Using the machine learning techniques for the creation of trading robot
Brosalin D. (Russian Federation)

Использование методов машинного обучения для создания биржевого робота
Бросалин Д. С. (Российская Федерация)

Бросалин Дмитрий Сергеевич / Brosalin Dmitriy Sergeevich – студент,
факультет кибернетики и информационной безопасности,
Национальный исследовательский ядерный университет,
Московский инженерно-физический институт, г. Москва

Аннотация*: в данной исследовательской работе рассматриваются математические и статистические методы для задач прогнозирования и ведения электронных торгов на бирже. Анализируются существующие модели машинного обучения для задач прогнозирования. Оценивается их точность на реальных примерах задач. Приводятся как математические выкладки, так и примеры программного кода.*
Abstract*: in that research work there are reviews on mathematical and statistical methods for predictions and online trading. Existing machine learning models for forecasting were analyzed and their accuracy was measured. There are both mathematical calculations and examples of programming code in that work.*

Ключевые слова*: машинное обучение, математика, линейная алгебра, статистика, временные ряды, биржевой робот, математическая модель, ARMA, ARIMA, AR, MA, метод наименьших квадратов, МНК, python, scikitlearn, pandas, numpy.*
Keywords*: machine learning, mathematics, linear algebra, statistics, time series, trading robot, mathematical models, ARMA, ARIMA, AR, MA, mean squared error, MSQ, python, scikitlearn, pandas, numpy.*

В настоящее время, наука об анализе данных с целью выявления скрытых зависимостей для их дальнейшего прогнозирования, считается неотъемлемой частью современного мира. Её название – машинное обучение. Сфера влияния машинного обучения распространилась практически на все современные виды деятельности. В медицинской отрасли машинное обучение применяется для прогнозирования состояния здоровья человека на ближайшие годы. Предположим, что у нас есть данные анализов объекта (человека), у которого есть признаки (содержание химических элементов в крови), и есть область знаний (исторические данные о заболеваниях). Сопоставив исходные данные о человеке с областью знаний, машинное обучение позволит спрогнозировать состояние здоровья человека на несколько лет вперед в зависимости от поставленной задачи.

Мы привыкли пользоваться поисковиками, такими как Google или Yandex. Эти компании используют анализ данных, введенных пользователями для обучения поисковой машины. Так как просто невозможно учесть все критерии запросов пользователей и релевантные ответы к ним.

Машинное обучение широко применяется в финансовом секторе. Например, банки разрабатывают рекомендательные системы. Такие системы, как скоринг, использующийся для принятия решения о выдаче человеку кредита, в зависимости от информации о человеке и исторических данных о людях, которые успешно выплатили кредит, и о тех, которые успешно не справились с такой задачей. При электронных торгах на бирже перед игроком стоит задача прогнозирования движения курса акций. И здесь машинное обучение приходит на помощь. По имеющимся историческим данным о курсе акций можно составить временной ряд, а имея информацию о пиковых значениях цен, прибегнуть к регрессионному анализу.

Виды моделей машинного обучения

Для решения поставленных задач в машинном обучении используются модели. Модель принято обучать данными, полученными из достоверных источников. Смоделировав исторические данные, становится возможным использовать модель для предсказания будущих величин или распознавания образов. Разнообразие моделей очень велико, что оправданно. Задачи, которое способно решать машинное обучение, не поддается исчислению. В настоящее время наиболее распространенными являются задачи: классификации, регрессии, кластеризации и выявления ассоциаций.

Предлагаю рассмотреть наиболее часто применяемые на практике модели: обучение с учителем и обучение без учителя.

Обучение с учителем

Подход к машинному обучению можно логично разделить на два типа: обучение с учителем, то есть на размеченных данных, и обучение без учителя. В случае обучения с учителем у нас есть данные, которые разделены на области признаков объектов и ответов. X = $(\mathbf{Xi}, \mathbf{Yi})^{l}$, i=1..l. Постановка задачи обучения с учителем заключается в том, чтобы найти такой алгоритм $Q(\alpha, X) \rightarrow min_{\alpha}$, на котором будет достигаться минимум функционала ошибки. Тип задачи обучения с учителем заключается в ответах, которые необходимо получить. В простейшем случае пространством ответов является множество 2 элементов, например {0, 1} или {-1, 1}. В этом случае множество объектов, которые имеют один ответ, называются классом, и перед нами задача классификации.

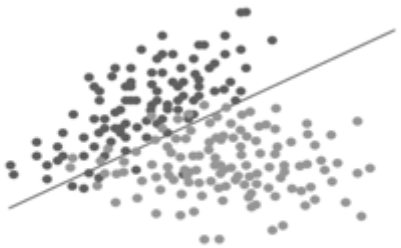

Рис. 1. Бинарная классификация

В общем случае задача классификации заключается в том, чтобы провести прямую через множества объектов таким образом, что разделяющая прямая отсекала один класс от другого. Задачи классификации бываю многоклассовыми. В этом случае множество ответов не ограничивается двумя числами, Y = {0, 1, …, K}.

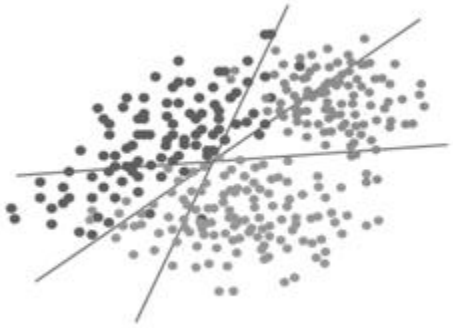

Рис. 2. Многоклассовая классификация

Случается так, что множеством ответов является множество вещественных чисел. В этом случае мы имеем дело с задачей восстановления регрессии. Например, задача предсказания роста по весу или прогнозирование выручки компании в зависимости от инвестиций в рекламу по телевидению, радио, газетам.

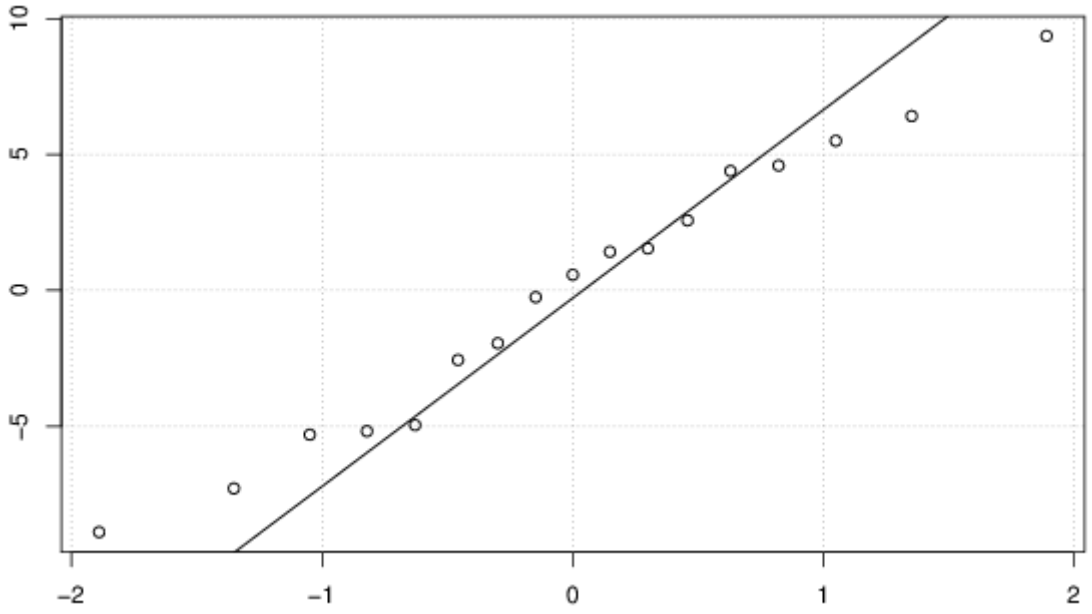

Рис. 3. Восстановление регрессии

Обучение без учителя

Задача обучения без учителя это такая задача, в которой нет множества признаков ответов, есть только признаки объектов и с ними нужно сделать какие-то действия. Например, задача кластеризации может быть рассмотрена как задача обучения без учителя. В этом случае мы располагаем некоторым набором объектов и нужно сгруппировать их, найти группы похожих объектов. Решая задачу кластеризации, мы сталкиваемся с несколькими проблемами. Мы не имеем представления, сколько кластеров имеется в наших данных, также мы не знаем истинных ответов, кластеров, которые нужно выделять.

