Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и технологий
Высшая школа программной инженерии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Классификация»

по дисциплине «Машинное обучение»

Выполнил
студент гр. 3530904/90102 Афанасьев Е. Д.

Руководитель
Старший преподаватель Селин И. А.
ВШПИ

Санкт-Петербург
2022

Оглавление

Задачи

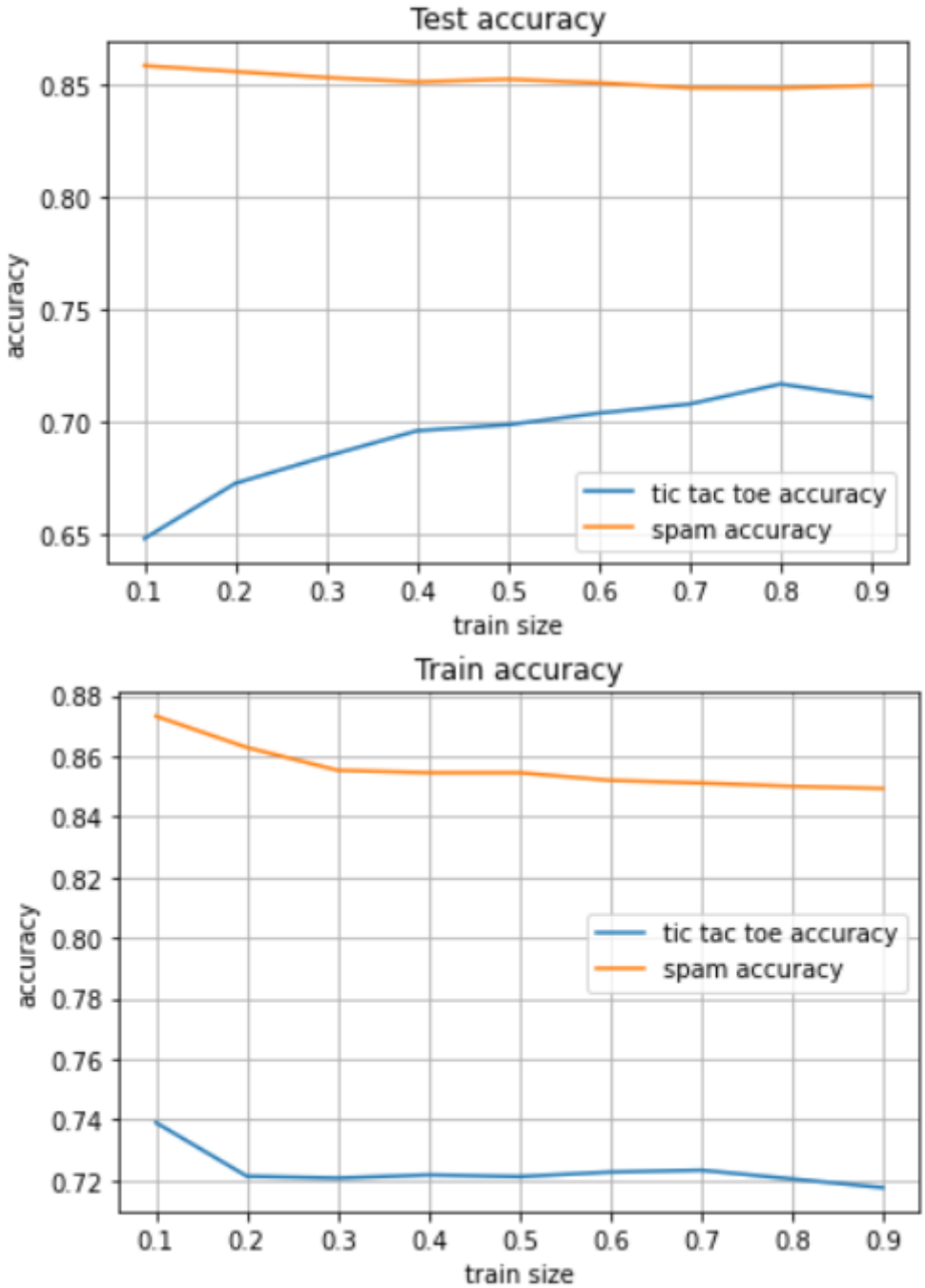
1. Исследуйте, как объем обучающей выборки и количество тестовых данных, влияет на точность классификации в датасетах про крестики-нолики (tic_tac_toe.txt) и о спаме e-mail сообщений (spam.csv) с помощью наивного Байесовского классификатора. Постройте графики зависимостей точности на обучающей и тестовой выборках в зависимости от их соотношения.
2. Сгенерируйте 100 точек с двумя признаками X1 и X2 в соответствии с нормальным распределением так, что одна и вторая часть точек (класс -1 и класс 1) имеют параметры: мат. ожидание X1, мат. ожидание X2, среднеквадратические отклонения для обеих переменных, соответствующие вашему варианту (указан в таблице). Построить диаграммы, иллюстрирующие данные. Построить Байесовский классификатор и оценить качество классификации с помощью различных методов (точность, матрица ошибок, ROC и PR-кривые). Является ли построенный классификатор «хорошим»?
3. Постройте классификатор на основе метода k ближайших соседей для обучающего множества Glass (glass.csv). Посмотрите заголовки признаков и классов. Перед построением классификатора необходимо также удалить первый признак Id number, который не несет никакой информационной нагрузки.
a. Постройте графики зависимости ошибки классификации от количества ближайших соседей.
b. Определите подходящие метрики расстояния и исследуйте, как тип метрики расстояния влияет на точность классификации.
c. Определите, к какому типу стекла относится экземпляр с характеристиками:
RI =1.516 Na =11.7 Mg =1.01 Al =1.19 Si =72.59 K=0.43 Ca =11.44 Ba =0.02 Fe =0.1
4. Постройте классификаторы на основе метода опорных векторов для наборов данных из файлов svmdataN.txt и svmdataNtest.txt, где N – индекс задания:
a. Постройте алгоритм метода опорных векторов с линейным ядром. Визуализируйте разбиение пространства признаков на области с помощью полученной модели (пример визуализации). Выведите количество полученных опорных векторов, а также матрицу ошибок классификации на обучающей и тестовой выборках.
b. Постройте алгоритм метода опорных векторов с линейным ядром. Добейтесь нулевой ошибки сначала на обучающей выборке, а затем на тестовой, путем изменения штрафного параметра. Выберите оптимальное значение данного параметра и объясните свой выбор. Всегда ли нужно добиваться минимизации ошибки на обучающей выборке?
c. Постройте алгоритм метода опорных векторов, используя различные ядра (линейное, полиномиальное степеней 1-5, сигмоидальная функция, гауссово). Визуализируйте разбиение пространства признаков на области с помощью полученных моделей. Сделайте выводы.
d. Постройте алгоритм метода опорных векторов, используя различные ядра (полиномиальное степеней 1-5, сигмоидальная функция, гауссово). Визуализируйте

разбиение пространства признаков на области с помощью полученных моделей. Сделайте выводы.
e. Постройте алгоритм метода опорных векторов, используя различные ядра (полиномиальное степеней 1-5, сигмоидальная функция, гауссово). Изменяя значение параметра ядра (гамма), продемонстрируйте эффект переобучения, выполните при этом визуализацию разбиения пространства признаков на области.
5. Постройте классификаторы для различных данных на основе деревьев решений:
a. Загрузите набор данных Glass из файла glass.csv.
Постройте дерево классификации для модели, предсказывающей тип (Type) по остальным признакам. Визуализируйте результирующее дерево решения. Дайте

интерпретацию полученным результатам. Является ли построенное дерево избыточным? Исследуйте зависимость точности классификации от критерия расщепления, максимальной глубины дерева и других параметров по вашему усмотрению.
b. Загрузите набор данных spam7 из файла spam7.csv. Постройте оптимальное, по вашему мнению, дерево классификации для параметра yesno. Объясните, как был осуществлён подбор параметров. Визуализируйте результирующее дерево решения. Определите наиболее влияющие признаки. Оцените качество классификации.
6. Загрузите набор данных из файла bank_scoring_train.csv. Это набор финансовых данных, характеризующий физических лиц. Целевым столбцом является «SeriousDlqin2yrs», означающий, ухудшится ли финансовая ситуация у клиента. Постройте систему по принятию решения о выдаче или невыдаче кредита физическому лицу. Сделайте как минимум 2 варианта системы на основе различных классификаторов. Подберите подходящую метрику качества работы системы исходя из специфики задачи и определите, принятие решения какой системой сработало лучше на bank_scoring_test.csv.

Исследование влияния соотношения обучающей и тестовой выборки на точность классификации

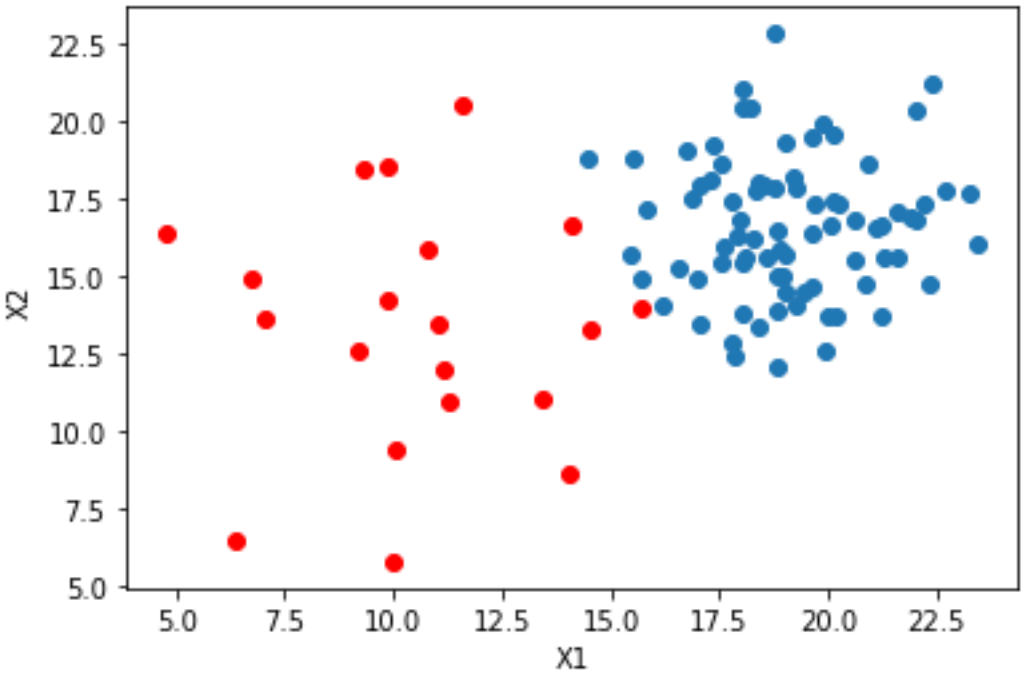
При построении классификаторов для набора данных крестики нолики и спам были использованы различные соотношения обучающих и тестовых данных от 0.1 до 0.9 с шагом в одну десятую.
Были получены два графика с точностью по тестовым и тренировочным данным.



На графике с тестовыми данными точность классификации спама практически не меняется от размера выборки, что нельзя сказать о крестиках ноликах.
На тренировочных данных изменения незначительны.
Можно сделать вывод, что эффективное соотношение тестовой и тренировочной части зависит от конкретных данных, особенно для точности на тестовых данных.

