**Лабораторная работа №1**

МЕТОДЫ КЛАССИФИКАЦИИ

По дисциплине «Машинное обучение»

Выполнил студент

группы 3530904/80102: Шерман М.Л.

Преподаватель: Селин И.А.

Оглавление

[Задачи 3](#_Toc66113428)

[Пункт 1 5](#_Toc66113429)

[Пункт 2 6](#_Toc66113430)

[Пункт 3 7](#_Toc66113431)

[Подпункт A 7](#_Toc66113432)

[Подпункт B 7](#_Toc66113433)

[Подпункт C 8](#_Toc66113434)

[Пункт 4 8](#_Toc66113435)

[Подпункт A 8](#_Toc66113436)

[Подпункт B 9](#_Toc66113437)

[Подпункт C 10](#_Toc66113438)

[Подпункт D 11](#_Toc66113439)

[Подпункт E 12](#_Toc66113440)

[Gamma=”Scale” 12](#_Toc66113441)

[Gamma=’auto’ 13](#_Toc66113442)

[Gamma=1 14](#_Toc66113443)

[Gamma=10 15](#_Toc66113444)

[Gamma=100 16](#_Toc66113445)

[Итог 16](#_Toc66113446)

[Пункт 5 17](#_Toc66113447)

[Подпункт A 17](#_Toc66113448)

[Подпункт B 18](#_Toc66113449)

[Пункт 6 19](#_Toc66113450)

[Вывод 22](#_Toc66113451)

[Приложение 23](#_Toc66113452)

# 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, ROС 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.
   1. Постройте графики зависимости ошибки классификации от количества ближайших соседей.
   2. Определите подходящие метрики расстояния и исследуйте, как тип метрики расстояния влияет на точность классификации.
   3. 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:
   1. Постройте алгоритм метода опорных векторов с линейным ядром. Визуализируйте разбиение пространства признаков на области с помощью полученной модели ([пример визуализации](https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html)). Выведите количество полученных опорных векторов, а также матрицу ошибок классификации на обучающей и тестовой выборках.
   2. 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?
   3. 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.
   4. 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.
   5. 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:
   1. 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.
   2. 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.

# Пункт 1

Размер обучающей и тестовой выборки имеет большое значения для создания качественной модели, особенно когда количество данных ограничено. Даже хорошая модель, от недостатка обучающих данных может недообучиться, а при излишнем изобилии переобучиться. Поэтому правильный выбор соотношения обучающей и тестовой выборки важен.

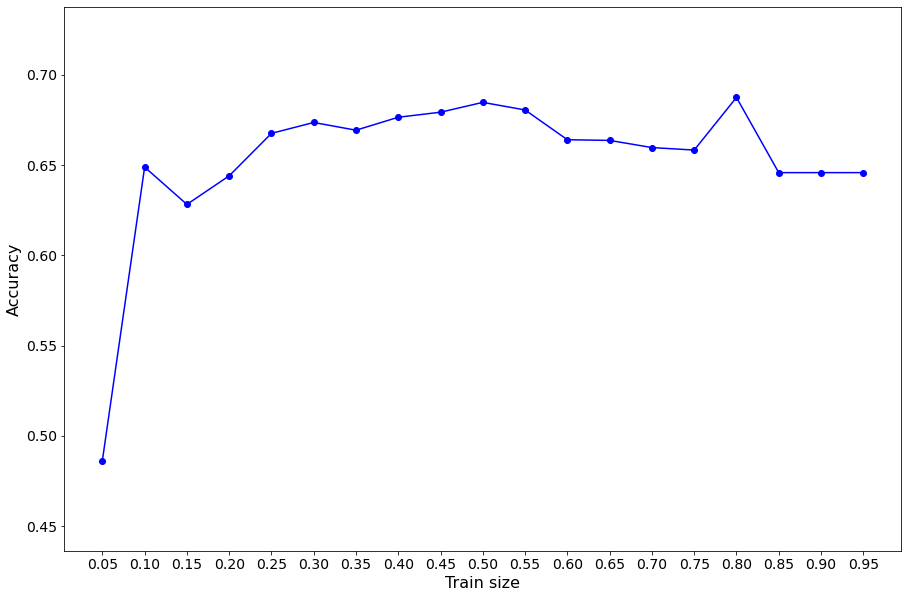
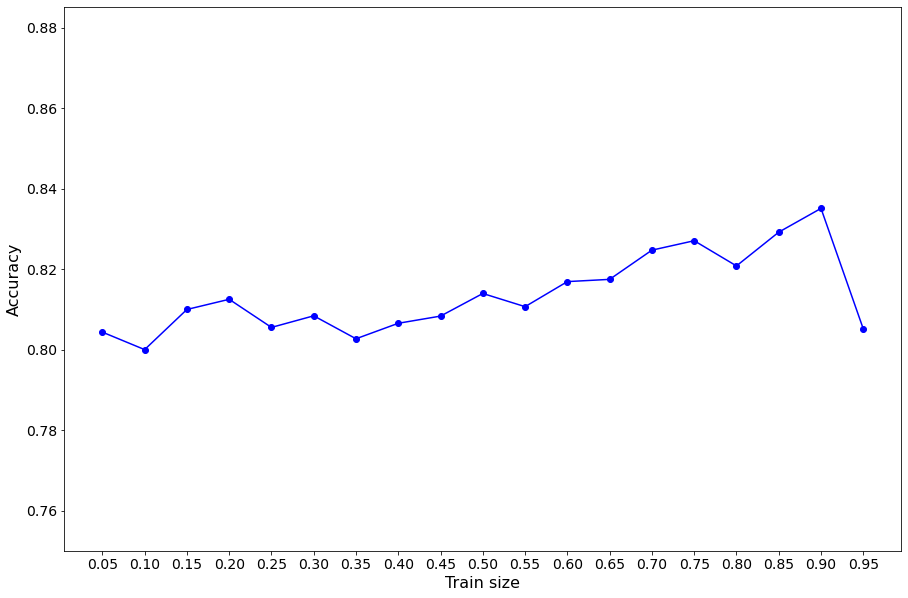
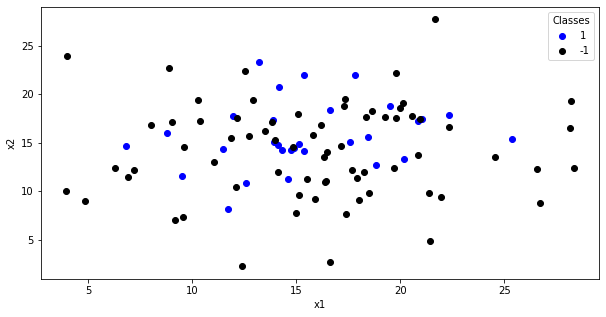
Рассмотрим графики зависимости точности классификации от объёма обучающей выборки.

