MAIN

October 22, 2020

[19]:
```
import seaborn as sns
import os
import pandas as pd
import numpy as np
import json
import plotly.graph_objects as go
import copy

from plotly.subplots import make_subplots
from scipy import stats

from misc import create_all_paths, get_class_dist_timeseries
from cleaning import load_arff_files
from feature_selecting import GreedyForwardSelector, CorrelationSelector
from modelling import CrossValidation, XGBoost, LogisticRegression
from preprocessing import Standardizer, MeanReplacement, clip_outliers,␣
↪ReSampler, CorrelationRemover, PCA, remove_outliers
from evaluation import get_f1_score, get_accuracy, get_auc, get_precision,␣
↪get_recall, get_all_measures, \
    get_threshold_for_optim_cost, get_weighted_accuracy,␣
↪get_params_for_best_measure_overall, get_auc, \
    get_params_for_best_model
from visualization import get_roc_curve
from __main_fs__ import main_corr, main_greedy, pred_function

# I am aware of this but get warnings nevetheless since i copy the dataframe in␣
↪some steps which sets the attribute is_copy
pd.set_option('chained_assignment',None)
create_all_paths()
```

[3]:
```
############### Exploration #################
dfs, file_names = load_arff_files("../data/raw_data/")
assert len(dfs) > 0, "You forgot to copy the arff files inside the data/
↪raw_data directory"
year_one = dfs[0]
year_one.head()
```

```
[3]:    net profit / total assets   total liabilities / total assets   \
   0                   0.089414                           0.656650
   1                   0.092379                           0.643260
   2                  -1.372700                           1.990500
   3                   0.056093                           0.524820
   4                   0.084977                           0.096562

      working capital / total assets   current assets / short-term liabilities   \
   0                         0.18322                                   1.33280
   1                         0.12866                                   1.26180
   2                        -1.06240                                   0.46628
   3                         0.30847                                   1.60930
   4                         0.32441                                   4.81130

      [(cash + short-term securities + receivables - short-term liabilities) /
   (operating expenses - depreciation)] * 365   \
   0                                        -47.9670
   1                                         22.0960
   2                                         -7.8927
   3                                        -26.7190
   4                                         63.1220

      retained earnings / total assets   EBIT / total assets   \
   0                           0.00000             0.112660
   1                           0.00000             0.110500
   2                           0.00000            -1.372700
   3                           0.35300             0.071484
   4                           0.26805             0.085542

      book value of equity / total liabilities   sales / total assets   \
   0                                   0.52288                 2.2558
   1                                   0.55457                 2.2507
   2                                  -0.49762                48.0050
   3                                   0.88152                 1.0442
   4                                   9.29990                 1.0984

      equity / total assets   …  (sales - cost of products sold) / sales   \
   0                0.34335   …                                 0.280410
   1                0.35674   …                                 0.037282
   2               -0.99052   …                                -0.009713
   3                0.46264   …                                 0.042332
   4                0.89802   …                                 0.089598

      (current assets - inventory - short-term liabilities) / (sales - gross profit
   - depreciation)   \
   0                                        0.260420
   1                                        0.258950
```

```
2                                            1.385900
3                                            0.121250
4                                            0.094627

   total costs /total sales   long-term liabilities / equity   \
0                 0.74418                         0.225860
1                 0.95226                         0.221070
2                 1.00730                         0.000000
3                 0.95767                         0.040020
4                 0.91040                         0.012744

   sales / inventory   sales / receivables   \
0             6.7518                6.7353
1                NaN                4.6606
2                NaN               63.7950
3             4.3271                6.4310
4             5.3233                9.4997

   (short-term liabilities *365) / sales   sales / short-term liabilities   \
0                             89.083                           4.0973
1                             79.689                           4.5803
2                             15.135                          24.1170
3                             96.845                           3.7689
4                             32.600                          11.1960

   sales / fixed assets  class
0                8.4735      0
1                5.9233      0
2              668.0900      1
3               10.3020      0
4                1.6140      0

[5 rows x 65 columns]
```

[4]:

```python
year_one.describe()
# We have all numerical columns. No categorical/character columns.
# There seems to be some Missing values present. Lets observe them
```

[4]:

```
       net profit / total assets   total liabilities / total assets   \
count                7024.000000                          7024.000000
mean                    0.034660                             0.560215
std                     4.565504                             5.350084
min                  -256.890000                           -72.162000
25%                     0.021182                             0.296678
50%                     0.075802                             0.482960
75%                     0.160268                             0.680233
max                    94.280000                           441.500000
```

```
       working capital / total assets   \
count                     7024.000000
mean                         0.119969
std                          5.275459
min                       -440.500000
25%                          0.026968
50%                          0.181275
75%                          0.362548
max                          1.000000

       current assets / short-term liabilities   \
count                              6997.000000
mean                                  2.629143
std                                  13.257356
min                                   0.000000
25%                                   1.063100
50%                                   1.502000
75%                                   2.460700
max                                1017.800000

       [(cash + short-term securities + receivables - short-term liabilities) /
(operating expenses - depreciation)] * 365   \
count                                       7.019000e+03
mean                                       -2.631672e+02
std                                         3.707460e+04
min                                        -2.722100e+06
25%                                        -4.449800e+01
50%                                        -5.373900e+00
75%                                         3.777050e+01
max                                         9.909000e+05

       retained earnings / total assets   EBIT / total assets   \
count                       7024.000000          7024.000000
mean                           0.059712             0.313876
std                            6.051113             8.353274
min                         -397.890000          -189.560000
25%                            0.000000             0.028023
50%                            0.000000             0.090109
75%                            0.146660             0.188667
max                          303.670000           453.770000

       book value of equity / total liabilities   sales / total assets   \
count                               7002.000000            7026.000000
mean                                   2.623996               5.552855
std                                   18.708327             101.995448
min                                 -141.410000               0.000005
```

```
25%                                 0.445710             1.037225
50%                                 1.015100             1.205750
75%                                 2.267675             2.132975
max                              1452.200000          3876.100000

       equity / total assets  …  (sales - cost of products sold) / sales  \
count            7024.000000  …                               7.027000e+03
mean                1.825832  …                              -1.577367e+02
std                33.836452  …                               1.322125e+04
min              -440.550000  …                              -1.108300e+06
25%                 0.300785  …                               2.031450e-02
50%                 0.492235  …                               6.338200e-02
75%                 0.675677  …                               1.376950e-01
max              1099.500000  …                               1.000000e+00

       (current assets - inventory - short-term liabilities) / (sales - gross
profit - depreciation)  \
count                                        7026.000000
mean                                            0.193243
std                                             4.344046
min                                          -315.370000
25%                                             0.056772
50%                                             0.175745
75%                                             0.351922
max                                           126.670000

       total costs /total sales  long-term liabilities / equity  \
count              7.027000e+03                     7026.000000
mean               1.587409e+02                        0.277829
std                1.322124e+04                        6.339149
min               -4.194000e-03                     -327.970000
25%                8.647650e-01                        0.000000
50%                9.388100e-01                        0.028438
75%                9.820150e-01                        0.273867
max                1.108300e+06                      119.580000

       sales / inventory  sales / receivables  \
count       6.892000e+03          7005.000000
mean        4.328830e+02            15.642228
std         2.612802e+04           261.554534
min         4.700000e-05             0.000016
25%         5.923950e+00             4.829000
50%         1.004050e+01             7.033700
75%         2.013900e+01            10.703000
max         2.137800e+06         21110.000000

       (short-term liabilities *365) / sales  \
```

```
count                                 7.027000e+03
mean                                  4.763202e+03
std                                   3.107835e+05
min                                   0.000000e+00
25%                                   4.322250e+01
50%                                   6.850900e+01
75%                                   1.063350e+02
max                                   2.501600e+07

       sales / short-term liabilities   sales / fixed assets        class
count                      6997.000000            6993.000000  7027.000000
mean                          8.126852             208.731950     0.038566
std                          19.996419            5140.708804     0.192571
min                           0.000015               0.000010     0.000000
25%                           3.425400               2.538600     0.000000
50%                           5.303200               4.637700     0.000000
75%                           8.357900               9.782200     0.000000
max                        1042.200000          294770.000000     1.000000

[8 rows x 65 columns]
```