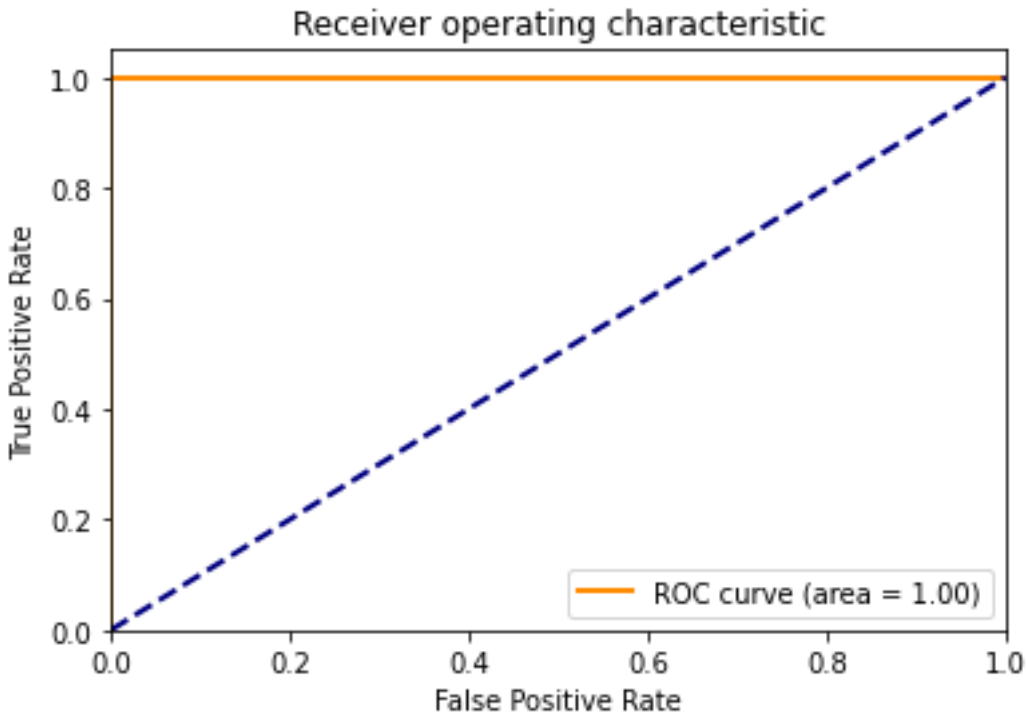
Оценка качества классификаторов с помощью различных методов

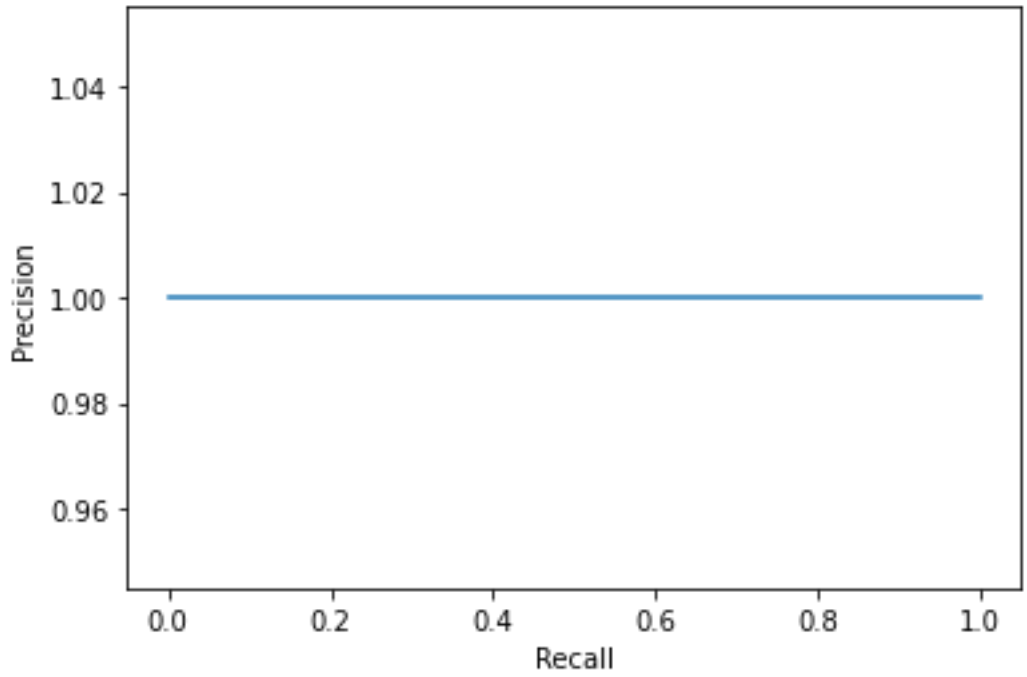
Был выполнен 2 вариант
Сгенерировано 20 точек класса -1 со стандартным отклонением 3, матожиданием 11 по X1 и матожиданием 13 по X2, и 80 точек класса 1 с дисперсией 4, матожиданием 19 для признака X1 и матожиданием 16 для X2
80 точек было выбрано для обучающей выборки и 20 для тестовой.
Вот визуализированные данные:



Был построен Байесовский классификатор.
Точность получилась 1.0
Матрица ошибок:

	0	1
0	3	0
1	0	17





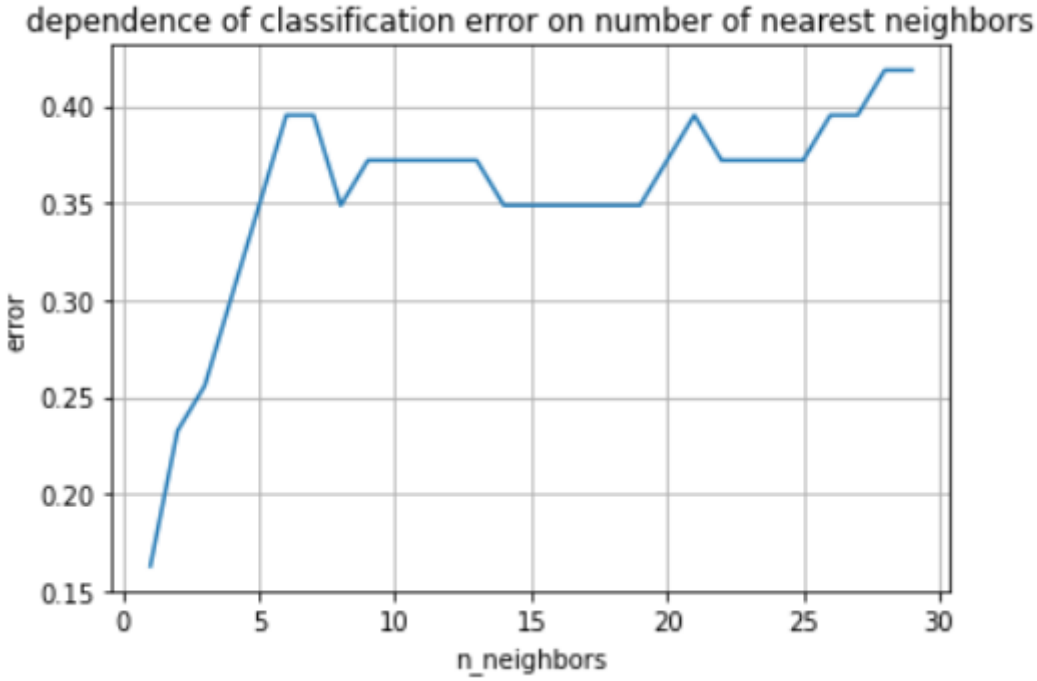
На представленных данных мы получили хорошие результаты, классификатор выдает хорошие значения на различных метриках, поскольку два класса отличаются друг от друга достаточно сильно.

KNN классификатор

В начале удалил ненужный столбец Id и посмотрел заголовки признаков и классов

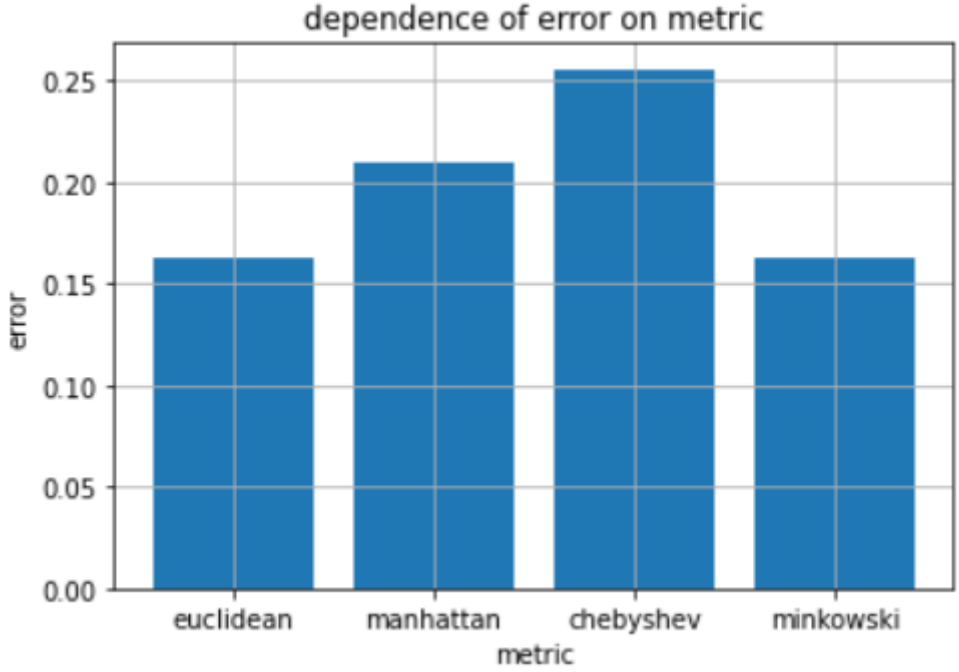
	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

Затем разбил данные на обучающую и тестовую часть, train_size=0.8
Далее построил график зависимости ошибки классификации от числа ближайших соседей



В данном случае наименьшая ошибка и, следовательно, лучший вариант – с одним соседом.

Затем сравнил ошибку для разных метрик расстояния

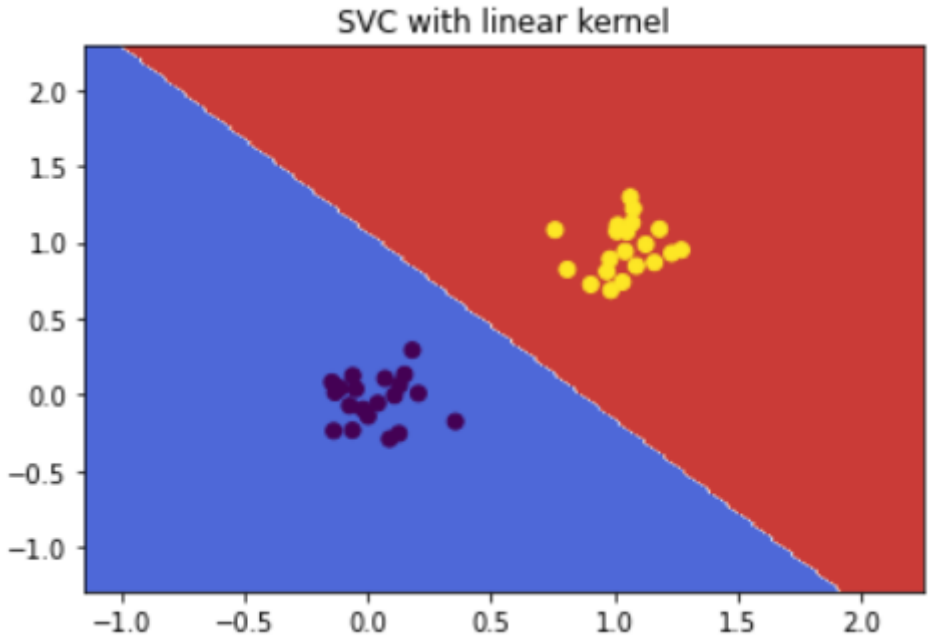


Наилучшей мерой расстояния является в данном случае мера Миньковского
Экземпляр с характеристиками RI =1.516 Na =11.7 Mg =1.01 Al =1.19 Si =72.59 K=0.43 Ca =11.44 Ba =0.02 Fe =0.1 принадлежит 5 типу стекла.

Метод опорных векторов

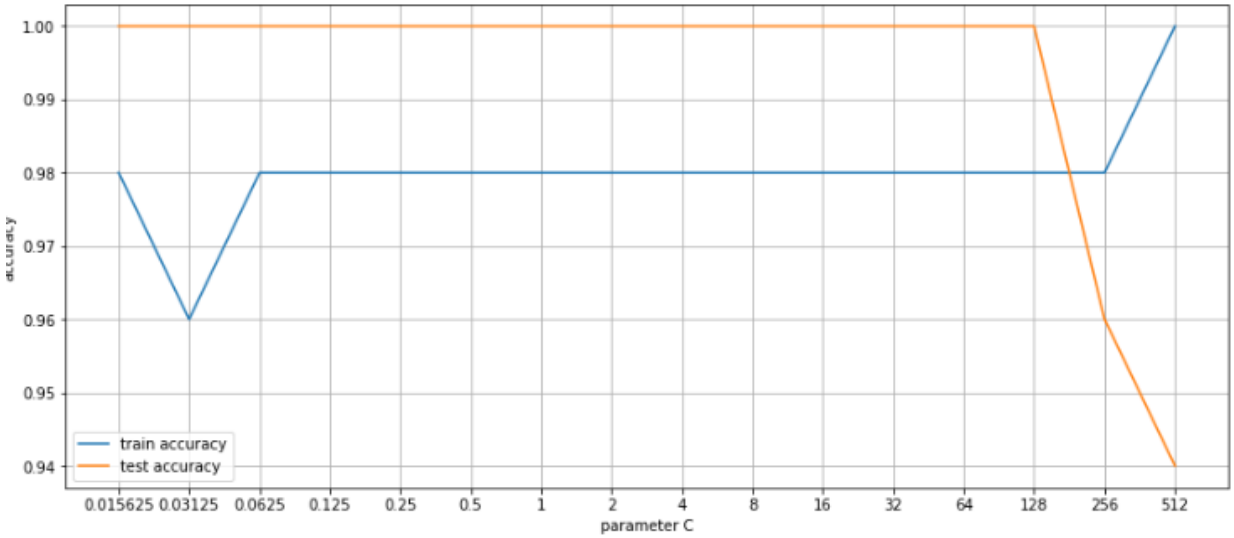
A) Был построен алгоритм на основе метода опорных векторов с линейным ядром, было получено по 3 опорных вектора для каждого класса, так же отсутствовали ошибки для тестовых и тренировочных данных.

```
Test accuracy  1.0
Train accuracy  1.0
Test confusion matrix
     0   1
0  20   0
1   0  20
Train confusion matrix
     0   1
0  20   0
1   0  20
Number of support vectors [3 3]
```