Рисунок . Зависимость точности модели от объёма обучающей выборки. Датасет "Спам в e-mail сообщениях

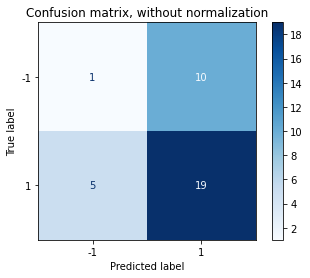
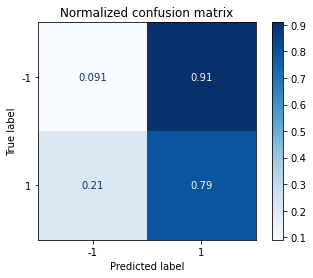
Рисунок . Зависимость точности модели от объёма обучающей выборки. Датасет "Крестики нолики"

Для датасета «Крестики-нолики» лучшим разбиением на обучающую и тестовую выборку оказалось 0.8 и 0.2 соответственно (от количества данных), а для датасета о спаме 0.9 и 0.1, однако я предположу, что во втором датасете имеет место быть особенность самих данных, так как обычно оптимальным соотношением является как раз 80% для обучающей и 20% для тестовой выборок. Если не брать во внимание этот выброс, то наилучшем решением будет задействовать 70-80% от изначальных данных на обучение модели.

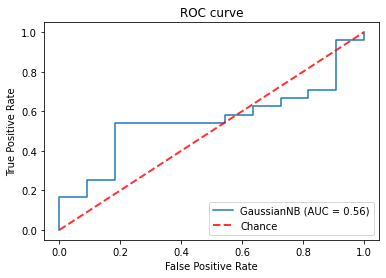
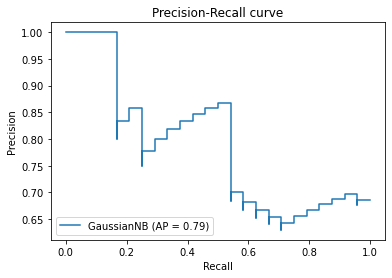
# Пункт 2

В данной части лабораторной необходимо было сгенерировать 100 точек с двумя признаками X1 и X2, где 30% относилось бы к классу 1, а остальные 70 к классу -1. Полученные данные представлены на графике:

Построим Байесовский классификатор и оценим качество классификации с помощью различных методов. Изначально рассмотрим матрицу ошибок.



Получившаяся матрица ошибок свидетельствует о достаточно посредственном качестве модели. Она хорошо распознаёт объекты класса -1 и практически не может определить класс 1. Такую же ситуацию демонстрируют ROC и PR кривые.



Столь низкое качество модели обусловлено сильной зашумлённостью данных. Байесовский классификатор не рассчитан на данные, которые настолько слабо отличаются друг от друга. Решением данной проблемы может стать подбор более комплексной модели, которая сможет распознать столь схожие классы данных.