[5]:

```
number_nas = (~np.isfinite(year_one)).sum(axis=0)
print("NAs in Total:{} %".format(round(sum(number_nas)/(year_one.
 ↪shape[1]*year_one.shape[0])*100, 2)))
for na, col in zip(number_nas, year_one.columns):
    print("{} / {} % missing values in column {}".format(na, round(na/year_one.
 ↪shape[0]*100, 2), col[:60]))
```

```
NAs in Total:1.28 %
3 / 0.04 % missing values in column net profit / total assets
3 / 0.04 % missing values in column total liabilities / total assets
3 / 0.04 % missing values in column working capital / total assets
30 / 0.43 % missing values in column current assets / short-term liabilities
8 / 0.11 % missing values in column [(cash + short-term securities + receivables
- short-term li
3 / 0.04 % missing values in column retained earnings / total assets
3 / 0.04 % missing values in column EBIT / total assets
25 / 0.36 % missing values in column book value of equity / total liabilities
1 / 0.01 % missing values in column sales / total assets
3 / 0.04 % missing values in column equity / total assets
39 / 0.56 % missing values in column (gross profit + extraordinary items +
financial expenses) /
30 / 0.43 % missing values in column gross profit / short-term liabilities
0 / 0.0 % missing values in column (gross profit + depreciation) / sales
3 / 0.04 % missing values in column (gross profit + interest) / total assets
2 / 0.03 % missing values in column (total liabilities * 365) / (gross profit +
depreciation)
25 / 0.36 % missing values in column (gross profit + depreciation) / total
```

```
liabilities
25 / 0.36 % missing values in column total assets / total liabilities
3 / 0.04 % missing values in column gross profit / total assets
0 / 0.0 % missing values in column gross profit / sales
0 / 0.0 % missing values in column (inventory * 365) / sales
1622 / 23.08 % missing values in column sales (n) / sales (n-1)
3 / 0.04 % missing values in column profit on operating activities / total
assets
0 / 0.0 % missing values in column net profit / sales
124 / 1.76 % missing values in column gross profit (in 3 years) / total assets
3 / 0.04 % missing values in column (equity - share capital) / total assets
25 / 0.36 % missing values in column (net profit + depreciation) / total
liabilities
311 / 4.43 % missing values in column profit on operating activities / financial
expenses
34 / 0.48 % missing values in column working capital / fixed assets
3 / 0.04 % missing values in column logarithm of total assets
0 / 0.0 % missing values in column (total liabilities - cash) / sales
0 / 0.0 % missing values in column (gross profit + interest) / sales
38 / 0.54 % missing values in column (current liabilities * 365) / cost of
products sold
30 / 0.43 % missing values in column operating expenses / short-term liabilities
25 / 0.36 % missing values in column operating expenses / total liabilities
3 / 0.04 % missing values in column profit on sales / total assets
3 / 0.04 % missing values in column total sales / total assets
2740 / 38.99 % missing values in column (current assets - inventories) / long-
term liabilities
3 / 0.04 % missing values in column constant capital / total assets
0 / 0.0 % missing values in column profit on sales / sales
30 / 0.43 % missing values in column (current assets - inventory - receivables)
/ short-term liab
84 / 1.2 % missing values in column total liabilities / ((profit on operating
activities + depre
0 / 0.0 % missing values in column profit on operating activities / sales
0 / 0.0 % missing values in column rotation receivables + inventory turnover in
days
0 / 0.0 % missing values in column (receivables * 365) / sales
134 / 1.91 % missing values in column net profit / inventory
31 / 0.44 % missing values in column (current assets - inventory) / short-term
liabilities
29 / 0.41 % missing values in column (inventory * 365) / cost of products sold
3 / 0.04 % missing values in column EBITDA (profit on operating activities -
depreciation) / tot
0 / 0.0 % missing values in column EBITDA (profit on operating activities -
depreciation) / sal
25 / 0.36 % missing values in column current assets / total liabilities
3 / 0.04 % missing values in column short-term liabilities / total assets
29 / 0.41 % missing values in column (short-term liabilities * 365) / cost of
```

```
products sold)
34 / 0.48 % missing values in column equity / fixed assets
34 / 0.48 % missing values in column constant capital / fixed assets
0 / 0.0 % missing values in column working capital
0 / 0.0 % missing values in column (sales - cost of products sold) / sales
1 / 0.01 % missing values in column (current assets - inventory - short-term
liabilities) / (sal
0 / 0.0 % missing values in column total costs /total sales
1 / 0.01 % missing values in column long-term liabilities / equity
135 / 1.92 % missing values in column sales / inventory
22 / 0.31 % missing values in column sales / receivables
0 / 0.0 % missing values in column (short-term liabilities *365) / sales
30 / 0.43 % missing values in column sales / short-term liabilities
34 / 0.48 % missing values in column sales / fixed assets
0 / 0.0 % missing values in column class
```