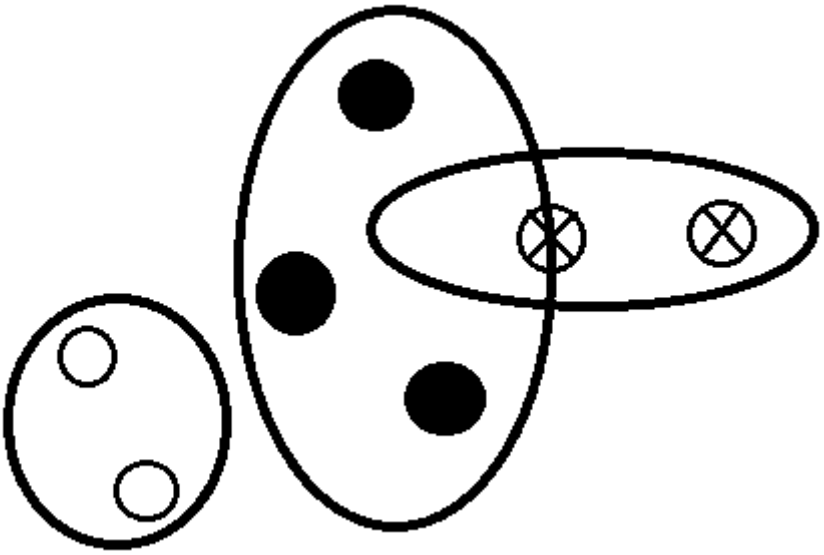

Рис. 4. Кластеризация

Примеры задач кластеризации: сегментация пользователей, поиск схожих пользователей в социальных сетях, поиск генов с похожими профилями экспрессии.

Вторым примером задачи обучения без учителя является задача визуализации. В этом типе задач необходимо нарисовать многомерную выборку, то есть отразить многомерную точку в двумерном пространстве. Необходимым условием такого отражения является тот факт, что визуализированные данные должны отображать структуру, соответствующую исходной многомерной выборке, чтобы, глядя на изображение, можно было понять, как устроены эти данные, и что с ними можно делать. На рисунке изображены сканы цифр, написанных от руки. Из рисунка видно, что каждое число образовывает группу схожих с ней изображений.

Третий пример задачи обучения без учителя – поиск аномалий. В данном типе задач требуется обнаруживать, что данный объект не похож на все остальные, другими словами - является аномальным. При обучении у нас есть примеры обычных объектов, не аномальных, а аномальных объектов либо нет вообще, либо настолько мало, что не представляется возможным воспользоваться классическими приемами обучения с учителем, чтобы классифицировать аномалии.

Примеры задач поиска аномалии: определение поломки в системах самолета, определении поломки высоконагруженной системы, выявление проблем в модели машинного обучения.

Прикладное применение машинного обучения

В текущей работе будет предложено рассмотрение трёх алгоритмов машинного обучения. В первой части главы будет показано прогнозирование сорта вина в зависимости от его характеристик на примере применения алгоритма К ближайших соседей. Будет проведен анализ данного алгоритма с целью выявления оптимального количества ближайших соседей для достижения большей точности прогноза. Во второй части будет затронута тема восстановления регрессии. Будет показано, как можно спрогнозировать выручку компании в зависимости от её вложений в рекламу по телевидению, радио и газетам. Также введем понятие стохастического градиентного спуска и попробуем применить его для задачи прогноза дохода компании. В заключение сравним результаты работы обучения с помощью нормального уравнения и стохастического градиентного спуска. Инструментарий, который будет использован: Python, Ipython notebook, scipy, sklearn, numpy, pandas.

Метод к ближайших соседей

Метод ближайших соседей, пожалуй, является самым простым алгоритмом в машинном обучении. Алгоритм относит поступивший объект u к тому классу, элементов которого окажется больше среди К ближайших соседей $x_u^{(i)}, i \in (1 \dots k)$: a(u, X^l, k) = $\sum_{i=1}^{k} [y_u^{(i)} = y]$.

Алгоритм К ближайших соседей чувствителен к масштабу признаков. Так, если масштаб одного из признаков существенно превосходит масштабы остальных признаков, то их значения практически не будут влиять на ответы алгоритма. Поэтому очень важно производить масштабирование признаков. Обычно это делается путем вычитания среднего значения признака и деления результата на стандартное отклонение текущего столбца. Для обучения модели будем пользоваться кросс-валидацией, суть проста. На I-м шаге I-блок выступает в качестве тестовой выборки, объединение всех остальных блоков – в качестве обучающей выборки. Таким образом, на каждом шаге алгоритм обучается на некоторой обучающей выборке, после чего вычисляется его качество на тестовой выборке.

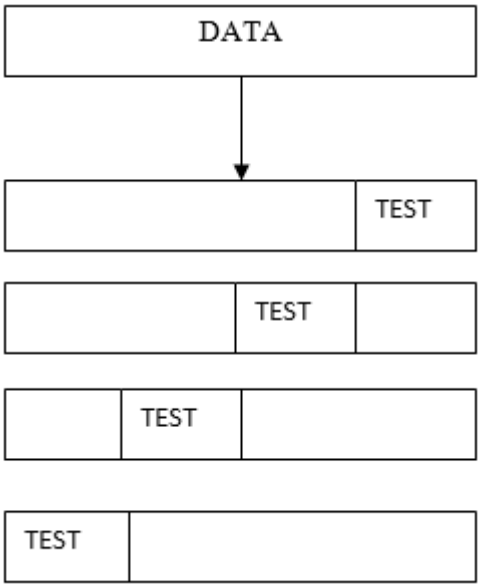

Рис. 5. Кросс-валидация

После выполнение m шагов мы получаем m показателей качества, усреднение которых и дает оценку кросс-валидации. Для обучения возьмем данные, описывающие вина [1]. Набор данных представляет информацию о винах, их свойствах, таких как: крепость алкоголя, яблочная кислота, магний, количество фенолов, влаваноиды и т. д. Более подробное описание признаков можно узнать, ознакомившись с интернет ресурсом. При обучении модели нам нужно узнать оптимальное количество соседей, чтобы получить лучшее качество работы алгоритма. Загрузим датасет, посмотрим, что он из себя представляет.

```
import pandas as pd

import sklearn as sk

data = pd.read_csv("wine.data", header=None)

data.head(10)
```

Рис. 6. Считывание данных

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045

Рис. 7. Данные о винах

Теперь выделим из данных целевой вектор, то есть вектор ответов и матрицу признаков.

```
X = data.ix[:,1:len(data.columns)]

Y = data.ix[:,0]
```

Рис. 8. Выделение целевого вектора и матрицы признков

Посмотрим матрицу признаков и вектор ответов для первых 5 объектов.

```
      1     2     3     4    5     6     7     8     9     10    11    12
0  14.23  1.71  2.43  15.6  127  2.80  3.06  0.28  2.29  5.64  1.04  3.92
1  13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28  4.38  1.05  3.40
2  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81  5.68  1.03  3.17
3  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18  7.80  0.86  3.45
4  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82  4.32  1.04  2.93

     13
0  1065
1  1050
2  1185
3  1480
4   735
0     1
1     1
2     1
3     1
4     1
```

Рис. 9. Признаки вин

Посмотрим при каком параметре K – количество ближайших соседей, достигается минимум функционала ошибки.

```
lst_of_answers  = []

lst = []

kf = KFold(len(X),  shuffle=True,  n_folds=5,  random_state=42)

for k in range(1,  51):

        knn = KNeighborsClassifier(n_neighbors=k)

        scores = cross_val_score(knn,  X,  Y, cv=kf,  scoring='accuracy')

        lst.append(scores.mean())

        lst_of_answers.append(str(k))

cnt = 0

for i in lst:

        if i == max(lst):

                print("Optimal  K: " + lst_of_answers[cnt]  + ", with  accuracy:  " +
str(lst[cnt]))

        cnt += 1


Optimal  K: 1, with  accuracy:  0.730476190476
```

Рис. 10. Определение точности алгоритма с параметром K = 1

Как видно из ответа, алгоритм работает плохо, один ближайший сосед - очень плохая практика для решения задач классификации. Попробуем поднять качество модели, для этого применим масштабирование признаков, то есть из каждого признака вычтем среднее значение, затем поделим результат на стандартное отклонение для текущего столбца:

```
lst_of_answers  = []

lst = []

X_scaled = scale(X)

for k in range(1, 51):

        knn = KNeighborsClassifier(n_neighbors=k)

        scores = cross_val_score(knn, X_scaled, Y, cv=kf, scoring='accuracy')

        lst.append(scores.mean())

        lst_of_answers.append(str(k))

cnt = 0

for i in lst:

        if i == max(lst):

                print("Optimal K: " + lst_of_answers[cnt] + ", with accuracy: " +
str(lst[cnt]))

        cnt += 1


Optimal K: 29, with accuracy: 0.977619047619
```

Рис. 11. Нахождение оптимального K

Таким образом, мы пришли к выводу, что при использовании методов обучения сначала необходимо отмасштабировать данные, по масштабированным данным произвести кросс-валидацию, найти необходимые параметры модели и только потом ее применять на практике.

Восстановление регрессии. нормальное уравнение

Линейная регрессия – один из наиболее хорошо изученных методов машинного обучения, позволяющий прогнозировать значения количественного признака в виде линейной комбинации прочих признаков с параметрами – весами модели. Оптимальные (в смысле минимальности некоторого функционала ошибки) параметры линейной регрессии можно найти аналитически с помощью нормального уравнения или численно с помощью методов оптимизации.

Модель регрессии – линейная: $a(x,w) = (x,w) = \Sigma\downarrow(i=1)\uparrow n \equiv f\downarrow i\ (x)\, w\downarrow i, w(R$. Линейная регрессия использует простой функционал качества – среднеквадратичную ошибку. Мы будем работать с выборкой, содержащей 3 признака. Для настройки параметров (весов) модели решается следующая задача:

$$\frac{1}{l}\sum_{i=1}^{l}(y_i - (w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{3i}))^2 \rightarrow min_{w_0,w_1,w_2,w_3}$$

где *xi*1, *xi*2, *xi*3 – значения признаков *i*-го объекта, *yi* – значение целевого признака *i*-го объекта, ℓ - число объектов в обучающей выборке.