# Пункт 3

## Подпункт A

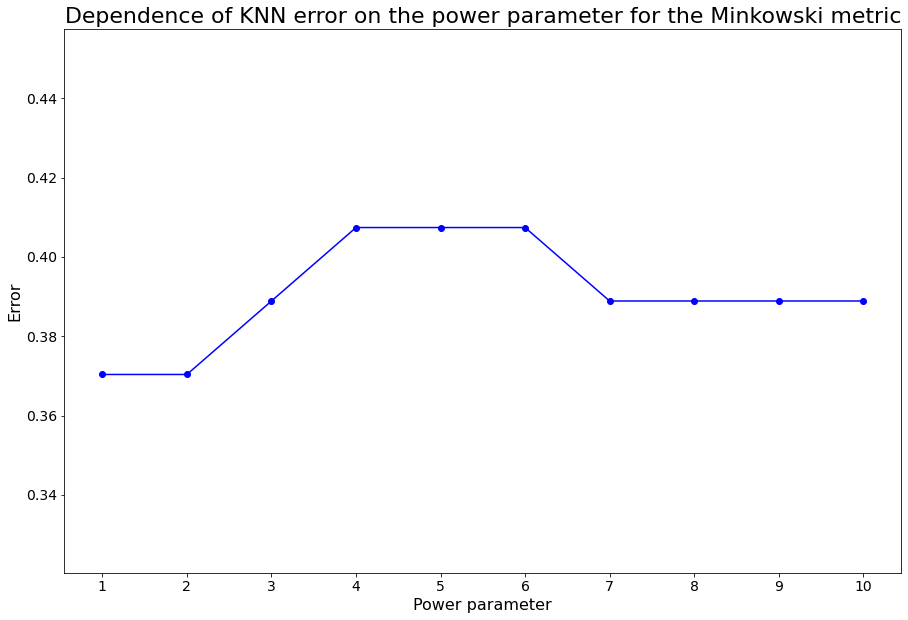
Для представленного набора данных оптимальным количеством соседей для метода KNN будет 1. Достаточно интересный случай, который как раз удалось изучить благодаря рассмотрению влияния количества соседей на точность метода. По-видимому, классы достаточно хорошо обособлены друг от друга, что и позволило методу показать наилучший результат при 1 соседе.

## Подпункт B

По умолчанию, в реализации классификатора KNN библиотеки scikit-learn в качестве метрики расстояния используется метрика Минковского.

Её формула выглядит так

где *p* задаётся пользователем. С помощью изменения параметра *p* можно изменять способ вычисления расстояния. Также особенностью метрики Минковского является то, что при *p*=1 мы получим Манхэттенскую метрику, а при *p*=2 метрику Чебышёва.

Рассмотрим получившиеся результаты:

Таким образом, параметры p=1 и p=2 (или Манхэттенская и Чебышёская метрики) являются наилучшими для предоставленного датасета.

## Подпункт C

Необходимо предсказать, 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.

Для более точного предсказания создадим три классификатора:

1. с параметрами по умолчанию;
2. с количеством соседей 1 и *p*=1;
3. с количеством соседей 15 и *p*=2 (данное количество соседей выбрано в связи с неплохими результатами, на фоне похожих значений).

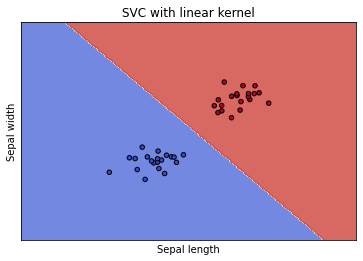
Все классификаторы отнесли представленное стекло к классу 5.

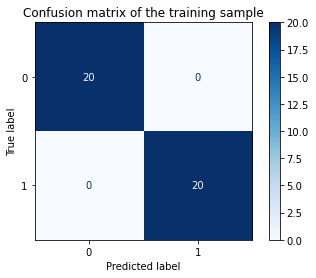
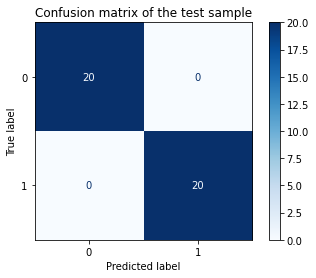
# Пункт 4

## Подпункт A

Построим алгоритм метода опорных векторов с линейным ядром, а затем визуализируем разбиение пространства признаков на области с помощью полученной модели.

В результате получим:

Данные из представленного набора очень хорошо различимы, что и показывает нам модель. Такая «бесшумная» модель, безусловно, подарок разработчику, однако в реальном мире с такими «чистыми» данными почти не сталкиваются. Теперь рассмотрим матрицы ошибок для обучающей и тестовой выборок:



Как и следовало ожидать, метод опорных векторов прекрасно справился с поставленной задачей классификации и не совершил ни одной ошибки. Также для каждого класса понадобилось по 3 вектора.

## Подпункт B

Воспользуемся такой же моделью, как и в предыдущем подпункте, и постараемся выяснить, как влияет изменение штрафного параметра на модель.

Возьмём значения штрафного параметра [1, 10, 100, 1000, 10000].

