# Analysis on All Data Files

On this notebook, I will apply the best optimized models on all five datasets.

## Load Libraries

```
In [1]:  # Import base libraries
         import pandas as pd
         import numpy as np
         from scipy.io import arff

         import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns

         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import GridSearchCV
         from xgboost import XGBClassifier

         from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import plot_confusion_matrix
         from sklearn.metrics import roc_auc_score, roc_curve, auc
         from sklearn.metrics import precision_score, recall_score, accuracy_score,
         from sklearn.utils import class_weight

         from functions import *

         from datetime import datetime

         import warnings
         warnings.filterwarnings('ignore')
```

## Load Data

There are five data files:

```
* data1, 1year.arff
* data2, 2year.arff
* data3, 3year.arff
* data4, 4year.arff
* data5, 5year.arff
```

Note: No cleaning applied to data. XGBoost Classifier can handle the missing values and outliers.

```
In [21]:  # Load all five data files

          data1 = arff.loadarff('data/1year.arff')
          df1 = pd.DataFrame(data1[0])

          data2 = arff.loadarff('data/2year.arff')
          df2 = pd.DataFrame(data2[0])

          data3 = arff.loadarff('data/3year.arff')
          df3 = pd.DataFrame(data3[0])

          data4 = arff.loadarff('data/4year.arff')
          df4 = pd.DataFrame(data4[0])

          data5 = arff.loadarff('data/5year.arff')
          df5 = pd.DataFrame(data5[0])
```

```
In [22]:  # Convert class/label type to binary

          df1['class'] = df1['class'].astype('int64')
          df2['class'] = df2['class'].astype('int64')
          df3['class'] = df3['class'].astype('int64')
          df4['class'] = df4['class'].astype('int64')
          df5['class'] = df5['class'].astype('int64')
```

```
In [23]:  # Size of datasets

          print('Size of datasets')
          print("Data 1 (Year1):", len(df1))
          print("Data 2 (Year2):", len(df2))
          print("Data 3 (Year3):", len(df3))
          print("Data 4 (Year4):", len(df4))
          print("Data 5 (Year5):", len(df5))
```

```
Size of datasets
Data 1 (Year1): 7027
Data 2 (Year2): 10173
Data 3 (Year3): 10503
Data 4 (Year4): 9792
Data 5 (Year5): 5910
```

## Imbalance Information

I am using both sample_weight and scale_pos_weight parameters to deal with the class imbalance.

- sample_weight: The weights for training sample are calculated for each dataset seperately and used when during training.

- scale_pos_weight: I provide certain values to initiate the classifier. I either use the imbalance ratio or square root of the imbalance ratio. These values are not exactly same for the datasets, but close enough to use a constant about average number.
    - Model 7 (max_depth=5): scale_pos_weight=20 (~imbalance ratio)
    - Model 8 (max_depth=4): scale_pos_weight=4.5 (~square root of imbalance ratio)

- Model 9 (max_depth=6): scale_pos_weight=20 (~imbalance ratio)

```
In [24]:  # Imbalance info using class_weight.compute_class_weight

          print('Class Weights:')

          df_list = [df1, df2, df3, df4, df5]

          for i, df in enumerate(df_list, start=1):
              class_weights = class_weight.compute_class_weight(class_weight='balance
              ratio = class_weights[1]/class_weights[0]
              sqrt_ratio = np.sqrt(class_weights[1]/class_weights[0])
              print(f'Data {i}: Ratio={round(ratio,3)}, sqrt(ratio)={round(sqrt_ratio

          # The values are very similar for train/test/whole datasets.
          # Training weights are used for data training.
```

```
Class Weights:
Data 1: Ratio=24.93, sqrt(ratio)=4.993, class_weights=[ 0.52005625 12.964
94465]
Data 2: Ratio=24.432, sqrt(ratio)=4.943, class_weights=[ 0.52046455 12.71
625   ]
Data 3: Ratio=20.218, sqrt(ratio)=4.496, class_weights=[ 0.52473022 10.60
909091]
Data 4: Ratio=18.014, sqrt(ratio)=4.244, class_weights=[0.52775682 9.5067
9612]
Data 5: Ratio=13.415, sqrt(ratio)=3.663, class_weights=[0.53727273 7.2073
1707]
```

```
In [25]:  # Imbalance info using class value_counts

          print('Imbalance Ratio, based on class value_counts:')

          df_list = [df1, df2, df3, df4, df5]

          for i, df in enumerate(df_list, start=1):
              val_counts = df['class'].value_counts()
              ratio= val_counts[0]/val_counts[1]
              sqrt_ratio= np.sqrt(val_counts[0]/val_counts[1])
              print(f'Data {i}: Ratio={round(ratio,3)}, sqrt(ratio)={round(sqrt_ratio

          # The values are very similar for train/test/whole datasets.
```

```
Imbalance Ratio, based on class value_counts:
Data 1: Ratio=24.93, sqrt(ratio)=4.993
Data 2: Ratio=24.432, sqrt(ratio)=4.943
Data 3: Ratio=20.218, sqrt(ratio)=4.496
Data 4: Ratio=18.014, sqrt(ratio)=4.244
Data 5: Ratio=13.415, sqrt(ratio)=3.663
```

