In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

DATASET_PATH = './diabetes.csv'
```

Подготовка датасета

В качестве тестовых данных был выбран датасет, направленный на распознавание диабета

https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)

In [2]:

```
df = pd.read_csv(DATASET_PATH)
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

Как видно из верхних 5 строк, датасет имеет вещественные признаки. Поэтому необходимо их перевести в несколько бинарных

Pregnancies - количество беременностей разбиваем на 3 группы. Бездетные, имеются дети и многодетные

Большинство параметров либо удаляем из-за ненадобности, либо разбиваем по нормам врачей

Age - возраст разбиваем по условному разбиению людей на

In [3]:

```
# Pregnancies
df['Pregnancies == 0'] = (df['Pregnancies'] == 0)
df['0 < Pregnancies <= 2'] = ((0 < df['Pregnancies']) & (df['Pregnancies'] <= 2))
df['2 < Pregnancies <= 4'] = ((2 < df['Pregnancies']) & (df['Pregnancies'] <= 4))
df['4 < Pregnancies <= 6'] = ((4 < df['Pregnancies']) & (df['Pregnancies'] <= 6))
df['6 < Pregnancies <= 8'] = ((6 < df['Pregnancies']) & (df['Pregnancies'] <= 8))
df['Pregnancies > 8'] = (8 < df['Pregnancies'])
del df['Pregnancies']

# Glucose
df['Glucose <= 70'] = (df['Glucose'] <= 70)
df['70 < Glucose <= 100'] = ((70 < df['Glucose']) & (df['Glucose'] <= 100))
df['100 < Glucose <= 122'] = ((100 < df['Glucose']) & (df['Glucose'] <= 122))
df['122 < Glucose'] = (122 < df['Glucose'])
del df['Glucose']

# BloodPressure
df['Be <= 61'] = (df['BloodPressure'] <= 61)
df['61 < Be <= 75'] = ((61 < df['BloodPressure']) & (df['BloodPressure'] <= 75))
df['75 < Be <= 85'] = ((75 < df['BloodPressure']) & (df['BloodPressure'] <= 85))
df['85 < Be'] = (85 < df['BloodPressure'])
del df['BloodPressure']

# SkinThickness
df['Ss <= 19.8'] = (df['SkinThickness'] <= 19.8)
df['19.8 < Ss <= 39.6'] = ((19.8 < df['SkinThickness']) & (df['SkinThickness'] <= 39.6
))
df['39.6 < Ss'] = (39.6 < df['SkinThickness'])
del df['SkinThickness']

# Insulin
df['Insulin <= 42.3'] = (df['Insulin'] <= 42.3)
df['42.3 < Insulin <= 84.6'] = ((42.3 < df['Insulin']) & (df['Insulin'] <= 84.6))
df['84.6 < Insulin <= 100'] = ((84.6 < df['Insulin']) & (df['Insulin'] <= 100))
df['100 < Insulin <= 169.2'] = ((100 < df['Insulin']) & (df['Insulin'] <= 169.2))
df['169.2 < Insulin'] = (169.2 < df['Insulin'])
del df['Insulin']

# BMI
del df['BMI']

# DiabetesPedigreeFunction
df['Dn <= 0.31'] = (df['DiabetesPedigreeFunction'] <= 0.31)
df['0.31 < Dn <= 0.55'] = ((0.31 < df['DiabetesPedigreeFunction']) & (df['DiabetesPedig
reeFunction'] <= 0.55))
df['0.55 < Dn <= 0.65'] = ((0.55 < df['DiabetesPedigreeFunction']) & (df['DiabetesPedig
reeFunction'] <= 0.65))
df['0.65 < Dn <= 0.78'] = ((0.65 < df['DiabetesPedigreeFunction']) & (df['DiabetesPedig
reeFunction'] <= 0.78))
df['0.78 < Dn'] = (0.78 < df['DiabetesPedigreeFunction'])
del df['DiabetesPedigreeFunction']

# Age
df['Age <= 27'] = (df['Age'] <= 27)
df['27 < Age <= 39'] = ((27 < df['Age']) & (df['Age'] <= 39))
df['39 < Age <= 57'] = ((39 < df['Age']) & (df['Age'] <= 57))
df['57 < Age'] = (57 < df['Age'])
del df['Age']
```

```
target = df['Outcome'].values
del df['Outcome']
data = np.array(df.values, dtype=np.int64)

X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size=0.15)
print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

```
(652, 31) (652,) (116, 31) (116,)
```

Обучение базовой модели

В качестве baseline был выбран RandomForest

Поверх случайных деревьев навесим GridSearch. Таким образом добавим кросс-валидацию, а также найдем лучше параметры для них

In [4]:

```
model = RandomForestClassifier()
param_grid = {
    'n_estimators': np.arange(3, 25, 1),
    'max_depth': np.arange(4, 6, 1),
    'min_samples_split': np.arange(2, 9, 2),
    'criterion': ['gini', 'entropy']
}
gs = GridSearchCV(
    model,
    param_grid=param_grid,
    scoring='accuracy',
    n_jobs=-1,
    cv=3,
    verbose=2
)
gs.fit(X_train, Y_train)
modelBaseline = gs.best_estimator_
```

```
Fitting 3 folds for each of 352 candidates, totalling 1056 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 1056 out of 1056 | elapsed:    4.2s finished
```

Обучение модели на основе алгоритма GALOIS

Алгоритм был взят из http://www.machinelearning.ru/wiki/images/6/6e/2015_417_KolmakovEA.pdf (http://www.machinelearning.ru/wiki/images/6/6e/2015_417_KolmakovEA.pdf)

Обучение занимает достаточно много времени вследствие неоптимальности модели

In [5]:

```
class CloseByOneAlgorithm:

    def __init__(self):
        self.G = None
        self.M = None
        self.I = None
        self.L = None

    def __call__(self, context):
        G, M = context.shape
        I = context
        # ({0, ... , G-1}, {0, ... , M-1}, I) - context
        self.G = set(np.arange(G))
        self.M = set(np.arange(M))
        self.I = (I == 1)
        self.L = [
            (set(), self.M)
        ]
        for g in range(G):
            D = self.__close_g_once(g)
            C = self.__close_M(D)
            self.__process(set([g]), g, (C, D))
        return self.L

    def __close_g_once(self, g):
        return set(np.where(self.I[g])[0])

    def __close_M(self, Y: set):
        return set(np.where(np.all(self.I[:, list(Y)], axis=1))[0])

    def __process(self, A, g, P):
        C, D = P
        dist = C - A
        if len(dist) > 0 and min(dist) < g:
            return
        self.L.append(P)
        for f in (self.G - C):
            if f <= g:
                continue
            Z = set.union(C, set([f]))
            Y = set.intersection(D, self.__close_g_once(f))
            X = self.__close_M(Y)
            self.__process(Z, f, (X, Y))
```

In [6]:

```
class modelGALOIS():

    def __init__(self):
        self.concepts = None
        self.Y = None

    def fit(self, X: np.ndarray, Y: np.ndarray):
        alg = CloseByOneAlgorithm()
        L = alg(X)
        M = len(X[0])
        self.concepts = []
        self.marks = []
        for A, B in L:
            if len(A) == 0:
                continue
            mark = Y[A.__iter__().__next__()]
            if (mark == Y[list(A)]).all():
                new_row = np.zeros(M)
                new_row[list(B)] = 1
                self.concepts.append(new_row)
                self.marks.append(mark)
        self.concepts = np.array(self.concepts)
        self.marks = np.array(self.marks)

    def predict(self, X: np.ndarray):
        X_t = (1 - X).T
        res = self.concepts @ X_t
        idx = np.argmin(res, axis=0)
        return self.marks[idx]
```

In [7]:

```
modelG = modelGALOIS()
modelG.fit(X_train, Y_train)
```

Сравнение моделей

In [8]:

```
Y_pred_bl = modelBaseline.predict(X_test)
Y_pred_g = modelG.predict(X_test)
```

Сравнение частот

	0_test	1_test
0_pred	TN	FP
1_pred	FN	TP

In [9]:

```
metrics.confusion_matrix(Y_pred_bl, Y_test, normalize='true')
```

Out[9]:

```
array([[0.68888889, 0.31111111],
       [0.23076923, 0.76923077]])
```

In [10]:

```
metrics.confusion_matrix(Y_pred_g, Y_test, normalize='true')
```

Out[10]:

```
array([[0.7195122 , 0.2804878 ],
       [0.26470588, 0.73529412]])
```

Accuracy

In [11]:

```
metrics.accuracy_score(Y_pred_bl, Y_test), metrics.accuracy_score(Y_pred_g, Y_test)
```

Out[11]:

```
(0.7068965517241379, 0.7241379310344828)
```

Precision

In [12]:

```
metrics.precision_score(Y_pred_bl, Y_test), metrics.precision_score(Y_pred_g, Y_test)
```

Out[12]:

```
(0.4166666666666667, 0.5208333333333334)
```

Recall

In [13]:

```
metrics.recall_score(Y_pred_bl, Y_test), metrics.recall_score(Y_pred_g, Y_test)
```

Out[13]:

```
(0.7692307692307693, 0.7352941176470589)
```

Вывод

Исходя из сравнения можно сделать вывод: использование решеток обоснованно для специфичных датасетов

В данном конкретном случае качество решеток немного лучше