[6]:

```
correlations = year_one.corr()
fig = go.Figure(data=go.Heatmap(
                    z=correlations,
                    y=correlations.index,
                    x=correlations.index))
fig.update_layout(xaxis={'showgrid': False, 'visible': False},
                  yaxis={'showgrid': False, 'visible': False})
fig.show()

# For a quick solution to reduce the multicollinearity we should drop the␣
 ↪following columns since they have the
# strongest and most correlations to other columns:
# EBITDA / sales, (sales - cost of products sold) / sales, total costs/total␣
 ↪sales, receivables*365 / sales, rotation receivables + inventory turnover in␣
 ↪days,
# profit on operating activities/sales
```

[7]:

```
# As expected, we have an unbalanced dataset. The positive class has a small␣
 ↪share with less than 7% of all data
# Therefore, I need to rebalance the datset while training to prevent the␣
 ↪algorithm to focus the negative class since it
# leads to a larger decrease of costs
for df, file_name in zip(dfs, file_names):
    print("Class Distribution for '{}' :".format(file_name))
    print(df["class"].value_counts())
```

```
Class Distribution for '../data/raw_data/1year.arff' :
0    6756
1     271
Name: class, dtype: int64
Class Distribution for '../data/raw_data/2year.arff' :
0    9773
```

```
1     400
Name: class, dtype: int64
Class Distribution for '../data/raw_data/3year.arff' :
0    10008
1      495
Name: class, dtype: int64
Class Distribution for '../data/raw_data/4year.arff' :
0    9277
1     515
Name: class, dtype: int64
Class Distribution for '../data/raw_data/5year.arff' :
0    5500
1     410
Name: class, dtype: int64
```

[8]:

```
bankruptcy_timeseries = get_class_dist_timeseries([df["class"] for df in dfs])
fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(go.Scatter(x=bankruptcy_timeseries["Year"],␣
↪y=bankruptcy_timeseries["Share of bankrupt customers"],
                         mode='lines', name='"Share of bankrupt customers"'),␣
↪secondary_y=True)
fig.add_trace(go.Scatter(x=bankruptcy_timeseries["Year"],␣
↪y=bankruptcy_timeseries["Customers"],
                         mode='lines', name='Number of Customers'),␣
↪secondary_y=False)
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)')
```

[9]:

```
rows, cols = 18, 4
columns = year_one.columns
fig = make_subplots(rows=rows, cols=cols, subplot_titles=["{} - {}".format(i,␣
↪col[:15]) for i, col in enumerate(columns)])
for i in range(year_one.shape[1]):
    row_val = int(i/cols) + 1
    col_val = (i) % cols + 1
    fig.add_trace(go.Histogram(x=remove_outliers(year_one.iloc[:, i], 3.5),␣
↪name="{} - {}".format(i, columns[i][:15])),
                 row=row_val,col=col_val)
fig.update_layout(height=1600, width=1000)
fig.show()
# Logtransform the following columns:
# 3, 7, 16, 19, 31, 32, 33, 35, 36, 39, 42, 43, 45, 46, 49, 50, 51, 59, 60-63
# invert and root squar the following columns:
# 9, 24, 37
```

[10]:

```
go.Figure(go.Histogram(x=remove_outliers((year_one.iloc[:, 3]), 5.5))).show()
go.Figure(go.Histogram(x=remove_outliers(np.log(year_one.iloc[:, 3]), 5.5))).
↪show()
```

```
c:\users\felix\appdata\local\programs\python\python37\lib\site-
packages\pandas\core\series.py:726: RuntimeWarning:

divide by zero encountered in log
```

[11]:
```
##################### PCA #########################
tmp_pca = dfs[0].copy()
tmp_pca = tmp_pca.drop("class", axis=1)
pca = PCA(tmp_pca.shape[1])
tmp_pca = MeanReplacement().process(tmp_pca)[0]
tmp_pca = Standardizer().process(tmp_pca)[0]


_ = pca.process(tmp_pca)
variance_ratio = pca._pca_model.explained_variance_ratio_
cum_ratio = np.cumsum(variance_ratio)

fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(len(variance_ratio)), y=cum_ratio,
                         mode='lines', name='Cumulated relative variance'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
                 title="PCA - Cumulated relative Variance")
fig.update_layout(shapes=[
    dict(type= 'line', yref= 'paper', y0= 0, y1= 1, xref= 'x', x0= 10, x1= 10),
    dict(type= 'line', yref= 'paper', y0= 0, y1= 1, xref= 'x', x0= 16, x1= 16),
    dict(type= 'line', yref= 'paper', y0= 0, y1= 1, xref= 'x', x0= 20, x1= 20),
    dict(type= 'line', yref= 'paper', y0= 0, y1= 1, xref= 'x', x0= 25, x1= 25)
])
fig.show()
```

