Faculty of Computer Science
Data Science Master Program

Ordered Sets for Data Analysis

Moscow 2024

Big HW: Neural FCA

Nasykhova Anastasiia

DATASET

stroke	
0	4700
1	209

→

stroke	
0	500
1	209

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
age	1.000000	0.303512	0.294454	0.316009	0.321822	0.540588
hypertension	0.303512	1.000000	0.123580	0.257510	0.131474	0.294552
heart_disease	0.294454	0.123580	1.000000	0.223900	0.072813	0.228139
avg_glucose_level	0.316009	0.257510	0.223900	1.000000	0.288076	0.263868
bmi	0.321822	0.131474	0.072813	0.288076	1.000000	0.113405
stroke	0.540588	0.294552	0.228139	0.263868	0.113405	1.000000

Gender: "Male", "Female" or "Other"

Age: The age of the patient.

Hypertension: Whether the patient has hypertension (1 for yes, 0 for no).

Heart_disease: Whether the patient has a history of heart disease (1 for yes, 0 for no).

Ever_married: Whether the patient has been married (1 for yes, 0 for no).

Work_type: The type of work the patient does (e.g., private, self-employed, government job, children, never work).

Residence_type: Whether the patient resides in an urban or rural area.

Avg_glucose_level: The patient's average glucose level in blood.

Bmi: Body Mass Index.

Smoking_status: The smoking habit of the patient (e.g., never smoked, formerly smoked, currently smoking or unknown if the information is unavailable for this patient).

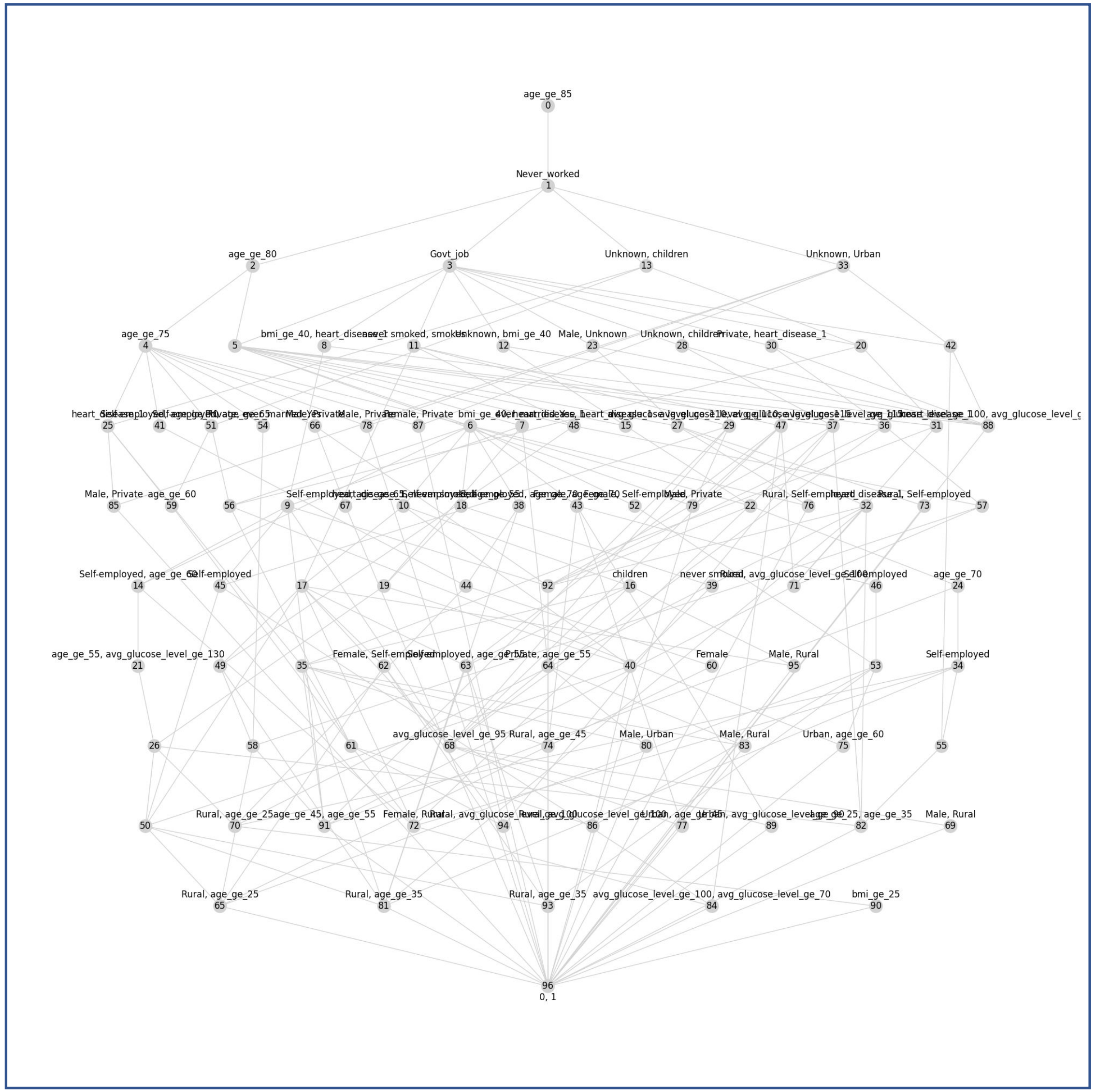
Stroke: The target variable indicating whether the patient had a stroke (1 for yes, 0 for no).