## Compare Model 7, 9, 10 Performance on All Datasets

```
In [26]: # Model 9: Best Model for Data 3, max_depth=4

         xgbParams_m9 = {
             'eval_metric': 'logloss',
             'random_state': 42,
             'scale_pos_weight': 4.5,
             'n_estimators': 125,
             'max_depth': 4,
             'min_child_weight': 3,
             'gamma': 0,
             'learning_rate': 0.20,
             'max_delta_step': 0,
             'reg_lambda': 1,
             'reg_alpha': 0,
             'subsample': 1,
             'colsample_bytree': 1
         }

         df_list = [df1, df2, df3, df4, df5]
         #df_list = [df1, df2]

         model_9_df = compare_datafiles_perf(df_list, xgbParams_m9, 'model_9', 1, 1,
         model_9_df
```
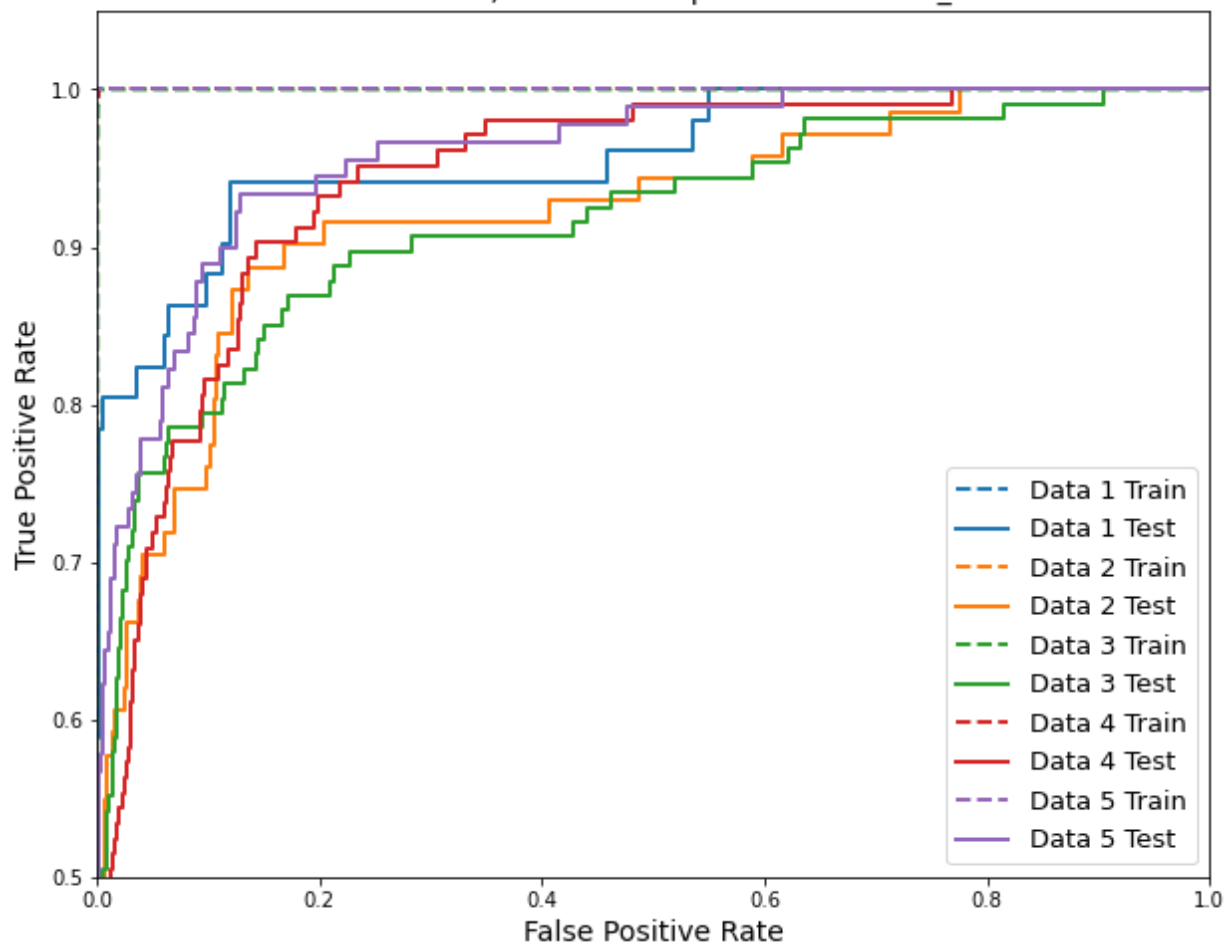
```
------model_9------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[26]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.9610 | 1.0000 | 0.9800 | 0.998 | 1.0000 |
| Data 1 | Test | 0.7190 | 0.8040 | 0.7590 | 0.982 | 0.9570 |
| Data 2 | Train | 0.8410 | 1.0000 | 0.9140 | 0.992 | 1.0000 |
| Data 2 | Test | 0.4830 | 0.6060 | 0.5380 | 0.964 | 0.9210 |
| Data 3 | Train | 0.7190 | 1.0000 | 0.8360 | 0.982 | 1.0000 |
| Data 3 | Test | 0.5540 | 0.7200 | 0.6260 | 0.956 | 0.9160 |
| Data 4 | Train | 0.7440 | 1.0000 | 0.8530 | 0.982 | 1.0000 |
| Data 4 | Test | 0.4960 | 0.6600 | 0.5670 | 0.947 | 0.9420 |
| Data 5 | Train | 0.9500 | 1.0000 | 0.9740 | 0.996 | 1.0000 |
| Data 5 | Test | 0.6600 | 0.7330 | 0.6950 | 0.951 | 0.9580 |
| Average | Train | 0.8430 | 1.0000 | 0.9114 | 0.990 | 1.0000 |
| Average | Test | 0.5824 | 0.7046 | 0.6370 | 0.960 | 0.9388 |

ROC Curve, Dataset Comparison for model_9

```
In [29]: # Model 7, max_depth=5

xgbParams_m7 = {
    'eval_metric': 'logloss',
    'random_state': 42,
    'scale_pos_weight': 20,
    'n_estimators': 125,
    'max_depth': 5,
    'min_child_weight': 3,
    'gamma': 0,
    'learning_rate': 0.20,
    'max_delta_step': 0,
    'reg_lambda': 0,
    'reg_alpha': 5,
    'subsample': 1,
    'colsample_bytree': 0.7
}

df_list = [df1, df2, df3, df4, df5]

model_7_df = compare_datafiles_perf(df_list, xgbParams_m7, 'model_7', 1, 1,
model_7_df
```
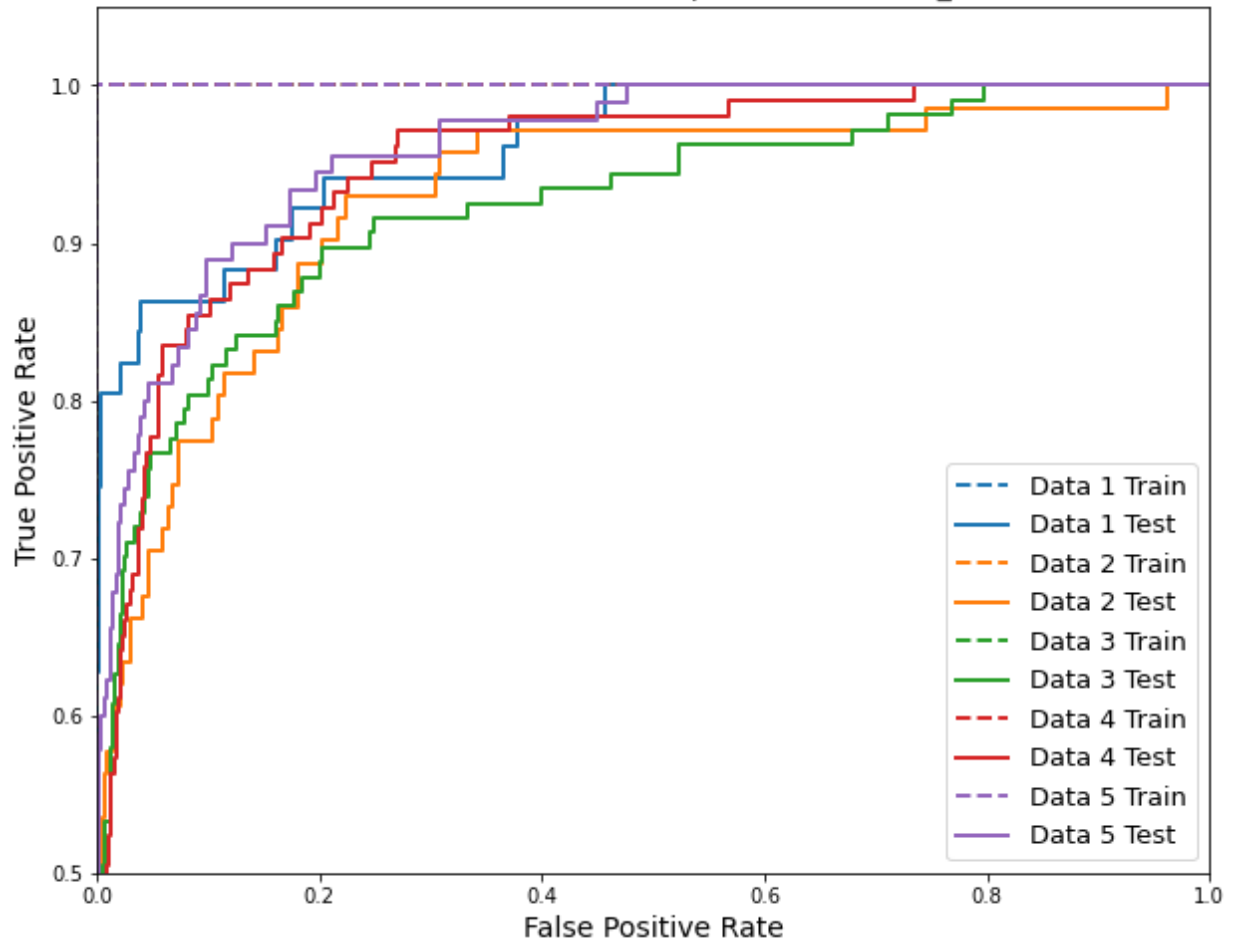
```
------model_7------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[29]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.8700 | 1.0000 | 0.9300 | 0.9940 | 1.0000 |
| Data 1 | Test | 0.6410 | 0.8040 | 0.7130 | 0.9770 | 0.9610 |
| Data 2 | Train | 0.8750 | 1.0000 | 0.9330 | 0.9940 | 1.0000 |
| Data 2 | Test | 0.5060 | 0.6200 | 0.5570 | 0.9660 | 0.9280 |
| Data 3 | Train | 0.7710 | 1.0000 | 0.8710 | 0.9860 | 1.0000 |
| Data 3 | Test | 0.5350 | 0.7200 | 0.6140 | 0.9540 | 0.9230 |
| Data 4 | Train | 0.8060 | 1.0000 | 0.8930 | 0.9870 | 1.0000 |
| Data 4 | Test | 0.5560 | 0.6800 | 0.6110 | 0.9550 | 0.9490 |
| Data 5 | Train | 0.8290 | 1.0000 | 0.9070 | 0.9860 | 1.0000 |
| Data 5 | Test | 0.6070 | 0.7890 | 0.6860 | 0.9450 | 0.9600 |
| Average | Train | 0.8302 | 1.0000 | 0.9068 | 0.9894 | 1.0000 |
| Average | Test | 0.5690 | 0.7226 | 0.6362 | 0.9594 | 0.9442 |

