МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**ПРИМЕНЕНИЕ МЕТОДОВ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДЛЯ ПРЕДСКАЗАНИЯ РЕЗУЛЬТАТОВ СПОРТИВНЫХ СОРЕВНОВАНИЙ**

Работу выполнил М.И. Ващанов

(подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики 3 курс

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Научный руководитель

доц., к.т.н., Т.А. Приходько

(подпись, дата) (инициалы, фамилия)

Нормоконтролер

ст.преп., к.т.н., Е.Е. Полупанова

(подпись, дата) (инициалы, фамилия)

Краснодар 2018

СОДЕРЖАНИЕ

[Введение 3](#_Toc515633211)

[1 Система сбора и хранения информации 4](#_Toc515633212)

[1.1 Постановка задачи для системы сбора информаци 4](#_Toc515633213)

[1.2 Поиск матчей и сбор статистики 4](#_Toc515633214)

[1.3 Промежуточное хранение данных 5](#_Toc515633215)

[1.4 Система хранения информации 6](#_Toc515633216)

[1.5 Реализация непрерывной работы системы сбора информации 7](#_Toc515633217)

[2 Система анализа данных 8](#_Toc515633218)

[2.1 Инструменты анализа данных 8](#_Toc515633219)

[2.2 Визуальный анализ данных 10](#_Toc515633220)

[2.3 Постановка задачи на прогнозирование результата матча 13](#_Toc515633221)

[2.4 Формальная постановка задачи классификации 14](#_Toc515633222)

[2.5 Общие сведения о градиентном бустинге на решающих деревьях 14](#_Toc515633223)

[2.6 Подбор гиперпараметров и обучение модели 16](#_Toc515633224)

[2.7 Оценка качества модели с помощью кросс-валидации 18](#_Toc515633225)

[3 Анализ полученных результатов 19](#_Toc515633226)

[3.1 Анализ системы в разрезе точности критериев оценки результатов матча 19](#_Toc515633227)

[3.2 Анализ системы в разрезе "уверенности" модели в своей классификации 20](#_Toc515633228)

[Заключение 23](#_Toc515633229)

[Список использованных источников 24](#_Toc515633230)

[Приложение Листинг программы 25](#_Toc515633231)

ВВЕДЕНИЕ

В настоящее время можно наблюдать бурное развитие систем и технологий сбора, хранения и обработки информации. Также расширяется и количество данных, которые используются в современных системах. В связи с этим стало возможным исследование и предсказание в режиме реального времени результата спортивных соревнований. В курсовой работе рассматриваются задачи сбора данных обо всех футбольных матчах, которые идут в момент работы системы, хранения и обновления этих данных, а также задачи анализа и предсказания результатов конкретного матча.

На данный момент существуют методы анализа статистики футбольного матча, которые доказали свою эффективность на практике, однако информации, какими способами были найдены эти методы в открытом доступе нет.

Целью работы является построение системы, предсказывающей результат футбольного матча по первым 15 минутам игры.

Курсовая работа состоит из двух частей.

Первая часть работы содержит описание работы систем сбора и хранения информации. В ней рассматриваются вопросы создания системы сбора данных из сети Интернет в режиме реального времени, отслеживание каждого футбольного матча, обнаруженного системой, до его завершения, а также вопросы способа хранения информации о матче.

Вторая часть данной работы посвящена интеллектуальному анализу данных. В неё входят построение предсказательных моделей, проверка их на корректность, выводы.

# 1 Система сбора и хранения информации

### Постановка задачи для системы сбора информации

Система сбора информации должна решать следующие три задачи. Первой задачей является поиск футбольных матчей, удовлетворяющих критерию времени, прошедшего с начала матча. Второй задачей является непосредственно сбор статистики о футбольном матче. Третьей задачей является промежуточное хранение данных до завершения матча.

### Поиск матчей и сбор статистики

Для решения задач поиска матчей и сбора статистики используется сайт betsapi.com. Для работы с сайтом используется библиотека для синтаксического разбора файлов HTML/XML beautiful soup 4. На сайте информация представляется в виде таблицы матчей. На рисунке 1 изображена таблица футбольных матчей с указанием времени с момента начала матча.



Рисунок 1 – Пример таблицы футбольных матчей

Для всех матчей, которые идут 10-20 минут, запоминается статистика игры. На рисунке 2 показан пример таблицы со статистикой для конкретного матча.

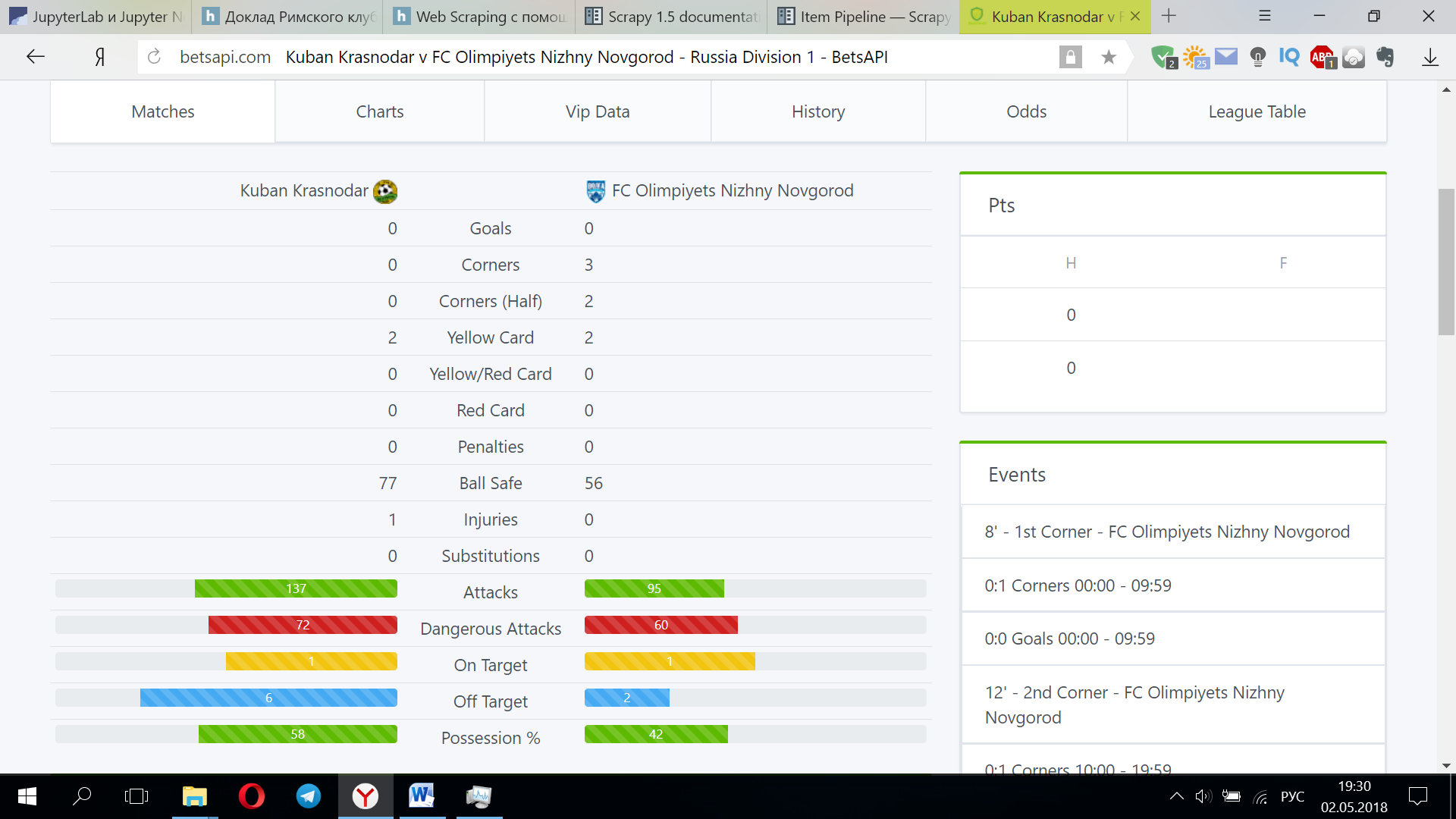


Рисунок 2 – Пример таблицы со статистикой матча

Эта статистика должна запоминаться системой, а после к ней должна быть добавлена и информация и количестве голов за весь матч. Следовательно, встаёт вопрос о том, как же хранить эту статистику и как организовать добавление данных.

