Московский Авиационный Институт

(Национальный Исследовательский Университет)

# **Отчет по лабораторной работе 2**

## **по курсу “Искусственный интеллект”**

**На тему: “Алгоритмы классификации”**

Студент: Иванов Д. В.

Группа: М80-304Б-17

Преподаватель: Самир Халид

Москва, 2020

Постановка задачи:

Необходимо реализовать алгоритмы машинного обучения. Применить данные алгоритмы на наборы данных, подготовленных в первой лабораторной работе. Провести анализ полученных моделей, вычислить метрики классификатора. Произвести тюнинг параметров в случае необходимости. Сравнить полученные результаты с моделями реализованными в scikit-learn. Аналогично построить метрики классификации. Показать, что полученные модели не переобучились. Также необходимо сделать выводы о применимости данных моделей к вашей задаче.

* Логистическая регрессия
* KNN
* SVM
* Дерево Решений
* Случайный лес

Реализация.

Логистическая Регрессия

Описание модели

θ — параметры модели.

xi  — элемент выборки (строка).

xji  — значение характеристики для элемента выборки.

***Гипотеза о логистической регрессии*:**

***Функция стоимости для обучения параметров θ:***

***Градиент:***

***Регуляризации:***

***L1*** *:*

***L2:***

***RMSprop градиентный спуск:***

gt — значение градиента в момент времени t.

ɣ — принадлежит интервалу (0,1).

ῃ — шаг обучения.

**E[g2]t — бегущее среднее в момент времени t:**

***Обучение параметров:***

*Обучение проходит на mini batches.*

Результат выполнения

**Реализованная модель.**

**Без регуляризации:**

Ошибки на валидации

accuracy = 0.6338100909322868

precision = 0.584378671285857

roc auc = 0.5692628608657975

Ошибки на Train --- Test выборках

accuracy = 0.7244918809120302 --- 0.7234802048027692

precision = 0.3806966299192814 --- 0.3814297760704582

roc auc = 0.7502104845361435 --- 0.7500780201265398

**L1 регуляризация:**

Оцененый параметр λ = 1

Ошибки на валидации

accuracy = 0.5780922833004926

precision = 0.5042834802615656

roc auc = 0.6039025344644553

Ошибки на Train --- Test выборках

accuracy = 0.6726402961574057 --- 0.6690704550371386

precision = 0.21913376676403154 --- 0.2154970486273775

roc auc = 0.7047294546034382 --- 0.7003878294633096

**L2 регуляризация:**

Оцененый параметр λ = 1

Ошибки на валидации

accuracy = 0.6162504865352665

precision = 0.5184112372925739

roc auc = 0.6100648645608078

Ошибки на Train --- Test выборках

accuracy = 0.6284931309270544 --- 0.6296964015288095

precision = 0.8235811426238261 --- 0.819450950997845

roc auc = 0.6658751793238991 --- 0.6653836279059693

**Sklearn.**

**Без регуляризации:**

Ошибки на валидации

test\_accuracy = 0.6762188692443609

test\_precision\_macro = 0.6627387972147134

test\_roc\_auc = 0.7317061087095794

Ошибки на Train --- Test выборках

accuracy = 0.7281818291085229 --- 0.7276988533929473

precision = 0.4978799585414115 --- 0.49554951747399983

roc auc = 0.7209326461509085 --- 0.7218719739255142

L1 регуляризация:

Оцененый параметр λ = 0.5

Ошибки на валидации

test\_accuracy = 0.6762188692443609

test\_precision\_macro = 0.6627387972147134

test\_roc\_auc = 0.7317061087095794

Ошибки на Train --- Test выборках

accuracy = 0.7279294222286325 --- 0.7291411264152304

precision = 0.5057947799868087 --- 0.5047315656329054

roc auc = 0.719308471615182 --- 0.7224474054654223

L2 регуляризация:

Оцененый параметр λ = 0.5

Ошибки на валидации

test\_accuracy = 0.6762188692443609

test\_precision\_macro = 0.6627387972147134

test\_roc\_auc = 0.7317061087095794

Ошибки на Train --- Test выборках

accuracy = 0.7316794673012896 --- 0.7316651042042258

precision = 0.5029052419988065 --- 0.5009837908741684

roc auc = 0.7252877933793803 --- 0.7268640168250026

**Выводы по модели:**

1. Из того, что метрики классификации на Train Test выборках не сильно различны следует, что модель не переобучилась.