Нахождение вектора оптимальных весов *w* может быть сделано и аналитически. Мы хотим найти такой вектор весов *w*, чтобы вектор *y*, приближающий целевой признак, получался умножением матрицы *X* (состоящей из всех признаков объектов обучающей выборки, кроме целевого) на вектор весов *w*. То есть, чтобы выполнялось матричное уравнение:

$$y = Xw$$

Домножим данное уравнение слева и справа на X^T и выразим вектор весов:

$$w = (X^T X)^{-1} X^T y$$

Матрица: $(X^T X)^{-1} X^T$ – называется псевдообратной для матрицы X. Нахождение псевдообратной матрицы – операция вычислительно сложная и нестабильная, так как мы можем попасть в ситуацию, когда определитель матрицы равен или очень близок к нулю, то есть проблема мультиколлинеарности, когда строка или столбец матрицы выражается через другие строки или столбцы матрицы.

Попытаемся спрогнозировать выручку компании в зависимости от уровня ее инвестиций в рекламу по телевидению, в газетах и по радио [2].

Загрузим датасет:

```
import pandas as pd

adver_data = pd.read_csv('advertising.scv')
```

Рис. 12. Загрузка датасета

Посмотрим, как выглядят данные, также посмотрим их статистики:

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

Рис. 13. Первые пять значений датасета

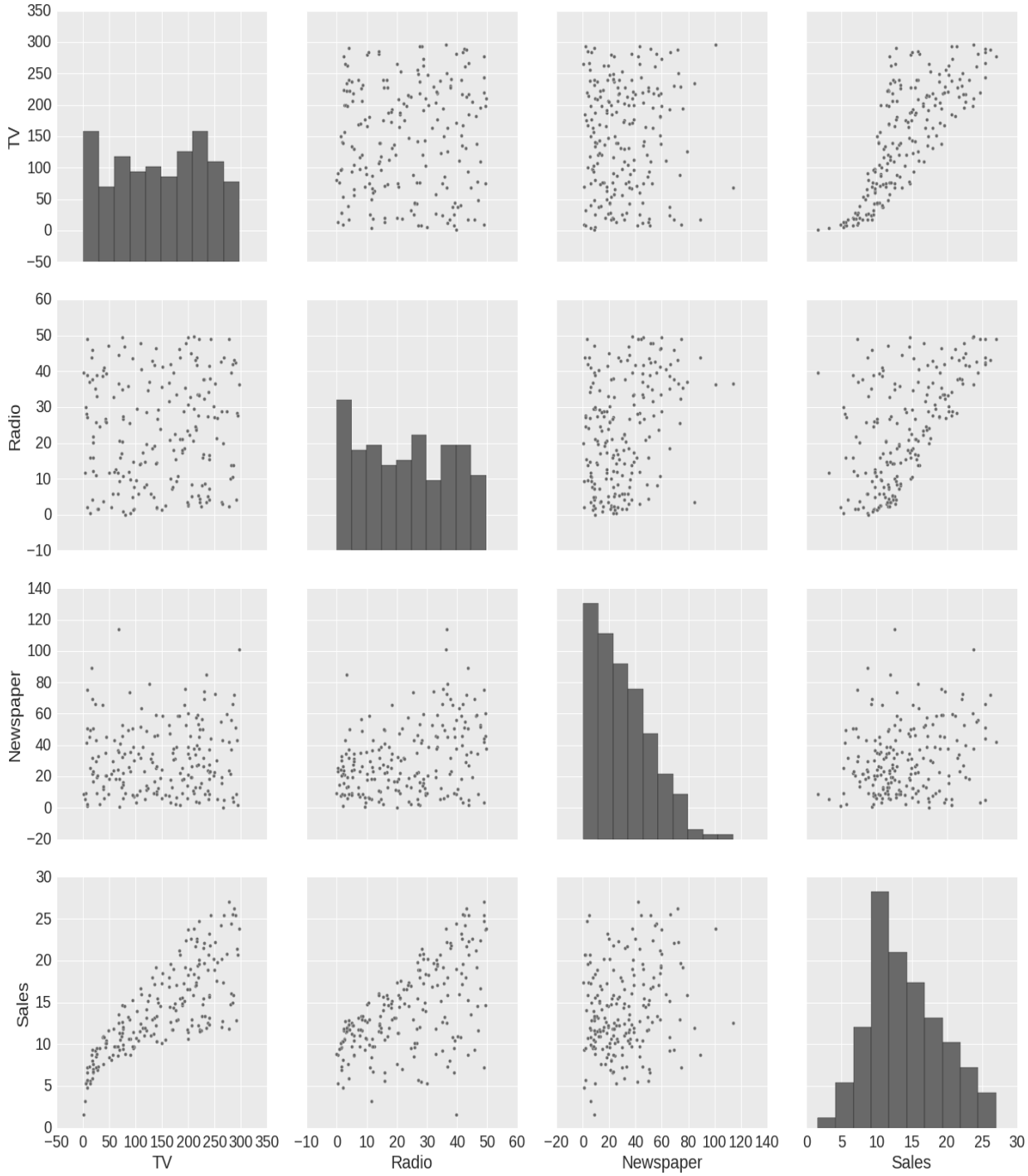

Рис. 14. Попарные зависимости данных

Отмасштабируем матрицу признаков объектов, вычтя из признака значение среднего, и результат поделим на стандартное отклонение:

```
i, j = 0, 0

for i in range(0, X.shape[0]):

    for j in range(0, X.shape[1])

        X[i][j] = (X[i][j] - means[j]) / stds[j]
```

Рис. 15. Масштабирование матрицы признаков

Добавим к матрице X столбец из единиц, используя методы hstack, ones и reshape библиотеки NumPy. Вектор из единиц нужен для того, чтобы не обрабатывать отдельно коэффициент *w*0 линейной регрессии.

```
import numpy as np

a = np.ones(X.shape[0])

a = a.reshape(a.size, 1)

X = np.hstack((X, a))
```

Рис. 16. Добавление вектора из единиц

Далее реализуем функцию – среднеквадратичную ошибку прогноза, чтобы оценивать качество модели, а также функцию прогноза для линейной модели, которая принимает на вход матрицу X и вектор весов линейной модели *w*:

```
def mserror(y, y_pred):
    return np.mean((y - y_pred)**2)


def linear_prediction(X, w):
    return np.dot(X, w)
```

Рис. 17. Объявление функций среднеквадратической ошибки и прогнозирования линейной модели

Основной функцией для прогноза будет являться функция, которая решает нормальное уравнение:

```
def normal_equation(X, y):
    part_pinv = np.linalg.pinv(X)
    res = np.dot(part_pinv, y)
    return res
```

Рис. 18. Функция для решения нормального уравнения

```
norm_eq_weights = normal_equation(X, y)

print(norm_eq_weights)

[ 3.91925365  2.79206274  -0.02253861  14.0225 ]
```

Рис. 19. Значение вектора весов

Теперь, обучив модель признаками и создав целевой вектор ответов, найдем качество модели на тестовой выборке, затем сравним полученные результат с реальными значениями, вычислив функционал среднеквадратической ошибки:

```
predictions = linear_prediction(X, norm_eq_weights)

err = mserror(y, predictions)

print(err)

2.78412631451
```

Рис. 20. Среднеквадратичная ошибка для линейной модели

Восстановление регрессии. стохастический градиентный спуск

Параметры *w*0,*w*1,*w*2,*w*3, по которым минимизируется среднеквадратичная ошибка, можно находить численно с помощью градиентного спуска. Градиентный шаг для весов будет выглядеть следующим образом:

$$\frac{\square}{l}\sum_{i=1}^{i=l}\left(y_i - \left(w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{3i}\right)\right)$$

$$\frac{\square}{l}\sum_{i=1}^{i=l} x_{ij}\left(y_i - \left(w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{3i}\right)\right), j\ (\{1,2,3\}$$

Здесь η – параметр, шаг градиентного спуска. Проблема градиентного спуска, описанного выше, в том, что на больших выборках считать на каждом шаге градиент по всем имеющимся данным может быть очень вычислительно сложно.

В стохастическом варианте градиентного спуска поправки для весов вычисляются только с учетом одного случайно взятого объекта обучающей выборки:

$$w_{\downarrow}0 \leftarrow w_{\downarrow}0 + \ 2(/l\ (y_{\downarrow}i - (w_{\downarrow}0 + \ w_{\downarrow}1\ x_{\downarrow}k1 + w_{\downarrow}2\ x_{\downarrow}k2 + w_{\downarrow}3\ x_{\downarrow}k3\ ))$$

$$w_{\downarrow}j \leftarrow w_{\downarrow}j + \ 2(/l\ x_{\downarrow}kj\ (y_{\downarrow}i - (w_{\downarrow}0 + \ w_{\downarrow}1\ x_{\downarrow}k1 + w_{\downarrow}2\ x_{\downarrow}k2 + w_{\downarrow}3\ x_{\downarrow}k3\ ))\ ,$$

j∈{1, 2, 3},

где k – случайный индекс, $k \in \{1,\ldots,\ell\}$.

