ДОДАТОК А

Контекстні діаграми

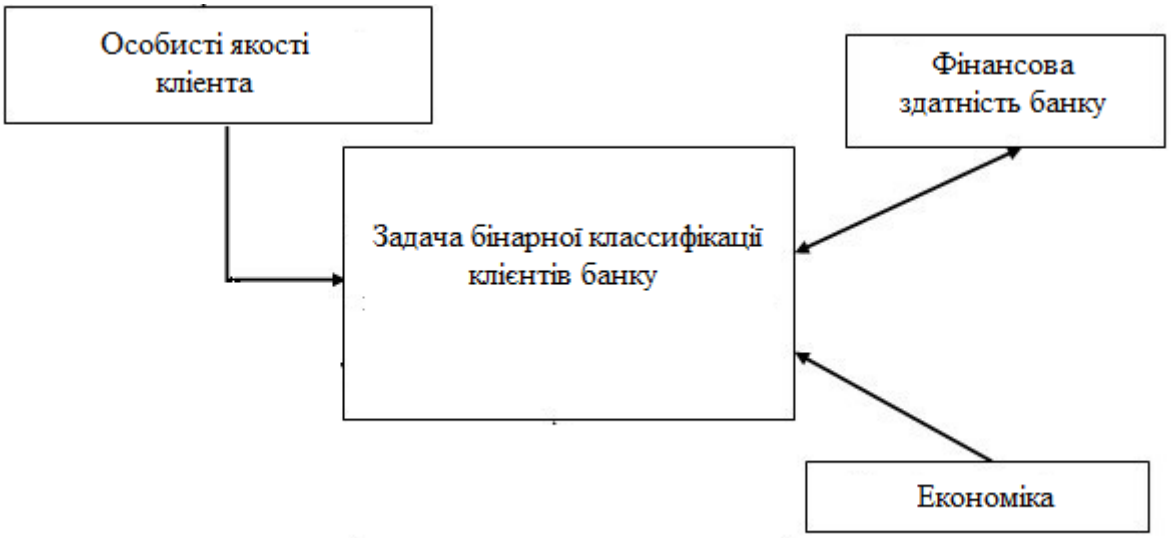


Рисунок А.1 – Модель зовнішнього середовища системи

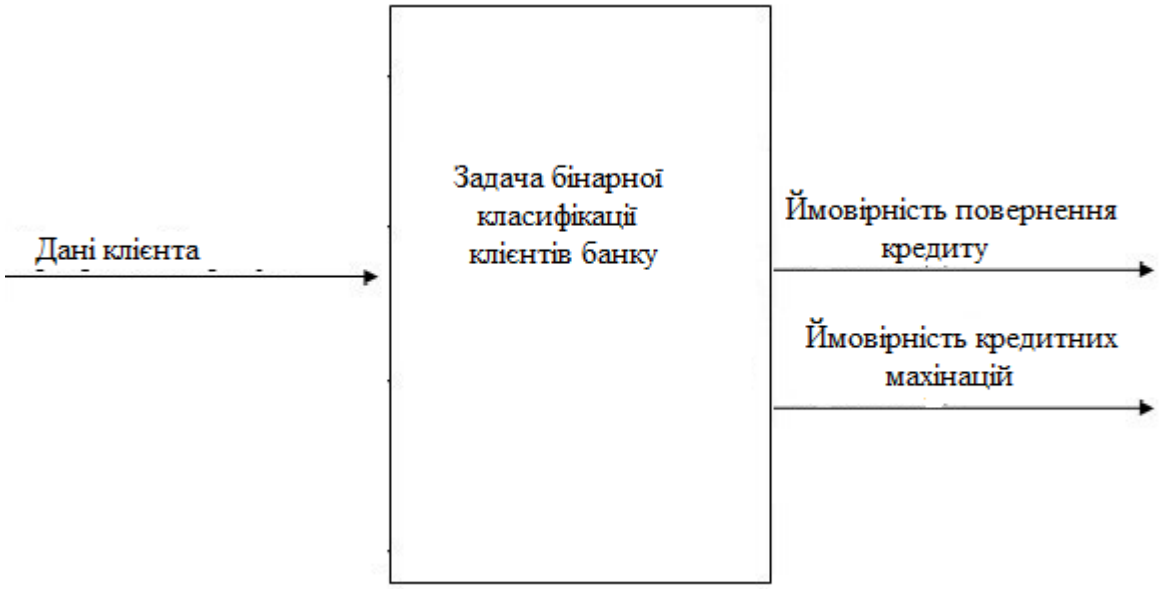


Рисунок А.2 – Модель типу «чорний ящик»