ROC Curve, Dataset Comparison for model_7

```
In [27]: # Model 10, max_depth=6

xgbParams_m10 = {
    'eval_metric': 'logloss',
    'random_state': 42,
    'scale_pos_weight': 20,
    'n_estimators': 80,
    'max_depth': 6,
    'min_child_weight': 3,
    'gamma': 0,
    'learning_rate': 0.25,
    'max_delta_step': 4,
    'reg_lambda': 1,
    'reg_alpha': 0,
    'subsample': 1,
    'colsample_bytree': 1,
}

df_list = [df1, df2, df3, df4, df5]

model_10_df = compare_datafiles_perf(df_list, xgbParams_m10, 'model_10', 1,
model_10_df
```

```
------model_10------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[27]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.9730 | 1.0000 | 0.9870 | 0.9990 | 1.000 |
| Data 1 | Test | 0.7140 | 0.7840 | 0.7480 | 0.9810 | 0.951 |
| Data 2 | Train | 0.9620 | 1.0000 | 0.9810 | 0.9980 | 1.000 |
| Data 2 | Test | 0.6340 | 0.6340 | 0.6340 | 0.9740 | 0.909 |
| Data 3 | Train | 0.9000 | 1.0000 | 0.9470 | 0.9950 | 1.000 |
| Data 3 | Test | 0.6360 | 0.7010 | 0.6670 | 0.9640 | 0.931 |
| Data 4 | Train | 0.8980 | 1.0000 | 0.9460 | 0.9940 | 1.000 |
| Data 4 | Test | 0.5640 | 0.6410 | 0.6000 | 0.9550 | 0.948 |
| Data 5 | Train | 0.9700 | 1.0000 | 0.9850 | 0.9980 | 1.000 |
| Data 5 | Test | 0.6370 | 0.7220 | 0.6770 | 0.9480 | 0.951 |
| Average | Train | 0.9406 | 1.0000 | 0.9692 | 0.9968 | 1.000 |
| Average | Test | 0.6370 | 0.6964 | 0.6652 | 0.9644 | 0.938 |

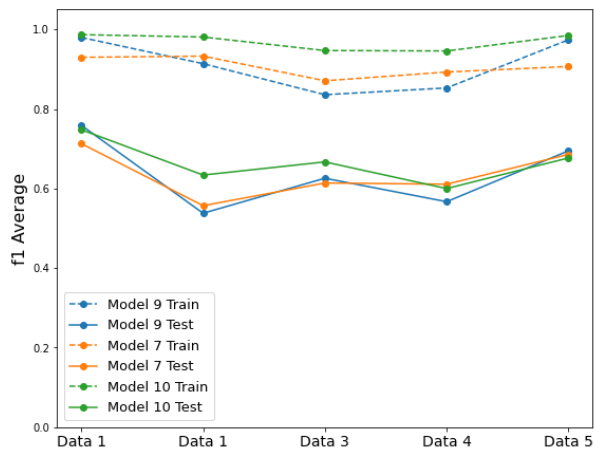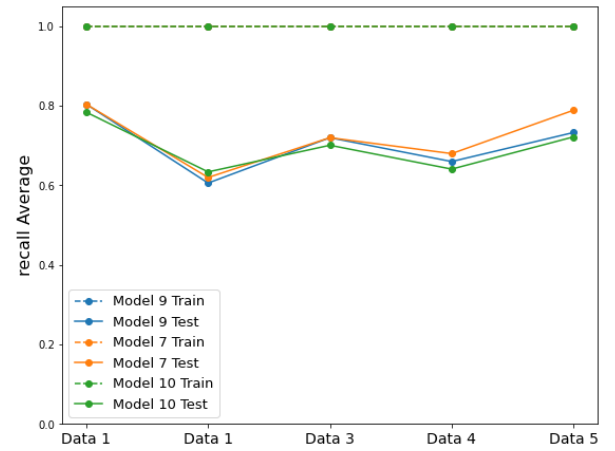ROC Curve, Dataset Comparison for model_10
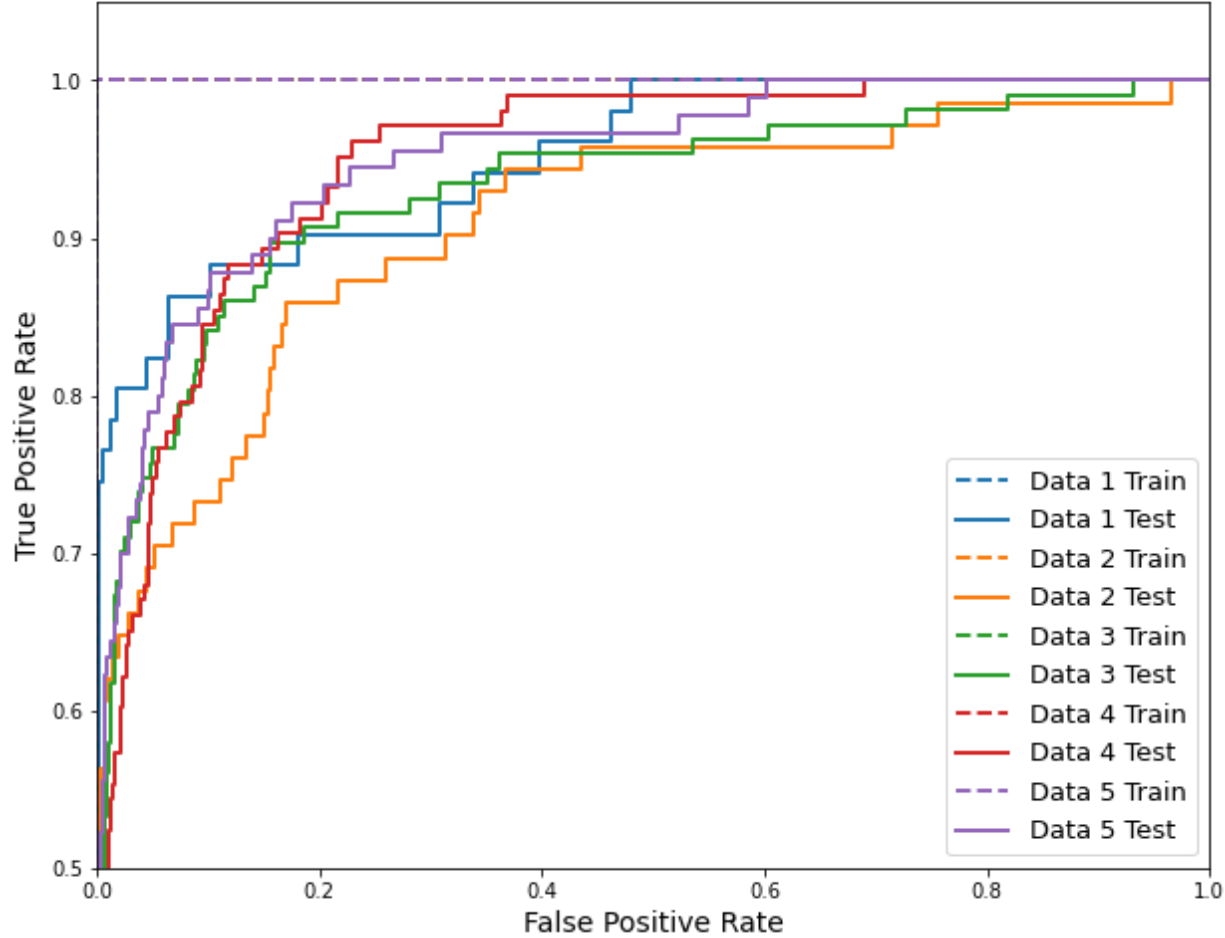
```python
model_df_list = [model_9_df, model_7_df, model_10_df]

