Федеральное государственное образовательное бюджетное учреждение

высшего образования

**«ФинансоВЫЙ УНИВЕРСИТЕТ**

**при Правительстве Российской Федерации»**

**(Финансовый университет)**

**Факультет информационных технологий и анализа больших данных**

**Департамент анализа данных и машинного обучения**

**КУРСОВАЯ РАБОТА**

по дисциплине: «Технологии анализа данных и машинного обучения»

по теме: **«Сравнение методов регрессии на реальных наборах данных»**

Выполнил: студент

группы ПИ19-3

Маринин Р.Г.

Руководитель: доцент, к/н   
Андриянов Никита Андреевич

Москва, 2022

**Содержание**

1. Введение (стр. 3)
2. Описание задачи регрессии (стр. 5)
3. Постановка задачи (стр. 9)
4. Описание предметной области (стр. 16)
5. Описание программы (стр. 22)
6. Заключение (стр. 37)
7. Список материалов (стр. 39)

**Введение**

На сегодняшний день информационные технологии стали практически неотъемлемой частью нашей жизни. Они активно внедряются или уже внедрены в различные области человеческой деятельности. Производство, государственные службы, всевозможные сферы услуг – везде присутствует необходимость в современных методах обработки огромных объемов данных, которые поступают каждый день. Человеческие возможности достаточно серьезно ограничены, когда дело касается работы с колоссальными потоками информации. Машины делают то, на что не способны люди, компенсируют их недостатки. Они могут обрабатывать десятки тысяч строк данных за считанные секунды. Но и у машин есть свои слабости. У них нет интуиции, нет чувств, которыми обладает человек. Они действуют по логическим алгоритмам и являются идеальными исполнителями. Сама суть программирования заключается в том, чтобы дать компьютеру инструкцию действий, которые он должен выполнять, на языке, который он поймет. Точно так же, как мы говорим человеку, что ему нужно сделать. У компьютера есть память, причем, в отличие от человеческой, абсолютно контролируемая. А при наличии памяти и способности принимать и исполнять указания появляется возможность чему-то научиться. Отсюда мы делаем логичный вывод: машины могут быть обучены человеком. Причем, как уже было сказано ранее, они не имеют человеческих чувств и руководствуются логическими алгоритмами. Значит, эти алгоритмы и являются ключом, посредством их мы можем научить машины выполнять поставленные задачи.

Итак, мы плавно подошли к понятию машинного обучения. Что оно из себя представляет? По сути, это один из типов программного обеспечения. Но здесь мы не решаем задачу напрямую, а учим компьютер находить верное решение посредством его "тренировки" на подобных задачах с правильными ответами. Это похоже на обучение человека: чтобы он начал правильно решать какие-то задания, нужно показать ему пример, а лучше несколько, чтобы он выработал алгоритм в своей голове. Представим обыкновенного студента. Он на занятиях решает какие-то типовые задания, которые ему даёт преподаватель, после чего пишет контрольную работу с такими же заданиями, за которую получает определённую оценку. В машинном обучении всё происходит по аналогии с этим примером. Имеется определённый набор данных, другими словами, датасет. Набор "типовых задач" для машины – это обучающая выборка из датасета. Это значит, что мы берём большую часть имеющихся данных и используем их для обучения программы. Оставшаяся часть датасета – тестовая выборка, которая является "контрольной работой" для машины. Оценка ставится из сравнения реального набора "ответов" с тем, что выдала программа. Как происходит оценивание – рассмотрим далее. А сейчас поговорим о регрессии.

**Описание задачи регрессии**

Теперь немного конкретнее о нашей задаче. Тема звучит так: "Сравнение методов регрессии на реальных наборах данных". Очевидно, необходимо дать понятие регрессии. Как уже было сказано ранее, машина руководствуется логическими алгоритмами. В нашем случае, этим алгоритмом является математическая функция. С её помощью даётся приблизительное описание имеющегося набора данных. Это позволяет программе предугадывать те значения, которых у нас ещё нет, на основе только исходных данных. Короче говоря, регрессия – это математическая функция, которая аппроксимированно описывает данные и применяется для прогнозирования. В зависимости от метода регрессии, мы будем применять ту или иную функцию. Это может быть как обыкновенная прямая линия, так и нелинейный график. Если мы собираемся строить модели на основе математических функций, то там должны присутствовать свои аргументы и значения функции (проще говоря, x и y). В регрессионной модели присутствуют зависимая (или эндогенная) переменная и одна или несколько независимых (экзогенных). Эндогенная переменная – это та переменная, значения которой вычисляются на основе экзогенных. Наборы значений экзогенных переменных задаются изначально.

Как строится регрессионная модель? Поначалу у нас есть функция в общем виде, например, простая линейная функция: y = a + bx. Здесь a – это свободный параметр, а b – параметр независимой переменной (количество таких параметров зависит от количества предикторов, то есть экзогенных переменных). Чтобы построить модель, нужно подобрать такие параметры, чтобы функция максимально точно описывала набор данных. Для осуществления такого подбора есть специальные математические методы. Самым точным из них является метод наименьших квадратов. Он будет осуществляться автоматически, но нужно понимать его суть. В чем заключается метод наименьших квадратов? Допустим у нас есть датасет определенной длины со столбцом значений зависимой переменной y и несколькими столбцами со значениями независимых переменных x. Наша задача – максимально точно аппроксимировать данные первого столбца с помощью функции, строящейся на значениях экзогенных переменных, умноженных на неизвестные параметры. По сути, весь датасет – переопределенная система уравнений, где искомыми величинами являются эти самые параметры. Наблюдаемое значение y равняется сумме значения нашей функции и случайной ошибки. Эта ошибка высчитывается как разность между прогнозом и реальным значением в данной строке (проще говоря, остаток). Соответственно, мы имеем дело с задачей оптимизации функции: нужно подобрать такие параметры, чтобы минимизировать ошибку. Решается эта задача с помощью матричных операций, в которых присутствует вычисление обратной матрицы. И этот пункт составляет основной недостаток метода. Матрица может быть вырожденной (т.е. с нулевым определителем) и не иметь ввиду этого обратной матрицы. Кроме того, сложность этого вычисления имеет кубическую зависимость от количества предикторов в датасете, что делает метод слишком трудоёмким при наличии большого их количества. Кроме того, если модель слишком сложна, может произойти переобучение, от чего в итоге пострадает её точность. Переобучение связано как с количеством предикторов, так и с величинами параметров модели, поэтому есть метод борьбы с этим, который будет описан позднее.