Рисунок А.3 – Контекстна діаграма досліджуваної системи

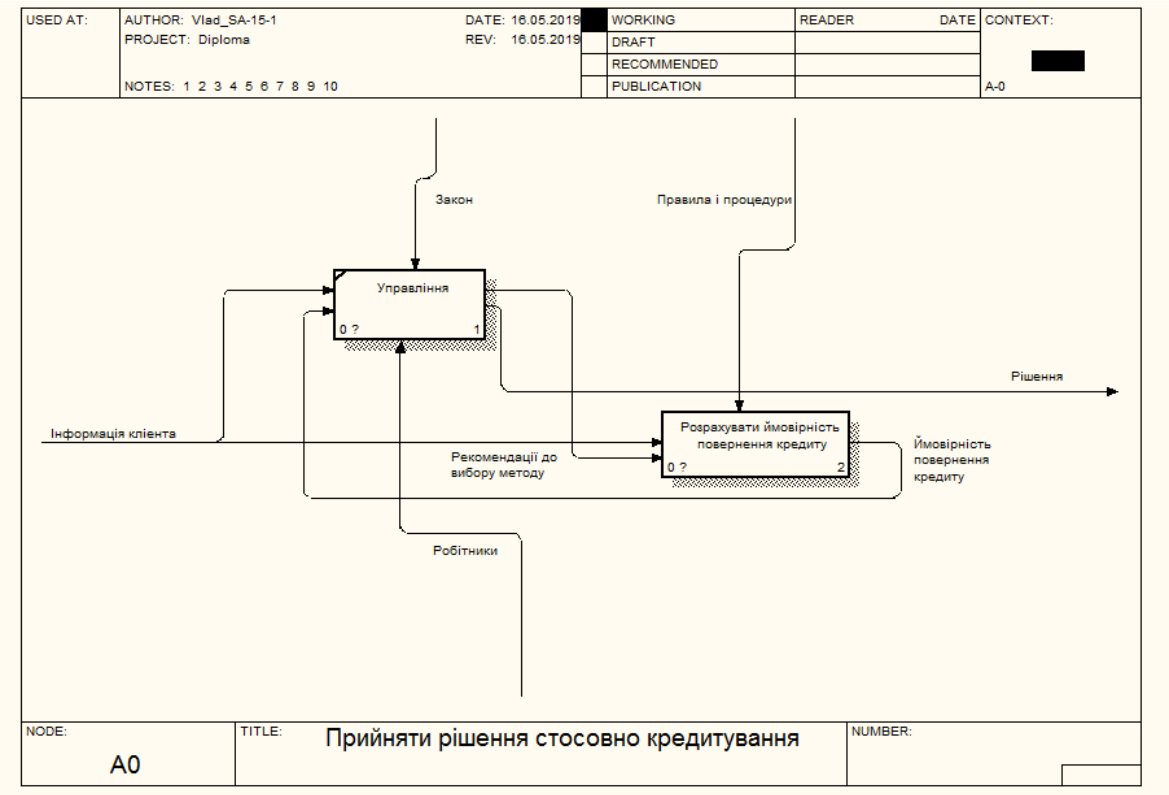


Рисунок А.4 – Підсумкова діаграма декомпозиції системи рівня А0

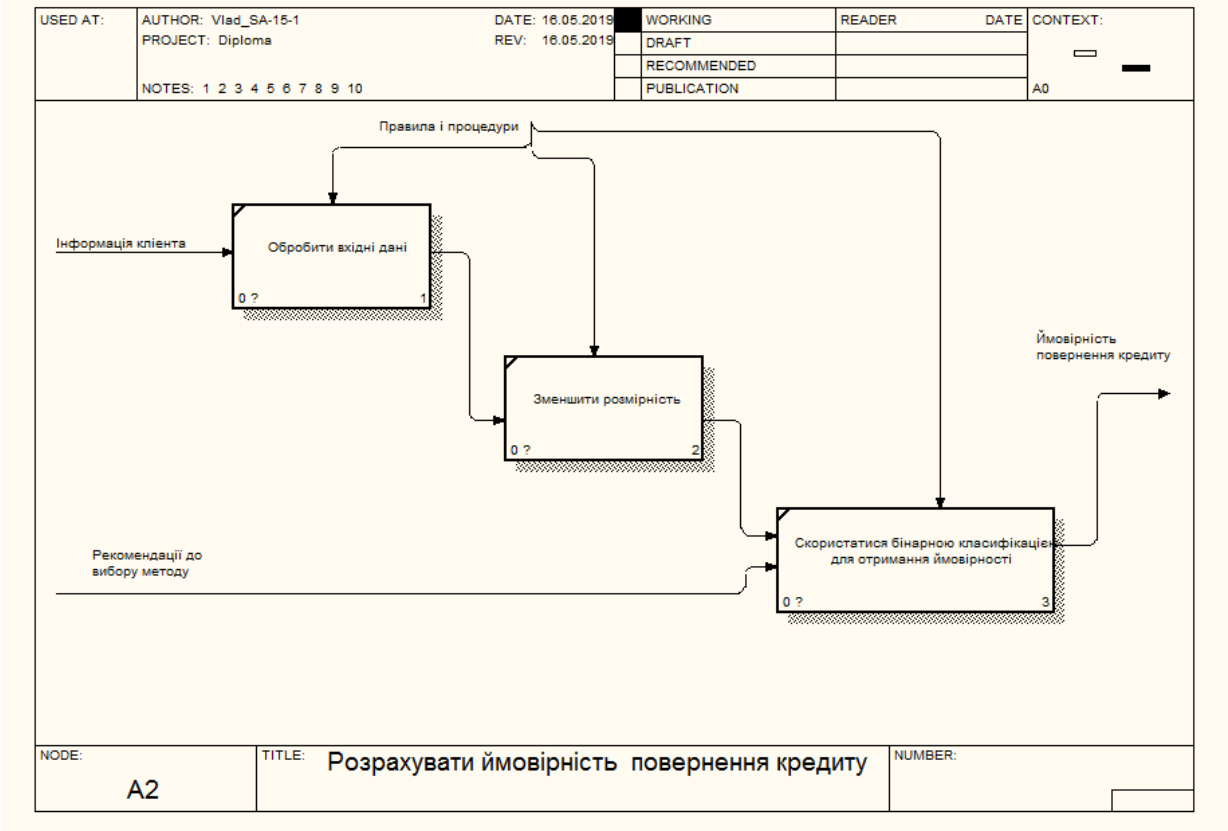


Рисунок А.5 – Підсумкова діаграма декомпозиції системи рівня А2

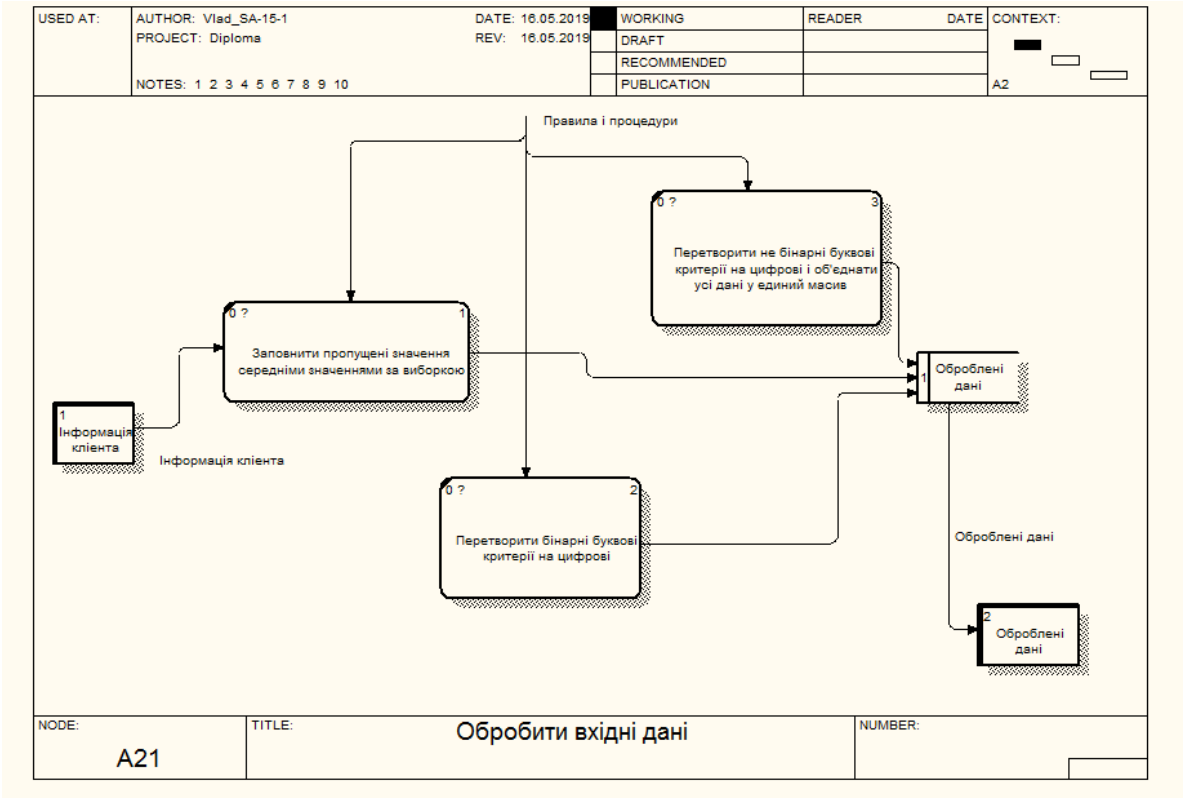


Рисунок А.6 – DFD-діаграма 1-го рівня декомпозиції

ДОДАТОК Б

Програмна реалізація методу аналізу ієрархій

```
(*Точность,время работі,Ресурс ЄВМ,Стоимость*)
M = {{1, 2, 4, 3}, {1/2, 1, 3, 2}, {1/4, 1/3, 1, 1/2}, {1/3, 1/2, 2, 1}, {"-", "-", "-", "-", "-"}};

Do[AppendTo[M[[i]], N[(∏_{j=1}^{4} M[[i, j]])^(1/4)]], {i, 1, 4}]
(o··· добавить в конец к   числ-ное приближение)

S = Sum[M[[i, 5]], {i, 1, 4}];
    сумма
AppendTo[M[[5]], S];
добавить в конец к

Do[AppendTo[M[[i]], M[[i, 5]] / S], {i, 1, 4}];
(o··· добавить в конец к)

PrependTo[M, {"", "K1", "K2", "K3", "K4", "Среднее геометрич", "Вектор приоритетов"}];
добавить в начало к
Do[PrependTo[M[[i]], {"K", i - 1}], {i, 2, 5}]
(o··· добавить в начало к)

Grid[M, Frame → All]
таблица  рамка  всё
```

	K1	K2	K3	K4	Среднее геометрич	Вектор приоритетов
{K, 1}	1	2	4	3	2.21336	0.466849
{K, 2}	1/2	1	3	2	1.31607	0.27759
{K, 3}	1/4	1/3	1	1/2	0.451801	0.0952951
{K, 4}	1/3	1/2	2	1	0.759836	0.160267
-	-	-	-	-	4.74107	

```
Sum[M[[i, 7]], {i, 2, 5}]
сумма
```

```
1.
```

```
Y = {};
Do[AppendTo[Y, N[Sum[M[[j, i]], {j, 2, 5}]]], {i, 2, 5}]
(o··· добавить в ко··· [··· сумма)

Y
```

```
{2.08333, 3.83333, 10., 6.5}
```

```
λ = Sum[Y[[i]] * M[[i + 1, 7]], {i, 1, 4}]
    сумма
```

```
4.03138
```