### Промежуточное хранение данных

Для промежуточного хранения данных было рассмотрено 3 варианта организации системы хранения.

Первым вариантом было сохранение в базу данных статистики с последующим дописыванием туда количества голов в первом тайме и во всей игре. Достоинством данной схемы является относительно простая реализация. Недостатком является то, что в количество обращений к базе данных будет в три раза больше, чем при других вариантах. Такая система будет плохо масштабируема в будущем.

Два других варианта основываются на идее о том, что сначала нужно собрать все данные матча, а потом записывать их все сразу в базу данных.

Вторым вариантом является сохранение всей информации в оперативную память. Преимуществами такого решения является очень высокая скорость, практически отсутствие затрат на запись и считывание данных, простая реализация. Минусом является потеря данных при сбое системы.

Третьим вариантом является сохранение информации в NoSQL хранилище Redis. Данный вариант является компромиссным между первым и вторым вариантами: так как в этом хранилище можно настроить произвольный таймер на запись данных в ПЗУ, то нагрузка на ПЗУ будет относительно невелика, а риск, что при сбое данные потеряются будет также сравнительно мал при достаточно маленьком значении таймера. Несмотря на кажущуюся привлекательность данного метода, он довольно сложен в реализации, и чтобы его использовать, необходимо ответить на некоторые другие вопросы. Например, что будет, если система даст сбой и не все данные будут записаны на диск? Каким образом обрабатывать вновь запущенной системе вариант, когда в хранилище уже есть какие-то данные, о которых текущей системе неизвестно? Даже если система знает, как обрабатывать данные, то возможен вариант, когда на сайте, с которого считывались данные, больше не существует страницы с тем матчем, статистика которого была записана. Это может произойти, например, при достаточно продолжительном отключении системы.

Ответы на эти вопросы выходят за рамки данной работы, поэтому для решения задачи промежуточного хранения данных был выбран вариант с хранением статистики в оперативной памяти.

### Система хранения информации

В качестве долговременного хранилища статистики была выбрана СУБД SQLite 3. Данная СУБД широко распространена, является простой в использовании. Одним из существенных минусов этой СУБД является то, что для доступа к ней используются файловые дескрипторы операционной системы. Поэтому она не предназначена для систем с очень высокой нагрузкой. Однако, для того, чтобы она стала непригодной, необходимо, чтобы количество запросов в секунду к ней достигало 1000 и более. Реализуемая система и близко не подбирается к этим параметрам, так как сложно представить, что в мире больше одной тысячи футбольных матчей начинаются одновременно. Следовательно, использование SQLite3 в качестве основной базы данных полностью оправдано.

### Реализация непрерывной работы системы сбора информации

Так как система работает постоянно, необходимо продумать механизмы планирования задач и распределения нагрузки. Эти механизмы реализованы в библиотеке Celery [1]. Celery – это распределенная асинхронная очередь заданий, которая обладает широким функционалом. Celery умеет:

– выполнять задания асинхронно или синхронно;

– выполнять периодические задания;

– выполнять отложенные задания;

– распределенное выполнение;

– в пределах одного worker'а возможно конкурентное выполнение нескольких задач (одновременно);

– выполнять задание повторно, если было выброшено исключение;

– ограничивать количество заданий в единицу времени;

– routing заданий;

– несложно мониторить выполнение заданий;

– выполнять подзадания;

– присылать отчеты об исключениях на email.

В очереди может быть четыре разных задания. Все задания периодические. Все задания запущены одни и тем же процессом, т.к. необходимо, чтобы они использовали совместные ресурсы (так как выбрана модель сохранения промежуточных данных в оперативную память). Задания выполняются асинхронно.

Первое задание отвечает за обновление текущих матчей и сбор статистики о матче в промежутке 10-20 минут с начала матча. Статистика записывается в переменную типа словарь, где ключом является id матча. Оно запускается каждые десять минут.

Второе и третье задания отвечают за добавление к статистике матча количества голов, забитых командами за первый тайм и за всю игру соответственно. Для каждого из заданий есть специальный словарь, в котором хранятся все матчи, которые оно должно просмотреть. Как только задания обработали все эти матчи, то происходит запись задания в базу данных. Эти задания выполняются один раз в десять минут.

Четвёртое задание отвечает за обновление переменных-словарей, которые используются вторым и третьим заданиями. Оно просматривает все матчи, и как только оно определяет, что закончился первый или второй тайм, то она добавляет соответствующий матч для просмотра вторым или третьим заданием. Это задание выполняется один раз в пять минут.

Таким образом, осуществлена задача поиска в режиме реального времени таких матчей, которые недавно начались, и задача сопровождения этих матчей до конца игры со сбором промежуточных статистик.

# 2 Система анализа данных

### 2.1 Инструменты анализа данных

Одним из наиболее популярных языков программирования для анализа данных является Python. Для него создано большое количество библиотек и фреймворков. В работе используется интерактивная оболочка IPython Jupyter Notebook. Данная оболочка поддерживает встроенный режим работы с самой популярной библиотекой визуализации для Python matplotlib. Также в ней присутствует функция разбиения исходного кода на независимые блоки, вследствие чего появляется возможность запуска программы “по частям”. Это является очень полезной функцией, потому что можно, например, один раз обучить некоторую модель в одном блоке кода, а анализ и тестирование её вывести в другие блоки, которые можно изменять, перезапускать, вручную менять последовательность их исполнения. Этот функционал получилось реализовать, потому что Python является интерпретируемым языком (т.е. исполняется построчно).

Для визуализации в работе используются библиотеки matplotlib и seaborn.

Хранение данных осуществляется в оперативной памяти с помощью библиотеки pandas. Pandas — программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Pandas предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами. Название библиотеки происходит от эконометрического термина «панельные данные», используемого для описания многомерных структурированных наборов информации. Основные возможности библиотеки:

– объект DataFrame для манипулирования индексированными массивами двумерных данных;

– инструменты для обмена данными между структурами в памяти и файлами различных форматов;

– встроенные средства совмещения данных и способы обработки отсутствующей информации;

– переформатирование наборов данных, в том числе создание сводных таблиц;

– срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных;

– вставка и удаление столбцов данных;

– возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение»;

– слияние и объединение наборов данных;

– иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности;

– работа с временными рядами: формирование временных периодов и изменение интервалов и т. д.

Также в работе используется библиотека для градиентного бустинга на деревьях eXtreme Gradient Boosting. С её помощью можно решать задачи классификации и регрессии. Также у неё есть инструменты для определения информативных признаков [2,3].

### 2.2 Визуальный анализ данных

Прежде чем работать с данными, необходимо как можно глубже понять их природу. Для этого часто используется визуальный анализ. Визуализация производится с помощью библиотеки seaborn.

Прежде чем строить графики и анализировать результат, необходимо знать, как кодируется каждый признак. Ниже представлена расшифровка признаков.

– goals – количество голов, забитых командой;

– corners – количество угловых;

– ycard – количество жёлтых карточек;

– rcard – количество красных карточек;

– penalt – количество пенальти;

– attack – количество атак;

– dangattack – количество опасных атак;

– ontarget – количество ударов в створ ворот;

– offtarget – количество ударов мимо ворот;

– possession – процент владения мячом домашней команды.

Все признаки кроме possession имеют в названии цифру 1 или 2, которые обозначают, к какой команде относится эта статистика. Если стоит цифра 1, то статистика для домашней команды, если цифра 2 – то для команды, играющей в гостях. Например, признак goals2 трактуется как количество голов, забитых до 15 минуты игры командой, играющей в гостях.

На рисунке 3 приведена тепловая карта для матрицы корреляций всех пар признаков. Оранжевыми точками обозначены матчи с количеством голов домашней команды большим нуля. Синими точками обозначены матчи, в которых домашняя команда ни разу не забила. По диагонали расположены распределения признаков.

