Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Национальный исследовательский университет**

**«Высшая школа экономики»**

Факультет компьютерных наук

ООП «Прикладная математика и информатика»

**Отчёт о прохождении учебной практики**

|  |  |
| --- | --- |
| **Студент:** | Суходольская Евгения Дмитриевна |
| **Группа:** | 156(ТИ) |

|  |  |
| --- | --- |
| **Организация (место прохождения учебной практики):** ПАО «МТС» |  |

**Руководитель от организации:** Николай Трошнев

Москва, 2017

**Оглавление**

1. Введение – стр. 3
2. Основная часть – стр. 3-11

2.1) Место практики – «ПАО» МТС – стр. 3-4

2.2) Цели и задачи практики: написание алгоритма – стр. 4-5

2.3) Разделение задачи на подзадачи: базовая часть – стр. 5-6

2.4) Особенности задачи – стр. 6-8

2.5) Инструменты и методы, освоенные на практике в МТС – стр. 8-10

2.6) Результаты работы алгоритма и их применимость – стр. 10-11

1. Заключение – стр. 11
2. Список использованной литературы – стр. 12
3. Приложение – стр. 12-22

**Введение**

Целью моей практики была автоматизация регулирования стоимости мобильных устройств в интернет-магазине МТС относительно цен конкурентов. Мне была поставлена следующая задача: написание алгоритма мэтчинга названий мобильных устройств из разных интернет-магазинов по их описаниям, который:

- для каждого мобильного телефона с сайта конкурента возвращал соответствующий телефон из интернет-магазина МТС,

- был расширяем на новые характеристики устройств,

- требовал бы минимальных затрат при обновлении списка устройств.

**Основная часть**

**Место практики – «ПАО» МТС**

Местом прохождения моей практики было ПАО «МТС» (В 2015 году 25 июня было принято решение о смене фирменного наименования Открытое акционерное общество «Мобильные ТелеСистемы» (ОАО «МТС») на Публичное акционерное общество «Мобильные ТелеСистемы» (ПАО «МТС») в связи с изменением действующего законодательства РФ) - ведущая компания в России и странах СНГ по предоставлению услуг мобильной и фиксированной связи, доступа в интернет, кабельного и спутникового ТВ-вещания, цифровых сервисов и мобильных приложений, финансовых услуг и сервисов электронной коммерции, а также конвергентных ИТ-решений в сфере системной интеграции, интернета вещей, мониторинга, обработки данных и облачных вычислений. В России, Украине, Беларуси, Армении и Туркменистане услугами мобильной связи Группы МТС пользуются около ста десяти миллионов абонентов. Бренд МТС входит в TOP-500 самых дорогих брендов мира по версии Brand Finance. Это сжатое описание позволяет заметить, что компания достаточно известна, по крайней мере в России, поэтому узнать о стажировке «MTS Flash», в которой я приняла участие, было нетрудно, хотя саму стажировку простой я бы не назвала.

Если быть более конкретной, местом моей летней стажировки был отдел управления данными департамента стратегического маркетинга МТС, или же, как мне его представили в первый день, отдел «IT big data». Отдел состоит из команды «датасаентистов» (надо сказать, я ни разу не слышала, чтобы члены отдела называли себя «специалистами по работе с большими данными», напротив, повсеместно используется профессиональный сленг, так, например, отдел разработчиков называют «девопсами», а менеджеров финансового отдела - «финиками»), под руководством Николая Трошнева,

участвующих в различных проектах и занимающихся решением задач, связанных с анализом данных. Так, это может быть кластеризация сайтов, посещаемых абонентами МТС, для выявления областей их интересов, или поиск зависимости между числом продаж определенных девайсов в различные временные периоды в салонах Москвы и построение предсказаний на основе известных характеристик.

**Цели и задачи практики: написание алгоритма**

Целью моей практики было решение одной из таких задач, которая была разделена на две части: сбор информации из интернета в онлайн режиме (написание краулера) и написание алгоритма, обрабатывающего данную информацию и выдающего необходимый результат. Я занималась второй частью задачи, то есть написанием алгоритма, хотя в конце стажировки мне была предоставлена возможность принять участие и в первой части – мне было поручено найти причину неправильной работы краулера и, по возможности, устранить ее. Остановлюсь подробно на содержании задания, его смысле и значимости.

Неудивительно что у крупной компании, занимающейся, в том числе, продажей мобильных устройств, есть интернет-магазин. Так как одной из важных составляющих успеха продаж является правильная цена товара – такая, что она не слишком отличается от цен конкурентов, у интернет-магазина МТС есть цель в режиме реального времени обновлять стоимость устройств в соответствии с текущей ситуацией на рынке мобильных телефонов. До нынешнего момента отслеживание изменений в ценах конкурентов выполнялось вручную, что занимало очень много времени и было затратно с точки зрения человеческих ресурсов. Соответственно, отделу, где я проходила стажировку, была поставлена задача автоматизации процесса обновления стоимости товаров. Перейдем к описанию той части задачи, что была поручена мне.

Одним из этапов автоматизации мониторинга конкурентов является поиск связок телефонов в интернет-магазине МТС и на сайтах конкурентов. Верно найденная связка позволяет сравнить стоимость девайсов, оценить разницу цен и принять решение, нуждается ли в регулировке цена девайса в магазине МТС. Представим, что нам необходимо написать алгоритм поиска таких пар телефонов. Первый вопрос, который хотелось бы задать: что мы можем подать алгоритму на вход? Какой информацией, необходимой для решения задачи мы располагаем? Итак, со стороны конкурентов предыдущий этап автоматизации – краулер - передает полное название (содержащее бренд, модель, цвет, объем памяти) девайса и его цену; cо стороны интернет-магазина МТС мы имеем название, полные технические характеристики, цвет и также цену. Тогда зададим второй, не менее важный вопрос, что конкретно должен «уметь» такой алгоритм, то есть, если мы знаем, что подается на вход, что мы хотим получить на выход? Передо мной была сформулирована цель: нужно построить алгоритм, который: для каждого телефона с сайта конкурента умел бы возвращать соответствующий телефон из интернет-магазина МТС, если же такого телефона нет, то алгоритм должен выдать специальное значение, или же не возвращать ничего; был бы расширяем на новые характеристики телефонов, то есть алгоритм должен был бы работать не только на существующих в данный момент в базе интернет-магазина МТС моделях, но и в случае добавления новых устройств с некоторым набором характеристик, уметь также качественно находить связку «телефон конкурента – телефон из базы»; а также требовал бы минимальных вложений при обновлении списка телефонов со временем, проще говоря, сохранял бы результаты своей работы, чтобы не начинать каждый раз «с нуля».

**Разделение задачи на подзадачи: базовая часть**

Во время написания алгоритма мне помогло то, чему меня научили меня первые два курса – построение решения любой задачи должно происходить поэтапно. Задачу нужно разбить на подзадачи и решать их последовательно, именно так можно наиболее эффективно достичь требуемого результата. Первое, с чего я начала – это определение базовых правил для нахождения связок.

Так как мне была предоставлена свобода действий в выборе системы, согласно которой алгоритм будет искать соответствия, я решила разбить строку, в которой содержится информация об устройстве, на пять составляющих. Первая, буквенная, отвечающая как за вендор (компания-поставщик (зачастую, производитель) товаров и услуг под своей торговой маркой, например, Apple или Asus), так и за название модели (например, Zenfone) или за название серии (Desire для HTC). В буквенной части строки могут присутствовать лишь символы кириллицы или латиницы. Вторая, цифровая, являющаяся либо годом выпуска телефона (Samsung Galaxy A3 (2017), к примеру, отличается от той же модели предыдущего года), либо номером его серии (iPhone 5, например). В этой части, очевидно, могут быть только цифры от 0 до 9. Третья, «смешанная», которая в девяноста девяти процентах случаев являлась серией модели (так у модели Asus Zenfone 3 это ZE552KL или же ZE550KL), обязательно включающая в себя хотя бы одну цифру и хотя бы один символ латинского или кириллического алфавита. Четвертая составляющая - цвет модели – слово русского или английского языка. И последняя, пятая, часть, отвечающая за объем памяти устройства, в которой обязательно содержится как объем памяти – число, так и единица его измерения, то есть, в данном случае, мегабайт или гигабайт.