Напишем функцию stochastic_gradient_step, реализующую шаг стохастического градиентного спуска для линейной регрессии. Функция должна принимать матрицу X, векторы y и w, число train_ind – индекс объекта обучающей выборки (строки матрицы X), по которому считается изменение весов, а также число ***η*** (eta) – шаг градиентного спуска (по умолчанию ***eta*** = 0.01). Результатом будет вектор обновленных весов. Наша реализация функции будет явно написана для данных с 3 признаками:

```
def stochastic_gradient_step(X,  y, w, train_ind,  eta=0.01):
        grad0 = X[train_ind][0]*(y[train_ind]  - (w[3] + w[0]*X[train_ind][0]  +
                w[1]*X[train_ind][1]  +
                w[2]*X[train_ind][2])
                )
        grad1 = X[train_ind][1]*(y[train_ind]  - (w[3] + w[0]*X[train_ind][0]  +
                w[1]*X[train_ind][1]  +
                w[2]*X[train_ind][2])
                )
        grad2 = X[train_ind][2]*(y[train_ind]  - (w[3] + w[0]*X[train_ind][0]  +
                w[1]*X[train_ind][1]  +
                w[2]*X[train_ind][2])
                )
        grad3 = (y[train_ind]  - (w[3] + w[0]*X[train_ind][0]  +
                w[1]*X[train_ind][1]  +
                w[2]*X[train_ind][2])
                )
        return  w + (2 * eta / X.shape[0])  * np.array([grad0,  grad1, grad2, grad3])
```

Рис. 21. Функция градиентного шага

Создадим функцию stochastic_gradient_descent, реализующую стохастический градиентный спуск для линейной регрессии. Функция принимает на вход следующие аргументы:

- X – матрица, соответствующая обучающей выборке;
- y – вектор значений целевого признака;
- w_init – вектор начальных весов модели;
- eta – шаг градиентного спуска (по умолчанию 0.01);
- max_iter – максимальное число итераций градиентного спуска (по умолчанию 10000);
- max_weight_dist – минимальное евклидово расстояние между векторами весов на соседних итерациях градиентного спуска, при котором алгоритм прекращает работу (по умолчанию 1e-8);
- seed – число, используемое для воспроизводимости сгенерированных псевдослучайных чисел (по умолчанию 42);
- verbose – флаг печати информации (например, для отладки, по умолчанию False).

На каждой итерации в вектор (список) должно записываться текущее значение среднеквадратичной ошибки. Функция должна возвращать вектор весов w а также вектор (список) ошибок.

```
def stochastic_gradient_descent(X, y, w_init, eta=1e-2, max_iter=1e4,
        min_weight_dist=1e-8, seed=42, verbose=False):
        weight_dist = np.inf
        w = w_init
        errors = []
        iter_num = 0
        np.random.seed(seed)
        w_lst = []
        while weight_dist > min_weight_dist and iter_num < max_iter:
                random_ind = np.random.randint(X.shape[0])
                w_cur = stochastic_gradient_step(X, y, w, random_ind, eta=1e-2)
                weight_dist = np.linalg.norm(w_cur - w, ord = 2)
                w = w_cur
                err = mserror(linear_prediction(X, w), y)
                errors.append(err)
                iter_num += 1
        return w, errors
```

Рис. 22. Реализация градиентного спуска

Запустим 10^5 итераций стохастического градиентного спуска. Укажем вектор начальных весов w_init, состоящий из нулей. Зададим параметры eta и seed равными их значениям по умолчанию (eta = 0.01, seed = 42). Проделаем это для того, чтобы понять, как меняется функционал ошибки в зависимости от итерации, в идеальном случае, если все реализовано корректно, должны увидеть сходимость.

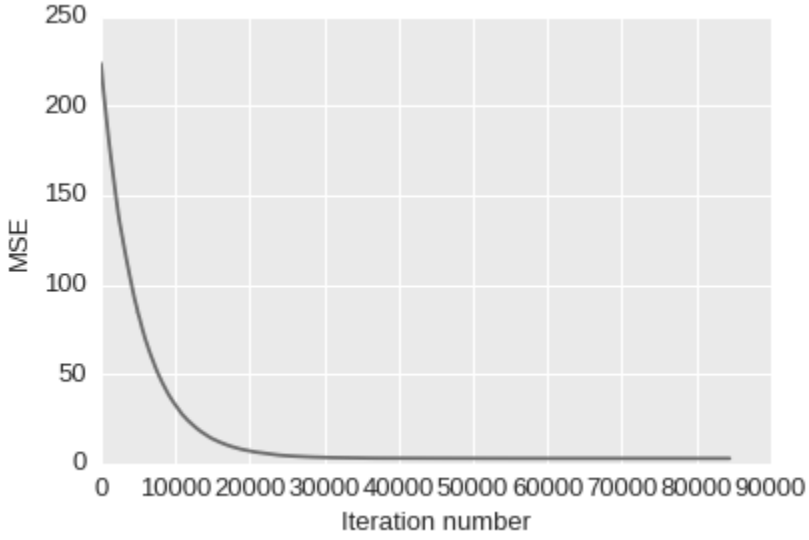

Рис. 3. Зависимость среднеквадратической ошибки от количества итераций

Посмотрим вектор весов, к которому сошелся метод:

```
print(stoch_grad_desc_weights)
[ 3.91069256e+00  2.78209808e+00  -8.10462217e-03  1.40190566e+01 ]
```

Рис. 24. Вектор весов, полученный после градиентного спуска

Оценим среднеквадратичную оценку прогноза:

```
w_init  = np.zeros(4)
stoch_grad_desc_weights,  stoch_errors_by_iter  = stochastic_gradient_descent(X,  y,
w_init,  eta=1e-2, max_iter=1e5,
min_weight_dist=1e-8,  seed=42, verbose=False)
predictions  = linear_prediction(X,  weights)
err = mserror(y,  predictions)
print(err)
2.78441258841
```

Рис. 25. Среднеквадратическая ошибка при использовании стохастического градиентного спуска

Выводы

По полученным результатам видно, что как стохастический градиентный спуск, так и нормальное уравнение показывают одинаковый ответ. Но есть некоторые нюансы. Например, при решении поставленной задачи с помощью нормального уравнения мы не застрахованы от мультиколлинеарности, когда строки представимы в виде линейной комбинации других. В этом случае просто не представляется возможным вычислить первообратные матрицы признаков, а значит, посчитать вектор весов. Также достаточно долгое время понадобиться на расчеты, и если выборка большая, то мы не сможем оперативно прогнозировать данные.

В случае стохастического градиентного спуска мы берем только по одной случайной строке матрицы признаков и продолжаем вычисления до сходимости функционала ошибки. Это отнимает меньше времени, а значит, подходит для онлайн обучения, когда признаки объектов приходят в реальном режиме времени, и от нас требуется оперативно делать прогнозы.

Временные ряды. Основные понятия

Временным рядом называется последовательность значений, описывающих протекающий во времени процесс, измеренных за равные промежутки времени. Данные представленные в виде временных рядов широко распространены в мире. Их используют для анализа экономических задач, таких как прогнозирование цен на акции или в медицине для анализа ЭКГ. Вектор функции $\begin{pmatrix} X_1(t) \\ Xr(t) \end{pmatrix}$, все компоненты Xi(t), i = 1 .. r которых действительны, а t принимает значение 0, 1, 2,… Такую совокупность функций называют r-компонентным векторным временным рядом [3]. Переменная t, как правило, соответствует времени выполнения или наблюдения измерений. Среди временных рядов выделяются стационарные и нестационарные ряды. Ряд Y(t) называется стационарным, если закон распределения вероятностей случайной величины Y(t) не зависит от t.

В данной работе будет применено прогнозирование временного ряда. Другими словами, будет построена модель для предсказания будущих событий, основываясь на исторически известные данные. В качестве модели будет использована модель смешанного авто регрессионного скользящего среднего (ARIMA, Auto Regression Integrated Moving Average). С помощью данной модели можно описывать как стационарные, так и не стационарные временные ряды. Разница между стационарным и нестационарным временным рядом заключается в том, что случае стационарного временного ряда данные колеблются относительно некоторого фиксированного уровня, а в не стационарных временных рядах такого фиксированного уровня нет.

Метод наименьших квадратов

Пусть w – набор из неизвестных переменных, а $Y_i^*(w)$ – значения функции в данных точках w_i, которую нам необходимо найти, предсказать. То есть понять, как значения ряда w_i соотносятся со значениями $Y_i^*(w)$, при этом не зная действительную зависимость Y_i. Задача метода наименьших квадратов заключается в подборе таких значений w, чтобы значения функций $Y_i^*(w)$ были максимально близки к значениям Y_i. По существу, это задача решения неопределенной системы уравнений $Y_i^*(w) = Y_i$. Нам необходимо найти максимальную близость левой части уравнения к правой. Суммируя выше написанное, можно сказать, что смысл метода наименьших квадратов заключается в выборе в качестве меры близости предсказанного и реального значения суммы квадратов отклонений левых и правых частей уравнений. $Q_t(w) = \sum_{t=t_0}^{t} (Y_i^* - Y_i)^2$.