Чтобы сравнить методы регрессии, необходимо будет оценивать модели, построенные в соответствии с этими методами. Для этого задействуем два числовых параметра, отображающие качество и точность регрессионной модели: коэффициент детерминации и среднеквадратическую ошибку. Коэффициент детерминации – это квадрат общего коэффициента корреляции между зависимой переменной и всеми независимыми. Также его можно высчитывать как долю объясненной дисперсии (ESS) в общей (TSS). Формула следующая:

R^2=\frac {ESS} {TSS}

Так как корреляция подразумевает статистическую зависимость, коэффициент детерминации показывает, насколько точно регрессионная модель описывает данные. По этому параметру оценивается качество модели. Чем лучше регрессия описывает данные, тем он больше (максимальное значение – единица). Принято считать, что при приемлемом качестве модели значение коэффициента детерминации превышает 0,5. Среднеквадратическая ошибка (Mean Squared Error или MSE) – это сумма квадратов остатков регрессионной модели, деленная на объем выборки. Формула MSE:

Изображение выглядит как текст, часы

Автоматически созданное описание

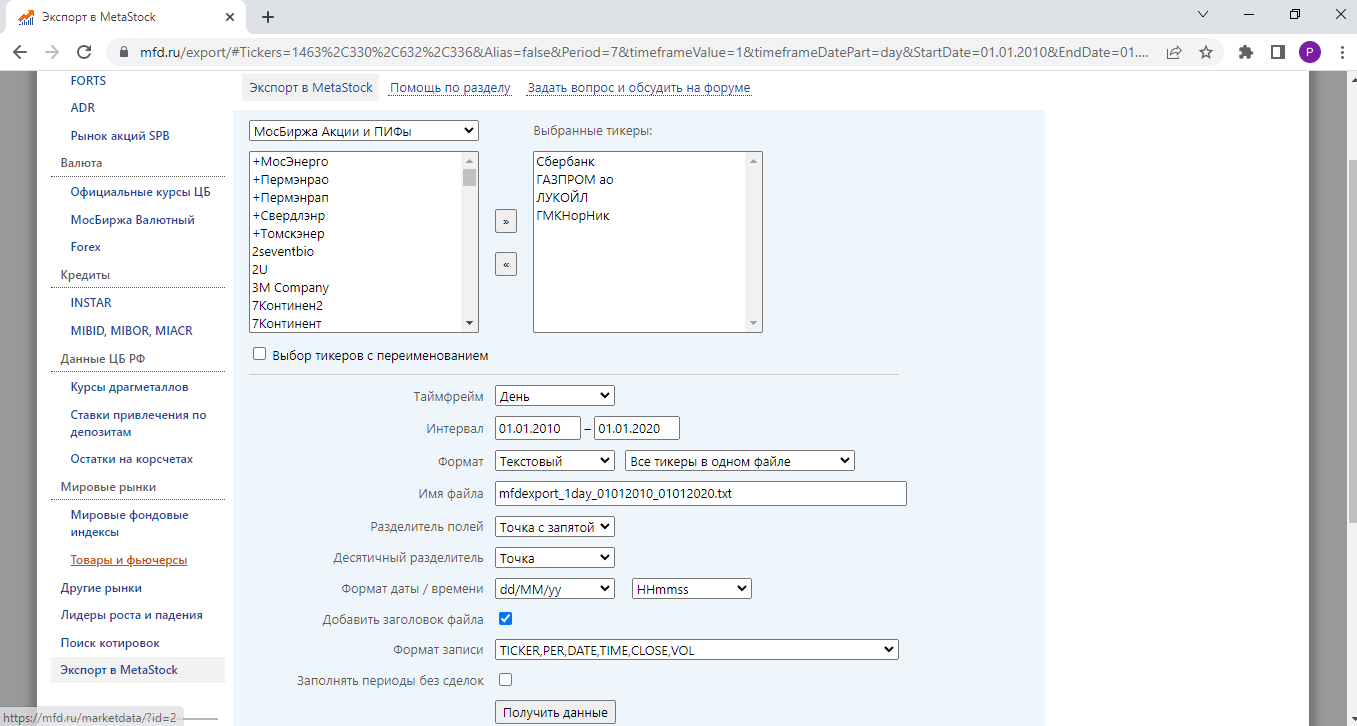
Остатки представляют из себя разность между наблюдаемыми значениями и спрогнозированными, проще говоря, они являются показателем того, насколько ошиблась модель. Соответственно, чем больше среднеквадратическая ошибка, тем модель хуже. Но, в отличие от коэффициента детерминации, MSE не является долевой величиной и зависит от среднего значения набора данных. Например, если значения эндогенной переменной одной модели измеряются в тысячах, а другой – в дробях от единицы, то при пропорциональном отклонении спрогнозированных значений от реальных коэффициенты детерминации этих моделей будут равны, а вот среднеквадратические ошибки, как и среднеквадратические отклонения, будут сильно разниться. Этот факт сыграет в работе важную роль, так как мы собираемся сравнивать различные методы регрессии, и в некоторых из них данные придётся шкалировать ввиду особенностей их алгоритмов, которые мы рассмотрим в постановке задачи.

Помимо коэффициента детерминации и среднеквадратической ошибки нам нужен будет график, чтобы визуально анализировать регрессионную модель. Но если она будет иметь более одного предиктора, то показать её на двумерном графике не получится, так как количество измерений равняется количеству независимых переменных. Поэтому целесообразно будет показать на графике не саму регрессионную функцию, а распределение остатков модели. Будем использовать так называемый Residuals Plot, то есть график остатков. По сути, это диаграмма рассеяния, отображающая влияние величин спрогнозированных значений эндогенной переменной на их отклонения от наблюдаемых значений этой самой переменной. Как интерпретировать такой график? Нужно обратить внимание на распределение остатков. Если они разбросаны случайно – между ними и прогнозами зависимой переменной нет статистической связи, или она незначительна. В таком случае говорят, что остатки гомоскедастичны. Для модели это хороший знак. Другое дело, когда на графике наблюдается определённый тренд. Это может быть линейное повышение или понижение, цикличные колебания. Это знак того, что модель не оптимальна. В таком случае сам график будет для нас подсказкой. Если мы видим, что остатки выстраиваются в прямую линию – значит, предпочтительнее будет использовать регрессию на основе линейной функции. Если на графике изображены циклические колебания – больше подойдёт нелинейная регрессия.

**Постановка задачи**

Почти все необходимые определения даны, есть общее представление о машинном обучении и регрессионных моделях. Теперь поговорим о том, что конкретно нам нужно сделать. Первым делом необходимо найти реальный набор данных для исследования. Пусть он включает в себя перечень значений зависимой переменной и нескольких независимых. Эндогенной переменной будет то, что мы собираемся аппроксимировать функцией и в будущем прогнозировать. А экзогенными – те факторы, которые предположительно влияют на зависимую переменную. Причём эти наборы значений обязательно должны быть связаны построчно, чтобы в ходе операций с датасетом они не перемешались, и анализ был справедливым. Если наша задача – сравнить методы регрессии, то данные нужно брать непрерывного типа. Для бинарных, номинальных и порядковых данных есть такой метод машинного обучения как классификация, сейчас речь идёт не о нём.