Каждой из пяти составляющих я поставила в соответствие одно правило, на вход которому подаются характеристики двух моделей (в строковом формате), между которыми нужно проверить отношение равенства. Правило должно определять принадлежность очередной единицы строки (разделителем в которой выступает пробел) к своей составляющей и затем, после выделения нужной части в обоих характеристиках, сравнивать и возвращать «вероятность» совпадениях этих множеств. Вероятность записана мной в кавычках потому, что привычное ее определение предполагает, что единица является индикатором полной идентичности, у меня же каждое правило возвращало количество совпавших у моделей единиц строки, подпавших под область действия правила. Я не нормировала возвращаемое значение потому, что количество лексем, описывающих каждую из пяти составляющих, может варьироваться, как и общее количество лексем, соответственно, нормирование делением на количество элементов в строке было бы некорректным, а иной, более сложный, способ нормализовать значение искать не было бы смысла, так как нормализация влияла бы не на результат, а лишь на его наглядность.

**Особенности задачи**

В ходе написания данных пяти базовых правил, я столкнулась с некоторыми особенностями задачи, которые необходимо было учитывать при разработке. Поскольку не существует единого стандарта описания характеристик телефона, то идентичные модели могли быть описаны совершенно по-разному. Так, для правила, работающего с цветами, пришлось вручную создавать словарь исключений, использование которого помогло учитывать трудности языкового перевода: например, цвет модели gold мог быть обозначен в одном интернет-магазине как «золотистый», а в другом, как «золотой», а серый цвет мог быть переведен как американский «gray» или британский «grey». В случае работы с памятью пришлось учитывать, что единица объема памяти устройства может быть указана как в сокращенном варианте написания «гб», так и полностью «gigabyte». А при работе с серией телефона надо было следить за тем, чтобы серия была признана правилом одинаковой, несмотря на то, что в одном описании она представлена полностью, а в другом «расклеена» на составляющие («meizu pro 6» и «meizu PRO6»). Также необходимо было учитывать, что прописные и строчные буквы могут появляться в названиях в любом порядке и количестве, а также могут возникать кавычки и некоторые другие символы, которые не должны влиять на определение равенства лексем.

Используя описанное выше, можно было бы добиться некоторого результата, но так как от меня требовалось решить задачу максимально качественно, то необходимо было расширять список правил, обрабатывающих поданную на вход информацию. Так мною было решено воспользоваться инструментами текстового анализа, так как несмотря на то, что одно и то же описание может отличаться по составу лексем, различные описания отличаются по нему еще больше. Я реализовала правило, подсчитывающее количество совпадающих биграмм (биграмма - два слова, которые являются соседними в тексте) у полного описания сравниваемых моделей. В ходе написания правила я познакомилась c функцией ngrams модуля nltk.util библиотеки NLTK (библиотека NLTK - пакет [библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и программ для символьной и статистической [обработки естественного языка](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B5%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0)). Затем мною было написано правило, считающее меру Жаккара (или же коэффициент Жаккара) между двумя описаниями устройств, которые мы хотим сравнить для выявления связки. Она вычисляется, как пересечение двух множеств – в данном случае количество совпавших лексем в двух строках, поделенное на их объединение – общее количество различных лексем. Третье правило подсчитывало известное мне еще из курса «Алгоритмы и структуры данных» расстояние Левенштейна (или же редакционное расстояние) между буквенными последовательностями, получавшимися исключением пробелов из строк - характеристик моделей. Его значение - минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Надо сказать, что если значения, возвращаемые первыми двумя правилами, также являются «вероятностями» совпадения, как и в базовых правилах, то есть, чем больше вероятность – возвращенное число – тем больше шанс, что мы обрабатываем два описания одного и того же устройства, то значение, возвращенное третьим правилом, напротив, тем больше, чем вероятнее то, что характеристики различны. Если бы я пользовалась самым примитивным способом подсчета общей вероятности того, что связка корректна, то сложив бы все «вероятности», выданные первыми семью правилами, я бы вычла значение, возвращенное восьмым.

Надо заметить, что добавление еще двух, казалось бы, очевидной полезности, правил - появившихся впоследствии, когда ко мне пришло более общее видение задачи – вычисляющих абсолютную и относительную (доля модуля разности цен сравниваемых устройств от цены телефона из стороннего интернет-магазина) разницу в цене девайсов, и использующих эти два значения, как еще две своеобразные «вероятности» совпадения, (грубо говоря, чем меньше разница в цене и доля в цене, тем больше шанс, что девайс один и тот же), практически не повлияло на качество работы алгоритма, в отличие от трех правил, использующих методологию анализа текста.

**Инструменты и методы, освоенные на практике в МТС**

На практике в МТС я поняла, что никто не требует от тебя абсолютных знаний в предметной области, умения пользоваться всеми инструментами, в ней применяющихся. Гораздо важнее умение и желание учиться. И если этап придумывания и написания правил был пройден мной почти самостоятельно, то, когда встал вопрос: «Итак, у нас есть некоторое количество правил, возвращающих значения, каким-то образом отображающие степень сходства предоставленных описаний устройств. Как определить по набору этих значений, является ли найденная связка корректной?», мне рассказали о таких понятиях машинного обучения, как классификация (один из разделов [машинного обучения](http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), посвященный решению следующей задачи: имеется множество объектов, разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется [обучающей выборкой](http://www.machinelearning.ru/wiki/index.php?title=%D0%92%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B0). Классовая принадлежность остальных объектов не известна. Требуется построить [алгоритм](http://www.machinelearning.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), способный классифицировать произвольный объект из исходного множества) и обучение с учителем (один из разделов [машинного обучения](http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), посвященный решению следующей задачи: имеется множество объектов и множество возможных ответов. Существует некоторая зависимость между ответами и объектами, но она неизвестна. Известна только конечная совокупность прецедентов — пар «объект, ответ», называемая [обучающей выборкой](http://www.machinelearning.ru/wiki/index.php?title=%D0%9E%D0%B1%D1%83%D1%87%D0%B0%D1%8E%D1%89%D0%B0%D1%8F_%D0%B2%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B0). На основе этих данных требуется восстановить зависимость, то есть построить [алгоритм](http://www.machinelearning.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества. Под учителем понимается либо сама [обучающая выборка](http://www.machinelearning.ru/wiki/index.php?title=%D0%9E%D0%B1%D1%83%D1%87%D0%B0%D1%8E%D1%89%D0%B0%D1%8F_%D0%B2%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B0), либо тот, кто указал на заданных объектах правильные ответы).

Мне было объяснено буквально на пальцах, что моя задача – обучить модель-классификатор, который будет впоследствии определять корректность связки, составив для нее вручную обучающую выборку, где объектом были бы значения, возвращенные для пар характеристик десятью моими правилами, а ответом – проставленные мною нули и единицы в зависимости от того, совпадают ли элементы пар. Затем метод обучения модели был мною изменен для увеличения размера обучающей выборки – я собрала какое-то количество совпадающих характеристик в пары, а затем нашла декартово произведение элементов пар, которые отвечали за описание модели из стороннего интернет-магазина, со всеми характеристиками девайсов, имеющимися в базе МТС. Таким образом, я получила некоторое количество единиц и огромное количество нулей. Мне оставалось только взять столько отрицательных ответов, сколько нужно, чтобы избежать переобучения, и перемешать их с положительными. Таким методом я пользовалась вплоть до окончания стажировки.

Какую же модель я должна была обучить? Поскольку у меня отсутствовал опыт работы с алгоритмами машинного обучения, я самостоятельно не могла выбрать подходящий для данной задачи классификатор, поэтому мне было поручено изучить и применить на практике «random forest» - это множество решающих деревьев, на основе которых окончательное решение принимается голосованием по большинству. Я узнала о достоинствах «случайного леса» - способности эффективно обрабатывать данные с большим числом признаков и классов, высокой параллелизуемости и масштабируемости, о его недостатках – большом размере получающихся моделей (на что мне сказали не обращать внимания), а также о библиотеке scikit-learn, предоставляющей собой реализацию целого ряда алгоритмов для обучения с учителем и обучения без учителя, в которой в том числе находится и реализация «random forest».