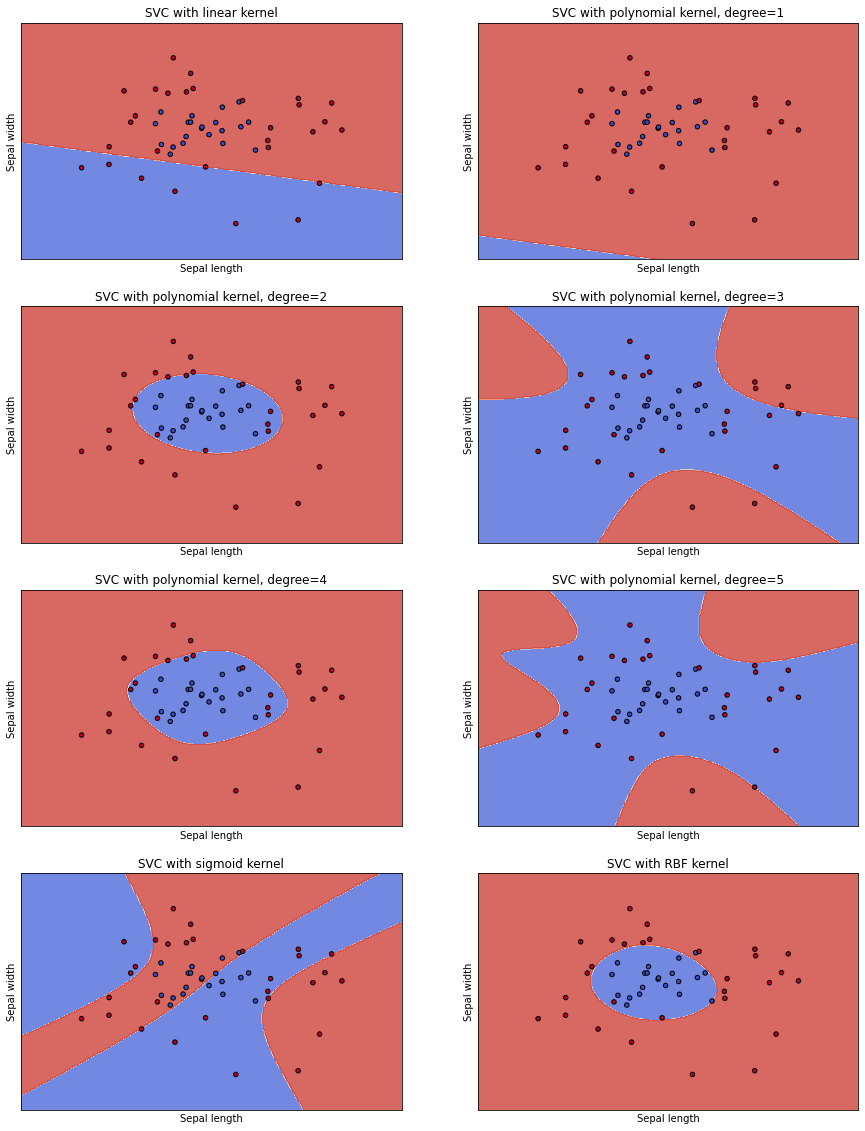
Получаем следующие результаты:

На обучающей выборке [0.98, 0.98, 0.98, 0.98, 1.0, 1.0].

На тестовой [1.0, 1.0, 1.0, 1.0, 0.94, 0.94].

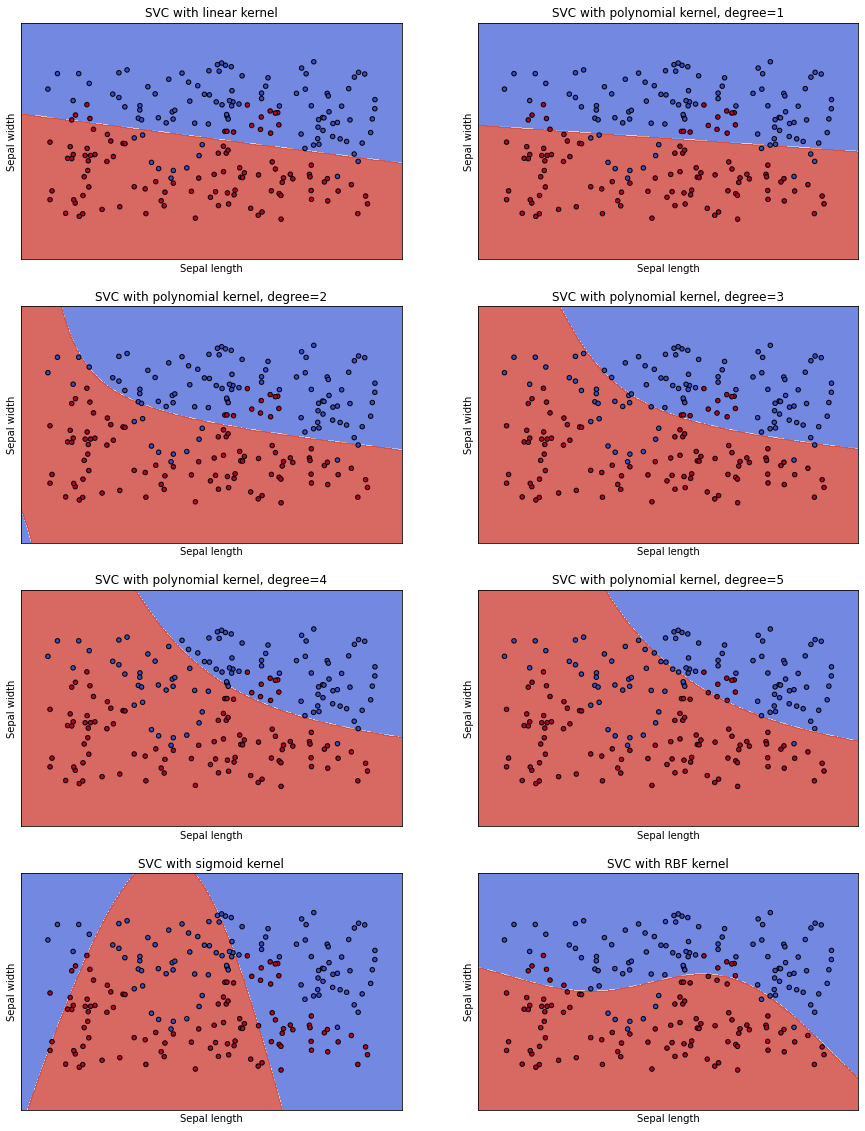
По умолчанию в реализации метода SVM библиотеки scikit-learn штрафной параметр равен 1. Для данной модели этот параметр оптимальный, так как при его увеличении, строгость регуляризации падает. Чем меньше строгость, тем больше модель переобучается на train-выборке, что негативно сказывается на качестве модели (лучше всего это видно при штрафных параметрах 1000 и 10000). Поэтому лучше пожертвовать точностью на обучающей выборке и позволить модели вычленить общие закономерности, а не частные особенности.

