570
0.4858 45.3392
75 0.7614 0.4815 0.7571
0.4858 45.5762
76 0.7614 0.4814 0.7575
0.4857 45.0921
77 0.7616 0.4813 0.7573
0.4856 45.3502
78 0.7616 0.4812 0.7575
0.4855 45.3492
79 0.7618 0.4811 0.7576
0.4854 45.0481
80 0.7619 0.4810 0.7577
0.4852 45.2202
81 0.7619 0.4809 0.7577
0.4851 45.3112
82 0.7620 0.4808 0.7578
0.4850 45.2822
83 0.7621 0.4806 0.7580
0.4850 45.3052
84 0.7621 0.4805 0.7580
0.4849 45.3802
85 0.7623 0.4804 0.7583

```
0.4847  45.2342
     86      0.7623        0.4803       0.7583       0.4847
45.5032
     87      0.7623        0.4802       0.7585
0.4846  45.3282
     88      0.7624        0.4801       0.7585
0.4845  45.2192
     89      0.7625        0.4800       0.7585
0.4844  45.2102
     90      0.7625        0.4799       0.7587
0.4844  45.1391
     91      0.7625        0.4798       0.7586
0.4843  45.0781
     92      0.7626        0.4797       0.7587
0.4842  45.1391
     93      0.7626        0.4797       0.7588
0.4842  45.3437
     94      0.7627        0.4796       0.7588
0.4841  45.1922
     95      0.7627        0.4795       0.7589
0.4841  45.4912
     96      0.7628        0.4794       0.7590
0.4840  45.6948
```

[ ]:
```python
class DNN_Drop_Tanh(nn.Module):
    def __init__(self, layer_sizes):
        super(DNN_Drop_Tanh, self).__init__()
        self.layers=nn.ModuleList()
        activation = nn.Tanh()
        if layer_sizes == 'empty':
            layer_sizes = []
        else:
            layer_sizes = [int(size) for size in layer_sizes.split('_')]

        if len(layer_sizes)==0:
            self.layers.append(nn.Linear(36,1))
            self.layers.append(nn.Sigmoid())
        else:
            for i, hidden_size in enumerate(layer_sizes):
                if i==0:
                    self.layers.append(nn.Linear(36,hidden_size))
                    self.layers.append(nn.Dropout(p=0.25))
                    self.layers.append(activation)
                    input_size=hidden_size
                else:
                    self.layers.append(nn.Linear(input_size,hidden_size))
                    self.layers.append(activation)
```

```
                    input_size=hidden_size
            self.layers.append(nn.Linear(input_size,1))
            self.layers.append(nn.Sigmoid())

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x.squeeze()
```

[ ]:
```
DNN_Drop_Tanh_model=NeuralNetBinaryClassifier(
    DNN_Drop_Tanh,
    module__layer_sizes=best_trial['params_layer_sizes'],
    criterion=nn.BCELoss,
    optimizer=optim.Adam,
    optimizer__lr=best_trial['params_lr'],
    max_epochs=int(best_trial['params_max_epochs']),
    batch_size= int(best_trial['params_batch_size']),
    callbacks=[('early_stopping',cb.EarlyStopping(patience=10)),
               ('train_acc',cb.EpochScoring(scoring='accuracy',
                                            lower_is_better=False,
                                            on_train=True))],
    verbose=5
)

DNN_Drop_Tanh_model.fit(X_train, y_train)
```

```
  epoch    accuracy    train_loss    valid_acc    valid_loss      dur
-------  ----------  ------------  -----------  ------------  -------
      1      0.6524        0.6200       0.6901
0.5828  43.2982
      2      0.6934        0.5791       0.7079
0.5610  43.2725
      3      0.7058        0.5634       0.7171
0.5486  42.5536
      4      0.7136        0.5528       0.7224
0.5404  42.4893
      5      0.7191        0.5456       0.7276
0.5334  43.0775
      6      0.7234        0.5396       0.7309
0.5282  43.8563
      7      0.7268        0.5355       0.7335
0.5248  43.2018
      8      0.7285        0.5326       0.7349
0.5223  43.5530
      9      0.7300        0.5302       0.7363
0.5202  42.9380
     10      0.7310        0.5286       0.7375
```