2. Модель логистической регрессии считается простой, но даже так она выдает acuurace 0.7. В качестве baseline она применима. Но не полностью описывает данные.

KNN

Описание модели

**Алгоритм**

1. Построить kdtree на Train данных

1. Все объекты помещаются в один большой bounding box, описывающий объекты исходного множества (при этом каждый объект ограничен своим bounding box-ом)
2. Он разбивается (делится) плоскостью, параллельной одной из его сторон, на два.

Выбранный способ разделения:

* 1. Рассекать по медиане: отсортируем все примитивы по одной из координат, а медианой назовем элемент (или центр элемента), который находится на средней позиции в отсортированном списке. Секущая плоскость будет проходить через медиану так, что количество элементов слева и справа будет примерно равным.

1. В дерево добавляются два новых узла. Таким же образом каждый из полученных параллелепипедов разбивается на два и т.д.
2. Процесс завершается либо по специальному критерию, либо при достижении определенной глубины дерева, либо же при достижении определенного числа элементов внутри узла дерева. Некоторые элементы могут одновременно входить как в левый, так и в правый узлы (например, когда в качестве элементов дерева рассматриваются треугольники).

2. Нахождение k-ближайших соседей для каждого элемента predict данных

1. Начиная с корневого узла, алгоритм рекурсивно перемещается вниз по дереву так же, как и при вставке точки поиска (т. Е. Он перемещается влево или вправо в зависимости от того, является ли точка меньше или больше текущего узла в разделенное измерение).
2. Как только алгоритм достигает узла, он проверяет эту точку узла и, если расстояние лучше, эта точка узла сохраняется как «текущая лучшая».
3. Алгоритм раскручивает рекурсию дерева, выполняя следующие шаги на каждом узле:
   * + 1. Если текущий узел ближе, чем текущий лучший, то он становится текущим лучшим.
       2. Алгоритм проверяет, могут ли быть какие-либо точки на другой стороне плоскости разбиения, которые ближе к точке поиска, чем текущая наилучшая. В принципе, это делается путем пересечения расщепляющейся гиперплоскости с гиперсферой вокруг точки поиска, радиус которой равен текущему ближайшему расстоянию. Поскольку все гиперплоскости выровнены по оси, это реализовано в виде простого сравнения, чтобы увидеть, меньше ли расстояние между координатой расщепления точки поиска и текущим узлом, чем расстояние (общие координаты) от точки поиска до текущего лучшего.
       3. Если гиперсфера пересекает плоскость, на другой стороне плоскости могут быть более близкие точки, поэтому алгоритм должен перемещаться вниз по другой ветви дерева от текущего узла, ища более близкие точки, следуя тому же рекурсивному процессу, что и весь поиск. ,
       4. Если гиперсфера не пересекает плоскость расщепления, то алгоритм продолжает идти вверх по дереву, и вся ветвь на другой стороне этого узла удаляется.
4. Когда алгоритм завершает этот процесс для корневого узла, поиск завершается.

3. Выбор из наиболее частого встречающегося таргета среди всех соседей в качестве предсказания.

Результат выполнения

**Реализованная модель**

k — ближайших соседей = 11

Ошибки на валидации

accuracy = 0.5896000000000001

precision = 0.6939540727186643

roc auc = 0.5542814946770922

Ошибки на Train --- Test выборках

accuracy = 0.8424 --- 0.808

precision = 0.9156530408773679 --- 0.8963700234192038

roc auc = 0.8306608081066507 --- 0.78451079214438

**Sklearn**

k — ближайших соседей = 11

Ошибки на валидации

test\_accuracy = 0.3471

test\_precision\_macro = 0.2829001463122244

test\_roc\_auc = 0.17431190110781372

Ошибки на Train --- Test выборках

accuracy = 0.8789333333333333 --- 0.834

precision = 0.9294117647058824 --- 0.9028103044496487

roc auc = 0.8694222817113569 --- 0.8134194715939831

**Вывод по модели:**

1. Из того, что метрики классификации на Train Test выборках не сильно различны следует, что модель не переобучилась. Низкая метрика на кросс валидации связана с зависимостью результата от соседей.

