Задача 3. Оцінювання дефолту клієнта за допомогою скорингової моделі на основі мереж Байєса та дерев рішень

1. Для набору навчальних даних з файлу Data_Base_I.xls побудувати початкову структуру мережі Байєса для задачі кредитування. Розбити вхідну вибірку на навчальну(90%) та тестову (для перевірки якості моделі використати 10% вхідної вибірки).
2. Виконати навчання структури і параметрів мережі Байєса.
3. Сформувати висновок – визначити ймовірність події – дефолту клієнта.
4. Виконати перевірку якості моделі за допомогою тестової вибірки.
5. Обчислити загальну похибку моделі та похибки класифікації.
6. Побудувати скорингову моделі у вигляді дерева рішень. В якості значення порогу при класифікації розгляньте випадки 95%, 90%, 85 та 80%.
7. Обчислити загальну похибку моделі (CA – common accuracy).
8. Обчислити похибки класифікації –1-го, 2-го роду та загальну.
9. Оформити протокол у вигляді порівняння обох методів та зробити висновки.

Частина 1 – Мережі Байеса

Паралельно із програмою GeNIe, ми провели моделювання мережі Байеса за допомогою пакету pgmpy(Probabilistic Graphical Models) для Python. Звичайно, використовуючи вже дискретизовані дані.

Навчання мережі Байеса поділяється на два етапи — спочатку виконується навчання структури, потім навчання умовних ймовірностей вершин.

Навчання структури. Із деяких міркувань, нам відомо що змінні Age та Gender мають бути незалежними. Отже, заборонимо ребра, що вказують на ці змінні(black_list).

Також логічно, що змінна Age впливає на змінну Maritial_status(молоді люди частіше SINGLE, більш дорослі — MARRIED), змінна Maritial_status впливає на змінну Children(у неодружених людей як правило немає дітей, у одружених зазвичай 1-3 дітей). Зафіксуємо ці ребра і почнемо будувати структуру із них.

Обраний метод пошуку — Hill Climb Search (стохастичний пошук, заснований на ідеї максимізації функції скорингу(оцінки якості моделі)).

Теперь строим саму сеть Байеса

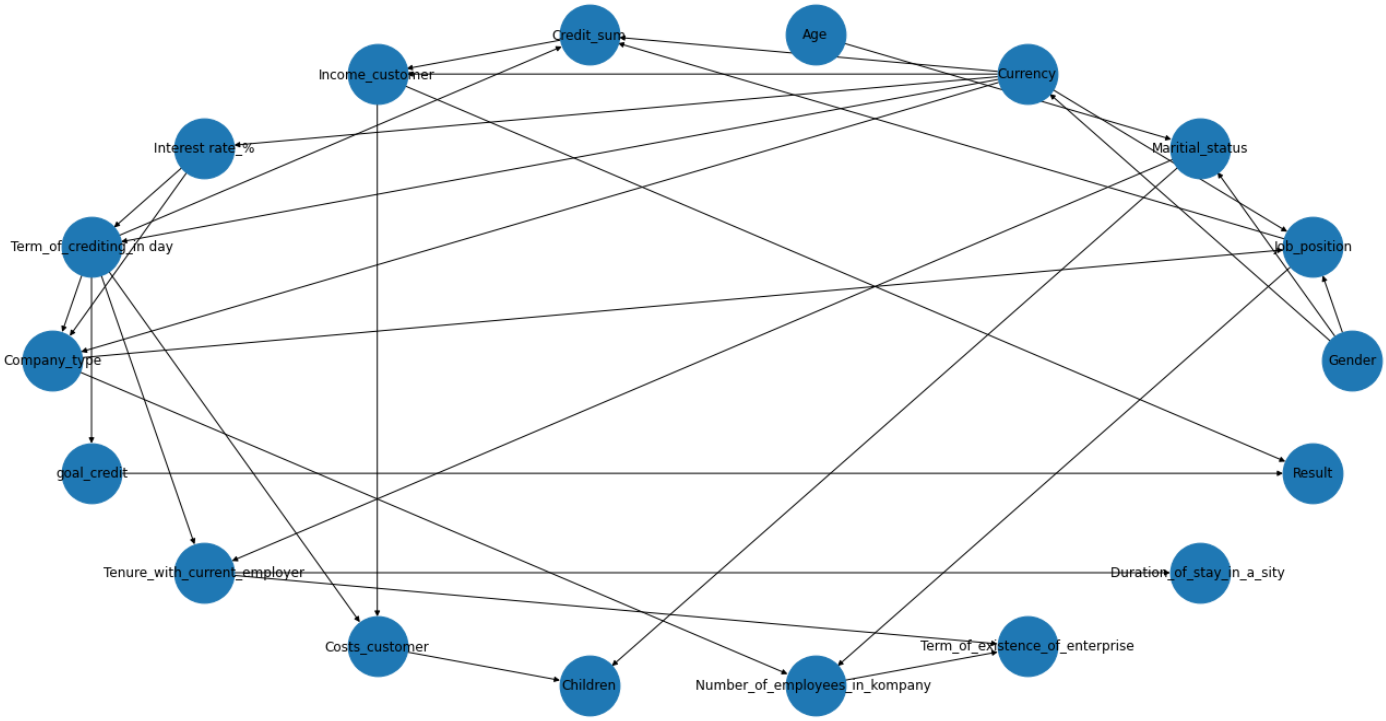
```
#Вводим список обязательных и запрещенных ребер.
black_list_edges=[(i,'Gender')for i in X_train.columns]\
                        +[(i,'Age') for i in X_train.columns]
black_list_edges.remove(('Gender','Gender'))
black_list_edges.remove(('Age','Age'))
fixed_edges_list=[('Age','Maritial_status'), ('Maritial_status','Children')]
```

```
from pgmpy.models import BayesianModel
from pgmpy.estimators import HillClimbSearch, MaximumLikelihoodEstimator,BayesianEstimator, PC
from pgmpy.estimators import BicScore, BDeuScore, K2Score

#построение структуры(направленный ациклический граф)
es=HillClimbSearch(X_train, scoring_method=BicScore(X_train))
best_model=es.estimate(black_list=black_list_edges,fixed_edges=fixed_edges_list,epsilon=1e-6)
```