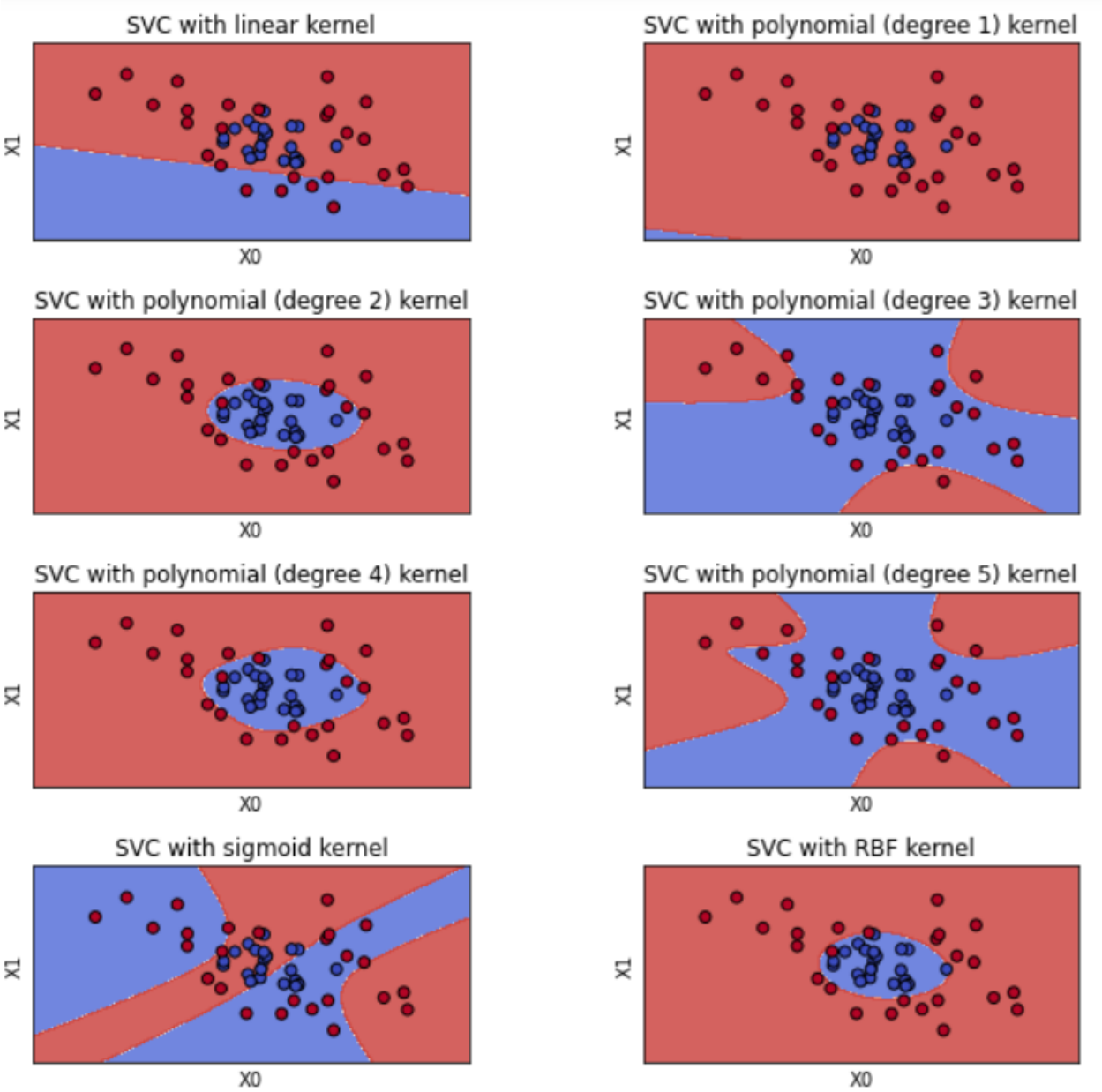
b) Для понимания какой параметр нужен для зануления ошибки на обучающей и тестовой выборке был нарисован следующий график из которого видно, что для тестовой выборки это 512, а для тестовой достаточно самого маленького значения 2**(-6). Также видно, что при достижении нулевой ошибки на обучающей выборке ухудшается точность на тестовой, то есть происходит переобучение. Не стоит ориентироваться в первую очередь

на метрики обучающей выборки.



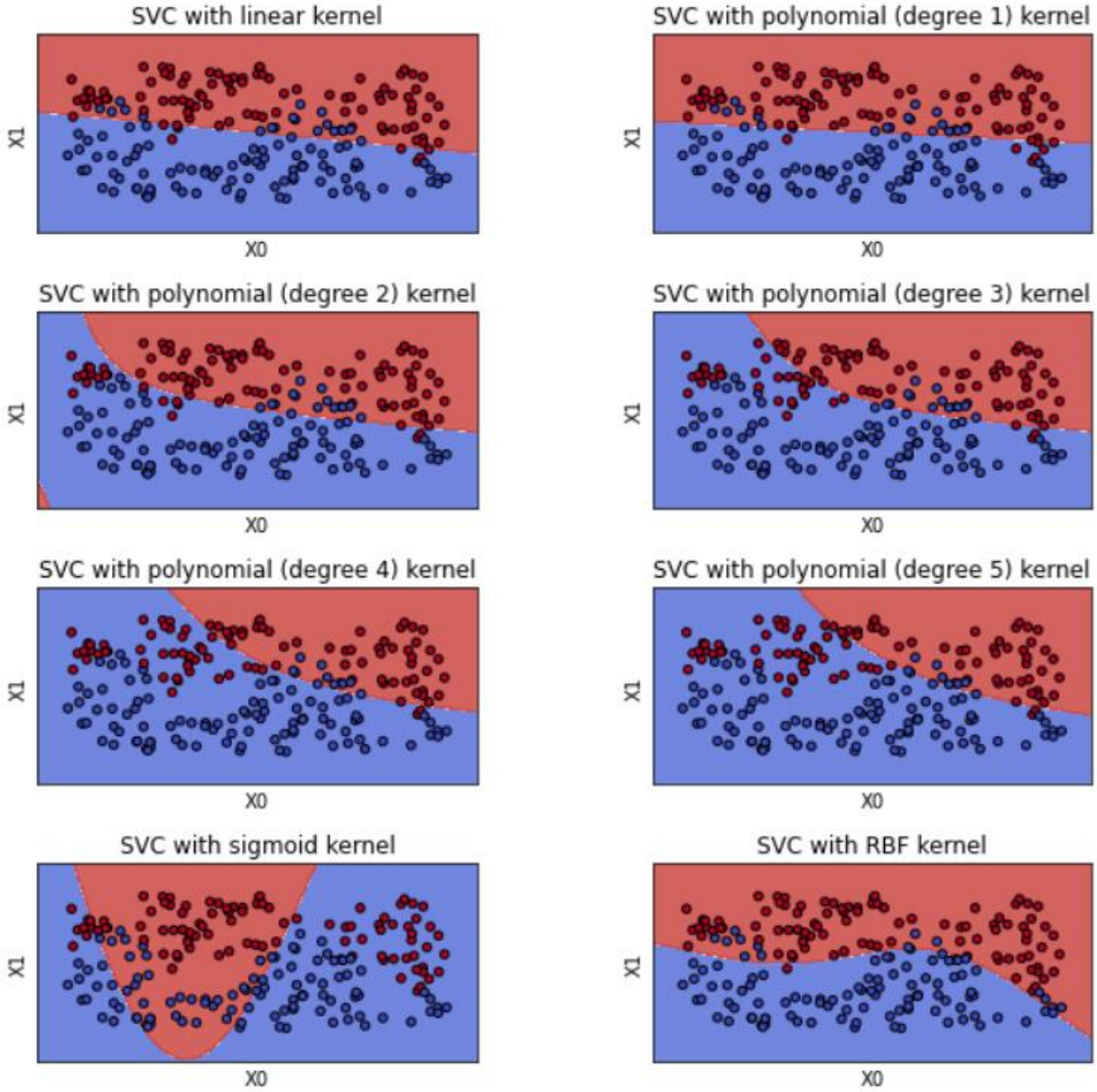
с) Построим и обучим модели с разными модификациями ядра и визуализируем полученные результаты. Хорошо справились с классификацией полиномиальные ядра с

четными степенями и гауссово ядро, остальные очень далеки от истины.

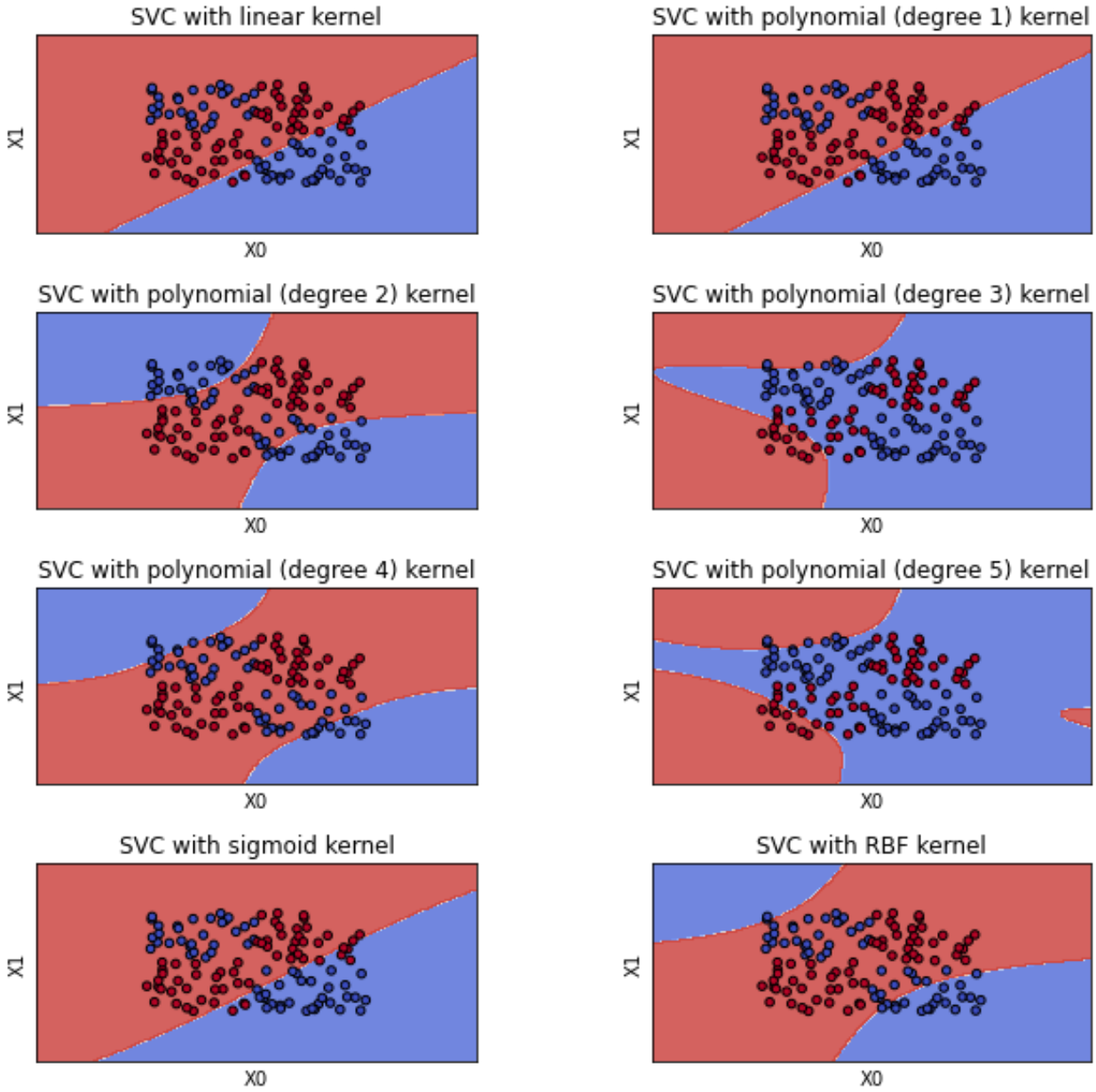


d) Поступим также с этим набором данных, обучим модели с разными ядрами. Хорошие результаты показали модели с линейным ядром, гауссовым и полиномиальный первой