Поскольку моей задачей было сделать максимально хорошо ищущее правильные связки решение, то не менее важным этапом моей работы было оценивание результатов алгоритма. В начале стажировки мне были известны простейшие методы оценки, вроде процентного содержания правильных ответов в их общем числе, но они были недостаточно эффективны. Во-первых, необходимо было оценить работу «random forest», чтобы избежать переобучения, то есть той ситуации, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении. Так я познакомилась с кросс-валидацией -  методом оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных. Во-вторых, я узнала о таком инструменте наглядного представлении качества работы алгоритма, как precision-recall кривая, где precision – мера точности, характеризующая, сколько полученных от классификатора положительных ответов являются правильными - чем больше точность, тем меньше число ложных попаданий, - a recall - мера полноты, характеризующая способность классификатора «угадывать» как можно большее число положительных ответов из ожидаемых. Критерием качества выступает площадь под PR-кривой. С помощью приведенных методов оценки я узнала, что мой итоговый алгоритм работает правильно в девяноста пяти процентах случаев.

**Результаты работы алгоритма и их применимость**

Написанный мной алгоритм, объединяющий три основные составляющие: правила, модель и оценку, - в одно целое, соответствует заявленным требованиям. Доказательство этого я приведу, кратко описав структуру моего решения. Так как алгоритм работает с информацией, которую удобно хранить в табличной форме, все входные и выходные были размещаются в «датафреймах» -  это структура хранения данных в пакете pandas. Данный пакет делает Python мощным инструментом для анализа данных. Пакет дает возможность строить сводные таблицы, выполнять группировки, предоставляет удобный доступ к табличным данным. Алгоритм получает на вход две таблицы, с характеристиками и ценами телефонов магазина МТС и сторонних интернет магазинов. Затем обрабатывается декартово произведение этих таблиц: каждая полученная пара «телефон из магазина МТС – телефон из стороннего интернет магазина» проверяется на сходство применением десяти имеющихся правил. Датафрейм с рассчитанными коэффициентами совпадения каждой из пар передается обученной модели «random forest», которая определяет, с какой вероятностью они идентичны, связки, элементы которых совпадают с вероятностью больше 0,7 (данное значение было признано оптимальным при оценке) считаются корректными, телефонам из сторонних-интернет магазинов, которым пара по данным критериям была не найдена, ставится в соответствие специальное значение «пара не была найдена». Полученные результаты сохраняются в csv файл, который является результатом работы алгоритма. При последующих запусках сохраненные данные используются таким образом: найденные ранее корректные связки остаются, алгоритм работает лишь c парами, в которых присутствует специальное значение, и теми телефонами, которые появились при обновлении как базы МТС, так и сторонних интернет-магазинов. Так удовлетворяется требование минимальных вложений при обновлении списка телефонов со временем.

Решенная мною задача может использоваться не только для урегулирования цен, но и для создания рекомендательных сервисов: используя произвольный набор правил из имеющихся десяти и переобучив модель, можно добиться нахождения связок, в которых элементы совпадают по определенным характеристикам, например, при отсутствии запрошенного клиентом телефона в магазине, автоматически предлагать ему ту же модель другого цвета, если она имеется в наличии.

**Заключение**

Навыки, полученные мной в ходе прохождения практики, несомненно принесут мне огромную пользу. Во-первых, мне легче будет пройти курс «Машинное обучение» не только потому, что я уже успела воспользоваться на практике некоторыми методами и инструментами, изучающимися в курсе, такими как кросс-валидация, решающие деревья, с библиотеками pandas и scikit-learn, с интерактивной оболочкой питона jupyter notebook, но и потому, что теперь мне ясен смысл курса, понятна его применимость в работе. Во-вторых, этот небольшой опыт работы в компании, в команде специалистов, помог преодолеть страх собственного невежества, боязнь не справиться: нет ничего плохого в том, чтобы задавать вопросы, делать что-то неправильно, главное, учиться на ошибках и не бояться думать.

Стажировка была полезна хотя бы потому, что я узнала, что такое работать, быть по-настоящему ответственным за результаты своего труда, более того, мне стало более понятно, чем мне может быть интересно заниматься в будущем - анализом данных, если быть точнее, анализом текста.

**Список использованной литературы**

1. Введение в анализ данных с помощью pandas: <https://habrahabr.ru/post/196980/>
2. Семинары по выбору моделей, Е. Соколов: <http://www.machinelearning.ru/wiki/images/1/1c/Sem06_metrics.pdf>
3. Случайный лес («Random forest»): <https://alexanderdyakonov.wordpress.com/2016/11/14/%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B9-%D0%BB%D0%B5%D1%81-random-forest/>
4. Введение в scikit-learn, Игорь Субботин: <http://igorsubbotin.blogspot.ru/2015/01/intro-to-scikit-learn.html>
5. Документация библиотеки scikit-learn: [http://scikit-learn.org/stable/documentation.html](https://clck.yandex.ru/redir/nWO_r1F33ck?data=NnBZTWRhdFZKOHQxUjhzSWFYVGhXYXlfemtpYlNpeTJ1ZmtoWGNJWmdtQUQzejVRQ2VpaFBYN2ZFVUFyRmF0QzFmWlFzenY0U25pU0pjQUtscTVFaEdkcGdVaXB1azQ0VTFXOUxBeTRYOEZkWXR6VVNOMnNpcHlDaENHZlZmcFVsOU9QT3JnV1JCaHFsRE41ZGxsUTFR&b64e=2&sign=09f11d72efc79a7add1100ed41241d7c&keyno=17)
6. Документация библиотеки pandas: [http://pandas.pydata.org](https://clck.yandex.ru/redir/nWO_r1F33ck?data=NnBZTWRhdFZKOHQxUjhzSWFYVGhXZHkwd3NGc09CdFpYeG1teXp0SzhDRDRmUExjOWdseVRfc3V3Q1ozOEQ3bWlvbmxLaVVUb3I5ZG1ZU1VodkR5bkdCRDZoY2FVcW5QMDdMYV9LSUk3ZE0&b64e=2&sign=34fe63c8e3ed167081ae823567e19784&keyno=17)
7. Статья «Немного о precision и recall»: <http://blog.gramant.ru/2012/06/06/f1-measure/>
8. Учебное пособие «Автоматическая обработка текстов на естественном языке и компьютерная лингвистика». Большакова Е.И., Клышинский Э.С., Ландэ Д.В., Носков А.А., Пескова О.В., Ягунова Е.В

**Приложение**

**Десять правил с комментариями**

**def** colour\_parser\_eng():*#Функция, превращающая файл с цветами на английском языке в множество* f = open(**'colours\_wiki.txt'**, **'r'**)  
 set\_of\_colours = set()  
 **for** lines **in** f:   
 hash\_s = lines.find(**'#'**) *#разделитель в файле* lines = lines[:hash\_s - 1]  
 curr\_colour = lines.lower().split()  
 **for** i **in** range(len(curr\_colour)):  
 curr\_colour[i] = curr\_colour[i].strip(**'(,\«»"\"'')'**)  
 **if** curr\_colour[i].isalpha():  
 set\_of\_colours.add(curr\_colour[i])  
 **return** set\_of\_colours  
  
**def** colour\_parser\_rus(): *#Функция, превращающая файл с цветами на русском языке в множество* f = open(**'colours\_rus.txt'**, **'r'**)  
 set\_of\_colours = set()  
 **for** lines **in** f:   
 hash\_s = lines.find(**'#'**) *#разделитель в файле* lines = lines[:hash\_s - 1]  
 curr\_colour = lines.lower().split()  
 **for** i **in** range(len(curr\_colour)):  
 curr\_colour[i] = curr\_colour[i].strip(**'(,\«»"\"'')'**)  
 **if** curr\_colour[i].isalpha():  
 set\_of\_colours.add(curr\_colour[i])  
 **return** set\_of\_colours  
  