1st STRATEGY. BASELINE

```
df_bin_strategy1 = ordinal(
    df_balanced,
    ["age", "avg_glucose_level", "bmi"],
    [
        [25, 35, 45, 55, 60, 65, 70, 75, 80, 85],
        [70, 90, 95, 100, 110, 115, 120, 125, 130, 135],
        [18, 20, 25, 30, 35, 40],
    ],
    style=">=",
)

df_bin_strategy1 = nominal(
    df_bin_strategy1, ["gender", "work_type", "Residence_type", "smoking_status"]
)
df_bin_strategy1 = dichotomic(
    df_bin_strategy1, ["hypertension", "heart_disease", "ever_married"]
)
```

	classifier	accuracy	precision	recall	f1_score
4	Random Forest	0.746479	0.641026	0.531915	0.739809
5	CatBoost	0.739437	0.625000	0.531915	0.733583
6	XGBoost	0.732394	0.609756	0.531915	0.727367
2	Logistic Regression	0.732394	0.615385	0.510638	0.725354
3	Decision Tree	0.718310	0.581395	0.531915	0.714954
1	Naive Bayes	0.704225	0.528090	1.000000	0.707928
0	kNN	0.704225	0.553191	0.553191	0.704225



2nd STRATEGY. BASELINE

```
df_bin_strategy2 = ordinal(
    df_balanced,
    ["age", "avg_glucose_level", "bmi"],
    [
        [25, 35, 45, 55, 60, 65, 70, 75, 80, 85],
        [70, 90, 95, 100, 110, 115, 120, 125, 130, 135],
        [18, 20, 25, 30, 35, 40],
    ],
    style=">=<=",
)

df_bin_strategy2 = nominal(
    df_bin_strategy2, ["gender", "work_type", "Residence_type", "smoking_status"]
)
df_bin_strategy2 = dichotomic(
    df_bin_strategy2, ["hypertension", "heart_disease", "ever_married"]
)
```