Пусть на вход системы поступили два вектора X и У. Связь между ними нам неизвестна. Давайте найдем эту связь, а также аппроксимируем пришедшую на вход зависимость У(X) методом наименьших квадратов. Для этого будем использовать язык Python, а также библиотеки, необходимые для анализа данных: numpy, scipy, matplotlib. Продемонстрируем пришедшие на вход системы данные.

Таблица 1. Сгенерированные случайные числа

I	Xi, 10^-1	Yi, 10^-1	I	Xi, 10^-1	Yi, 10^-1	i	Xi, 10^-1	Yi, 10^-1
1	0.747139	0.041241	8	2.216164	2.1115220	15	6.423176	6.2798301
2	0.808072	0.296335	9	2.613301	2.5649689	16	7.306958	6.5155498
3	1.058513	0.315873	10	3.723365	2.6410298	17	7.805258	7.4598499
4	1.211347	0.327252	11	3.95953	3.6557723	18	8.005979	8.6286727
5	1.362910	0.687259	12	4.039241	5.5371714	19	9.261030	8.7890424
6	1.636960	1.225493	13	5.362331	5.7069968	20	9.623829	9.3075027
7	1.660332	1.457618	14	5.955518	5.7943989			

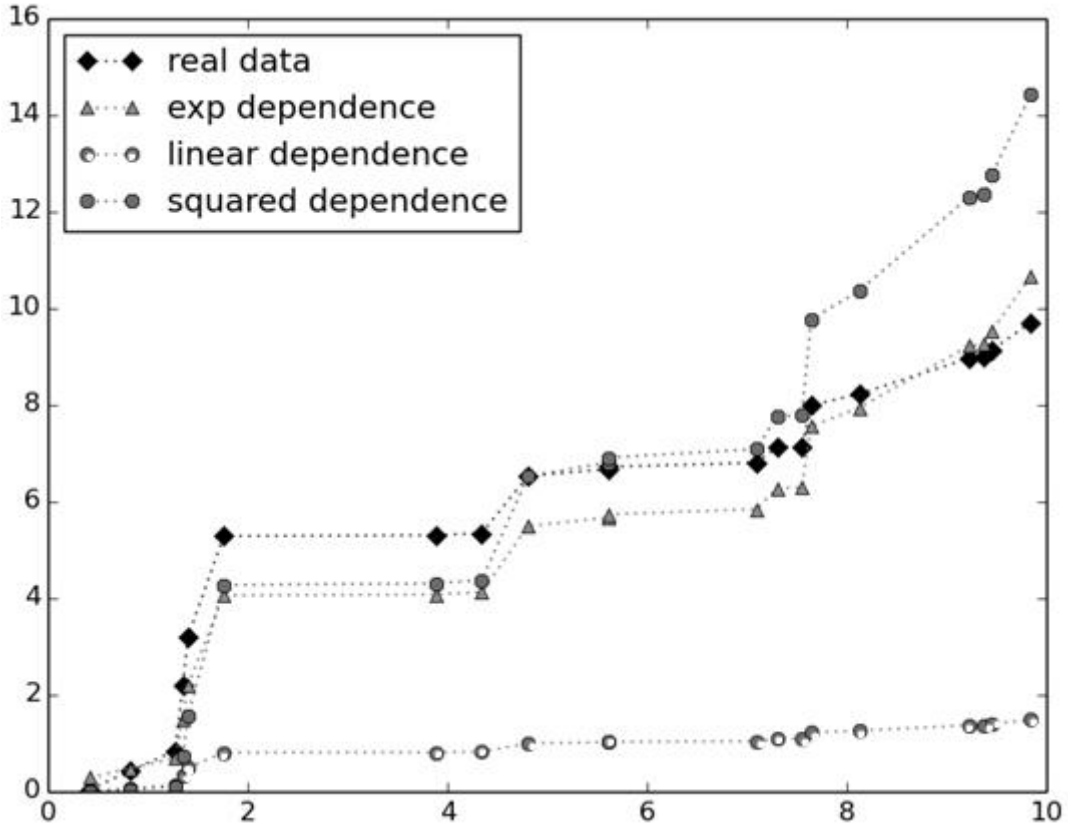

Рис. 26. Оценка линейной, экспоненциальной и квадратичной аппроксимации исходной зависимости

Так давайте предположим, что полученную на вход неизвестную зависимость можно аппроксимировать экспоненциальной зависимостью с некоторым коэффициентом k перед x. По смыслу применение метода наименьших квадратов есть отыскание решения уравнений $Y_i^*(w) = e^{kx} = Y_i$, или, другими словами, нахождение максимальной близости между правой и левой частью уравнений. Также предположим, что функционал ошибки можно минимизировать, попробовав аппроксимировать исходную зависимость линейной и квадратичной (параболической) функциями. Изучая библиотеку scipy языка Python, обнаружили модуль, способный помочь при использовании метода наименьших квадратов.

```
def fitf(koeff,  x):
        return  koeff[1]*exp(koeff[0]*x)  + koeff[2]


def fitf_2(koeff,  x):
        return  koeff[0]*x


def fitf_3(koeff,  x):
        return  koeff[0]*x**2


def residuals(koeff,  x, y):
        return  [y - fitf(koeff,  x), y - fitf_2(koeff,  x), y - fitf_3(koeff,  x)]
```

Рис. 27. Реализация функций для экспоненциальной, линейной и квадратичной аппроксимации

Здесь мы объявили функцию, которая, по нашему соображению, может аппроксимировать распределение, поступившее на вход системы. Функция residuals (koeff, x, y), принимает на вход данные по осям x и y, полученные в ходе аппроксимации, соответственно, и возвращает разности между аппроксимируемыми значениями и значениями У, полученными на входе системы. Данные функции было необходимо ввести, чтобы использовать МНК (метод наименьших квадратов). Найдем вид аппроксимирующих функций, а затем построим их.

```
val_x = optimize.leastsq(residuals, k[:], arg=(x,y), full_output = True)[0]

val_lin = optimize.leastsq(residuals, k[:], arg=(x,y), full_output = True)[0]

val_sq = optimize.leastsq(residuals, k[:], arg=(x,y), full_output = True)[0]

line_real = plt.plot(y, x, "bD:", label='real data')

line_exp = plt.plot(y, fitf(val_x[:], x), 'r^:', label='exp dependence')

line_lin = plt.plot(y, fitf_2(val_lin[:], x), 'go:', label='linear dependence')

line_sq = plt.plot(y, fitf_3(val_sq[:], x), 'm8:', label='squred dependence')

plt.legend(loc='best')

plt.show()
```

Рис. 28. Построение полученных аппроксимирующих значений

Анализируя полученные результаты, вполне обоснованно можно сказать, что, сделав предположение о том, что экспоненциальная зависимость хорошо аппроксимирует входящее распределение и применив МНК, мы оказались правы.

Модель авторегрессии

Смысл линейной модели авторегрессии заключается в предположении, что прогнозируемое значение за горизонтом событий. Момент времени t+d есть ничто иное, как линейная комбинация значений временного ряда в предыдущие n наблюдений.

Пусть мы имеем l = t-n+1 моментов в истории ряда. Тогда опишем объекты временного ряда с помощью матрицы значений и вектора решений. Объекты – моменты времени, наблюдения. $F_{l*n} = \begin{pmatrix} y_t & \cdots & y_{t-n+1} \\ \vdots & \ddots & \vdots \\ y_n & \cdots & y_1 \end{pmatrix}$, целевой вектор – $\begin{matrix} y_{t+1} \\ y_t \\ y_{n+1} \end{matrix}$. Данная модель хорошо работает, когда надо спрогнозировать небольшое количество элементов ряда в будущем. Так как в каждый новый момент времени поступают новые признаки объекта t+1 временного ряда, то нам необходимо обновлять матрицу F, а также целевой вектор Y. Матрица обновляется следующим образом: W = (F^T * F)^-1 * F_y^T. В этом случае полученный функционал квадрата ошибки представляет следующую запись: Q_t(w, X^l) = $\sum_{i=n+1}^{t+1} (y_i^*(w) - y_i)^2 = \|F_w - y\|^2 \to min_w$.

Учтём временной ряд с весами. Каждому историческому значению временного ряда будем ставить в соответствие коэффициент β, который тем меньше, чем больше ряд уходит в историю. Тогда МНК представим в форме: $\sum_{i=0}^{t} \beta^{t-i} * (y_i - c)^2$, где c = y_i^*. Положим $\beta \in (0,1)$. Продифференцировав

$\sum_{i=0}^{t} \beta^{t-i} * (y_i - c)^2$ по переменной c и приравняв полученное выражение к 0 и выразив c, получим аналитическое решение: $c = y^*_{t+1} = \frac{\sum_{i=0}^{t} \beta^i * y_{t-i}}{\sum_{i=0}^{t} \beta^i}$. Полученное выражение формула Надарая-Ватсона [4]. Определение константы *c* будет происходить по ее взвешенной обучающей выборке с учетом веса объекта β^t. Чем дальше объект с индексом i будет находиться от текущего положения, тем меньше будет его вес, а это значит, тем меньше будет его вклад в прогнозируемое значение. Вес объектов будем вычисляться по геометрической убывающей прогрессией, знаменатель которой β.