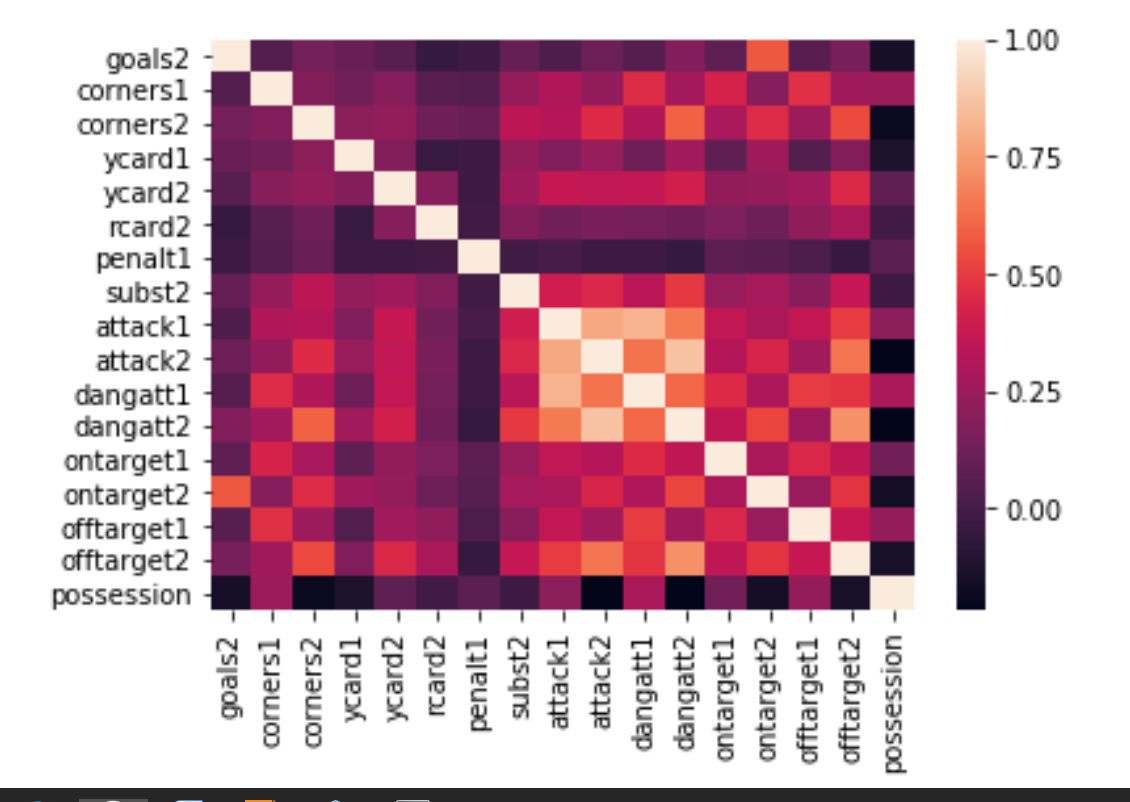


Рисунок 3 – Тепловая карта для матрицы корреляции признаков

Анализируя эту тепловую карту, можно сделать несколько наблюдений.

Признаки количества атак и опасных атак имеют сильную корреляцию. Это обусловлено тем, что много матчей начинается в одно и то же время (может идти в среднем до 15 матчей одновременно), а просмотреть все матчи сразу нельзя. На обработку одного матча уходит около 20 секунд. Следовательно, матчи, которые просмотрены последними, идут существенно больше времени, чем матчи, просмотренные в самом начале.

Ожидаемо высокие коэффициенты корреляции между количеством ударов в створ ворот у гостевой команды и количеством забитых ей мячей, между владением мячом и количеством атак и опасных атак у обеих команд. В целом собранная выборка очень хорошо описывает связь параметров между собой. Все существенные корреляции имеют логическое обоснование.

Ещё одним информативным видом графиков является график boxplot. На рисунке 4 изображён этот тип графиков. Синим цветом выделено распределение матчей, в зависимости от рассматриваемого признака, когда домашняя команда не забила ни одного гола. Зелёным цветом выделено распределение матчей, в зависимости от рассматриваемого признака, когда домашняя команда забила хотя бы один гол.

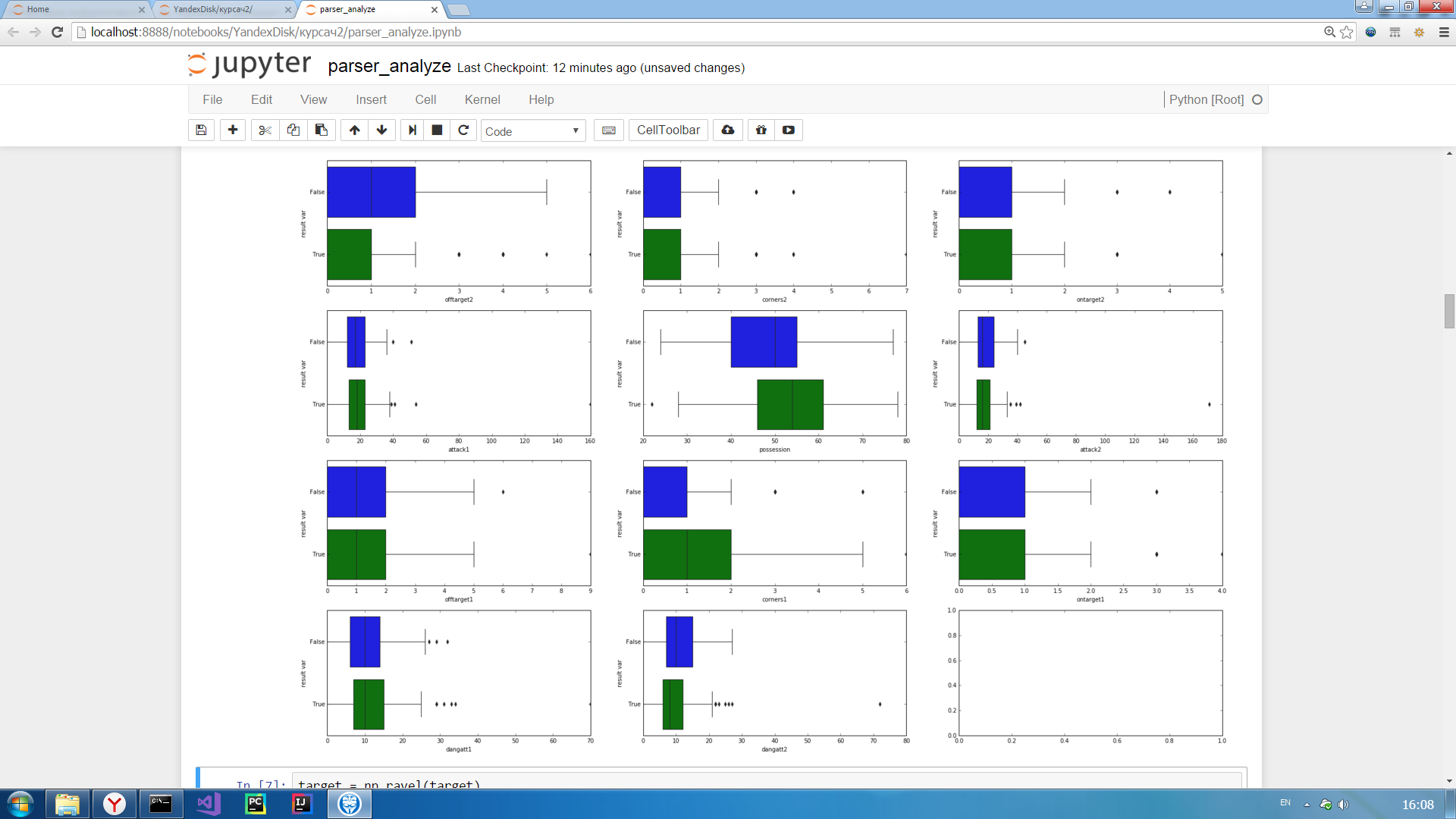


Рисунок 4 – Графики boxplot

Из показанных графиков видно, что если домашняя команда забивает голы, то в среднем она уже на 15 минуте больше владеет мячом. Также хорошо заметно, что если у домашней команды много угловых на 15 минуте игры, то по статистике она забьёт хотя бы один гол.