Итак, требования к датасету изложены. Какие данные мы будем исследовать? Под эти требования идеально подойдут котировки акций. Они всегда выражены в числах с плавающей точкой. Кроме того, в таких датасетах можно использовать удобный связывающий параметр – дату. Ведь мы собираемся брать данные о котировках нескольких компаний, соответственно, как уже было сказано до этого, эти данные нужно построчно связать, чтобы исследование имело смысл. Возьмём их с сайта mfd.ru. Пусть это будут котировки акций Сбербанка, Газпрома, Лукойла и НорНикеля. Выбираем раздел МосБиржа Акции и ПИФы и находим тикеры Сбербанк, ГАЗПРОМ ао, ЛУКОЙЛ, ГМКНорНик. Ставим таймфрейм в один день. Данные возьмём за 10 лет: с первого января 2010 года по первое января 2020 года. В каждой таблице нас будет интересовать два столбца: дата и цена закрытия. На сайте это выглядит так:



В чем будет заключаться смысл такого исследования? Мы примем котировки акций Сбербанка за эндогенную переменную, а все остальные будут экзогенными. Это можно интерпретировать следующим образом: мы предполагаем, что котировки акций Сбербанка с 2010 по 2020 год имели статистическую зависимость от котировок акций Газпрома, Лукойла и НорНикеля. В регрессионной модели мы обучим программу аппроксимировать данные о ценах закрытия акций Сбербанка с помощью функции, значение которой будет высчитываться по значениям цен закрытия акций остальных компаний.

Сразу строить модель по набору данных было неправильным решением. В нем могут оказаться пустые значения, выбросы (значения вне разумного диапазона), ошибки в шкалах. Кроме того, мы могли ошибиться в предположении о наличии статистической зависимости между котировками. Некоторые экзогенные признаки могут оказаться несущественными, то есть практически не оказывать влияния на эндогенную переменную. Эти нюансы негативно скажутся на точности модели, поэтому нужно проверить датасет и, в случае обнаружения таких проблем, исправить их. Для этого в начале работы необходимо провести предобработку данных, чтобы датасет был чистым и готовым к исследованию. Об этом подробнее поговорим уже на этапе описания программы. Допустим, датасет прошёл предобработку, и по нему уже можно строить регрессионные модели. Какие методы мы будем применять для их построения? Рассмотрим пять методов регрессии: линейная, полиномиальная, гребневая или ридж, лассо и эластичная сеть.

Начнём с обыкновенной линейной регрессии. Что она из себя представляет? Это понятно из названия – линейную функцию. Но тут не всё так просто. Всё зависит от количества предикторов в модели. Это упоминалось в описании задачи регрессии, когда разговор шёл о графиках. Например, при наличии одного предиктора регрессия будет представлять из себя прямую линию на плоскости. При наличии двух – уже плоскость в трёхмерном пространстве. По сути, количество измерений всегда больше количества предикторов на единицу (эта единица – эндогенная переменная, она тоже является измерением). Линейная регрессия имеет аддитивную структуру, то есть представляет из себя сумму. Допустим, в модели присутствует только один существенный предиктор. Тогда регрессия будет парной, и её формула в общем виде следующая: y = a + bx. В данной формуле y – эндогенная переменная, x – экзогенная, a – свободный параметр модели, b – параметр экзогенной переменной. Если же предикторов несколько, то регрессия именуется множественной, её формула немного сложнее, но имеет тот же смысл: y = a + b1\*x1 + b2\*x2 + ... + bn\*xn. Здесь b1, b2, ..., bn – параметры экзогенных переменных, x1, x2, ..., xn – сами экзогенные переменные, n – количество предикторов. В зависимости от того, сколько у нас окажется существенных признаков, мы будем строить либо парную регрессионную модель, либо множественную.

Естественно, не во всех случаях обыкновенная линейная регрессия будет оптимальной аппроксимацией данных. Как уже было сказано ранее, при повышенной сложности модели может произойти переобучение. Это связано с тем, что её параметры (они же весовые коэффициенты) приобретают слишком маленькие или слишком большие значения. Из-за этого понижается способность регрессии к аппроксимации, так как она начинает больше отклоняться от среднего значения. Другими словами, повышается её дисперсия. Избежать переобучения можно несколькими способами. Мы можем уменьшить размерность датасета, провести отбор наиболее значимых предикторов, но гораздо более оптимальный метод – регуляризация. Раз переобучение зависит от величин весовых коэффициентов, значит, нужно повлиять на их вычисление таким образом, чтобы они не приобретали экстремальные значения. При вычислении коэффициентов программа минимизирует ошибку, поэтому ошибка – ключ к изменению коэффициентов. Регуляризация заключается в том, что к этой самой ошибке прибавляется некоторая величина – так называемый штраф. Существует два вида регуляризации: L1 и L2. Они отличаются формулой для вычисления этого самого штрафа. В регуляризации L1 это произведение суммы абсолютных значений (модулей) весовых коэффициентов на параметр альфа (его мы будем указывать при создании моделей по методам регрессии, в которых используется регуляризация). В регуляризации L2 – то же самое, только в произведение входит уже не сумма весовых коэффициентов, а сумма их квадратов. Эти квадраты провоцируют гораздо большие штрафы за экстремальные значения параметров, чем просто модули. Поэтому допустимое пространство этих параметров сужается. Функция потерь для регуляризации L1 следующая:

Изображение выглядит как текст

Автоматически созданное описание

а в L2 вместо || стоит .

После того, как мы дали определение регуляризации, становится очень просто описать три следующих метода регрессии: гребневая или ридж, лассо и эластичная сеть. Всё эти методы, по сути, являются той же самой линейной функцией, но с применением регуляризации. В гребневой регрессии применяется регуляризация L2, в лассо – L1, а эластичная сеть является комбинацией этих двух видов в указанных пропорциях (их мы тоже будем указывать при создании регрессионной модели по данному методу). Стоит отметить, что регрессия лассо может свести весовые коэффициенты к нулю, а вот ридж регрессия такого не делает: она снижает коэффициенты, но полностью аннулировать их не способна. В этом состоит основное различие этих методов.

Данные не всегда можно качественно аппроксимировать с помощью прямой линии. Они могут быть распределены по кривой, например, по параболе. Другими словами, нелинейно. В таких случаях методы регрессии на основе линейной функции становятся неэффективными, выгоднее использовать полиномиальную регрессию. В чем её смысл? В основе полиномиальной функции лежит многочлен (полином) определенной степени. Она указывается при создании регрессионной модели, что делает регрессию достаточно гибкой и хорошо контролируемой. Полиномиальная функция первой степени – это есть обыкновенная линейная функция, потому что формулы у них совпадают. Если степень повышать, то у функции появляются точки перегиба, и количество параметров регрессионной модели увеличивается. Поэтому на определённом этапе повышения степени начинается переобучение модели. Значит, есть какой-то предел её точности, который мы и будем искать. Такой метод регрессии гораздо сложнее, но он может оказаться значительно качественнее остальных рассматриваемых нами методов.

Сделаем небольшое заключение по постановке задачи. Что нам в итоге необходимо сделать? Первое: взять набор данных с указанного сайта и импортировать его в программу. Второе: провести предобработку данных и привести его в надлежащее состояние для последующего исследования. Третье: с помощью вышеперечисленных методов регрессии построить модели и сравнить их по качеству и точности с помощью оценочных параметров, указанных в описании задачи регрессии, и визуального анализа графиков остатков. И, наконец, четвертое: выявить наилучшую регрессионную модель из имеющихся для нашего набора данных, сделать вывод по проделанной работе.