**def** bigrams(string\_to\_compare, model): *#функция, берущая на вход 2 строки, состоящие из нескольких токенов, создающая в каждой строке список всевозможных сочетаний по 2  
#и считающая количество совпавших пар - биграмм - между строками* **import** nltk  
 **from** nltk.util **import** ngrams  
 string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,\«»"\"'')'**)  
 bigrams\_s = list(ngrams(string\_to\_compare, 2))  
 model = model.lower().split()  
 probability = 0  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 bigrams\_m = list(ngrams(model, 2))  
 all\_to\_comp = len(bigrams\_m)  
 **for** i **in** range(all\_to\_comp):  
 **if** bigrams\_m[i] **in** bigrams\_s:  
 probability += 1  
 **return** probability  
  
**def** jaccard(string\_to\_compare, model): *#функция, считающая меру Жаккара между строками - пересечение слов в 2х строках, деленное на их объединение* string\_to\_compare = string\_to\_compare.lower().split()  
 a = len(string\_to\_compare)  
 **for** i **in** range(a):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,\«»"\"'')'**)  
 model = model.lower().split()  
 b = len(model)  
 **for** i **in** range(b):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 model = set(model)  
 string\_to\_compare = set(string\_to\_compare)  
 **return** len(model & string\_to\_compare) / len(string\_to\_compare | model)  
  
**def** colour(string\_to\_compare, model): *#функция, находящая в строках цвет и проверяющая, совпадает ли он* set\_of\_colours\_eng = colour\_parser\_eng() *#множество всех цветов по-английски* set\_of\_colours\_rus = colour\_parser\_rus() *#множество всех цветов по-русски* dct\_colours\_rus = {**'золотой'**: **'gold'**, **'черный'**: **'black'**, **'чёрный'**: **'black'**, **'белый'**: **'white'**, **'синий'**: **'blue'**, **'серебряный'**: **'silver'**, **'серебристый'**: **'silver'**, **'красный'**:**'red'**, **'желтый'**:**'yellow'**, **'жёлтый'**: **'yellow'**, **'космос'**: **'space'**, **'розовый'**:**'pink'**, **'розовое'**:**'rose'**, **'золото'**:**'gold'**, **'зеленый'**:**'green'**, **'светлый'**:**'light'**, **'темный'**:**'dark'**, **'жемчужный'**: **'pearl'**, **'рубиновый'**:**'ruby'**, **'теплый'**:**'warm'**, **'коричневый'**:**'brown'**, **'топаз'**: **'topaz'**, **'космический'**:**'cosmic'**, **'глубокий'**:**'deep'**, **'металлик'**:**'metallic'**, **'красное дерево'**:**'mahogany'**, **'кремовый'**:**'cream'**, **'лаймовый'**:**'lime'**, **'сиреневый'**: **'lilac'**, **'стальной'**:**'steel'**, **'бронзовый'**:**'bronze'**, **'оникс'**:**'onyx'**, **'циан'**:**'cyan'**, **'пурпурный'**: **'magenta'**, **'коралловый'**:**'coral'**, **'фуксия'**:**'fuchsia'**,**'оранжевый'**:**'orange'**, **'песочный'**:**'sand'**, **'янтарный'**:**'amber'**, **'винный'**:**'wine'**, **'ультра'**:**'ultra'**, **'фиолетовый'**:**'purple'**, **'титан'**:**'titanium'**, **'небесный'**:**'sky'**,**'арктик'**:**'arctic'**, **'индиан'**:**'indian'**,**'шоколадный'**:**'chocolate'**, **'мечта'**:**'dream'**, **'яркий'**:**'blaze'**, **'морской'**:**'navy'**, **'неоновый'**:**'neon'**, **'бриллиантовый'**:**'diamond'**, **'мятный'**:**'mint'**, **'медный'**:**'copper'**, **'либерти'**:**'liberty'**, **'фэйшн'**:**'fashion'**, **'платиновый'**:**'platinum'**, **'аква'**:**'aqua'**, **'индиго'**:**'indigo'**, **'сапфировый'**:**'sapphire'**, **'ледяной'**:**'ice'**, **'орхидея'**:**'orchid'**, **'пастельный'**:**'pastel'**, **'ваниль'**:**'vanilla'**, **'нефрит'**:**'jade'**, **'каштан'**:**'chestnut'**, **'опал'**:**'opal'**}  
 dct\_colours\_eng = {**'white'**:**'белый'**, **'blue'**: **'синий'**, **'red'**:**'красный'**, **'space'**: **'космос'**, **'pink'**:**'розовый'**, **'rose'**:**'розовое'**, **'cosmic'**:**'космический'**, **'topaz'**:**'топаз'**, **'brown'**:**'коричневый'**, **'ruby'**:**'рубиновый'**, **'pearl'**:**'жемчужный'**, **'deep'**:**'глубокий'**, **'warm'**:**'теплый'**, **'apple'**:**'яблочный'**, **'dark'**:**'темный'**, **'green'**:**'зеленый'**, **'light'**:**'светлый'**, **'wave'**:**'морской'**, **'metallic'**:**'металлик'**, **'charcoal'**:**'древесный уголь'**, **'mahogany'**:**'красное дерево'**, **'cream'**:**'кремовый'**, **'lime'**:**'лаймовый'**, **'blackberry'**:**'ежевичный'**, **'lilac'**:**'сиреневый'**, **'steel'**:**'стальной'**, **'midnight'**:**'полночь'**, **'misty'**:**'туманный'**, **'bronze'**: **'бронзовый'**, **'onyx'**:**'оникс'**, **'cyan'**:**'циан'**, **'magenta'**:**'пурпурный'**, **'coral'**:**'коралловый'**, **'cool'**:**'свежий'**, **'fuchsia'**:**'фуксия'**, **'orange'**:**'оранжевый'**, **'sand'**:**'песочный'**, **'golden'**:**'золотой'**, **'amber'**:**'янтарный'**, **'wine'**:**'винный'**, **'star'**:**'звездный'**, **'ultra'**:**'ультра'**, **'purple'**:**'фиолетовый'**, **'titanium'**:**'титан'**, **'magic'**:**'магический'**, **'vista'**:**'виста'**, **'sky'**:**'небесный'**, **'surf'**:**'сёрф'**, **'slate'**:**'грифельный'**, **'arctic'**:**'арктик'**, **'indian'**:**'индиан'**, **'chocolate'**:**'шоколадный'**, **'dream'**:**'мечта'**, **'navy'**:**'морской'**, **'neon'**:**'неоновый'**, **'diamond'**:**'бриллиантовый'**, **'mint'**:**'мятный'**, **'copper'**:**'медный'**, **'liberty'**:**'либерти'**, **'platinum'**:**'платиновый'**, **'aqua'**:**'аква'**, **'indigo'**:**'индиго'**, **'sapphire'**:**'сапфировый'**, **'ice'**:**'ледяной'**, **'orchid'**:**'орхидея'**, **'pastel'**:**'пастельный'**, **'vanilla'**:**'ваниль'**, **'jade'**:**'нефрит'**, **'chestnut'**:**'каштан'**, **'opal'**:**'опал'**}  
 *#два словаря, использующиеся, если в одной строке название цвета по-русски, в другой - по-английски* string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,""«»'')'**)  
 model = model.lower().split()  
 probability = 0  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,""«»'')'**)  
 **if** model[i].isalpha(): *#цвета не содержат цифр* **if** model[i] **in** set\_of\_colours\_eng **and** model[i] != **"apple" and** model[i] != **"blackberry"**: *#если токен - цвет по-английски, но не является моделью телефона, то* **if** model[i] == **'gray' or** model[i] == **'grey'**: *#обрабатываем серый цвет, а также далее другие 4 цвета - исключения из-за разности написания* **if 'gray' in** string\_to\_compare **or 'grey' in** string\_to\_compare **or 'серый' in** string\_to\_compare:  
 probability += 1  
 **elif** model[i] == **'black'**:  
 **if 'черный' in** string\_to\_compare **or 'чёрный' in** string\_to\_compare **or 'black' in** string\_to\_compare:  
 probability += 1  
 **elif** model[i] == **'yellow'**:  
 **if 'желтый' in** string\_to\_compare **or 'жёлтый' in** string\_to\_compare **or 'yellow' in** string\_to\_compare:  
 probability += 1   
 **elif** model[i] == **'gold'**:  
 **if 'золото' in** string\_to\_compare **or 'золотой' in** string\_to\_compare **or 'gold' in** string\_to\_compare:  
 probability += 1  
 **elif** model[i] == **'silver'**:  
 **if 'серебряный' in** string\_to\_compare **or 'серебристый' in** string\_to\_compare **or 'silver' in** string\_to\_compare:  
 probability += 1   
 **elif** model[i] **in** string\_to\_compare: *#если в другой строке есть совпадающий токен-цвет* probability += 1  
 **elif** model[i] **in** dct\_colours\_eng: *#если мы знаем перевод этого слова на русский* **if** dct\_colours\_eng[model[i]] **in** string\_to\_compare: *#и перевод этого слова присутствует в другой строке* probability += 1  
 **elif** model[i] **in** dct\_colours\_eng: *#если в английском файле не оказалось данного цвета, но мы ранее видели, что данный цвет используется в описании телефонов и мы записали его перевод,* **if** dct\_colours\_eng[model[i]] **in** string\_to\_compare: *#и его перевод оказался во второй строке* probability += 1   
 **elif** model[i] **in** set\_of\_colours\_rus: *#если токен - цвет по-русски* **if** model[i] **in** dct\_colours\_rus: *#если мы знаем перевод этого слова на английский* **if** dct\_colours\_rus[model[i]] **in** string\_to\_compare: *#и перевод этого слова присутствует в другой строке* probability += 1  
 **elif** model[i] **in** string\_to\_compare: *#если в другой строке есть совпадающий токен-цвет* probability += 1   
 **elif** model[i] == **'серый'**: *#проверяем цвет-исключение* **if 'gray' in** string\_to\_compare **or 'grey' in** string\_to\_compare **or 'серый' in** string\_to\_compare:  
 probability += 1  
 **elif** model[i] **in** dct\_colours\_rus: *#если в русском файле не оказалось данного цвета, но мы ранее видели, что данный цвет используется в описании телефонов и мы записали его перевод,* **if** dct\_colours\_rus[model[i]] **in** string\_to\_compare: *#и его перевод оказался во второй строке* probability += 1   
 **return** probability  
  