model_names_list = ['Model 9', 'Model 7', 'Model 10']

plot_compare_model_metricsAvg(model_df_list, model_names_list, 1)
```
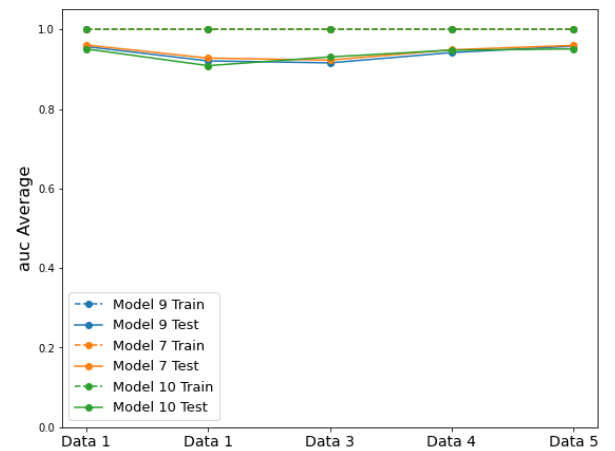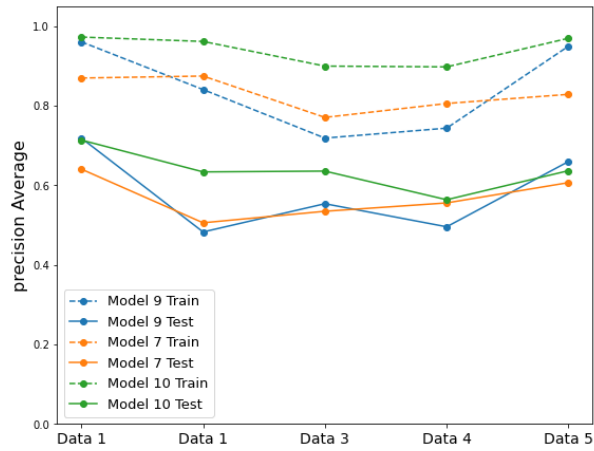


## Comments

Model 9 (max_depth=4):

```
* Previously Selected performance for data 3
* Overfitting is least on Data 3
```

Model 7 (max_depth=5):

```
* Performance is similar to Model 9
* More smooth performance on all datasets
```

Model 10 (max_depth=6):

```
* Highest metrics
* But overfitting is sligtly larger
```

Selected model: Model 7

# Some Tuning and Comparing

```
In [9]: # sample_weight only, no scale_pos_weight

xgbParams = {
    'eval_metric': 'logloss',
    'random_state': 42,
    #'scale_pos_weight': 4.5,
    'n_estimators': 125,
    'max_depth': 4,
    'min_child_weight': 3,
    'gamma': 0,
    'learning_rate': 0.20,
    'max_delta_step': 0,
    'reg_lambda': 1,
    'reg_alpha': 0,
    'subsample': 1,
    'colsample_bytree': 1
}

df_list = [df1, df2, df3, df4, df5]

model_noSpw_df = compare_datafiles_perf(df_list, xgbParams, 'noSpw', 1, 0,
model_noSpw_df
```

```
------noSpw------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[9]:

| Sample Data | | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.9780 | 1.0000 | 0.9890 | 0.9990 | 1.0000 |
| Data 1 | Test | 0.8700 | 0.7840 | 0.8250 | 0.9880 | 0.9680 |
| Data 2 | Train | 0.9450 | 1.0000 | 0.9720 | 0.9980 | 1.0000 |
| Data 2 | Test | 0.6830 | 0.5770 | 0.6260 | 0.9760 | 0.9360 |
| Data 3 | Train | 0.8840 | 1.0000 | 0.9380 | 0.9940 | 1.0000 |
| Data 3 | Test | 0.7000 | 0.6540 | 0.6760 | 0.9680 | 0.9310 |
| Data 4 | Train | 0.9260 | 1.0000 | 0.9610 | 0.9960 | 1.0000 |
| Data 4 | Test | 0.6090 | 0.5150 | 0.5580 | 0.9570 | 0.9450 |
| Data 5 | Train | 0.9730 | 1.0000 | 0.9860 | 0.9980 | 1.0000 |
| Data 5 | Test | 0.7360 | 0.7110 | 0.7230 | 0.9590 | 0.9580 |
| Average | Train | 0.9412 | 1.0000 | 0.9692 | 0.9970 | 1.0000 |
| Average | Test | 0.7196 | 0.6482 | 0.6816 | 0.9696 | 0.9476 |

```
In [10]: #n_estimators=100

xgbParams_11 = {
    'eval_metric': 'logloss',
    'random_state': 42,
    'scale_pos_weight': 4.5,
    'n_estimators': 100,
    'max_depth': 4,
    'min_child_weight': 3,
    'gamma': 0,
    'learning_rate': 0.20,
    'max_delta_step': 0,
    'reg_lambda': 1,
    'reg_alpha': 0,
    'subsample': 1,
    'colsample_bytree': 1
}

df_list = [df1, df2, df3, df4, df5]

model_nEst100_df = compare_datafiles_perf(df_list, xgbParams_11, 'nEst100',
model_nEst100_df
```

```
------nEst100------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[10]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.8980 | 1.0000 | 0.946 | 0.9960 | 1.000 |
| Data 1 | Test | 0.6610 | 0.8040 | 0.726 | 0.9780 | 0.957 |
| Data 2 | Train | 0.6730 | 1.0000 | 0.804 | 0.9800 | 1.000 |
| Data 2 | Test | 0.4020 | 0.6620 | 0.500 | 0.9540 | 0.925 |
| Data 3 | Train | 0.6090 | 1.0000 | 0.757 | 0.9700 | 1.000 |
| Data 3 | Test | 0.4910 | 0.7480 | 0.593 | 0.9480 | 0.916 |
| Data 4 | Train | 0.6250 | 1.0000 | 0.769 | 0.9680 | 1.000 |
| Data 4 | Test | 0.4270 | 0.6800 | 0.524 | 0.9350 | 0.938 |
| Data 5 | Train | 0.8420 | 1.0000 | 0.914 | 0.9870 | 1.000 |
| Data 5 | Test | 0.6110 | 0.7330 | 0.667 | 0.9440 | 0.959 |
| Average | Train | 0.7294 | 1.0000 | 0.838 | 0.9802 | 1.000 |
| Average | Test | 0.5184 | 0.7254 | 0.602 | 0.9518 | 0.939 |

