Санкт-Петербургский Политехнический университет имени Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

ЛАБОРАТОРНАЯ РАБОТА №1

«КЛАССИФИКАЦИЯ»

по дисциплине «Статистическое моделирование случайных процессов и систем»

Выполнил студент гр. 3530904/70103 Русаков Е.С.

Преподаватель Селин И.

Санкт-Петербург
2020

Оглавление

Задание

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, 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. На основе как минимум двух из изученных методов постройте классификаторы, предсказывающий значение столбца «SeriousDlqin2yrs». Определите, какой из классификаторов сработал лучше. Оцените качество классификации на bank_scoring_test.csv для обоих классификаторов.

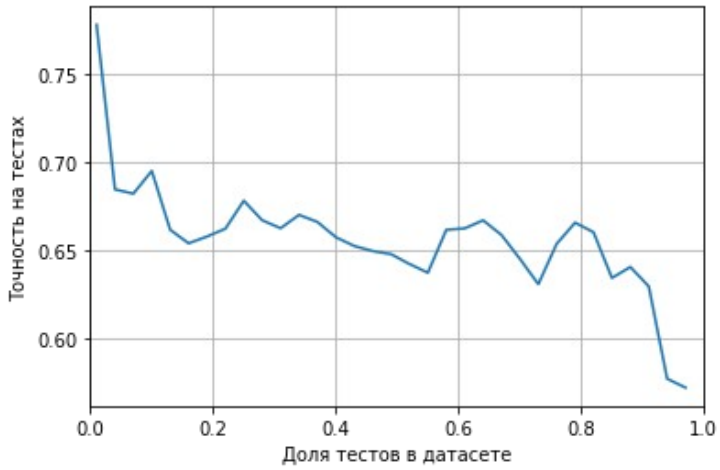
Ход работы

1. Влияние объёма выборки на точность предсказаний

Используя наивный байесовский классификатор, изучить влияние объёма обучающей выборки на точность предсказаний.

а) Классификатор крестиков-ноликов

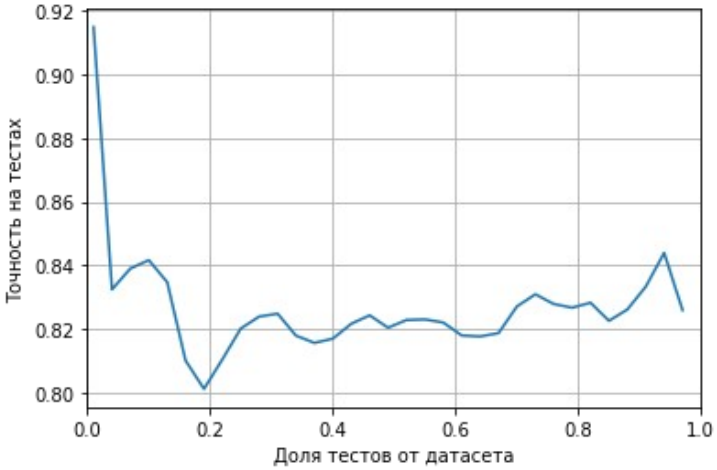
Из файла с данными загружена выборка, представляющая собой набор партий из крестиков-ноликов с пометками выигрыша или проигрыша ведущей стороны. Из выборки выделяется тестовая часть, а остаток будет использоваться для обучения классификатора. Для классификатора были написаны процедуры тренировки и тестирования классификатора (см. Приложение 1.1). Также организован подсчёт точностей и вывод графика зависимости точности получившегося классификатора на тестовой выборке от доли обучающей выборки.



Из графика можно заключить, что с увеличением объёма обучающей выборки точность в целом растёт. Левый пик вызван малым абсолютным числом тестов, влекущим необъективную оценку. Правый спад же вызван малостью абсолютного числа обучающих примеров.

б) Классификатор спама

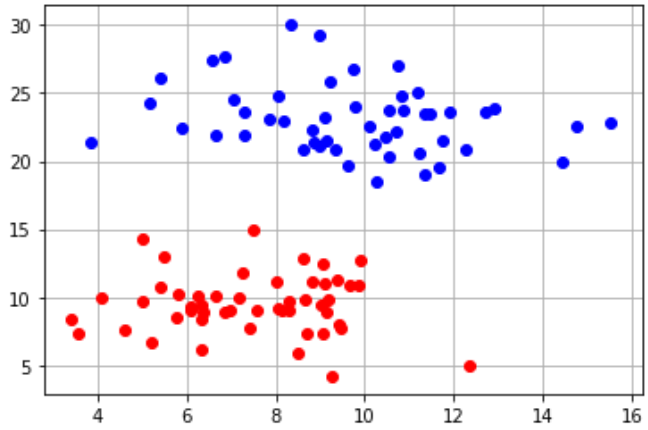
Здесь вместо самописного классификатора используется предлагаемый библиотекой SkLearn, а именно GaussianNB. (Код программы см. Приложение 1.2)



По отсутствию резкого спада в правой части можно судить, что байесову классификатору может быть достаточно даже небольшого объёма данных, чтобы показать хорошие результаты на обширном наборе тестов. С дальнейшим увеличением объёма тренировочных данных точность почти не растёт. Резкий скачок слева объясняется относительно маленьким и необъективным набором тестов.

2. Байесовский классификатор точек и его характеристики

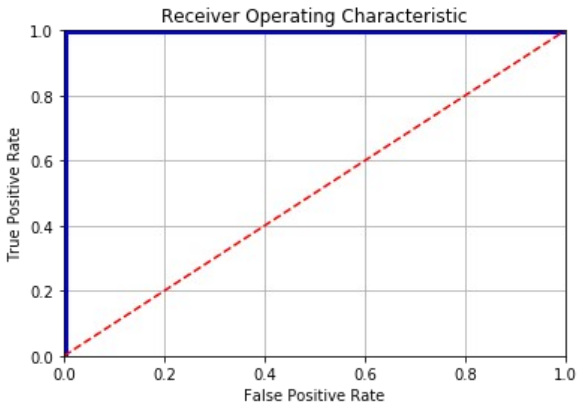
Требуется сгенерировать выборку из точек на плоскости двух классов: «+1» и «-1». Их координаты подчиняются нормальным распределениям с заданными параметрами, отобразить данные графически, построить байесовский классификатор для этих данных и проверить его различными методами: точность, матрица ошибок, ROC- и PR-кривые.

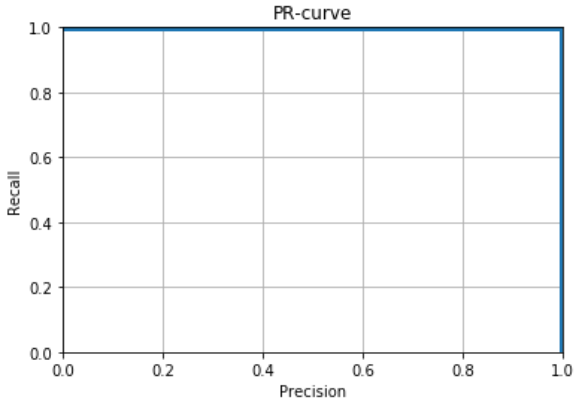


Использовался классификатор GaussianNB из библиотеки Sklearn. Полный текст программы см. в Приложении 2.
В тестовую выборку определено 20% всех данных. После обучения классификатор показал 100% точность на тесте. Матрица ошибок выглядит так:

	+1	-1
+1	10	0
-1	0	10

В матрице ошибок по столбцам обозначены предсказанная принадлежность образцов к классам, а по строкам их истинная принадлежность. Как видно, оба десятка образцов предсказана правильно. Именно поэтому ROC- и PR-кривые представляют собой «идеальные» углы.



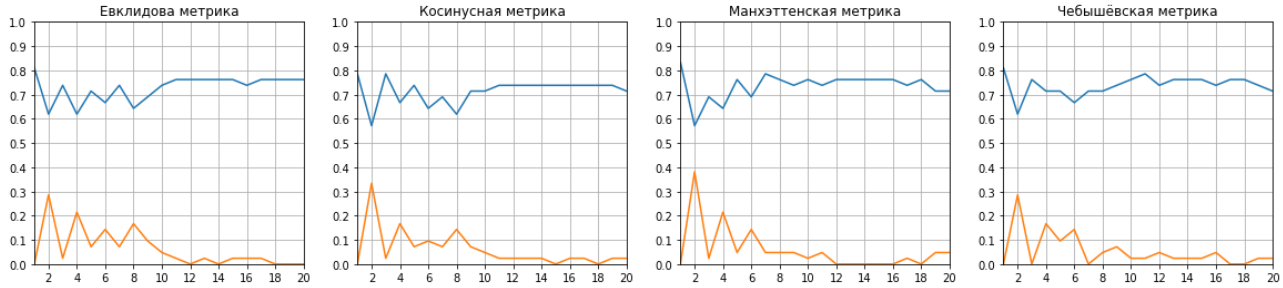


3. Метод ближайших соседей

Требуется построить классификатор на основе метода k ближайших соседей для классификации данных, представленных в файле glass.csv.

а, б) Анализ зависимости точности от числа соседей

Написан классификатор на основе метода k ближайших соседей, применяющий несколько метрик расстояния между объектами: евклидову, косинусную, манхэттенскую и чебышёвскую (код см. Приложение 3.1). Построены графики точности каждой из метрик в зависимости от числа «голосующих» соседей, где так же берётся во внимание случаи, когда «голоса» делятся поровну и невозможно сделать предсказание.



На графиках по вертикали отложена точность, а по горизонтали – число используемых соседей. Синей линией показана точность метрик, а оранжевой доля случаев, где голоса поделились поровну.

в) Определение нового типа стекла

Требуется определить тип стекла с параметрами 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.