**def** levenstain(string\_to\_compare, model): *#функция, считающая расстояние Левенштейна между сплошными строками без пробелов* **from** nltk.metrics **import** edit\_distance  
 string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,\«»"\"'')'**)  
 string\_to\_compare = **''**.join(string\_to\_compare)  
 model = model.lower().split()  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 model = **''**.join(model)  
 **try**:  
 dist = edit\_distance(model, string\_to\_compare) / len(model)  
 **except**:  
 dist = -1  
 **return** dist  
  
*#четыре правила ниже отвечают за проверку совпадения памяти устройствв паре***def** mem\_normalize(mem): *#функция, которая переводит часть, отвечающую за единицу объема памяти, в 2 английские буквы* fir = mem.find(**'gb'**)  
 **if** fir > 0:  
 mem = mem[:fir] + **'gb'** sec = mem.find(**'гб'**)  
 **if** sec > 0:  
 mem = mem[:sec] + **'gb'** thir = mem.find(**'mb'**)   
 **if** thir > 0:  
 mem = mem[:thir] + **'mb'** four = mem.find(**'мб'**)  
 **if** four > 0:  
 mem = mem[:four] + **'mb'  
 return** mem  
   
**def** changing(memory): *#превращает в шаблон представление памяти устройства (считаем, что число в строке всегда идет перед размером памяти)  
 #память записывается в формате число слитно с 2мя английскими буквами, означающими объем памяти* result = []  
 **for** i **in** range(len(memory)):  
 memory[i] = memory[i].strip(**'(,\«»"\"'')'**)  
 **if** memory[i].find(**'gb'**) == 0 **or** memory[i].find(**'гб'**) == 0 **or** memory[i].find(**'мб'**) == 0 **or** memory[i].find(**'mb'**) == 0:  
 **if** i != 0 **and** memory[i-1].isdigit():  
 curr = memory[i - 1] + memory[i]  
 curr = mem\_normalize(curr)  
 result.append(curr)  
 **elif** memory[i].find(**'gb'**) > 0 **or** memory[i].find(**'гб'**) > 0 **or** memory[i].find(**'мб'**) > 0 **or** memory[i].find(**'mb'**) > 0:  
 result.append(mem\_normalize(memory[i]))  
 **return** result  
   
**def** mem\_storage(string\_to\_compare, model): *#сравнивает шаблонизированную память в 2х строках* string\_to\_compare = string\_to\_compare.lower().split()  
 model = model.lower().split()   
 mem\_str = changing(string\_to\_compare)  
 mem\_mod = changing(model)  
 prob = 0  
 **for** i **in** range(len(mem\_mod)):  
 **if** mem\_mod[i] **in** mem\_str:  
 prob += 1  
 **return** prob   
  
**def** is\_it\_mem\_st(mem\_str): *#проверяет, имеет ли токен отношение к памяти устройства* **if** mem\_str.find(**'gb'**) >= 0 **or** mem\_str.find(**'гб'**) >= 0 **or** mem\_str.find(**'мб'**) >= 0 **or** mem\_str.find(**'mb'**) >= 0:  
 **return True  
 else**:  
 **return False  
  
def** name\_model(string\_to\_compare, model): *#функция, сравнивающая токены, которые являются названиями модели* set\_of\_colours\_eng = colour\_parser\_eng() *#множество всех цветов по-английски* set\_of\_colours\_rus = colour\_parser\_rus() *#множество всех цветов по-русски* string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,""'')'**)  
 model = model.lower().split()  
 probability = 0  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 **if** model[i] == **'apple' or** model[i] == **'blackberry'**: *#модели-исключения, так как одновременно являются и цветами* **if** model[i] **in** string\_to\_compare:  
 probability += 1  
 **if** model[i].isalpha() **and not** is\_it\_mem\_st(model[i]) **and** model[i] **not in** set\_of\_colours\_eng **and** model[i] **not in** set\_of\_colours\_rus: *#проверяем, не является ли буквенный токен цветом или памятью устройства* **if** model[i] **in** string\_to\_compare: *#если нет, то это скорее всего название модели, поэтому мы ищем такое же во 2й строке* probability += 1   
 **if** probability == 0: *#усиливаем важность правила* probability = -1  
 **return** probability  
  
**def** num\_model(string\_to\_compare, model): *#также в этом правиле может выполняться сравнение года выпуска модели,  
 #функция, сравнивающая токены, состоящие из цифр (не являющиеся объемом памяти модели), то есть номера модели* string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,\«»"\"'')'**)  
 model = model.lower().split()  
 param\_kol = len(model)  
 probability = 0  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 **if** model[i].isdigit():  
 **if** i < len(model) - 1: *#здесь мы проверяем,что число не является размером памяти устройства (с учетом стандартного построения словосочетания - объем, единица объема)* **if** model[i + 1] != **'mb' and** model[i + 1] != **'gb' and** model[i + 1] != **'мб' and** model[i + 1] != **'гб'**:  
 **if** model[i] **in** string\_to\_compare:  
 probability += 1  
 **else**:  
 **if** model[i] **in** string\_to\_compare:  
 probability += 1   
 **if** probability == 0: *#усиливаем важность правила* probability = -1  
 **return** probability  
  