$$CIk = \frac{\lambda - 4}{4 - 1}$$

```
0.0104596
```

$$CRk = \frac{CIk}{0.9}$$

```
0.0116217

(*logistReger,knn,LinRegr*)
M1 = {{1, 2, 4}, {1/2, 1, 3}, {1/4, 1/3, 1}};
M2 = {{1, 1/2, 1/3}, {2, 1, 1/2}, {3, 2, 1}};
M3 = {{1, 2, 1/2}, {1/2, 1, 1/2}, {2, 2, 1}};
M4 = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};

(*K1*)

Do[AppendTo[M1[[i]], N[(∏_{j=1}^{3} M1[[i, j]])^(1/3)]], {i, 1, 3}]
(o··· добавить в конец к   числ-ное приближение)

S1 = Sum[M1[[i, 4]], {i, 1, 3}];
    сумма
```

```
      сумма
Do[AppendTo[M1[[i]], M1[[i, 4]] / S1], {i, 1, 3}];
добавить в конец к
(*K2*)
Do[AppendTo[M2[[i]], N[(∏_{j=1}^{3} M2[[i, j]])^(1/3)]], {i, 1, 3}]
добавить в конец к   числовое приближение

S2 = Sum[M2[[i, 4]], {i, 1, 3}];
      сумма
Do[AppendTo[M2[[i]], M2[[i, 4]] / S2], {i, 1, 3}];
добавить в конец к
(*K3*)
Do[AppendTo[M3[[i]], N[(∏_{j=1}^{3} M3[[i, j]])^(1/3)]], {i, 1, 3}]
добавить в конец к   числовое приближение

S3 = Sum[M3[[i, 4]], {i, 1, 3}];
      сумма
Do[AppendTo[M3[[i]], M3[[i, 4]] / S3], {i, 1, 3}];
добавить в конец к
(*K4*)
Do[AppendTo[M4[[i]], N[(∏_{j=1}^{3} M4[[i, j]])^(1/3)]], {i, 1, 3}]
добавить в конец к   числовое приближение

S4 = Sum[M4[[i, 4]], {i, 1, 3}];
      сумма
Do[AppendTo[M4[[i]], M4[[i, 4]] / S4], {i, 1, 3}];
добавить в конец к
(*P*)
p1 = p2 = p3 = p4 = {};
Do[AppendTo[p1, M1[[i, 5]]], {i, 1, 3}]
добавить в конец к
Do[AppendTo[p2, M2[[i, 5]]], {i, 1, 3}]
```

```
Do[AppendTo[p3, M3[[i, 5]]], {i, 1, 3}]
добавить в конец к
Do[AppendTo[p4, M4[[i, 5]]], {i, 1, 3}]
добавить в конец к

Print["K1 = ", Grid[M1, Frame → All], p1]
печатать      таблица   рамка   всё
Print["K2 = ", Grid[M2, Frame → All], p2]
печатать      таблица   рамка   всё
Print["K3 = ", Grid[M3, Frame → All], p3]
печатать      таблица   рамка   всё
Print["K4 = ", Grid[M4, Frame → All], p4]
печатать      таблица   рамка   всё
```

K1 =

1	2	4	2.	0.558425
1/2	1	3	1.14471	0.319618
1/4	1/3	1	0.43679	0.121957

{0.558425, 0.319618, 0.121957}

K2 =

1	1/2	1/3	0.550321	0.163424
2	1	1/2	1.	0.296961
3	2	1	1.81712	0.539615

{0.163424, 0.296961, 0.539615}

K3 =

1	2	1/2	1.	0.310814
1/2	1	1/2	0.629961	0.1958
2	2	1	1.5874	0.493386

{0.310814, 0.1958, 0.493386}

K4 =

1	1	1	1.	0.333333
1	1	1	1.	0.333333
1	1	1	1.	0.333333

{0.333333, 0.333333, 0.333333}

```
(*C1*)
Y1 = {};
Do[AppendTo[Y1, N[Sum[M1[[j, i]], {j, 1, 3}]]], {i, 1, 3}]
добавить в коне···   сумма
λ1 = Sum[Y1[[i]] * M1[[i, 5]], {i, 1, 3}];
```

```
CIk1 = (λ1 - 3)/(3 - 1); CRk1 = CIk1/0.58;
(*C2*)
Y2 = {};
Do[AppendTo[Y2, N[Sum[M2[[j, i]], {j, 1, 3}]]], {i, 1, 3}]
   добавить в коне··· |··· сумма
λ2 = Sum[Y2[[i]] * M2[[i, 5]], {i, 1, 3}];
     сумма
CIk2 = (λ2 - 3)/(3 - 1); CRk2 = CIk2/0.58;
(*C3*)
Y3 = {};
Do[AppendTo[Y3, N[Sum[M3[[j, i]], {j, 1, 3}]]], {i, 1, 3}]
   добавить в коне··· |··· сумма
λ3 = Sum[Y3[[i]] * M3[[i, 5]], {i, 1, 3}];
     сумма
CIk3 = (λ3 - 3)/(3 - 1); CRk3 = CIk3/0.58;
(*C4*)
Y4 = {};
Do[AppendTo[Y4, N[Sum[M4[[j, i]], {j, 1, 3}]]], {i, 1, 3}]
   добавить в коне··· |··· сумма
λ4 = Sum[Y4[[i]] * M4[[i, 5]], {i, 1, 3}];
     сумма
CIk4 = (λ4 - 3)/(3 - 1); CRk4 = CIk4/0.58;
CIkmass = {CIk1, CIk2, CIk3, CIk4};
CRkmass = {CRk1, CRk2, CRk3, CRk4};

Print[CIk1, " ", CIk2, " ", CIk3, " ", CIk4]
печатать
Print[CRk1, " ", CRk2, " ", CRk3, " ", CRk4]
печатать

0.00914735 0.00460136 0.0268108 0.

0.0157713 0.00793337 0.0462255 0.

P = {p1, p2, p3, p4};
P = Transpose[P];
    транспозиция
pk = {{M[[2, 7]]}, {M[[3, 7]]}, {M[[4, 7]]}, {M[[5, 7]]}};
Print[" p = ", MatrixForm[P], "*", MatrixForm[pk], " = ", MatrixForm[P.pk]]
печатать   матричная форма   матричная форма   матричная форма
```

$$p = \begin{pmatrix} 0.558425 & 0.163424 & 0.310814 & 0.333333 \\ 0.319618 & 0.296961 & 0.1958 & 0.333333 \\ 0.121957 & 0.539615 & 0.493386 & 0.333333 \end{pmatrix} * \begin{pmatrix} 0.466849 \\ 0.27759 \\ 0.0952951 \\ 0.160267 \end{pmatrix} = \begin{pmatrix} 0.389106 \\ 0.303728 \\ 0.307166 \end{pmatrix}$$

```
CI = CIk + pk;
CIA = CIk + Sum[pk[[i]] * CIkmass[[i]], {i, 1, 4}]
            сумма
RI = 0.90 + 0.58
CR = CIA/RI

{0.0185622}

1.48

{0.012542}
```

ДОДАТОК В

Програмна реалізація задачі кредитного скорінгу

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import re
%matplotlib inline
from sklearn import linear_model, datasets, metrics
from sklearn.decomposition import PCA as sklearnPCA
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
plt.style.use('ggplot')
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/crx.data'
data = pd.read_csv(url, header=None, na_values='?')
```

```
[ ]  data.shape
```

```
(690, 16)
```

```
[ ]  data.tail()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	260.0	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	200.0	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	200.0	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	280.0	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	0.0	0	-

```
[ ]  data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	202.0	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0	+

```
[ ]  data.columns = ['A' + str(i) for i in range(1, 16)] + ['class']
     data.head()
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	class
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	202.0	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0	+

```
[ ]  data.describe()
```

	A2	A3	A8	A11	A14	A15
count	678.000000	690.000000	690.000000	690.00000	677.000000	690.000000
mean	31.568171	4.758725	2.223406	2.40000	184.014771	1017.385507
std	11.957862	4.978163	3.346513	4.86294	173.806768	5210.102598
min	13.750000	0.000000	0.000000	0.00000	0.000000	0.000000
25%	22.602500	1.000000	0.165000	0.00000	75.000000	0.000000
50%	28.460000	2.750000	1.000000	0.00000	160.000000	5.000000
75%	38.230000	7.207500	2.625000	3.00000	276.000000	395.500000
max	80.250000	28.000000	28.500000	67.00000	2000.000000	100000.000000

```
[ ]  categorial_columns = [c for c in data.columns if data[c].dtype.name == 'object']
     numerical_columns = [c for c in data.columns if data[c].dtype.name != 'object']
```

```
[ ]  print(categorial_columns)
     print(numerical_columns)
```

```
['A1', 'A4', 'A5', 'A6', 'A7', 'A9', 'A10', 'A12', 'A13', 'class']
['A2', 'A3', 'A8', 'A11', 'A14', 'A15']
```

```
[ ]  data[categorial_columns].describe()
```

	A1	A4	A5	A6	A7	A9	A10	A12	A13	class
count	678	684	684	681	681	690	690	690	690	690
unique	2	3	3	14	9	2	2	2	3	2
top	b	u	g	c	v	t	f	f	g	-
freq	468	519	519	137	399	361	395	374	625	383

```
[ ]  from pandas.plotting import scatter_matrix
     scatter_matrix(data,figsize=(16, 16), alpha=0.05, marker="o")
```

```
[ ]
```