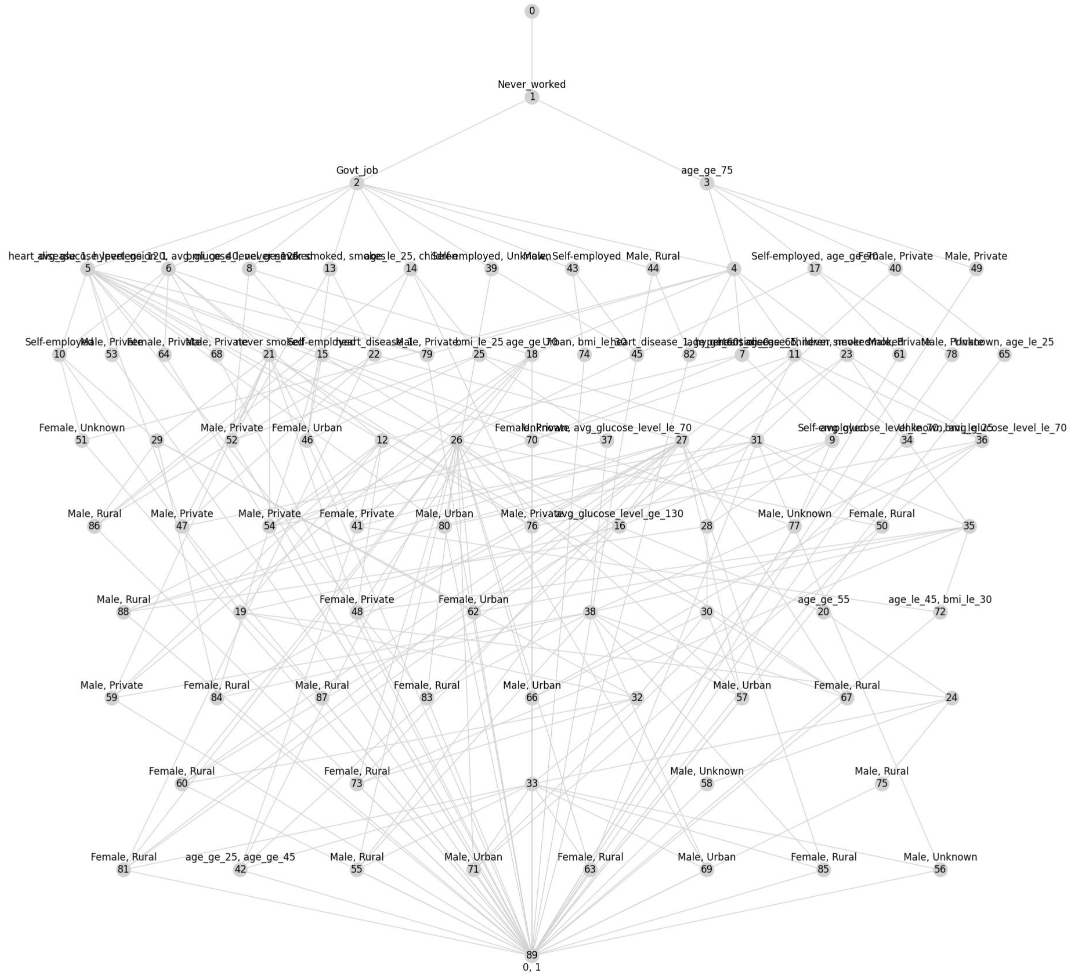
	classifier	accuracy	precision	recall	f1_score
5	CatBoost	0.739437	0.613636	0.574468	0.737167
2	Logistic Regression	0.739437	0.625000	0.531915	0.733583
1	Naive Bayes	0.718310	0.542169	0.957447	0.724388
0	kNN	0.718310	0.574468	0.574468	0.718310
3	Decision Tree	0.718310	0.581395	0.531915	0.714954
4	Random Forest	0.718310	0.589744	0.489362	0.710899
6	XGBoost	0.718310	0.589744	0.489362	0.710899

3rd STRATEGY. BASELINE

```
df_bin_strategy3 = ordinal(
    df_balanced,
    ["age", "avg_glucose_level", "bmi"],
    [
        [25, 45, 55, 60, 65, 70, 75],
        [70, 90, 95, 120, 125, 130],
        [25, 30, 35, 40],
    ],
    style=">=<=",
)

df_bin_strategy3 = nominal(
    df_bin_strategy3, ["gender", "work_type", "Residence_type", "smoking_status"]
)
df_bin_strategy3 = dichotomic(
    df_bin_strategy3, ["hypertension", "heart_disease", "ever_married"]
)
```