**def** check\_series(str\_ch): *#правило, проверяющее, что токен является серией, то есть состоит не только из букв/цифр, не является нормализованной строкой объема памяти устройства* **if not** str\_ch.isalpha() **and not** str\_ch.isdigit() **and** str\_ch.find(**'gb'**) < 0 **and** str\_ch.find(**'mb'**) < 0 **and** str\_ch.find(**'гб'**) < 0 **and** str\_ch.find(**'мб'**) < 0:  
 **return True   
 else**:  
 **return False  
   
def** series(string\_to\_compare, model): *#правило, сравнивающее токены-серии телефонов* string\_to\_compare = string\_to\_compare.lower().split()  
 **for** i **in** range(len(string\_to\_compare)):  
 string\_to\_compare[i] = string\_to\_compare[i].strip(**'(,""'')'**)  
 model = model.lower().split()  
 **for** i **in** range(len(model)):  
 model[i] = model[i].strip(**'(,\«»"\"'')'**)  
 sum\_prob = 0  
 size = len(model)  
 length = len(string\_to\_compare)  
 **for** i **in** range(size):  
 **if** check\_series(model[i]): *#если очередной токен - серия* **for** j **in** range(length): *#рассматриваем все токены второй строки* **if** check\_series(string\_to\_compare[j]): *#если токен во 2 строке тоже серия* flag = 0  
 **if** model[i].find(string\_to\_compare[j]) >= 0: *#проверяем, не является токен 1й строки, который мы рассматриваем, подсловом токена 2й строки, и наоборот* flag = 1  
 **elif** string\_to\_compare[j].find(model[i]) >= 0:  
 flag = 1  
 sum\_prob += flag *#если является, учитываем это* **if** sum\_prob == 0: *#необходимо проверить, нет ли во 2й строке серии, расклеенной по 2м или 3м токенам (больше обычно не бывает), разделенной на буквы и цыфрв (PRO6 -> pro 6)  
 #тогда найденная такая же серия в первой строке, но в привычнолм виде, не смэтчится с расклеенной (если расклейка произошла в 1й строке, то токены просто не воспримутся, как серия)* adglutinate = []   
 **for** i **in** range(length - 1): *#склеиваем 2ую строку по 2 токена в прямом порядке* adglutinate.append(string\_to\_compare[i] + string\_to\_compare[i + 1])  
 **for** i **in** range(length - 2): *#по 3 токена в прямом* adglutinate.append(string\_to\_compare[i] + string\_to\_compare[i + 1] + string\_to\_compare[i + 2])  
 **for** i **in** range(1, length): *#по 2 обратном порядке* adglutinate.append(string\_to\_compare[i] + string\_to\_compare[i - 1])  
 **for** i **in** range(2, length): *#по 3 в обратном (с конца в начало)* adglutinate.append(string\_to\_compare[i] + string\_to\_compare[i - 1] + string\_to\_compare[i - 2])   
 **for** i **in** range(size):   
 **if** check\_series(model[i]): *#если очередной токен - серия* **for** j **in** range(len(adglutinate)): *#рассматриваем все токены из списка склеек из 2й строки* **if** check\_series(adglutinate[j]): *#если токен в списке склеек из 2й строки тоже серия* flag = 0  
 **if** model[i].find(adglutinate[j]) >= 0: *#проверяем, не является токен 1й строки, который мы рассматриваем, подсловом токена склеек из 2й строки, и наоборот* flag = 1  
 **elif** adglutinate[j].find(model[i]) >= 0:  
 flag = 1  
 sum\_prob += flag *#если является, учитываем это* **if** sum\_prob == 0: *#стараемся сделать правило с серией самым главным* sum\_prob = -1.5  
 **return** sum\_prob  
  
**def** price\_abs(price\_to\_compare, price): *#правило, возвращающее модуль разницы цены телефонов из пары* **if** price\_to\_compare == **"NaN" or** price\_to\_compare == **"NaN"**:  
 **return** -999  
 **return** abs(price\_to\_compare - price)  
  
**def** price\_comp(price\_to\_compare, price): *#правило, возвращающее процент модуля разницы цены телефонов от цены телефона из стороннего интернет-магазина* **if** price\_to\_compare == **"NaN" or** price\_to\_compare == **"NaN"**:  
 **return** -999  
 **return** abs(price\_to\_compare - price) / price

**Обучение и оценка «random forest»**

**import** pandas **as** pd  
info\_true\_arts\_1c = pd.read\_excel(**"shop\_mts\_with\_prices.xlsx"**) *#записываем в df информацию о телефонах интернет-магазина мтс с ценами*external\_phones = pd.read\_excel(**"pattern\_file.xlsx"**) *#записываем в df информаицю о телефонах из других интернет-магазинов с ценами*external\_phones = external\_phones[[**'title'**, **'price'**]].drop\_duplicates() *#оставляем только нужные нам колонки с краткой информацией о телефоне и его цене без дубликатов*known\_pairs = pd.read\_csv(**'Пары\_найденные\_в\_базе.txt'**, **'>'**, header=**None**, names=[**'phone{}'**.format(i) **for** i **in** range(1,3)]) *#записываем в df пары, вручную размеченные и найденные ранее в базе(аболютно точно правильные),  
#и собранные в определенном формате в файл.*known\_pairs = pd.concat([known\_pairs, known\_pairs.rename(columns={**'phone1'**:**'phone2'**, **'phone2'**:**'phone1'**})]).drop\_duplicates() *#записываем в эту же df пары, написанные в обратном порядке,  
#чтобы не пропустить уже найденные пары*external\_phones[**'ones'**] = 1  
info\_true\_arts\_1c[**'ones'**] = 1 *# как и в основном скрипте для запуска, создаем две одинаковые колонки для того, чтобы сделать декартово произведение телефонов в 2х таблицах*info\_true\_labeled = external\_phones.loc[external\_phones[**'title'**].isin(known\_pairs.phone2)] *#записываем в df только те строки из таблицы сторонних телефонов, в которых присутствуют телефоны,  
#у которых найдены и записаны в файле, отмеченном выше, пары в базе мтс, чтобы в декартовом произведении присутствовали лишь те телефоны из сторонних магазинов, у которых мы знаем  
#верную пару из базы мтс*all\_pairs = pd.merge(info\_true\_labeled.reset\_index(), info\_true\_arts\_1c, on=**'ones'**).drop(**'ones'**, axis=1) *#декартово произведение пар, удалена лишняя колонка 1*all\_pairs = all\_pairs.set\_index([**'Name'**, **'title'**]) *#делаем индексом df пару телефонов для удобства подготовки данных для обучения*known\_pairs[**'ones'**] = 1  
all\_pairs[**'Match'**] = known\_pairs.set\_index([**'phone1'**,**'phone2'**])[**'ones'**] *#сделав так же индексом пару телефонов в таблице верных пар, проставляем в общей таблице пар единицы напротив пар,  
#которые являются верными*all\_pairs[**'Match'**] = all\_pairs[**'Match'**].fillna(0) *#все остальные пары являются по построению неправильными, поэтому их размечаем 0*all\_pairs = all\_pairs.reset\_index() *#сбрасываем индекс и индексируем с 0*all\_pairs = pd.concat([all\_pairs[all\_pairs[**'Match'**]==1],  
 all\_pairs[all\_pairs[**'Match'**]==0]\  
 .sample(int(all\_pairs[**'Match'**].sum()\*10), random\_state=0) *# disbalance remains* ]).sample(frac=1, random\_state=0) *# shuffle #для того, чтобы данные не вызвали неправильного обучения, перемешиваем   
 #пары, размеченные 0 и 1, и пар с 0 (которых, очевидно, во много раз больше, чем правильных) берем в 10 раз больше, чем пар с 1, балансируя выборку*all\_pairs = all\_pairs.rename(columns={**"price"**: **"Price"**, **"title"**: **"Real"**, **"Name"**: **"Top\_guess"**}).drop(**'index'**, axis=1).reset\_index().fillna(-999)  
*#приводим таблицу в удобный для обучения вид - переименовываем и удаляем лишние колонки, заполняем пропущенные цены -999, переиндексируем***from** all\_rules **import** jaccard **as** rule\_jaccard, bigrams **as** rule\_bigrams, name\_model **as** rule\_model, series **as** rule\_series, num\_model **as** rule\_num\_model, mem\_storage **as** rule\_mem\_storage, colour **as** rule\_colour, levenstain **as** rule\_levenstain  
**from** all\_rules **import** price\_comp **as** rule\_price\_comp, price\_abs **as** rule\_price\_abs  
  
price\_rules = [(**'Prob\_abs\_price'**, rule\_price\_abs), (**'Prob\_comp\_price'**, rule\_price\_comp)]  
rules = [(**'Prob\_jaccard'**, rule\_jaccard), (**'Prob\_bigrams'**, rule\_bigrams), (**'Prob\_model'**, rule\_model), (**'Prob\_series'**, rule\_series),  
(**'Prob\_num\_model'**, rule\_num\_model), (**'Prob\_mem\_storage'**, rule\_mem\_storage), (**'Prob\_colour'**, rule\_colour), (**'Prob\_levenstain'**, rule\_levenstain)]  
  