Вот результаты программы (код см. Приложение 3.2):

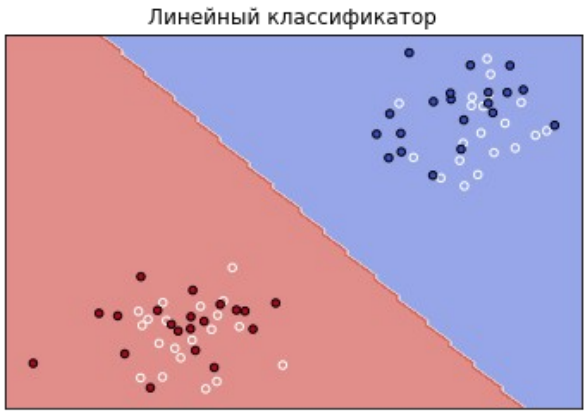
Для числа соседей с 1 по 20 все четыре метрики оказались единогласны – это стекло №5.

4. Метод опорных векторов

Для построения классификатора используется SVM из библиотеки Sklearn.

а) Классификатор с линейным ядром

Для визуализации используются функции, предложенные в документации Sklearn, модифицированные для параллельного отображения двух выборок точек.



На рисунке точками с чёрной обводкой показаны тестовые данные, а с белой – обучающие. Как видно по рисунку, классификатор нашёл прямую успешно, делящую оба типа точек по классам. Следовательно, матрица ошибок классификатора:

20	0
0	20

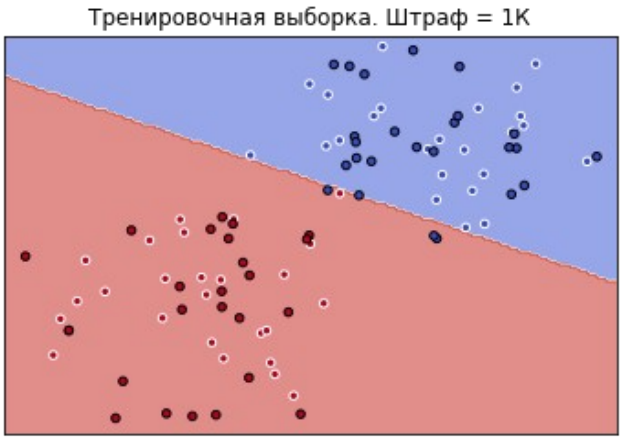
Классификатор имеет следующие опорные вектора:

```
[[0.90356343 0.72236951]
 [0.98510067 0.68430049]
 [0.80835351 0.82126184]
 [0.14874741 0.13128834]
 [0.18027464 0.29152253]
 [0.20454273 0.00740181]]
```

б) Переобучение

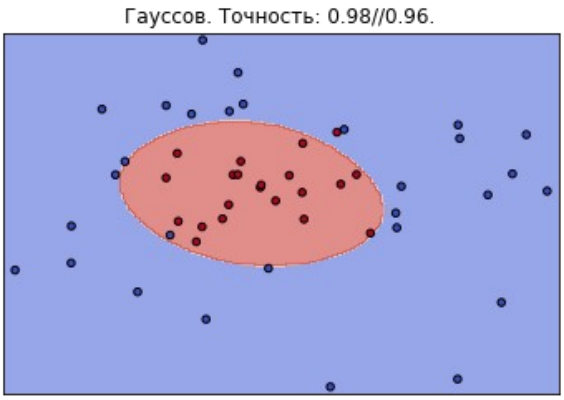
Попытаемся добиться переобучения путём регулировки штрафного параметра. Пусть он будет равен 100.

Для удобства отобразим на холсте сразу же и тестовую выборку. На нём хорошо заметно, что один из красных кружков из тестовой выборки находится среди синих и провести прямую так, чтобы без изъянов разделить обе выборки никак не получится.



Что выбирать в таком случае и как: хороший результат на тестах или на обучении? Ответ лежит посередине. В любом случае точности на тестировании и обучении не должны сильно различаться. В нашем случае потеря одной точки там или сям – невеликая, разумная и неизбежная потеря.

в) Проверка разных ядер в методе опорных векторов



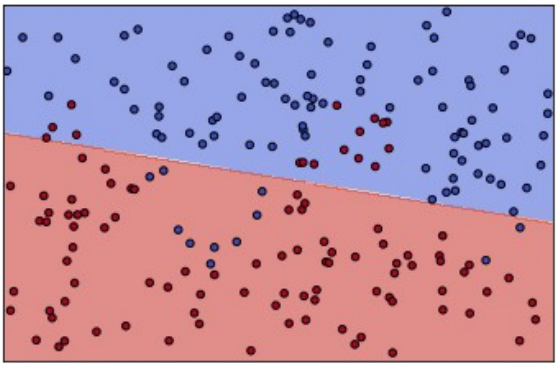
На рисунках подписано название используемого ядра и точности на обучающей и тестовой выборках через двойную косую черту.

Выводы следующие:

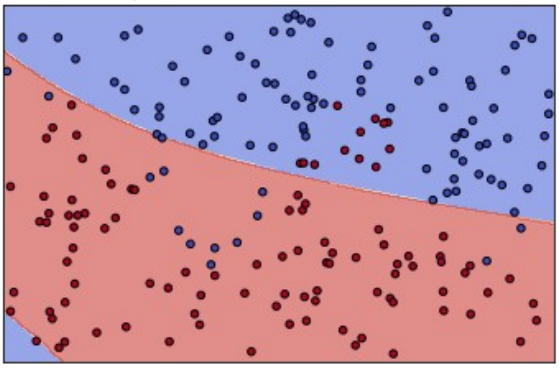
- Данные, оказывается, расположены скоплением одного класса в окружении другого
- Чётность/нечётность степеней полиномиального ядра влияет на замкнутость/незамкнутость границ классов и, следовательно, на результат в нашем случае. Это нужно учитывать при выборе ядра, если известно расположение классов.
- В отличие от полиномов, ограниченных сверху или снизу, или не ограниченных вовсе, сигмоидальное ядро очерчивает границу, формой напоминающую обратную пропорциональность.
- Гауссово ядро представляет собой подобие эллипсоида. Оно показало наилучшие результаты на обучающей и тестовой выборках

г) То же самое, что и предыдущее задание, но с другими данными

Линейный. Точность: 0.86//0.865.

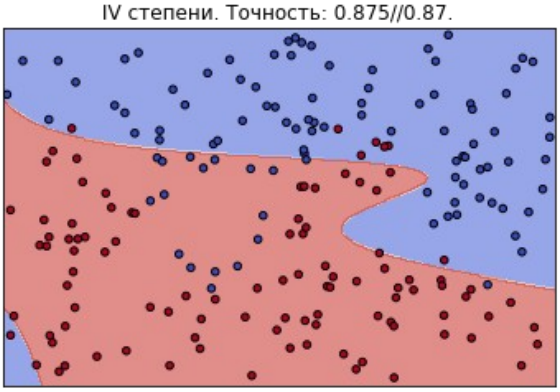


Квадратичный. Точность: 0.85//0.865.

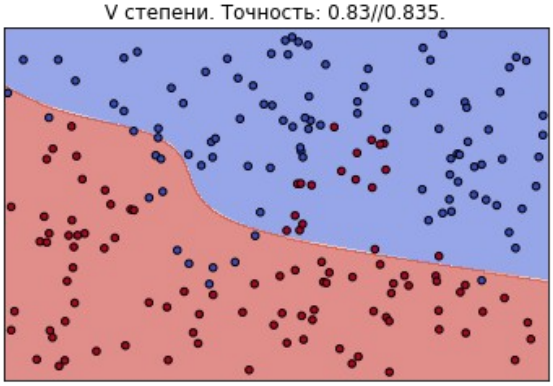


Кубический. Точность: 0.885//0.865.

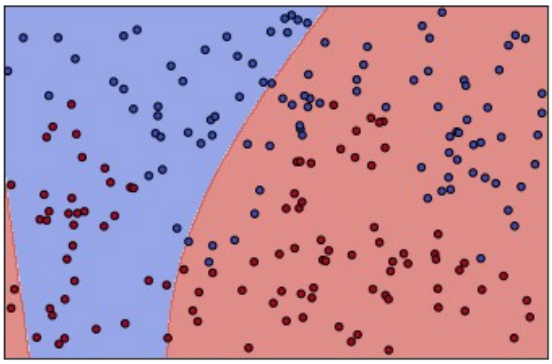
IV степени. Точность: 0.875//0.87.



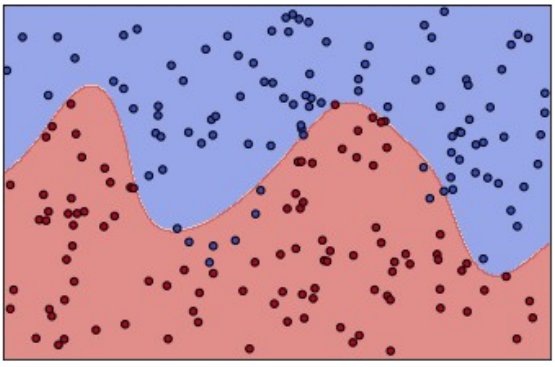
V степени. Точность: 0.83//0.835.



Сигмоидальный. Точность: 0.61//0.51.



Гауссов. Точность: 0.98//0.95.

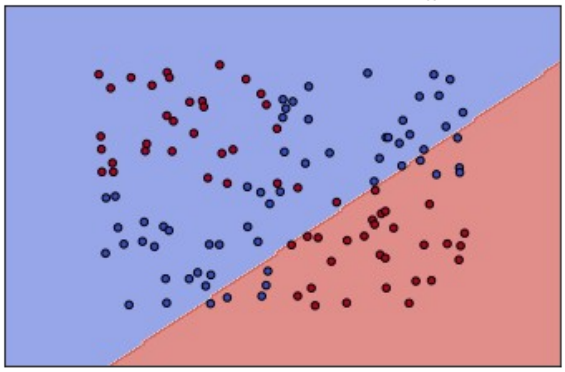


В этом случае данные расположены волнами, скорее всего по синусоиде.