## Подпункт C

Рассмотрим влияние различных ядер на точность классификации метода опорных векторов. Будем использовать следующие ядра: линейное, полиномиальное степеней 1-5, сигмоидальная функция, гауссово.

Излишнее увеличение степени не идёт на пользу методу SVM с полиномиальным ядром. Однако и слишком простые алгоритмы (ядра) не дают нужной степени точности. Наилучшим ядром, среди всех использованных, оказалось гауссово ядро.

## Подпункт D

Задание данного подпункта аналогично предыдущему, за исключением нового набора данных. Рассмотрим работу знакомых классификаторов на новых данных.

Качество ядер схоже с предыдущим случаем, за исключением того, что теперь также неплохо себя показали линейное ядро и полиномиальное со степенью 1.

## Подпункт E

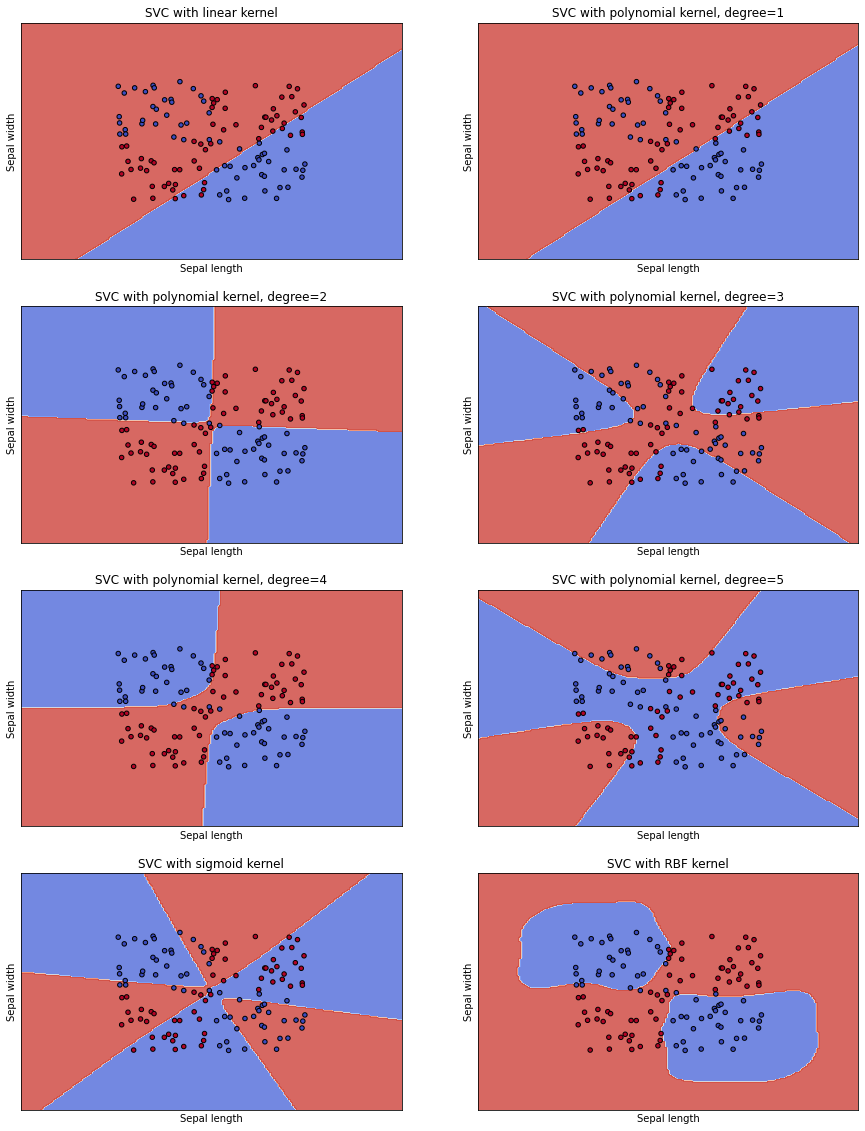
В дополнение к предыдущему подпункту, теперь необходимо рассмотреть влияние параметра gamma на метод опорных векторов. Этот параметр присутствует только в следующих ядрах: полиномиальное, сигмоидальная функция, гауссово. Однако, для наглядности, продемонстрируем также результат линейного ядра.