**Описание предметной области**

Итак, задача поставлена. Теперь нужны инструменты для её выполнения. Данную работу мы будем выполнять на языке программирования Python. Этот язык хорошо подходит для анализа данных и машинного обучения, имеет все необходимые для этого библиотеки, которые мы рассмотрим далее. Кроме того, он имеет довольно простой и лаконичный синтаксис относительно других языков программирования, что положительно влияет на понимание кода и, как следствие, на скорость разработки. Используем среду разработки Jupyter Notebook, так как она является наиболее удобной для подобных проектов. В ней результаты кода сразу выводятся на экран, что избавит нас от необходимости писать дополнительные функции для этого. Ячеечная структура облегчает восприятие информации, разделяя её на сегменты, благодаря чему становится гораздо проще как программировать, так и демонстрировать результаты работы. При необходимости файл .ipynb можно конвертировать в формат pdf и использовать его как готовый отчёт.

В описании задачи регрессии и постановке задачи были описаны математические алгоритмы, которые будет выполнять компьютер. Но с помощью чего он будет это делать? Для анализа данных и машинного обучения на языке программирования Python существует множество инструментов. Мы будем использовать четыре основных библиотеки: numpy, pandas, sklearn и matplotlib. Теперь о каждой по порядку.

Начнём с pandas. Это высокоуровневая библиотека, которая сделана поверх другой, более низкоуровневой библиотеки под названием numpy. Поскольку последняя реализована на быстрых компилируемых языках, таких как C и Fortran, pandas обладает хорошей оптимизацией, что играет немаловажную роль в работе с большими данными. Кроме того, благодаря тому, что данный программный модуль создан на основе numpy, он предоставляет частичную возможность работать с методами и типами данных последнего. И этой возможности нам вполне достаточно, чтобы не импортировать numpy отдельно. Почему pandas является высокоуровневой библиотекой? Разберём этот вопрос детально. Numpy работает с довольно примитивными структурами данных – массивами. Использование такого программного модуля требует определённых знаний в области линейной алгебры, в частности, матричных вычислений. Numpy направлен, прежде всего, на оптимизацию. За счёт чего она достигается? Python сам по себе является медленным языком по сравнению с другими, это цена его понятности и лаконичности. Он удобен тем, что не накладывает много ограничений при работе со своими структурами данных. Типизация нестрогая, размер списков заранее задавать не обязательно, в одном списке можно хранить данные разных типов – всё это требует определённых затрат памяти и, как следствие, снижения производительности. В отличие от стандартных списков list в Python, массивы numpy имеют определённые ограничения. Их размерность задаётся заранее и является фиксированной. Все данные внутри этих массивов имеют один определённый тип. Numpy избегает питоновских циклов, используя гораздо более оптимизированные циклы компилируемых языков. Тот факт, что Python является интерпретируемым языком, делает его ещё медленнее, когда дело касается повторяющихся действий. Ведь интерпретация – это построчный перевод кода программы в машинный код, а это означает, что каждая итерация будет обходиться ценой повторяющейся обработки одних и тех же строк. С компиляцией таких проблем нет, там код переводится в машинный сразу целиком, поэтому на компилируемых языках циклы гораздо быстрее. И numpy использует это преимущество, что даёт ему хорошую оптимизацию, которая позволяет сэкономить очень много времени при работе с массивами, содержащими миллионы значений. Теперь вернёмся к pandas. Эта библиотека более дружелюбна к пользователю и практически не требует познаний в линейной алгебре. Её основное преимущество – презентабельность. Этот программный модуль предоставляет возможность работать с такими типами данных как DataFrame и Series. Они выглядят гораздо понятнее массивов numpy, особенно в сочетании с особенностями вывода среды Jupyter Notebook. DataFrame, по сути, является тем же двумерным массивом. Но, в отличие от обыкновенной матрицы, каждый столбец в этой структуре данных имеет своё название. Кроме того, DataFrame позволяет сочетать в себе столбцы с разными типами данных, что важно при работе с датасетами, в которых одновременно присутствуют, например, непрерывные и номинальные признаки. Стоит отметить одну характерную особенность, которая присуща как pandas, так и numpy, и даёт им хорошую прибавку в производительности: эти библиотеки никогда не создают копии данных в памяти, если пользователь явно этого не потребует. Для чего нам нужен pandas конкретно в этой работе? Мы собираемся импортировать наборы данных из текстовых файлов, которые получили с сайта mfd.ru, и их нужно будет преобразовать в единый DataFrame, с которым мы и будем дальше работать.

А работать мы дальше будем с библиотекой под названием sklearn (или scikit-learn). Этот программный пакет является основным для работы в области машинного обучения. Именно благодаря ему можно будет строить регрессионные модели и оценивать их качество. Sklearn написан на языках Python, C, C++ и Cython. Он содержит функции для разбивки выборки, предобработки, построения регрессионных моделей, их оценки с помощью определённых метрик, шкалирования данных. Этот программный пакет применяется в областях Data Science и Machine Learning и позволяет автоматизировать такие практические задачи как распознавание текста, изображений, речи, рукописного ввода, биржевой технический анализ, прогнозирование данных разного рода и многие другие. Что особенно удобно, он хорошо сочетается с pandas, так как допускает применение типа данных DataFrame в своих методах. Рассмотрим конкретные составные части библиотеки sklearn, которые пригодятся нам в работе, и их функционал. Начнём с модуля sklearn.linear\_model. Этот модуль является самым важным в нашей программе. Он отвечает за построение регрессионных моделей. В нём содержатся все необходимые нам методы регрессии, реализованные в классах LinearRegression, Ridge, Lasso, ElasticNet. Что касается полиномиальной регрессии, она будет задействовать класс LinearRegression и некоторые преобразования, о которых будет написано далее. Следующий программный модуль – sklearn.model\_selection. Из него нам понадобится всего одна функция под названием train\_test\_split. Эта функция предназначена для разбиения датасета на обучающую и тестовую выборки в указанных пропорциях. Можно было бы сделать это и вручную, но это было бы сложнее и дольше, пусть и не так ощутимо. Теперь о модуле sklearn.preprocessing. Здесь стоит обратить внимание на два класса: StandardScaler и PolynomialFeatures. Первый нужен для шкалирования данных. Как уже говорилось до этого, в методах регрессии ридж, лассо и эластичная сеть применяется регуляризация, смысл которой заключается в добавлении к величине ошибки штрафного значения. Так вот если стандартное отклонение, которое, кстати, зависит от среднего значения набора данных, будет слишком мало или велико, то штрафное значение тоже будет увеличенным. Это может привести к потере точности регрессионной модели. Шкалирование – способ избежать этой губительной зависимости посредством перевода данных в пространство меньшей размерности. Их абсолютные величины уменьшаются, а значит, уменьшается и модуль стандартного отклонения. При этом исходные пропорции сохраняются, соответственно, описание такого набора данных эквивалентно описанию исходного, нешкалированного набора. Класс PolynomialFeatures – это вышеупомянутое преобразование, необходимое для создания полиномиальной регрессионной модели посредством класса LinearRegression. Оно изменяет входной набор значений экзогенных переменных в соответствии с указанной степенью. Чем она выше, тем больше размерность матрицы этих значений, а следовательно, больше весовых значений содержит модель. Функция перестает быть линейной. Теперь пару слов о модуле sklearn.pipeline. Здесь нам понадобится класс Pipeline. Это ещё один инструмент для упрощения кода. Он позволяет завернуть создание полиномиальной (и не только) регрессии в одну лаконичную строку. Без этого класса код программы был бы более громоздким и трудным для восприятия. И, наконец, модуль sklearn.metrics. Здесь содержатся функции для вычисления величин, с помощью которых мы будем оценивать качество модели. Опять же, эти величины можно посчитать и без импорта данного модуля, но зачем, как говорится, изобретать велосипед, когда уже есть готовое автоматизированное решение? Нам понадобятся две функции: mean\_squared\_error для вычисления среднеквадратической ошибки и r2\_score для вычисления коэффициента детерминации.