0.5182 42.5984
11 0.7322 0.5267 0.7381
0.5167 42.9491
12 0.7333 0.5251 0.7398
0.5148 43.2430
13 0.7342 0.5236 0.7403
0.5137 42.7973
14 0.7346 0.5226 0.7411
0.5125 42.6485
15 0.7357 0.5215 0.7415
0.5117 42.5059
16 0.7360 0.5207 0.7421
0.5107 42.6686
17 0.7368 0.5196 0.7428
0.5100 42.4930
18 0.7371 0.5188 0.7433
0.5089 42.4885
19 0.7376 0.5181 0.7439
0.5082 42.4193
20 0.7381 0.5173 0.7447
0.5074 42.4184
21 0.7387 0.5167 0.7448
0.5069 42.4961
22 0.7390 0.5162 0.7453
0.5063 42.7414
23 0.7392 0.5159 0.7457
0.5061 42.5305
24 0.7398 0.5151 0.7459
0.5053 42.4510
25 0.7403 0.5147 0.7460
0.5051 42.6189
26 0.7402 0.5141 0.7469
0.5043 42.8022
27 0.7406 0.5138 0.7470
0.5038 43.2285
28 0.7408 0.5131 0.7472
0.5035 42.2659
29 0.7410 0.5128 0.7474
0.5033 42.4795
30 0.7412 0.5123 0.7477
0.5027 42.4452
31 0.7422 0.5118 0.7480
0.5021 42.7119
32 0.7419 0.5117 0.7482
0.5019 43.1154
33 0.7419 0.5114 0.7484
0.5015 43.2237
34 0.7422 0.5112 0.7483 0.5014

43.3152
35 0.7426 0.5108 0.7487
0.5010 43.2194
36 0.7428 0.5103 0.7488
0.5006 44.4180
37 0.7426 0.5103 0.7489
0.5003 43.5290
38 0.7433 0.5099 0.7494
0.4998 44.4049
39 0.7431 0.5096 0.7495
0.4996 43.6038
40 0.7435 0.5094 0.7498
0.4994 42.9270
41 0.7439 0.5090 0.7500
0.4990 43.6573
42 0.7437 0.5089 0.7498 0.4989
42.7483
43 0.7440 0.5085 0.7500
0.4988 43.2458
44 0.7443 0.5082 0.7503
0.4982 43.5870
45 0.7443 0.5084 0.7503 0.4982 42.8102
46 0.7446 0.5079 0.7508
0.4979 42.6699
47 0.7444 0.5077 0.7506 0.4979
42.8752
48 0.7445 0.5076 0.7506 0.4978
42.7987
49 0.7450 0.5073 0.7509
0.4974 43.0043
50 0.7449 0.5072 0.7509
0.4972 43.3505
51 0.7453 0.5069 0.7511
0.4972 42.5483
52 0.7450 0.5069 0.7512 0.4971
42.6313
53 0.7452 0.5068 0.7518
0.4966 44.9462
54 0.7453 0.5065 0.7516
0.4964 43.0898
55 0.7453 0.5064 0.7517
0.4964 43.2977
56 0.7456 0.5063 0.7520
0.4961 43.1852
57 0.7455 0.5061 0.7520
0.4959 44.1020
58 0.7455 0.5059 0.7517 0.4962 43.5132
59 0.7460 0.5057 0.7521

```
0.4960  43.5272
     60       0.7461          0.5057          0.7520          0.4958
43.6358
     61       0.7459          0.5056          0.7522
0.4956  43.1271
     62       0.7463          0.5053          0.7523
0.4956  44.1329
     63       0.7462          0.5053          0.7525
0.4953  43.6123
     64       0.7461          0.5051          0.7526          0.4954
44.2579
     65       0.7462          0.5050          0.7526          0.4953
42.6788
     66       0.7464          0.5049          0.7524          0.4953
43.2345
     67       0.7462          0.5049          0.7528
0.4950  42.9500
     68       0.7465          0.5047          0.7527
0.4949  42.7918
     69       0.7466          0.5045          0.7528
0.4948  42.8446
     70       0.7470          0.5045          0.7528
0.4947  43.6839
     71       0.7465          0.5044          0.7532
0.4945  44.0765
     72       0.7469          0.5044          0.7529          0.4947  43.4981
     73       0.7470          0.5042          0.7530
0.4944  44.2225
     74       0.7470          0.5040          0.7532
0.4941  44.1198
     75       0.7472          0.5041          0.7530          0.4943  43.3895
     76       0.7471          0.5040          0.7530          0.4942  42.7456
     77       0.7471          0.5040          0.7531          0.4942  43.4948
     78       0.7471          0.5038          0.7535
0.4938  42.9358
     79       0.7477          0.5037          0.7533          0.4939
42.8748
     80       0.7475          0.5035          0.7531          0.4939  43.3929
     81       0.7474          0.5035          0.7534          0.4937  43.3228
     82       0.7473          0.5035          0.7535          0.4937  43.2774
     83       0.7475          0.5033          0.7537
0.4934  43.4167
     84       0.7477          0.5033          0.7536          0.4933
43.6219
     85       0.7474          0.5034          0.7537          0.4935  43.7797
     86       0.7475          0.5032          0.7537
0.4933  43.0335
     87       0.7478          0.5030          0.7537
```