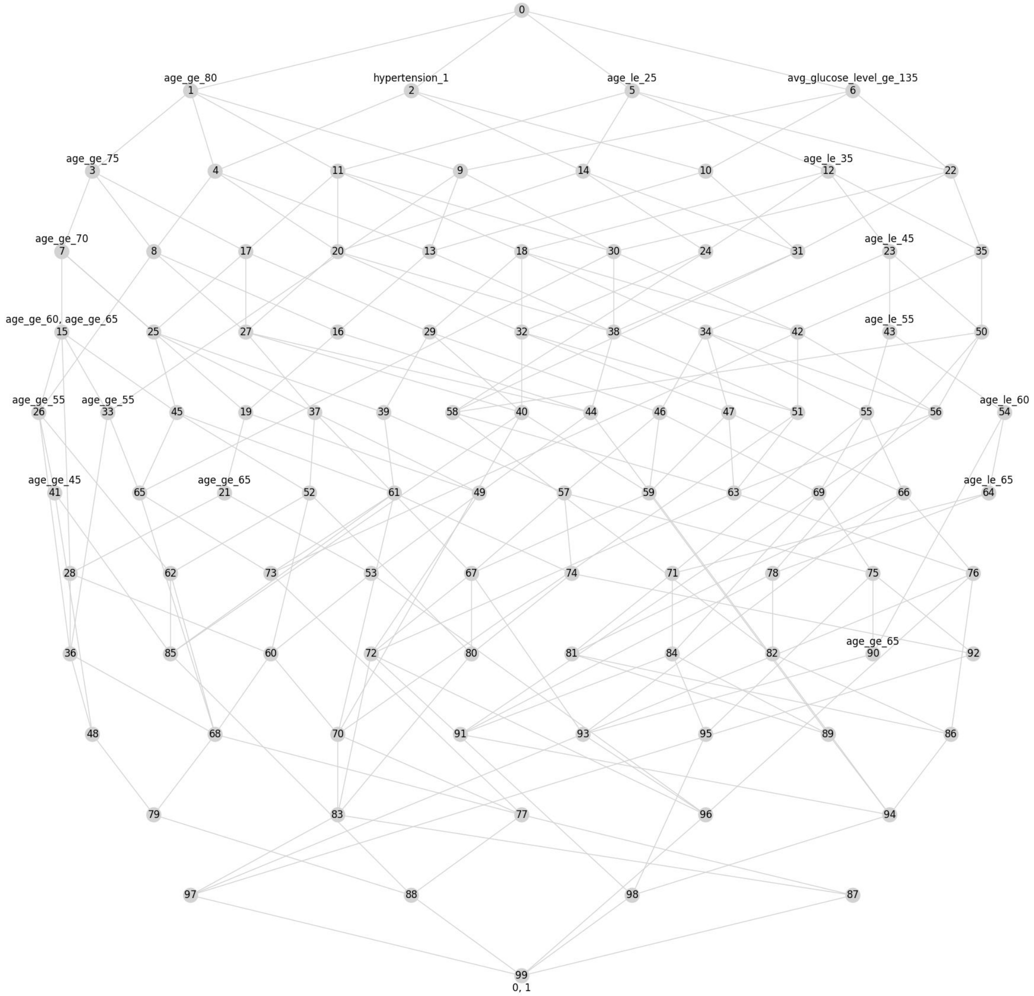
	classifier	accuracy	precision	recall	f1_score
1	Naive Bayes	0.767606	0.592105	0.957447	0.774072
4	Random Forest	0.746479	0.634146	0.553191	0.741717
5	CatBoost	0.739437	0.613636	0.574468	0.737167
2	Logistic Regression	0.739437	0.625000	0.531915	0.733583
6	XGBoost	0.732394	0.609756	0.531915	0.727367
3	Decision Tree	0.725352	0.611111	0.468085	0.714668
0	kNN	0.704225	0.558140	0.510638	0.700702