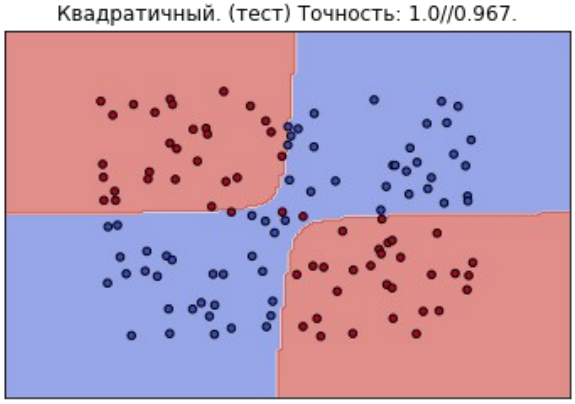
- Степень полиноиальных ядер влияет, правильнее сказать, не на замкнутость/незамкнутость линий разграничения, а скорее на общую возможную сложность из формы
- Также замечено, что классификаторы с полиномиальными ядрами строились куда быстрее,чем с сигмоидальным или гауссовым. Однако гауссов классификатор снова лучше смог приспособиться к данным и показал лучшие результаты на проверке.

д) Попробуем добиться эффекта переобучения, установив штрафной параметр gamma=10^2.

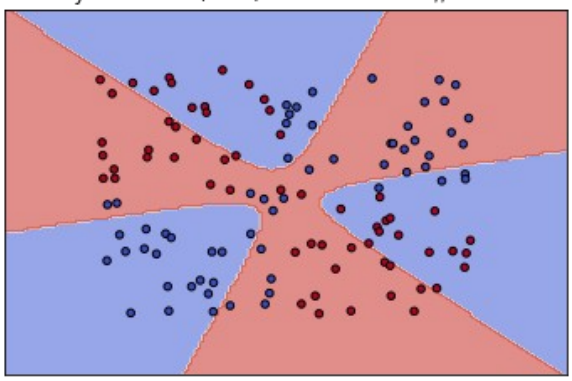
Линейный. (тест) Точность: 0.667//0.65.



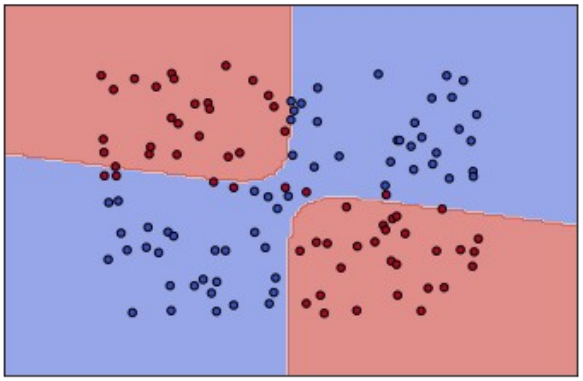
Квадратичный. (тест) Точность: 1.0//0.967.



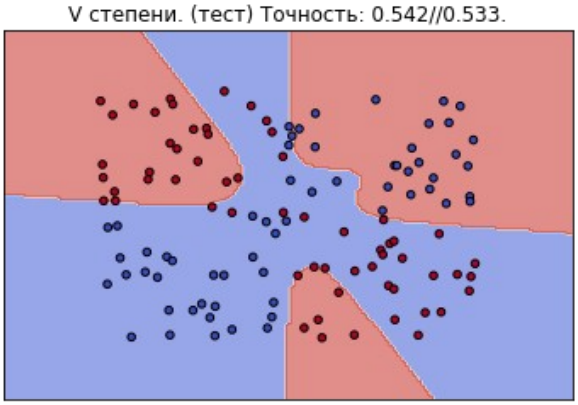
Кубический. (тест) Точность: 0.567//0.517.



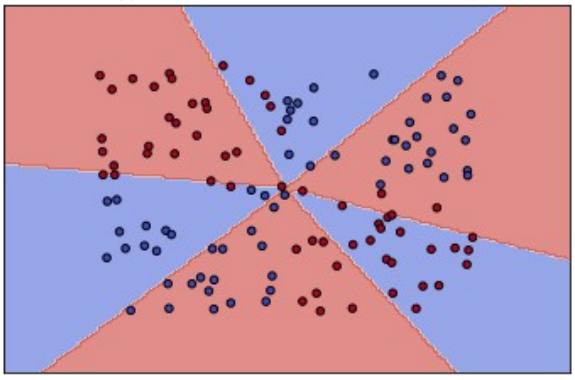
IV степени. (тест) Точность: 1.0//0.942.

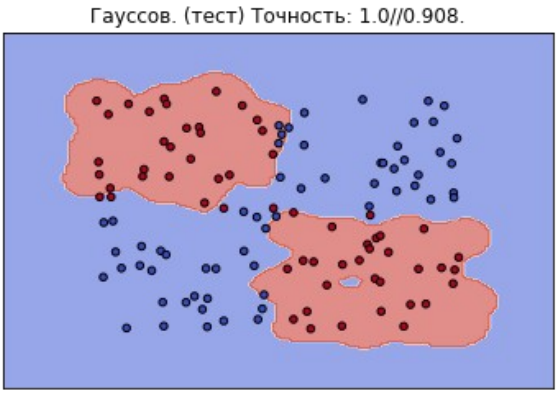


V степени. (тест) Точность: 0.542//0.533.



Сигмоидальный. (тест) Точность: 0.592//0.45.

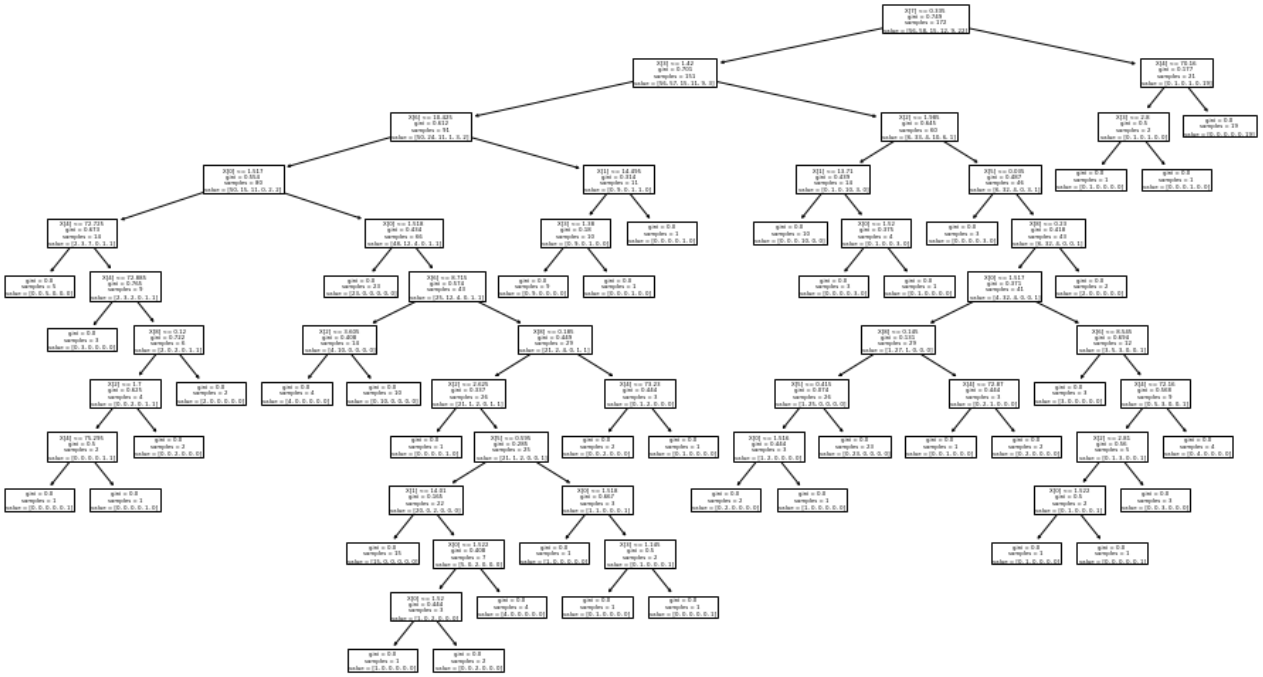




Путём увеличения штрафного параметра достичь переобучения получилось. Человеческим глазом видно, что данные расположены по четвертям прямоугольной системы координат, однако это не совсем очевидно для алгоритмов обучения. Неподходящие классификаторы даже с большим штрафом не смогли достичь хорошей точности. Лидировавший ранее классификатор на основе гауссова ядра в этот раз подошёл к задаче слишком буквально: его решение неплохо подошло под имеющиеся данные, но использование его на новых чревато неправильной классификацией. Лучше всех здесь показали себя полиномиальные ядра II и IV степеней: они разграничили не только имеющиеся данные, но и области потенциального появления новых точек.