2. Модель KNN считается простой, но даже так она выдает acuurace свыше 0.85. На данных показывает себя достаточно хорошо.

SVM

Описание модели

Алгоритм строит **границу принятия решения** для разделения отрицательных и положительных выборок.

**Правило принятия решения**

Чтобы определить, находится ли неизвестный образец справа или слева от границы решения, вектор **u** можно с проецировать на вектор **w**, найдя их точечное произведение. Если значение проекции **u** на **w** больше неизвестного **c**, то выборка **u** находится на правой стороне границы или наоборот.

(1)

Без ограничения общности *уравнение 1* можно изменить, предполагая, что ***c = -b.***

(2)

**Параметризация Маржа модели СВМ**

Для того, чтобы быть в состоянии определить расстояние от каждого опорного вектора на границе решения, правило принятия решения для обоих классов следует рассматривать отдельно. Из-за математического удобства будет использоваться расстояние до **1** , после чего полученное уравнение может быть умножено на любую константу для пропорционального увеличения / уменьшения запаса.

Таким образом, *уравнение 2*для положительных (xplus) и отрицательных (xminus) выборок можно изменить, как это описано в *уравнениях 3.1*и *3.2*.

(3.1)

(3.2)

Хотя есть два отдельных уравнения для разных классов, умножив *уравнение 3.1* на +1 и *3.2* на -1, уравнения можно объединить в одно (*уравнение 5*), что будет более математически удобно. Для этого в *уравнение* будет включена новая переменная **yᵢ**(*уравнение 4*).

(4)

(5)

**Задача оптимизации**

В предыдущих разделах обсуждались ограничения для определения границы решения SVM. Тем не менее, главная цель SVM - найти границу решения с наибольшим расстоянием.

Чтобы найти расстояние между опорными векторами, можно использовать любой из положительных и отрицательных образцов, расположенных на опорных векторах. Расстояние будет равно проекции разности векторов на единичный вектор в направлении вектора нормали **w**. Единичный вектор может быть рассчитан путем деления компонентов вектора **w** на его длину.

(6)

*Уравнение 6* может быть дополнительно упрощено с помощью уравнений

(7)

(8)

Наконец, было получено простое уравнение (8), которое описывает расстояние между опорными векторами. Цель состоит в том, чтобы максимизировать его, а также с учетом ограничений, обсужденных выше.

(9)

Уравнение 9 описывает этапы преобразования задачи оптимизации в квадратичную оптимизацию, которая является более математически удобной в процессе получения градиентов относительно **w**.

**Ограниченная квадратичная оптимизация с множителями Лагранжа**

Для задачи ограниченной оптимизации множители Лагранжа могут быть использованы просто для преобразования ограниченной задачи оптимизации в безусловную, которая удобнее в вычислительном отношении. Теория множителей Лагранжа утверждает, что:

Для минимального / максимального решения функции f (x), ограниченной g (x), кривая уровня функции f (x) касается кривой уровня g (x).Таким образом, если кривые уровня являются касательными, их градиенты должны быть параллельными.

(10)

Используя уравнение 10, ограниченная задача оптимизации SVM преобразуется в неограниченную. Новое уравнение будет целевой функцией SVM с суммированием по всем ограничениям.

(11)

В *уравнении 11* множитель Лагранжа не был включен в качестве аргумента целевой функции L (w, b). По мере того как значение множителя **альфа** будет установлено условно в ходе численного процесса оптимизации , таких, что:

* Если образец на векторе **поддержки: α = 1**
* еще: **α = 0**

Для минимизации целевой функции будет реализован подход градиентного спуска. Здесь требуется вычисление градиента L (w, b) относительно **w** и **b,** которые описаны в уравнениях ниже.