[12]:
```
##################### MODELLING ###################
cost_weight = 20
primary_measure = "Weighted Accuracy"

probs_lr = []
params = {"model": {"max_iter": 1000}, "preprocessors": [MeanReplacement(),␣
 ↪Standardizer(), PCA(0.999),
                                                          ReSampler("down")]}

dfs, file_names = load_arff_files("../data/raw_data/")

cv = CrossValidation(folds=10)
all_measures = {}
if os.path.exists("../data/cleaned_data/recorded_measures_lr.json"):
    with open("../data/cleaned_data/recorded_measures_lr.json", "r") as f:
        recorded_measures = json.load(f)
```

```python
else:
    recorded_measures = None

for i, df in enumerate(dfs):
    print("#####   {} Year   ####".format(i + 1))
    df = df.copy()
    label = df.pop("class")
    # this is kick only to kick most extreme outlier
    df = clip_outliers(df.copy(), 5.5)
    model = LogisticRegression(max_iter=params["model"]["max_iter"])
    probs = cv.run(data=df, label=label, model=model,
                   preprocessors=params["preprocessors"])
    probs_lr.append(probs)  #
    measures = get_all_measures(probs, label, 0.5)

    base_line_measures = get_all_measures(np.random.choice([0, 1],␣
↪size=len(label)), label, 0.5)
    print("F1, Baseline: {}, LogisticRegression: {}".
↪format(base_line_measures["f1"], measures["f1"]))
    print("Acc, Baseline: {}, LogisticRegression: {}".
↪format(base_line_measures["acc"], measures["acc"]))
    print("AUC, Baseline: {}, LogisticRegression: {}".
↪format(base_line_measures["auc"], measures["auc"]))
    print("Recall, Baseline: {}, LogisticRegression: {}".
↪format(base_line_measures["recall"], measures["recall"]))
    print("Precision, Baseline: {}, LogisticRegression: {}".
↪format(base_line_measures["precision"],
                                                                   ␣
↪measures["precision"]))

    threshold, costs = get_threshold_for_optim_cost(probs, label,␣
↪weight=cost_weight)

    print("Threshold: {}".format(threshold))
    print("Costs: {}".format(costs))
    weighted_accuracy = get_weighted_accuracy([1 if p > threshold else 0 for p␣
↪in probs], label, cost_weight)
    measures["Weighted Accuracy"] = weighted_accuracy
    measures["params"] = copy.deepcopy(params)
    measures["params"]["preprocessors"] = [(p.__class__.__name__, p.__dict__)␣
↪for p in params["preprocessors"]]
    measures["probs"] = probs.tolist()
    measures["label"] = label.tolist()
    print("Weighted Accuracy, Baseline: {}, LogisticRegression: {}".format(
        get_weighted_accuracy(np.random.choice([0, 1], len(label)), label,␣
↪cost_weight),
```

```
        weighted_accuracy
    ))
    print("All measures with cost optimal threshold: {}".
↪format(get_all_measures(probs, label, threshold)))
    all_measures["LR Year {}".format(i + 1)] = measures
    if recorded_measures is not None:
        if measures[primary_measure] > np.max([m["LR Year {}".format(i +␣
↪1)][primary_measure]
                                               for m in recorded_measures]):
            print("Found new best measure {} with value {}".
↪format(primary_measure, measures[primary_measure]))

if recorded_measures is None:
    recorded_measures = [all_measures]
else:
    recorded_measures.append(all_measures)
with open("../data/cleaned_data/recorded_measures_lr.json", "w") as f:
    json.dump(recorded_measures, f)

get_roc_curve(probs_lr[i], label)
```

```
#####   1 Year   ####
F1, Baseline: 0.06481965499215893, LogisticRegression: 0.1573926868044515
Acc, Baseline: 0.4908211185427636, LogisticRegression: 0.6983065319481998
AUC, Baseline: 0.47485984850967516, LogisticRegression: 0.7761312071380038
Recall, Baseline: 0.4575645756457565, LogisticRegression: 0.7306273062730627
Precision, Baseline: 0.03488045007032349, LogisticRegression:
0.08819599109131403
Threshold: 0.536
Costs: 3429
Weighted Accuracy, Baseline: 0.4963042049934297, LogisticRegression:
0.7183804204993429
All measures with cost optimal threshold: {'auc': 0.7761312071380038, 'f1':
0.16876122082585276, 'acc': 0.7364451401736161, 'recall': 0.6937269372693727,
'precision': 0.09606540623403168}
Found new best measure Weighted Accuracy with value 0.7183804204993429
#####   2 Year   ####
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-12-2babe9752394> in <module>
     25     model = LogisticRegression(max_iter=params["model"]["max_iter"])
     26     probs = cv.run(data=df, label=label, model=model,
---> 27                    preprocessors=params["preprocessors"])

     28     probs_lr.append(probs)  #
     29     measures = get_all_measures(probs, label, 0.5)
```

```
D:\CodingProjects\AccentureApplication\src\modelling.py in run(self, data,␣
↪label, model, preprocessors)
    128         predictions = np.array([np.NaN for _ in range(len(indices))])
    129         for i in range(self.folds):
--> 130             train_data, test_data = data.iloc[[k for k in range(data.
↪shape[0]) if k not in test_indices[i]], :], \
    131                                     data.iloc[test_indices[i], :]
    132             train_label, test_label = [v for j, v in enumerate(label) i␣
↪j not in test_indices[i]], \

D:\CodingProjects\AccentureApplication\src\modelling.py in <listcomp>(.0)
    128         predictions = np.array([np.NaN for _ in range(len(indices))])
    129         for i in range(self.folds):
--> 130             train_data, test_data = data.iloc[[k for k in range(data.
↪shape[0]) if k not in test_indices[i]], :], \
    131                                     data.iloc[test_indices[i], :]
    132             train_label, test_label = [v for j, v in enumerate(label) i␣
↪j not in test_indices[i]], \

KeyboardInterrupt:
```