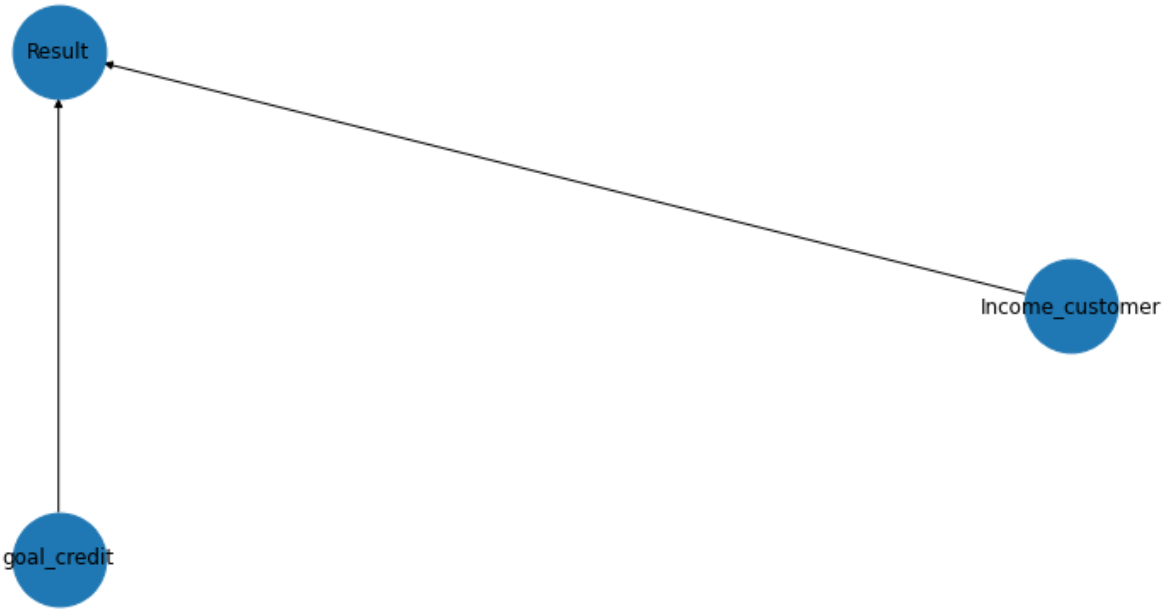
```
  0%|          | 31/1000000 [00:19<170:24:12,  1.63it/s]
```

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.MultiDiGraph()
G.add_edges_from(best_model.edges())

plt.figure(figsize=(20,10))
#nx.draw_networkx(G,node_size=3000)
nx.draw_circular(G,node_size=3000, with_labels=True)
```

Відобразимо отриманий граф залежностей(структуру мережі)



Бачимо що на Result впливають лише дві змінні. Отже, граф можна спростити:



Навчання параметрів. Розглянемо два методи навчання параметрів — Maximum Likelihood та Bayesian Search. Порівняємо їх оцінки ймовірностей:

```
#Maximum Likelihood
edges=[i for i in best_model.edges() if (i[1]=='Result')]
model_mle=BayesianModel(edges)
model_mle.fit(data=X_train,estimator=MaximumLikelihoodEstimator)
res_mle=pd.DataFrame(columns=['Result_bad','Result_good','true value'])
step=30 #делаю выводы кусками по 30 шт, чтобы отслеживать прогресс

y_test=X_test['Result']
temp=list(model_mle.nodes())
temp.remove('Result')

for i in np.arange(0,len(X_test),step):
    y_predicted1=model_mle.predict_probability(X_test[temp][i:i+step])
    y_predicted1['true value']=y_test[i:i+step]
    res_mle=res_mle.append(y_predicted1)
    #print(i/step)
print(res_mle)
```

	Result_bad	Result_good	true value
0	0.041925	0.958075	good
1	0.035866	0.964134	good
2	0.153351	0.846649	bad
3	0.011480	0.988520	good
4	0.087302	0.912698	good
...	...	...	...
1495	0.035866	0.964134	good
1496	0.035866	0.964134	good
1497	0.035866	0.964134	good
1498	0.035866	0.964134	good
1499	0.011480	0.988520	good

[1500 rows x 3 columns]