0.4932	43.1658				
88	0.7477	0.5030	0.7538		
0.4931	44.4393				
89	0.7476	0.5030	0.7536	0.4933	42.4962
90	0.7480	0.5027	0.7538		
0.4931	43.4481				
91	0.7478	0.5027	0.7538		
0.4930	43.9287				
92	0.7480	0.5027	0.7540	0.4929	
44.2013					
93	0.7480	0.5027	0.7540	0.4928	
45.1403					
94	0.7477	0.5027	0.7539	0.4929	44.3651
95	0.7478	0.5025	0.7541		
0.4927	44.3250				
96	0.7481	0.5024	0.7540	0.4927	
44.8951					
97	0.7481	0.5024	0.7540	0.4928	
44.3049					
98	0.7483	0.5025	0.7543		
0.4923	44.0652				
99	0.7483	0.5022	0.7540	0.4927	
43.6367					
100	0.7484	0.5022	0.7543	0.4924	
42.9681					
101	0.7482	0.5022	0.7543	0.4922	
44.5033					
102	0.7482	0.5020	0.7542	0.4924	44.4209
103	0.7483	0.5021	0.7545	0.4924	44.5269
104	0.7483	0.5020	0.7543	0.4922	44.5926
105	0.7485	0.5020	0.7542	0.4924	
44.3873					
106	0.7486	0.5017	0.7545		
0.4921	44.6575				
107	0.7488	0.5016	0.7545		
0.4920	44.5618				
108	0.7486	0.5016	0.7548		
0.4919	45.0579				
109	0.7485	0.5017	0.7549	0.4918	
46.7542					
110	0.7487	0.5014	0.7547	0.4921	44.5132
111	0.7488	0.5015	0.7546	0.4920	44.6486
112	0.7487	0.5014	0.7547	0.4918	
44.5033					
113	0.7488	0.5013	0.7546	0.4918	
44.4598					
114	0.7487	0.5014	0.7546	0.4918	44.5609
115	0.7486	0.5014	0.7546	0.4918	44.5493