[13]:

```
probs_xgb = []
dfs, file_names = load_arff_files("../data/raw_data/")
params = {"model": dict(val_share=0.2, n_rounds=8, lambda_=5,
                        additional_booster_params={"params": {"max_depth": 4,␣
↪"subsample": 0.5,
                                                                              ␣
↪"colsample_bytree": 0.5}},
                        verbose=True), "preprocessors": []}

cv = CrossValidation(folds=10)
all_measures = {}
if os.path.exists("../data/cleaned_data/recorded_measures_xgb.json"):
    with open("../data/cleaned_data/recorded_measures_xgb.json", "r") as f:
        recorded_measures = json.load(f)
else:
    recorded_measures = None

for i, df in enumerate(dfs):
    print("#####   {} Year   ####".format(i + 1))
    df = df.copy()
    label = df.pop("class")
    model = XGBoost(**params["model"]) #, "scale_pos_weight": (df.shape[0] - np.
↪sum(np.sum(df["class"])) / np.sum(df["class"])
    probs = cv.run(data=df, label=label, model=model,␣
↪preprocessors=params["preprocessors"])
```

```
    probs_xgb.append(probs)

    measures = get_all_measures(probs, label, 0.5)
    base_line_measures = get_all_measures(np.random.choice([0, 1],␣
↪size=len(label)), label, 0.5)

    print("F1, Baseline: {}, XGBoost: {}".format(base_line_measures["f1"],␣
↪measures["f1"]))
    print("Acc, Baseline: {}, XGBoost: {}".format(get_accuracy([1 for _ in␣
↪range(len(label))], label),
                                                  measures["acc"]))
    print("AUC, Baseline: {}, XGBoost: {}".format(base_line_measures["auc"],␣
↪measures["auc"]))
    print("Recall, Baseline: {}, XGBoost: {}".
↪format(base_line_measures["recall"], measures["recall"]))
    print("Precision, Baseline: {}, XGBoost: {}".
↪format(base_line_measures["precision"],
                                                        measures["precision"]))
    threshold, costs = get_threshold_for_optim_cost(probs, label,␣
↪weight=cost_weight)
    print("Threshold: {}".format(threshold))
    print("Costs: {}".format(costs))
    weighted_accuracy = get_weighted_accuracy([1 if p > threshold else 0 for p␣
↪in probs], label, cost_weight)
    measures["Weighted Accuracy"] = weighted_accuracy
    measures["params"] = copy.deepcopy(params)
    measures["params"]["preprocessors"] = [(p.__class__.__name__, p.__dict__)␣
↪for p in params["preprocessors"]]
    measures["probs"] = probs.tolist()
    measures["label"] = label.tolist()
    print("Weighted Accuracy, Baseline: {}, XGB: {}".format(
        get_weighted_accuracy(np.random.choice([0, 1], len(label)), label,␣
↪cost_weight),
        weighted_accuracy
    ))
    print("All measures with cost optimal threshold: {}".
↪format(get_all_measures(probs, label, threshold)))
    all_measures["XGB Year {}".format(i + 1)] = measures
    if recorded_measures is not None:
        if measures[primary_measure] > np.max([m["XGB Year {}".format(i +␣
↪1)][primary_measure]
                                                for m in recorded_measures]):
            print("Found new best measure {} with value {}".
↪format(primary_measure, measures[primary_measure]))

if recorded_measures is None:
```

```python
    recorded_measures = [all_measures]
else:
    recorded_measures.append(all_measures)
with open("../data/cleaned_data/recorded_measures_xgb.json", "w") as f:
    json.dump(recorded_measures, f)

get_roc_curve(probs_xgb[i], label)
```

```
#####   1 Year   ####
[0]     train-logloss:0.47751   eval-logloss:0.47417
[1]     train-logloss:0.34625   eval-logloss:0.35450
[2]     train-logloss:0.26195   eval-logloss:0.28129
[3]     train-logloss:0.21269   eval-logloss:0.23499
[4]     train-logloss:0.18021   eval-logloss:0.20286
[5]     train-logloss:0.14959   eval-logloss:0.18266
[6]     train-logloss:0.13082   eval-logloss:0.17082
[7]     train-logloss:0.11753   eval-logloss:0.16327
[0]     train-logloss:0.47834   eval-logloss:0.47785
[1]     train-logloss:0.34669   eval-logloss:0.35716
[2]     train-logloss:0.26194   eval-logloss:0.28508
[3]     train-logloss:0.21393   eval-logloss:0.23770
[4]     train-logloss:0.18157   eval-logloss:0.20842
[5]     train-logloss:0.15140   eval-logloss:0.18881
[6]     train-logloss:0.13066   eval-logloss:0.17716
[7]     train-logloss:0.11994   eval-logloss:0.16892
[0]     train-logloss:0.47386   eval-logloss:0.47628
[1]     train-logloss:0.34424   eval-logloss:0.36247
[2]     train-logloss:0.25968   eval-logloss:0.29566
[3]     train-logloss:0.21156   eval-logloss:0.25096
[4]     train-logloss:0.17793   eval-logloss:0.22222
[5]     train-logloss:0.14806   eval-logloss:0.20543
[6]     train-logloss:0.12898   eval-logloss:0.19646
[7]     train-logloss:0.11791   eval-logloss:0.19000
[0]     train-logloss:0.47771   eval-logloss:0.47668
[1]     train-logloss:0.34867   eval-logloss:0.35782
[2]     train-logloss:0.26356   eval-logloss:0.28531
[3]     train-logloss:0.21505   eval-logloss:0.23684
[4]     train-logloss:0.18257   eval-logloss:0.20751
[5]     train-logloss:0.15341   eval-logloss:0.18885
[6]     train-logloss:0.13389   eval-logloss:0.17937
[7]     train-logloss:0.12308   eval-logloss:0.17136
[0]     train-logloss:0.47565   eval-logloss:0.47273
[1]     train-logloss:0.34539   eval-logloss:0.35226
[2]     train-logloss:0.25956   eval-logloss:0.27842
[3]     train-logloss:0.21102   eval-logloss:0.23120
[4]     train-logloss:0.17835   eval-logloss:0.19914
[5]     train-logloss:0.14882   eval-logloss:0.17924
[6]     train-logloss:0.12897   eval-logloss:0.16827
```

```
[7]	train-logloss:0.11692	eval-logloss:0.16062
[0]	train-logloss:0.47789	eval-logloss:0.47803
[1]	train-logloss:0.34791	eval-logloss:0.35996
[2]	train-logloss:0.26478	eval-logloss:0.28755
[3]	train-logloss:0.21592	eval-logloss:0.24113
[4]	train-logloss:0.18291	eval-logloss:0.20959
[5]	train-logloss:0.15437	eval-logloss:0.19060
[6]	train-logloss:0.13410	eval-logloss:0.18097
[7]	train-logloss:0.12193	eval-logloss:0.17303
[0]	train-logloss:0.47598	eval-logloss:0.47374
[1]	train-logloss:0.34517	eval-logloss:0.35296
[2]	train-logloss:0.26049	eval-logloss:0.27969
[3]	train-logloss:0.21310	eval-logloss:0.23075
[4]	train-logloss:0.17916	eval-logloss:0.19960
[5]	train-logloss:0.15044	eval-logloss:0.18010
[6]	train-logloss:0.13040	eval-logloss:0.16883
[7]	train-logloss:0.11856	eval-logloss:0.15813
[0]	train-logloss:0.47758	eval-logloss:0.47579
[1]	train-logloss:0.34852	eval-logloss:0.35725
[2]	train-logloss:0.26349	eval-logloss:0.28577
[3]	train-logloss:0.21570	eval-logloss:0.23822
[4]	train-logloss:0.18193	eval-logloss:0.20622
[5]	train-logloss:0.15227	eval-logloss:0.18926
[6]	train-logloss:0.13278	eval-logloss:0.17976
[7]	train-logloss:0.12127	eval-logloss:0.17119
[0]	train-logloss:0.47640	eval-logloss:0.47841
[1]	train-logloss:0.34579	eval-logloss:0.35899
[2]	train-logloss:0.25960	eval-logloss:0.28615
[3]	train-logloss:0.21131	eval-logloss:0.24085
[4]	train-logloss:0.17871	eval-logloss:0.20978
[5]	train-logloss:0.14947	eval-logloss:0.19138
[6]	train-logloss:0.12951	eval-logloss:0.18150
[7]	train-logloss:0.11825	eval-logloss:0.17442
[0]	train-logloss:0.47546	eval-logloss:0.47292
[1]	train-logloss:0.34423	eval-logloss:0.35461
[2]	train-logloss:0.25815	eval-logloss:0.28212
[3]	train-logloss:0.21002	eval-logloss:0.23508
[4]	train-logloss:0.17720	eval-logloss:0.20622
[5]	train-logloss:0.14703	eval-logloss:0.18803
[6]	train-logloss:0.12806	eval-logloss:0.17760
[7]	train-logloss:0.11684	eval-logloss:0.17027
F1, Baseline: 0.07391763463569166, XGBoost: 0.4431818181818182
Acc, Baseline: 0.03856553294435748, XGBoost: 0.9721075850291732
AUC, Baseline: 0.5083765913147587, XGBoost: 0.892825893179003
Recall, Baseline: 0.5166051660516605, XGBoost: 0.2878228782287823
Precision, Baseline: 0.0398066533977822, XGBoost: 0.9629629629629629
Threshold: 0.102
Costs: 1832
```

```
Weighted Accuracy, Baseline: 0.4929369250985545, XGB: 0.8495400788436268
All measures with cost optimal threshold: {'auc': 0.892825893179003, 'f1':
0.44469026548672563, 'acc': 0.9285612636971681, 'recall': 0.7416974169741697,
'precision': 0.3175355450236967}
#####   2 Year   ####
[0]     train-logloss:0.47752   eval-logloss:0.47723
[1]     train-logloss:0.35342   eval-logloss:0.36116
[2]     train-logloss:0.27544   eval-logloss:0.29049
[3]     train-logloss:0.22536   eval-logloss:0.24404
[4]     train-logloss:0.19322   eval-logloss:0.21257
[5]     train-logloss:0.16683   eval-logloss:0.19414
[6]     train-logloss:0.14558   eval-logloss:0.18422
[7]     train-logloss:0.13402   eval-logloss:0.17631
[0]     train-logloss:0.47853   eval-logloss:0.47505
[1]     train-logloss:0.35417   eval-logloss:0.35735
[2]     train-logloss:0.27598   eval-logloss:0.28435
[3]     train-logloss:0.22911   eval-logloss:0.23769
[4]     train-logloss:0.19789   eval-logloss:0.20747
[5]     train-logloss:0.17186   eval-logloss:0.18852
[6]     train-logloss:0.15119   eval-logloss:0.17713
[7]     train-logloss:0.14050   eval-logloss:0.16829
[0]     train-logloss:0.47572   eval-logloss:0.47592
[1]     train-logloss:0.35117   eval-logloss:0.36046
[2]     train-logloss:0.27388   eval-logloss:0.29187
[3]     train-logloss:0.22429   eval-logloss:0.24658
[4]     train-logloss:0.19327   eval-logloss:0.21833
[5]     train-logloss:0.16750   eval-logloss:0.19950
[6]     train-logloss:0.14741   eval-logloss:0.18965
[7]     train-logloss:0.13770   eval-logloss:0.18211
[0]     train-logloss:0.47747   eval-logloss:0.47845
[1]     train-logloss:0.35853   eval-logloss:0.36172
[2]     train-logloss:0.28527   eval-logloss:0.29020
[3]     train-logloss:0.23805   eval-logloss:0.24541
[4]     train-logloss:0.20499   eval-logloss:0.21637
[5]     train-logloss:0.18309   eval-logloss:0.19757
[6]     train-logloss:0.15640   eval-logloss:0.18687
[7]     train-logloss:0.14543   eval-logloss:0.18013
[0]     train-logloss:0.47951   eval-logloss:0.47906
[1]     train-logloss:0.36140   eval-logloss:0.36157
[2]     train-logloss:0.28947   eval-logloss:0.28989
[3]     train-logloss:0.24204   eval-logloss:0.24614
[4]     train-logloss:0.20871   eval-logloss:0.21655
[5]     train-logloss:0.18685   eval-logloss:0.19690
[6]     train-logloss:0.16512   eval-logloss:0.18562
[7]     train-logloss:0.15432   eval-logloss:0.17705
[0]     train-logloss:0.47568   eval-logloss:0.47692
[1]     train-logloss:0.35538   eval-logloss:0.35790
[2]     train-logloss:0.27866   eval-logloss:0.28493
```

```
[3]     train-logloss:0.23212   eval-logloss:0.24240
[4]     train-logloss:0.19805   eval-logloss:0.21472
[5]     train-logloss:0.17615   eval-logloss:0.19581
[6]     train-logloss:0.15520   eval-logloss:0.18283
[7]     train-logloss:0.14301   eval-logloss:0.17572
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-afe5c48075c3> in <module>
     19     label = df.pop("class")
     20     model = XGBoost(**params["model"]) #, "scale_pos_weight": (df.
 ↪shape[0] - np.sum(np.sum(df["class"])) / np.sum(df["class"])
---> 21     probs = cv.run(data=df, label=label, model=model,␣
 ↪preprocessors=params["preprocessors"])
     22     probs_xgb.append(probs)
     23

D:\CodingProjects\AccentureApplication\src\modelling.py in run(self, data,␣
 ↪label, model, preprocessors)
    128         predictions = np.array([np.NaN for _ in range(len(indices))])
    129         for i in range(self.folds):
--> 130             train_data, test_data = data.iloc[[k for k in range(data.
 ↪shape[0]) if k not in test_indices[i]], :], \
    131                                     data.iloc[test_indices[i], :]
    132             train_label, test_label = [v for j, v in enumerate(label) i␣
 ↪j not in test_indices[i]], \

D:\CodingProjects\AccentureApplication\src\modelling.py in <listcomp>(.0)
    128         predictions = np.array([np.NaN for _ in range(len(indices))])
    129         for i in range(self.folds):
--> 130             train_data, test_data = data.iloc[[k for k in range(data.
 ↪shape[0]) if k not in test_indices[i]], :], \
    131                                     data.iloc[test_indices[i], :]
    132             train_label, test_label = [v for j, v in enumerate(label) i␣
 ↪j not in test_indices[i]], \

KeyboardInterrupt:
```