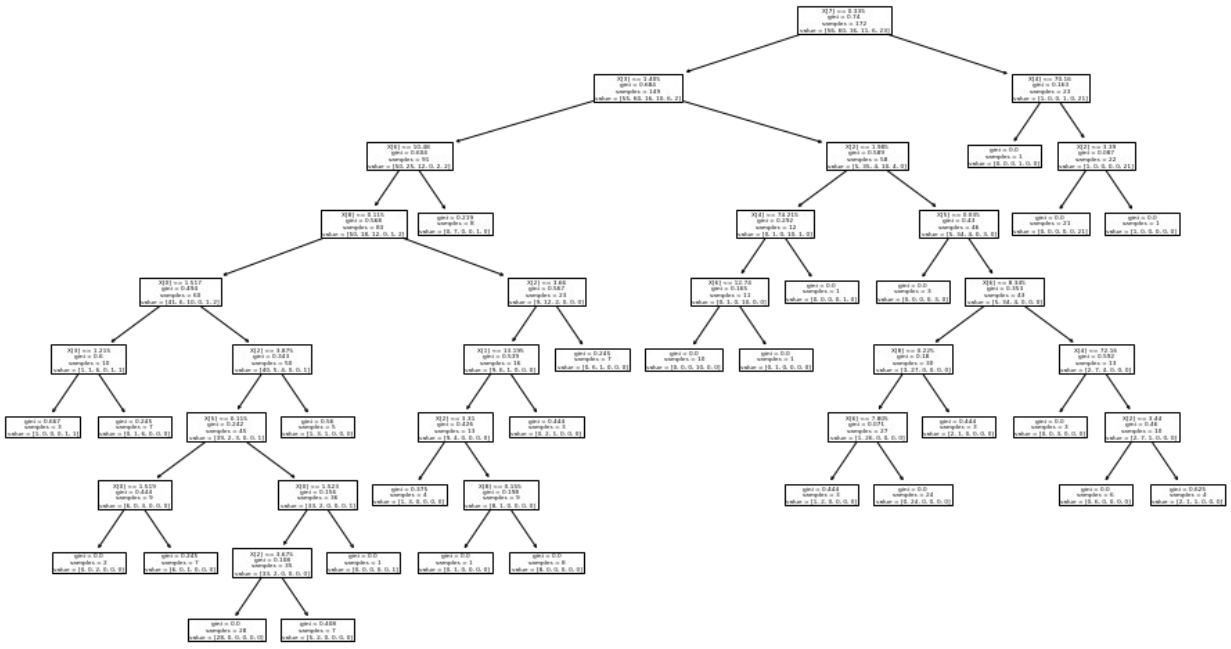
5. Деревья решений

а) Исследование классификатора на основе метода деревьев решений.
Испольуется классификатор DesicionTreeClassifier() из библиотеки SkLearn. В качестве данных используется датасет из glass.csv. Параметры классификатора стандартные: ограничение на глубину отсутствует, параметр расщепления ветви равен двум.
Под тесты определено 20% выборки.

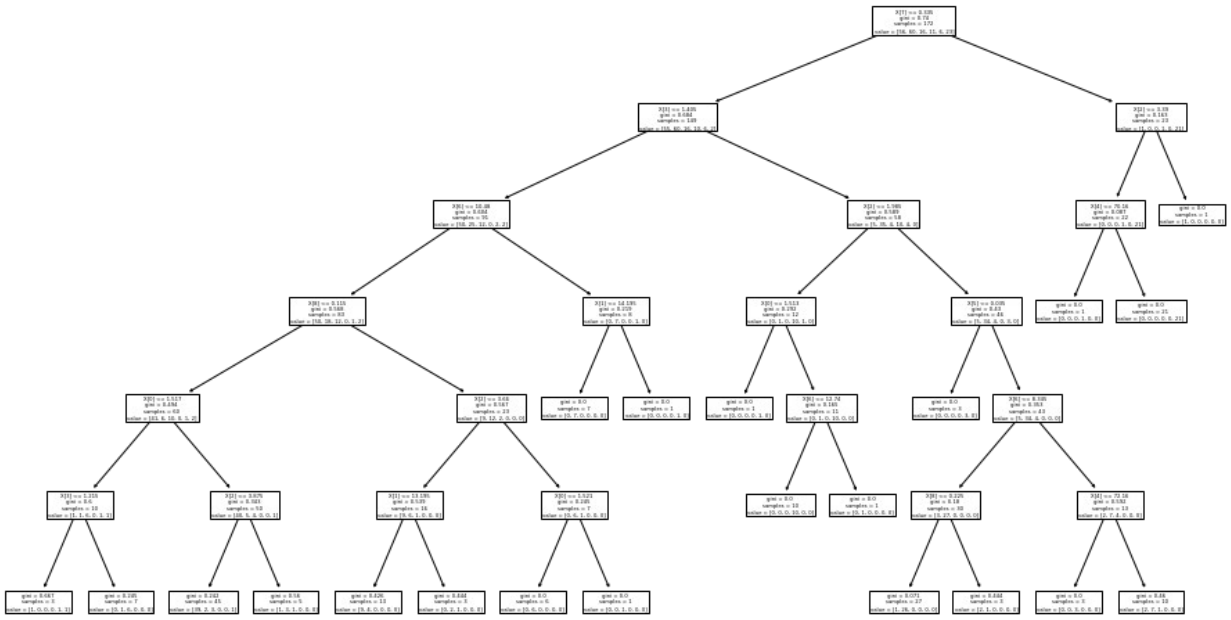


Точность получившегося дерева: 100% на обучающей и 71% на тестовой выборке.
Алгоритм явно пытается всю выборку преобразовать в дерево, однако точность на тесте от такого подхода разительно отличается от точности на тренировочной выборке.

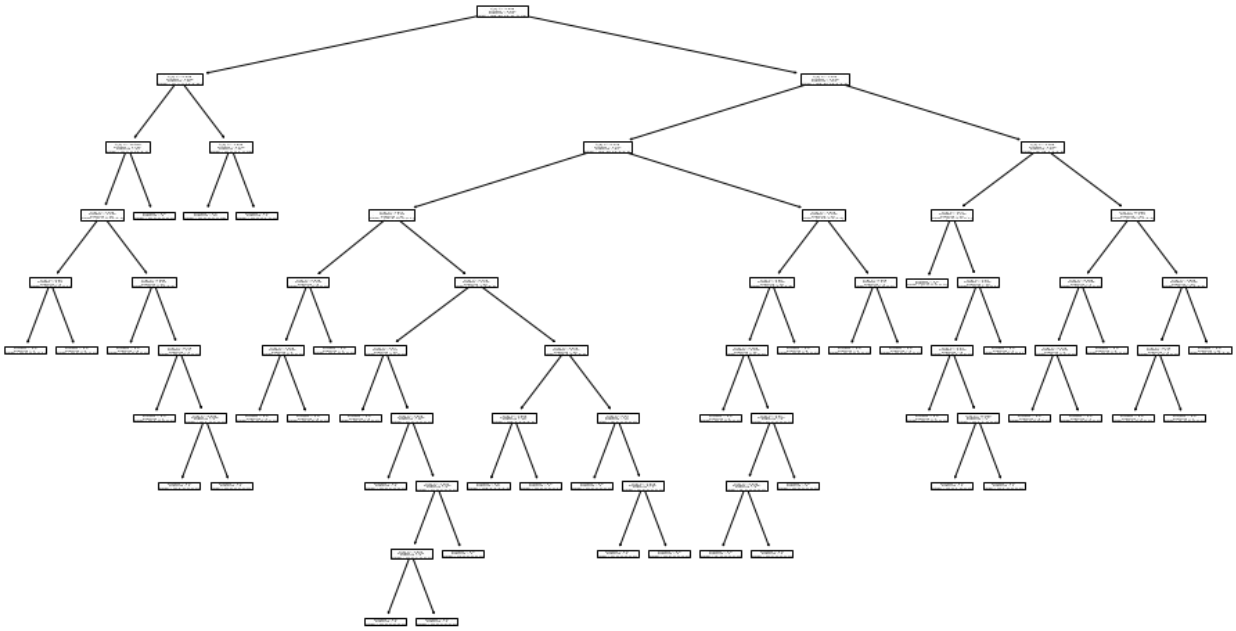
Установим параметр расщепления 0.05.



Ожидаемо, точность такого классификатора упала до значений 90%//69%. Разница всё ещё сохраняется и может ввести в заблуждение.
Попробуем ограничивать не параметр расщепления, а максимальную глубину. Установим её значение равным шести.

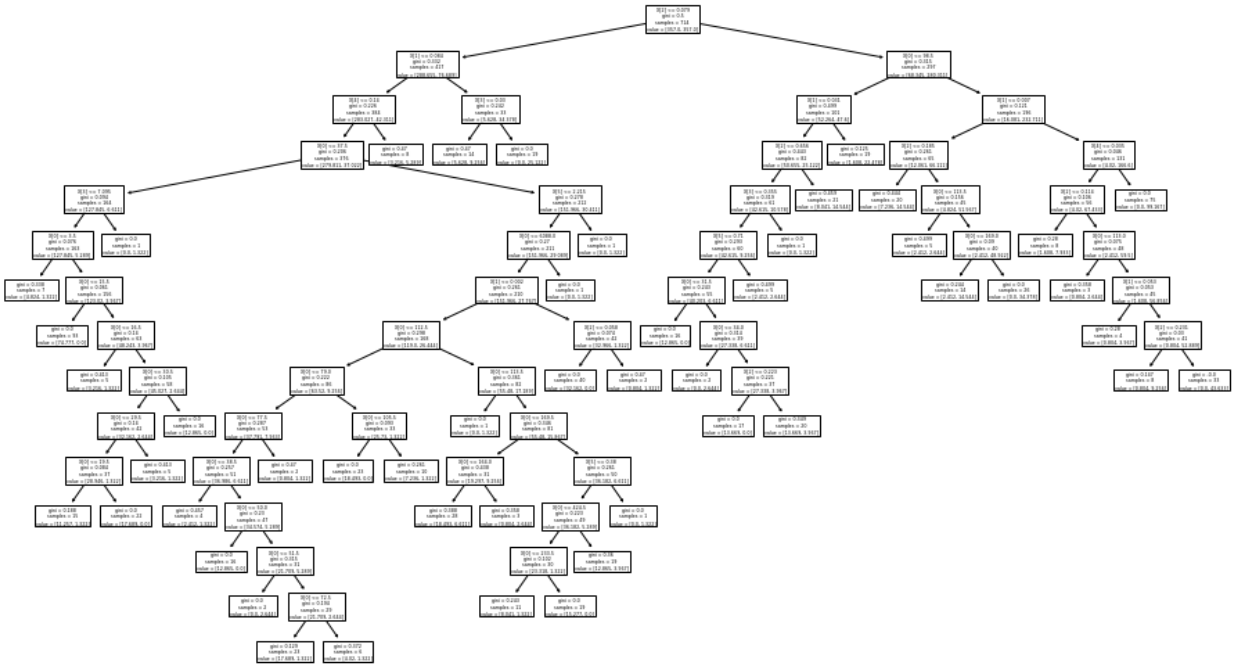


Точность этого дерева 88%//64%.
Изменим теперь критерий расщепления вершин с критерия Джини на энтропический, оставив остальные параметры по умолчанию
Получилось дерево с точностями 100%//71%. К сожалению,. Результат всё тот же: разрыв между обучающей и тестовой выборках слишком велик.