	A2	A3	A8	A11	A14	A15
A2	1.000000	0.202317	0.395751	0.185912	-0.079812	0.018553
A3	0.202317	1.000000	0.298902	0.271207	-0.224242	0.123121
A8	0.395751	0.298902	1.000000	0.322330	-0.077163	0.051345
A11	0.185912	0.271207	0.322330	1.000000	-0.120096	0.063692
A14	-0.079812	-0.224242	-0.077163	-0.120096	1.000000	0.066853
A15	0.018553	0.123121	0.051345	0.063692	0.066853	1.000000

```
[ ]  data.count(axis=0)
```

```
A1       678
A2       678
A3       690
A4       684
A5       684
A6       681
A7       681
A8       690
A9       690
A10      690
A11      690
A12      690
A13      690
A14      677
A15      690
class    690
dtype: int64
```

```
[ ]  data = data.fillna(data.median(axis=0), axis=0)
     data.count(axis=0)
```

```
A1       678
A2       690
A3       690
A4       684
A5       684
A6       681
A7       681
A8       690
A9       690
A10      690
A11      690
A12      690
A13      690
A14      690
A15      690
class    690
dtype: int64
```

```
[ ]  data['A1'].describe().top
```

```
'b'
```

```
[ ]  data['A1'] = data['A1'].fillna(data['A1'].describe().top)
```

```
[ ]  data_describe = data.describe(include=[object])
     for c in categorial_columns:
         data[c] = data[c].fillna(data_describe[c]['top'])
```

```
[ ]  data.describe(include=[object])
```

	A1	A4	A5	A6	A7	A9	A10	A12	A13	class
count	690	690	690	690	690	690	690	690	690	690
unique	2	3	3	14	9	2	2	2	3	2
top	b	u	g	c	v	t	f	f	g	-
freq	480	525	525	146	408	361	395	374	625	383

```
[ ]  data.describe()
```

	A2	A3	A8	A11	A14	A15
count	690.000000	690.000000	690.000000	690.00000	690.000000	690.000000
mean	31.514116	4.758725	2.223406	2.40000	183.562319	1017.385507
std	11.860245	4.978163	3.346513	4.86294	172.190278	5210.102598
min	13.750000	0.000000	0.000000	0.00000	0.000000	0.000000
25%	22.670000	1.000000	0.165000	0.00000	80.000000	0.000000
50%	28.460000	2.750000	1.000000	0.00000	160.000000	5.000000
75%	37.707500	7.207500	2.625000	3.00000	272.000000	395.500000
max	80.250000	28.000000	28.500000	67.00000	2000.000000	100000.000000

```
[ ]  binary_columns    = [c for c in categorial_columns if data_describe[c]['unique'] == 2]
     nonbinary_columns = [c for c in categorial_columns if data_describe[c]['unique'] > 2]
     print (binary_columns, nonbinary_columns)
```

```
['A1', 'A9', 'A10', 'A12', 'class'] ['A4', 'A5', 'A6', 'A7', 'A13']
```

```
[ ]  data.describe()
```

	A2	A3	A8	A11	A14	A15
count	690.000000	690.000000	690.000000	690.00000	690.000000	690.000000
mean	31.514116	4.758725	2.223406	2.40000	183.562319	1017.385507
std	11.860245	4.978163	3.346513	4.86294	172.190278	5210.102598
min	13.750000	0.000000	0.000000	0.00000	0.000000	0.000000
25%	22.670000	1.000000	0.165000	0.00000	80.000000	0.000000
50%	28.460000	2.750000	1.000000	0.00000	160.000000	5.000000
75%	37.707500	7.207500	2.625000	3.00000	272.000000	395.500000
max	80.250000	28.000000	28.500000	67.00000	2000.000000	100000.000000

```
[ ]  data.at[data['A1'] == 'b', 'A1'] = 0
     data.at[data['A1'] == 'a', 'A1'] = 1
     data['A1'].describe()
```

```
count    690.000000
mean       0.304348
std        0.460464
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: A1, dtype: float64
```

```
data_describe = data.describe(include=[object])
data_describe
```

```
data_describe = data.describe(include=[object])
data_describe
```

	A4	A5	A6	A7	A9	A10	A12	A13	class
count	690	690	690	690	690	690	690	690	690
unique	3	3	14	9	2	2	2	3	2
top	u	g	c	v	t	f	f	g	-
freq	525	525	146	408	361	395	374	625	383

```
[ ]  for c in binary_columns[1:]:
         top = data_describe[c]['top']
         top_items = data[c] == top
         data.loc[top_items, c] = 0
         data.loc[np.logical_not(top_items), c] = 1
```

```
[ ]      data[binary_columns].describe()
```

	A1	A9	A10	A12	class
count	690.000000	690.000000	690.000000	690.000000	690.000000
mean	0.304348	0.476812	0.427536	0.457971	0.444928
std	0.460464	0.499824	0.495080	0.498592	0.497318
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

```
[ ]  data.describe()
```

```
[ ] data.describe()
```

	A1	A2	A3	A8	A9	A10	A11	A12	A14	A15	class
count	690.000000	690.000000	690.000000	690.000000	690.000000	690.000000	690.00000	690.000000	690.000000	690.000000	690.000000
mean	0.304348	31.514116	4.758725	2.223406	0.476812	0.427536	2.40000	0.457971	183.562319	1017.385507	0.444928
std	0.460464	11.860245	4.978163	3.346513	0.499824	0.495080	4.86294	0.498592	172.190278	5210.102598	0.497318
min	0.000000	13.750000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	22.670000	1.000000	0.165000	0.000000	0.000000	0.00000	0.000000	80.000000	0.000000	0.000000
50%	0.000000	28.460000	2.750000	1.000000	0.000000	0.000000	0.00000	0.000000	160.000000	5.000000	0.000000
75%	1.000000	37.707500	7.207500	2.625000	1.000000	1.000000	3.00000	1.000000	272.000000	395.500000	1.000000
max	1.000000	80.250000	28.000000	28.500000	1.000000	1.000000	67.00000	1.000000	2000.000000	100000.000000	1.000000

```
[ ] data['A4'].unique()
```

```
array(['u', 'y', 'l'], dtype=object)
```

```
[ ] data_nonbinary = pd.get_dummies(data[nonbinary_columns])
    print (data_nonbinary.columns)
```

```
Index(['A4_l', 'A4_u', 'A4_y', 'A5_g', 'A5_gg', 'A5_p', 'A6_aa', 'A6_c',
       'A6_cc', 'A6_d', 'A6_e', 'A6_ff', 'A6_i', 'A6_j', 'A6_k', 'A6_m',
       'A6_q', 'A6_r', 'A6_w', 'A6_x', 'A7_bb', 'A7_dd', 'A7_ff', 'A7_h',
       'A7_j', 'A7_n', 'A7_o', 'A7_v', 'A7_z', 'A13_g', 'A13_p', 'A13_s'],
      dtype='object')
```

```
[ ] data_numerical = data[numerical_columns]
    data_numerical = (data_numerical - data_numerical.mean()) / data_numerical.std()
    data_numerical.describe()
```

	A2	A3	A8	A11	A14	A15
count	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02	6.900000e+02
mean	-2.581751e-15	2.085288e-16	1.879334e-16	2.067589e-16	4.827057e-17	-1.448117e-18
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.497787e+00	-9.559198e-01	-6.643947e-01	-4.935286e-01	-1.066043e+00	-1.952717e-01
25%	-7.456942e-01	-7.550425e-01	-6.150897e-01	-4.935286e-01	-6.014412e-01	-1.952717e-01
50%	-2.575087e-01	-4.035072e-01	-3.655762e-01	-4.935286e-01	-1.368388e-01	-1.943120e-01
75%	5.221970e-01	4.919034e-01	1.200038e-01	1.233822e-01	5.136044e-01	-1.193615e-01
max	4.109180e+00	4.668645e+00	7.851932e+00	1.328414e+01	1.054901e+01	1.899821e+01

```
[ ] data = pd.concat((data_numerical, data[binary_columns], data_nonbinary), axis=1)
    data = pd.DataFrame(data, dtype=float)
    print(data.shape)
    print (data.columns)
    print(data)
```

```
[ ] sclearn_PCA = sklearnPCA(n_components = 2)
    sklearn_transf = sclearn_PCA.fit_transform(data)
    X = sklearn_transf;
    y = data['class']
```

```
[ ] print (X.shape)
    print (y.shape)
    N, d = X.shape
```

```
(690, 2)
(690,)
```

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)