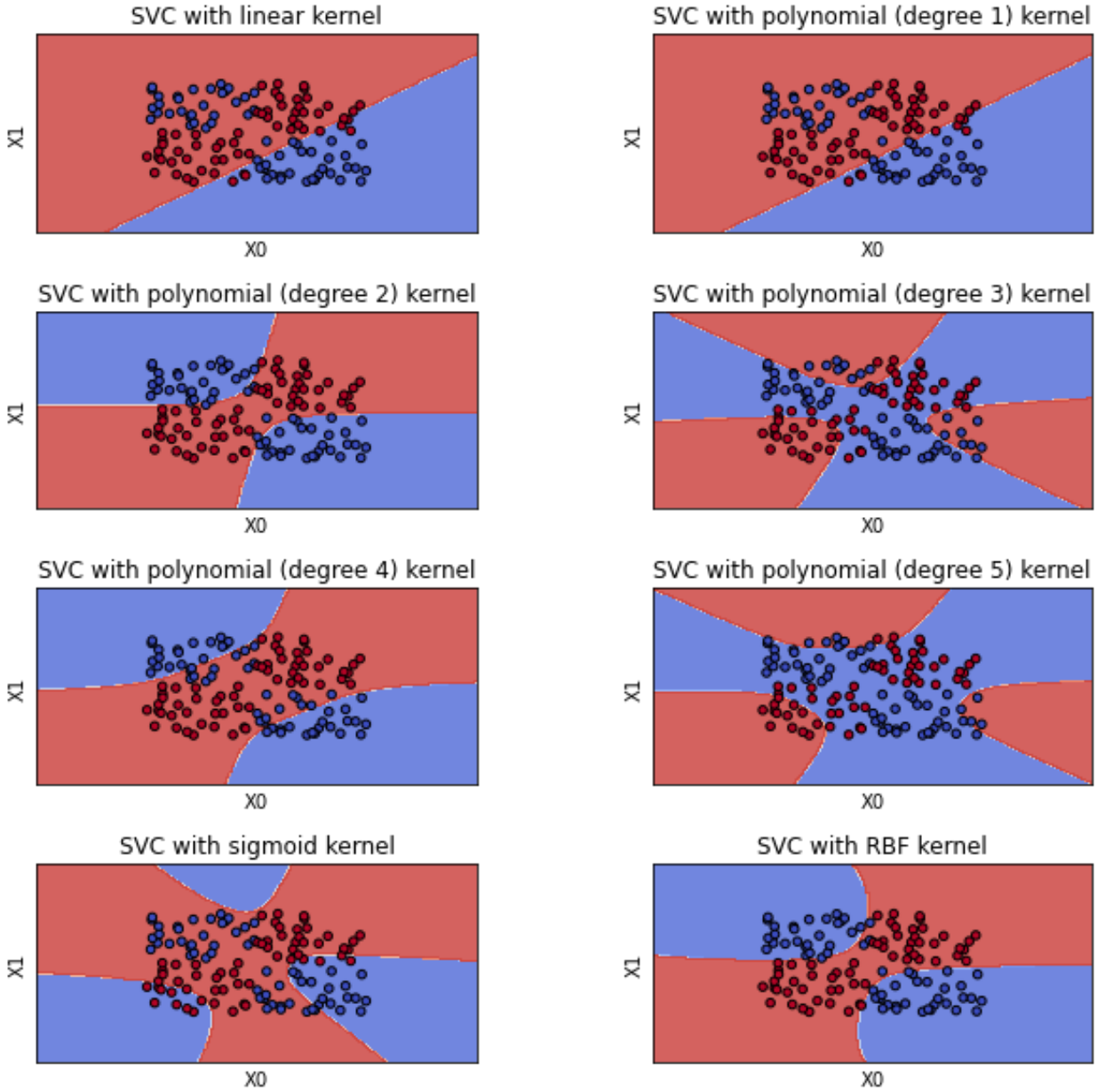
степени. Уостальных дела обстоят хуже.



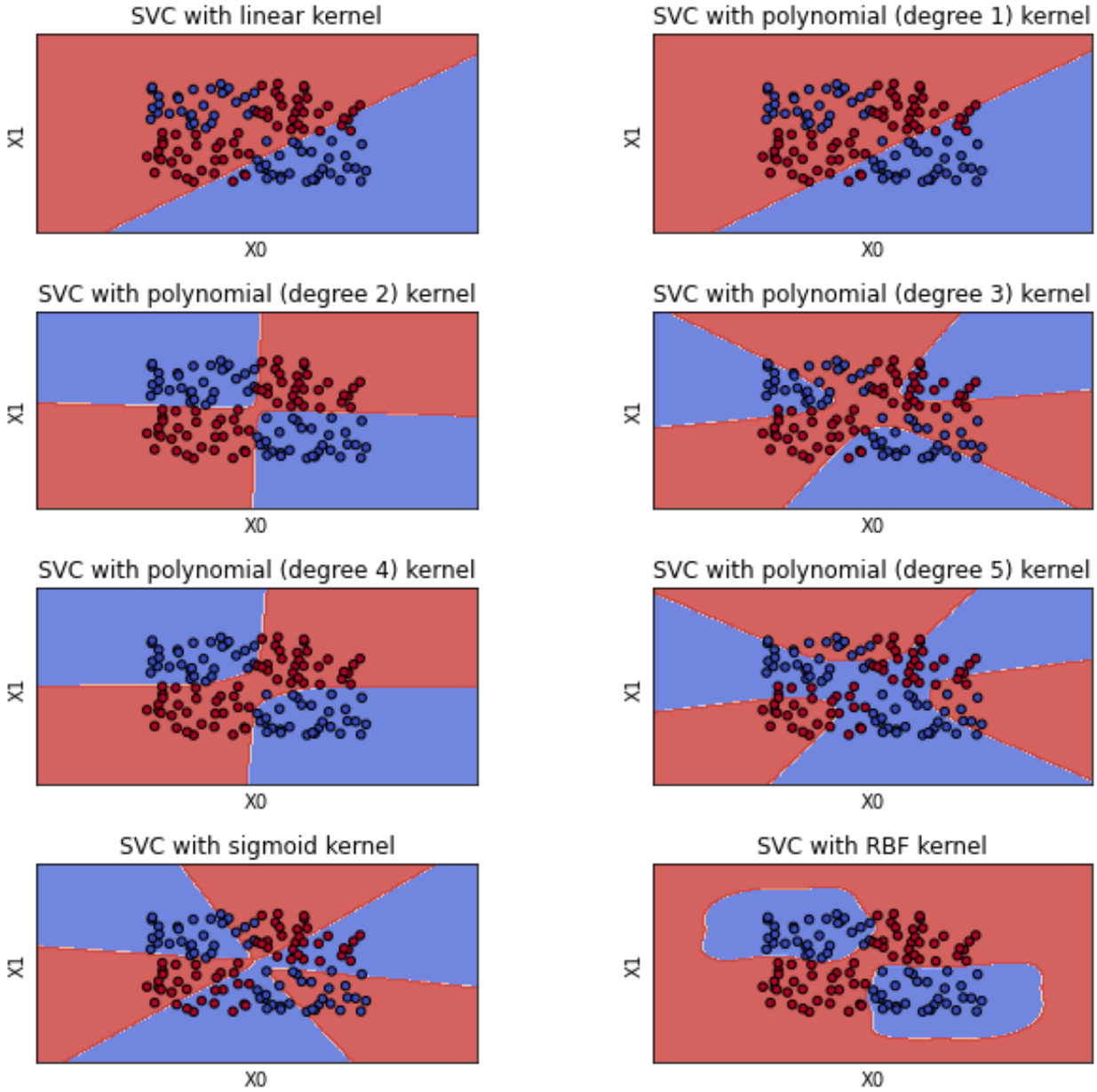
e) Построим те же модели с разными показателями гамма и попробуем продемонстрировать переобучение.
Гамма = 0.1


SVC with linear kernel
SVC with polynomial (degree 1) kernel
SVC with polynomial (degree 2) kernel
SVC with polynomial (degree 3) kernel
SVC with polynomial (degree 4) kernel
SVC with polynomial (degree 5) kernel
SVC with sigmoid kernel
SVC with RBF kernel
X0
X1

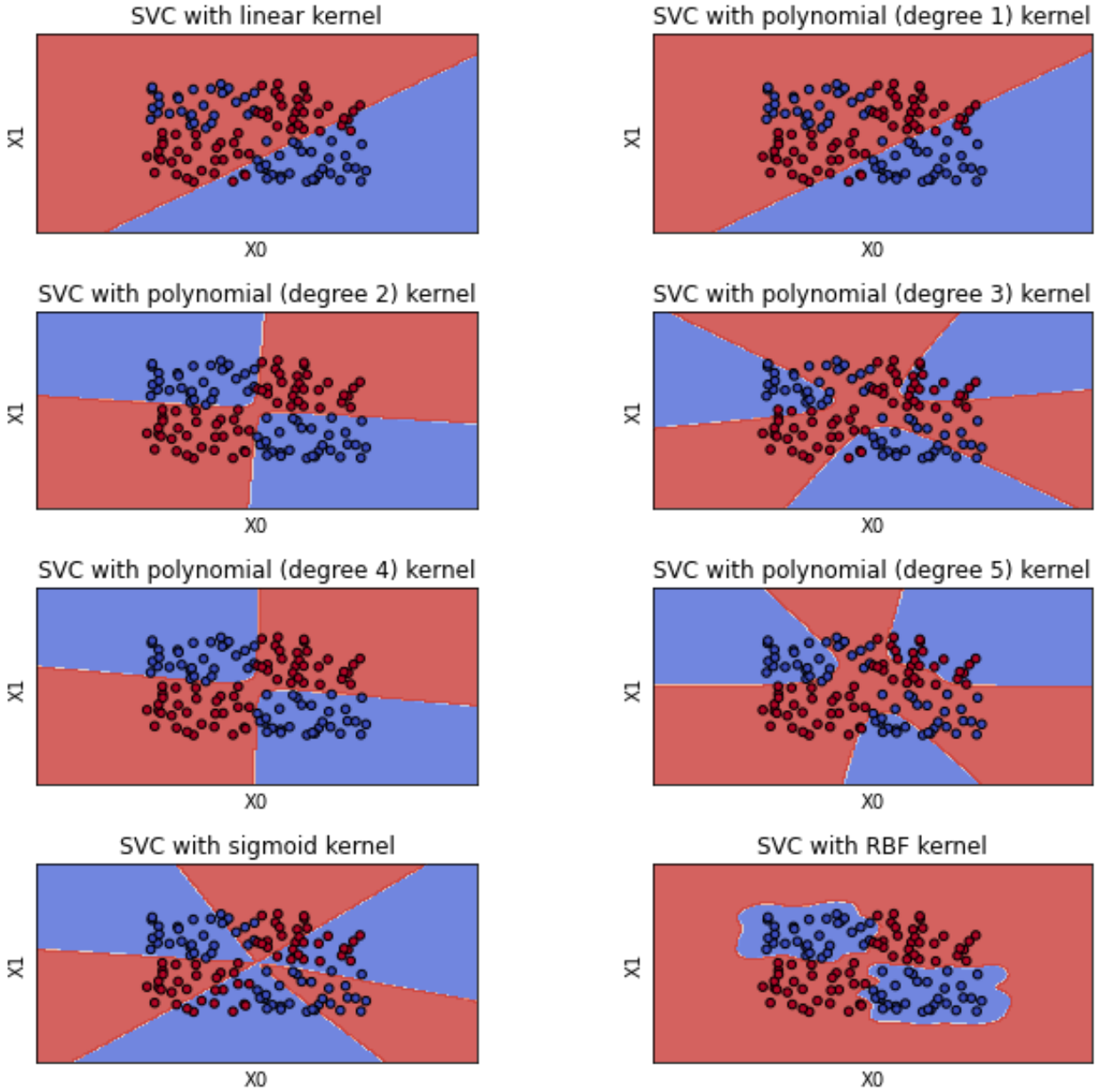
Гамма = 1


SVC with linear kernel
SVC with polynomial (degree 1) kernel
SVC with polynomial (degree 2) kernel
SVC with polynomial (degree 3) kernel
SVC with polynomial (degree 4) kernel
SVC with polynomial (degree 5) kernel
SVC with sigmoid kernel
SVC with RBF kernel
X1
X0

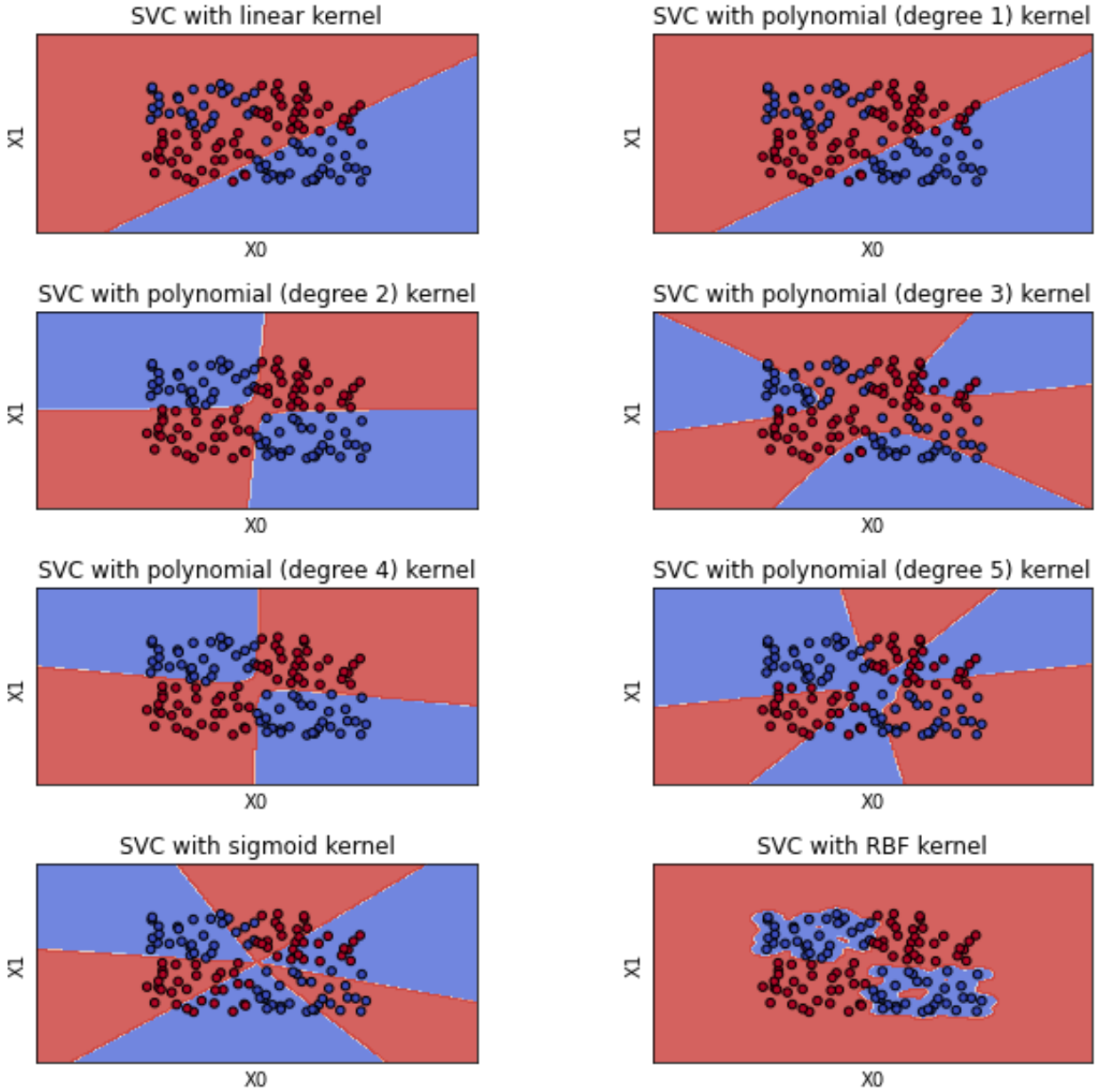
Гамма = 10


SVC with linear kernel
SVC with polynomial (degree 1) kernel
SVC with polynomial (degree 2) kernel
SVC with polynomial (degree 3) kernel
SVC with polynomial (degree 4) kernel
SVC with polynomial (degree 5) kernel
SVC with sigmoid kernel
SVC with RBF kernel
X1
X0