[15]:

```
############## EVALUATAION ######################

params_xgb = get_params_for_best_measure_overall("auc", "../data/cleaned_data/
 ↪recorded_measures_xgb.json")
params_lr = get_params_for_best_measure_overall("auc", "../data/cleaned_data/
 ↪recorded_measures_lr.json")
```

```
model_first_year_xgb = get_params_for_best_model("auc", 1, "../data/
↪cleaned_data/recorded_measures_xgb.json")
model_first_year_lr = get_params_for_best_model("auc", 1, "../data/cleaned_data/
↪recorded_measures_lr.json")

measure = "auc"
measures = {"xgb": [params_xgb[key][measure] for key in params_xgb.keys()],
            "lr": [params_lr[key][measure] for key in params_lr.keys()],
           "x": [key[4:] for key in params_xgb.keys()]}

fig = go.Figure()
fig.add_trace(go.Scatter(x=measures["x"], y=measures["xgb"],
                         mode='lines', name='XGBoost Performance'))
fig.add_trace(go.Scatter(x=measures["x"], y=measures["lr"],
                         mode='lines', name='Logistic Regression Performance'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
                  title="{} - Best Measure for all run".format(measure.upper()))
fig.show()

measures_one_model = {"xgb": [model_first_year_xgb[key][measure] for key in␣
↪model_first_year_xgb.keys()],
            "lr": [model_first_year_lr[key][measure] for key in␣
↪model_first_year_lr.keys()],
           "x": [key[4:] for key in model_first_year_xgb.keys()]}

fig = go.Figure()
fig.add_trace(go.Scatter(x=measures_one_model["x"], y=measures_one_model["xgb"],
                         mode='lines', name='XGBoost Performance'))
fig.add_trace(go.Scatter(x=measures_one_model["x"], y=measures_one_model["lr"],
                         mode='lines', name='Logistic Regression Performance'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
                  title="{} - Measure for best Model in Year One".format(measure.
↪upper()))
fig.show()
```