    N_train, _ = X_train.shape
    N_test,  _ = X_test.shape
    print (N_train, N_test)
```

```
483 207
```

```
[ ] from sklearn.linear_model import LogisticRegressionCV

    logis = LogisticRegressionCV()
    res = logis.fit(X_train, y_train)
```

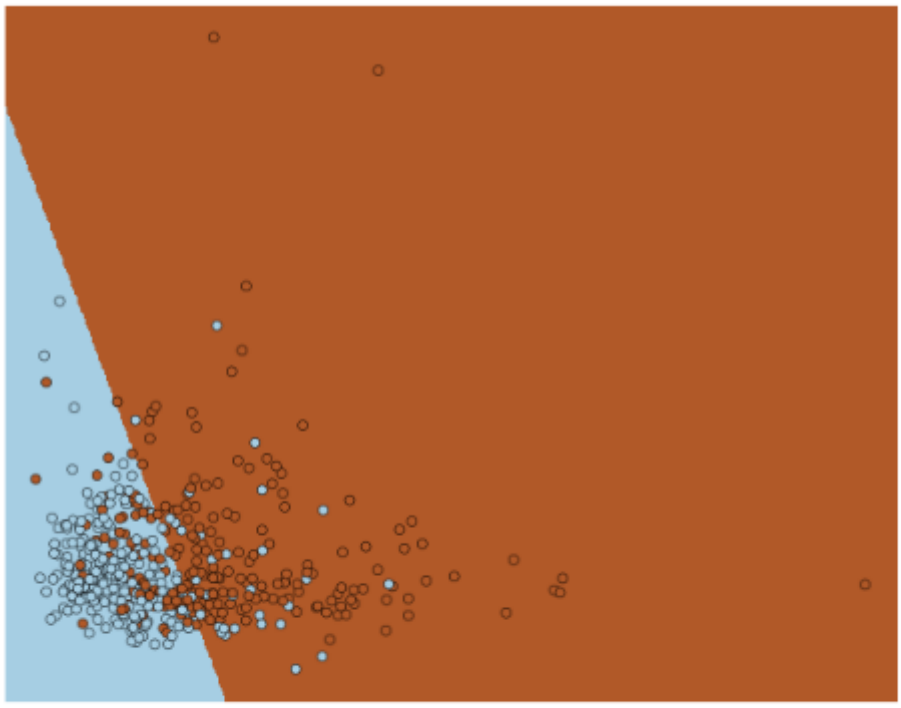
```
[ ]  print(res.scores_)
```

```
{1.0: array([[0.54037267, 0.62111801, 0.68944099, 0.73291925, 0.72049689,
         0.71428571, 0.71428571, 0.71428571, 0.71428571, 0.71428571],
        [0.54037267, 0.60248447, 0.7515528 , 0.80745342, 0.80745342,
         0.8136646 , 0.8136646 , 0.8136646 , 0.8136646 , 0.8136646 ],
        [0.54037267, 0.63354037, 0.75776398, 0.77018634, 0.77018634,
         0.77639752, 0.77639752, 0.77639752, 0.77639752, 0.77639752]])}
```

```
[ ]  y_train_predict = logis.predict(X_train)
     y_test_predict = logis.predict(X_test)

     err_train = np.mean(y_train != y_train_predict)
     err_test  = np.mean(y_test  != y_test_predict)
     print (1-err_train, 1-err_test)
```

```
0.7701863354037267 0.8115942028985508
```

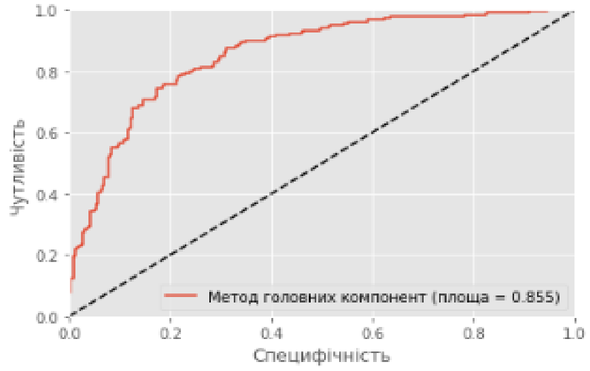
```
[ ]  X=np.array(X_train)
     Y=np.array(y_train)

     h = 0.02

     x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
     y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
     xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
     Z = logis.predict(np.c_[xx.ravel(), yy.ravel()])

     Z = Z.reshape(xx.shape)
     plt.figure(1, figsize=(10, 8))
     plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

     plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)

     plt.xlim(xx.min(), xx.max())
     plt.ylim(yy.min(), yy.max())
     plt.xticks(())
     plt.yticks(())

     plt.show()
```



```
fprPCA, tprPCA, thresholds = roc_curve(y_train, logis.predict_proba(X_train)[:, 1])
roc_aucPCA = auc(fprPCA, tprPCA)
```

```
[ ] plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)
    plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('Специфічність')
    plt.ylabel('Чутливість')
    plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x7fa6ccb41b00>
```




```
X = data.drop(('class'), axis=1)
y = data['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 11)
res = logis.fit(X_train, y_train)
y_train_predict = logis.predict(X_train)
y_test_predict = logis.predict(X_test)
```

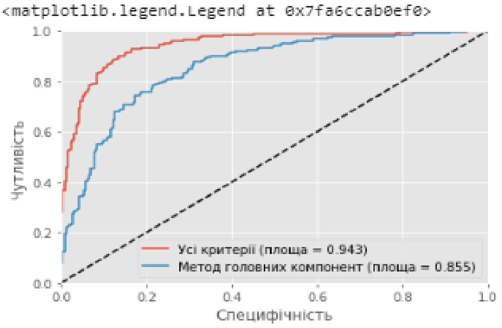
```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:947: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

```
[ ] err_train = np.mean(y_train != y_train_predict)
    err_test  = np.mean(y_test  != y_test_predict)
    print (1-err_train, 1-err_test)
```

```
0.8799171842650103 0.9033816425120773
```

```
fprAll, tprAll, thresholds = roc_curve(y_train, logis.predict_proba(X_train)[:, 1])
roc_aucAll = auc(fprAll, tprAll)

# plt.subplots(figsize = (10,5))
plt.plot(fprAll, tprAll, label='Усі критерії (площа = %0.3f)' % roc_aucAll)
plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)
plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Специфічність')
plt.ylabel('Чутливість')
plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x7fa6ccab0ef0>
```




```
[ ] res = logis.fit(X_train, y_train)
    y_train_predict = logis.predict(X_train)
    y_test_predict = logis.predict(X_test)
```

```
columns = np.array(X_train.columns)
finalErrTrain = 0.5
for i in range (X_train.shape[1]):
  for j in range(i+1,X_train.shape[1]):
        res = logis.fit(X_train[columns[[i,j]]], y_train)
        y_train_predict = logis.predict(X_train[columns[[i,j]]])
        err_train = np.mean(y_train != y_train_predict)
        if finalErrTrain>err_train :
          cTemp = X_train[columns[[i,j]]]
          finalI = i
          finalJ = j
          finalErrTrain = err_train
```

```
[ ]  usedCriterias=[]
     usedCriterias = np.append(usedCriterias,columns[[finalI,finalJ]])
     removedColumns = np.delete(columns,[finalI,finalJ])
     print(usedCriterias)
     print(removedColumns)
```

```
['A14' 'A9']
['A2' 'A3' 'A8' 'A11' 'A15' 'A1' 'A10' 'A12' 'A4_l' 'A4_u' 'A4_y' 'A5_g'
 'A5_gg' 'A5_p' 'A6_aa' 'A6_c' 'A6_cc' 'A6_d' 'A6_e' 'A6_ff' 'A6_i' 'A6_j'
 'A6_k' 'A6_m' 'A6_q' 'A6_r' 'A6_w' 'A6_x' 'A7_bb' 'A7_dd' 'A7_ff' 'A7_h'
 'A7_j' 'A7_n' 'A7_o' 'A7_v' 'A7_z' 'A13_g' 'A13_p' 'A13_s']
```

```
[ ]  res = logis.fit(X_train[usedCriterias], y_train)
     y_test_predict = logis.predict(X_test[usedCriterias])
     err_test  = np.mean(y_test  != y_test_predict)
     print('Train: ', 1-finalErrTrain,'Test: ', 1-err_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv
  warnings.warn(CV_WARNING, FutureWarning)
Train:  0.8509316770186335 Test:  0.8695652173913043
```

```
for j in range (40):
  b = cTemp
  updated = 0
  for i in range (removedColumns.shape[0]):
    a = np.array(X_train[removedColumns[[i]]])
    c = np.concatenate((b,a),axis = 1)
    res = logis.fit(c, y_train)
    y_train_predict = logis.predict(c)
    err_train = np.mean(y_train != y_train_predict)
    if finalErrTrain>err_train :
      finalI = i
      finalErrTrain = err_train
      cTemp = c
      updated = 1
  if updated == 0:
    break
  usedCriterias = np.append(usedCriterias,removedColumns[finalI])
  removedColumns = np.delete(removedColumns,finalI)
```

```
[ ]  print(usedCriterias)
     print(removedColumns)
```