Гамма = 50


SVC with linear kernel
SVC with polynomial (degree 1) kernel
SVC with polynomial (degree 2) kernel
SVC with polynomial (degree 3) kernel
SVC with polynomial (degree 4) kernel
SVC with polynomial (degree 5) kernel
SVC with sigmoid kernel
SVC with RBF kernel
X1
X0

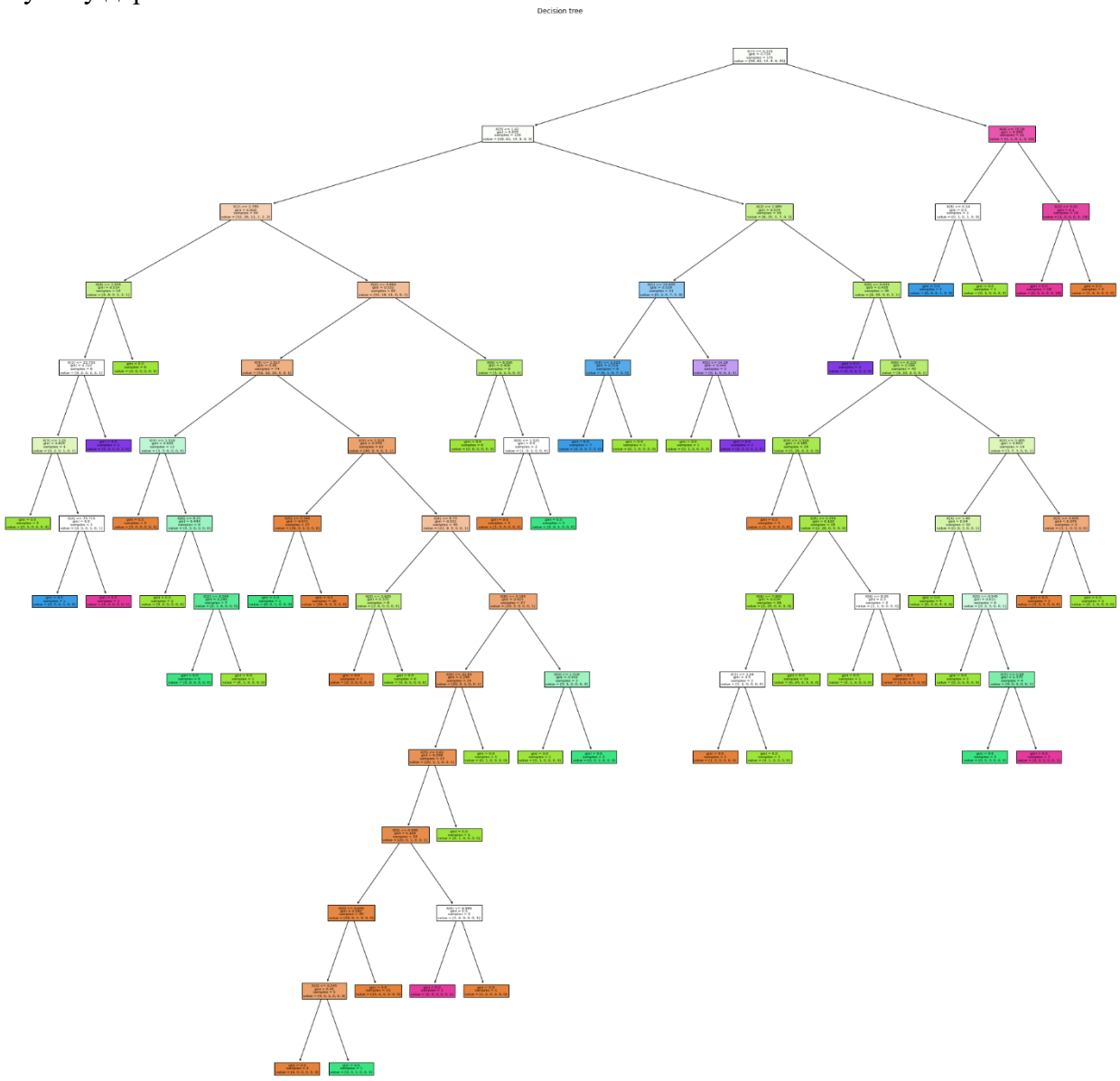
Гамма = 150



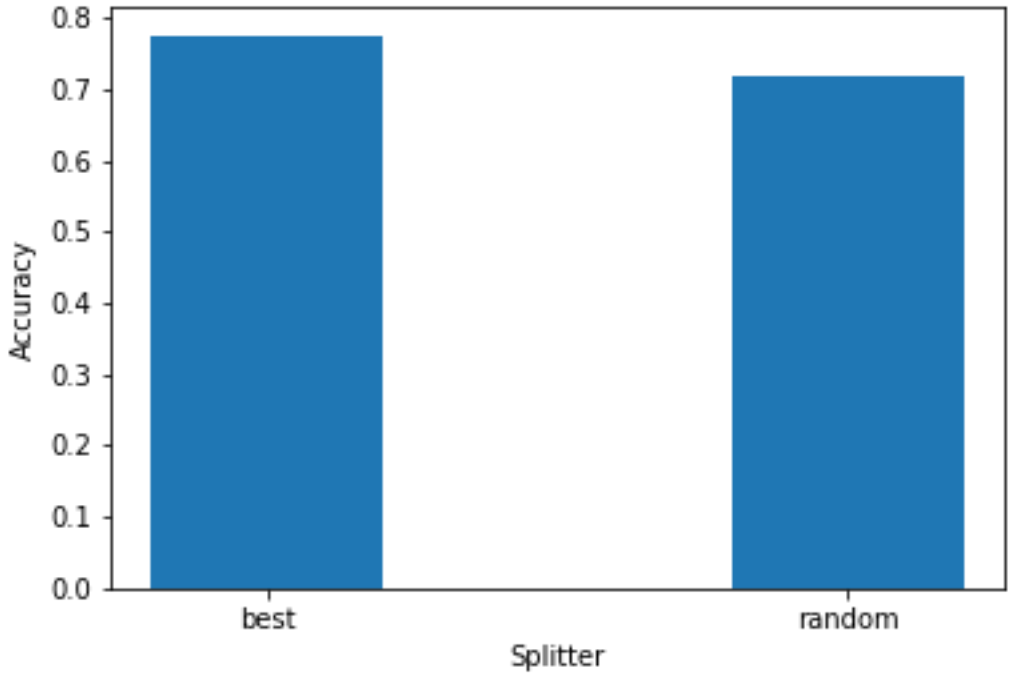
Можем наблюдать переобучение в последнем случае. С гаммой равной 150 гауссово ядро демонстрирует переобучение, появляются красные вкрапления в синих областях там, где их явно быть не должно.

Дерево решений

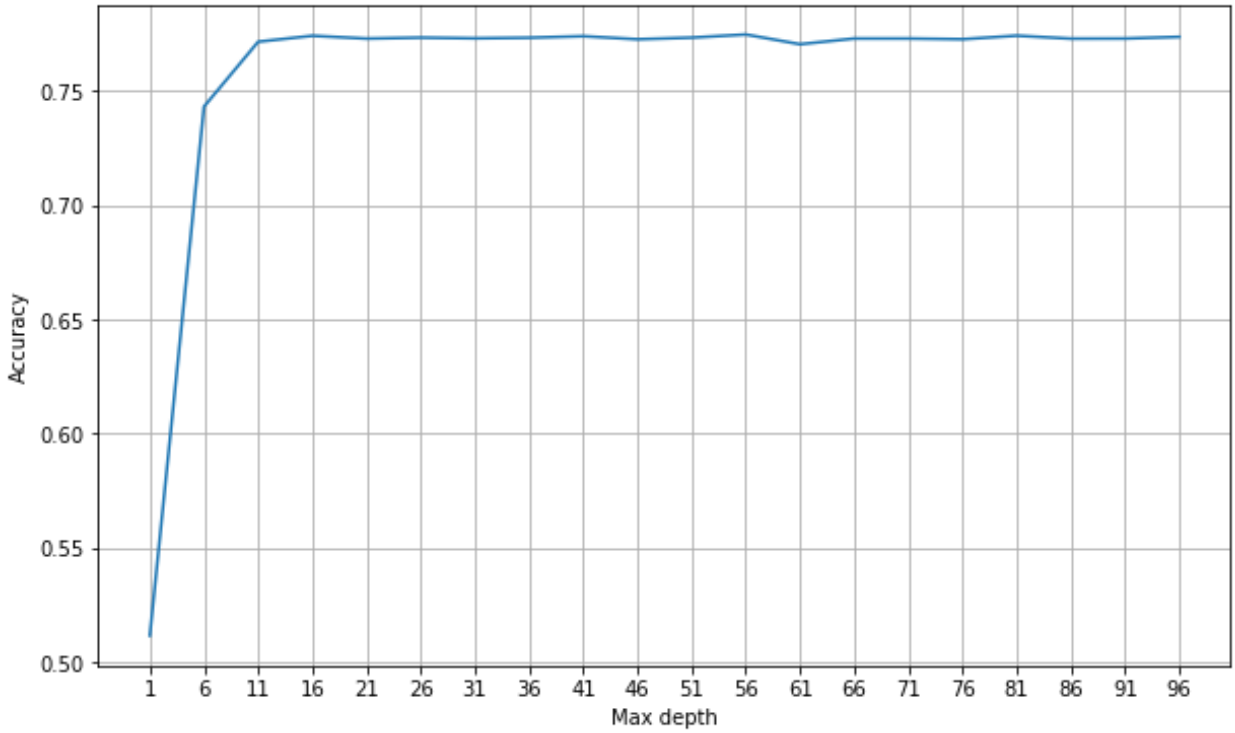
а) Построим дерево решений с параметрами по умолчанию. Получаем точность 0.767 и глубину дерева 13.



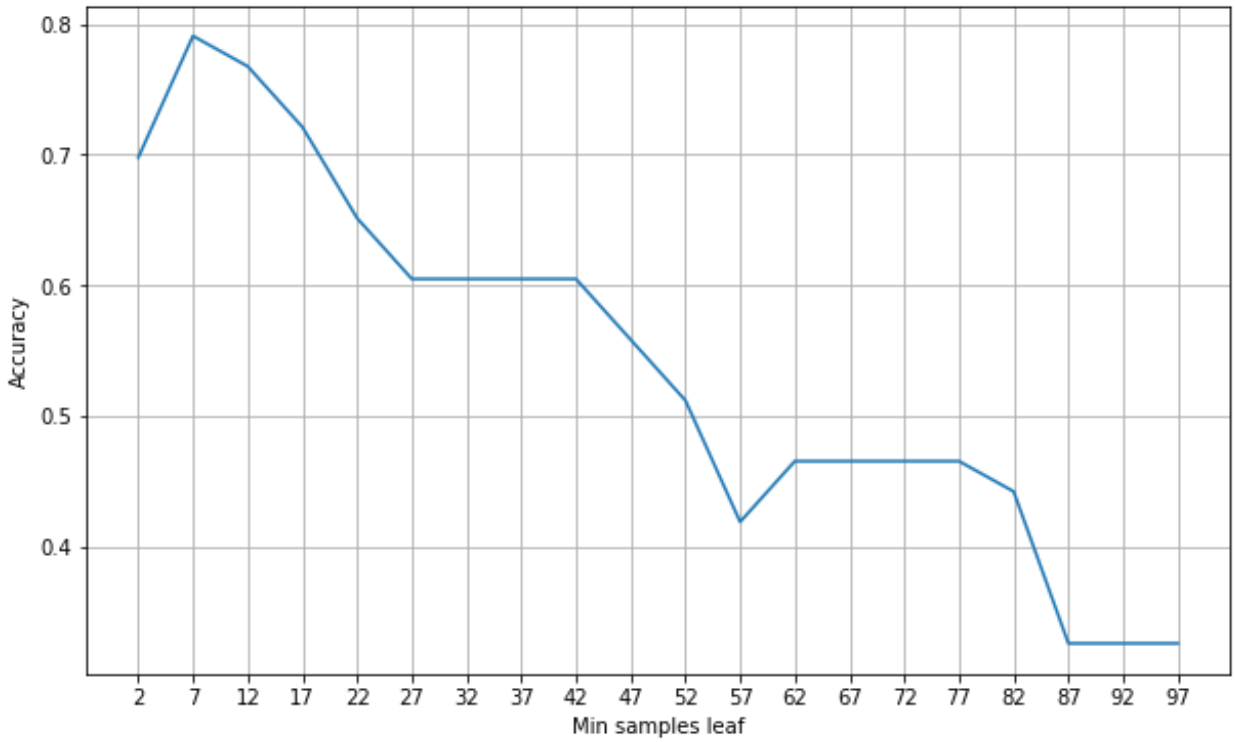
Затем посмотрим на влияние параметра splitter на точность. Я заметил, что от раза к разу значения точности сильно разнятся, поэтому я обучил по 1000 раз модель с каждым видом разделителя и взял среднюю точность. Также поступил и с остальными параметрами. ‘best’ показывает несколько лучшие результаты, чем ‘random’.