### 2.3 Постановка задачи на прогнозирование результата матча

Вероятнее всего, бессмысленно ставить задачу точного предсказания счёта всего матча, зная статистику только первых 15 минут игры. Поэтому необходимо выбрать такой критерий, который бы, с одной стороны, как можно более точно описывал результат матча, а с другой стороны был бы истинным с наибольшей долей вероятности. Рассмотрим два критерия.

Первый критерий будет предсказывать количество голов, забитых командой, находящейся дома. Второй критерий будет предсказывать вероятность того, что первая домашняя команда забьёт хотя бы один гол. Очевидно, что первый критерий более информативен, однако интуитивно легче предсказать второй критерий.

В качестве задачи прогнозирования могут выступать задачи регрессии и классификации на исходном наборе данных. Рассмотрим особенности разных подходов к прогнозированию.

Для прогнозирования второго критерия самым очевидным выбором является решение задачи бинарной классификации. Для прогнозирования первого критерия можно использовать как классификацию, так и регрессию. Может показаться, что для предсказания этого критерия лучше подойдёт решение задачи регрессии, так как множество количества голов является числовым упорядоченным множеством. Но с другой стороны в результате решения будет получено вещественное число, которое надо как-то интерпретировать (желательнее всего получить вероятность того, что результатом будет целое число либо меньшее этого результата, либо большее результата). Из-за сложности задачи интерпретации гораздо легче представить целевой признак в виде классов для задачи классификации и решать именно эту задачу, так как в этой задаче обычно не сложно вывести вероятности причисления предсказываемого объекта к каждому из классов.

Таким образом, для предсказания обоих критериев было решено использовать решение задачи классификации.

### 2.4 Формальная постановка задачи классификации

Пусть X – множество описаний объектов, Y – конечное множество меток классов. Существует неизвестная целевая зависимость — отображение , значения которой известны только на объектах конечной обучающей выборки Требуется построить алгоритм , способный классифицировать произвольный объект .

Существуют и различные виды задачи:

1) двухклассовая классификация. Наиболее простой в техническом отношении случай, который служит основой для решения более сложных задач.

2) Многоклассовая классификация. Когда число классов достигает многих тысяч (например, при распознавании иероглифов или слитной речи), задача классификации становится существенно более трудной. Задача многоклассовой классификации может быть сведена к последовательности задач двухклассовой классификации [4, 5].

### 2.5 Общие сведения о градиентном бустинге на решающих деревьях

Решающие деревья воспроизводят логические схемы, позволяющие получить окончательное решение о классификации объекта с помощью ответов на иерархически организованную систему вопросов. Причём вопрос, задаваемый на последующем иерархическом уровне, зависит от ответа, полученного на предыдущем уровне. Дерево является простой моделью, склонной к переобучению. Из-за этого на практике используются ансамбли деревьев.

Существует две широко используемые стратегии обучения деревьев: решающий лес и градиентный бустинг. Решающий лес обучается по следующему алгоритму:

а) случайно выбирается некоторое количество признаков и объектов;

б) происходит обучение одного дерева на выборке из пункта a;

в) пока количество деревьев не станет равным N, выполняются пункты a-б. N - гиперпараметр алгоритма, отвечающий за конечное количество деревьев в лесу.

Градиентный бустинг работает по другому принципу. Следующее дерево строится с помощью предыдущей модели так, чтобы увеличить точность предсказания. Такая стратегия иначе называется аддитивным обучением:

(1)

где t – количество этапов обучения (или количество деревьев),– предсказанное значение, – i-ое решающее дерево. Целевая функция имеет вид:

, (2)

где – функция ошибки, – параметр регуляризации. Функция ошибки обязательно должна быть дифференцируемой, так как при обучении модели решается задача оптимизации функции lметодом градиентного спуска [3].

### 2.6 Подбор гиперпараметров и обучение модели

Гиперпараметры – настройки алгоритма, решающего задачу. В случае с градиентным бустингом на решающих деревьях гиперпараметрами являются:

– максимальная глубина одного дерева;

– коэффициент регуляризации;

– количество деревьев в модели;

– шаг обучения (learning rate);

– минимальный порог разбиения выборки;

– целевая функция и другие.

Наиболее простой способ подбора гиперпараметров – перебор по заданной сетке. Этот подход называется GridSearch. Достоинства метода в том, что если задать плотную сетку, то можно получить хорошую модель на выходе. Очевидным минусом является большой объём вычислений. Существует много вариаций поиска по сетке без полного перебора. Можно, например, использовать подход случайного выбора параметров сетки или стохастического градиентного спуска [6]. Однако стоит отметить, что для цели курсовой работы подойдёт и обычный поиск по сетке, так как выборка не очень большая и, следовательно, вычислительных ресурсов для обучения модели хватает даже у ноутбука.

На рисунке 5 показан процесс задания сетки перебора параметров, поиска оптимальных параметров с помощью инструмента GridSearchCV и результат этого поиска.

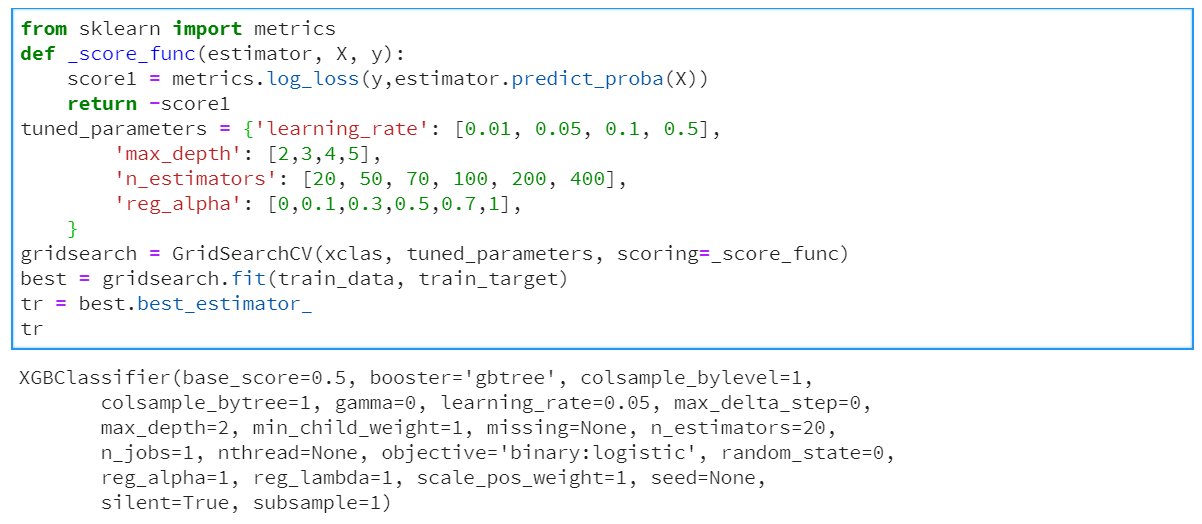


Рисунок 5 – поиск оптимальных значений гиперпараметров

На рисунке 6 показано обучение модели, нахождение вероятностей принадлежности для всех объектов из тестовой выборки к каждому из классов и нахождение максимума из этих вероятностей. Тестовая выборка имеет размер 15% от всей выборки.

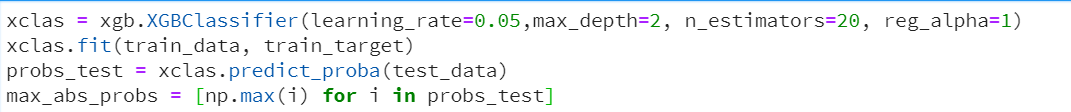


Рисунок 6 – процесс обучения модели и получения предсказания для тестовой выборки

На рисунке 7 изображена плотность распределения списка max\_abs\_probs, в котором хранятся максимумы среди вероятностей для каждого класса для всех объектов тестовой выборки.