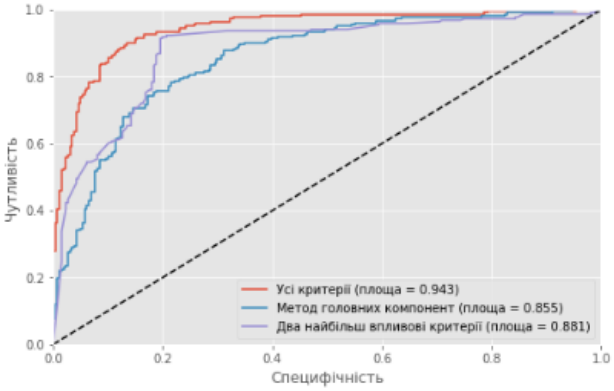
```
['A14' 'A9' 'A6_ff' 'A2' 'A13_p']
['A3' 'A8' 'A11' 'A15' 'A1' 'A10' 'A12' 'A4_l' 'A4_u' 'A4_y' 'A5_g'
 'A5_gg' 'A5_p' 'A6_aa' 'A6_c' 'A6_cc' 'A6_d' 'A6_e' 'A6_i' 'A6_j' 'A6_k'
 'A6_m' 'A6_q' 'A6_r' 'A6_w' 'A6_x' 'A7_bb' 'A7_dd' 'A7_ff' 'A7_h' 'A7_j'
 'A7_n' 'A7_o' 'A7_v' 'A7_z' 'A13_g' 'A13_s']
```

```
res = logis.fit(X_train[usedCriterias], y_train)
y_train_predict = logis.predict(X_train[usedCriterias])
y_test_predict = logis.predict(X_test[usedCriterias])
err_train = np.mean(y_train != y_train_predict)
err_test  = np.mean(y_test  != y_test_predict)
print('Train: ', 1-err_train,'Test: ', 1-err_test)
```

```
Train:  0.8612836438923396 Test:  0.8792270531400966
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will chang
  warnings.warn(CV_WARNING, FutureWarning)
```

```
res = logis.fit(X_train[usedCriterias], y_train)
fprStep, tprStep, thresholds = roc_curve(y_train, logis.predict_proba(X_train[usedCriterias])[:, 1])
roc_aucStep = auc(fprStep, tprStep)

res = logis.fit(X_train[usedCriterias[[0,1]]], y_train)
fprTwoMain, tprTwoMain, thresholds = roc_curve(y_train, logis.predict_proba(X_train[usedCriterias[[0,1]]])[:, 1])
roc_aucTwoMain = auc(fprTwoMain, tprTwoMain)

plt.subplots(figsize = (8,5))
plt.plot(fprAll, tprAll, label='Усі критерії (площа = %0.3f)' % roc_aucAll)
plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)
plt.plot(fprTwoMain, tprTwoMain, label='Два найбільш впливові критерії (площа = %0.3f)' % roc_aucTwoMain)
# plt.plot(fprStep, tprStep, label='Покрокова регресія (площа = %0.3f)' % roc_aucStep)

plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Специфічність')
plt.ylabel('Чутливість')
plt.legend(loc="lower right")
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change
  warnings.warn(CV_WARNING, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default value of cv will change
  warnings.warn(CV_WARNING, FutureWarning)
<matplotlib.legend.Legend at 0x7fa6cc69bac8>
```



ДОДАТОК Г

Програмна реалізація задачі Fraud-скорінгу

```
[ ] from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
```

```
    Mounted at /content/drive
```

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sb
    import re
    %matplotlib inline
    from sklearn import linear_model, datasets, metrics
    from sklearn.linear_model import LinearRegression
    from sklearn.decomposition import PCA as sklearnPCA
    from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
    from sklearn.svm import SVC # SVM algorithm
    from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm
    from sklearn.ensemble import VotingClassifier # Random forest tree algorithm
    from xgboost import XGBClassifier # XGBoost algorithm
    from sklearn.metrics import roc_curve
    from sklearn.metrics import auc
    import time
    plt.style.use('ggplot')
```

```
[ ] # IMPORTING DATA

    data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/creditcard.csv', header=None, na_values='?',low_memory=False, skiprows=1)
```

```
[ ] data[30].describe()
```

```
    count    284807.000000
    mean          0.001727
    std           0.041527
    min           0.000000
    25%           0.000000
    50%           0.000000
    75%           0.000000
    max           1.000000
    Name: 30, dtype: float64
```

```
[ ] categorial_columns = [c for c in data.columns if data[c].dtype.name == 'object']
    numerical_columns = [c for c in data.columns if data[c].dtype.name != 'object']
```

```
[ ] print(categorial_columns)
    print(numerical_columns)
```

```
    []
    ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9', 'A10', 'A11', 'A12', 'A13', 'A14', 'A15', 'A16', 'A17', 'A18', 'A19', 'A20', 'A21', 'A22', 'A23', 'A24', 'A25', 'A26', 'A27', 'A28', 'A29', 'A30', 'class']
```

```
data.count(axis=0)
```

```
[ ] data = data.fillna(data.median(axis=0), axis=0)
    data.count(axis=0)
```

```
data['A1'].describe()
```

```
    count    284807.000000
    mean      94813.859575
    std       47488.145955
    min           0.000000
    25%       54201.500000
    50%       84692.000000
    75%      139320.500000
    max      172792.000000
    Name: A1, dtype: float64
```

```
data.at[data['A1'] == 'b', 'A1'] = 0
data.at[data['A1'] == 'a', 'A1'] = 1
data['A1'].describe()
```

```
count    284807.000000
mean      94813.859575
std       47488.145955
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: A1, dtype: float64
```

```
data.describe()
```

```
data_numerical = data[numerical_columns]
data_numerical = (data_numerical - data_numerical.mean()) / data_numerical.std()
data_numerical.describe()
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8
mean	-1.399629e-15	-2.369347e-15	-9.299088e-17	6.265771e-15	1.966167e-16	1.757901e-15	-2.227124e-16	1.663279e-15	2.504277e-16	1.
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.0
min	-1.996580e+00	-2.879850e+01	-4.403521e+01	-3.187168e+01	-4.013912e+00	-8.240795e+01	-1.963602e+01	-3.520933e+01	-6.130242e+01	-1.2
25%	-8.552105e-01	-4.698909e-01	-3.624701e-01	-5.872131e-01	-5.993777e-01	-5.010677e-01	-5.766811e-01	-4.478852e-01	-1.746801e-01	-5.
50%	-2.131450e-01	9.245335e-03	3.965677e-02	1.186122e-01	-1.401721e-02	-3.936675e-02	-2.058043e-01	3.241718e-02	1.871979e-02	-4.
75%	9.372158e-01	6.716927e-01	4.867194e-01	6.774557e-01	5.250073e-01	4.433457e-01	2.991620e-01	4.611099e-01	2.740780e-01	5.
max	1.642055e+00	1.253349e+00	1.335773e+01	6.187982e+00	1.191872e+01	2.521409e+01	5.502005e+01	9.747807e+01	1.675150e+01	1.

```
sclearn_PCA = sklearnPCA(n_components = 2)
sklearn_transf = sclearn_PCA.fit_transform(data)
X = sklearn_transf;
y = data['class']
```

```
print (X.shape)
print (y.shape)
N, d = X.shape
```

```
(284807, 2)
(284807,)
```

```
print(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.6,train_size = 0.3, random_state = 11)

N_train, _ = X_train.shape
N_test,  _ = X_test.shape
print (N_train, N_test)
```

```
85442 170885
```

```
from sklearn.linear_model import LogisticRegressionCV

logis = LogisticRegressionCV(solver="liblinear")

logis_MM_start_time = time.time()
res = logis.fit(X_train, y_train)
logis_MM_end_time = time.time()
```

```
y_train_predict = logis.predict(X_train)
y_test_predict = logis.predict(X_test)

err_train = np.mean(y_train != y_train_predict)
err_test  = np.mean(y_test  != y_test_predict)
print("main component")
print (1-err_train, 1-err_test)
```

```
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')

tree_start_time = time.time()
clf = tree_model.fit(X_train, y_train)
tree_end_time = time.time()

tree_train = tree_model.predict(X_train)
tree_test = tree_model.predict(X_test)

err_tree_train = np.mean(y_train != tree_train)
err_tree_test  = np.mean(y_test  != tree_test)

print("decision tree")
print (1-err_tree_train, 1-err_tree_test)
```

```
decision tree
0.9985019077268791 0.9981859145039061
```

```
[ ] svm = SVC(kernel='poly',probability=True)
    svm.fit(X_train, y_train)

    svm_start_time = time.time()
    svm_yhat = svm.predict(X_test)
    svm_end_time = time.time()

    svm_train_predict = svm.predict(X_train)
    svm_test_predict = svm.predict(X_test)

    err_svm_train = np.mean(y_train != svm_train_predict)
    err_svm_test  = np.mean(y_test  != svm_test_predict)
    print("svm")
    print (1-err_svm_train, 1-err_svm_test)
```

```
svm
0.9984082769598089 0.9982210258360886
```

```
[ ] clf1 = LogisticRegressionCV()
    clf2 = RandomForestClassifier(n_estimators=50, random_state=1)

    voting = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2)], voting='soft')
    voting.fit(X_train, y_train)

    voting_start_time = time.time()
    voting_yhat = voting.predict(X_test)
    voting_end_time = time.time()

    voting_train_predict = voting.predict(X_train)
    voting_test_predict = voting.predict(X_test)

    err_voting_train = np.mean(y_train != voting_train_predict)
    err_voting_test  = np.mean(y_test  != voting_test_predict)
    print("log reg + random forest")
    print (1-err_voting_train, 1-err_voting_test)
```

```
log reg + random forest
0.9984082769598089 0.9982210258360886
```