### Gamma=”Scale”

### Gamma=’auto’

### Gamma=1

### Gamma=10



### Gamma=100

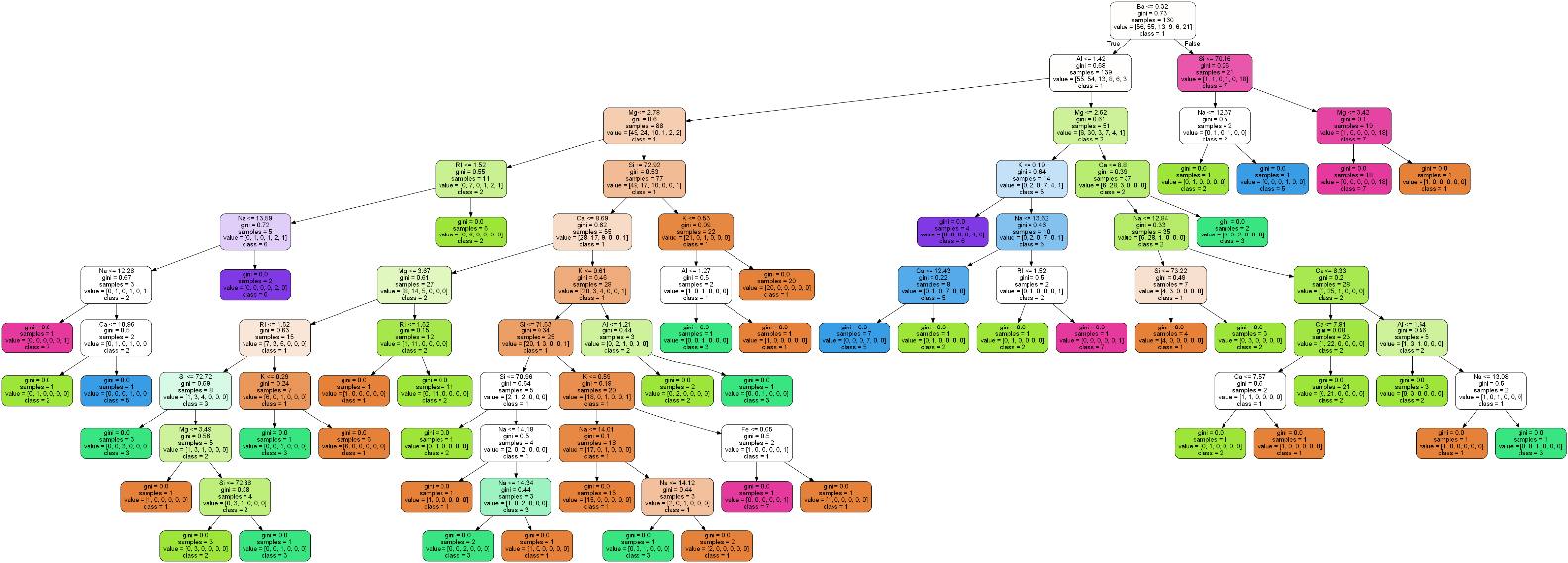
### Итог

Параметр гаммы «scale» хорошо разделяет плоскость на классы, что позволяет их использовать в большинстве реализаций SVM. Параметры «auto» и 1 показывают себя хуже, однако на некоторых ядрах демонстрируют более-менее приличные результаты, чего нельзя сказать о параметрах 10 и 100. При увеличении параметра гамма всё более резко виднеется перспектива переобучения модели. Особенно хорошо это видно при использовании гауссовского ядра. Также, при увеличении данного параметра становится невозможным применение полиномиального ядра с нечётной степенью (за исключением 1), что в некоторых случаях может являться серьёзной проблемой.

Таким образом, в большинстве случаев следует обойтись параметром «scale», что и предлагает библиотека scikit-learn в методе SVM по умолчанию.

# Пункт 5

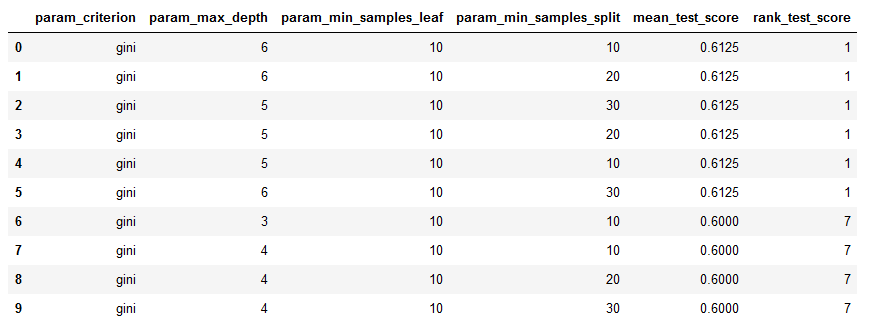
## Подпункт A

Рассмотрим в действии новый классификатор– дерево решений. Для этого возьмём подготовленный набор данных и обучим модель. Рассмотрим полученную точность: на обучающей выборке она составила 1, а на тестовой примерно 0.63. Таким образом можно сделать вывод, что дерево решений – классификатор, который очень легко переобучается, если не давать строгие ограничения. Рассмотрим визуализацию полученной модели:

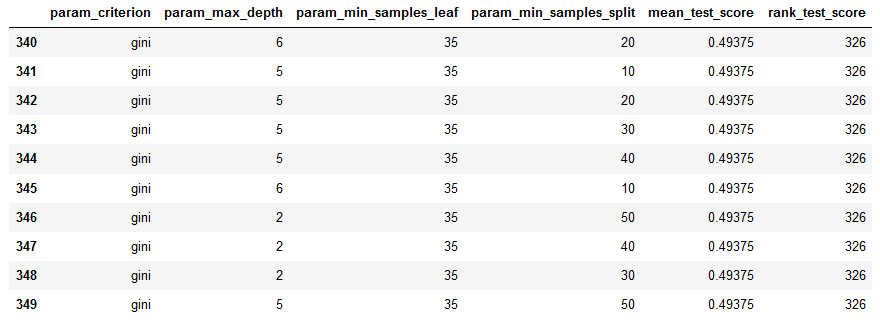
Полученное дерево решений оказалось чересчур избыточным. Понятно, что без дополнительных ограничений, качество данного классификатора катастрофически падает. Рассмотрим, как точность классификации зависит от различных параметров дерева. Будем изменять следующие параметры:

* Критерий: gini, entropy
* Глубина дерева: [2, 6]
* Минимальный размер узла: 10, 20, 30, 40, 50
* Минимальный размер листьев: 10, 15, 20, 25, 30, 35, 40

В качестве итога посмотрим на 10 наилучших комбинаций параметров и 10 наихудших.

Наилучшие:

Наихудшие:

При беглом взгляде самым значимым параметром оказывается min\_samples\_leaf (минимальный размер листьев). Он должен быть не более 10, затем следует min\_samples\_split (Минимальный размер узла). У всех наилучших данных он не превышает 30. Однако не стоит забывать и о глубине дерева. Безусловно при значении 5 и 6 мы получаем лучшие значения, но также очень неплохо смотрится на их фоне модель с параметрами max\_depth=3, min\_samples\_leaf=10, min\_samples\_split=10, criterion=entropy. Хоть результат немного хуже, но глубина дерева равно всего лишь 3, что выглядит достаточно интересно.

Если взглянуть на наихудшие комбинации, то снова значимым является параметр min\_samples\_leaf. Достаточно логично, ведь всего в представленном датасете 214 объектов, в то время как 35\*7=245, из-за этого часто возникает ситуация, когда некоторые классы, в связи с их большим объёмом, «поглощают» наименее распространённые. В связи с этим для данного датасета критически важно это ограничение. Для примера возьмём следующие два набора параметров:

* max\_depth=5, min\_samples\_leaf=10, min\_samples\_split=10, criterion=entropy
* max\_depth=5, min\_samples\_leaf=35, min\_samples\_split=10, criterion=entropy

Единственное их отличие в параметре min\_samples\_leaf, но при этом классификатор, построенный на основе первого имеет один из наилучших показателей, а второй – из наихудших. Также плохие результаты показали деревья, с глубиной 2 и слишком большим размером листьев и узлов.

Обучив модель на наилучшем из показателей, находим точность – 0.72. Очень приятная прибавка в 9% по сравнению с переобученным деревом.

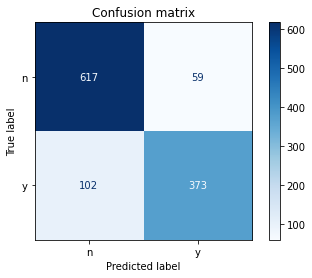
## Подпункт B

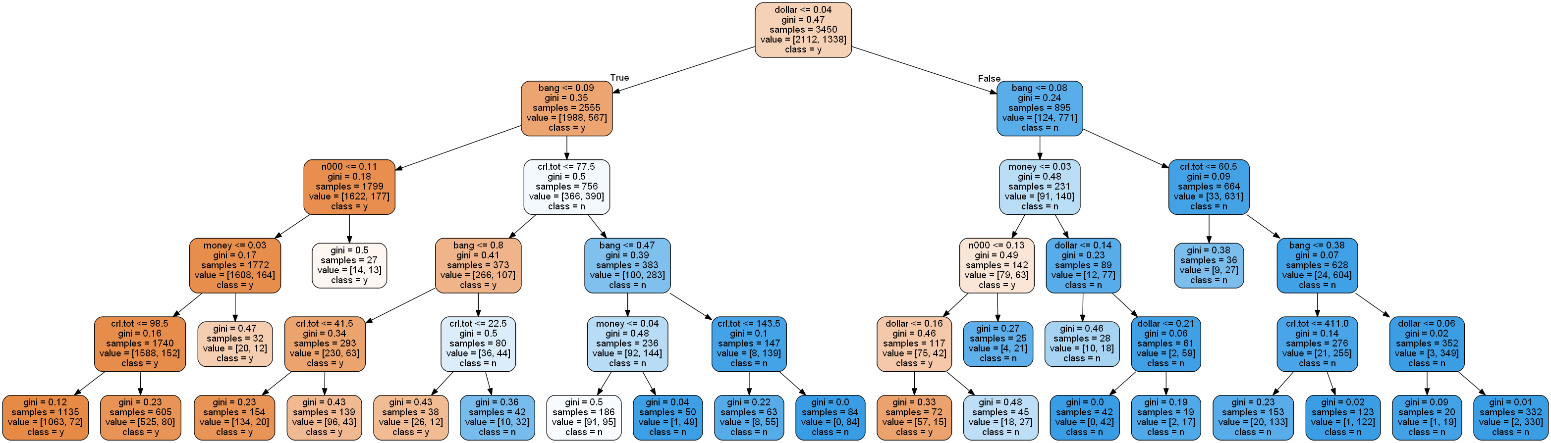
Необходимо построить оптимальное дерево классификации для нового набора данных. В данном датасете уже более 4000 объектов, что позволяет более детально исследовать «обобщение» модели. С помощью функции GridSearchCV из библиотеки Scikit-Learn подберём наилучшие параметры для модели. Будем искать среди следующих параметров:

* Критерий: gini, entropy
* Глубина дерева: [2, 6]
* Минимальный размер узла: 25, 40, 50, 60, 75, 100
* Минимальный размер листьев: 10 + 2 \* k, где k ∈ [1, 19]

В результате, наилучшим оказался классификатор со следующими параметрами:

* Критерий: gini
* Глубина дерева: 5
* Минимальный размер узла: 40
* Минимальный размер листьев: 18

При данных параметрах точность метода составляет 0.86, что весьма неплохо. По матрице ошибок видно, что дерево решений гораздо лучше предсказывает классы «no», чем «yes».