**def** calc\_rules(info\_top, col1, col2):  
 **for** i **in** range(len(rules)):  
 colname, foo = rules[i]  
 info\_top[colname] = info\_top.apply(**lambda** row: foo(row[col1], row[col2]), axis=1)  
 **return** info\_top  
  
**def** calc\_price(info\_top, col1, col2):  
 **for** i **in** range(len(price\_rules)):  
 colname, foo = price\_rules[i]  
 info\_top[colname] = info\_top.apply(**lambda** row: foo(row[col1], row[col2]), axis=1)  
 **return** info\_top   
*#часть с 29 по 46 строку такая же в файле основного скрипта, где приведены комментарии*all\_pairs = calc\_rules(all\_pairs, col1=**'Real'**, col2=**'Top\_guess'**) *#рассчитываем для подготовленных данных вероятность совпадения пар по 8 правилам на информацию о телефонах*all\_pairs = calc\_price(all\_pairs, col1=**'Price'**, col2=**"Price\_mts"**) *#-||- по правилам на цены*all\_pairs = all\_pairs.drop\_duplicates(subset=[**"Real"**, **"Top\_guess"**, **"Price"**]) *#убираем дубликаты*feature\_names = [x[0] **for** x **in** rules] + [x[0] **for** x **in** price\_rules] *#список строковых названий фич (правил), использующихся для отбора пар*X = all\_pairs[feature\_names]  
*#X - df, состоящая из вероятностей совпадения каждой из проверяемых пар по числу правил, найденных алгоритмом, т.е. на пару телефон - телефон  
#возвращается строка из 8 вероятностей совпадения телефонов, возвращенных правилами. Эти вероятности мы отбираем в Х.*y = all\_pairs[:][**'Match'**] *#y - разметка пар 0 и 1 в зависимости от их совпадения***from** sklearn **import** ensemble  
rf = ensemble.RandomForestClassifier(n\_estimators=15, max\_depth=7)  
rf = rf.fit(X, y) *#обучаем модель на выборке***from** sklearn.externals **import** joblib  
joblib.dump(rf, **'random\_forest.pkl'**) *#сохраняем обученную модель***from** sklearn.model\_selection **import** cross\_val\_predict  
**from** sklearn.metrics **import** f1\_score  
y\_pred = cross\_val\_predict(rf, X, y, cv=10) *#оцениваем работу модели*print(**"f1\_score:"**, f1\_score(y, y\_pred, average=**'binary'**), **"\n"**)

**Построение PR-curve**

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.metrics **import** precision\_recall\_curve  
prob\_in\_shops = pd.read\_excel(**'file.xlsx'**) *#здесь должен быть файл, в котором стоит отдельной колонкой разметка 0 и 1 пар, абсолютно верная, пары и вероятности, возвращенные random forest*prob\_in\_shops = prob\_in\_shops.dropna() *#не должно быть пустых ячеек, иначе график не построится*reals\_list = prob\_in\_shops[**"Match"**].tolist() *#match пока проставляется руками*match\_prob\_list = prob\_in\_shops[**"Probability"**].tolist()  
y\_true = np.array(reals\_list)  
y\_scores = np.array(match\_prob\_list)  
precision, recall, threshold = precision\_recall\_curve(y\_true, y\_scores) *#рассчитываем на основе данных*plt.clf()  
plt.plot(recall, precision, lw=2, color=**'navy'**,  
 label=**'Precision-Recall curve for matches'**)  
plt.xlabel(**'Recall'**)  
plt.ylabel(**'Precision'**)  
plt.ylim([0.0, 1.05])  
plt.xlim([0.0, 1.0])  
plt.legend(loc=**"lower left"**)  
plt.show()  
plt.savefig(**'Precision-Recall curve for matches.pdf'**) *#строим pr-curve*

**Общий алгоритм с комментариями**

**import** pandas **as** pd  
*#Здесь пока идет кусок с объяснением, как пользоваться результатом прошлой работы алгоритма, если изменение в числе телефонов происходит в сторонних интернет-магазинах;  
#так как требуется выполнение противоположной задачи, а именно, учитывать изменения в нашем - мтс - интернет-магазине, код будет изменен.*info\_checked = pd.read\_excel(**"Top\_match.xlsx"**) *#записываем в df результаты (найденные пары) работы прошлого запуска алгоритма (необходимо для упрощения поиска новых пар, так как  
#новые телефоны добавляются в небольшом количестве, а старые пары, найденные правильно, остаются актуальными)*info\_other\_shops = pd.read\_excel(**"pattern\_file.xlsx"**) *#записываем в df информацию о телефонах из интернет-магазинов (не мтс, сторонних)*info\_other\_shops = info\_other\_shops.rename(columns={**"price"**: **"Price"**, **"title"**:**"Real"**}) *#переименовываем колонки с ценой телефонов из других интернет-магазинов и их кратким описанием  
#(серия, номер, модель, цвет, память устройства) для того, чтобы можно было соединить информацию из 2х таблиц*top\_pair\_known = info\_checked.merge(info\_other\_shops, on=**"Real"**).dropna().drop\_duplicates(subset=**"Real"**) *#соединяем информацию из 2х таблиц с помощью колонки, содержащей краткую информацию о телефонах,  
#чтобы получить df, содержащий пары телефонов, которые не придется уже мэтчить с помощью алгоритма (отличается от таблицы info\_checked тем, что там встречаются телефоныв, проверенные алгоритмом, на которым  
#не удалось найти пару)*top\_pair\_known = top\_pair\_known[[**"Real"**,**"Top\_match"**]].reset\_index() *#оставляем в таблице 2 колонки - найденную пару, индексируем с 0***del** top\_pair\_known[**"index"**]*#удаляем ненужную колонку со старыми индексами*info\_other\_shops[**'Model\_exists'**] = ~info\_other\_shops[**"Real"**].isin(top\_pair\_known[**"Real"**]) *#создаем в df сторонних телефонов колонку, показывающую, была ли найдена ранее пара телефону  
#из другого интернет-магазина в базе телефонов мтс. Была - false, не была - true*info\_other\_shops = info\_other\_shops.loc[info\_other\_shops[**'Model\_exists'**]] *#оставляем только те телефоны, которым пары в базе телефонов мтс найдены не были*info\_other\_shops = info\_other\_shops.rename(columns={**"Real"**:**"title"**}) *#переименовываем обратно колонку с краткой информаицей о телефонах для алгоритма*info\_mts = pd.read\_excel(**"shop\_mts\_with\_prices.xlsx"**) *#записываем в df информацию о телефонах из интрнет-магазина мтс*info\_other\_shops[**'ones'**] = 1  
info\_mts[**'ones'**] = 1 *#для того, чтобы сделать декартово произведение 2х таблиц - телефонов из других интрнет-магазинов и магазина мтс, то есть рассмотреть всевозможные пары телефоны,  
#которые получаются при соединении каждого телефона из 1ой таблицы со всеми телефонами другой, создаем 2 одинаковые колонки в обеих таблицах*all\_pairs = info\_other\_shops.merge(info\_mts, how=**'outer'**, on=**'ones'**).fillna(-999) *#создаем таблицу декартовых произведений. Так как не у всех телефонов записаны цены, то  
#недостающие цены заполняем -999, что не меняет выбор random forest, но упрощает его использование***del** all\_pairs[**"store"**]  
**del** all\_pairs[**"datetime"**]  
**del** all\_pairs[**"region"**]  
**del** all\_pairs[**"Model\_exists"**] *#удаляем лишние колонки***from** all\_rules **import** jaccard **as** rule\_jaccard, bigrams **as** rule\_bigrams, name\_model **as** rule\_model, series **as** rule\_series, num\_model **as** rule\_num\_model, mem\_storage **as** rule\_mem\_storage, colour **as** rule\_colour, levenstain **as** rule\_levenstain  
**from** all\_rules **import** price\_comp **as** rule\_price\_comp, price\_abs **as** rule\_price\_abs *#импортируем 8 правил на краткую информаицю о телефоне (правило на совпадение серии, номера,  
#названия модели, совпадения цвета и памяти устройства, а также 3 правила на проверку сходства 2х предложений: расстояние Левенштейна, мера Жаккара и количество совпавших биграмм)  
#импортируем 2 правила, возвращающих абсолютную и относительную разницу между ценами телефонов. Импортируем из файла all\_rules.py с написанными мной правилами.*price\_rules = [(**'Prob\_abs\_price'**, rule\_price\_abs), (**'Prob\_comp\_price'**, rule\_price\_comp)]  
rules = [(**'Prob\_jaccard'**, rule\_jaccard), (**'Prob\_bigrams'**, rule\_bigrams), (**'Prob\_model'**, rule\_model), (**'Prob\_series'**, rule\_series),  
(**'Prob\_num\_model'**, rule\_num\_model), (**'Prob\_mem\_storage'**, rule\_mem\_storage), (**'Prob\_colour'**, rule\_colour), (**'Prob\_levenstain'**, rule\_levenstain)]  
*# создаем списки кортежей ("название колонки, в которой будет содержаться вероятность сходства пар, возвращенная конкретным правилом", названием, под которым импортировано правило),  
# отдельно для правил с ценами, отдельно для правил с информацией о телефонах (нужно для того, чтобы применение правил можно было описать в простом цикле)***def** model\_match(string\_to\_compare, model):  
 token1 = string\_to\_compare.split()[0].lower().strip(**'(,\"\"'')'**)  
 token2 = model.split()[0].lower().strip(**'(,\"\"'')'**)  
 **return** token1 == token2 *#правило, отмечающее пары, которые почти точно не являются описанием одного и того же телефона. Оно основано на том, что  
 #несовпадение первого токена почти точно, по тому, как представлена информация о телефонах, означает, что телефоны разных моделей, то есть нужная пара не была найдена, потому  
 #что в базе мтс такой модели нет***def** calc\_rules(info\_top, col1, col2):  
 **for** i **in** range(len(rules)):  
 colname, foo = rules[i]  
 info\_top[colname] = info\_top.apply(**lambda** row: foo(row[col1], row[col2]), axis=1)  
 **return** info\_top *#функция, которая применяет к паре телефонов 8 правил, проверяющих совпадение информации о телефонах, и записывает результат в колонку правила в df***def** calc\_price(info\_top, col1, col2):  
 **for** i **in** range(len(price\_rules)):  
 colname, foo = price\_rules[i]  
 info\_top[colname] = info\_top.apply(**lambda** row: foo(row[col1], row[col2]), axis=1)  
 **return** info\_top *#функция, которая применяет к ценами пары телефонов правила, связанные со стоимостью моделей, и записывает результат в колонку правила в df***import** seaborn **as** sns  
**import** numpy **as** np  
**from** multiprocessing **import** Pool  
  