(12.1)

(12.2)

(12.3)

Результат выполнения

**Реализованная модель**

C = 26

Ошибки на валидации

accuracy = 0.6269506210885225

precision = 0.6022078848635231

roc auc = 0.5017249056251604

Ошибки на Train --- Test выборках

accuracy = 0.6221469008040962 --- 0.6192399221172568

precision = 0.016614843431012282 --- 0.016209125831537524

roc auc = 0.7135424961801137 --- 0.68034248618037

**Sklearn**

C = 6

Ошибки на валидации

Ошибки на валидации

test\_accuracy = 0.22300000000000003

test\_precision\_macro = 0.12443900732472804

test\_roc\_auc = 0.19829515229515232

Ошибки на Train --- Test выборках

accuracy = 0.70935948749384 --- 0.7070743491742987

precision = 0.36204026508370235 --- 0.3554764358662044

roc auc = 0.724563598058007 --- 0.7249103784925696

**Вывод по модели:**

1. Из того, что метрики классификации на Train Test выборках не сильно различны следует, что модель не переоблучилась.

2. Реализованная модель показала aссuracн 0.6, а sklearn 0.7. В применении их на данных нет смысла, так как они не описывают их полностью.

Решающее дерево

Описание модели

**Дерево решений представляет собой двоичное дерево:**

* Узел представляет одну входную переменную (X) и точку разделения этой переменной.
* Конечные узлы (также называемые терминальными узлами) дерева содержат выходную переменную (y), которая используется для прогнозирования.

**Рекурсивный алгоритм**

1. На каждом шаге выбирается тот признак, при разделении по которому прирост информации оказывается наибольшим. Создание разделения включает в себя три части:
2. Вычисление показателя Джини
3. Разделение набора данных:
   1. Проверка находится ли значение атрибута ниже или выше значения разделения
   2. присвоение его левой или правой группе соответственно.
4. Оценка всех разделений:

* Проверяется каждое значение каждого атрибута как разделение кандидатов, оценивается стоимость разделения и находится наилучшее возможное разделение, которое мы могли бы сделать.
* Как только будет найдено лучшее разделение, оно используется в качестве узла в дереве решений.

1. Выборка делится на левую и правую часть.

* Выбирается разделение с лучшей стоимостью.

Используется функция стоимости Джини :

* Все входные переменные и все возможные точки разделения оцениваются и выбираются жадным образом на основе функции стоимости.

1. Процедура повторяется рекурсивно к каждой из частей, пока энтропия не окажется равной нулю или какой-то малой величине (если дерево не подгоняется идеально под обучающую выборку во избежание переобучения).

Дополнительные условия остановка:

* Достижении нужной глубины дерева
* Достижение минимальные записи узла

**Представление дерева**

* Используется словарь для представления узла в дереве решений, поскольку мы можем храним данные по имени.
* При выборе наилучшего разделения и использовании его в качестве нового узла для дерева хранится индекс выбранного атрибута, значение этого атрибута, по которому нужно разделяется, и две группы данных, разделенные по выбранной точке разделения.
* Каждая группа данных представляет собой собственный небольшой набор данных, состоящий только из тех строк, которые были назначены левой или правой группе в процессе разделения.

**Предсказание**

После создания дерева можно перемещаться с новой строкой данных, следующей за каждой ветвью с разбиениями, пока не будет сделан окончательный прогноз.

Результат выполнения.

**Реализованная модель**

Максимальная глубина = 9

Минимальные запись узла = 2

Ошибки на валидации

accuracy = 0.781

precision = 0.6394400207177451

roc auc = 0.7510276252314176

Ошибки на Train --- Test выборках

accuracy = 0.9093333333333333 --- 0.832

precision = 0.948905109489051 --- 0.8163265306122449

roc auc = 0.8979576929819435 --- 0.8229715489989462

**Sklearn**

Максимальная глубина = 4

минимальные запис узла = 4

Ошибки на валидации

test\_accuracy = 0.22300000000000003

test\_precision\_macro = 0.12443900732472804

test\_roc\_auc = 0.19829515229515232

Ошибки на Train --- Test выборках

accuracy = 0.7573333333333333 --- 0.684

precision = 0.6642335766423357 --- 0.5408163265306123

roc auc = 0.7383245139059808 --- 0.6665608913334744

**Вывод по модели:**

1. Из того, что метрики классификации на Train Test выборках не сильно различны следует, что модель не переоблучилась.