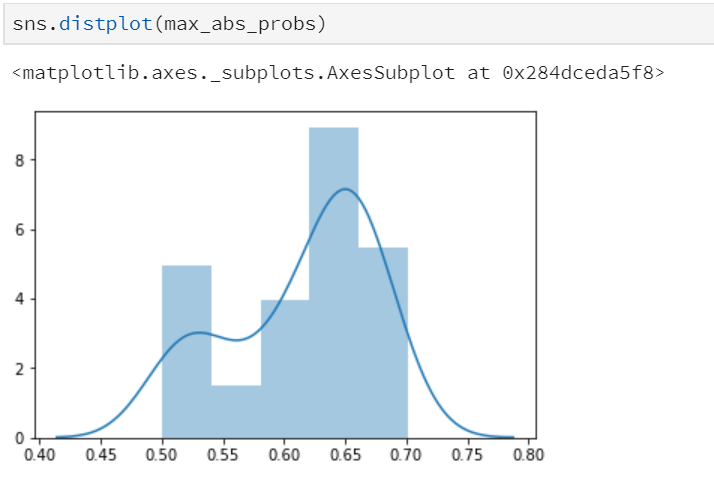


Рисунок 7 – плотность распределения списка max\_abs\_probs

### 2.7 Оценка качества модели с помощью кросс-валидации

Оценка точности модели по одной тестовой выборке не является достаточно точной. Поэтому для более надёжной оценки должна использоваться одна из стратегий кросс-валидации. Так как исходный массив данных не упорядочен по времени (то есть не является временным рядом) и доли классов примерно равны, то можно использовать простую стратегию KFold. Её суть заключается в том, что выборка разбивается на k равных частей. На k-1 части осуществляется обучение модели, на оставшейся части проводится тестирование модели. Процесс реализуется таким способом, что каждая часть должна один раз быть в качестве тестовой. Результаты работы модели на тестовых выборках усредняются. Из-за того, что модели тестировались и обучались на разных выборках точность и устойчивость модели может быть определена более точно, чем при одном тестировании модели.

# 3 Анализ полученных результатов

### 3.1 Анализ системы в разрезе точности критериев оценки результатов матча

Первый критерий предсказывает количество голов, забитых командой, находящейся дома. Очевидно, что критерий может быть переопределён и для команды, находящейся в гостях.

На рисунке 8 показан процесс кросс-валидации и точность модели

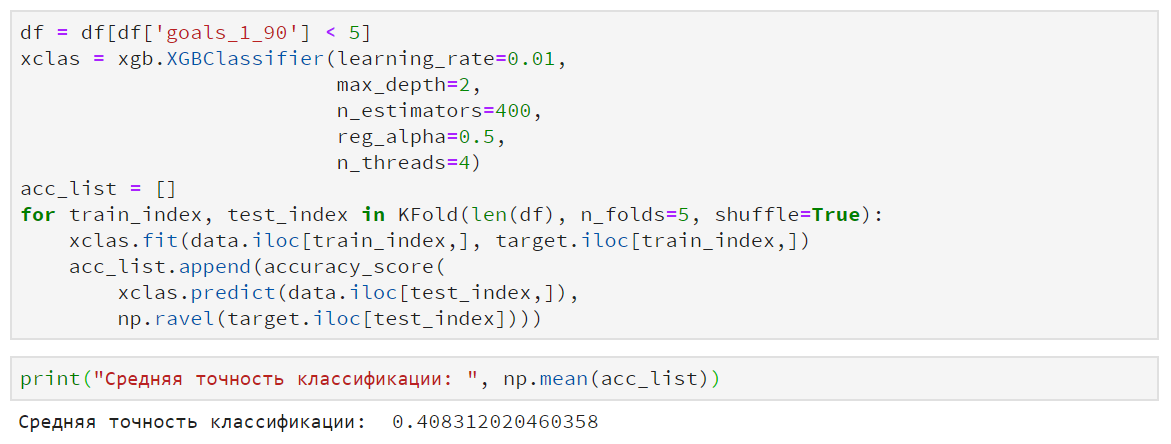


Рисунок 8 – проведение стратегии KFold для первого критерия

Предположение, сделанное в пункте 2.3, оправдалось. Модель не смогла найти сильных зависимостей между переменными и точно предсказывать результат матча (точность предсказания равняется 40%). Однако стоит отметить, что данная задача является задачей многоклассовой классификации с числом классов равным пяти. Значит, точность модели в два раза превышает точность слепого угадывания класса, что является неплохим показателем для такой вариативной игры как футбол.

Рассмотрим теперь более слабый второй критерий, который предсказывает, забьёт ли домашняя команда хотя бы один гол или нет. На рисунке 9 показан процесс кросс-валидации и точность модели для второго критерия.

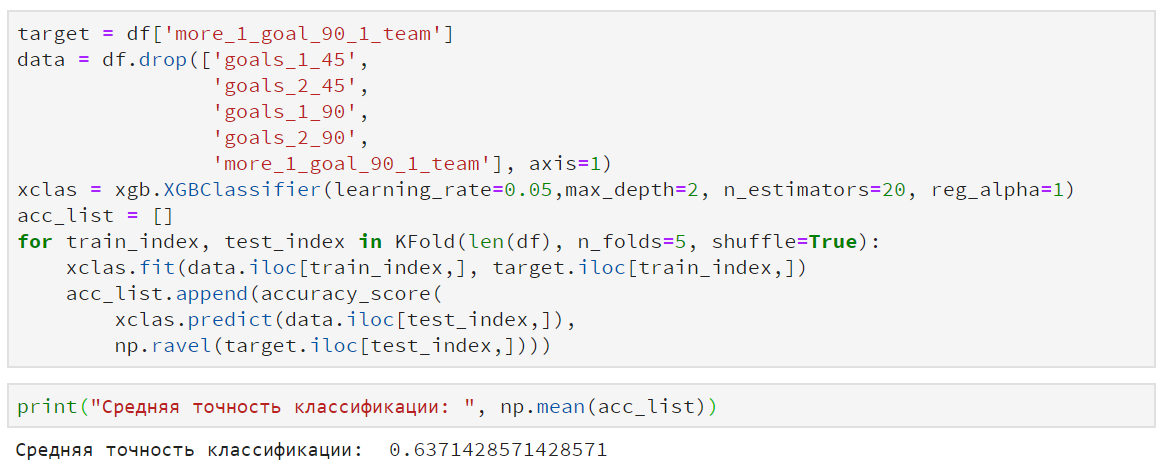


Рисунок 9 – проведение стратегии KFold для второго критерия

Классификатор смог предсказать 63% матчей. Этот критерий, как и говорилось в пункте 2.3, легче предсказать. Стоит отметить, что сложность модели для второго критерия существенно меньше, чем для первого. Это можно увидеть, например, по оптимальному количеству деревьев. Для первого критерия необходимо было 400 деревьев, а для второго всего 20.

### 3.2 Анализ системы в разрезе "уверенности" модели в своей классификации

Важным моментом для поставленной задачи является "степень уверенности" классификатора в своей правоте. В качестве "степени уверенности" можно считать долю наибольшего класса в листе дерева, куда спустился классифицируемый объект. Рассмотрим распределение этой величины для тестовой выборки для каждого критерия и проанализируем зависимость точности и "уверенности" в предсказании. Хорошим сигналом будет вывод о том, что значения средней точности и средней "уверенности" по всей тестовой выборке достаточно похожи.