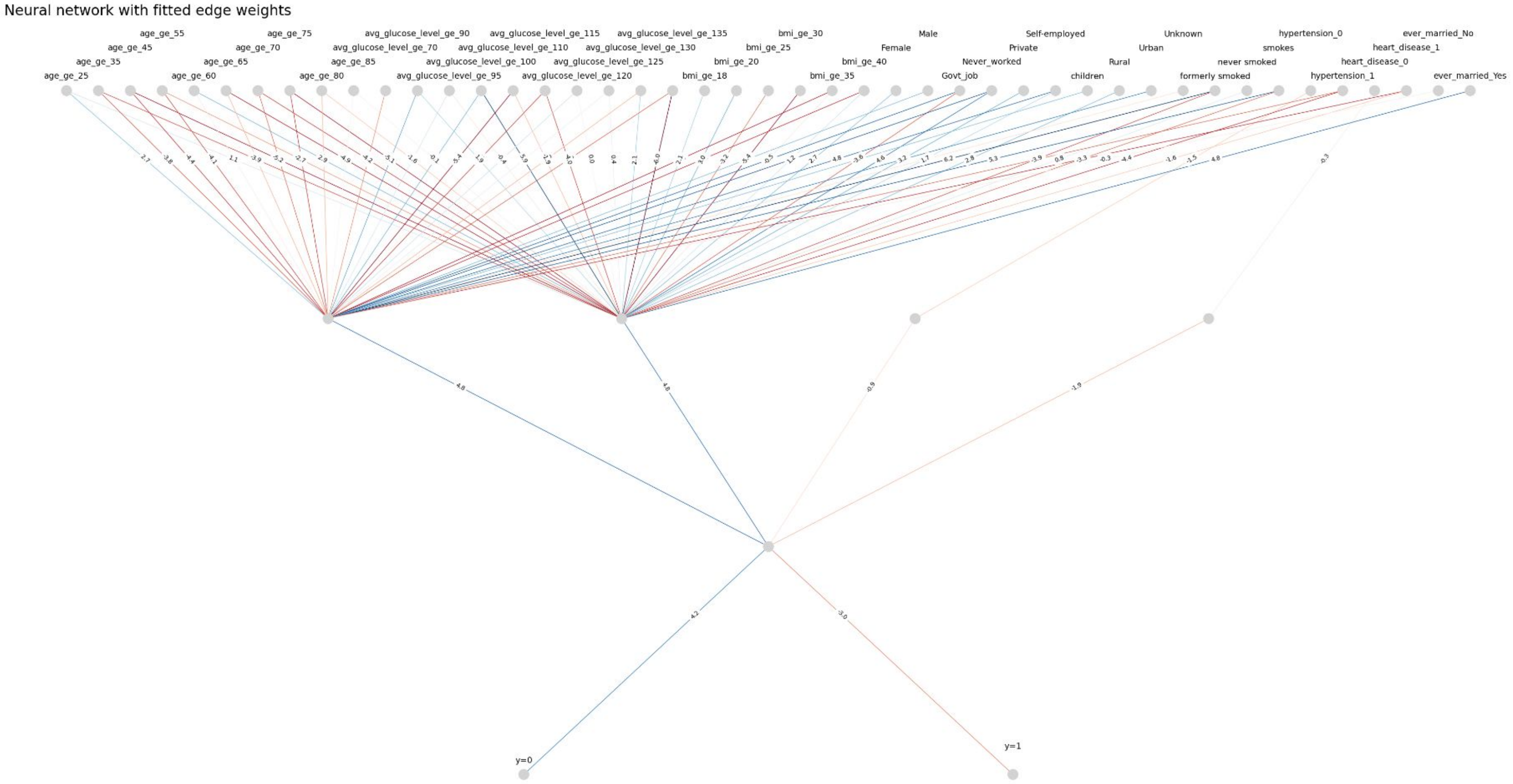
4th STRATEGY. BASELINE

```
feature with column_index 1 = age_le_25
feature with column_index 3 = age_le_35
feature with column_index 4 = age_ge_45
feature with column_index 5 = age_le_45
feature with column_index 6 = age_ge_55
feature with column_index 7 = age_le_55
feature with column_index 8 = age_ge_60
feature with column_index 9 = age_le_60
feature with column_index 10 = age_ge_65
feature with column_index 11 = age_le_65
feature with column_index 12 = age_ge_70
feature with column_index 14 = age_ge_75
feature with column_index 16 = age_ge_80
feature with column_index 38 = avg_glucose_level_ge_135
feature with column_index 66 = hypertension_1
```

	classifier	accuracy	precision	recall	f1_score
1	Naive Bayes	0.732394	0.578947	0.702128	0.737827
4	Random Forest	0.739437	0.613636	0.574468	0.737167
5	CatBoost	0.739437	0.613636	0.574468	0.737167
2	Logistic Regression	0.732394	0.604651	0.553191	0.729206
0	kNN	0.725352	0.595238	0.531915	0.721158
6	XGBoost	0.718310	0.581395	0.531915	0.714954
3	Decision Tree	0.725352	0.611111	0.468085	0.714668