```
#настройка параметров(условных распределений) Bayesian Search
model=BayesianModel(edges)
model.fit(data=X_train, estimator=BayesianEstimator,equivalent_sample_size=20)
res=pd.DataFrame(columns=['Result_bad','Result_good','true value'])
step=30 #делаю выводы кусками по 30 шт, чтобы отслеживать прогресс

y_test=X_test['Result']
temp=list(model.nodes())
temp.remove('Result')

for i in np.arange(0,len(X_test),step):
    y_predicted1=model.predict_probability(X_test[temp][i:i+step])
    y_predicted1['true value']=y_test[i:i+step]
    res=res.append(y_predicted1)
    #print(i/step)
print(res)
```

	Result_bad	Result_good	true value
0	0.042399	0.957601	good
1	0.035925	0.964075	good
2	0.153475	0.846525	bad
3	0.011583	0.988417	good
4	0.089474	0.910526	good
...	...	...	...
1495	0.035925	0.964075	good
1496	0.035925	0.964075	good
1497	0.035925	0.964075	good
1498	0.035925	0.964075	good
1499	0.011583	0.988417	good

[1500 rows x 3 columns]

Видно, що оцінки алгоритмів не сильно відрізняються одне від одного. Також видно, що практично скрізь передбачена ймовірність good значно більша. Це пояснюється *дисбалансом класів* у вибірці(мало bad кредитів).

Типовий поріг класифікації (0.5) для даної задачі не підходить — тоді мережа буде давати константне передбачення “good” . Підберемо більший поріг:

```
MaximumLikelihoodEstimator
////////////////////////////////////////////////
threshold=0.8
[[   6   68]
 [   5 1421]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.951
precision(for bad)=tn/(tn+fn)=0.545, recall(for bad)=tn/(tn+fp)=0.081
////////////////////////////////////////////////
threshold=0.85
[[  34   40]
 [ 195 1231]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.843
precision(for bad)=tn/(tn+fn)=0.148, recall(for bad)=tn/(tn+fp)=0.459
////////////////////////////////////////////////
threshold=0.9
[[  34   40]
 [ 195 1231]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.843
precision(for bad)=tn/(tn+fn)=0.148, recall(for bad)=tn/(tn+fp)=0.459
////////////////////////////////////////////////
threshold=0.95
[[  44   30]
 [ 342 1084]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.752
precision(for bad)=tn/(tn+fn)=0.114, recall(for bad)=tn/(tn+fp)=0.595
```

```
BayesianSearch
////////////////////////////////////////////////
threshold=0.8
[[   6   68]
 [   5 1421]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.951
precision(for bad)=tn/(tn+fn)=0.545, recall(for bad)=tn/(tn+fp)=0.081
////////////////////////////////////////////////
threshold=0.85
[[  34   40]
 [ 196 1230]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.843
precision(for bad)=tn/(tn+fn)=0.148, recall(for bad)=tn/(tn+fp)=0.459
////////////////////////////////////////////////
threshold=0.9
[[  34   40]
 [ 196 1230]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.843
precision(for bad)=tn/(tn+fn)=0.148, recall(for bad)=tn/(tn+fp)=0.459
////////////////////////////////////////////////
threshold=0.95
[[  44   30]
 [ 345 1081]]
accuracy=(tp+tn)/(tp+fp+tn+fn)=0.75
precision(for bad)=tn/(tn+fn)=0.113, recall(for bad)=tn/(tn+fp)=0.595
```

Висновки.

Для даної задачі помилки типу FP(false positive — видали кредит, який не повернуть) значно “дорожчі”, ніж типу FN(false negative — не видали кредит, який би повернули). Тому, пороги 0.8 і нижче категорично не підходять, оскільки взагалі не вгадують кредити “bad” (видно із матриць помилок — лише 6 поганих кредитів вгадані правильно). Із зниженням порогу мережа взагалі наблизиться до константного передбачення “good”.

Метрика якості Accuracy не є об’єктивною для даної задачі(через дизбаланс класів). Константне передбачення “good” дає accuracy=0.95, але такий алгоритм нікому не потрібний. Ми зосереджуємо увагу на поганих кредитах, тому розглядаємо метрики

-precision(точність передбачення Negative — tn/(tn+fn))

-recall(міра виділення алгоритмом класу Negative — tn/(tn+fp))

В даній задачі метрика recall є приорітетною, оскільки ми прагнемо виділяти якомога більше об’єктів класу bad. Вибір порогу — на користь 0.95, можна навіть пробувати підвищувати (0.96, 0.97,0.98 тощо)