У нас есть инструменты для обработки датасета (pandas), для построения и оценки регрессионных моделей все необходимыми методами (sklearn). Теперь нужен инструмент для визуализации результатов. Для этого и предназначен программный модуль matplotlib.pyplot. О нём достаточно будет рассказать вкратце. Этот модуль позволяет строить двумерные графики, которые можно использовать как в качестве объекта визуального анализа, так и в качестве иллюстраций к презентуемой работе. Здесь есть функции для построения разных типов графиков, их кастомизации, настройки отображения, а также группировки (можно отобразить сразу несколько графиков в виде сетки указанной размерности). Какие графики нам понадобятся? В описании задачи регрессии уже был указан так называемый Residuals Plot. Его необходимо будет строить для каждой созданной регрессионной модели, чтобы засекать изменения в распределении остатков и, соответственно, делать выводы о модели для её сравнения с другими.

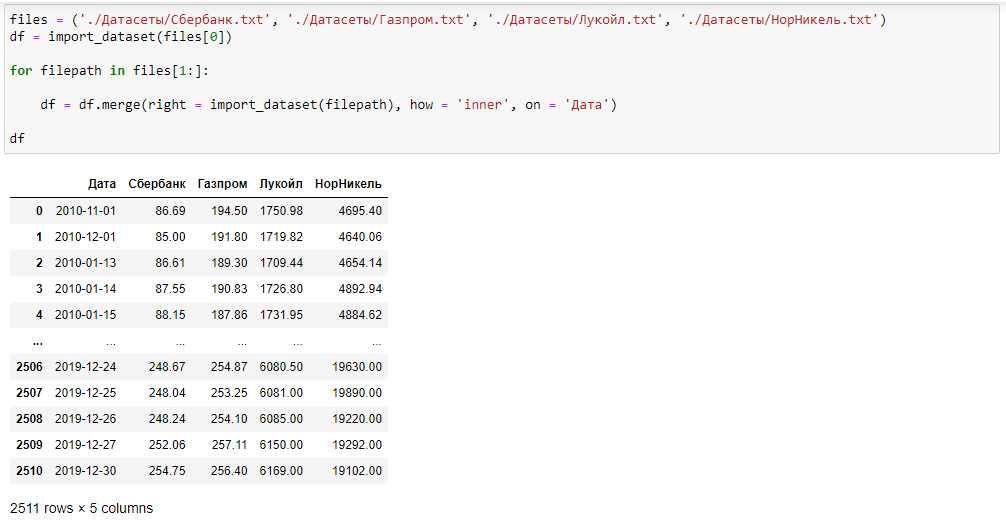
Из описания предметной области становится понятно, как будет выглядеть программа. Она будет написана на языке программирования Python в среде Jupyter Notebook. Алгоритм следующий: импортируем датасеты и объединяем их в единый pandas DataFrame, проводим предобработку, разбиваем датафрейм на обучающую и тестовую выборки, создаем несколько регрессионных моделей с помощью sklearn, после чего строим графики остатков посредством функций matplotlib и с помощью них и метрик качества модели сравниваем рассматриваемые методы регрессии.

**Описание программы**

Теперь пришло время детально рассмотреть саму программу. По классике код программы начинается с импорта всех необходимых библиотек для последующей работы. Используем инструменты pandas, sklearn и matplotlib. Первый этап, который нужно реализовать в программе – импорт и предобработка данных.

Так как у нас несколько датасетов в виде текстовых файлов, удобно написать универсальную функцию для того, чтобы их подтянуть и преобразовать в pandas DataFrame. Итак, функция import\_dataset. Каков алгоритм её работы? Она принимает на вход путь к текстовом файлу с данными и выводит готовый DataFrame. Импортирование файла происходит через функцию pandas.read\_csv. Её результат – DataFrame. В неё мы отправляем путь к датасету, а также устанавливаем несколько важных параметров. Первый – это разделитель (sep). Программа должна понять, каким образом разделить текст в файле по столбцам. Разделитель является для неё ориентиром. В нашем случае это точка с запятой. Далее идёт параметр usecols. Из всех имеющихся в датасете столбцов нам интересны только перечни дат и цен закрытия (<DATE> и <CLOSE>). Стоит отметить, что все текстовые файлы имеют одну и ту же структуру, поэтому данная функция будет универсальной. Параметр usecols представляет из себя список названий тех столбцов, которые мы хотим видеть в DataFrame. Крайний параметр – parse\_dates. Он нужен для того, чтобы столбец с датами приобрёл соответствующий тип datetime64. Без этой функции ему будет присвоен тип object. Указываем название столбца с датами в этом параметре. Итак, DataFrame сформирован. Но исходные названия столбцов слишком неудобны для работы с ними. Нужно их переименовать. Но в каждом текстовом файле необходимо использовать разные названия, чтобы в будущем различать их, ведь мы собираемся объединить все полученные посредством этой функции датафреймы в единый датафрейм. Значит, опять же, нужен универсальный алгоритм. Как правило для того, чтобы написать универсальную функцию, нужно найти что-то общее в группе объектов, с которыми эта функция работает. Наша функция работает с путями к текстовым файлам, которые представляют из себя строку. Название файла всегда идёт после крайнего символа "/", а расширение всегда одной длины (.txt). В следующей строке кода мы используем функции строкового типа данных. Rindex находит индекс последнего вхождения подстроки в строку. Что мы делаем? Мы делаем срез строки, чтобы выделить из неё название файла, которое мы будем использовать в качестве названия столбца. Для этого берём подстроку между крайним символом "/" и расширением файла. А дальше с помощью функции rename датафрейма переименовываем столбцы. Параметру inplace присваиваем значение True, чтобы изменить имеющийся DataFrame.