2. Так как решающее дерево обычно используется в купе со случайным .лесом, то из-за большой вероятности переобучения, я бы не стал использовать, как окончательную модель предсказания. Хотя переобучения не наблюдалось

Случайный лес

Описание модели

**Бутстрэп**

Метод бутстрэпа заключается в следующем. Пусть имеется выборка X размера N. Равномерно возьмем из выборки N объектов с возвращением.

**Случайный лес**

Алгоритм построения случайного леса, состоящего из N деревьев, выглядит следующим образом:

Для каждого n = 1,…,N:

1. Сгенерировать выборку Xn с помощью бутстрэпа;
2. Построить решающее дерево bn по выборке Xn:  
   — по заданному критерию мы выбираем лучший признак, делаем разбиение в дереве по нему и так до исчерпания выборки  
   — дерево строится, пока в каждом листе не более nmin объектов или пока не достигнем определенной высоты дерева  
   — при каждом разбиении сначала выбирается m случайных признаков из n исходных,  
   и оптимальное разделение выборки ищется только среди них.

Итоговый классификатор , — для выбирается решение голосованием по большинству.

Результат выполнения.

**Реализованная модель**

Количество деревьев = 30

Максимальная глубина = 9

Минимальные запись узла = 8

Ошибки на валидации

test\_accuracy = 0.621920452312341

test\_precision\_macro = 0.6029737105847312

test\_roc\_auc = 0.6539481925759321

Ошибки на Train --- Test выборках

accuracy = 0.7078924157329591 --- 0.6827543285746312

precision = 0.5912341746290322 --- 0.5702831820385739

roc auc = 0.7223746321823471 --- 0.7194832930575837

**Sklearn**

Количество деревьев = 20

Максимальная глубина = 9

минимальные запис узла = 7

Ошибки на валидации

test\_accuracy = 0.6564142996993695

test\_precision\_macro = 0.6272471830054627

test\_roc\_auc = 0.6884146537635737

Ошибки на Train --- Test выборках

accuracy = 0.7670645079868749 --- 0.7649455541934088

precision = 0.5664436697132448 --- 0.5604797151691183

roc auc = 0.7660907539864598 --- 0.7652482056655239

**Вывод по модели:**

1. Из того, что метрики классификации на Train Test выборках не сильно различны следует, что модель не переоблучилась.
2. Так как решающее дерево обычно используется в купе со случайным .лесом, то из-за большой вероятности переобучения, я бы не стал использовать, как окончательную модель предсказания. Хотя переобучения не наблюдалось

Дополнительная часть с временными рядами.

ARIMA

Описание модели

Модель **ARIMA(p,d,d)** для нестационарного ряда **Xt** имеет вид:

где **εt** — стационарный временной ряд;

**c, ai, bj** — параметры модели.

**p** — порядок компоненты AR

**d**— порядок интегрированного ряда

**q**— порядок компонетны MA

**Δd** - оператор разности временного ряда порядка d (последовательное взятие d раз разностей первого порядка — сначала от временного ряда, затем от полученных разностей первого порядка, затем от второго порядка и т.д.)