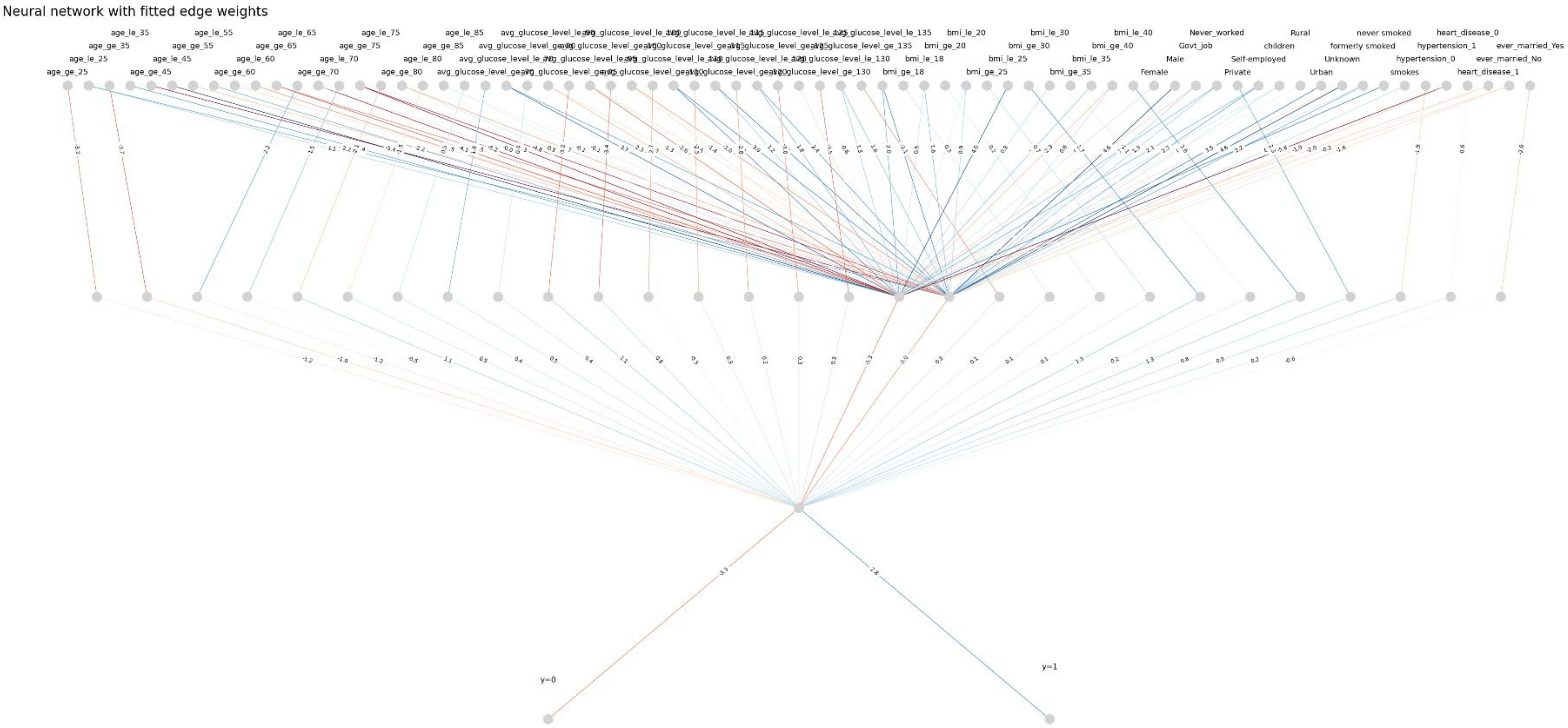


1st STRATEGY. FCA



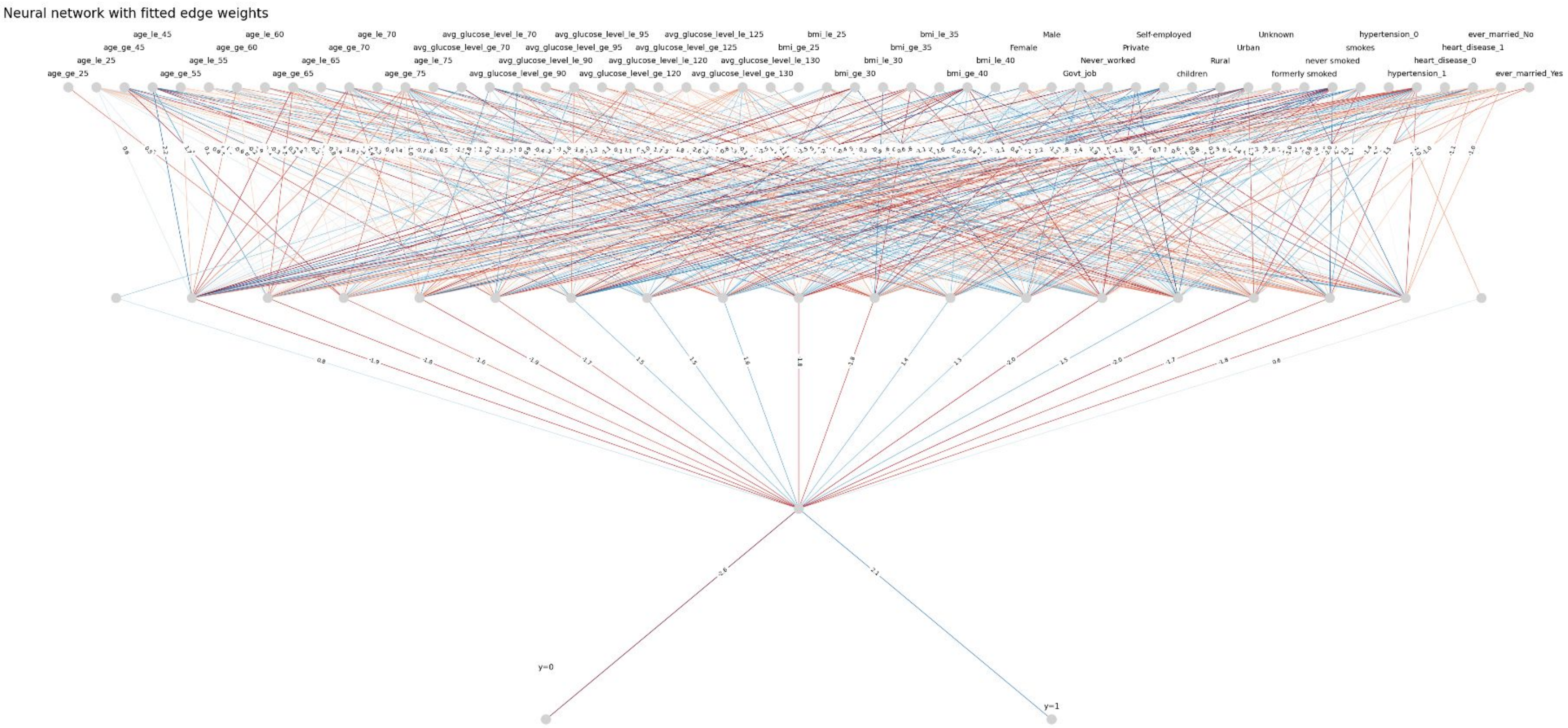
f1 = 0.62, recall = 0.6

2nd STRATEGY. FCA



f1 = 0.61, recall = 0.62

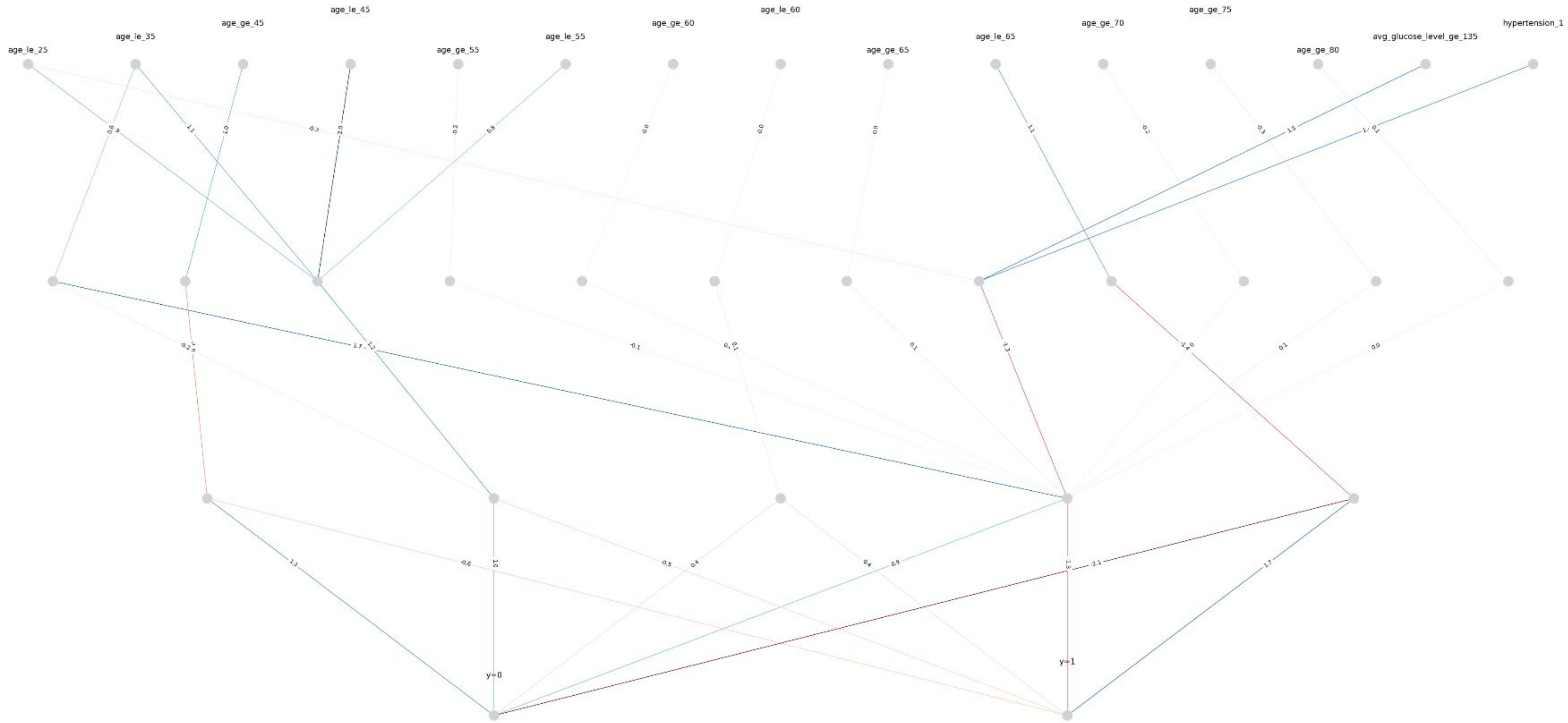
3rd STRATEGY. FCA



f1 = 0.62, recall = 0.62

4th STRATEGY. FCA

Neural network with fitted edge weights



f1 = 0.57, recall = 0.58

CHALLENGES

```
File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\fcapy\poset\poset.py:169, in POSet._direct_super_elements_nocache(self, element_index)
    167 def _direct_super_elements_nocache(self, element_index: int):
    168     """Return a set of indexes of closest elements of POSet bigger than ``element_index`` (w/out using cache)"""
--> 169     superelement_idxs = self.super_elements(element_index)
    170     for el_idx in list(superelement_idxs):
    171         if el_idx in superelement_idxs:

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\fcapy\poset\poset.py:138, in POSet._super_elements_cache(self, element_index)
    136 res = self._cache_superelements.get(element_index)
    137 if res is None:
--> 138     res = self._super_elements_nocache(element_index)
    139     self._cache_superelements[element_index] = frozenset(res)
    140 return set(res)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\fcapy\poset\poset.py:131, in POSet._super_elements_nocache(self, element_index)