116	0.7490	0.5012	0.7549	0.4915	44.4801
117	0.7488	0.5013	0.7549	0.4917	44.4623
118	0.7490	0.5013	0.7548	0.4914	44.5682
119	0.7493	0.5010	0.7550	0.4914	44.7761
120	0.7492	0.5011	0.7548	0.4913	44.3527
121	0.7488	0.5012	0.7550	0.4912	44.8006
122	0.7492	0.5008	0.7551	0.4911	44.6229
123	0.7492	0.5008	0.7553	0.4912	44.4321
124	0.7495	0.5009	0.7553	0.4912	44.7144
125	0.7495	0.5007	0.7552	0.4910	44.4169
126	0.7491	0.5009	0.7551	0.4913	44.4789
127	0.7494	0.5007	0.7552	0.4910	44.5431
128	0.7493	0.5007	0.7549	0.4912	44.5794
129	0.7495	0.5006	0.7551	0.4910	44.5392
130	0.7493	0.5008	0.7552	0.4909	44.7336
131	0.7497	0.5004	0.7552	0.4910	44.0059
132	0.7497	0.5002	0.7551	0.4909	42.6645
133	0.7497	0.5002	0.7551	0.4908	43.1390
134	0.7492	0.5004	0.7551	0.4909	42.6266
135	0.7494	0.5003	0.7553	0.4908	42.8311
136	0.7496	0.5002	0.7552	0.4906	43.4964
137	0.7498	0.5002	0.7553	0.4907	43.8874
138	0.7497	0.4999	0.7554	0.4905	42.6440
139	0.7498	0.5001	0.7551	0.4909	43.0796
140	0.7496	0.5000	0.7554	0.4906	43.5637
141	0.7498	0.5000	0.7554	0.4905	42.7243
142	0.7501	0.4999	0.7558	0.4904	43.2660
143	0.7500	0.4999	0.7555	0.4907	42.9106
144	0.7495	0.4999	0.7553	0.4905	44.2487
145	0.7501	0.4998	0.7553	0.4906	47.8477
146	0.7499	0.4997	0.7558	0.4903	45.0960
147	0.7499	0.4999	0.7554	0.4905	46.3724
148	0.7501	0.4999	0.7553	0.4906	44.9240

149	0.7499	0.4999	0.7554	0.4903	44.5038
150	0.7500	0.4998	0.7558	0.4903	45.1150
151	0.7500	0.4998	0.7554	0.4904	45.1867
152	0.7502	0.4996	0.7555	0.4903	45.4927
153	0.7501	0.4996	0.7555	0.4903	45.1948
154	0.7502	0.4996	0.7553	0.4903	45.0105
155	0.7504	0.4993	0.7555	0.4904	46.1276
156	0.7503	0.4995	0.7557	0.4900	45.2210
157	0.7500	0.4996	0.7555	0.4903	45.6175
158	0.7502	0.4995	0.7556	0.4900	45.7368
159	0.7504	0.4995	0.7555	0.4902	45.3041
160	0.7502	0.4994	0.7555	0.4900	45.5294
161	0.7504	0.4992	0.7556	0.4901	45.6349
162	0.7502	0.4993	0.7557	0.4898	45.4821
163	0.7502	0.4993	0.7555	0.4901	45.6908
164	0.7504	0.4993	0.7557	0.4900	45.6885
165	0.7503	0.4992	0.7557	0.4901	45.8940
166	0.7503	0.4991	0.7558	0.4900	45.7771
167	0.7505	0.4991	0.7560	0.4898	45.6850
168	0.7505	0.4991	0.7560	0.4897	45.7727
169	0.7505	0.4991	0.7559	0.4899	45.7290
170	0.7504	0.4990	0.7559	0.4898	45.7597
171	0.7505	0.4990	0.7560	0.4896	45.6226
172	0.7504	0.4990	0.7559	0.4898	45.7963
173	0.7506	0.4990	0.7560	0.4897	46.0728
174	0.7507	0.4990	0.7559	0.4896	45.8314
175	0.7506	0.4987	0.7560	0.4897	45.9546
176	0.7506	0.4987	0.7560	0.4895	46.1266
177	0.7507	0.4989	0.7560	0.4895	46.1029
178	0.7509	0.4988	0.7561	0.4897	47.0093
179	0.7509	0.4986	0.7564	0.4895	47.8346
180	0.7508	0.4987	0.7563	0.4894	47.2466
181	0.7508	0.4988	0.7564	0.4893	47.5830
182	0.7510	0.4987	0.7562	0.4896	46.0674