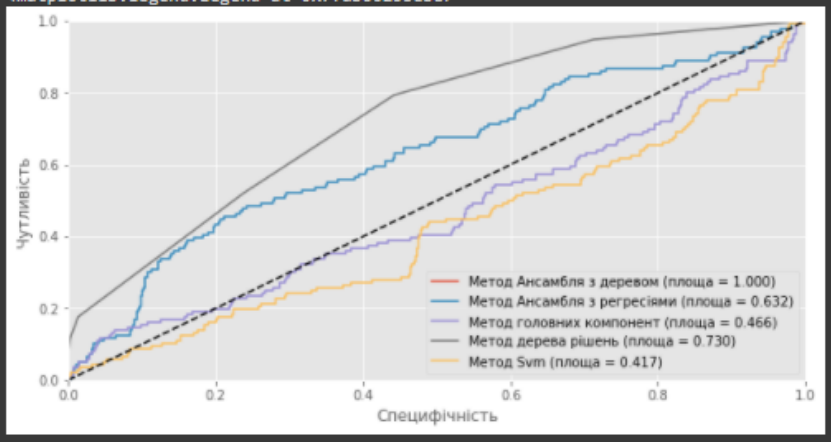
```
[ ] clf11 = LogisticRegressionCV()
    clf22 = LogisticRegressionCV(solver="liblinear", max_iter=30000000)

    voting1 = VotingClassifier(estimators=[('lr', clf11), ('rf', clf22)], voting='soft')
    voting1.fit(X_train, y_train)

    voting1_start_time = time.time()
    voting1_yhat = voting1.predict(X_test)
    voting1_end_time = time.time()
```

```
[ ]  from sklearn.tree import plot_tree
     plt.figure(figsize=(18,18))
     plot_tree(clf, filled=True)
     plt.show()
```

```
[ ]  X=np.array(X_train)
     Y=np.array(y_train)

[ ]  fprPCA, tprPCA, thresholds = roc_curve(y_train, logis.predict_proba(X_train)[:, 1])
     roc_aucPCA = auc(fprPCA, tprPCA)

[ ]  fprTree, tprTree, thresholdsTree = roc_curve(y_train, tree_model.predict_proba(X_train)[:, 1])
     roc_aucTree = auc(fprTree, tprTree)

[ ]  fprSvm, tprSvm, thresholdsSvm = roc_curve(y_train, svm.predict_proba(X_train)[:, 1])
     roc_aucSvm = auc(fprSvm, tprSvm)

[ ]  fprVoting, tprVoting, thresholdsVoting = roc_curve(y_train, voting.predict_proba(X_train)[:, 1])
     roc_aucVoting = auc(fprVoting, tprVoting)

[ ]  fprVoting1, tprVoting1, thresholdsVoting1 = roc_curve(y_train, voting1.predict_proba(X_train)[:, 1])
     roc_aucVoting1 = auc(fprVoting1, tprVoting1)
```

```
plt.subplots(figsize = (10,5))

plt.plot(fprVoting, tprVoting, label='Метод Ансамбля з деревом (площа = %0.3f)' % roc_aucVoting)
plt.plot(fprVoting1, tprVoting1, label='Метод Ансамбля з регресіями (площа = %0.3f)' % roc_aucVoting1)
plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)
plt.plot(fprTree, tprTree, label='Метод дерева рішень (площа = %0.3f)' % roc_aucTree)
plt.plot(fprSvm, tprSvm, label='Метод Svm (площа = %0.3f)' % roc_aucSvm)
plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Специфічність')
plt.ylabel('Чутливість')
plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x7fab60253eb8>
```