Времени на вычисление формулы $c = y^*_{t+1} = \frac{\sum_{i=0}^{t} \beta^i * y_{t-i}}{\sum_{i=0}^{t} \beta^i}$ понадобиться O(l), l – количество элементов временного ряда. При бесконечном (достаточно большом) ряде $\sum_{i=0}^{t} \beta^i = \frac{1}{1-\beta}$. Записав выражение $y^*_{t+1} = \frac{\sum_{i=0}^{t} \beta^i * y_{t-i}}{\sum_{i=0}^{t} \beta^i}$ для y_t, а после выразив y^*_{t+1} через y_t, получим: $y^*_{t+1} = y_t$ *(1-β) + $y^*_t * \beta$. Сделав замену 1-β = α, получим $y^*_{t+1} = y_t + \alpha(y_t - y^*_t) = \alpha * y_t + (1 - \alpha) * y^*_t$. Распишем смысл проделанной работы. Прогноз в следующий момент времени представляется возможным вычислить как линейную комбинацию с коэффициентами β, 1-β прогноза в прошлый момент времени и значения, которое наблюдаем в текущий момент времени. Это некоторое усреднение временного ряда и β нам дает то, что, усредняя все время два значения, мы фактически будем строить взвешенную сумму всех предыдущих y_t, и чем дальше в историю ряда будем уходить, тем меньше y_t, которые мы учитываем.

Рассмотрим, что будет происходить при выборе α. Если будем брать α малую, стремящуюся к 0, то будет происходить усреднение временного ряда, если большую, стремящуюся к 1, то будет учитываться больше вес последних данных, а исторические данные практически не будут вносить изменений. Формула $y^*_{t+1} = y_t + \alpha(y_t - y^*_t)$ называется экспоненциальное скользящее средние (ЭСС). ЭСС подходит для прогнозирования как стационарных рядов, так и не для стационарных.

Рассмотрим на примере работу ЭСС. Для этого воспользуемся языком программирования Python и библиотеками для анализа данных Pandas, numpy и matplotlib. Подадим на вход системы случайно сгенерированные числа и даты от 1 марта и до 20 включительно:

x = np.random.rand (20)*10;

y = pd.date_range (‘2016-3-1’, ‘2016-3-20’, freq=’D’).

Сопоставим временному ряду, случайно полученный вектор значений:

row = pd.Series (x, y).

Теперь запустим экспоненциальное скользящее среднее для полученного временного ряда с разными α = (0.1, 0.5, 0.9): ema_i = pd.ewma(α_i).

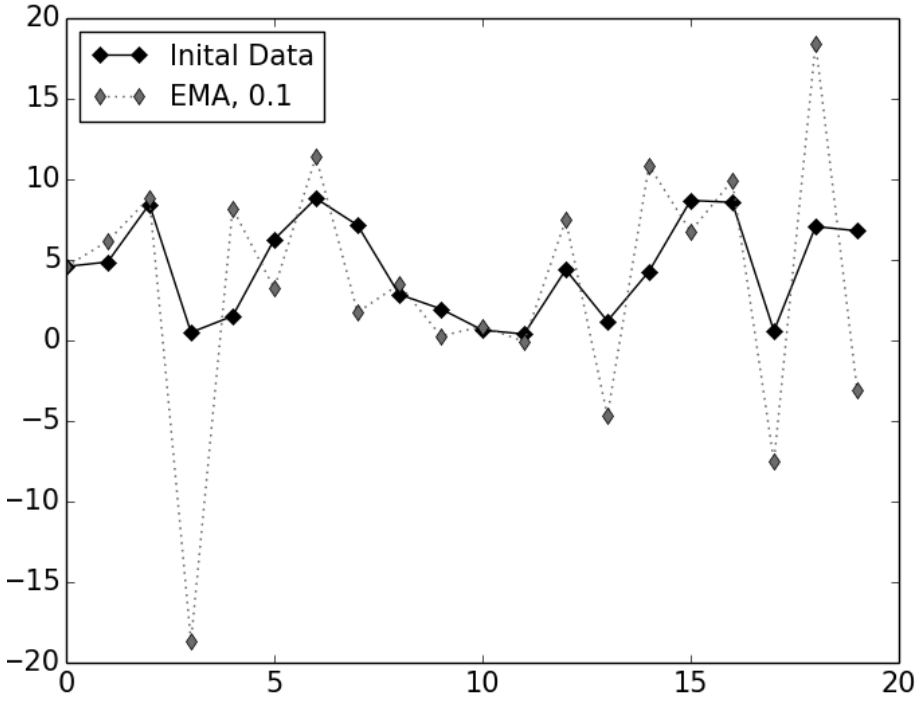

Рис. 29. Экспоненциальное скользящее среднее, α = 0.1

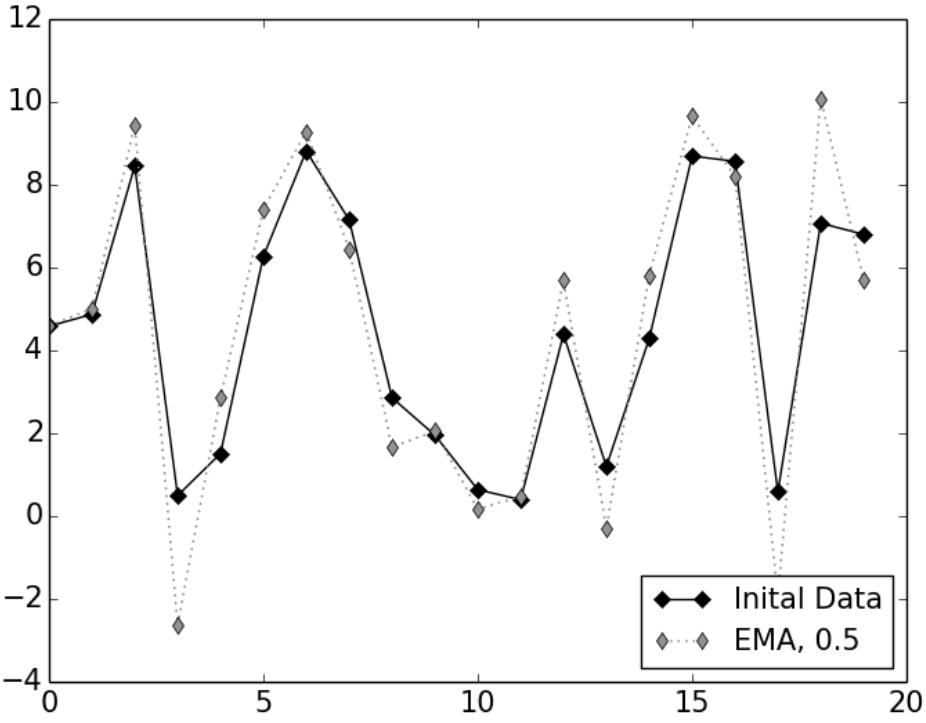

Рис. 30. Экспоненциальное скользящее среднее, α = 0.5

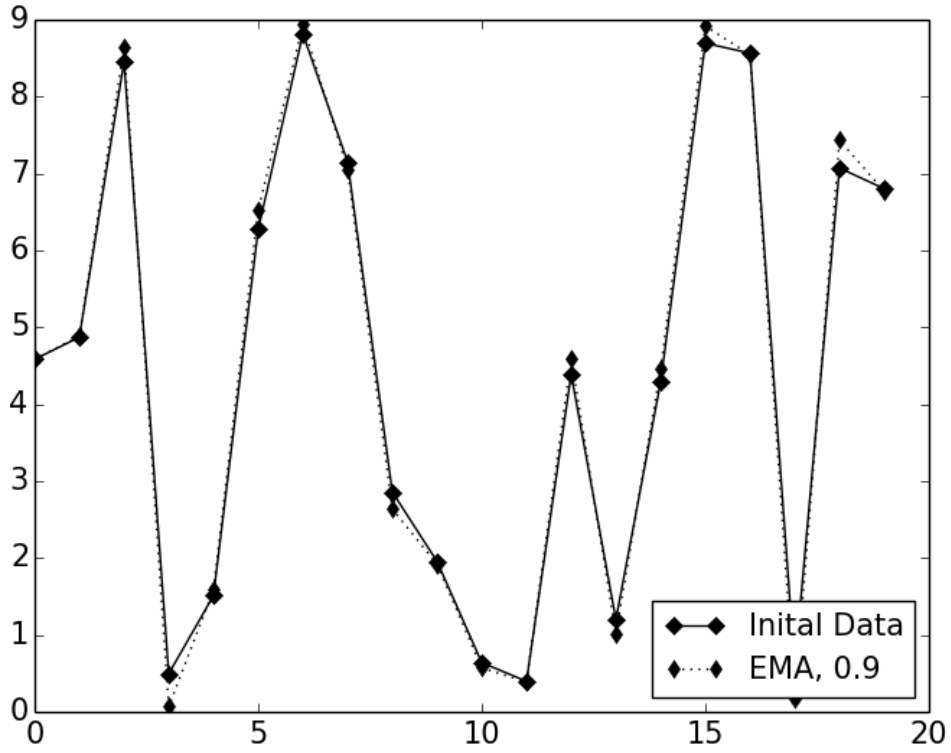

Рис. 31. Экспоненциальное скользящее среднее, α = 0.9

Анализируя полученные зависимости, можно сказать, что теоретические предположения об экспоненциальном скользящем среднем, сделанный выше, оказались верными.