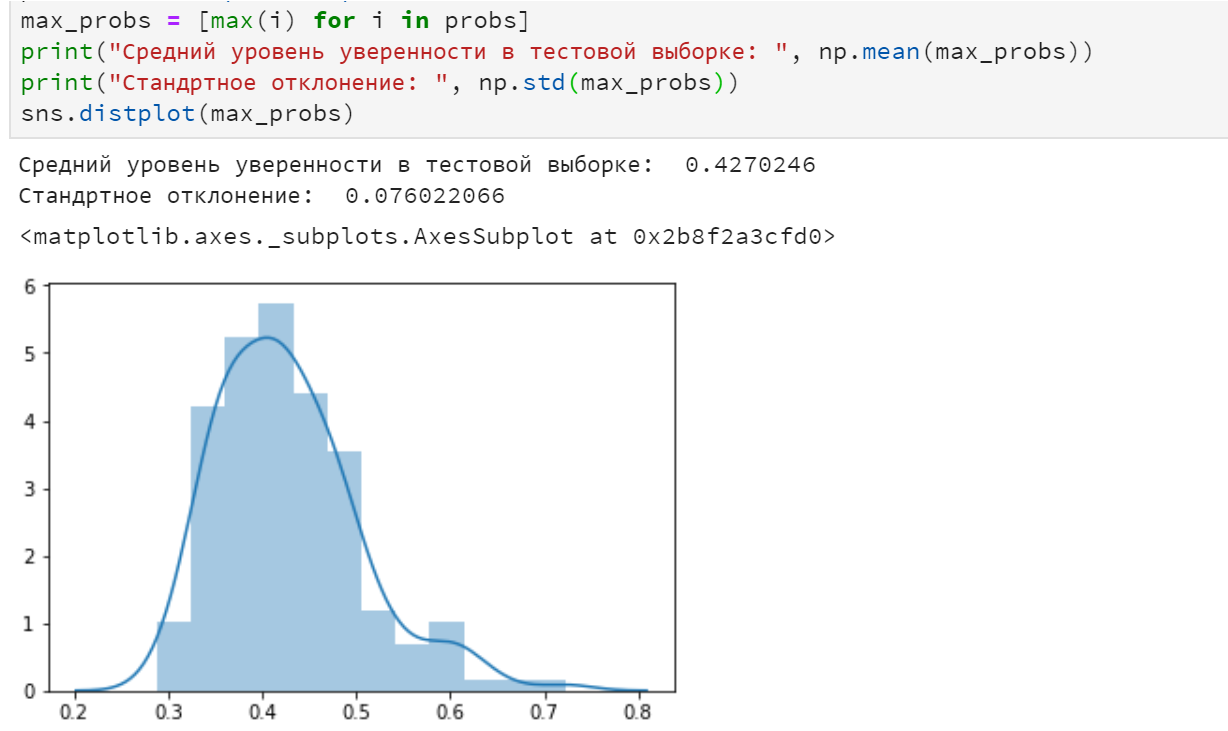


Рисунок 10 – распределение степени "уверенности" классификатора для первого критерия оценки результатов матча.

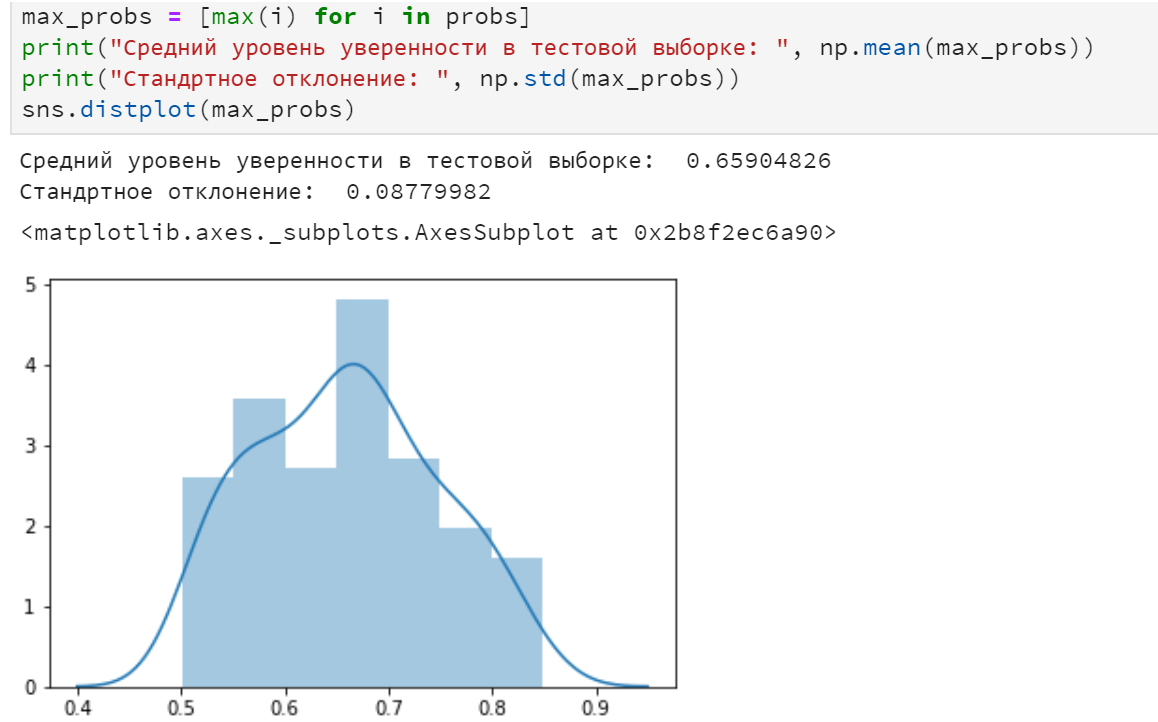


Рисунок 11 – распределение степени "уверенности" классификатора для второго критерия оценки результатов матча.

Как видно из графиков, среднее значение точности очень близко к среднему значению "уверенности" модели в классификации. Это означает, что модель не переобучилась и степень "уверенности" модели адекватна входным данным. Другими словами, если модель "не уверена" в классификации, то это можно увидеть по примерно равным вероятностям, которые она даёт для каждого класса. И наоборот, если модель "уверена" в классификации, то будет существовать класс, у которого вероятность намного больше всех остальных.

# ЗАКЛЮЧЕНИЕ

В ходе работы была реализована система сбора данных о футбольных матчах в реальном времени и система анализа этих данных. Полученные результаты достаточно хорошо могут предсказывать результат всего матча по первым 15 минутам игры. Однако существует предположение, что обе системы можно существенно улучшить. Систему сбора данных можно реализовать с помощью методов распределённого программирования, чтобы повысить устойчивость работы. В системе анализа данных можно попытаться найти новые признаки из существующих и новые взаимосвязи между признаками с помощью многослойных искусственных нейронных сетей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Документация по работе с библиотекой Celery. [Электронный ресурс]. – URL: http://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html.

2 Статья «XGBoost: A Scalable Tree Boosting System». 2016 [Электронный ресурс]. – URL: https://arxiv.org/abs/1603.02754.

3 Статья «Introduction to Boosted Trees». 2015 [Электронный ресурс]. – URL: http://xgboost.readthedocs.io/en/latest/model.html.

4 Дьяконов, A. Г. Анализ данных, обучение по прецедентам, логические игры, системы WEKA, RapidMiner и MatLab (практикум на ЭВМ кафедры математических методов прогнозирования). — МАКСПресс, 2010. — 278 с.

5 Онлайн-курс «Обучение на размеченных данных». [Электронный ресурс]. – URL: <https://www.coursera.org/learn/supervised-learning/home/welcome> (дата обращения: 10.03.2018).

6 Видеолекции курса «Машинное обучение». [Электронный ресурс]. – URL: https://yandexdataschool.ru/edu-process/courses/machine-learning (дата обращения 27.04.2018).

# ПРИЛОЖЕНИЕ

# Листинг программы

Файл celeryconfig.py:

broker\_url = 'pyamqp://'

result\_backend = 'rpc://'

task\_serializer = 'json'

result\_serializer = 'json'

accept\_content = ['json']

enable\_utc = True

beat\_schedule = {

'find-new-games-10-20': {

'task': 'tasks.get\_games\_between\_10\_20\_min',

'schedule': 600.0,

},

'update\_45': {

'task': 'tasks.process\_45\_min\_dict',

'schedule': 300.0,

},

'update\_90': {

'task': 'tasks.process\_90\_min\_dict',

'schedule': 300.0,

},

'update\_global': {

'task': 'tasks.process\_global\_dict',

'schedule': 600.0,

}

}

Файл src.py:

import asyncio

import datetime

import requests

from bs4 import BeautifulSoup as bs

import time

import pandas as pd

from collections import defaultdict

import sqlite3

from time\_consts import END\_1\_TIME\_MIN, END\_GAME\_MIN

SOCCER\_URL = "https://betsapi.com/ci/soccer"