[16]:
```
with open("../data/cleaned_data/recorded_measures_xgb.json", "r") as f:
    measures_xgb = json.load(f)
with open("../data/cleaned_data/recorded_measures_lr.json", "r") as f:
    measures_lr = json.load(f)
xgb_importance = {}
lr_importance = {}
for run in measures_xgb:
    for year in run.keys():
        if "importance" in run[year].keys():
            xgb_importance[year] = run[year]["importance"]

for run in measures_lr:
```

```python
    for year in run.keys():
        if "importance" in run[year].keys():
            lr_importance[year] = run[year]["importance"]

lr_importance = [val[0] for val in lr_importance.values()]
viz_data = pd.DataFrame(xgb_importance)
name_mapping = {"".join(c for c in col if c.isalnum()):col for col in dfs[0].columns}
viz_data = viz_data.rename(name_mapping, axis=0)

year_one_importance = viz_data.loc[np.isfinite(viz_data["XGB Year 1"]), "XGB Year 1"].sort_values(ascending=False)
year_five_importance = viz_data.loc[np.isfinite(viz_data["XGB Year 5"]), "XGB Year 5"].sort_values(ascending=False)

fig = go.Figure()
fig.add_trace(go.Bar(x=year_one_importance.index, y=year_one_importance,
                     name='XGB Feature IMportance (Gain)'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)', title="Year 1 Feature Importances")
fig.show()

fig = go.Figure()
fig.add_trace(go.Bar(x=year_five_importance.index, y=year_five_importance,
                     name='XGB Feature IMportance (Gain)'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)', title="Year 5 Feature Importances")
fig.show()


fig = go.Figure()
fig.add_trace(go.Bar(x=year_one_importance.index, y=year_one_importance,
                     name='XGB Feature IMportance (Gain) Year 1'))
fig.add_trace(go.Bar(x=year_five_importance.index, y=year_five_importance,
                     name='XGB Feature IMportance (Gain) Year 5'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)', title="Feature Importances",
                 legend=dict(
    yanchor="top",
    y=0.99,
    xanchor="left",
    x=0.60
))
fig.show()
```

```
intersect_values = np.intersect1d(year_five_importance.index,␣
↪year_one_importance.index)
len(intersect_values)
```

[16]: 22

[20]:
```
######### Feature Selection by Correlation ########
pd.set_option('chained_assignment', None)
cost_weight = 13
features_path = "../data/cleaned_data/features.json"
dfs, file_names = load_arff_files("../data/raw_data/")
dump_file = False

if os.path.exists(features_path):
    with open(features_path, "r") as f:
        features = json.load(f)
    if "corr_selector_max" not in features:
        features["corr_selector_max"] = {}
    if "corr_selector" not in features:
        features["corr_selector"] = {}
    if "greedy_selector" not in features:
        features["greedy_selector"] = {}
else:
    features = {"corr_selector": {}, "greedy_selector": {}, "corr_selector_max":
↪ {}}

for i, df in enumerate(dfs):
    corr_features, corr_measure = main_corr(data=df.copy(),␣
↪pred_function=pred_function(cost_weight),
                                            early_stopping=5, direction="max")

    features["corr_selector_max"]["Year{}".format(i + 1)] = {"features":␣
↪corr_features, "auc": corr_measure}
    if dump_file:
        with open(features_path, "w") as f:
            json.dump(features, f)
    print(corr_features)

for i, df in enumerate(dfs):
    corr_features, corr_measure = main_corr(data=df.copy(),␣
↪pred_function=pred_function(cost_weight),
                                            early_stopping=10, direction="min")

    features["corr_selector"]["Year{}".format(i + 1)] = {"features":␣
↪corr_features, "auc": corr_measure}
    if dump_file:
```

```
        with open(features_path, "w") as f:
            json.dump(features, f)
    print(corr_features)
```

```
Run: 0

D:\CodingProjects\AccentureApplication\src\feature_selecting.py:66:
RuntimeWarning:

invalid value encountered in double_scalars


Chosen column: (gross profit + depreciation) / sales  - with Measure value:
0.667866638701911
[0.667866638701911]
Run: 1


KeyboardInterrupt
```

[21]:

```
######## Greedy Feature Selection ###### CONSUMES A LOT OF TIME!
pd.set_option('chained_assignment', None)
cost_weight = 13
features_path = "../data/cleaned_data/features.json"
dfs, file_names = load_arff_files("../data/raw_data/")

if os.path.exists(features_path):
    with open(features_path, "r") as f:
        features = json.load(f)
    if "corr_selector_max" not in features:
        features["corr_selector_max"] = {}
    if "corr_selector" not in features:
        features["corr_selector"] = {}
    if "greedy_selector" not in features:
        features["greedy_selector"] = {}
else:
    features = {"corr_selector": {}, "greedy_selector": {}, "corr_selector_max":
 ↪ {}}

for i, df in enumerate(dfs):
    if "Year{}".format(i + 1) in features["greedy_selector"]:
        continue
    features_greedy, greedy_measure = main_greedy(data=df.copy(),␣
 ↪pred_function=pred_function(cost_weight))
    features["greedy_selector"]["Year{}".format(i + 1)] = {"features":␣
 ↪features_greedy, "auc": greedy_measure}
    if dump_file:
        with open(features_path, "w") as f:
```

```
        json.dump(features, f)
print(features_greedy)
```

Run: 0

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-17cab3be30b5> in <module>
     20     if "Year{}".format(i + 1) in features["greedy_selector"]:
     21         continue
---> 22     features_greedy, greedy_measure = main_greedy(data=df.copy(),␣
 ↪pred_function=pred_function(cost_weight))
     23     features["greedy_selector"]["Year{}".format(i + 1)] = {"features":␣
 ↪features_greedy, "auc": greedy_measure}
     24     if dump_file:

D:\CodingProjects\AccentureApplication\src\__main_fs__.py in main_greedy(data,␣
 ↪pred_function, early_stopping, tolerance, verbose, max_processes)
     15     features, measures = gfs.run_selection(data=data, label=label,␣
 ↪prediction_function=pred_function,
     16                                               ␣
 ↪early_stopping_iter=early_stopping,
---> 17                                               tolerance=tolerance,␣
 ↪verbose=verbose, max_processes=max_processes)
     18
     19     return features, measures

D:\CodingProjects\AccentureApplication\src\feature_selecting.py in␣
 ↪run_selection(self, data, label, prediction_function, early_stopping_iter,␣
 ↪tolerance, verbose, max_processes)
    115             if max_processes > 1:
    116                 measures = self.calculate_parallel(chosen_columns,␣
 ↪remaining_columns, prediction_function, data, label,
--> 117                                                    max_processes)
    118             else:
    119                 measures = self.calculate_sequential(chosen_columns,␣
 ↪remaining_columns, prediction_function, data, label)

D:\CodingProjects\AccentureApplication\src\feature_selecting.py in␣
 ↪calculate_parallel(chosen_columns, remaining_columns, prediction_function,␣
 ↪data, label, max_processes)
    152         for p in processes:
    153             while True:
--> 154                 if semaphore.acquire(timeout=1):
    155                     break
    156             p.start()
```

```
KeyboardInterrupt:
```

[26]:
```
####### Evaluation Feature Selection ####
with open("../data/cleaned_data/features.json","r") as f:
    features = json.load(f)

measures_year_one = {sel: features[sel]["Year1"]["auc"] for sel in features.
 ↪keys()}

year_one = {sel: features[sel]["Year1"] for sel in features.keys()}

for i in range(1, 6):
    year = "Year{}".format(i)
    print(year)
    for sel in features.keys():
        if year in features[sel]:
            print("Selector : {}, Number Features: {}, Measure: {}".
                  format(sel, len(features[sel]["Year{}".
 ↪format(i)]["features"]),
                        features[sel]["Year{}".format(i)]["auc"][-1]))

print("Union of features between greedy and corr sel: {}".format(
    len(np.intersect1d(year_one["corr_selector"]["features"],␣
 ↪year_one["greedy_selector"]["features"])))
)
print("Union of features between greedy and corr max sel: {}".format(
    len(np.intersect1d(year_one["corr_selector_max"]["features"],␣
 ↪year_one["greedy_selector"]["features"])))
)
fig = go.Figure()
x = list(range(70))
fig.add_trace(go.Scatter(x=x, y=measures_year_one["corr_selector"],
                         mode='lines', name='Correlation Selector Min␣
 ↪Correlation'))
fig.add_trace(go.Scatter(x=x, y=measures_year_one["corr_selector_max"],
                         mode='lines', name='Correlation Selector Max␣
 ↪Correlation'))
fig.add_trace(go.Scatter(x=x, y=measures_year_one["greedy_selector"],
                         mode='lines', name='Greedy Selector'))
fig.update_layout(paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
                 title=measure.upper(), legend=dict(
    yanchor="top",
    y=0.29,
    xanchor="left",
    x=0.60
))
```

```
fig.show()
```

```
Year1
Selector : corr_selector, Number Features: 18, Measure: 0.8796136384987296
Selector : greedy_selector, Number Features: 16, Measure: 0.9246464533917096
Selector : corr_selector_max, Number Features: 10, Measure: 0.7184320511055909
Year2
Selector : corr_selector, Number Features: 42, Measure: 0.8652532487465465
Selector : corr_selector_max, Number Features: 6, Measure: 0.7331988642177427
Year3
Selector : corr_selector, Number Features: 12, Measure: 0.8376566423628773
Selector : corr_selector_max, Number Features: 23, Measure: 0.7406769533867856
Year4
Selector : corr_selector, Number Features: 29, Measure: 0.8725941910832826
Selector : corr_selector_max, Number Features: 6, Measure: 0.6934307730466096
Year5
Selector : corr_selector, Number Features: 17, Measure: 0.8661359201773835
Selector : corr_selector_max, Number Features: 13, Measure: 0.8439465631929047
Union of features between greedy and corr sel: 4
Union of features between greedy and corr max sel: 4
```

[ ]:

```
```