**Лаговый оператор**

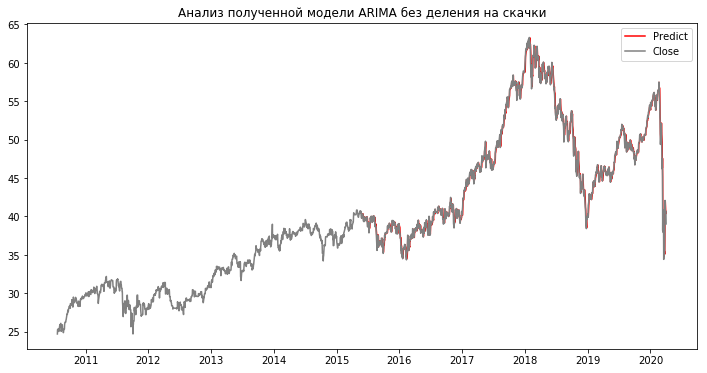
тогда

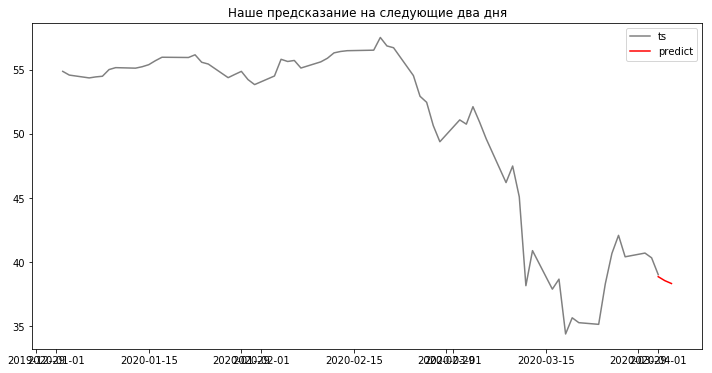
Результат выполнения

p = 1

q = 1

d = 1





R2: 0.683875056478572 | MAE: 1.2491238100992432 | MSE: 2.5450947752249053

SARIMA

Описание модели

Модель **SARIMA** указана (p,d,q)×(P,D,Q)s

В этой модели параметры обозначают следующее:

* **p** — порядок модели AR(p)
* **d**— порядок интегрирования исходных данных
* **q**— порядок модели MA(q)
* **P**— порядок сезонной составляющей SAR(P)
* **d**— порядок интегрирования сезонной составляющей
* **Q**— порядок сезонной составляющей SMA(Q)
* **s**— размерность сезонности(месяц, квартал и т. д.)
* **Δd** - оператор разности временного ряда порядка d (последовательное взятие d раз разностей первого порядка — сначала от временного ряда, затем от полученных разностей первого порядка, затем от второго порядка и т.д.)

Лаговый оператор

С точки зрения одномерной структурной модели это можно представить как

где ηt применимо только в случае ошибки измерения (хотя оно также используется в случае чисто регрессионной модели, т. е. если **p = q = 0**).

С точки зрения этой модели, регрессия с ошибками **SARIMA** может быть легко представлена как

эта модель используется, когда предоставляются экзогенные регрессоры.

Обратите внимание, что приведенные полиномы с запаздыванием формы будут записываться как:

Результат выполнения

p = 1

q = 1

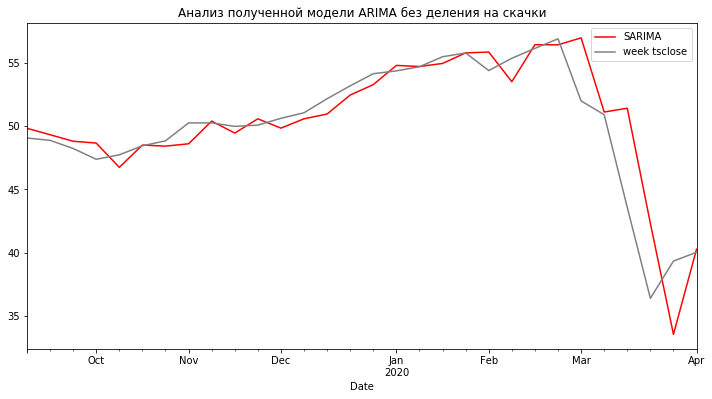
d = 1

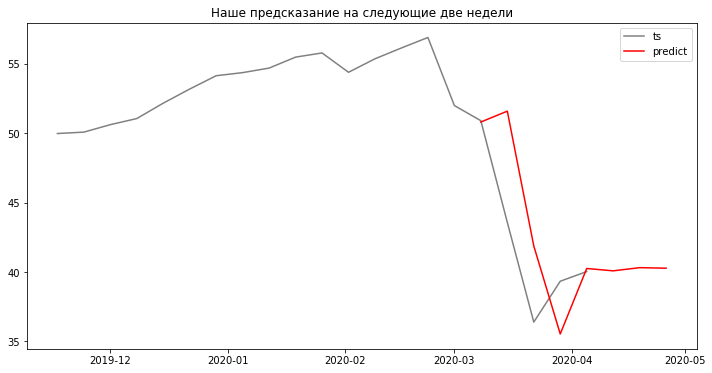
P = 1

Q = 1

D= 1

s = 52





Ошибка на одну неделю

R2: 0.7679341867617295 | MSE: 5.73218981683044 | MAE 1.3815974858414373

**Вывод по моделям:**

Данные модели могут хорошо себя показать в краткосрочной перспективе, на длинных дистанциях они не релевантны.