BASE\_URL = 'https://betsapi.com'

doubled\_stats = ['Possession %', 'Off Target', 'On Target', 'Dangerous Attacks', 'Attacks']

def get\_goals(game\_url: str):

html = get\_html(game\_url)

soup = bs(html, 'lxml')

game\_goals = soup.find('span', class\_='red-color').text.strip()

return tuple(map(int, game\_goals.split('-')))

def get\_html(url):

r = requests.get(url)

return r.text

def get\_html\_football():

return get\_html(SOCCER\_URL)

def get\_games\_10\_20\_list\_links(html):

soup = bs(html, 'lxml')

games = soup.find\_all('tr', class\_='c\_1')

goals = [(BASE\_URL + game.find('td', class\_='text-center').find('a').get('href'),

game.find('td', class\_='text-center').find('a').get('href')) for game in games if

10 <= int(game.find('span', class\_='race-time').text.strip()[:2]) <= 20]

return goals

def save\_finally\_stats(stats\_dict: dict):

goals1 = stats\_dict['Goals'][0]

goals2 = stats\_dict['Goals'][1]

corners1 = stats\_dict['Corners'][0]

corners2 = stats\_dict['Corners'][1]

ycard1 = stats\_dict['Yellow Card'][0]

ycard12 = stats\_dict['Yellow Card'][1]

rcard1 = stats\_dict['Red Card'][0]

rcard2 = stats\_dict['Red Card'][1]

penalt1 = stats\_dict['Penalties'][0]

penalt2 = stats\_dict['Penalties'][1]

subst1 = stats\_dict['Substitutions'][0]

subst2 = stats\_dict['Substitutions'][1]

attack1 = stats\_dict['Attacks'][0]

attack2 = stats\_dict['Attacks'][1]

dangatt1 = stats\_dict['Dangerous Attacks'][0]

dangatt2 = stats\_dict['Dangerous Attacks'][1]

ontarget1 = stats\_dict['On Target'][0]

ontarget2 = stats\_dict['On Target'][1]

offtarget1 = stats\_dict['Off Target'][0]

offtarget2 = stats\_dict['Off Target'][1]

possession = stats\_dict['Possession %'][0]

goals1\_45 = stats\_dict['goals\_45'][0]

goals2\_45 = stats\_dict['goals\_45'][1]

goals1\_90 = stats\_dict['goals\_90'][0]

goals2\_90 = stats\_dict['goals\_90'][1]

conn = sqlite3.connect('db')

cur = conn.cursor()

cur.execute('INSERT into stats values (?, ?, ?, ?, ?, ?, ?, ?,'

' ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',

(

goals1,

goals2,

corners1,

corners2,

ycard1,

ycard12,

rcard1,

rcard2,

penalt1,

penalt2,

subst1,

subst2,

attack1,

attack2,

dangatt1,

dangatt2,

ontarget1,

ontarget2,

offtarget1,

offtarget2,

possession,

goals1\_45,

goals2\_45,

goals1\_90,

goals2\_90

))

conn.commit()

cur.close()

conn.close()

def parse\_game(link: str):

game\_html = get\_html(link)

soup = bs(game\_html, 'lxml')

minute = int(soup.find('span', class\_='race-time').text.strip()[:-1])

print(minute)

if minute:

df\_list = pd.read\_html(game\_html) # read all tables

df\_stat = pd.DataFrame(df\_list[0]) # first table is stats

team1, stat, team2 = df\_stat[[0]], df\_stat[[1]], df\_stat[[2]] # 3 columns

stats\_dict = defaultdict(lambda: (0, 0))

for i, (team1, stat, team2) in df\_stat.iterrows():

print(team1, stat, team2)

stats\_dict[str(stat)] = (str(team1), str(team2))

# key in stats\_dict is name of statistic

for key in stats\_dict:

if key in doubled\_stats:

stats\_dict[key] = (stats\_dict[key][0].split()[0], stats\_dict[key][1].split()[0])

del stats\_dict['nan']

for key in stats\_dict:

stats\_dict[key] = (int(stats\_dict[key][0]), int(stats\_dict[key][1]))

return [datetime.datetime.now() + datetime.timedelta(END\_1\_TIME\_MIN - minute),

datetime.datetime.now() + datetime.timedelta(END\_GAME\_MIN - minute),

stats\_dict]

async def filter\_links(links: list) -> list:

filtered\_links = []

print('filter start: ' + str(datetime.datetime.now()))

for link in links:

game\_html = get\_html(link)

soup = bs(game\_html, 'lxml')

minute = int(soup.find('span', class\_='race-time').text.strip()[:2])

if 10 <= minute <= 20:

filtered\_links.append(link)

await asyncio.sleep(8)

print('filter end: ' + str(datetime.datetime.now()))

return filtered\_links

def process\_link\_for\_bot(link: str) -> bool:

game\_html = get\_html(link)

soup = bs(game\_html, 'lxml')

minute = int(soup.find('span', class\_='race-time').text.strip()[:2])

print(minute)

if 10 < minute < 20:

df\_list = pd.read\_html(game\_html)

df\_stat = pd.DataFrame(df\_list[0])

team1, stat, team2 = df\_stat[[0]], df\_stat[[1]], df\_stat[[2]]

temp\_dict = defaultdict(lambda: (0, 0))

for i, (team1, stat, team2) in df\_stat.iterrows():

print(team1, stat, team2)

temp\_dict[str(stat)] = (str(team1), str(team2))

for key in temp\_dict:

if key in doubled\_stats:

temp\_dict[key] = (temp\_dict[key][0].split()[0], temp\_dict[key][1].split()[0])

del temp\_dict['nan']

for key in temp\_dict:

temp\_dict[key] = (int(temp\_dict[key][0]), int(temp\_dict[key][1]))

goals1 = temp\_dict['Goals'][0]

goals2 = temp\_dict['Goals'][1]

corners1 = temp\_dict['Corners'][0]

corners2 = temp\_dict['Corners'][1]

ycard1 = temp\_dict['Yellow Card'][0]

ycard12 = temp\_dict['Yellow Card'][1]

rcard1 = temp\_dict['Red Card'][0]

rcard2 = temp\_dict['Red Card'][1]

penalt1 = temp\_dict['Penalties'][0]

penalt2 = temp\_dict['Penalties'][1]

subst1 = temp\_dict['Substitutions'][0]

subst2 = temp\_dict['Substitutions'][1]

attack1 = temp\_dict['Attacks'][0]

attack2 = temp\_dict['Attacks'][1]

dangatt1 = temp\_dict['Dangerous Attacks'][0]

dangatt2 = temp\_dict['Dangerous Attacks'][1]

ontarget1 = temp\_dict['On Target'][0]

ontarget2 = temp\_dict['On Target'][1]

offtarget1 = temp\_dict['Off Target'][0]

offtarget2 = temp\_dict['Off Target'][1]

possession = temp\_dict['Possession %'][0]

pos1\_ = possession >= 60

attack1\_ = (attack1 - attack2) \* 2

dangatt1\_ = (dangatt1 - dangatt2) \* 4

ontarget1\_ = (ontarget1 - ontarget2) \* 7

offtarget1\_ = (offtarget1 - offtarget2) \* 5

corners1\_ = (corners1 - corners2) \* 5

res1 = attack1\_ + dangatt1\_ + ontarget1\_ + offtarget1\_ + corners1\_

if pos1\_ and res1 > 80:

return True

pos2\_ = possession <= 40

attack2\_ = (attack2 - attack1) \* 2

dangatt2\_ = (dangatt2 - dangatt1) \* 4

ontarget2\_ = (ontarget2 - ontarget1) \* 7

offtarget2\_ = (offtarget2 - offtarget1) \* 5

corners2\_ = (corners2 - corners1) \* 5

res2 = attack2\_ + dangatt2\_ + ontarget2\_ + offtarget2\_ + corners2\_

if pos2\_ and res2 > 80:

return True

return False

return False

def get\_stats(links):

for link in links:

# game\_html = get\_html(link)

# soup = bs(game\_html, 'lxml')

parse\_game(link)

time.sleep(1)

def main():

html = get\_html(SOCCER\_URL)

games\_links = get\_games\_10\_20\_list\_links(html)

# test\_link = [games\_links[10]]

print(games\_links)

get\_stats(games\_links)

if \_\_name\_\_ == '\_\_main\_\_':

main()