num\_partitions = 50 *#number of partitions to split dataframe*num\_cores = 50

**def** parallelize\_dataframe(df, func):  
 df\_split = np.array\_split(df, num\_partitions)  
 pool = Pool(num\_cores)  
 df = pd.concat(pool.map(func, df\_split))  
 pool.close()  
 pool.join()  
 **return** df *#разделение df на num\_of\_partitions, чтобы распараллелить и увеличить скорость выполнения алгоритма, и применение алгоритма ко всем парам all\_pairs (func - параметр parallelize\_dataframe)***def** probability(prob, name):  
 **if** prob >= 0.7: *#порог отсечения (примерно такой найден был после построения pr-curve по полученным результатам)* **return** name *#фунция, отсекающая по порогу вероятности пары, которые можно считать верными и те, оторые алгоритм нашел неправильно, например, потому, что такого телефона в базе мтс нет***def** in\_base(matched, name):  
 **if** matched == **True**:  
 **return** name *#функция, проверяющая, был ли найден в базе мтс такой телефон (основывается на результатах правила model\_match)***def** multiply\_all\_pairs(all\_pairs):  
 all\_pairs[**'Model\_match'**] = all\_pairs.apply(**lambda** row: model\_match(row[**'title'**], row[**'Name'**]), axis=1) *#создание колонки, где каждой паре соответствует true, если первые токены совпадают,  
 #false - иначе* all\_pairs = calc\_rules(all\_pairs, col2=**'Name'**, col1=**'title'**) *#Name - колонка телефонов из базы мтс, title - из сторонних интернет-магазинов* all\_pairs = calc\_price(all\_pairs, col1=**'Price\_mts'**, col2=**'Price'**)  
 **return** all\_pairs *#основная функция, отвечающая за рассчет всех правил совпадения/не совпадения моделей в паре, записывающая в df по колонкам полученныке вероятности*all\_pairs = parallelize\_dataframe(all\_pairs, multiply\_all\_pairs) *#применяем алгорим ко всем получившимся парам, чтобы узнать вероятность того, что они описывают один и тот же телефон.  
#Распараллеливаем работу алгоритма.***from** sklearn.externals **import** joblib  
rf = joblib.load(**'random\_forest.pkl'**) *#загружаем обученный ранее на имеющихся правилах random forest*feature\_names = [x[0] **for** x **in** rules] + [x[0] **for** x **in** price\_rules] *#список строковых названий фич (правил), использующихся для отбора пар*X = all\_pairs[feature\_names] *#X - df, состоящая из вероятностей совпадения каждой из проверяемых пар по числу правил, найденных алгоритмом, т.е. на пару телефон - телефон  
#возвращается строка из 8 вероятностей совпадения телефонов, возвращенных правилами. Эти вероятности мы отбираем в Х.*all\_pairs[**'Probability'**] = rf.predict\_proba(X)[:,1] *#В общем dataframe, содержащем пары и информацию о их совпадении, создается столбец итоговых вероятностей совпадения телефонов внутри пар,  
#предсказанных на основе вероятностей, посчитанных каждым правилом. Итоговые вероятности определяет random forest.*all\_pairs = all\_pairs.sort\_values(**'Probability'**, ascending=**False**) *#сортируем пары по убыванию вероятности их совпадения*top\_pair = all\_pairs.groupby(**'title'**, as\_index=**False**).first() *#first - выбор лучшей пары (с самой большой вероятностью совпадения), top\_pair - табличка с лучшими совпадениями*top\_pair[**"Name"**] = top\_pair.apply(**lambda** row: in\_base(row[**'Model\_match'**], row[**'Name'**]), axis=1) *#убираем пару, найденную телефону в базе мтс, так как пара почти точно неправильная,   
#в силу отсутствия телефона данной модели в базе мтс*top\_pair[**"Name"**] = top\_pair.apply(**lambda** row: probability(row[**'Probability'**], row[**'Name'**]), axis=1) *#убираем пару, найденную телефону в базе мтс, так как вероятность  
#совпадения телефонов оказалась ниже порога, который показывает точность результата работы алгоритма*top\_pair = top\_pair.rename(columns={**"title"**: **"Real"**, **"Name"**: **"Top\_match"**}) *#переименовываем колонки таким образом, чтобы "real" отвечала за телефоны из других магазинов,  
#top\_match за пары, найденные этим телефонам в базе мтс*cols = top\_pair.columns.tolist()  
cols = cols[0:1] + cols[3:4]   
top\_pair = top\_pair[cols] *#оставляем только 2 колонки - пары телефонов, найденные алгоритмом*top\_pair = pd.concat([top\_pair, top\_pair\_known]) *#присоединям к результату старые пары, найденные алгоритмом еще при ранних запусках*top\_pair.to\_excel(**"Top\_match.xlsx"**) *#Это итоговая табличка, в которой записаны результаты*