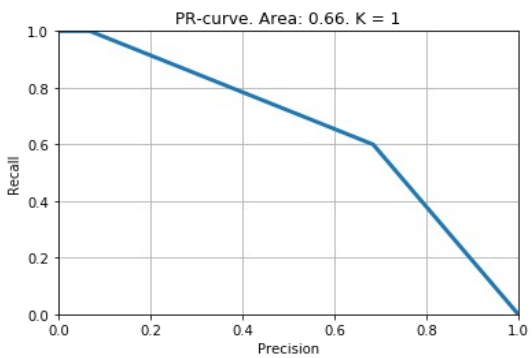


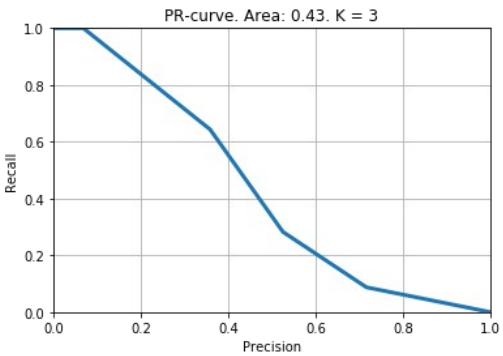
б) Построение дерева классификации спама
Из файла spam7.csv попробуем построить лучшее дерево классификации, меняя параметры.

Метод научного тыка показал, что лучших показателей дерево достигает при следующих параметрах:
1) Расщепление: 0.04
2) Максимальная глубина: N/A
3) Критерий расщепления: Джини
Такие параметры не только обеспечивают относительно хорошее качество тестирования, но и не вводят в заблуждение большой точностью на тренировочной выборке. Точность этого классификатора: 88%//85%. Кроме того, его построение занимает меньше времени, чем подгонка дерева целиком под тренировочную выборку.
Его визуализация представлена ниже

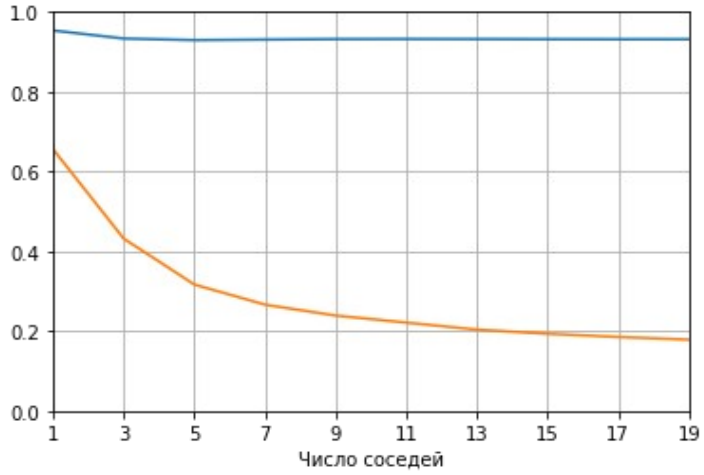


6. Сравнение классификаторов на банковских данных
В этой задаче сразу стоит обратить внимание на то, что предложенная обучающая и тестовая выборки разбалансированны (93:6). Поэтому оценка модели по простой точности малоинформативна. Для оценки классификаторов будем использовать PR-кривую и площадь под ней.
1) Классификатор k ближайших соседей используется предлагаемый библиотекой SkLearn. Несмотря на то, что он на первый взгляд очень успешнен в своей задаче (точность 95%), площадь под PR-кривой выдаёт его несовершенства.





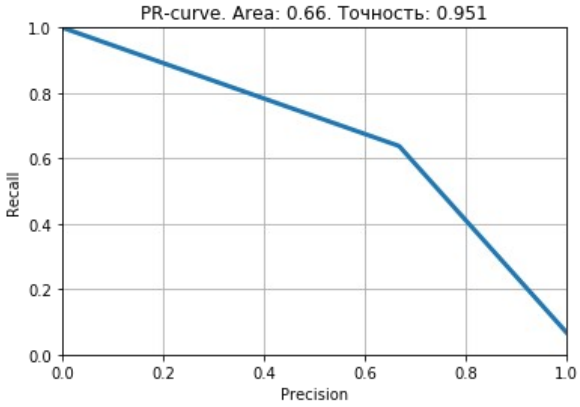
Площадь под кривой равна 0,66 и убывает увеличением числа соседей. Любопытно, что точность при этом почти не падает, что ещё раз подтверждает несостоятельность точности как метрики для нашего случая.



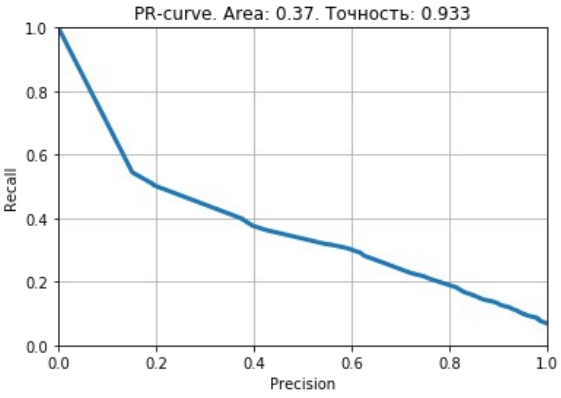
На рисунке синим проведена точность классификатора, а оранжевым – площадь под PR-кривой.

2) Классификатор на методе опорных векторов предполагалось проверить способом, описанным в пункте 4 настоящего отчёта, а именно построив несколько классификаторов и сравнив их между собой. Однако задача оказалась настолько объёмной, что построение даже одного классификатора с линейным ядром на полном предложенном датасете безвозвратно «подвешивало» интерпретатор, ввиду чего от идеи пришлось отказаться.

3) Классификатор на методе дерева решений.
Построим оба дерева: «неоптимизированное» и «оптимизированное» и проверим, какое лучше.



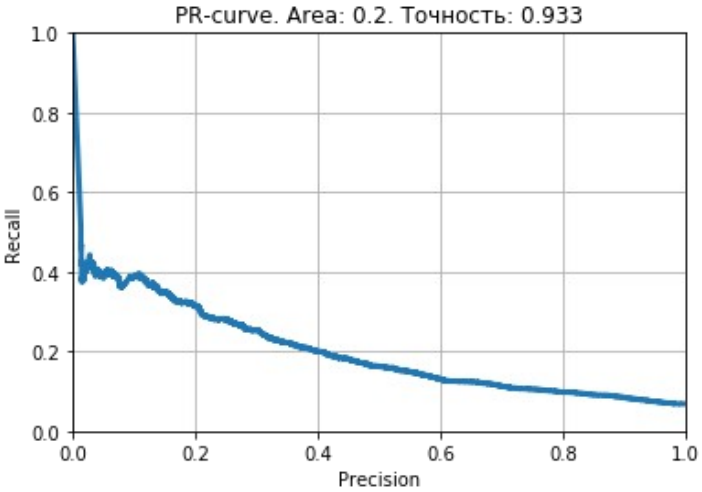
Неоптимизированное дерево



Оптимизированное дерево

Хоть точность упала незначительно, площадь под графиком сократилась вдвое. Теперь явно видно, чем мы жертвовали во имя простоты, накладывая ограничения на генерацию дерева.

4) Байесовский классификатор тоже участвует в тесте, а именно GaussianNB. Вот его PR-кривая:



Показатель точности этого классификатора выглядит хорошо, но не стоит обманываться: на нашем несбалансированном датасете можно получать 93% точности, причисляя все объекты к одному классу. На самом деле по критерию площади под PR-графиком это самый плохой классификатор из рассматриваемых.

Вывод

В проведённом тестировании классификаторов разного типа на несбалансированной задаче с одинаковым результатом лидируют классификатор kNN с K=1 и дерево решений на основе критерия Джини.

Приложение

1.1. Код самописного байесовского классификатора

```
# имена столбцов для переименовывания
col_names = ['x11', 'x12', 'x13', 'x21', 'x22', 'x23', 'x31', 'x32', 'x33', 'res']

# шаблон для подсчёта статистики
stat_patt = pos_stat = pd.DataFrame({'x': [0, 0, 0, 0, 0, 0, 0, 0, 0],
                                     'o': [0, 0, 0, 0, 0, 0, 0, 0, 0],
                                     'b': [0, 0, 0, 0, 0, 0, 0, 0, 0]}).T
stat_patt += 10**(-7)
stat_patt.columns = col_names[:-1]

# чтение данных
data = pd.read_csv('lab1_data/tic_tac_toe.txt', sep=',', header=None)
data.columns = col_names
data = data.sample(frac=1.)

pos_samp_num = len(data[data['res'] == 'positive'])
neg_samp_num = len(data[data['res'] == 'negative'])

# классы для предсказаний
classes = data['res'].unique()


def train(data):
    classifier = []
    for cls in classes:
        curr_data = data[data['res'] == cls].loc[:,'x11':'x33']
        #print (curr_data)
        stat = pd.DataFrame.copy(stat_patt)

        for _, row in curr_data.iterrows():
            for col in col_names[:-1]:
                stat.loc[row.loc[col], col] += 1

        classifier.append((cls, stat/len(curr_data)))

    return classifier

def test(classifier, test_sample):
    pred_column = {'predicted': [], 'probability': []}
    for _, sample in test_sample.iterrows():
        max_prob = 0.
        pred_label = 'none'
        for name, cls in classifier:
            curr_prob = 1.
            for col in col_names[:-1]:
                curr_prob *= cls.loc[sample.loc[col], col]
            if (curr_prob > max_prob):
                max_prob = curr_prob
                pred_label = name
        pred_column['predicted'].append(pred_label)
        pred_column['probability'].append(max_prob)
    return test_sample.join(pd.DataFrame(pred_column))


# головная программа

sample_sizes = [i/100 for i in range(1, 100, 3)]
accuracies = []

for test_size in sample_sizes:
```

```
    # формирование тестовой базы
    test_data = data[data['res'] == 'positive'][: int(test_size * pos_samp_num)]
    test_data = test_data.append(data[data['res'] == 'negative'][: int(test_size *
neg_samp_num)])
    test_data.reset_index(inplace=True, drop=True)

    # формирование тренировочной базы
    train_data = data[data['res'] == 'positive'][int(test_size * pos_samp_num):]
    train_data = train_data.append(data[data['res'] == 'negative'][int(test_size *
neg_samp_num):])
    train_data.reset_index(inplace=True, drop=True)

    # небольшие сведения
    print('Размер тренировочной базы выигрышей//проигрышей: {} // {}, доля тестов от
всего датасета: {}%'
          .format(len(train_data[train_data['res'] == 'positive']),
                  len(train_data[train_data['res'] == 'negative']),
                  test_size * 100))

    classifier = train(train_data)
    res = test(classifier, test_data)

    #
    acc = 0
    for _, row in res.iterrows():
        if (row['res'] == row['predicted']):
            acc += 1

    accuracies.append(acc / len(test_data))

plt.plot(sample_sizes, accuracies)
plt.xlabel('Доля тестов от датасета')
plt.ylabel('Точность на тестах')
plt.xlim(0, 1)
plt.grid()
plt.show()
```