файл tasks.py:

from celery import Celery

import src

import time

from time\_consts import \*

from datetime import datetime, timedelta

import copy

import logging

logging.basicConfig(level=logging.DEBUG, filename='log.log')

app = Celery()

app.config\_from\_object('celeryconfig')

GLOBAL\_GAMES\_DICT = {}

GLOBAL\_45\_MIN\_GAMES = {}

GLOBAL\_90\_MIN\_GAMES = {}

@app.task

def get\_games\_between\_10\_20\_min():

html = src.get\_html\_football()

games = src.get\_games\_10\_20\_list\_links(html)

for game\_link, game\_rel\_link in games:

GLOBAL\_GAMES\_DICT[game\_rel\_link] = src.parse\_game(game\_link)

# Add info to global\_dict. key - game id, value - stats and time 45', 90'.

time.sleep(DELAY\_BETWEEN\_GAMES\_SEC)

logging.info(datetime.now())

print(GLOBAL\_GAMES\_DICT)

@app.task

def process\_global\_dict():

local\_dict = copy.deepcopy(GLOBAL\_GAMES\_DICT)

for game\_id, [time\_45, time\_90, game\_stats] in local\_dict.items():

if time\_45 is not None and time\_45 <= datetime.now() <= time\_45 + timedelta(minutes=2):

GLOBAL\_45\_MIN\_GAMES[game\_id] = game\_stats

if time\_90 is not None and time\_90 <= datetime.now():

GLOBAL\_90\_MIN\_GAMES[game\_id] = game\_stats

del GLOBAL\_GAMES\_DICT[game\_id]

@app.task

def process\_45\_min\_dict():

for game\_rel\_link, game\_stats in GLOBAL\_45\_MIN\_GAMES.items():

goals1, goals2 = src.get\_goals(src.BASE\_URL + game\_rel\_link)

game\_stats['goals\_45'] = (goals1, goals2)

GLOBAL\_GAMES\_DICT[game\_rel\_link][2] = copy.deepcopy(game\_stats)

GLOBAL\_GAMES\_DICT[game\_rel\_link][0] = None

del GLOBAL\_45\_MIN\_GAMES[game\_rel\_link]

time.sleep(DELAY\_BETWEEN\_GAMES\_SEC)

@app.task

def process\_90\_min\_dict():

for game\_rel\_link, game\_stats in GLOBAL\_90\_MIN\_GAMES.items():

goals1, goals2 = src.get\_goals(src.BASE\_URL + game\_rel\_link)

game\_stats['goals\_90'] = (goals1, goals2)

# update db

src.save\_finally\_stats(game\_stats)

del GLOBAL\_90\_MIN\_GAMES[game\_rel\_link]

time.sleep(DELAY\_BETWEEN\_GAMES\_SEC)

файл time\_consts:

TIMER\_UPDATE\_MAIN\_TABLE\_MIN = 10

DELAY\_BETWEEN\_GAMES\_SEC = 10

END\_1\_TIME\_MIN = 45

END\_GAME\_MIN = 90

Файл parser\_analyze:

%matplotlib inline

from matplotlib import pyplot as plt

import sqlite3

import pandas as pd

import numpy as np

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import euclidean\_distances,\

mean\_squared\_error,\

mean\_absolute\_error,\

accuracy\_score

from sklearn.cross\_validation import KFold, LeaveOneOut

from sklearn.grid\_search import GridSearchCV

import seaborn as sns

import xgboost as xgb

conn = sqlite3.connect('db')

df = pd.read\_sql('SELECT \* FROM stats', conn)

df = df.query('possession>0 and goals1<1 and offtarget2<10')

df['more\_1\_goal\_90\_1\_team'] = df.iloc[:,-2] >= 1

target = df[['more\_1\_goal\_90\_1\_team']]

data = df.drop(['goals\_1\_45',

'goals\_2\_45',

'goals\_1\_90',

'goals\_2\_90',

'goals1',

'penalt2',

'rcard1',

'subst1',

'more\_1\_goal\_90\_1\_team'], axis=1)

corr\_matrix = data.corr()

sns.heatmap(corr\_matrix, cmap="YlGnBu")

features = list(data.columns)

sns.pairplot(df[features + ['more\_1\_goal\_90\_1\_team']], hue='more\_1\_goal\_90\_1\_team')

# Boxplot features/target. (a-la 1 kursach)

rows = 6

cols = 3

fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize=(28, 18))

features = list(set(data.columns) - set(['more\_1\_goal\_90\_1\_team', 'penalt1', 'ycard1', 'rcard2', 'subst2']))

print(features, len(features))

for idx, feat in enumerate(features):

sns.boxplot(y='more\_1\_goal\_90\_1\_team', x=feat, data=df, ax=axes[idx // cols, idx % cols], orient='h')

# axes[idx // 4, idx % 4].legend()

axes[idx // cols, idx % cols].set\_xlabel('res')

axes[idx // cols, idx % cols].set\_ylabel(feat)

target = np.ravel(target)

xclas = xgb.XGBClassifier(learning\_rate=0.05, max\_depth=2, n\_estimators=300)

# ?GridSearchCV

xclas.fit(data, target)

# xgb.plot\_importance(xclas, height=0.7)

from sklearn import metrics

def \_score\_func(estimator, X, y):

score1 = metrics.log\_loss(y,estimator.predict\_proba(X))

return -score1

tuned\_parameters = {'learning\_rate': [0.01, 0.05, 0.1, 0.5],

'max\_depth': [2,3,4,5],

'n\_estimators': [20, 50, 70, 100, 200, 400],

'reg\_alpha': [0,0.1,0.3,0.5,0.7,1],

}

xclas = xgb.XGBClassifier(learning\_rate=0.05,

max\_depth=2,

n\_estimators=20,

reg\_alpha=1,

n\_threads=4)

target = df[df['goals\_1\_90'] < 5]['goals\_1\_90']

data = df[df['goals\_1\_90'] < 5].drop(['goals\_1\_45',

'goals\_2\_45',

'goals\_1\_90',

'goals\_2\_90',

'more\_1\_goal\_90\_1\_team'], axis=1)

train\_data = data.iloc[:180,]

train\_target = target.iloc[:180]

test\_data = data.iloc[180:,]

test\_target = target.iloc[180:,]

print(train\_data.shape)

print(test\_data.shape)

print(train\_target.shape)

print(test\_target.shape)

gridsearch = GridSearchCV(xclas, tuned\_parameters, scoring=\_score\_func)

best = gridsearch.fit(train\_data, train\_target)

tr = best.best\_estimator\_

print(tr)

df = df[df['goals\_1\_90'] < 5]

xclas = xgb.XGBClassifier(learning\_rate=0.01,

max\_depth=2,

n\_estimators=400,

reg\_alpha=0.5,

n\_threads=4)

acc\_list = []

for train\_index, test\_index in KFold(len(df), n\_folds=5, shuffle=True):

xclas.fit(data.iloc[train\_index,], target.iloc[train\_index,])

acc\_list.append(accuracy\_score(

xclas.predict(data.iloc[test\_index,]),

np.ravel(target.iloc[test\_index])))

target = df['more\_1\_goal\_90\_1\_team']

data = df.drop(['goals\_1\_45',

'goals\_2\_45',

'goals\_1\_90',

'goals\_2\_90',

'more\_1\_goal\_90\_1\_team'], axis=1)

xclas = xgb.XGBClassifier(learning\_rate=0.03,max\_depth=2, n\_estimators=200, reg\_alpha=1)

acc\_list = []

for train\_index, test\_index in KFold(len(df), n\_folds=5, shuffle=True):

xclas.fit(data.iloc[train\_index,], target.iloc[train\_index,])

acc\_list.append(accuracy\_score(

xclas.predict(data.iloc[test\_index,]),

np.ravel(target.iloc[test\_index,])))

probs = xclas.predict\_proba(test\_data)

max\_probs = [max(i) for i in probs]

print("Средний уровень уверенности в тестовой выборке: ", np.mean(max\_probs))

print("Стандртное отклонение: ", np.std(max\_probs))

sns.distplot(max\_probs)

print("Средняя точность классификации: ", np.mean(acc\_list))