Дальше импорт данных становится очень простым. Создаём список с путями к нашим датасетам, сначала импортируем тот, который будет являться эндогенной переменной – Сбербанк. А потом запускаем цикл с обходом оставшейся части списка путей. И в каждой итерации соединяем созданный функцией датафрейм с тем, что получили в прошлой итерации (начинаем как раз с датафрейма с котировками акций Сбербанка). Делается это с помощью функции merge датафрейма, в которой мы указываем несколько параметров. Right – это DataFrame, который мы будем присоединять справа. How – способ соединения датафреймов. Нам нужен inner, аналогичный так называемому inner join на языке SQL. Необходимо, чтобы данные о котировках совпадали по датам, иначе анализ будет бессмысленным. Этот параметр обеспечивает такое совпадение и берёт только те строки, в которых указанный целевой параметр (дата) – один и тот же в обоих датафреймах. Параметр on – это и есть целевой столбец. Указываем название столбца с датами.



В результате цикла получаем датафрейм с пятью столбцами: один – с датами, другой – с котировками четырёх компаний. Он содержит 2511 строку. На этом импорт заканчивается и начинается предобработка.

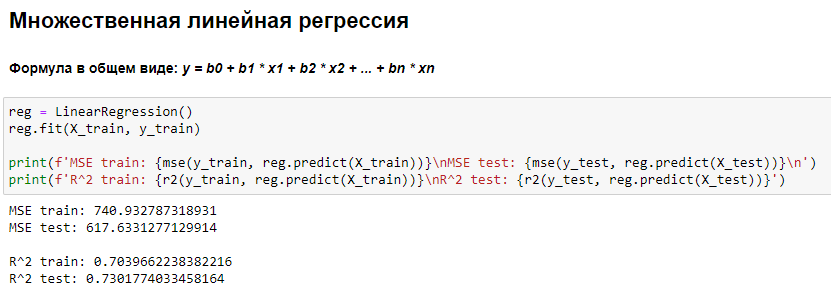
Полученный датафрейм необходимо проверить на наличие пустых записей, ошибок в шкалах (аномалий), выбросов и несущественных признаков. Воспользуемся функцией DataFrame.info() и получим информацию о типах данных в столбцах и количестве непустых значений в каждом из них. Уже отсюда можно сделать несколько выводов. Первое – все данные о котировках имеют непрерывный числовой тип float64. Второе: аномалий нет, так как все типы указаны верно. Если бы присутствовали ошибки в шкалах, вместо того же float64 отобразился бы тип object, тогда был бы смысл проверить столбцы. Третье: нет пустых значений, так как количество непустых равняется общему количеству строк. В последнем можно убедиться ещё одним способом. DataFrame.isnull().sum() выведет количество пустых записей в каждом столбце. Везде стоят нули, что и требовалось доказать.

Теперь нужно позаботиться о выбросах. Напишем для этого функцию delete\_outliers. Снова создаём стартовый DataFrame с одним столбцом с датами, к которому будем по очереди присоединять столбцы. Их будем очищать от выбросов по отдельности. По определению, выбросами считаются величины, которые либо превышают верхнее нормальное значение, либо стоят ниже нижнего нормального. Верхнее нормальное значение высчитывается как сумма третьего квартиля столбца и 1,5 межквартильного размаха (это разность третьего и первого квартиля). Нижнее нормальное значение – это разность первого квартиля столбца и 1,5 межквартильного размаха. С помощью функции quartile найдём первый и третий квартили каждого столбца, а дальше всё по вышеуказанным формулам. Обрабатываем столбцы от выбросов с помощью DataFrame.query(). Эта функция позволяет в виде строки указать условие отбора строк датафрейма, а остальные превращает в пустые значения. Итак, пользуемся уже знакомой нам функцией merge для объединения столбцов и в конце с помощью dropna() удаляем пустые значения. Функция для удаления выбросов готова. Вызываем её, и теперь в нашем датафрейме 2162 строки.

Чтобы определить несущественные признаки, нужно оценить их статистическую связь с зависимой переменной. Эту связь отображает коэффициент корреляции. Построим корреляционную матрицу по столбцам с данными котировок. Если связь имеется – столбец отобразится в матрице. Вызываем для этого функцию corr датафрейма. Можно было построить так называемую тепловую карту с помощью библиотеки seaborn, но сути это бы не поменяло. Мы видим, что все столбцы присутствуют в матрице, а значит, имеют статистическую связь с эндогенной переменной и являются существенными. Соответственно, удалять какие-либо предикторы нецелесообразно. Предобработка завершена, датафрейм имеет 2162 строки и 5 столбцов, считая столбец с датами.

Теперь начинается этап построения регрессионных моделей. Во-первых, разобьём датафрейм на вектор значений эндогенной переменной (y) и матрицу значений экзогенных (X). Делаем это обычными срезами и берём атрибуты values в обеих переменных (это обыкновенные массивы numpy). Далее применяем импортированную функцию train\_test\_split, возвращающую кортеж с четырьмя объектами: обучающие выборки из вектора y и матрицы X и тестовые выборки из них же. В параметрах указываем долю тестовой выборки в общем количестве наблюдений (оптимальная доля – 30%). Стоит обратить внимание на параметр random\_state. Его мы приравниваем к нулю, чтобы разбиение на выборки не было случайным и не изменялось при повторном запуске кода.

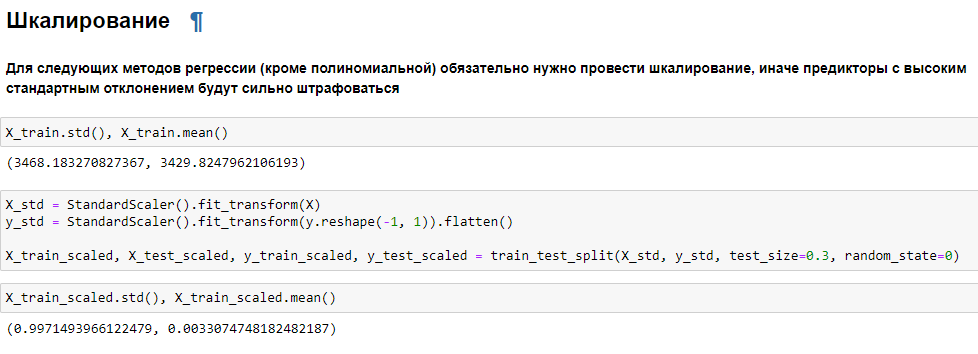
Построим модель множественной линейной регрессии. Создаём объект класса LinearRegression и применяем метод fit, в который передаём обучающие выборки X\_train и y\_train. Модель создана. Теперь необходимо оценить её. Пользуемся импортированными функциями для вычисления метрик r2\_score и mean\_squared\_error. В этих функциях необходимо использовать вектор спрогнозированных значений, который мы получим из функции модели predict. В данную функцию передаётся матрица значений экзогенных переменных. В зависимости от того, для какой выборки метрики нам нужны, мы будем передавать туда X\_train или X\_test. Для полноценного анализа отобразим метрики и для обучающей, и для тестовой выборки.