Модель прогнозирования arima (p, q, d)

Модель прогнозирования ARIMA, разработанная Боком-Дженкинсом, является интегрированной моделью авторегрессии и скользящего среднего. ARIMA широко применяется для прогнозирования нестационарных временных рядов. Суть метода в том, что представляется получить из нестационарного временного ряда стационарный, путем взятия разностей некоторого порядка от исходного временного ряда.

В рассматриваемой модели используется итеративный подход к нахождению подходящей модели, удовлетворяющей поставленной задаче среди множества класса моделей. После выбора происходит ее сопоставление с историческими данными, оценивается точность работы. Если нас устраивает, как модель описывает ряды, то ее оставляют, в противном случае процесс повторятся, но уже с использованием другой модели.

Рассмотрим стратегию Бокса-Дженкинса [3]:

Рис. 32. Стратегия Бокса-Дженкинса

Из чего состоит arima (p, q, d)

Напомню, что авторегрессинная модель n-го порядка выглядит следующим образом [3]:

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t$$

Перед нами формула, позволяющая значение временного ряда в момент времени t выразить как линейную комбинацию предыдущих значений ряда, помноженных на коэффициент. $\phi_1 \ldots \phi_p$ - оцениваемые коэффициенты, $Y_{t-1} \ldots Y_{t-p}$, p исторических элементов временного ряда.

Продемонстрируем пример прогнозирования, применив авторегрессию. Возьмем наблюдаемые данные средние стоимости акций компании IBM за любой промежуток в 5 дней.

Таблица 2. Прогнозирование временного ряда с использованием авторегрессионной модели

Период	Значение Yt	Прогнозы Yt^	Остатки
t-5	90	76.4	13.6
t-4	78	67.5	10.5
t-3	87	74	13
t-2	99	69.1	29.9
t-1	72	62.7	9.3
t		77.2	

Взяв модель AR(2), что означает прогнозирование временного ряда, исходя из последних двух элементов ряда. Использовав метод наименьших квадратов, нашли следующие коэффициенты: $\phi_0^{=115.2}$, $\phi_1^{=} -0.535$ $\phi_2^{=0.005}$. Работая с методом наименьших квадратов, мы минимизировали функционал ошибки, изменяя коэффициенты. Затем выбрали те коэффициенты, которые показывали на обучающей выборке минимальную ошибку. Теперь, подставляя коэффициенты в уравнение авторегрессии, получим значение Y = 77.2.

Вообще, говоря о прогнозировании временных рядов в примере, мы взяли слишком малую выборку. Надежнее выбирать выборки с большим количеством элементов временного ряда. Выражение AR(n) имеет смысл, что мы выбираем n последних элементов ряда для прогноза события, находящееся за горизонтом событий.

Напомню про скользящее среднее. Модель со скользящим средним описывается следующим уравнением [3]:

$$Y_t = \mu - w_1 \varepsilon_{t-1} - w_2 \varepsilon_{t-2} + \cdots + \phi_q \varepsilon_{t-q} + \varepsilon_t$$

μ - постоянное среднее процесса;

$\varepsilon_t \ldots \varepsilon_{t-q}$ - ошибки в предыдущие моменты времени;

$\phi_t \cdots \phi_q$ - оцениваемые коэффициенты.

Данное уравнение похоже на уравнение авторегрессии за исключением того, что прогнозируемый элемент зависит от предыдущих значений ошибок. Таким образом, модели MA (скользящее среднее) дают прогноз на основе линейной комбинации ограниченного количества ошибок прошлых значений, в то время как модели AR (авторегрессия) делают прогноз на основе линейной функции аппроксимации ограниченного числа прошлых значений.

Таблица 3. Прогнозирование временного ряда с использованием модели скользящего среднего

Период	Значение Yt	Прогнозы Yt^	Остатки
t-5	90	76.1	13.9
t-4	78	69.1	8.9
t-3	87	75.3	11.7
t-2	99	72	27
t-1	72	64.3	7.7
t		77.2	

Спрогнозируем новое значение ряда, применяя MA (2). Проделаем ту же процедуру, что было описано с авторегрессией. Получили следующие коэффициенты: $\mu_0^{=75.4}$, $w_1^{=0.5667}$, $w_2^{=} - 0.3560$.

Теперь, подставляя коэффициенты в уравнение скользящего среднего, получим значение Y = 77.2.

Теперь поставим задачу совместить авторегрессию и скользящее среднее. Получится модель ARMA (p, q), p и q – параметры AR и MA соответственно [3].

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t - w_1 \varepsilon_{t-1} - w_2 \varepsilon_{t-2} - \cdots - \phi_q \varepsilon_{t-q}$$

Анализируя полученную формулу, делаем вывод, что ARMA (p, q) делает прогноз, который зависит от предыдущих и текущих значений Y, а также зависит от предыдущих и текущих величин ошибки e.

Итак, опишем, что такое модель ARIMA (p,q,d). Предположим, что мы имеем какой-то ряд. Возьмем последние разности этого ряда [3]:

$$Y_t = Y_t - Y_{t-1})$$.

Если разности изменяются в окрестностях некоторого фиксированного значения, то есть разности постоянны, то мы имеем дело со стационарным временным рядом. Теперь предположим, что ряд является нестационарным, а значит, взятие разности первого порядка по ряду покажет нам хаотичный разброс значений около фиксированного уровня. Поставим перед собой задачу взять разность столько раз, чтобы получить стационарность.

Пример для разности второго порядка [3]:

$$(Y_{t-1} - Y_{t-2}) = Y_t - 2Y_{t-1} + Y_{t-2})$$

Значение d в модели ARIMA (p, q, d) показывает порядок вычета.

Архитектура приложения

В разработанном приложении использовались языки программирования Java и Python. Java выступил в роли сервера, который посылает запросы на сервер биржи по протоколу HTTPS. Полученные данные от сервера биржи сохраняет в таблицу Stocks в СУБД MySQL. Также сервер Java поддерживает связь с клиентским модулем, написанным на Python, через протокол TCP/IP. Роль клиентского модуля на Python – анализ данных и прогнозирование финансового временного ряда. После расчета прогноза, принятия решения о покупке или продаже модуль python отправляет сообщение по протоколу TCP/IP серверу Java о действии, которое необходимо совершить: купить, продать, ничего не делать.

Структура таблиц базы данных

В данной работе, в СУБД было организовано только две таблицы: STOCKS и REPORT. Таблица STOCKS служит для сохранения текущих значений валютных пар, полученных сервером Java. К этой таблице также обращается клиент Python для формирования обучающей выборки. Столбцы таблицы STOCKS:

STOCK_ID – первичный ключ таблицы типа INTEGER.

BID – цена покупки акции, тип – FLOAT.

ASKID - цена покупки акции, тип – FLOAT.

TIME – актуальное время для свойств акции – TIMESTAMP.

NAME – название акции.

Также была создана таблица REPORT, в которой занесены спрогнозированные значения и действительные значения на рассматриваемый момент времени. Остальные столбцы соответствуют таблице STOCKS.