1.2. Классификация спама

```
# классификатор спама
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

test_sizes = [i/100 for i in range(1, 100, 3)]
accs = []

data = pd.read_csv('lab1_data/spam.csv').drop(['Unnamed: 0'], axis=1).sample(frac=1.)
X_data = data.iloc[:, :-1].to_numpy()
Y_data = LabelEncoder().fit_transform(data[:]['type'].to_numpy())

for test_size in sample_sizes:

    X, X_t, Y, Y_t = train_test_split(X_data, Y_data, test_size=test_size,
shuffle=False)

    gnb = GaussianNB()
    gnb.fit(X, Y)

    accs.append(accuracy_score(Y_t, gnb.predict(X_t)))
```

```
pict = plt.plot(sample_sizes, accs)
plt.xlabel('Доля тестов от датасета')
plt.ylabel('Точность на тестах')
plt.xlim(0, 1)
plt.grid()
plt.show()
```

2.1. Метрики классификатора

```
# class +1
Na = 50
Da = 2
Max1 = 7
Max2 = 10
# class -1
Nb = 50
Db = 3
Mbx1 = 10
Mbx2 = 23

a_x1 = np.random.normal(Max1, Da, Na)
a_x2 = np.random.normal(Max2, Da, Na)
b_x1 = np.random.normal(Mbx1, Db, Nb)
b_x2 = np.random.normal(Mbx2, Db, Nb)

pict = plt.figure()
pict = plt.scatter(a_x1, a_x2, color='red')
pict = plt.scatter(b_x1, b_x2, color='blue')
pict = plt.grid(True)

plt.show()

# построение классификатора
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score, roc_curve,
auc, precision_recall_curve
#from sklearn.metrics import plot_precision_recall_curve
#from sklearn.metrics import plot_roc_curve

Y_data = np.array([1]*Na + [-1]*Nb)
X_data = np.concatenate((np.stack((a_x1, a_x2), axis=1), np.stack((b_x1, b_x2),
axis=1)), axis=0)

X, X_t, Y, Y_t = train_test_split(X_data, Y_data, test_size=0.2, shuffle=True)

classifier = GaussianNB()
classifier.fit(X, Y)

print(f'Точность  обуч//тест: {accuracy_score(classifier.predict(X),
Y)}//{accuracy_score(classifier.predict(X_t), Y_t)}')
conf_matr = pd.DataFrame(confusion_matrix(Y_t, classifier.predict(X_t)), index=['+1',
'-1'], columns=['+1', '-1'])
conf_matr

probs = classifier.predict_proba(X_t)
preds = probs[:, 1]

fpr, tpr, threshold = roc_curve(Y_t, preds)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc, linewidth=5)
```

```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid(True)
plt.show()

# PR-curve
precision, recall, thresholds = precision_recall_curve(Y_t, preds)
precision = np.insert(precision, 0, 0)
recall = np.insert(recall, 0 , 1)
plt.title('PR-curve')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.plot(precision, recall, '-', linewidth=5)
plt.grid(True)
plt.xlabel('Precision')
plt.ylabel('Recall')
plt.show()
```

3.1-2. Классификация стекла

```
data = pd.read_csv('lab1_data/glass.csv').sample(frac=1.)
test_data = data[: int(.2 * len(data))]
train_data = data[int(.2 * len(data)): ]
#data = data.drop('Id', axis=1)
#data = data.sample(frac=0.1)

# евклидово расстояние
def euql_dist(rowA , rowB ):
    return sum( (rowA['RI': 'Fe'] - rowB['RI': 'Fe'])**2)

# косинусная мера
def cos_dist(rowA, rowB):
    return sum(rowA['RI': 'Fe'] * rowB['RI': 'Fe']) / (sum(rowA['RI': 'Fe']**2) *
sum(rowB['RI': 'Fe']**2) )**0.5

# манхэттенское
def manh_dist(rowA, rowB):
    return sum(abs(rowA['RI': 'Fe'] - rowB['RI': 'Fe']))

# Чебышёва
def cheb_dist(rowA, rowB):
    return max(abs(rowA['RI': 'Fe'] - rowB['RI': 'Fe']))

def test(sample, K):
    distances = pd.DataFrame(columns=['Type', 'Euql_Dist', 'Cos_Dist', 'Manh_Dist',
'Cheb_Dist'], index=train_data.index, dtype=float)
    res = pd.Series(index=['Type', 'Pred_Euql', 'Pred_Cos', 'Pred_Manh', 'Pred_Cheb'],
dtype=float)

    # цикл распознавания
    for idx, row in train_data.iterrows():
        distances.at[idx, 'Type'] = row['Type']
        distances.at[idx, 'Cheb_Dist'] = cheb_dist(row, sample)
        distances.at[idx, 'Manh_Dist'] = manh_dist(row, sample)
        distances.at[idx, 'Euql_Dist'] = euql_dist(row, sample)
        distances.at[idx, 'Cos_Dist'] = cos_dist(row, sample)

    # заполняем итоговую таблицу
    distances.sort_values(by=['Euql_Dist'], inplace=True, ascending=True)
    try:
```

```
        res['Pred_Euql'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Euql'] = -1

    distances.sort_values(by=['Cos_Dist'], inplace=True, ascending=False)
    try:
        res['Pred_Cos'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Cos'] = -1

    distances.sort_values(by=['Manh_Dist'], inplace=True, ascending=True)
    try:
        res['Pred_Manh'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Manh'] = -1

    distances.sort_values(by=['Cheb_Dist'], inplace=True, ascending=True)
    try:
        res['Pred_Cheb'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Cheb'] = -1

    return res

# масивы для результатов точности
accs_euql = []
err_euql = []
accs_cos = []
err_cos = []
accs_manh = []
err_manh = []
accs_cheb = []
err_cheb = []

rep_len = len(test_data)

neighbours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

for K in neighbours:

    # отчётная таблица
    report = pd.DataFrame(columns=['Type', 'Pred_Euql', 'Pred_Cos', 'Pred_Manh',
'Pred_Cheb'],
                          index=test_data.index, dtype=float)

    # цикл подбора образцов
    for idx, sample in test_data.iterrows():
        samplereport = test(sample, K)

        #samplereport['Type'] = sample['Type']

        report.at[idx, 'Type'] = sample['Type']
        report.at[idx, 'Pred_Euql'] = samplereport['Pred_Euql']
        report.at[idx, 'Pred_Cos'] = samplereport['Pred_Cos']
        report.at[idx, 'Pred_Manh'] = samplereport['Pred_Manh']
        report.at[idx, 'Pred_Cheb'] = samplereport['Pred_Cheb']


    accs_euql.append(len(report[report['Pred_Euql'] == report['Type']]) / rep_len)
    err_euql.append(len(report[report['Pred_Euql'] == -1]) / rep_len)
    accs_cos.append(len(report[report['Pred_Cos'] == report['Type']]) / rep_len)
    err_cos.append(len(report[report['Pred_Cos'] == -1]) / rep_len)
    accs_manh.append(len(report[report['Pred_Manh'] == report['Type']]) / rep_len)
```

```
    err_manh.append(len(report[report['Pred_Manh'] == -1]) / rep_len)
    accs_cheb.append(len(report[report['Pred_Cheb'] == report['Type']]) / rep_len)
    err_cheb.append(len(report[report['Pred_Cheb'] == -1]) / rep_len)

gridsize = (1, 4)
x_ticks = [i for i in range(0, 21, 2)]
y_ticks = [i/10 for i in range(0, 11)]
y_labels = [i/10 for i in range(0, 11, 2)]
fig = plt.figure(figsize=(20,4))
pict1 = plt.subplot2grid(gridsize, (0, 0))
pict1 = plt.plot(neighbours, accs_euql)
pict1 = plt.plot(neighbours, err_euql)
pict1 = plt.grid(True)
pict1 = plt.xticks(x_ticks, x_ticks)
pict1 = plt.yticks(y_ticks)
pict1 = plt.xlim(min(neighbours), max(neighbours))
pict1 = plt.ylim(0, 1)
pict1 = plt.title('Евклидова метрика')

pict2 = plt.subplot2grid(gridsize, (0, 1))
pict2 = plt.plot(neighbours, accs_cos)
pict2 = plt.plot(neighbours, err_cos)
pict2 = plt.xticks(x_ticks, x_ticks)
pict2 = plt.yticks(y_ticks)
pict2 = plt.grid(True)
pict2 = plt.xlim(min(neighbours), max(neighbours))
pict2 = plt.ylim(0, 1)
pict2 = plt.title('Косинусная метрика')

pict3 = plt.subplot2grid(gridsize, (0, 2))
pict3 = plt.plot(neighbours, accs_manh)
pict3 = plt. plot(neighbours, err_manh)
pict3 = plt.xticks(x_ticks, x_ticks)
pict3 = plt.yticks(y_ticks)
pict3 = plt.grid(True)
pict3 = plt.xlim(min(neighbours), max(neighbours))
pict3 = plt.ylim(0, 1)
pict3 = plt.title('Манхэттенская метрика')

pict4 = plt.subplot2grid(gridsize, (0, 3))
pict4 = plt.plot(neighbours, accs_cheb)
pict4 = plt.plot(neighbours, err_cheb)
pict4 = plt.xticks(x_ticks, x_ticks)
pict4 = plt.yticks(y_ticks)
pict4 = plt.grid(True)
pict4 = plt.xlim(min(neighbours), max(neighbours))
pict4 = plt.ylim(0, 1)
pict4 = plt.title('Чебышёвская метрика')

plt.show()
```