Рассмотрим полученное дерево решений:

Наиболее влияющими признаками на мой взгляд являются «dollar» и «bang». Первый признак разделяет данные на две группы, где есть подавляющее большинство одного из классов, «bang» встречается достаточно часто и позволяет более точечно определить класс объекта.

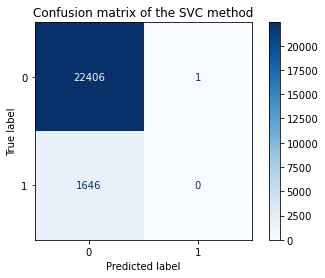
В результате у нас получилась достаточно неплохая модель с приемлемой точностью, но со слишком большой ошибкой второго рода. Если пользователь готов мириться с ошибками второго рода, то данный классификатор ему подойдёт.

# Пункт 6

В данном задании необходимо рассмотреть не менее двух систем, которые должны сделать вывод и выдаче или невыдаче кредита физическому лицу. Попробуем использовать следующие классификаторы:

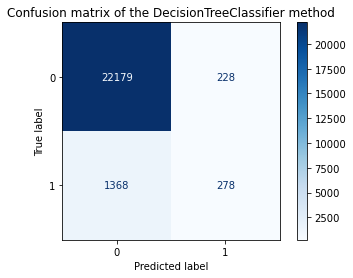
* Метод опорных векторов
* K ближайших соседей
* Дерево решений

Первоначально рассмотрим метода опорных векторов. В предыдущих пунктах достаточно хорошо себя показала реализация этого метода из библиотеки scikit-learn с параметрами по умолчанию. Воспользуемся ей и посмотрим на точность: 0.932. Довольно неплохой результат, однако для уверенности посмотрим на матрицу ошибок:

Матрица ошибок показывает всю трагичность ситуации. Метод опорных векторов определил все (за исключением одного) объекты к классу 0. Высокая точность обусловлена подавляющим большинством объектов с классом 0. Интерпретируя это на реальную систему банка, данная модель говорит, что ни у кого не будет наблюдаться ухудшения финансовой ситуации. Такая система ни в коем случае не должна быть использована банком.

Рассмотрим другой классификатор – дерево решений. В качестве ограничений для дерева будем использовать следующие параметры:

* Критерий: gini
* Глубина дерева: 10
* Минимальный размер узла: 100
* Минимальный размер листьев: 10

Достаточно маленькие значения размера узла и размера листьев выбраны специально, так как количество объектов с классом 1 мало и эти ограничения помогут в более тщательном поиске. В результате получившееся точность 0.934. Почти не отличается от метода опорных векторов. Взглянем на матрицу ошибок:

Ситуация лучше, по сравнению с предыдущим классификатором, однако всё также большое число объектов ошибочно отнесено к классу 0, что для банка критически важно. Хоть и результат улучшился, он всё также не может использоваться банком.

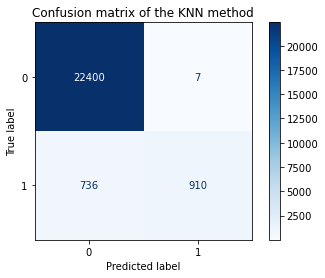
Наконец, воспользуемся наиболее простым классификатором K ближайших соседей. Так как он из представленных трёх наименее ресурсозатратный, для поиска оптимальным параметров воспользуемся функцией GridSearchCV. Будем искать среди следующих параметров:

* Количество соседей: [1, 20]
* Весовая функция: uniform, distance
* Степень метрики Минковского: [1, 3]

В результате, наилучшими параметрами оказались:

* Количество соседей: 18
* Весовая функция: distance
* Степень метрики Минковского: 1

Полученная точность: 0.97

Матрица ошибок:

На удивление, один из самых простых классификаторов показал наилучший результат. Больше половины объектов класса 1 определены правильно. Из всех трёх рассмотренных классификаторов, K ближайших соседей – наилучший вариант для разрабатываемой системы. Безусловно, всё ещё много некорректно определённых объектов класса 1, что является существенно проблемой, но при наличии соответствующих вычислительных мощностей, можно найти модель, которая будет наилучшим образом отвечать требованиям банка.

# Вывод

В результате проделанной работы были исследованы различные классификаторы, такие как Байесовский, K ближайших соседей, метод опорных векторов, решающие деревья. Был получен опыт подбора оптимальных параметров для каждого классификатора, а также рассмотрены различные способы оценивания полученных моделей. Получены начальные знания по предобработке данных и их анализу.

Таким образом, благодаря этой работе был заложен фундамент для дальнейшего изучения методов машинного обучения, обработки и анализа данных.

# Приложение

Весь код и график можно найти в следующем репозитории:

<https://github.com/Mark-Sherman-SE/ML-Labs>