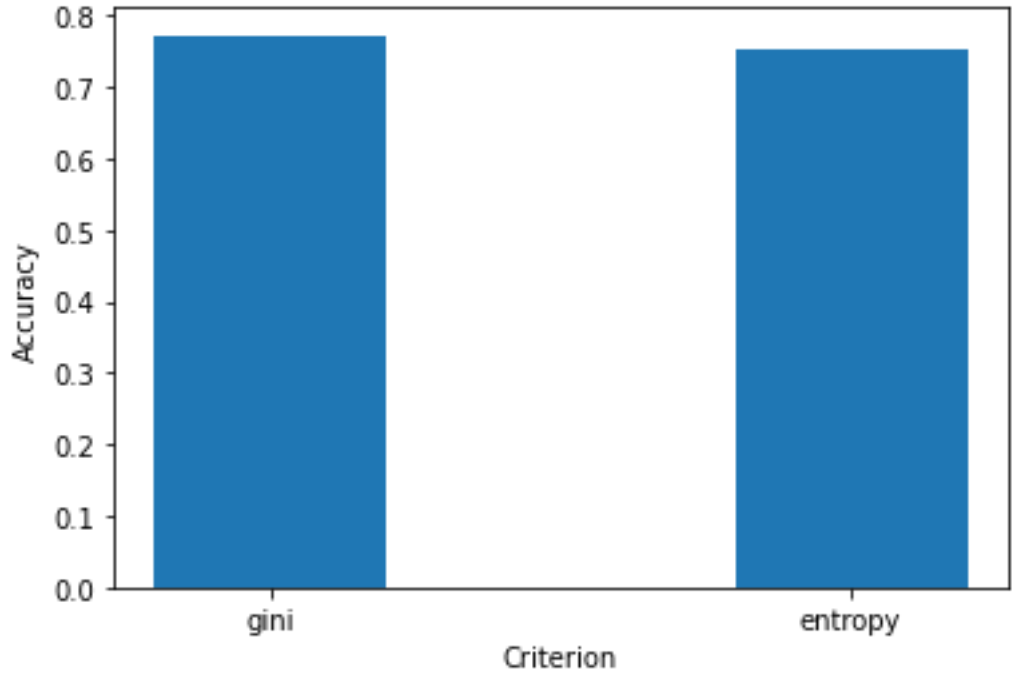
Начиная с определенной глубины, примерно 16, точность перестает расти.



С ростом числа минимального количества листьев уменьшается точность. Оптимальное значение в данном случае – это 7.



Критерий джини совсем немного лучше, чем критерий энтропии.

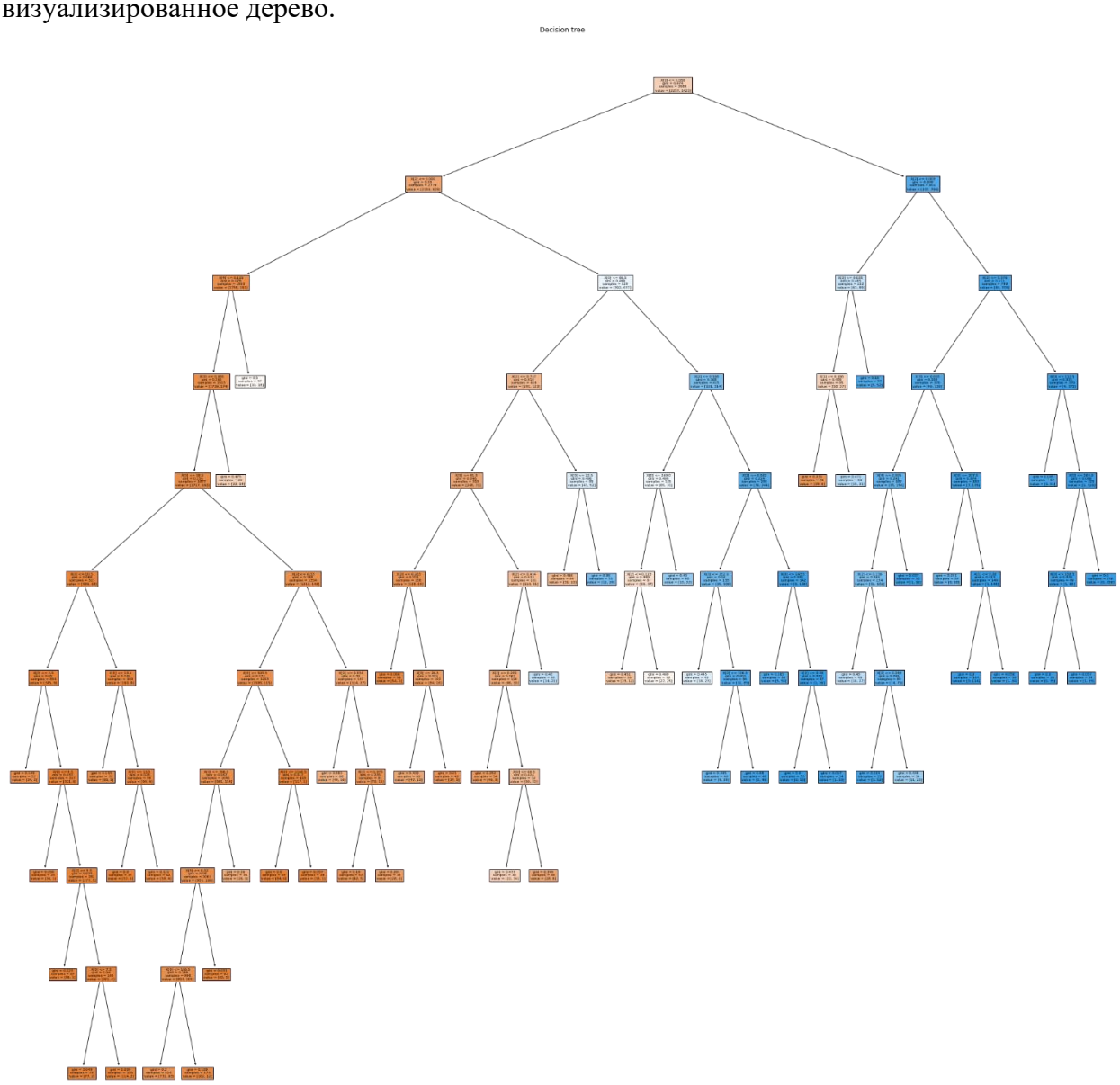


b) Для определения лучших параметров был использован GridSeacrCV, который перебирает все возможные варианты переданных заранее параметров, обучается на каждом с кросс-валидацией и выбирает лучший из них. Вот лучшие параметры

```
: 1 tree_grid.best_params_, tree_grid.best_score_, tree_grid.best_estimator_
```

```
: ({'criterion': 'gini',
   'max_depth': 10,
   'min_samples_leaf': 34,
   'splitter': 'best'},
  0.8678529953264409,
  DecisionTreeClassifier(max_depth=10, min_samples_leaf=34))
```

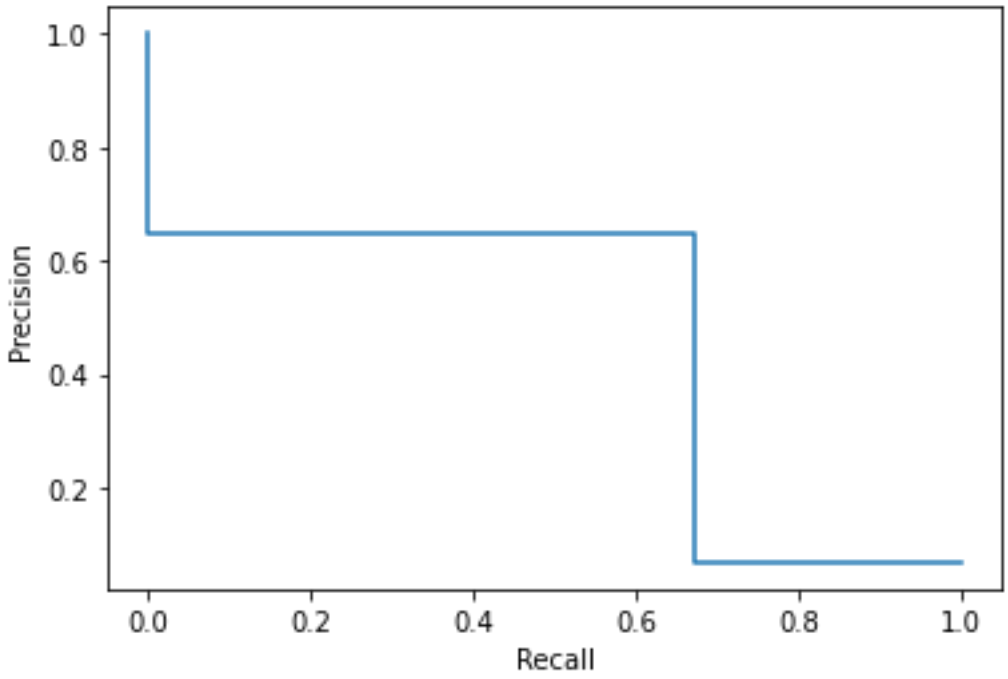
Построим классификатор с такими параметрами и получим точность 0.855, что почти на две сотых лучше точности с параметрами по умолчанию. Вот получившееся визуализированное дерево.



Классификация banking_score

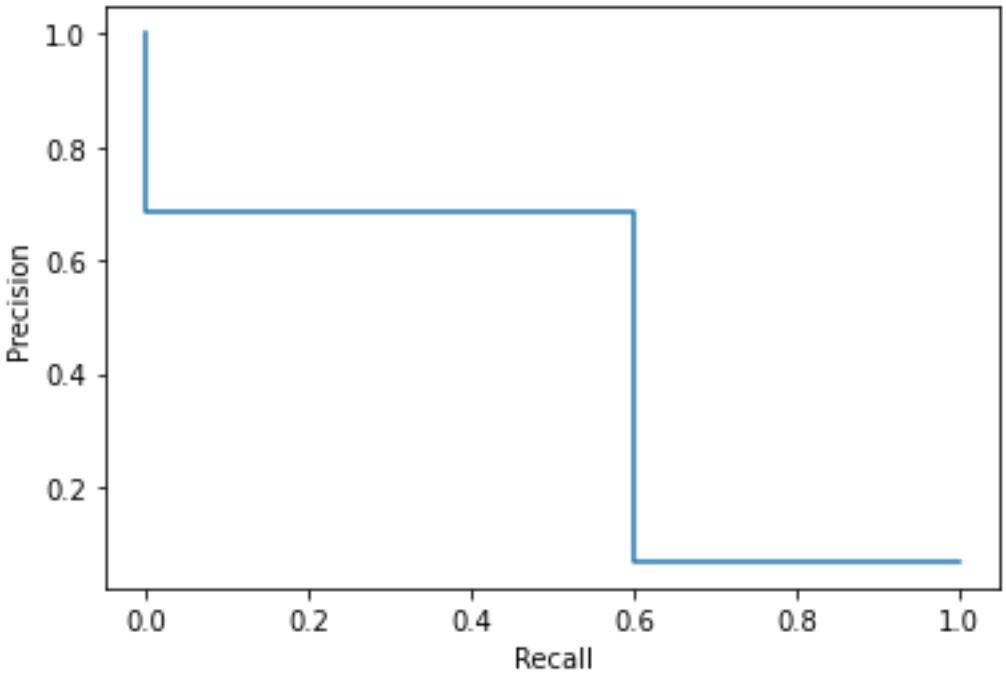
Я решил попробовать решить эту задачу с помощью трех различных методов: ближайших соседей, опорных векторов и дерева решений. Целевой столбец имеет явный дисбаланс классов, поэтому по метрике точность нельзя будет корректно оценить эффективность модели. Для этого я использую матрицу ошибок, f-меру и PR-кривую, поскольку они позволят понять насколько та или иная модель эффективна. Для проверки неправильности использования accuracy, будем смотреть на нее тоже.
Дерево решений выдает точность 0.952, f-меру 0.66, следующую матрицу ошибок и PR-кривую. Дерево решений дает неплохие результаты, судя по кривой и по матрице ошибок, лучше, чем наугад.