3.3. Определение типа нового стекла

```
neighbours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
40]
new_glass = pd.Series([1.516, 11.7, 1.01, 1.19, 72.59, 0.43, 11.44, 0.02, 0.1],
                      index=['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe'])

# эта функция проверки просматривает ВСЕ доступные данны из glass.csv
def test(sample, K):
    distances = pd.DataFrame(columns=['Type', 'Euql_Dist', 'Cos_Dist', 'Manh_Dist',
'Cheb_Dist'], index=data.index, dtype=float)
    res = pd.Series(index=['Type', 'Pred_Euql', 'Pred_Cos', 'Pred_Manh', 'Pred_Cheb'],
```

```
dtype=float)

    # цикл распознавания
    for idx, row in data.iterrows():
        distances.at[idx, 'Type'] = row['Type']
        distances.at[idx, 'Cheb_Dist'] = cheb_dist(row, sample)
        distances.at[idx, 'Manh_Dist'] = manh_dist(row, sample)
        distances.at[idx, 'Euql_Dist'] = euql_dist(row, sample)
        distances.at[idx, 'Cos_Dist'] = cos_dist(row, sample)

    # заполняем итоговую таблицу
    distances.sort_values(by=['Euql_Dist'], inplace=True, ascending=True)
    try:
        res['Pred_Euql'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Euql'] = -1

    distances.sort_values(by=['Cos_Dist'], inplace=True, ascending=False)
    try:
        res['Pred_Cos'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Cos'] = -1

    distances.sort_values(by=['Manh_Dist'], inplace=True, ascending=True)
    try:
        res['Pred_Manh'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Manh'] = -1

    distances.sort_values(by=['Cheb_Dist'], inplace=True, ascending=True)
    try:
        res['Pred_Cheb'] = stat.mode(distances[0: K]['Type'])
    except:
        res['Pred_Cheb'] = -1

    return res

res = pd.DataFrame()

for K in neighbours:
    subreport = test(new_glass, K)
    subreport['Neighbours'] = K
    res = res.append(subreport, ignore_index=True).drop(['Type'], axis=1)

print('Таблица предсказаний типа стекла в зависимости от числа используемых соседей и
метрики расстояния:')
res.columns=['Число соседей', "Чебышевская", "Косинусная", "Евклидова",
"Манхэттенская"]
res
```

4.1. Метод опорных векторов. Линейное ядро.

```
def plot_single_graph(model_linear, X, y, *args, **kwargs):
    plt_title = kwargs.get('title', None)
    xlabel = kwargs.get('xlabel', None)
    ylabel = kwargs.get('ylabel', None)
    margin = kwargs.get('margin', .1)
    addit_x = kwargs.get('additional_x', [])
    addit_y = kwargs.get('additional_y', [])

    X0, X1 = [line[0] for line in X], [line[1] for line in X]
    add_x0, add_x1 = [line[0] for line in addit_x], [line[1] for line in addit_x]
    xx, yy = make_meshgrid(X0, X1, margin = margin)
```

```
    fig, ax = plt.subplots()

    plot_contours(ax, model_linear, xx, yy, cmap=plt.cm.coolwarm, alpha=0.6)
    ax.scatter(add_x0, add_x1, c=addit_y, cmap=plt.cm.coolwarm, s=20, edgecolors='w')
    ax.scatter(X0, X1, c=y, 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(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(plt_title)

    plt.show()

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

def make_meshgrid(x, y, h=.02, *args, **kwargs):
    margin = kwargs.get('margin', 0.1)
    x_min, x_max = min(x) - margin, max(x) + margin
    y_min, y_max = min(y) - margin, max(y) + margin
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

data = pd.read_csv('lab1_data/svmdata_a.txt', sep='\t')
test_data = pd.read_csv('lab1_data/svmdata_a_test.txt', sep='\t')

X_t = test_data[['X1', 'X2']].to_numpy()
Y_t = preprocessing.LabelEncoder().fit_transform(test_data[:]['Color'])

classifier = svm.SVC(kernel='linear')
X = data[['X1', 'X2']].to_numpy()
Y = data[:]['Color'].to_numpy()
Y = preprocessing.LabelEncoder().fit_transform(Y)

classifier.fit(X, Y)

plot_single_graph(classifier, X_t, Y_t, title='Линейный классификатор', additional_x=X,
additiona_y=Y)

print('Матрица ошибок:\n', confusion_matrix(Y_t, classifier.predict(X_t)))
print('Опорные векторы: \n', classifier.support_vectors_)
```

4.2. Переобучение

```
data = pd.read_csv('lab1_data/svmdata_b.txt', sep='\t')
test_data = pd.read_csv('lab1_data/svmdata_b_test.txt', sep='\t')
X = data[['X1', 'X2']].to_numpy()
Y = preprocessing.LabelEncoder().fit_transform(data[:]['Colors'])
X_t = test_data[['X1', 'X2']].to_numpy()
Y_t = preprocessing.LabelEncoder().fit_transform(test_data[:]['Colors'])

classifier = svm.SVC(kernel='linear', C=1000)
classifier.fit(X, Y)
res = classifier.predict(X)

plot_single_graph(classifier, X_t, Y_t, title='Тренировочная выборка. Штраф = 1K',
additional_x=X, additional_y=Y)
```

4.3-4. Проверка классификаторов на разных выборках

```
Cc = 2.

classifiers = [ (svm.SVC(kernel='linear', C=Cc, gamma='auto'), 'Линейный.'),
                (svm.SVC(kernel='poly', degree=2, C=Cc, gamma='auto'),
"Квадратичный."),
                (svm.SVC(kernel='poly', degree=3, C=Cc, gamma='auto'), "Кубический."),
                (svm.SVC(kernel='poly', degree=4, C=Cc, gamma='auto'), "IV степени."),
                (svm.SVC(kernel='poly', degree=5, C=Cc, gamma='auto'), "V степени."),
                (svm.SVC(kernel='sigmoid', C=Cc, gamma='auto'), "Сигмоидальный."),
                (svm.SVC(kernel='rbf', C=Cc, gamma='auto'), "Гауссов.")]

data = pd.read_csv('lab1_data/svmdata_d.txt', sep='\t')
test_data = pd.read_csv('lab1_data/svmdata_d_test.txt', sep='\t')
X = data[['X1', 'X2']].to_numpy()
Y = preprocessing.LabelEncoder().fit_transform(data[:]['Colors'])
X_t = test_data[['X1', 'X2']].to_numpy()
Y_t = preprocessing.LabelEncoder().fit_transform(test_data[:]['Colors'])

print('*точность выводится в формате точность_на_обучении//точность_на_тесте')

for cls, name in classifiers:
    cls.fit(X, Y)
    acc_l = accuracy_score(Y, cls.predict(X))
    acc_t = accuracy_score(Y_t, cls.predict(X_t))

    plot_single_graph(cls, X_t, Y_t, title=name+f' Точность: {acc_l}//{acc_t}.')
```

4.5. Переобучение

```
# e-д
# Снова то же самое, но в этот раз нужно не только визуализировать, но и
# добиться эффекта переобучения.
# Для этого установим большой штрафной параметр gammar
# По сути то же самое, что и предыдущее задание, но с другими данными

Cc = 1.
gamma = 100.

classifiers = [ (svm.SVC(kernel='linear', C=Cc, gamma=gamma), 'Линейный.'),
                (svm.SVC(kernel='poly', degree=2, C=Cc, gamma=gamma), "Квадратичный."),
                (svm.SVC(kernel='poly', degree=3, C=Cc, gamma=gamma), "Кубический."),
                (svm.SVC(kernel='poly', degree=4, C=Cc, gamma=gamma), "IV степени."),
                (svm.SVC(kernel='poly', degree=5, C=Cc, gamma=gamma), "V степени."),
                (svm.SVC(kernel='sigmoid', C=Cc, gamma=gamma), "Сигмоидальный."),
                (svm.SVC(kernel='rbf', C=Cc, gamma=gamma), "Гауссов.")]

data = pd.read_csv('lab1_data/svmdata_e.txt', sep='\t')
test_data = pd.read_csv('lab1_data/svmdata_e_test.txt', sep='\t')
X = data[['X1', 'X2']].to_numpy()
Y = preprocessing.LabelEncoder().fit_transform(data[:]['Colors'])
X_t = test_data[['X1', 'X2']].to_numpy()
Y_t = preprocessing.LabelEncoder().fit_transform(test_data[:]['Colors'])

print('*точность выводится в формате точность_на_обучении//точность_на_тесте')

for cls, name in classifiers:
    cls.fit(X, Y)
    acc_l = accuracy_score(Y, cls.predict(X))
    acc_t = accuracy_score(Y_t, cls.predict(X_t))


for cls, name in classifiers:
```

```
    acc_l = accuracy_score(Y, cls.predict(X))
    acc_t = accuracy_score(Y_t, cls.predict(X_t))
    plot_single_graph(cls, X, Y, title=name+f'(трен.) Точность:
{acc_l:.3}//{acc_t:.3}.', margin=.5)
    plot_single_graph(cls, X_t, Y_t, title=name+f' (тест) Точность:
{acc_l:.3}//{acc_t:.3}.', margin=.5)
```