183	0.7507	0.4988	0.7563	0.4895	45.9285
184	0.7512	0.4985	0.7564	0.4893	45.6782
185	0.7508	0.4986	0.7563	0.4893	46.0549
186	0.7508	0.4987	0.7564	0.4893	45.7432
187	0.7511	0.4985	0.7566	0.4893	45.2855
188	0.7509	0.4984	0.7563	0.4893	45.2488
189	0.7510	0.4984	0.7564	0.4893	45.2753
190	0.7512	0.4983	0.7564	0.4893	45.1482
191	0.7509	0.4985	0.7564	0.4892	45.5951
192	0.7509	0.4984	0.7565	0.4892	45.1829
193	0.7507	0.4984	0.7564	0.4892	45.5710
194	0.7515	0.4982	0.7566	0.4891	45.5560
195	0.7509	0.4983	0.7566	0.4890	45.4165
196	0.7512	0.4983	0.7565	0.4890	46.1503
197	0.7510	0.4983	0.7567	0.4891	45.5400
198	0.7512	0.4982	0.7567	0.4890	46.1405
199	0.7512	0.4983	0.7568	0.4889	47.9292
200	0.7512	0.4983	0.7564	0.4891	48.2998
201	0.7511	0.4982	0.7566	0.4889	48.4889
202	0.7513	0.4981	0.7567	0.4889	48.3059
203	0.7512	0.4981	0.7566	0.4891	48.5652
204	0.7512	0.4980	0.7569	0.4889	48.3063
205	0.7510	0.4982	0.7569	0.4889	48.7938
206	0.7512	0.4982	0.7568	0.4888	48.6477
207	0.7512	0.4980	0.7568	0.4889	48.5303
208	0.7514	0.4979	0.7567	0.4890	48.9834
209	0.7512	0.4980	0.7568	0.4888	49.4000
210	0.7511	0.4981	0.7570	0.4886	48.4729
211	0.7515	0.4978	0.7569	0.4888	48.3141
212	0.7511	0.4980	0.7570	0.4889	49.2851
213	0.7513	0.4978	0.7572	0.4887	49.2781
214	0.7513	0.4980	0.7570	0.4887	48.7440
215	0.7517	0.4976	0.7569	0.4886	48.5652
216	0.7516	0.4977	0.7571	0.4886	48.6107
217	0.7512	0.4977	0.7569	0.4888	48.5872

```
    218       0.7513        0.4977        0.7571        0.4886  48.6941
    219       0.7513        0.4978        0.7570        0.4887  48.8698
Stopping since valid_loss has not improved in the last 10 epochs.
```

[ ]:
```
<class 'skorch.classifier.NeuralNetBinaryClassifier'>[initialized](
  module_=DNN_Drop_Tanh(
    (layers): ModuleList(
      (0): Linear(in_features=28, out_features=112, bias=True)
      (1): Dropout(p=0.25, inplace=False)
      (2): Tanh()
      (3): Linear(in_features=112, out_features=56, bias=True)
      (4): Tanh()
      (5): Linear(in_features=56, out_features=28, bias=True)
      (6): Tanh()
      (7): Linear(in_features=28, out_features=14, bias=True)
      (8): Tanh()
      (9): Linear(in_features=14, out_features=7, bias=True)
      (10): Tanh()
      (11): Linear(in_features=7, out_features=1, bias=True)
      (12): Sigmoid()
    )
  ),
)
```

[ ]:
```
y_pred_tanh = DNN_Drop_Tanh_model.predict(X_test)
```

Using prediction to calculate class-wise accuracy.

[ ]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_tanh)
tn, fp, fn, tp = cm.ravel()

# Class-wise accuracy
accuracy_0_tanh = tn / (tn + fp)
accuracy_1_tanh = tp / (tp + fn)

print("Accuracy for class 0:", accuracy_0_tanh)
print("Accuracy for class 1:", accuracy_1_tanh)
```

```
Accuracy for class 0: 0.73659102203081
Accuracy for class 1: 0.7750671807454141
```

[ ]:
```
y_pred_drop = DNN_Drop_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred_drop)
tn, fp, fn, tp = cm.ravel()

# Class-wise accuracy
```

```
accuracy_0_drop = tn / (tn + fp)
accuracy_1_drop = tp / (tp + fn)

print("Accuracy for class 0:", accuracy_0_drop)
print("Accuracy for class 1:", accuracy_1_drop)
```