```
In [11]: # reg_lambda=5

xgbParams = {
    'eval_metric': 'logloss',
    'random_state': 42,
    'scale_pos_weight': 4.5,
    'n_estimators': 125,
    'max_depth': 4,
    'min_child_weight': 3,
    'gamma': 0,
    'learning_rate': 0.20,
    'max_delta_step': 0,
    'reg_lambda': 5,
    'reg_alpha': 0,
    'subsample': 1,
    'colsample_bytree': 1
}

df_list = [df1, df2, df3, df4, df5]

model_lambda5_df = compare_datafiles_perf(df_list, xgbParams, 'lambda5', 1,
model_lambda5_df
```

```
------lambda5------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[11]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.9170 | 1.0000 | 0.9570 | 0.9960 | 1.000 |
| Data 1 | Test | 0.7320 | 0.8040 | 0.7660 | 0.9820 | 0.968 |
| Data 2 | Train | 0.7950 | 1.0000 | 0.8860 | 0.9900 | 1.000 |
| Data 2 | Test | 0.4730 | 0.6200 | 0.5370 | 0.9630 | 0.940 |
| Data 3 | Train | 0.6470 | 1.0000 | 0.7850 | 0.9750 | 1.000 |
| Data 3 | Test | 0.5070 | 0.7200 | 0.5950 | 0.9500 | 0.924 |
| Data 4 | Train | 0.7330 | 1.0000 | 0.8460 | 0.9810 | 1.000 |
| Data 4 | Test | 0.5220 | 0.6890 | 0.5940 | 0.9500 | 0.952 |
| Data 5 | Train | 0.8600 | 1.0000 | 0.9250 | 0.9890 | 1.000 |
| Data 5 | Test | 0.6190 | 0.7780 | 0.6900 | 0.9470 | 0.961 |
| Average | Train | 0.7904 | 1.0000 | 0.8798 | 0.9862 | 1.000 |
| Average | Test | 0.5706 | 0.7222 | 0.6364 | 0.9584 | 0.949 |

```
In [12]:  #n_estimators=100
          # reg_lambda=5

          xgbParams = {
              'eval_metric': 'logloss',
              'random_state': 42,
              'scale_pos_weight': 4.5,
              'n_estimators': 100,
              'max_depth': 4,
              'min_child_weight': 3,
              'gamma': 0,
              'learning_rate': 0.20,
              'max_delta_step': 0,
              'reg_lambda': 5,
              'reg_alpha': 0,
              'subsample': 1,
              'colsample_bytree': 1
          }

          df_list = [df1, df2, df3, df4, df5]

          model_nEst100_lambda5_df = compare_datafiles_perf(df_list, xgbParams, 'nEst
          model_nEst100_lambda5_df
```

```
------nEst100_lambda5------
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
Sample weights are used!
```

Out[12]:

| Sample Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.8430 | 1.0000 | 0.9150 | 0.9930 | 1.000 |
| Data 1 | Test | 0.6360 | 0.8240 | 0.7180 | 0.9770 | 0.967 |
| Data 2 | Train | 0.6630 | 1.0000 | 0.7980 | 0.9790 | 1.000 |
| Data 2 | Test | 0.4380 | 0.6900 | 0.5360 | 0.9580 | 0.935 |
| Data 3 | Train | 0.5390 | 1.0000 | 0.7000 | 0.9600 | 1.000 |
| Data 3 | Test | 0.4180 | 0.7570 | 0.5380 | 0.9340 | 0.920 |
| Data 4 | Train | 0.6000 | 1.0000 | 0.7500 | 0.9650 | 1.000 |
| Data 4 | Test | 0.4420 | 0.6990 | 0.5410 | 0.9380 | 0.947 |
| Data 5 | Train | 0.7860 | 1.0000 | 0.8800 | 0.9820 | 1.000 |
| Data 5 | Test | 0.6100 | 0.8330 | 0.7040 | 0.9470 | 0.961 |
| Average | Train | 0.6862 | 1.0000 | 0.8086 | 0.9758 | 1.000 |
| Average | Test | 0.5088 | 0.7606 | 0.6074 | 0.9508 | 0.946 |

```
In [29]: model_df_list2 = [model_7_df, model_noSpw_df, model_nEst100_df, model_lambd

         model_names_list2 = ['Model 7', 'nsSpw', 'nEst100', 'lambda5', 'nEst100_lam

         plot_compare_model_metricsAvg(model_df_list2, model_names_list2)
```
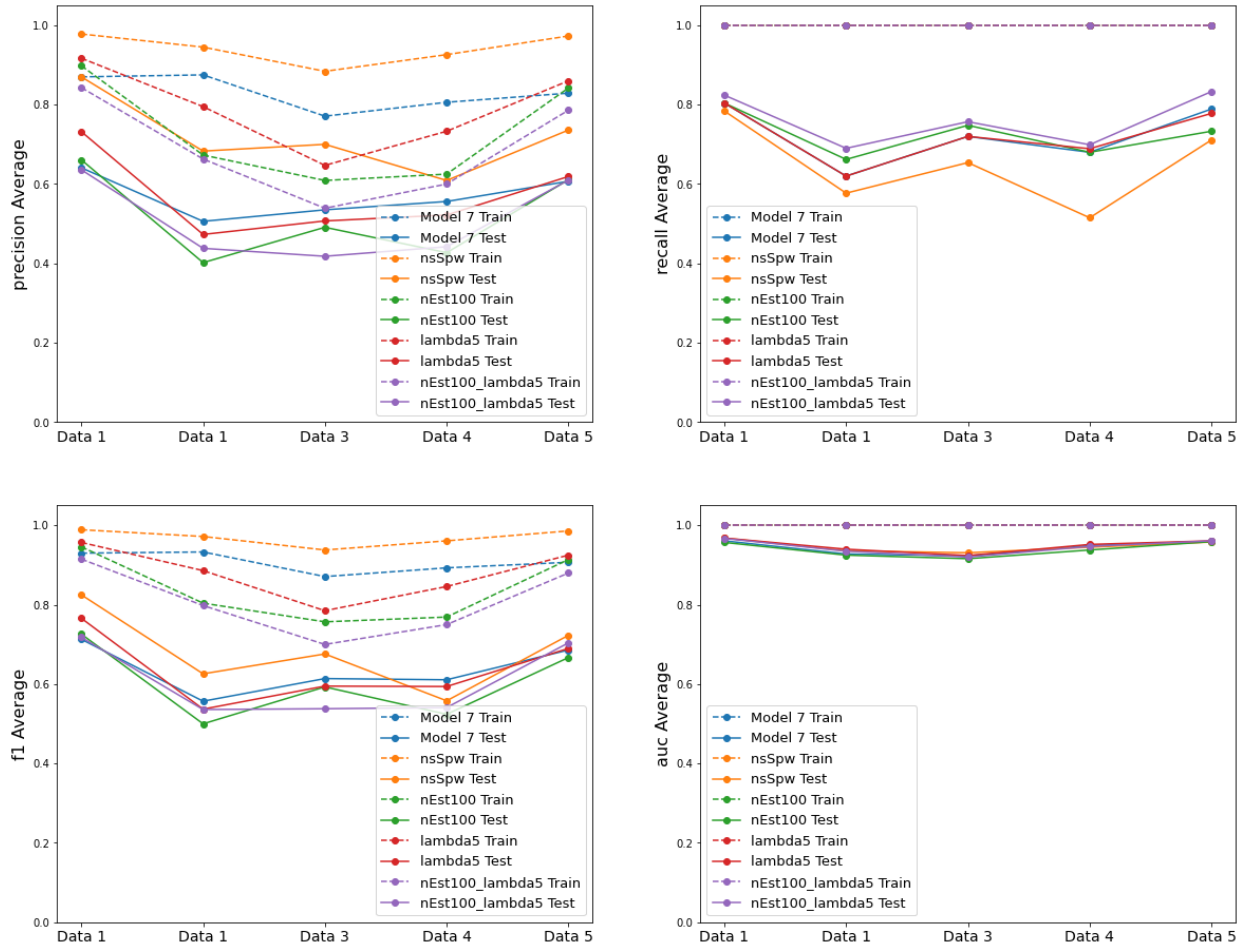


## Comments

After some tuning, still best performance is Model 7.