5.1. Деревья. Визуализация и оптимизация

```
from sklearn import tree
import graphviz
from sklearn.metrics import accuracy_score

data = pd.read_csv('lab1_data/glass.csv').sample(frac=1.)
test_data = data[0:int(np.trunc(len(data)*0.2))]
data = data[int(np.trunc(len(data)*0.2)):]
X = data.loc[:, 'RI': 'Fe'].to_numpy()
Y = data[:]['Type'].to_numpy()
X_t = test_data.loc[:, 'RI': 'Fe'].to_numpy()
Y_t = test_data[:]['Type']

classifier = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=0.05)
classifier.fit(X, Y)

graph_data = tree.export_graphviz(classifier, out_file=None,
feature_names=data.columns[1: -1], \
                                  class_names=[f'{i}' for i in np.unique(Y)],
filled=True, impurity=False)
graph = graphviz.Source(graph_data)
graph.render('first_tree', format='png', view=False)
#graph  # большой и страшный граф, кооторый попросту не вмещается сюда.

# Поэтому тут его мини-версия
fig, ax = plt.subplots(figsize=(16,9))
tree.plot_tree(classifier, ax=ax)
plt.show()

print(f'Точность {accuracy_score(Y, classifier.predict(X))}//{accuracy_score(Y_t,
classifier.predict(X_t))}')

#  Ограничим максимальную глубину шестью
classifier = tree.DecisionTreeClassifier(max_depth=6, min_samples_split=2)
classifier.fit(X, Y)

fig, ax = plt.subplots(figsize=(16,9))
tree.plot_tree(classifier, ax=ax)
plt.show()

print(f'Точность {accuracy_score(Y, classifier.predict(X)):.2}//{accuracy_score(Y_t,
classifier.predict(X_t)):.2}')

# Попробуем изменить критерий. По умолчанию используется джини, поставим энтропический.
# Глубину при этом ограничивать не будем

classifier = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2,
criterion='entropy')
classifier.fit(X, Y)

fig, ax = plt.subplots(figsize=(14,7))
tree.plot_tree(classifier, ax=ax)
plt.show()
```

```
print(f'Точность {accuracy_score(Y, classifier.predict(X)):.2}//{accuracy_score(Y_t,
classifier.predict(X_t)):.2}')
```

5.2. Классификация спама деревом

```
from sklearn import preprocessing

data = pd.read_csv('lab1_data/spam7.csv').sample(frac=1.)
test_data = data.loc[:int(len(data)*0.2)]
data = data.loc[int(len(data)*0.2):]

X = data.iloc[:, :-1].to_numpy()
Y = preprocessing.LabelEncoder().fit_transform(data[:]['yesno'])
X_t = test_data.iloc[:, :-1].to_numpy()
Y_t = preprocessing.LabelEncoder().fit_transform(test_data[:]['yesno'])

classifier = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=0.04,
criterion='gini', \
                                         max_features=None, class_weight='balanced')
# 1. Любопятно, что значение параметра min_samples_split=.04 не только генерирует
покрасивей дерево, но и
#    поднимает точность на тестах чуть-чуть
# 2. Лучшая точность достигается при значении параметра max_features=None
# 3. class_weight='ballanced' позволило на .01 поднять точность на тестах, но дерево
снова разбалансировалось
#
#
#
classifier.fit(X, Y)

fig, ax = plt.subplots(figsize=(16,9))
tree.plot_tree(classifier, ax=ax)
plt.show()

print(f'Точность {accuracy_score(Y, classifier.predict(X)):.2}//{accuracy_score(Y_t,
classifier.predict(X_t)):.2}')
```

6. Тесты разных классификаторов

```
def plot_pr_curve(classifier, X_t, Y_t, comment):

    probs = classifier.predict_proba(X_t)
    preds = probs[:, 1]

    precision, recall, thresholds = precision_recall_curve(Y_t, preds)
    #precision = np.insert(precision, 0, 0)
    #recall = np.insert(recall, 0 , 1)
    plt.title(f'PR-curve. Area: {auc(recall, precision):.3}. {comment}')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.plot(recall, precision, '-', linewidth=3)
    plt.grid(True)
    plt.xlabel('Precision')
    plt.ylabel('Recall')
    plt.show()

test_data = pd.read_csv('lab1_data/bank_scoring_test.csv', sep='\t')
data = pd.read_csv('lab1_data/bank_scoring_train.csv', sep='\t')
X = data.iloc[:, 1:].to_numpy()
Y = data.iloc[:, 0].to_numpy()
X_t = test_data.iloc[:, 1:].to_numpy()
Y_t = test_data.iloc[:, 0].to_numpy()
```

```
# classifiers tests
# 1. kNN
from sklearn.neighbors import KNeighborsClassifier
neighbours = [1, 3 ,5 ,7, 9, 11, 13, 15, 17, 19 ]
accs = []
areas = []
for n in neighbours:

    knn = KNeighborsClassifier(n_neighbors=n, n_jobs=4, algorithm='auto'  )
    knn.fit(X, Y)
    #print(f'{n} neighbours, accuracy: {accuracy_score(knn.predict(X_t), Y_t)}')
    accs.append(accuracy_score(knn.predict(X_t), Y_t))
    plot_pr_curve(knn, X_t, Y_t, f'K = {n}')

    probs = knn.predict_proba(X_t)
    preds = probs[:, 1]
    precision, recall, thresholds = precision_recall_curve(Y_t, preds)
    areas.append(auc(recall, precision))


plt.plot(neighbours, accs)
plt.plot(neighbours, areas)
plt.grid(True)
plt.ylim(.0, 1)
plt.xlim(1, 19)
plt.xlabel('Число соседей')
#plt.ylabel('Точность/Площадь под PR-кривой')
plt.xticks(neighbours, neighbours)
plt.show()

# 2. SVC
# Эта штука даже один классификатор не может осилить, не то чтобы сравнить несколько :/
# Кроме того, этот код вешает питоновское ядро, и его приходтся насильно
перезапускать :/

Cc = 1.
classifiers = [ (svm.SVC(kernel='linear', C=Cc, gamma='auto'), 'Линейный.')
#                (svm.SVC(kernel='poly', degree=2, C=Cc, gamma='auto'),
"Квадратичный."),
#                (svm.SVC(kernel='poly', degree=3, C=Cc, gamma='auto'), "Кубический."),
#                (svm.SVC(kernel='poly', degree=4, C=Cc, gamma='auto'), "IV степени."),
#                (svm.SVC(kernel='poly', degree=5, C=Cc, gamma='auto'), "V степени."),
#                (svm.SVC(kernel='sigmoid', C=Cc, gamma='auto'), "Сигмоидальный."),
#                (svm.SVC(kernel='rbf', C=Cc, gamma='auto'), "Гауссов.")
              ]

for cls, name in classifiers:
    cls.fit(X, Y)
    acc_t = accuracy_score(Y_t, cls.predict(X_t))

    plot_single_graph(cls, X_t, Y_t, title=name+f' Точность: {acc_t}')

# 3. Дерево решений
from sklearn import tree
from sklearn.metrics import accuracy_score , auc

tr = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=2, criterion='gini')
tr.fit(X, Y)

#fig, ax = plt.subplots(figsize=(16,9))
#tree.plot_tree(tr, ax=ax)
#plt.show()
```

```
print(f'Точность дерева: {accuracy_score(tr.predict(X_t),
Y_t)}//{accuracy_score(tr.predict(X_t), Y_t)}')
print(f'Accuracy matrix: \n {confusion_matrix(Y_t, tr.predict(X_t))}')
#plot_single_graph(tr, X_t, Y_t, ' full tree') # не хватает памяти

probs = tr.predict_proba(X_t)
preds = probs[:, 1]

precision, recall, thresholds = precision_recall_curve(Y_t, preds)
plt.title(f'PR-curve. Area: {auc(recall, precision):.3}. Точность:
{accuracy_score(tr.predict(X_t), Y_t):.3}')
#precision = np.insert(precision, 0, 0)
#recall = np.insert(recall, 0 , 1)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.plot(recall, precision, '-', linewidth=3)
plt.grid(True)
plt.xlabel('Precision')
plt.ylabel('Recall')
plt.show()


tr = tree.DecisionTreeClassifier(max_depth=None, min_samples_split=0.04,
criterion='gini')
tr.fit(X, Y)

#fig, ax = plt.subplots(figsize=(16,9))
#tree.plot_tree(tr, ax=ax)
#plt.show()

print(f'Точность дерева: {accuracy_score(tr.predict(X_t),
Y_t)}//{accuracy_score(tr.predict(X_t), Y_t)}')
print(f'Accuracy matrix: \n {confusion_matrix(Y_t, tr.predict(X_t))}')
#plot_single_graph(tr, X_t, Y_t, ' simple tree') #не хватило памяти

probs = tr.predict_proba(X_t)
preds = probs[:, 1]
precision, recall, thresholds = precision_recall_curve(Y_t, preds)
plt.title(f'PR-curve. Area: {auc(recall, precision):.3}. Точность:
{accuracy_score(tr.predict(X_t), Y_t):.3}')
#precision = np.insert(precision, 0, 0)
#recall = np.insert(recall, 0 , 1)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.plot(recall, precision, '-', linewidth=3)
plt.grid(True)
plt.xlabel('Precision')
plt.ylabel('Recall')
plt.show()

# 4. Наивный байесовский классификатор
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X, Y)

probs = gnb.predict_proba(X_t)
preds = probs[:, 1]
precision, recall, thresholds = precision_recall_curve(Y_t, preds)
plt.title(f'PR-curve. Area: {auc(recall, precision):.2}. Точность:
{accuracy_score(tr.predict(X), Y):.3}')
#precision = np.insert(precision, 0, 0)
#recall = np.insert(recall, 0 , 1)
```

```
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.plot(recall, precision, '-', linewidth=3)
plt.grid(True)
plt.xlabel('Precision')
plt.ylabel('Recall')
plt.show()


print(f'Точность байесовского классификатора: {accuracy_score(gnb.predict(X),
Y)}//{accuracy_score(gnb.predict(X_t), Y_t)}')
```