    129 def _super_elements_nocache(self, element_index: int):
    130     """Return a set of indexes of elements of POSet bigger than element #``element_index`` (without using cache)"""
--> 131     sup_indexes = {i for i in range(len(self)) if self.leq_elements(element_index, i) and i != element_index}
    132     return sup_indexes

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\fcapy\poset\poset.py:263, in POSet._leq_elements_cache(self, a_index, b_index)
    261     res = b_index in self._cache_superelements[a_index]
    262 else:
--> 263     res = self._leq_elements_nocache(a_index, b_index)
    264     self._cache_leq[key] = res
    265 return res

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\fcapy\poset\poset.py:251, in POSet._leq_elements_nocache(self, a_index, b_index)
    249 def _leq_elements_nocache(self, a_index: int, b_index: int):
    250     """Compare two elements of POSet by their indexes (without using cache)"""
--> 251     return self._leq_func(self._elements[a_index], self._elements[b_index])

TypeError: 'NoneType' object is not callable
```

```
df_bin.index
```

```
Index([   0,    2,    3,    4,    5,    6,    7,    9,   10,   11,
       ...
       5098, 5100, 5101, 5102, 5103, 5104, 5106, 5107, 5108, 5109],
      dtype='int64', length=4909)
```

Spliting the data to train and test

```
df_bin = df_bin.set_index(np.arange(4909).astype(str))
```

```
df_bin.index
```

```
Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
       ...
       '4899', '4900', '4901', '4902', '4903', '4904', '4905', '4906', '4907',
       '4908'],
      dtype='object', length=4909)
```

```
print("Recall score:", recall_score(y_test.values.astype("int"), y_pred[1]))
print("Precision score:", precision_score(y_test.values.astype("int"), y_pred[1]))
print("F1      score:", f1_score(y_test.values.astype("int"), y_pred[1]))
print("Accuracy score:", accuracy_score(y_test.values.astype("int"), y_pred[1]))
```

```
Recall score: 0.0
Precision score: 0.0
F1      score: 0.0
Accuracy score: 0.5588235294117647
```

CONCLUSION

- The number of epochs depends on the dataset, the metric, and the method of adding nonlinearities (on average, the ideal value is 5000).
- The best metric is **recall** (it is suitable for most binarization strategies).
- All other hyperparameters (nnl function, method for concept selecting) need to be selected based on a validation sample (it is difficult to identify a common pattern for different experiments).

FCA models work at the level, and in some cases even better than standard machine learning algorithms, while significantly increasing the interpretability of the model.

THANK YOU FOR YOUR ATTENTION!