# Final Model

Model 7 has the best performance.

```python
In [7]:  # Final Model Parameters
         # Model 7, max_depth=5

         xgbParams_final = {
             'eval_metric': 'logloss',
             'random_state': 42,
             'scale_pos_weight': 20,
             'n_estimators': 125,
             'max_depth': 5,
             'min_child_weight': 3,
             'gamma': 0,
             'learning_rate': 0.20,
             'max_delta_step': 0,
             'reg_lambda': 0,
             'reg_alpha': 5,
             'subsample': 1,
             'colsample_bytree': 0.7
         }
```

```
In [8]: # Final Model Results for All data files

        df_list = [df1, df2, df3, df4, df5]

        final_model_df = compare_datafiles_perf(df_list, xgbParams_final, 'final_mo
        final_model_df
```

------final_model------
Sample weights are used!
Sample weights are used!
Sample weights are used!
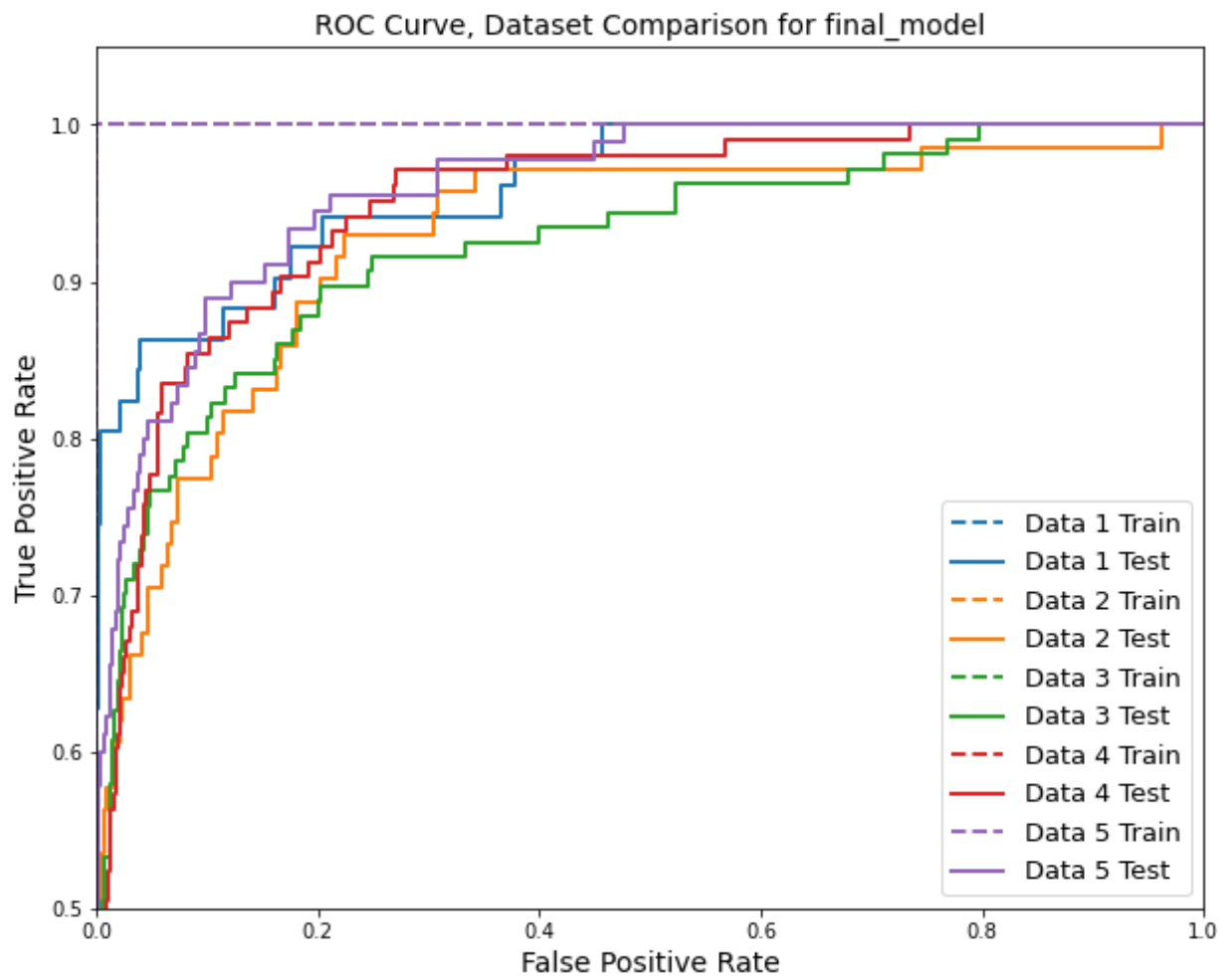Sample weights are used!
Sample weights are used!

Out[8]:

| Data | Sample | precision | recall | f1 | accuracy | auc |
|---|---|---|---|---|---|---|
| Data 1 | Train | 0.8700 | 1.0000 | 0.9300 | 0.9940 | 1.0000 |
| Data 1 | Test | 0.6410 | 0.8040 | 0.7130 | 0.9770 | 0.9610 |
| Data 2 | Train | 0.8750 | 1.0000 | 0.9330 | 0.9940 | 1.0000 |
| Data 2 | Test | 0.5060 | 0.6200 | 0.5570 | 0.9660 | 0.9280 |
| Data 3 | Train | 0.7710 | 1.0000 | 0.8710 | 0.9860 | 1.0000 |
| Data 3 | Test | 0.5350 | 0.7200 | 0.6140 | 0.9540 | 0.9230 |
| Data 4 | Train | 0.8060 | 1.0000 | 0.8930 | 0.9870 | 1.0000 |
| Data 4 | Test | 0.5560 | 0.6800 | 0.6110 | 0.9550 | 0.9490 |
| Data 5 | Train | 0.8290 | 1.0000 | 0.9070 | 0.9860 | 1.0000 |
| Data 5 | Test | 0.6070 | 0.7890 | 0.6860 | 0.9450 | 0.9600 |
| Average | Train | 0.8302 | 1.0000 | 0.9068 | 0.9894 | 1.0000 |
| Average | Test | 0.5690 | 0.7226 | 0.6362 | 0.9594 | 0.9442 |

## ROC Curve, Dataset Comparison for final_model



Legend:
- Data 1 Train (dashed)
- Data 1 Test
- Data 2 Train (dashed)
- Data 2 Test
- Data 3 Train (dashed)
- Data 3 Test
- Data 4 Train (dashed)
- Data 4 Test
- Data 5 Train (dashed)
- Data 5 Test

```
In [9]:  # Data 1 Final Model
         d1_final_model = xgb_model_report2(1, df1, xgbParams_final, 'final_model',
```

Sample weights are used!
Data 1 Classification Report:

Training Data:

|  | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 | 5401 |
| 1 | 0.87 | 1.00 | 0.93 | 220 |
| | | | | |
| accuracy | | | 0.99 | 5621 |
| macro avg | 0.93 | 1.00 | 0.96 | 5621 |
| weighted avg | 0.99 | 0.99 | 0.99 | 5621 |

Testing Data:

|  | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.98 | 0.99 | 1355 |
| 1 | 0.64 | 0.80 | 0.71 | 51 |
| | | | | |
| accuracy | | | 0.98 | 1406 |
| macro avg | 0.82 | 0.89 | 0.85 | 1406 |
| weighted avg | 0.98 | 0.98 | 0.98 | 1406 |

```
In [10]:  # Data 2 Final Model
          d2_final_model = xgb_model_report2(2, df2, xgbParams_final, 'final_model',
```

Sample weights are used!
Data 2 Classification Report:

Training Data:
               precision    recall  f1-score   support

           0       1.00      0.99      1.00      7809
           1       0.88      1.00      0.93       329

    accuracy                           0.99      8138
   macro avg       0.94      1.00      0.97      8138
weighted avg       0.99      0.99      0.99      8138

Testing Data:
               precision    recall  f1-score   support

           0       0.99      0.98      0.98      1964
           1       0.51      0.62      0.56        71

    accuracy                           0.97      2035
   macro avg       0.75      0.80      0.77      2035
weighted avg       0.97      0.97      0.97      2035

```
In [11]:  # Data 3 Final Model
          d3_final_model = xgb_model_report2(3, df3, xgbParams_final, 'final_model',
```

Sample weights are used!
Data 3 Classification Report:

Training Data:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99      8014
           1       0.77      1.00      0.87       388

    accuracy                           0.99      8402
   macro avg       0.89      0.99      0.93      8402
weighted avg       0.99      0.99      0.99      8402

Testing Data:
               precision    recall  f1-score   support

           0       0.98      0.97      0.98      1994
           1       0.53      0.72      0.61       107

    accuracy                           0.95      2101
   macro avg       0.76      0.84      0.79      2101
weighted avg       0.96      0.95      0.96      2101

```
In [12]:  # Data 4 Final Model
          d4_final_model = xgb_model_report2(4, df4, xgbParams_final, 'final_model',
```

Sample weights are used!
Data 4 Classification Report:

Training Data:
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      7421
           1       0.81      1.00      0.89       412

    accuracy                           0.99      7833
   macro avg       0.90      0.99      0.94      7833
weighted avg       0.99      0.99      0.99      7833
```

Testing Data:
```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1856
           1       0.56      0.68      0.61       103

    accuracy                           0.95      1959
   macro avg       0.77      0.82      0.79      1959
weighted avg       0.96      0.95      0.96      1959
```

```
In [13]:  # Data 5 Final Model
          d5_final_model = xgb_model_report2(5, df5, xgbParams_final, 'final_model',
```

Sample weights are used!
Data 5 Classification Report:

Training Data:
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      4408
           1       0.83      1.00      0.91       320

    accuracy                           0.99      4728
   macro avg       0.91      0.99      0.95      4728
weighted avg       0.99      0.99      0.99      4728
```

Testing Data:
```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1092
           1       0.61      0.79      0.69        90

    accuracy                           0.95      1182
   macro avg       0.79      0.87      0.83      1182
weighted avg       0.95      0.95      0.95      1182
```

## Best Predictor Atttributes

```
In [14]: # List important attitudes

         final_model_list = [d1_final_model, d2_final_model, d3_final_model, d4_fina

         df_important_Attr = pd.DataFrame()
         cols = df1.columns

         for i,df in enumerate(final_model_list, start=1):
             importance = df.feature_importances_
             attribute_importance = pd.DataFrame([cols, importance], index=['Attribu
             attribute_importance.sort_values(by='Importance', ascending=False, inpl
             df_important_Attr[f'Data{i}'] = attribute_importance['Attribute'][0:20]

         df_important_Attr
```

Out[14]:

|    | Data1  | Data2  | Data3  | Data4  | Data5  |
|----|--------|--------|--------|--------|--------|
| 0  | Attr6  | Attr34 | Attr34 | Attr62 | Attr62 |
| 1  | Attr34 | Attr59 | Attr52 | Attr33 | Attr34 |
| 2  | Attr22 | Attr10 | Attr10 | Attr31 | Attr64 |
| 3  | Attr24 | Attr5  | Attr26 | Attr6  | Attr7  |
| 4  | Attr27 | Attr7  | Attr27 | Attr34 | Attr27 |
| 5  | Attr8  | Attr26 | Attr25 | Attr27 | Attr6  |
| 6  | Attr25 | Attr27 | Attr35 | Attr26 | Attr42 |
| 7  | Attr1  | Attr11 | Attr19 | Attr42 | Attr16 |
| 8  | Attr49 | Attr15 | Attr59 | Attr52 | Attr49 |
| 9  | Attr45 | Attr25 | Attr31 | Attr13 | Attr5  |
| 10 | Attr5  | Attr16 | Attr13 | Attr57 | Attr48 |
| 11 | Attr13 | Attr24 | Attr24 | Attr21 | Attr25 |
| 12 | Attr21 | Attr41 | Attr16 | Attr64 | Attr51 |
| 13 | Attr59 | Attr23 | Attr37 | Attr41 | Attr43 |
| 14 | Attr51 | Attr58 | Attr58 | Attr1  | Attr31 |
| 15 | Attr37 | Attr20 | Attr6  | Attr5  | Attr23 |
| 16 | Attr35 | Attr42 | Attr2  | Attr37 | Attr2  |
| 17 | Attr48 | Attr17 | Attr5  | Attr24 | Attr50 |
| 18 | Attr50 | Attr13 | Attr21 | Attr59 | Attr21 |
| 19 | Attr42 | Attr22 | Attr12 | Attr15 | Attr14 |

```
In [15]:  #Common Best predictors

          set1 = set(df_important_Attr['Data1'])
          set2 = set(df_important_Attr['Data2'])
          set3 = set(df_important_Attr['Data3'])
          set4 = set(df_important_Attr['Data4'])
          set5 = set(df_important_Attr['Data5'])

          important_attr_common = set.intersection(set1, set2, set3, set4, set5)
          important_attr_common
```
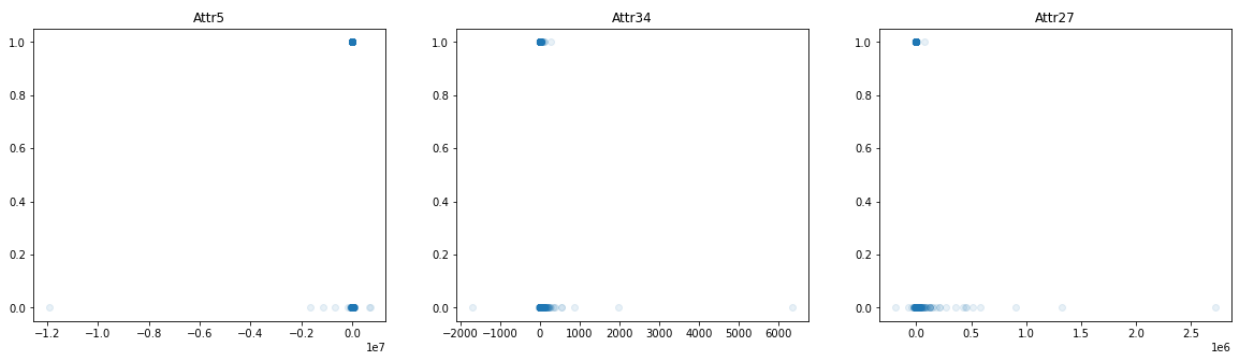
Out[15]: {'Attr27', 'Attr34', 'Attr5'}

```
In [19]:  # Scatter Graphs, Data 3

          plt.figure(figsize=(20, 5))

          for i, col in enumerate(important_attr_common, start=1):
              plt.subplot(1, 3, i)
              plt.scatter(df3[col], df3['class'], alpha=0.1)
              plt.title(col)

          plt.savefig('figures/scatter_d3_importantAttr.png')
```

```
In [20]: # Histograms

plt.figure(figsize=(20, 5))

for i, col in enumerate(important_attr_common, start=1):
    plt.subplot(1, 3, i)
    plt.hist(df3[col], bins=100, log=True)
    plt.title(col)

plt.savefig('figures/hist_d3_log_importantAttr.png')
```
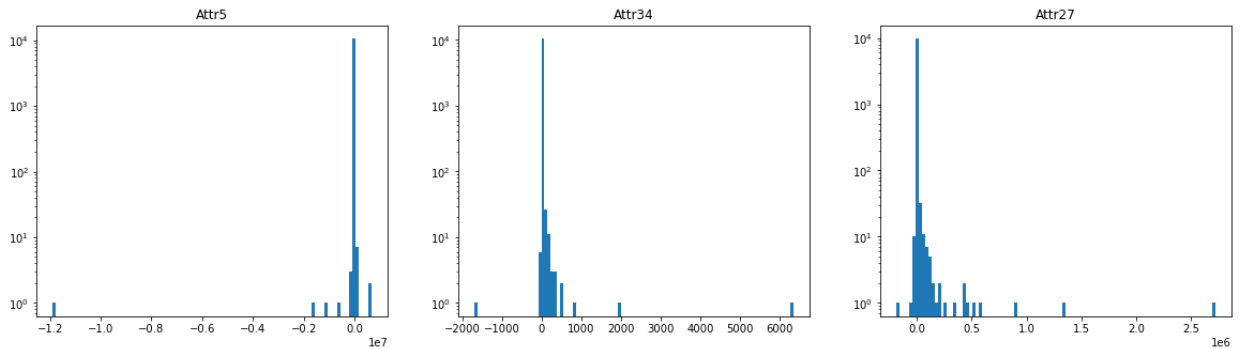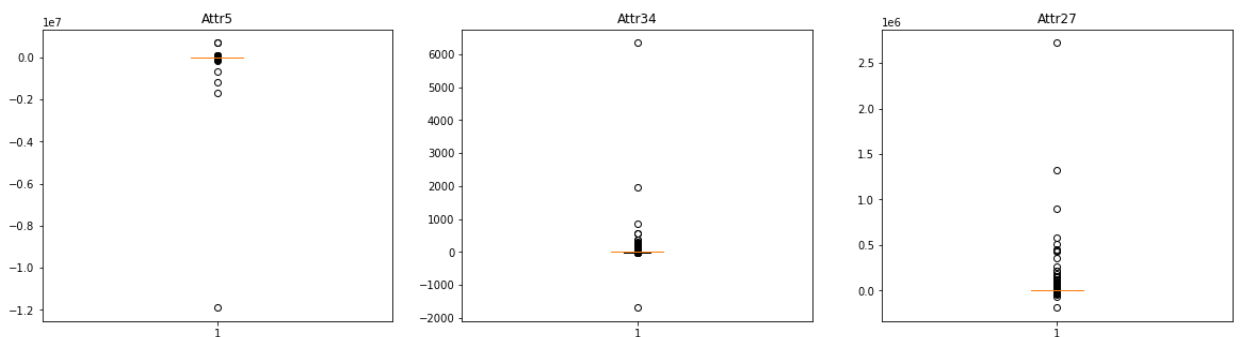


```
In [18]: # Box plots

plt.figure(figsize=(20, 5))

for i, col in enumerate(important_attr_common, start=1):
    plt.subplot(1, 3, i)
    plt.boxplot(df3[col].dropna())
    plt.title(col)

plt.savefig('figures/boxPlot_d3_importantAttr.png')

## Boxplot only draws the attribute only when missing entries removed.
```



# Interpretation of Results

5-year Period (Data 1):

- Model correctly identifies the 80.4% of the true bankrupt companies, which will bankrupt 5 years later. (recall)
- Among the model predicted bankruptcy companies, 64.1% of them are true bankrupt companies, which will bankrupt 5 years later. (precision)

- The Harmonic Mean of Precision and Recall (f1-score) is 71.3%.

4-year Period (Data 2):

- Model correctly identifies the 62.0% of the true bankrupt companies, which will bankrupt 4 years later. (recall)
- Among the model predicted bankruptcy companies, 50.6% of them are true bankrupt companies, which will bankrupt 4 years later. (precision)
- The Harmonic Mean of Precision and Recall (f1-score) is 55.7%.

3-year Period (Data 3):

- Model correctly identifies the 72.0% of the true bankrupt companies, which will bankrupt 3 years later. (recall)
- Among the model predicted bankruptcy companies, 53.5% of them are true bankrupt companies, which will bankrupt 3 years later. * The Harmonic Mean of Precision and Recall (f1-score) is 61.4%.

2-year Period (Data 4):

- Model correctly identifies the 68.0% of the true bankrupt companies, which will bankrupt 2 years later. (recall)
- Among the model predicted bankruptcy companies, 55.6% of them are true bankrupt companies, which will bankrupt 2 years later. (precision)
- The Harmonic Mean of Precision and Recall (f1-score) is 61.1%.

1-year Period (Data 5):

- Model correctly identifies the 78.9% of the true bankrupt companies, which will bankrupt 1 years later. (recall)
- Among the model predicted bankruptcy companies, 60.7% of them are true bankrupt companies, which will bankrupt 1 years later. (precision)
- The Harmonic Mean of Precision and Recall (f1-score) is 68.6%.

On Average:

- Model correctly identifies the 72.3% of the true bankrupt companies. (recall)
- Among the model predicted bankruptcy companies, 56.9% of them are true bankrupt companies. (precision)
- The Harmonic Mean of Precision and Recall (f1-score) is 63.6%.

Class 0 predictions:

- Model correctly identifies the ~97% of the true still operating companies. (recall, class 0)
- Among the model predicted still operating companies, ~98% of them are true still operating companies. (precision, class 0)
- The Harmonic Mean of Precision and Recall (f1-score, class 0) is ~98%.

**Best common predictors**

- X27: profit on operating activities / financial expenses
- X34: operating expenses / total liabilities

- X5: [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365

# Conclusion

I had three main challenges in this project:

1. **Class Imbalance:**

   - Resolved.
   - Balancing the sample increased the recall and so decreased the precision. The balance sample has moderate evaluation scores. See Model 2 for Data 3.
   - However overfitting didn't improve much.

2. **Low recall score:**

   - I managed to improve recall significantly for all datasets. For instance, Data 3 recall score improved from 0.467 to 0.720.
   - I couldn't go for higher recall value, because the precision was decreasing dramatically (below 0.5). The recall and precision are inversely proportional.

3. **Large overfitting:**

   - I did decrease the overfitting, but not on desired level.
   - I tuned parameters which are effecive on overfitting, and find the optimum designs that produces low overfitting, large recall and moderate precision.
   - However, I couldn't enforce larger reduction in overfitting, since it causes the precision go below 0.5. which is the random guess probability.

Overall, my model correctly identifies

- 72.3% of the true bankrupt companies
- 97% of the true still operating companies

# Future Work

- Create seperate final models for each dataset; not just one final model that applies on all.
- Search for alternative Classifier methods/tools.
- Simply/shorten functions that are created during the project. They have repeating codes.

Type *Markdown* and LaTeX: $\alpha^2$