```
Accuracy for class 0: 0.7181775643870101
Accuracy for class 1: 0.7920073403070538
```

[ ]:
```
y_pred_nodrop = DNN_No_Drop_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred_nodrop)
tn, fp, fn, tp = cm.ravel()

# Class-wise accuracy
accuracy_0_nodrop = tn / (tn + fp)
accuracy_1_nodrop = tp / (tp + fn)

print("Accuracy for class 0:", accuracy_0_nodrop)
print("Accuracy for class 1:", accuracy_1_nodrop)
```

```
Accuracy for class 0: 0.7665877937632773
Accuracy for class 1: 0.7570540586602867
```

[ ]:
```
y_proba_drop = DNN_Drop_model.predict_proba(X_test)[:, 1]
```

[ ]:
```
y_proba_nodrop = DNN_No_Drop_model.predict_proba(X_test)[:, 1]
y_proba_tanh = DNN_Drop_Tanh_model.predict_proba(X_test)[:, 1]
```

This section is purely for creating csvs for graph creation.

[ ]:
```
def Get_Metrics(model_name,model_history):
    epoch_training_losses = model_history[:, 'train_loss']
    epoch_validation_losses = model_history[:, 'valid_loss']
    epoch_training_accuracies = model_history[:, 'accuracy']
    epoch_validation_accuracies = model_history[:, 'valid_acc']
    epochs = model_history[:, 'epoch']
    # Models sometimes have one less accuracy
    if model_name == 'Tanh w/ Drop' or model_name == 'LeakyReLu w/ Drop':
        epoch_training_accuracies.insert(0,0)

    return pd.DataFrame({
    'epoch': epochs,
    'model': [model_name for _ in epoch_training_losses],
    'training_loss': epoch_training_losses,
    'validation_loss': epoch_validation_losses,
    'training_accuracy': epoch_training_accuracies,
    'validation_accuracy': epoch_validation_accuracies
    })
```

```
# Create a DataFrame for aggregated epoch metrics
metrics_df = Get_Metrics('LeakyReLu w/ Drop', DNN_Drop_model.history)
#metrics_df = pd.concat([metrics_df, Get_Metrics('Tanh w/ Drop',␣
↪DNN_Drop_Tanh_model.history)])
```

[ ]:
```
class_wise_acc = pd.DataFrame({
    'model': 'LeakyReLu w/ Drop',
    'accuracy0': [accuracy_0_drop],
    'accuracy1': [accuracy_1_drop],
})

#class_wise_acc = pd.concat([class_wise_acc, pd.DataFrame({
#    'model': 'LeakyReLu w/o Drop',
#     'accuracy0': [accuracy_0_nodrop],
#     'accuracy1': [accuracy_1_nodrop]
# })])

# class_wise_acc = pd.concat([class_wise_acc,  pd.DataFrame({
#     'model': 'Tanh w/ Drop',
#     'accuracy0': [accuracy_0_tanh],
#     'accuracy1': [accuracy_1_tanh]
# })])
```

[ ]:
```
results_df = pd.DataFrame({'model': 'LeakyReLu w/ Drop','y_test': y_test,␣
↪'y_predictions': y_pred_drop, 'y_proba': y_proba_drop})
#results_df = pd.concat([results_df, pd.DataFrame({'model': 'LeakyReLu w/o␣
↪Drop','y_test': y_test, 'y_predictions': y_pred_nodrop, 'y_proba':␣
↪y_proba_nodrop})])
#results_df = pd.concat([results_df, pd.DataFrame({'model': 'Tanh w/␣
↪Drop','y_test': y_test, 'y_predictions': y_pred_tanh, 'y_proba':␣
↪y_proba_tanh})])

# Exporting results
results_df.to_csv('drop_results.csv', index=False)

# Exporting metrics
metrics_df.to_csv('drop_metrics.csv', index=False)
```

[ ]:
```
# Exporting class accuracies
class_wise_acc.to_csv('drop_class_acc.csv', index=False)
```