Какие выводы можно сделать? Коэффициент детерминации говорит о приемлемом качестве модели. О среднеквадратической ошибке пока сказать нечего, так как мы ещё не начали сравнивать разные методы регрессии.

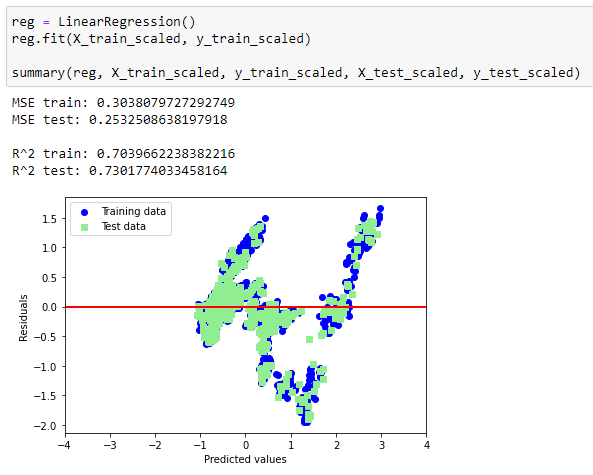
Далее нам понадобится универсальная функция для вывода характеристик модели. Создаём функцию summary, принимающую на вход саму модель, X\_train, y\_train, X\_test и y\_test. Вывод коэффициента детерминации и MSE делается так же, как мы сделали это с первой моделью. Помимо этого, нам ещё нужен график остатков. Вернее, два графика: по обучающей и тестовой выборке. Можно изобразить на одной плоскости две диаграммы рассеяния разных цветов. Функция plt.scatter() нужна для построения такой диаграммы. В неё мы передаём вектор реальных значений эндогенной переменной (он выступает в качестве абсциссы) и вектор остатков (определение остатков уже было дано), а также параметры для оформления. Далее идут функции для кастомизации графиков (сетка, расположение легенды, пометки на осях, красная линия на нулевой горизонтальной оси). Стоит обратить внимание на функцию plt.xlim(). Далее мы будем работать только со шкалированными данными, нам хватит диапазона отображения от -4 до 4 по абсциссе. Далее – отображение графика функцией plt.tight\_layout(). Универсальная функция summary готова.

Теперь приступим к шкалированию. Для этого создаём объект класса StandardScaler и используем метод fit\_transform, в который передаём матрицу значений X (исходную, не по выборкам). То же самое повторяем для вектора y. Шкалированные данные разбиваем на выборки с помощью train\_test\_split по старой схеме. Сравниваем среднее значение и стандартное отклонение исходных и шкалированных данных.

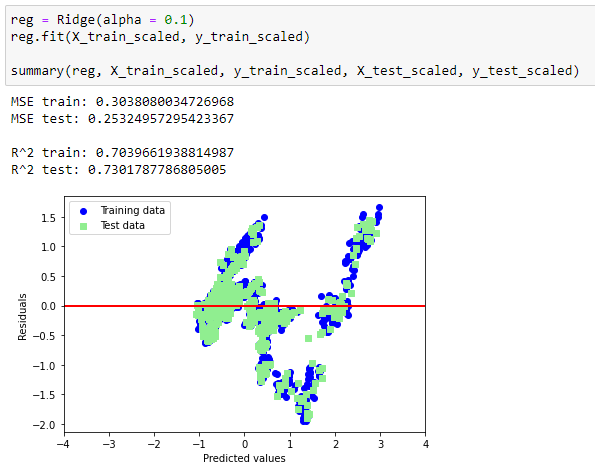


У последних они значительно уменьшились, значит, шкалирование прошло успешно.

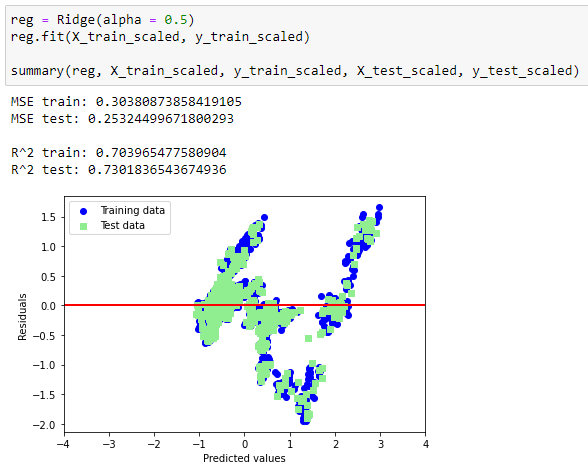
Далее построение моделей становится очень простым. Ещё раз создадим модель множественной линейной регрессии, но уже на шкалированных данных (это необходимо для адекватного сравнения среднеквадратических ошибок). Воспользуемся функцией summary.



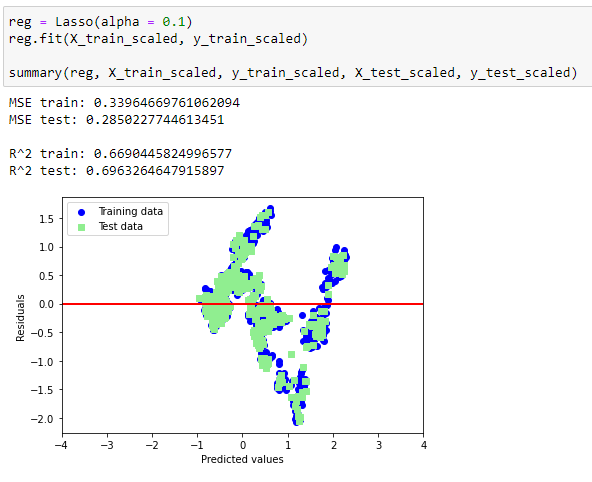
Как и ожидалось, коэффициенты детерминации не изменились, а MSE значительно уменьшились. Но распределение остатков напоминает нелинейный график, значит, такая регрессия не оптимальна. Теперь построим две модели – ридж и лассо с параметром alpha = 0,1.



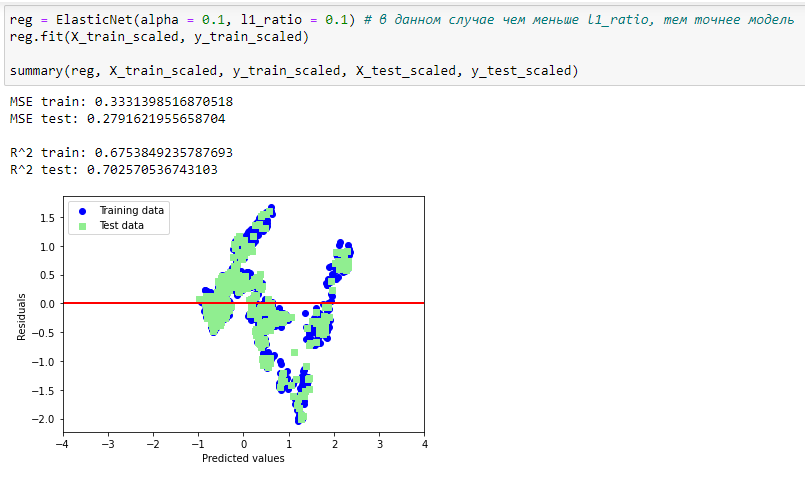
То же самое, но с alpha = 0,5:



Гребневая регрессия по характеристикам оказалась незначительно лучше множественной линейной. Посмотрим на регрессию лассо:

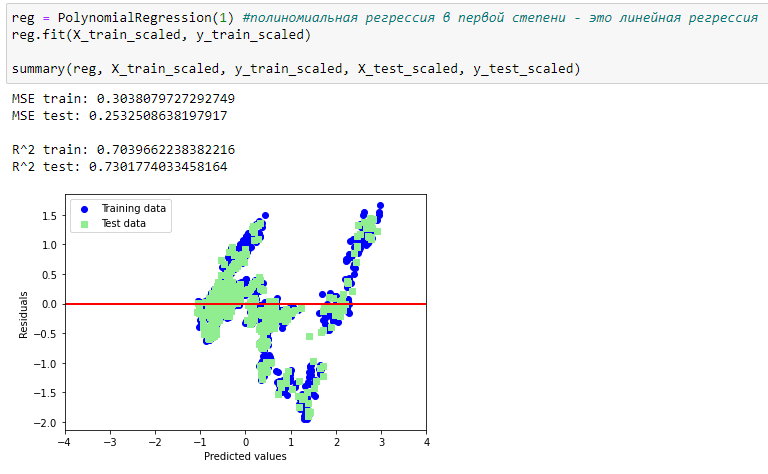


Она оказалась хуже обеих этих моделей. Теперь эластичная сеть. Так как регрессия лассо показала себя хуже, чем гребневая, мы установим низкую долю регуляризации L1. Укажем l1\_ratio = 0,1.

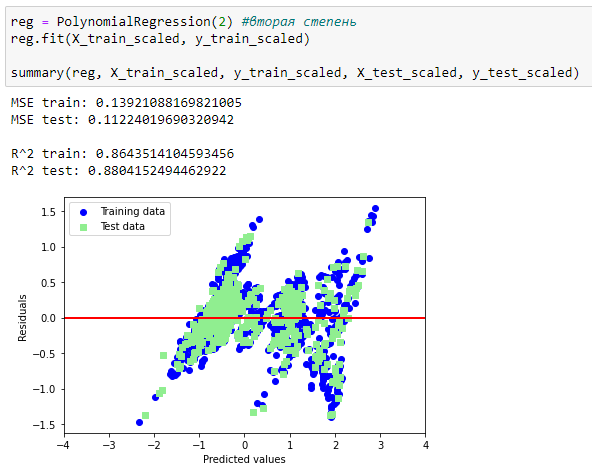


Что в итоге? По характеристикам эта модель лучше, чем лассо, но хуже двух остальных.

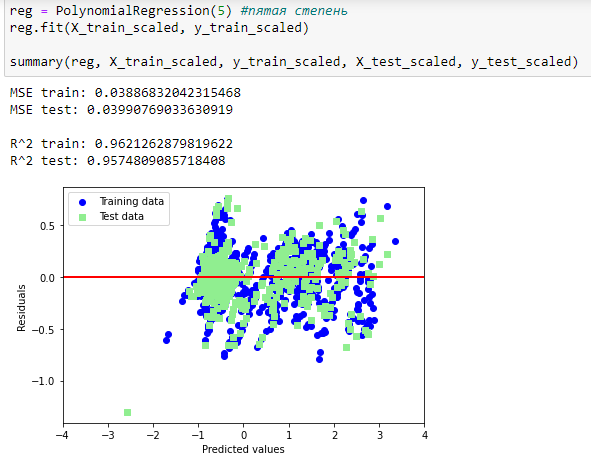
Теперь о полиномиальной регрессии. Её построение требует более сложных действий. Чтобы не загромождать программу повторяющимся кодом, напишем функцию PolynomialRegression. Она будет принимать на вход степень полинома. Вывод функции – объект класса Pipeline, в атрибуты которого передаём два кортежа: строка «poly» в паре с объектом класса PolynomialFeatures с переданной в него степенью и строка «lin\_reg» в паре с объектом класса LinearRegression. Теперь с помощью этой функции можно создать полиномиальную модель нужной степени точно так же, как мы это делали с остальными моделями. Будем строить их с постепенным повышением степени и отслеживать изменения характеристик.

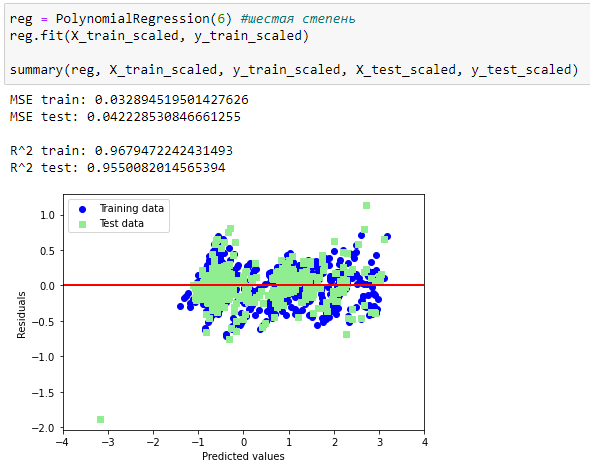


Полиномиальная модель первой степени эквивалентна множественной линейной, характеристики те же.



Вторая степень значительно улучшила точность, MSE по обеим выборкам начала уменьшаться.





С увеличением степени качество модели растёт, но на шестой степени началось переобучение – увеличилась MSE тестовой выборки, упал коэффициент детерминации. Делаем логичный вывод: пятая степень выжимает максимум из полиномиальной модели. По характеристикам она оказалась лучше всех предыдущих методов регрессии. График остатков стал больше напоминать случайное распределение.

**Заключение**

Итак, в данной работе мы рассмотрели пять разных методов регрессии и сравнили модели на их основе. Полиномиальная модель пятой степени оказалась самой точной из них. О чём это говорит? Как минимум, о том, что данные распределены нелинейно. Какие выводы можно сделать по данной работе? В процессе регрессионного анализа нужно учитывать множество факторов, которые могут повлиять на качество модели. Можно ли такую модель использовать для реального прогнозирования? Если рассудить логически, можно, но есть свои нюансы. Регрессия – это, по определению, тренд. Она строится на основе имеющихся данных и описывает их. Но вполне возможны резкие изменения, ведь не всегда всё идёт стабильно. И эти изменения регрессия не в состоянии предугадать. Например, тот же биржевой рынок непостоянен, котировки могут как внезапно вырасти в цене, так и упасть. Поэтому в процессе прогнозирования нужно не только руководствоваться аналитическими выводами, но и обращать внимание на различные предпосылки, которые вносят свои коррективы и могут радикально изменить ситуацию.

**Список материалов**

1. https://russianblogs.com/article/7167804943
2. <https://nagornyy.me/it/regressionnye-modeli-v-python>