Схема приложения

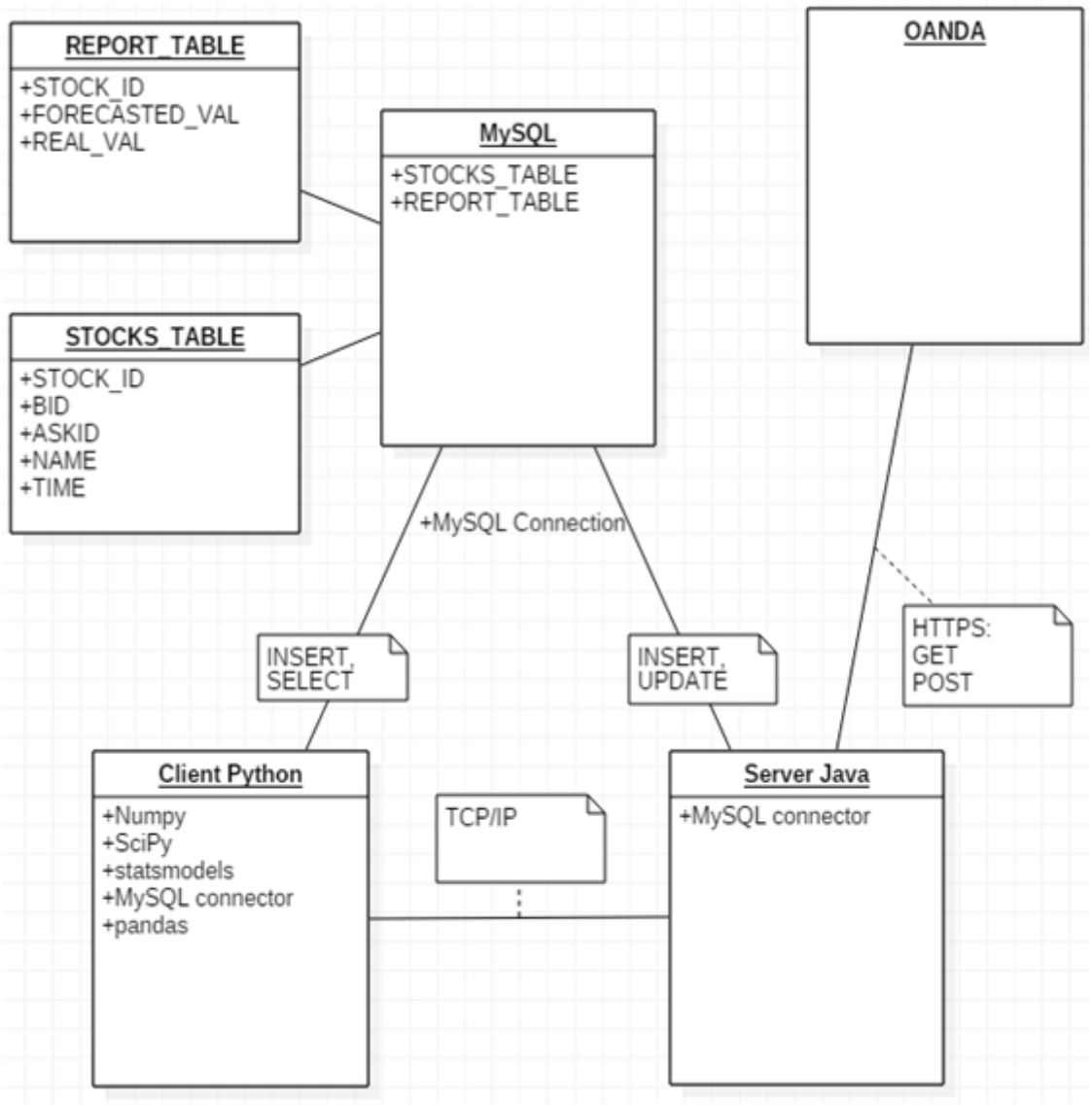

Рис. 33. Схема приложения

Описание технологий приложения

На стороне Python были задействованы библиотеки для анализа данных, машинного обучения и статистики. Библиотека Numpy – позволяет генерировать многомерные матрицы, вектора, а также эффективно их обрабатывать. Библиотека SciPy – дала возможность эффективно считать произведения матриц, находить среднеквадратичную ошибку, средние значения, вычислять первообратную матрицы, а также решать матричные уравнения. Библиотека Statmodels – предоставила возможность работать с ARIMA, вычислять адекватные коэффициенты для выбранной модели. Библиотека Pandas – позволила обрабатывать датасеты, загруженные из СУБД, и преобразовывать их во временные ряды.

Измерение точности работы биржевого робототта

Для оценки точности работы торгового робота был построен доверительный интервал. Интервал откладывался от действительных значений валютной пары евро-доллар. Доверительный интервал искался следующим образом:

- Была взята выборка действительных значений валютных пар.
- Было найдено значение стандартного отклонения.
- Стандартное отклонение было отложено в большую и меньшую стороны относительно действительных значений.

Использованная формула, по которой строился доверительный интервал: X +- 1.96*SEM, где X – действительное значение курса, SEM – стандартная ошибка среднего. SEM = S /$\sqrt{n}$, S – стандартное отклонение, n – количество наблюдений. В данной формуле попадание спрогнозированной величины в интервал означает, что полученное значение находится в 95 % доверительном интервале.

Представлю алгоритм вычисления и построения доверительного интервала, а также спрогнозированных величин. Полностью приложение робота можно посмотреть:

1) Серверный модуль [7]

2) Аналитический модуль [8].

```
__author__ = 'dimas'
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from MySQL_db_Connection.ConnectionHandler import DataHandler
def Parser(string):
        whole_lst = string.split("\n")
        time_lst, real_data_lst, predicted_data_lst = [], [], []
        for i in whole_lst:
        lst = i.split(" ")
        if len(lst) > 3:
                time_lst.append(lst[2] + " " + lst[3])
                real_data_lst.append(float(lst[0]))
                predicted_data_lst.append(float(lst[1]))
                time_lst = pd.to_datetime(time_lst)
                real_data_series = pd.TimeSeries(real_data_lst, index=time_lst)
                predicted_data_series = pd.TimeSeries(predicted_data_lst,
index=time_lst)
        return real_data_lst, predicted_data_lst, time_lst
```

Рис. 34.1. Первая часть кода, отвечающего за оценку работы робота

```
data = DataHandler("localhost", "Dimas", "Dimas", "Forex")
s = data.GetReportForResearch()
real_data, predicted_data, time_ser = Parser(s)
real_data = np.array(real_data)
predicted_data = np.array(predicted_data)
deviation = np.std(real_data) / np.sqrt(len(real_data))
real_data_plus = real_data + 1.96*deviation
real_data_minus = real_data - 1.96*deviation
plt.plot_date(x=time_ser, y=real_data, fmt="b-", label="Real Data", linewidth=2.0)
plt.plot_date(x=time_ser, y=real_data_plus, fmt="r--", label="Trustee Interval")
plt.plot_date(x=time_ser, y=real_data_minus, fmt="r--")
plt.plot_date(x=time_ser, y=predicted_data, fmt="kD:", label="Predicted Data", linewidth=2.0)
plt.legend(loc='best')
plt.show()
```

Рис. 34.2. Вторая часть кода, отвечающего за оценку работы робота

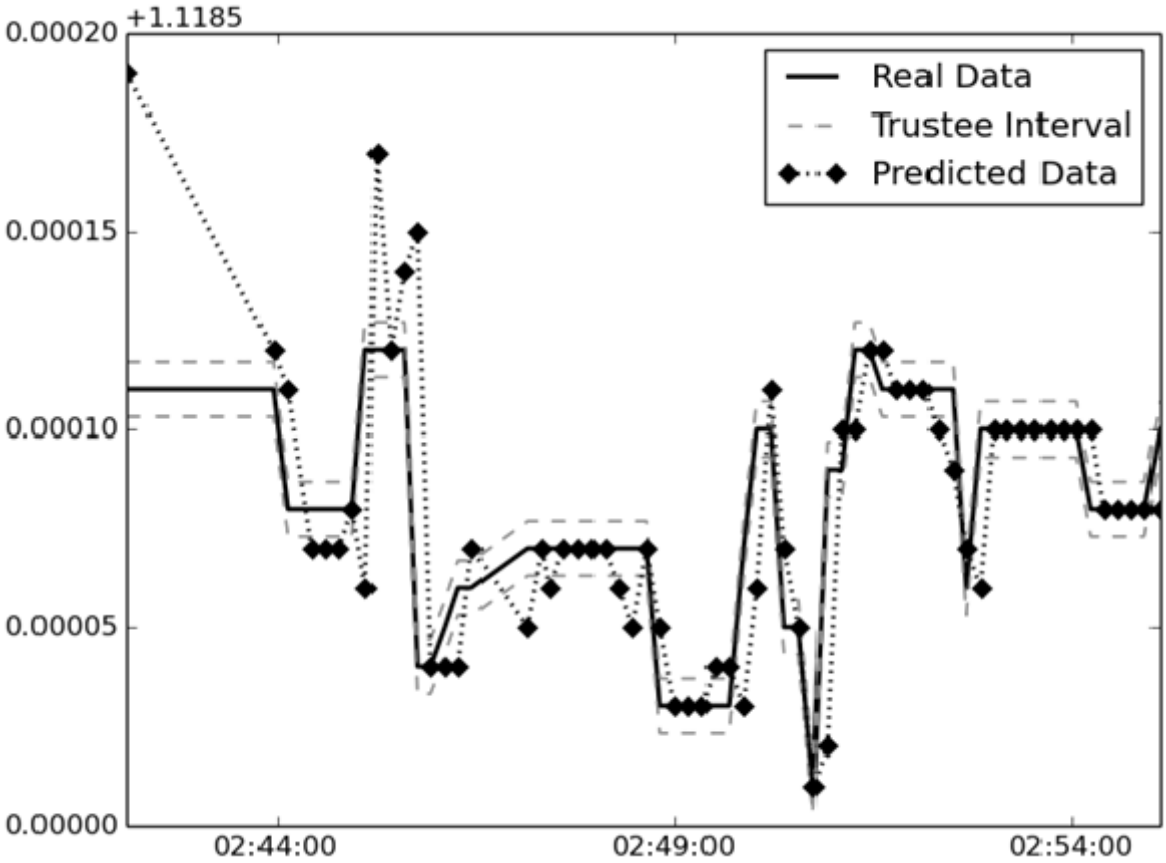

Рис. 35. Оценка точности работы робота

Литература

1. UC Irvine Machine Learning Repository [Электронный ресурс]. – Режим доступа: https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data (дата обращения: 20.05.2016).
2. An Introduction to Statistical Learning [Электронный ресурс]. – Режим доступа: http://www-bcf.usc.edu/~gareth/ISL/data.html (дата обращения: 17.05.2016).
3. *Бриллинджер Д.* Временные ряды. Обработка данных и теория. М.: Мир, 1980. 532 с.
4. *Андерсон Т.* Статистический анализ временных рядов. М.: Мир, 1976. 744 с.
5. *Швагер Джек.* Технический анализ. Полный курс. М.: Альпина Паблишер, 2001. 768 с.
6. *Флах П.* Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2015. 400 с.
7. GitHub repositories [Электронный ресурс]. – Режим доступа: https://github.com/DimasBro0110/Stocs (дата обращения: 8.6.2016).
8. GitHub repositories [Электронный ресурс]. – Режим доступа: https://github.com/DimasBro0110/ForexAnalysis (дата обращения: 8.6.2016).