	0	1
0	21798	609
1	537	1109



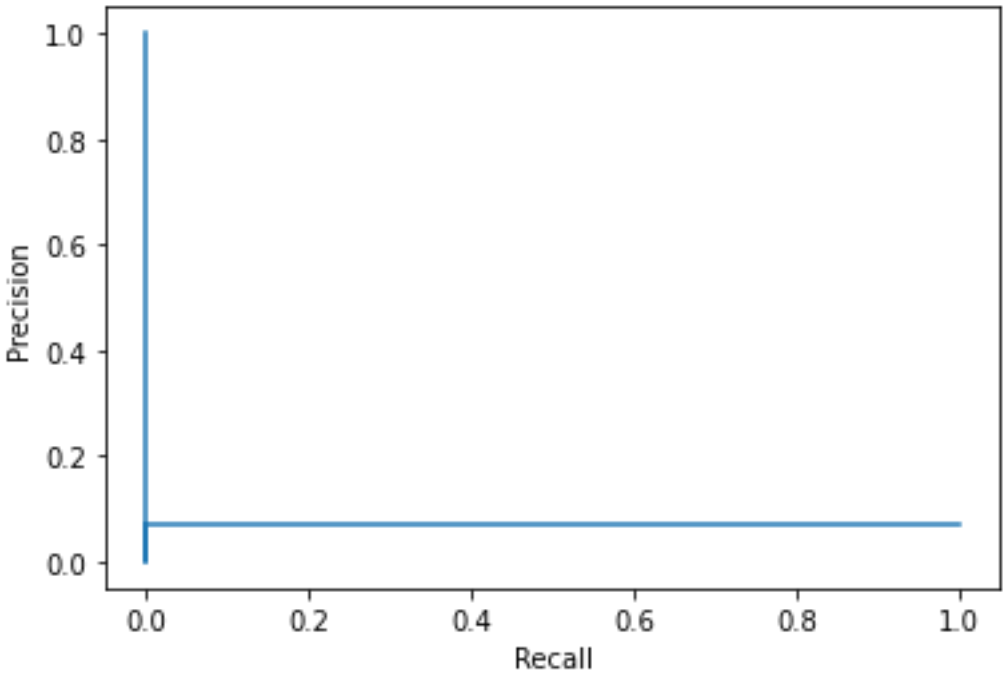
Метод ближайших соседей выдал лучший результат с параметром количества соседей равным одному. Точность получилась 0.953, f-меру 0.64. Решение получилось несколько хуже, чем у дерева решений, но все еще лучше случайного.

	0	1
0	21952	455
1	659	987



Метод опорных векторов показал, казалось бы, неплохую точность 0.93, однако f-мера 0.0 и в матрице ошибок видно, что модель просто разрешает всем получать кредит, что очень плохо. То же видно и на PR кривой.

	0	1
0	22406	1
1	1646	0



Таким образом, лучшая модель для данного случая – дерево решений, метод ближайших соседей оказался чуть хуже.

Приложения

1. train_test_split.py

```
# In[2]:


import sklearn
from csv import reader
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[3]:


# loading csv file into dataframe
file_spam = 'spam.csv'
sp_ds = pd.read_csv(file_spam)

# change type to 0 and 1
type_map = {'nonspam': 0, 'spam': 1}
sp_ds['type'] = sp_ds['type'].map(type_map)

x_spam = sp_ds.drop('type', axis=1)
y_spam = sp_ds.type


# In[4]:


# load txt file into dataframe
file_ttt = 'tic_tac_toe.txt'
ttt_ds = pd.read_csv(file_ttt, header=None)

# change results to 0 and 1
result_map = {'positive': 1, 'negative': 0}
ttt_ds[9] = ttt_ds[9].map(result_map)

# change feartures to numerical
t_map = {'x': 1, 'o': 2, 'b': 3}
for i in range(9):
    ttt_ds[i] = ttt_ds[i].map(t_map)


x_ttt = ttt_ds.drop(9, axis=1)
y_ttt = ttt_ds[9]


# In[6]:
```

```
spam_scores = [0]*9
ttt_scores = [0]*9
spam_scores_tr = [0]*9
ttt_scores_tr = [0]*9
train_size = [i/10 for i in range(1, 10)]

times = 100
for i in range(times):
    for tr in train_size:
        # split data
        x_s_train, x_s_test, y_s_train, y_s_test = train_test_split(x_spam, y_spam, train_size=tr)
        x_ttt_train, x_ttt_test, y_ttt_train, y_ttt_test = train_test_split(x_ttt, y_ttt, train_size=tr)

        # init Gaussian Naive Bayes
        nb_spam = GaussianNB()
        nb_t = GaussianNB()

        nb_spam.fit(x_s_train, y_s_train)
        nb_t.fit(x_ttt_train, y_ttt_train)

        sp_score = nb_spam.score(x_s_test, y_s_test)
        t_score = nb_t.score(x_ttt_test, y_ttt_test)
        sp_score_tr = nb_spam.score(x_s_train, y_s_train)
        t_score_tr = nb_t.score(x_ttt_train, y_ttt_train)

        spam_scores_tr[int(tr*10 - 1)] += sp_score_tr / times
        ttt_scores_tr[int(tr*10 - 1)] += t_score_tr / times
        spam_scores[int(tr*10 - 1)] += sp_score / times
        ttt_scores[int(tr*10 - 1)] += t_score / times


# In[8]:


plt.plot(ttt_scores, label="tic tac toe accuracy")
plt.plot(spam_scores, label="spam accuracy")
plt.xticks(range(9), train_size)
plt.title('Test accuracy')
plt.xlabel('train size')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.show()


# In[9]:


plt.plot(ttt_scores_tr, label="tic tac toe accuracy")
plt.plot(spam_scores_tr, label="spam accuracy")
```

```
plt.xticks(range(9), train_size)
plt.title('Train accuracy')
plt.xlabel('train size')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.grid(True)
plt.show()
```

2. norm_dist_classification.py

```
#!/usr/bin/env python
# coding: utf-8

# In[41]:


import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[42]:


cl0_mean = (11, 13)
cl1_mean = (19, 16)
cl0_std = 3
cl1_std = 2
n = 100
n1 = 20
n2 = 80


# In[43]:


dist0_x1 = (cl0_std * np.random.randn(n1) + cl0_mean[0])
dist0_x2 = (cl0_std * np.random.randn(n1) + cl0_mean[1])
dist1_x1 = (cl1_std * np.random.randn(n2) + cl1_mean[0])
dist1_x2 = (cl1_std * np.random.randn(n2) + cl1_mean[1])

dist0 = [[dist0_x1[i], dist0_x2[i]] for i in range(n1)]
dist1 = [[dist1_x1[i], dist1_x2[i]] for i in range(n2)]

X = np.concatenate((dist0, dist1))
y = [-1] * n1 + [1] * n2

ds = [[X[i], y[i]] for i in range(n)]
```

In[44]:

```
plt.scatter(dist0_x1, dist0_x2, color='red')
plt.scatter(dist1_x1, dist1_x2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

In[45]:

```
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8)
```

In[46]:

```
nb = GaussianNB()
nb.fit(train_X, train_y)
y_pred = nb.predict(test_X).reshape(-1, 1)
```

In[47]:

```
metrics.accuracy_score(test_y, y_pred)
```

In[48]:

```
conf_matr = metrics.confusion_matrix(test_y, y_pred)
pd.DataFrame(conf_matr)
```

In[49]:

```
fpr, tpr, tresholds = metrics.roc_curve(test_y, y_pred)
area = metrics.auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(
    fpr,
    tpr,
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % area,
```

```
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()


# In[50]:


precision, recall, _ = metrics.precision_recall_curve(test_y, y_pred)
pr_display = metrics.PrecisionRecallDisplay(precision=precision, recall=recall).plot()
```

3. knn.py

```
# In[1]:


import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


data = pd.read_csv('glass.csv')
data = data.drop('Id', axis=1)
data.head()


# In[3]:


X, y = data.drop('Type', axis=1), data.Type


# In[4]:


train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8, random_state=42)


# In[5]:
```

```
n_neighbors = np.arange(1, 30)
n_err = []

for n in n_neighbors:
    clf = KNeighborsClassifier(n_neighbors=n)
    clf.fit(train_X, train_y)
    y_pred = clf.predict(test_X)
    acc = accuracy_score(test_y, y_pred)
    n_err.append(1 - acc)


# In[6]:


plt.plot(n_neighbors, n_err)
plt.ylabel('error')
plt.xlabel('n_neighbors')
plt.title('dependence of error on number of nearest neighbors')
plt.grid(True)
plt.show()


# In[7]:


metrics = ['euclidean', 'manhattan', 'chebyshev', 'minkowski']
m_err = []

for m in metrics[:5]:
    clf = KNeighborsClassifier(metric=m, n_neighbors=1)
    clf.fit(train_X, train_y)
    y_pred = clf.predict(test_X)
    acc = accuracy_score(test_y, y_pred)
    m_err.append(1 - acc)


# In[8]:


plt.bar(metrics, m_err)
plt.ylabel('error')
plt.xlabel('metric')
plt.title('dependence of error on metric')
plt.grid(True)


# In[11]:


glass_sample = np.array([1.516, 11.7, 1.01, 1.19, 72.59, 0.42, 11.44, 0.02, 0.1])
```

```
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(train_X, train_y)
clf.predict(glass_sample.reshape(1, -1))
```

4. svm.py

```
# In[1]:


import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[2]:


def make_meshgrid(x, y, h=0.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy


# In[3]:


def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# ## A

# In[4]:


#data preprocessing
data_a = pd.read_table('svmdata_a.txt')
test_data_a = pd.read_table('svmdata_a_test.txt')

color_map = {'red': -1, 'green': 1}
data_a['Color'] = data_a['Color'].map(color_map)
test_data_a['Color'] = test_data_a['Color'].map(color_map)

train_X_a = data_a.drop('Color', axis=1)
train_y_a = data_a.Color
test_X_a = test_data_a.drop('Color', axis=1)
test_y_a = test_data_a.Color
```

In[5]:

```
#create and train svm model with linear kernel
clf_a = svm.SVC(kernel='linear')
clf_a.fit(train_X_a, train_y_a)
```

In[6]:

```
#check accuracy and confusion matrix on train and test
pred_a = clf_a.predict(test_X_a)
pred_tr_a = clf_a.predict(train_X_a)
conf_m_a = confusion_matrix(test_y_a, pred_a)
conf_m_tr_a = confusion_matrix(train_y_a, pred_tr_a)

print('Test accuracy ', accuracy_score(test_y_a, pred_a))
print('Train accuracy ', accuracy_score(train_y_a, pred_tr_a))
print('Test confusion matrix\n', pd.DataFrame(conf_m_a))
print('Train confusion matrix\n', pd.DataFrame(conf_m_tr_a))
print('Number of support vectors', clf_a.n_support_)
```

In[7]:

```
x0, x1 = train_X_a.to_numpy()[:, 0], train_X_a.to_numpy()[:, 1]
xx, yy = make_meshgrid(x0, x1)
z = clf_a.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
```

In[8]:

```
plt.contourf(xx, yy, z, cmap=plt.cm.coolwarm)
plt.scatter(x0, x1, c=train_y_a)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title('SVC with linear kernel')
plt.show()
```

B

In[9]:

```
#data preprocessing
```

```
data_b = pd.read_table('svmdata_b.txt')
test_data_b = pd.read_table('svmdata_b_test.txt')

color_map = {'red': -1, 'green': 1}
data_b['Colors'] = data_b['Colors'].map(color_map)
test_data_b['Colors'] = test_data_b['Colors'].map(color_map)

train_X_b = data_b.drop('Colors', axis=1)
train_y_b = data_b.Colors
test_X_b = test_data_b.drop('Colors', axis=1)
test_y_b = test_data_b.Colors


# In[10]:


tolerance = [2**i for i in range(-6, 10)]
train_acc = []
test_acc = []

for tol in tolerance:
    clf_b = svm.SVC(kernel='linear', C=tol)
    clf_b.fit(train_X_b, train_y_b)

    pred_train_b = clf_b.predict(train_X_b)
    pred_test_b = clf_b.predict(test_X_b)
    tr_acc = accuracy_score(train_y_b, pred_train_b)
    t_acc = accuracy_score(test_y_b, pred_test_b)

    train_acc.append(tr_acc)
    test_acc.append(t_acc)


# In[11]:


plt.figure(figsize=(14,6))
plt.plot(train_acc, label='train accuracy')
plt.plot(test_acc, label='test accuracy')
plt.xticks(range(len(tolerance)), tolerance)
plt.xlabel('parameter C')
plt.ylabel('accuracy')
plt.legend(loc='best')
plt.grid(visible=True)
plt.show()