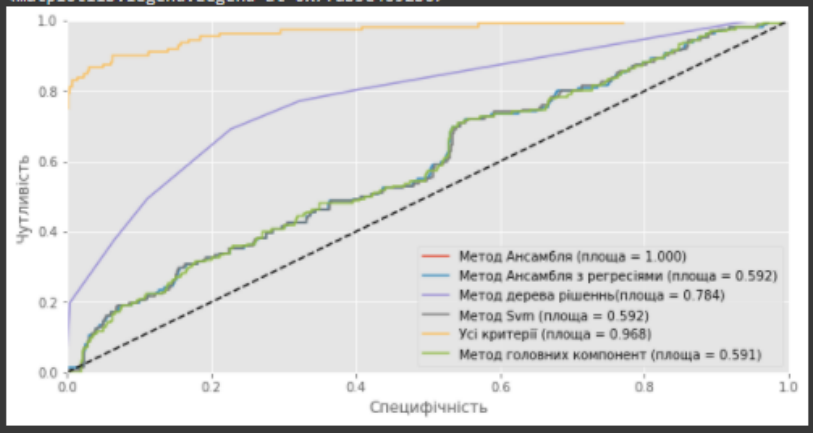

```
X = data.drop(('class'), axis=1)
y = data['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, train_size = 0.2, random_state = 11)

logis_all_start_time = time.time()
res = logis.fit(X_train, y_train)
logis_all_end_time = time.time()

y_train_predict = logis.predict(X_train)
y_test_predict = logis.predict(X_test)

y_test_predict1 = logis.predict_proba(X_test)
print(np.concatenate((np.transpose([y_test_predict[:10]]), y_test_predict1[:10],np.transpose([np.array(y_test[:10])])),axis=1))
```

```
err_train = np.mean(y_train != y_train_predict)
err_test  = np.mean(y_test  != y_test_predict)
print('all  Train: ', 1-err_train,'Test: ', 1-err_test)
```

```
all  Train:  0.999280209265989 Test:  0.9990519995786665
```

```
fprAll, tprAll, thresholds = roc_curve(y_train, logis.predict_proba(X_train)[:, 1])
roc_aucAll = auc(fprAll, tprAll)

plt.subplots(figsize = (10,5))
plt.plot(fprVoting, tprVoting, label='Метод Ансамбля (площа = %0.3f)' % roc_aucVoting)
plt.plot(fprVoting1, tprVoting1, label='Метод Ансамбля з регресіями (площа = %0.3f)' % roc_aucVoting1)
plt.plot(fprTree, tprTree, label='Метод дерева рішенньь(площа = %0.3f)' % roc_aucTree)
plt.plot(fprSvm, tprSvm, label='Метод Svm (площа = %0.3f)' % roc_aucSvm)
plt.plot(fprAll, tprAll, label='Усі критерії (площа = %0.3f)' % roc_aucAll)
plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)

plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Специфічність')
plt.ylabel('Чутливість')
plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x7fab5e4692b0>
```




```
res = logis.fit(X_train, y_train)
y_train_predict = logis.predict(X_train)
y_test_predict = logis.predict(X_test)
```

```
logis_2main_filter_start_time = time.time()

columns = np.array(X_train.columns)
finalErrTrain = 0.5
for i in range (X_train.shape[1]):
  for j in range(i+1,X_train.shape[1]):
        res = logis.fit(X_train[columns[[i,j]]], y_train)
        y_train_predict = logis.predict(X_train[columns[[i,j]]])
        err_train = np.mean(y_train != y_train_predict)
        print(err_train)
        if finalErrTrain>err_train :
          cTemp = X_train[columns[[i,j]]]
          finalI = i
          finalJ = j
          finalErrTrain = err_train

logis_2main_filter_end_time = time.time()
```

```
[ ] usedCriterias=[]
    usedCriterias = np.append(usedCriterias,columns[[finalI,finalJ]])
    removedColumns = np.delete(columns,[finalI,finalJ])
    print(usedCriterias)
    print(removedColumns)
```

```
['A13' 'A15']
['A1' 'A2' 'A3' 'A4' 'A5' 'A6' 'A7' 'A8' 'A9' 'A10' 'A11' 'A12' 'A14'
 'A16' 'A17' 'A18' 'A19' 'A20' 'A21' 'A22' 'A23' 'A24' 'A25' 'A26' 'A27'
 'A28' 'A29' 'A30']
```

```
[ ] logis_main2_start_time = time.time()
    res = logis.fit(X_train[usedCriterias], y_train)
    logis_main2_end_time = time.time()

    y_test_predict = logis.predict(X_test[usedCriterias])
    err_test  = np.mean(y_test  != y_test_predict)
    print('two main Train: ', 1-finalErrTrain,'Test: ', 1-err_test)

    y_test_predict1 = logis.predict_proba(X_test[usedCriterias])
    print(np.concatenate((np.transpose([y_test_predict[:10]]), y_test_predict1[:10],np.transpose([np.array(y_test[:10])])),axis=1))
```

```
two main Train:  0.9992275416513052 Test:  0.9988237772550121
[[0.00000000e+00 9.99748224e-01 2.51776207e-04 0.00000000e+00]
 [0.00000000e+00 9.99733659e-01 2.66341427e-04 0.00000000e+00]
 [0.00000000e+00 9.99693045e-01 3.06955408e-04 0.00000000e+00]
 [0.00000000e+00 9.99613826e-01 3.86174348e-04 0.00000000e+00]
 [0.00000000e+00 9.99814905e-01 1.85095440e-04 0.00000000e+00]
 [0.00000000e+00 9.99550308e-01 4.49692185e-04 0.00000000e+00]
 [0.00000000e+00 9.99805075e-01 1.94925379e-04 0.00000000e+00]
 [0.00000000e+00 9.99364435e-01 6.35565067e-04 0.00000000e+00]
 [0.00000000e+00 9.99862534e-01 1.37466235e-04 0.00000000e+00]
 [0.00000000e+00 9.96054522e-01 3.94547827e-03 0.00000000e+00]]
```

```
[ ] logis_step_filter_start_time = time.time()

    for j in range (40):
      b = cTemp
      updated = 0
      for i in range (removedColumns.shape[0]):
        a = np.array(X_train[removedColumns[[i]]])
        c = np.concatenate((b,a),axis = 1)
        res = logis.fit(c, y_train)
        y_train_predict = logis.predict(c)
        err_train = np.mean(y_train != y_train_predict)
        if finalErrTrain>err_train :
          finalI = i
          finalErrTrain = err_train
          cTemp = c
          updated = 1
      if updated == 0:
        break
      usedCriterias = np.append(usedCriterias,removedColumns[finalI])
      removedColumns = np.delete(removedColumns,finalI)

      logis_step_filter_end_time = time.time()
```

```
[ ] print(usedCriterias)
    print(removedColumns)
```

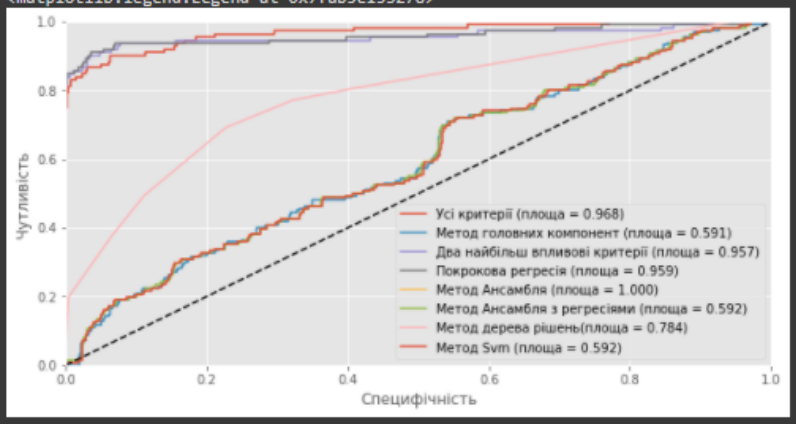
```
['A13' 'A15' 'A22' 'A18']
['A1' 'A2' 'A3' 'A4' 'A5' 'A6' 'A7' 'A8' 'A9' 'A10' 'A11' 'A12' 'A14'
 'A16' 'A17' 'A19' 'A20' 'A21' 'A23' 'A24' 'A25' 'A26' 'A27' 'A28' 'A29'
 'A30']
```

```
[ ] logis_step_start_time = time.time()
    res = logis.fit(X_train[usedCriterias], y_train)
    logis_step_end_time = time.time()

    y_train_predict = logis.predict(X_train[usedCriterias])
    y_test_predict = logis.predict(X_test[usedCriterias])
    err_train = np.mean(y_train != y_train_predict)
    err_test  = np.mean(y_test  != y_test_predict)
    print(' most valued Train: ', 1-err_train,'Test: ', 1-err_test)
    y_test_predict1 = logis.predict_proba(X_test[usedCriterias])

    print(np.concatenate((np.transpose([y_test_predict[:10]]), y_test_predict1[:10],np.transpose([np.array(y_test[:10])])),axis=1))
```

```
res = logis.fit(X_train[usedCriterias], y_train)
fprStep, tprStep, thresholds = roc_curve(y_train, logis.predict_proba(X_train[usedCriterias])[:, 1])
roc_aucStep = auc(fprStep, tprStep)

res = logis.fit(X_train[usedCriterias[[0,1]]], y_train)
fprTwoMain, tprTwoMain, thresholds = roc_curve(y_train, logis.predict_proba(X_train[usedCriterias[[0,1]]])[:, 1])
roc_aucTwoMain = auc(fprTwoMain, tprTwoMain)
```

```
[ ] plt.subplots(figsize = (10,5))

    plt.plot(fprAll, tprAll, label='Усі критерії (площа = %0.3f)' % roc_aucAll)
    plt.plot(fprPCA, tprPCA, label='Метод головних компонент (площа = %0.3f)' % roc_aucPCA)
    plt.plot(fprTwoMain, tprTwoMain, label='Два найбільш впливові критерії (площа = %0.3f)' % roc_aucTwoMain)
    plt.plot(fprStep, tprStep, label='Покрокова регресія (площа = %0.3f)' % roc_aucStep)
    plt.plot(fprVoting, tprVoting, label='Метод Ансамбля (площа = %0.3f)' % roc_aucVoting)
    plt.plot(fprVoting1, tprVoting1, label='Метод Ансамбля з регресіями (площа = %0.3f)' % roc_aucVoting1)
    plt.plot(fprTree, tprTree, label='Метод дерева рішень(площа = %0.3f)' % roc_aucTree)
    plt.plot(fprSvm, tprSvm, label='Метод Svm (площа = %0.3f)' % roc_aucSvm)


    plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('Специфічність')
    plt.ylabel('Чутливість')
    plt.legend(loc="lower right")
```

```
<matplotlib.legend.Legend at 0x7fab5e153278>
```



ВІДОМІСТЬ АТЕСТАЦІЙНОЇ РОБОТИ

Позначення	Найменування	Дод. відомості
	Текстові документи	
1	Пояснювальна записка	76 с.
2	Презентаційний матеріал	27 с.
	Інші документи	
3	Роздруківки програм	19 с.
4	Рецензія	2 с.
5	Відгук керівника	1 с.

					Застосування методів машинного навчання для оцінювання кредитних ризиків			
Змін	**Арк.**	**Номер докум.**	**Підп.**	**Дата**				
Розроб.		**Коновалов В.С.**			**(Тема роботи) Відомість атестаційної роботи**		**Аркуш**	**Аркушів**
Перевір.		**Гибкіна Н.В.**						
Н. контр.		**Сидоров М.В.**				**ХНУРЕ Кафедра ПМ**		
Затв.		**Тевяшев А.Д.**						