# ## C

# In[12]:
```

```
data_c = pd.read_table('svmdata_c.txt')
test_data_c = pd.read_table('svmdata_c_test.txt')

color_map = {'red': -1, 'green': 1}
data_c['Colors'] = data_c['Colors'].map(color_map)
test_data_c['Colors'] = test_data_c['Colors'].map(color_map)

train_X_c = data_c.drop('Colors', axis=1)
train_y_c = data_c.Colors
test_X_c = test_data_c.drop('Colors', axis=1)
test_y_c = test_data_c.Colors


# In[13]:


models = (
    svm.SVC(kernel='linear'),
    svm.SVC(kernel='poly', degree=1),
    svm.SVC(kernel='poly', degree=2),
    svm.SVC(kernel='poly', degree=3),
    svm.SVC(kernel='poly', degree=4),
    svm.SVC(kernel='poly', degree=5),
    svm.SVC(kernel='sigmoid'),
    svm.SVC(kernel='rbf')
)
models = (m.fit(train_X_c, train_y_c) for m in models)


# In[14]:


titles = (
    "SVC with linear kernel",
    "SVC with polynomial (degree 1) kernel",
    "SVC with polynomial (degree 2) kernel",
    "SVC with polynomial (degree 3) kernel",
    "SVC with polynomial (degree 4) kernel",
    "SVC with polynomial (degree 5) kernel",
    "SVC with sigmoid kernel",
    "SVC with RBF kernel",
)


# In[15]:


x0, x1 = train_X_c.to_numpy()[:, 0], train_X_c.to_numpy()[:, 1]
xx, yy = make_meshgrid(x0, x1)


# In[16]:
```

```
fig, sub = plt.subplots(4, 2, figsize=(10,10))
plt.subplots_adjust(wspace=0.4, hspace=0.4)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(x0, x1, c=train_y_c, cmap=plt.cm.coolwarm, edgecolors="k")
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel("X0")
    ax.set_ylabel("X1")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)


# ## D

# In[17]:


data_d = pd.read_table('svmdata_d.txt')
test_data_d = pd.read_table('svmdata_d_test.txt')

color_map = {'red': -1, 'green': 1}
data_d['Colors'] = data_d['Colors'].map(color_map)
test_data_d['Colors'] = test_data_d['Colors'].map(color_map)

train_X_d = data_d.drop('Colors', axis=1)
train_y_d = data_d.Colors
test_X_d = test_data_d.drop('Colors', axis=1)
test_y_d = test_data_d.Colors


# In[18]:


models_d = (
    svm.SVC(kernel='linear'),
    svm.SVC(kernel='poly', degree=1),
    svm.SVC(kernel='poly', degree=2),
    svm.SVC(kernel='poly', degree=3),
    svm.SVC(kernel='poly', degree=4),
    svm.SVC(kernel='poly', degree=5),
    svm.SVC(kernel='sigmoid'),
    svm.SVC(kernel='rbf')
)
models_d = (m.fit(train_X_d, train_y_d) for m in models_d)


# In[19]:
```

```
titles = (
    "SVC with linear kernel",
    "SVC with polynomial (degree 1) kernel",
    "SVC with polynomial (degree 2) kernel",
    "SVC with polynomial (degree 3) kernel",
    "SVC with polynomial (degree 4) kernel",
    "SVC with polynomial (degree 5) kernel",
    "SVC with sigmoid kernel",
    "SVC with RBF kernel",
)


# In[20]:


x0, x1 = train_X_d.to_numpy()[:, 0], train_X_d.to_numpy()[:, 1]
xx, yy = make_meshgrid(x0, x1)


# In[21]:


fig, sub = plt.subplots(4, 2, figsize=(10,10))
plt.subplots_adjust(wspace=0.4, hspace=0.4)

for clf, title, ax in zip(models_d, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(x0, x1, c=train_y_d, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel("X0")
    ax.set_ylabel("X1")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)


# ## E

# In[22]:


data_e = pd.read_table('svmdata_e.txt')
test_data_e = pd.read_table('svmdata_e_test.txt')

color_map = {'red': -1, 'green': 1}
data_e['Colors'] = data_e['Colors'].map(color_map)
test_data_e['Colors'] = test_data_e['Colors'].map(color_map)

train_X_e = data_e.drop('Colors', axis=1)
```

```
train_y_e = data_e.Colors
test_X_e = test_data_e.drop('Colors', axis=1)
test_y_e = test_data_e.Colors


# In[23]:


x0, x1 = train_X_e.to_numpy()[:, 0], train_X_e.to_numpy()[:, 1]
xx, yy = make_meshgrid(x0, x1)


# In[24]:


titles = (
    "SVC with linear kernel",
    "SVC with polynomial (degree 1) kernel",
    "SVC with polynomial (degree 2) kernel",
    "SVC with polynomial (degree 3) kernel",
    "SVC with polynomial (degree 4) kernel",
    "SVC with polynomial (degree 5) kernel",
    "SVC with sigmoid kernel",
    "SVC with RBF kernel",
)


# In[27]:


gammas = [0.1, 1, 10, 50, 150]

for gamma in gammas:
    print('gamma ', gamma)

    models_e = (
        svm.SVC(kernel='linear', gamma=gamma),
        svm.SVC(kernel='poly', degree=1, gamma=gamma),
        svm.SVC(kernel='poly', degree=2, gamma=gamma),
        svm.SVC(kernel='poly', degree=3, gamma=gamma),
        svm.SVC(kernel='poly', degree=4, gamma=gamma),
        svm.SVC(kernel='poly', degree=5, gamma=gamma),
        svm.SVC(kernel='sigmoid', gamma=gamma),
        svm.SVC(kernel='rbf', gamma=gamma)
    )
    models_e = (m.fit(train_X_e, train_y_e) for m in models_e)

    fig, sub = plt.subplots(4, 2, figsize=(10,10))
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    for clf, title, ax in zip(models_e, titles, sub.flatten()):
        plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
```

```
        ax.scatter(x0, x1, c=train_y_e, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xlabel("X0")
        ax.set_ylabel("X1")
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(title)

plt.show()
```

5. forest.py

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# ## A

# In[2]:


glass_data = pd.read_csv('glass.csv')
glass_data = glass_data.drop('Id', axis=1)

X, y = glass_data.drop('Type', axis=1), glass_data.Type
glass_data.info()


# In[3]:


train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8, random_state=42)


# In[4]:


plt.figure(figsize=(35,35))
clf = DecisionTreeClassifier()
clf.fit(train_X, train_y)
plot_tree(clf, filled=True)
plt.title('Decision tree')
```

```
plt.show()


# In[5]:


y_pred = clf.predict(test_X)
print('accuracy ', accuracy_score(test_y, y_pred))
clf.get_depth()


# In[25]:


splitter = ['best', 'random']
accuracy = {'best': 0,
            'random': 0
            }
k = 1000

for split in splitter:
    for i in range(k):
        tree = DecisionTreeClassifier(splitter=split)
        tree.fit(train_X, train_y)

        pred_y = tree.predict(test_X)
        acc = accuracy_score(test_y, pred_y)

        accuracy[split]+= acc/k


# In[42]:


plt.bar(splitter, list(accuracy.values()), width=0.4)
plt.xticks(range(len(splitter)), splitter)
plt.xlabel('Splitter')
plt.ylabel('Accuracy')
plt.show()


# In[63]:


max_depth = [i for i in range(1, 100, 5)]
accuracy_md = dict()

k = 1000
for i in range(k):
    for depth in max_depth:
        tree = DecisionTreeClassifier(max_depth=depth)
        tree.fit(train_X, train_y)
```

```
        pred_y = tree.predict(test_X)
        acc = accuracy_score(test_y, pred_y)

        if depth in accuracy_md.keys():
            accuracy_md[depth] += acc/k
        else:
            accuracy_md[depth] = acc/k


# In[64]:


accuracy_md


# In[72]:


plt.figure(figsize=(10,6))
plt.plot(list(accuracy_md.values()))
plt.xticks(range(len(max_depth)), max_depth)
plt.xlabel('Max depth')
plt.ylabel('Accuracy')
plt.grid()
plt.show()


# In[97]:


min_samples_leaf = [i for i in range(2, 100, 5)]
accuracy_leaf = dict()

k = 100
for i in range(k):
    for leaf in min_samples_leaf:
        tree = DecisionTreeClassifier(min_samples_leaf=leaf)
        tree.fit(train_X, train_y)

        pred_y = tree.predict(test_X)
        acc = accuracy_score(test_y, pred_y)

        if depth in accuracy_leaf.keys():
            accuracy_leaf[leaf] += acc
        else:
            accuracy_leaf[leaf] = acc


# In[98]:
```

```
accuracy_leaf


# In[99]:


plt.figure(figsize=(10,6))
plt.plot(list(accuracy_leaf.values()))
plt.xticks(range(len(min_samples_leaf)), min_samples_leaf)
plt.xlabel('Min samples leaf')
plt.ylabel('Accuracy')
plt.grid()
plt.show()


# In[118]:


criterion = ['gini', 'entropy']
accuracy_crit = {'gini': 0,
                 'entropy': 0
                }
k = 1000

for crit in criterion:
    for i in range(k):
        tree = DecisionTreeClassifier(criterion=crit)
        tree.fit(train_X, train_y)

        pred_y = tree.predict(test_X)
        acc = accuracy_score(test_y, pred_y)

        accuracy_crit[crit] += acc/k


# In[ ]:




# In[119]:


plt.bar(criterion, list(accuracy_crit.values()), width=0.4)
plt.xticks(range(len(criterion)), criterion)
plt.xlabel('Criterion')
plt.ylabel('Accuracy')
plt.show()


# ## B
```

In[106]:

```
spam = pd.read_csv('spam7.csv')

yn_map = {'y': 1, 'n': -1}
spam['yesno'] = spam['yesno'].map(yn_map)

X_spam = spam.drop('yesno', axis=1)
y_spam = spam.yesno

train_X_spam, test_X_spam, train_y_spam, test_y_spam = train_test_split(X_spam, y_spam, train_size=0.8, random_state=42)
```

In[140]:

```
spam.describe()
```

In[126]:

```
tree = DecisionTreeClassifier()
tree.fit(train_X_spam, train_y_spam)

pred = tree.predict(test_X_spam)

acc = accuracy_score(test_y_spam, pred)
```

In[128]:

```
from sklearn.model_selection import GridSearchCV
params = {
    'max_depth': np.arange(1, 30, 3),
    'splitter': ['best'],
    'min_samples_leaf': np.arange(2, 50, 4),
    'criterion': ['gini', 'entropy'],
}
clf = DecisionTreeClassifier()
tree_grid = GridSearchCV(clf, params, cv=5, scoring='accuracy', n_jobs=-1)

tree_grid.fit(X_spam, y_spam)
```

In[130]:

```
tree.get_depth()


# In[133]:


tree_grid.best_params_, tree_grid.best_score_, tree_grid.best_estimator_


# In[135]:


best_tree = tree_grid.best_estimator_

best_tree.fit(train_X_spam, train_y_spam)

b_pred = best_tree.predict(test_X_spam)

b_acc = accuracy_score(test_y_spam, b_pred)


# In[137]:


b_acc, acc


# In[138]:


plt.figure(figsize=(35,35))
plot_tree(best_tree, filled=True)
plt.title('Decision tree')
plt.show()
```

6. banking_score.py

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:


import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
```

In[5]:

```
data = pd.read_csv('bank_scoring_train.csv', delimiter='\t')
test_data = pd.read_csv('bank_scoring_test.csv', delimiter='\t')

X_train = data.drop('SeriousDlqin2yrs', axis=1)
y_train = data.SeriousDlqin2yrs
X_test = test_data.drop('SeriousDlqin2yrs', axis=1)
y_test = test_data.SeriousDlqin2yrs
```

Decision tree

In[6]:

```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

y_pred_tree = tree.predict(X_test)
acc = accuracy_score(y_test, y_pred_tree)
f1 = f1_score(y_test, y_pred_tree)
acc, f1
```

In[7]:

```
tree.get_depth(), tree.get_n_leaves(), tree.get_params()
```

In[8]:

```
conf_matr = confusion_matrix(y_test, y_pred_tree)
pd.DataFrame(conf_matr)
```

In[9]:

```
precision, recall, _ = metrics.precision_recall_curve(y_test, y_pred_tree)
pr_display = metrics.PrecisionRecallDisplay(precision=precision, recall=recall).plot()
```

SVM

In[10]:

```
svm_clf = svm.SVC()
svm_clf.fit(X_train, y_train)

y_pred_svm = svm_clf.predict(X_test)
acc_svm = accuracy_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
acc_svm, f1_svm


# In[11]:


conf_matr_svm = confusion_matrix(y_test, y_pred_svm)
pd.DataFrame(conf_matr_svm)


# In[12]:


precision, recall, _ = metrics.precision_recall_curve(y_test, y_pred_svm)
pr_display = metrics.PrecisionRecallDisplay(precision=precision, recall=recall).plot()


# # KNN

# In[ ]:


knn_clf = KNeighborsClassifier(n_neighbors=1)
knn_clf.fit(X_train, y_train)

y_pred_knn = knn_clf.predict(X_test)
acc_knn = accuracy_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)
acc_knn, f1_knn


# In[ ]:


conf_matr_svm = confusion_matrix(y_test, y_pred_knn)
pd.DataFrame(conf_matr_svm)


# In[ ]:


precision, recall, _ = metrics.precision_recall_curve(y_test, y_pred_knn)
pr_display = metrics.PrecisionRecallDisplay(precision=precision, recall=recall).plot()
```

In[ ]: