МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный университет им. П.Г. Демидова»

Кафедра компьютерных сетей

Сдано на кафедру

«14» июня 2020 г.

Заведующий кафедрой

доктор ф.-м. наук, профессор

(степень, звание)

__ __

Глызин С.Д..

(подпись)

(ФИО)

Машинное обучение в анализе эффективности игроков в командных видах спорта

(направление подготовки 01.03.02 «Прикладная математика и информатика»)

Научный руководитель

_____к.ф.-м.н._____

(степень, звание)

___ __

Богомолов Ю.В.

(подпись) *(ФИО)*

«14» июня 2020 г.

Студент группы ИВТ-42(БО)

_ Кононец Д.А.

(подпись) *(ФИО)*

«14» июня 2020 г.

Ярославль 2020 г.

Реферат

Объём: 55 стр., 5 глав, 11 источников, 5 приложений

Ключевые понятия: статистический анализ соревновательной деятельности волейболистов, ассистент волейбольного статистика, экспресс-анализ, проблема оптимального состава, задача необходимости замены, многослойный персептрон

Объектом исследования является задача сферы волейбольного статистического анализа о подборе оптимальной замены во время матча.

Цель работы – разработка алгоритма машинного обучения для реализации концепции ассистента волейбольного статистика.

В ходе работы рассмотрен простой случай анализа необходимости замены на основании одного игрового элемента. Выполнена процедура обучения многослойной сети прямого распространения для моделирования индивидуальной логики оценки специалистом показателей игроков. Предложены дальнейшие шаги по улучшению качества обучения и рассмотрению других возможных решений поставленной задачи. Сформулирован переход к решению задачи о необходимости замены в общем виде и проблеме оптимального состава в указанной предметной области.

Оглавление

1. Введение.....4
1.1. Предметная область.....4
1.2. Цель и задачи.....5
2. Подходы к разработке ассистента волейбольного статистика 7
2.1. Общая структура возможных задач.....7
2.2. Использование методов машинного обучения для решения задачи о подборе оптимальной замены.....9
2.3. Общий вид задачи о подборе оптимальной замены.....10
2.4. Определение типа поставленной задачи, методов решения и необходимых технических средств.....12
3. Реализация.....15
3.1. Упрощённый вариант задачи.....15
3.2. Проблема объёма обучающей выборки.....15
3.3. Определение вида входного и целевого векторов.....16
3.4. Алгоритм преобразования отчёта в формат входных данных.....18
3.5. Определение предполагаемых результатов для набора данных.....19
3.6. Модель персептрона для задачи «необходимости замены» 20
4. Оценка результатов работы.....24
4.1. Обучение многослойного персептрона.....24
4.2. Интерпретация результатов работы алгоритма.....28
4.3. Необходимые шаги для дальнейшей работы по обучению ассистента.....29
4.4. Переход к общему виду и другим задачам.....30
5. Вывод.....32
6. Источники.....33
Приложение 1.....35
Приложение 2.....36
Приложение 3.....37
Приложение 4.....45

Приложение 5.....48
Приложение 6.....52

1. Введение:

1.1. Предметная область

С развитием информационных технологий стала актуальной разработка специализированных программных комплексов в рамках спортивного статистического анализа. Это позволило обрабатывать значительный объём разнообразных данных. В качестве объекта исследования в данной работе используется статистический анализ соревновательной деятельности волейболистов. В сравнении с другими командными видами спорта специфика волейбольных правил, а также особенности взаимодействий игроков имеют предпосылки к дальнейшему изучению и совершенствованию технологии. Это возможно путём широкого применения различных математических моделей для получения и анализа статистических данных. Современные программные комплексы, являющиеся лидерами в волейбольном статистическом анализе в России и мировом сообществе, такие как итальянский Data Project (Data Volley, Data Video)[1] и российский Volleyball Analyzer[1], позволяют в достаточном объёме обработать игровые действия и сформировать необходимые и подробные статистические отчёты. Полученные данные ещё не являются ответом на все вопросы тренерского штаба. Если процесс ведения статистики опирается на стандарты программных комплексов, то выводы более субъективны и зависят от конкретного игрока, ситуации (дополнительной информации) и специалиста при одних и тех же оценках. Не всегда есть возможность детально и качественно оценить данные. Например, решения о замене приходится принимать за считанные секунды. Стоит отметить,

что связь между значительной частью элементов не поддаётся однозначной алгоритмической оценке. В таких случаях выводы аналитика опираются на интуитивные соображения, основанные на опыте. Появляется необходимость систематизировать подобные решения с целью выполнения качественного анализа в условиях ограниченного времени и решения задач прогнозирования. Таким образом, возникают предпосылки использования алгоритмов машинного обучения.

1.2. Цель и задачи

Целью данного исследования является разработка ассистента волейбольного статистика. Современные программные комплексы для волейбольного статистического анализа на данный момент не используют подобные решения. Реализация идеи ассистента позволит расширить возможности существующих средств и может стать основой для разработки новых программных комплексов. Такие проекты подразумевают нововведения по двум основным направлениям: запись игровых действий и анализ данных.

К первой категории задач относятся вопросы, связанные с подходом к подробному учёту значительной части игровых элементов. Для качественного анализа важно аккуратно указать происходящее на площадке. Современный подход к анализу требует рассмотрения значительного числа параметров в каждом действии. Запись таких данных в условиях ограниченного времени (3 командных действия выполняются примерно за 5-6 секунд) имеет ряд особенностей и связанных с ними недостатков существующих программных комплексов.

Ко второй категории относится разработка ассистента позволяющего смоделировать логику оценки данных специалистом. Решение не предполагает замену специалиста, наоборот оно должно стать полезным инструментом в его деятельности. Вопросами разработки ассистента мы будем заниматься в ходе данной работы. Для достижения поставленной цели необходимо:

1) Сформулировать общие задачи, решения которых на основе статистических показателей играют наиболее важную роль в работе волейбольного тренера, выбрать на реализацию одну, формализовать условие задачи в общем виде
2) Выбрать алгоритм для получения первичных результатов с опорой на особенности задачи и возможности совершенствования её условия, изучить особенности применения алгоритма и его реализации
3) Изучить возможности составления отчётов в программе Data Volley, детально изучить структуру отчёта
4) Упростить условие задачи до вида, наиболее доступного с точки зрения восприятия и удобного для получения первичного решения
5) Разработать алгоритмы преобразования данных из формата отчётов в формат входных данных задачи, составить выборку
6) Реализовать выбранный алгоритм и выполнить процедуру обучения, оценить результаты
7) Рассмотреть другие алгоритмы машинного обучения на примере поставленной задачи, выбрать наиболее удачный

8) Перейти к реализации других более общих задач (для каждой потребуется проделать аналогичную работу)

2. Подходы к разработке ассистента волейбольного статистика

2.1. Общая структура возможных задач

Несмотря на широкий спектр вопросов, решаемых в рамках статистической оценки соревновательной деятельности волейболистов, все они, так или иначе, сводятся к задачам о принятии решения относительно состава либо тактических действий. Другими словами, специалист выполняет классификацию показателей по степени их соответствия качественной игре и делает прогноз, что некое принятое решение может положительно повлиять на игру команды. Приведём общую структуру подобных задач:

- Предматчевая подготовка
    - Определение оптимального состава игроков: исходя из статистических данных эффективности игроков в общекомандных взаимодействиях исследуемой команды и предполагаемого соперника, а также мнения специалиста (неявных статистических данных, основанных на опыте), подобрать оптимальный стартовый состав команды.
    - Определение набора рекомендованных тактических действий: исходя из статистических данных эффективности комбинационной составляющей игры исследуемой команды и действий защиты предполагаемого соперника, а также мнения специалиста (неявных статистических данных, основанных на опыте), составить набор рекомендованных комбинаций.

- Экспресс-анализ (выполняется по ходу игры)
  - Выбор оптимальной замены
    - Вынужденная замена: исходя из статистических данных эффективности игроков в общекомандных взаимодействиях исследуемой команды и соперника, а также мнения специалиста (неявных статистических данных, основанных на опыте), определить позицию в основном составе (если не определена заранее) и выбрать оптимальную замену.
    - Тактическая замена (под конкретную задачу): исходя из статистических данных эффективности игроков, в рамках оцениваемых элементов исследуемой команды и соперника, а также мнения специалиста (неявных статистических данных, основанных на опыте), определить позицию в основном составе и выбрать оптимальную замену.
  - Выбор оптимальной комбинации: исходя из статистических данных эффективности комбинационной составляющей игры исследуемой команды и действий защиты предполагаемого соперника на текущий момент времени, а также мнения специалиста (неявных статистических данных, основанных на опыте), определить оптимальную комбинацию на последующий розыгрыш. Допускается альтернативная формулировка: определить целесообразность использования какой-либо комбинации.

- Решение о тайм-ауте: определить необходимость взять тайм-аут, исходя из текущих показателей (например, несколько ошибок подряд).

Можно заметить, что в формулировках прослеживается общий подход. Вопросы, связанные с экспресс-анализом, затрагивают задачи предматчевой подготовки с той лишь разницей, что решаются непосредственно на игре. При выборе тактической замены, специалист пользуется теми же соображениями, что и при выборе вынужденной, однако может пренебрегать рядом показателей, не имеющих большого значения в конкретном случае. Например, если игрока выпускают исключительно на приём на несколько розыгрышей, нет необходимости рассматривать другие возможные действия: подача, атака, блок (специфика волейбола позволяет делать подобные выводы). Комбинации можно рассмотреть как эффективность взаимодействия ряда игроков, следовательно, и эта задача некоторым образом сводится к подбору оптимального состава. Следовательно, мы можем пользоваться универсальными инструментами при решении задач. Но обучение алгоритма будет опираться на опыт и мнение конкретного специалиста относительно конкретных игроков, действий и ситуаций.

Путём анализа логики игры нами обобщены ключевые задачи, решаемые специалистами, и мы можем опираться на перечисленные факты при формализации условий задач.

2.2. Использование методов машинного обучения для решения задачи о подборе оптимальной замены

В качестве приоритетной мы будем рассматривать задачу экспресс-анализа и подбора оптимальной замены. Данную задачу можно разделить на две подзадачи:

1) Определить, требуется ли команде замена и кого из игроков на площадке необходимо заменить (далее «необходимость замены»)
2) Определить, кто из запасных игроков наиболее подходит для замены (далее «оптимальность замены»).

В данной работе мы подробно рассмотрим вопрос об оценке результатов деятельности игроков на площадке (необходимость замены). В качестве программы для работы с волейбольной статистикой используется итальянский продукт Data Volley 2007 [2]. В современном профессиональном волейболе используется подход с жёстко закреплённым амплуа. Таким образом, каждый игрок оценивается по результатам тех действий, за которые он отвечает. Действия последовательны. Например, от качества приёма косвенно может зависеть и качество атаки, но напрямую влияние на результат показатели приёма оказывают только в случае ошибок (одна из оценок игрового элемента). Концепция программы Data Volley предполагает возможность дать игровому действию до 6 оценок [3]. В подсчёте результатов могут учитываться наборы характеристик элемента, другие действия и оценки других игроков. В частности, понятие «позитивный приём» включает в себя две оценки «хороший приём» и «отличный приём». Отметим, что каждая оценка ставится не по опыту специалиста, а по объективным критериям (доводка мяча, тип

передачи следом, игрок выполнивший передачу), которые являются стандартом в работе статистика. Аналогичный подход используется и для других игровых действий. Путём классических и некоторых специальных формул специализированная программа выполняет расчёты показателей игровых элементов и формирует различные отчёты. Нами будет рассматриваться подробный отчёт по каждой оценке действия на площадке.

Решение о замене игрока принимается тренером не только на основе отчёта. Могут учитываться дополнительные факторы (в том числе психологические). Например, при плохих показателях в начале партии тренер может оставить игрока на площадке, если знает по опыту, что он способен их улучшить на следующем игровом отрезке, либо если игроку требуется набрать опыт. Каждый такой фактор можно формализовать и оценить конкретным алгоритмом. Однако, подобные показатели индивидуальны у различных специалистов и самих игроков, а в целом может оцениваться значительное множество факторов. Поэтому для решения поставленной задачи и комплексной оценке описанных выше показателей мы будем использовать методы машинного обучения.

2.3. Общий вид задачи о подборе оптимальной замены

Первым шагом будет формализация поставленной задачи в общем виде. Мы определим вид входных данных. Ожидаемым ответом будет игрок, наиболее требующий замены, либо вывод, что замена команде не требуется. Рассмотрим вектор, где каждая координата является одним из игровых показателей волейболиста (номер игрока также является показателем).

Координаты упорядочены по игрокам. Дополнительно указаны показатели состояния игры:

$$X=(N_1, Ind_{11}, \ldots, Ind_{1m}, \ldots, N_n, Ind_{n1}, \ldots, Ind_{nm}, Part, StG, T).$$

Если амплуа игрока или обстоятельства таковы, что он не выполнял какие-то действия на площадке, то соответствующая координата принимает значение по умолчанию. Такие значения зависят от конкретного показателя. В качестве состояния игры учитываются:

- Part – часть сета (партии). Принимает целочисленные значения на [1..3]. Каждый сет делится на 3 части: до 8 очков (набранных одной из команд), до 16 очков, до конца партии. 5-ая партия состоит из 2-ух частей, независимо от счёта.
- StG – числовое значение равное изменению разницы в счёте за прошедший игровой отрезок (например: было – 8:6, стало 9:10, StG = -3). Является на данный момент единственной характеристикой связывающей показатели игры команды с конкретным матчем. В дальнейшем планируется рассмотреть этот вопрос и найти способ охарактеризовать действия команды относительно игры соперника. В данном случае указанный формат позволяет одним числом заложить в компоненту вектора входных данных разницу в счёте и динамику. Рассмотрение разницы в счёте и собственно счёта для алгоритма не является достаточно информативным и требует дополнительного указания динамики. Интуитивно специалист оценивает счёт по аналогичной схеме. Таким образом, мы формализуем на первый взгляд тривиальный, однако имеющий свои особенности шаг.

- T – кол-во взятых тайм-аутов (0, 1 или 2). Данный показатель имеет смысл учесть, как влияющий на состояние игроков элемент.
- C_k – допускается ли правилами замена данного игрока k (1 – да, 0 – нет). Один из показателей игрока (Ind_k)
- PIn_k – положение игрока k (в качестве характеристики игрока Ind_k): на поле/ на замене (1 – на поле, 0 – на замене). Данный параметр в связке с предыдущим указывает алгоритму, как рассматривать игрока и допустимо ли его рассмотрение. В дальнейшем планируется рассмотреть показатели подробнее. Требуется учесть для профессионального волейбола позицию (амплуа) на площадке и возможный набор амплуа у замены.
- E – эффективность игрового действия (в качестве характеристики игрока Ind_k). Характеристика конкретного рассматриваемого элемента (приёма, атаки и др.) каждого игрока. Стартовым значениям координаты на новую игру являются итоговые значения в предыдущей игре. Показатель E может быть вычислен по различным формулам и в зависимости от конкретного действия. Можно учитывать разницу с показателями предыдущих замеров для учёта динамики. Общими подходами считаются отношение положительных оценок к их общему числу (результативность) и отношение разницы положительных и отрицательных (иногда без учёта нейтральных) оценок к общему числу. Мы же будем использовать более совершенные формулы, о чём пойдёт речь далее (см. реализацию).

В качестве целевого вектора рассмотрим бинарный набор, упорядоченный относительно игроков входного вектора (отсутствие необходимости замены рассматриваем как n+1 игрока):

$$Y=(y_1,\dots,y_n,y_{n+1}).$$

В данном случае 1 считаем значением «замена необходима» (отсутствие необходимости замены в команде эквивалентно необходимости замены «отсутствия необходимости»), 0 – «не требуется».

2.4. Определение типа поставленной задачи, методов решения и необходимых технических средств

Задача «необходимости замены» относится к задачам классификации [4]. Пусть X — множество описаний игроков, Y — множество наименований игроков. Существует неизвестная целевая зависимость — отображение $y^{*}:X\to Y$, значения которой известны только на объектах конечной обучающей выборки $X=\{(x_1,y_1),\dots,(x_m,y_m)\}$. Требуется построить алгоритм $a:X\to Y$, способный классифицировать произвольный набор показателей игроков $x\in X$. В нашем случае классификация бинарная: множество значений признака принадлежит множеству {0 – замена не требуется, 1 – замена требуется}.

На данном этапе в качестве алгоритма предлагается использовать нейронную сеть прямого распространения (многослойный персептрон) [5]. Схема персептрона и принцип работы нейрона представлены ниже:

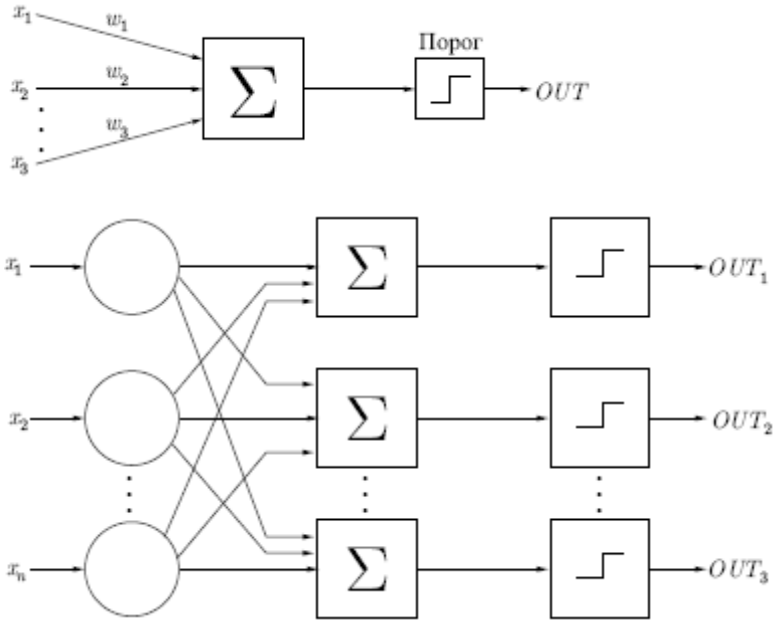


Схема персептрона (однослойного)

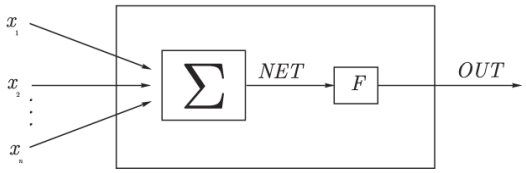


Схема работы одного нейрона (F – функция активации)

Данный вид нейросетей применяется для решения задач указанного типа. Выбор алгоритма обусловлен универсальностью метода. Решение задачи предполагает переход от частных и простых случаев, к общим и более сложным. Их вид относительно формализации пока не определён однозначно, но предполагается, что останется в рамках описанной концепции. Таким образом, использование легко адаптируемого с точки зрения реализации персептрона наиболее подходит для получения первичных результатов и оценки. Далее предполагается опробовать следующие алгоритмы классификации: Random Forest, XGBoost и др. Для реализации персептрона планируется использовать средства языка Python и библиотек scikit-learn [6], Keras [7]. Выбор Keras обусловлен наличием интуитивного понимания структуры

модели на этапе проектирования и возможности детальной настройки параметров. Отчёты для генерации обучающей выборки будут получены средствами профессионального программного обеспечения для статистического анализа соревновательной деятельности волейболистов Data Volley 2007. Преобразование данных из отчётов программы к необходимой форме предполагается выполнить авторскими алгоритмами. Вся работа выполняется на языке Python.

3. Реализация

3.1. Упрощённый вариант задачи

В качестве первичного решения мы рассмотрим упрощённый вариант задачи «необходимости замены». Для оценки используем единственный игровой элемент – приём. Набором показателей будут два показателя приёма и один показатель состояния игры. На игровом отрезке рассматриваются все игроки без указания статуса (на поле/на замене). Если в результате на замену будет предложен игрок, которого нет на поле, это эквивалентно ответу, что замена не требуется.

3.2. Проблема объёма обучающей выборки

В работе по использованию волейбольной статистикой в качестве обучающей выборки есть существенная проблема. Для качественного обучения и оценки работы алгоритма требуется большой объём данных. Многие такие данные (например, в медицине) собираются длительное время за счёт оценки значительного кол-ва пациентов. Любая спортивная команда имеет свойство изменчивости (иногда по ходу сезона). Меняться может и тренер, и состав. Случается, состав меняется полностью. Обучение предполагается под конкретного тренера, но с учётом его работы с конкретными игроками. Таким образом, в идеале обучение без потери ряда факторов необходимо проводить на данных установленного состава команды. Мы можем пренебречь незначительным изменением и даже учитывать данные другой команды, с которой тренер уже работал, но и результат будет более общий.

Стоит отметить, что в данной работе используются отчёты с реальных матчей. Для их получения требуется подробная запись, а не общедоступные результаты. Такие данные нам удалось получить только от одной команды и за прошедший сезон (используются данные волейбольного клуба «Ярославич» сезона 2019/20 чемпионата России высшей лиги А).

Сезон состоял из 24 матчей. Это очень мало. Матч состоит из 3-5 партий. Таким образом, мы можем расширить выборку, если рассмотрим сет за единицу. Но и так объём недостаточный для работы. На этапе формализации задачи мы рассматривали показатель Part, отвечающий за игровой отрезок партии. Можно сделать детализацию данных на отрезках. Однако в этом случае мы потеряем объективность показателей с точки зрения статистики. По ходу партии показатели учитываются с начала партии до момента оценки. Объективность отчётов растёт прямо пропорционально объёму записанных данных. Поэтому мы ввели показатель Part в качестве указателя на рассматриваемый этап сета с точки зрения статистики. Другие партии могут оцениваться независимо, а могут с учётом прошедших сетов (на усмотрение специалистов). Мы будем рассматривать партии независимо друг от друга. Теперь можно сделать детализацию и взять за единицу выборки часть партии от её начала. В этом случае первый игровой отрезок останется наименее объективным. Второй – с учётом первого будет уже более информативным. Третий – полностью соответствует объективным показателям сета. Показатель Part позволяет считать каждый отрезок уникальным. Такой подход наиболее соответствует работе со статистикой на игре.

3.3. Определение вида входного и целевого векторов

Требуется сгенерировать средствами программы Data Volley по 3 отчёта на каждый матч в соответствии с отрезком партии. В отчёт включаются только оценки приёма. Выполняется детализация по игрокам и по сетам. Рассмотрим пример отчёта по приёму (см. Приложение 1). Наша задача выбрать необходимые показатели и преобразовать их в формат входного вектора. В каждом отчёте присутствует разное кол-во игроков. Максимально возможное число в данной выборке 6 (универсальностью пока пренебрегаем, в идеале на вход подаётся вся команда). Если в отчёте не хватает игроков, мы добавляем мнимых до нужного числа, но в показатель «номер» указываем -1 (не существующий номер), а остальные координаты обнуляем.

Оценками приёма будут показатель Ind (в данном случае – индекс игрового действия) и результативность позитивного приёма (оценки + и # в отчёте) с учётом его ценности в команде (E). Рассмотрим их подробнее:

$$Ind = Tot_{¿} \times W_{¿} + Tot_{¿} \times W_{¿} + Tot_{-¿} \times W_{-¿} + Tot_{i} \times W_{i} + Tot_{¿ \times \frac{W_{¿} \times Tot_{¿} + W_{¿}}{Tot} \times factor_{¿}}{¿}{¿}$$

- Tot_h – кол-во оценок игрового действия типа h.
- Tot – общее кол-во оценок игрового действия
- Factor (factor = 1) – Вес Ind игрового действия (учитывается в общих оценках игрока по всем игровым элементам).
- W_h – вес оценки h:

h	=	/	-	!	+	#
W	-3	-3	-1	0	7	10

Формула используется программой DataVolley и является взвешенной средней оценкой. Значения factor и W определены стандартом.

В качестве эффективности в работе предлагается использовать следующую формулу (формула автора работы):

¿

- pos_p – кол-во оценок игрового действия, удовлетворяющих позитивному по смыслу задачи результату
- tot_p – общее кол-во оценок игрового действия
- $max_{1<=k<=n}(pos_k)$ – максимальное по команде кол-во оценок игрового действия, удовлетворяющих позитивному по смыслу задачи результату.

Эта формула учитывает значимость позитивного показателя среди показателей в команде. Она является более информативной в сравнении с классической результативностью. Например: 1 позитивный приём из двух и 5 из 10 дают одинаковую результативность, если не учитывать, что во втором случае игрок чаще вступал в игру и позитивного приёма у него больше. Однако если позитивного приёма у команды нет, используется классическая формула.

Вектор входных данных имеет вид:

$$X=(Part, N_1, Ind_1, E_1, \dots, N_6, Ind_6, E_6).$$

В качестве целевого возьмём вектор, где каждая координата отвечает за статус игрока (0 – замена не требуется, 1 – требуется замена):

$$Y=(y_1, \dots, y_6, y_7).$$

3.4. Алгоритм преобразования отчёта в формат входных данных

Алгоритм преобразования отчётов и получения данных в требуемом виде использует библиотеки xlrd и xlwt языка Python. Выбор обусловлен форматом отчётов. Data Volley 2007 работает только с форматом “.xls”. Алгоритм включает следующие шаги:

- Перебор имеющихся отчётов
- Чтение Excel файла отчёта. Перебор игроков проход по требуемым ячейкам файла
- Запись необходимых данных с распределением по сетам в виртуальной выборке
- Запись выборки в файл с соблюдением необходимой размерности (заполнением недостающих данных по умолчанию, как было описано ранее)
- Дополнительно реализованы алгоритмы чтения готовой выборки в массив numpy. Использование данного формата данных и соответствующей библиотеки обусловлено требованиями библиотек для реализации алгоритмов машинного обучения.

Подробная реализация приведена в приложении (см. Приложение 3). Все отчёты Data Volley одного формата, следовательно, отчёты по другим игровым элементам обрабатываются аналогичным образом (решение универсально).

3.5. Определение предполагаемых результатов для набора данных

Для каждой записи обучающей выборки необходимо указать результат. Нашей задаче на данный момент не требуется оценка специалиста. Формат входных данных доступный и не предполагает большого тренерского опыта. Идея ассистента подразумевает обучение под любого тренера и специфику его оценок показателей игроков. На начальном этапе для нас важно увидеть способность алгоритма решать поставленную задачу на простом примере с точки зрения понимания. Поэтому мы сами смоделируем возможные действия тренера.

Алгоритм определяет игрока, которого требуется сменить. Рассматриваются тренерские оценки показателей приёма независимо и в комплексе. Выводы делаются на основе определённых пороговых значений. Мы выбираем значения показателей интуитивно, опираясь на наши представления об их влиянии на игру. Для двух игроков учитывается фактор отсутствия необходимости замены при неудовлетворительных показателях на различных отрезках сета. В случае определения на замену нескольких игроков выбор делается по наихудшим показателям. Приоритет отдаётся показателю Ind. Далее рассматривается показатель E. Приведём описание алгоритма:

- Перебор игроков и выбор претендентов на замену
    - Рассматривается 3 случая. Игрок является претендентом на замену, если:
        - $E < 0,3$

        - Ind < 5 и при этом E < 0,5. Такой случай описывает ситуацию, когда игрок примерно половину приёма делает хорошим, но при этом другая половина – ошибки, ведущие к потере очков. Мы исключаем не самый плохой показатель E, не учитывающий долю фатальных ошибок
        - Ind < 4
    - При Part < 3 (начало и середина партии) игрок под номером 13 не включается в список претендентов, а при Part < 2 не включается игрок под номером 18
- Среди претендентов выбирается тот, чей показатель Ind ниже, в случае равенства оценивается E. Если претендентов нет, замена не требуется.

Реализация алгоритма представлена в приложении (см. Приложение 4).

В результате мы имеем обучающую выборку объёмом 288 записей. Каждая запись характеризует игроков по показателям приёма и содержит предполагаемое решение о замене (см. Приложение 2). Выборка полностью подготовлена к процедуре обучения.

3.6. Модель персептрона для задачи «необходимости замены»

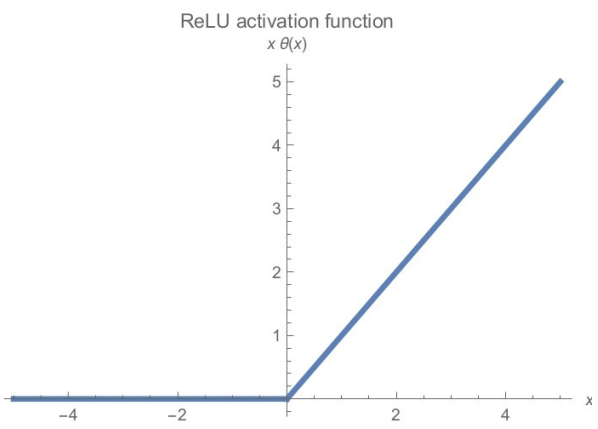
На данном этапе для решения задачи «необходимости замены» используется многослойный персептрон. Можно провести параллель с задачей о распознавании рукописных цифр – одной из классических задач машинного обучения. Двумерное представление цифры, являющееся набором

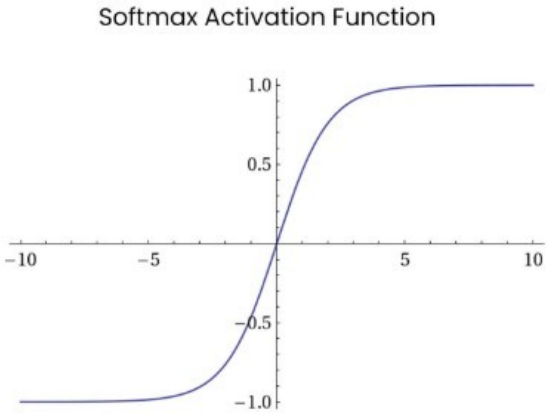
характеристик точек изображения, раскладывается в одномерный упорядоченный входной набор. Решается задача бинарной классификации для каждой из 10 цифр относительно вероятности совпадения написания (схожести) с эталоном. Выбирается наиболее вероятный результат. Нам необходимо выполнить аналогичные действия. В задаче «необходимости замены» набор входных данных фактически представляет собой двумерный набор из игроков с их показателями, который аналогично раскладывается в одномерное представление. Небольшое отличие в том, что мы отдельно указываем общие параметры игры. Таким образом, понятие «необходимость замены» закодировано набором показателей игроков. Целевой вектор в нашем случае показывает принадлежность рассматриваемого понятия какому-то игроку либо отсутствию необходимости замены. Можно сделать вывод, что наша задача сводится к классической задаче. Поэтому в качестве начальных параметров модели мы используем параметры одного из предлагаемых решений задачи классификации рукописных цифр [8].

На входе имеем слой размерностью 19 (по кол-ву входных данных в наборе). Рассматриваем случай с одним скрытым слоем на кол-ве нейронов, равном кол-ву входов. Модель имеет 7 выходов:

```
visible = Input(shape = (input_size,), name = 'Input')
hidden1 = Dense(19, activation = 'relu', name =
'Dense_1')(visible)
output = Dense(output_size, activation = 'softmax',
                name = 'Output')(hidden1).
```

В качестве функций активации мы используем: на скрытых слоях – ReLu [6], на выводящем – Softmax [6].





$$ReLu(x)=\begin{cases} 0, x<0 \\ x, x\geq 0 \end{cases}$$

$$Softmax(x)_i=\frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

Аналогично устанавливаем параметры, необходимые для обучения модели: функция потерь – Categorical Crossentropy [10], оптимизация – RMSprop [9], метрики оценки качества обучения:

self.model.compile(loss = 'categorical_crossentropy',
optimizer = 'rmsprop',
metrics = ['accuracy',
metrics.Precision(), metrics.Recall()]).

Отдельно остановимся на метриках. Помимо предлагаемой Accuracy (Аккуратность) [11] мы используем Precission (точность) [11], и Recall (Полнота) [11]. Возможны следующие исходы классификации:

- Игрок, соответствующий критериям необходимости замены объявлен кандидатом на замену, т.е. положительный класс распознан как положительный (True Positive — TP).

- Игрок с удовлетворительными показателями, распознан как не нуждающийся в замене, т.е. отрицательный класс распознан как отрицательный (True Negative — TN).
- На замену предложен игрок, не нуждающийся в замене, т.е. имела место ошибка (I рода), в результате которой отрицательный класс был распознан как положительный (False Positive — FP).
- Игрок с неудовлетворительными показателями не был рекомендован на замену, т.е. имела место ошибка (II рода), в результате которой положительный класс был распознан как отрицательный (False Negative — FN).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Аккуратность (доля правильных ответов алгоритма) не даёт полного представления о качестве обучения. Например, такая метрика не учитывает ситуацию с неравными классами (доля заведомо выше из-за преобладания объектов одного из классов). Проблему решают точность (доля правильных ответов в пределах класса) и полнота (доля верно предсказанных объектов, относящихся к положительному классу).

Обучающая выборка делится в соотношении 3:1. Учитывая незначительный объём выборки, такое распределение оптимальное. В качестве параметра в процедуру обучения передаётся одна эпоха. Далее это число увеличивается, а результаты сравниваются. Исходя из оценок качества обучения, мы корректируем параметры модели: количество

слоёв, количество нейронов в слое, функции активации, функцию потерь и оптимизатор. Обучение проводим методом fit() с сохранением истории обучения:

self.history = self.model.fit(train_x, train_y,
validation_data = (test_x, test_y),
epochs = 1).

Дополнительно рассматриваем оценки качества решения задачи. Делим объекты на две категории: игроки, отсутствие необходимости замены. Считаем долю правильных ответов алгоритма для каждой категории. Таким образом мы ответим на вопрос, какую часть задачи алгоритм решает лучше (определяет игрока для замены либо отсутствие необходимости замены).

4. Оценка результатов работы

4.1. Обучение многослойного персептрона

Обучение модели в установленной конфигурации дало низкие результаты. Увеличение кол-ва эпох не давало серьёзной положительной динамики. Экспериментальным путём установлено, что лучшие результаты обучения получаются при следующей конфигурации модели:

visible = Input(shape = (input_size,), name = 'Input')
hidden1 = Dense(29, activation = 'relu', name = 'Dense_1')(visible)
hidden2 = Dense(29, activation = 'relu', name = 'Dense_2')(hidden1)
output = Dense(output_size, activation = 'softmax', name = 'Output')(hidden2)

self.model.compile(loss = 'binary_crossentropy', optimizer = 'rmsprop',
metrics = ['accuracy', metrics.Precision(), metrics.Recall()]).

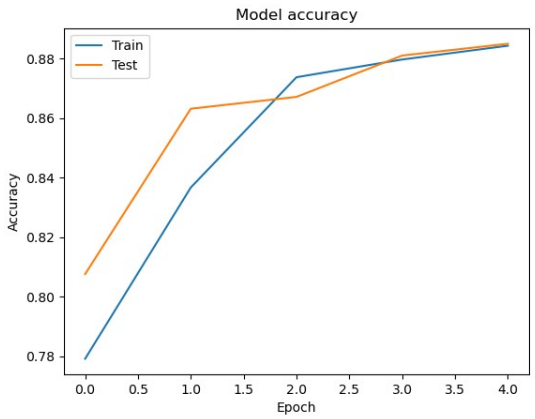
Можно заметить, что изменилась не только архитектура модели (два слоя с увеличенным числом нейронов), но и функция потерь. Мы используем Binary Crossentropy [10], что логично с точки зрения типа задачи (бинарная классификация). Подробная реализация алгоритма представлены в приложении (см. Приложение 5). Сравнительный анализ исходной и полученной конфигураций приведён ниже:

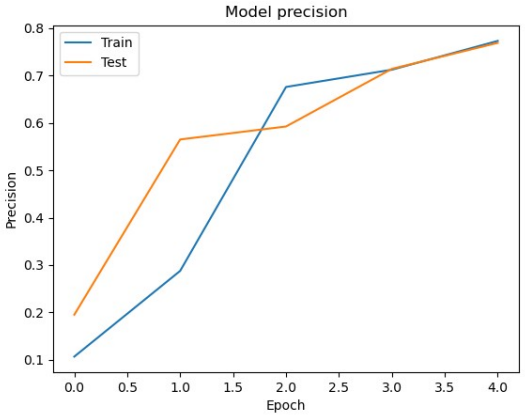
Оценка	Исходная	Полученная

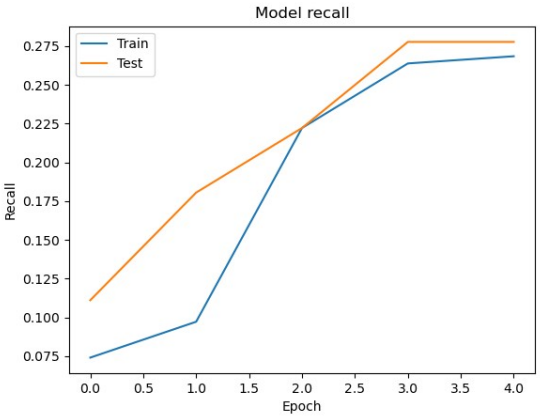
	конфигурация модели	конфигурация модели
Accuracy	0,11	0,86
Precision	0,11	0,48
Recall	0,08	0,17
Доля правильно классифицированных игроков тестовой выборки	0,03	0,54
Доля правильно классифицированного отсутствия необходимости замены	0,45	0,00

Судить о качестве работы алгоритма пока рано. Метрики показали достаточно низкую оценку. Вероятнее всего проблема будет решена при увеличении кол-ва эпох. Мы будем увеличивать с шагом 5. Ниже представлены графики динамики показателей по метрикам при различном кол-ве эпох обучения (выделены ключевые этапы).

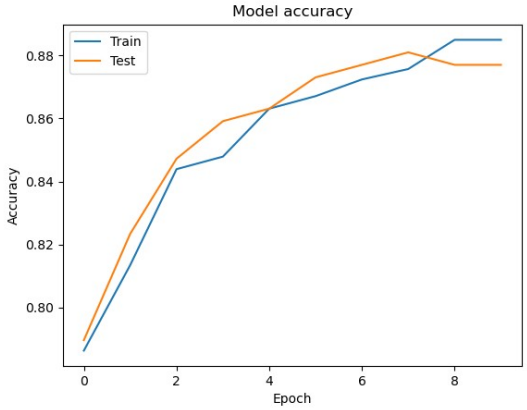
5 эпох:

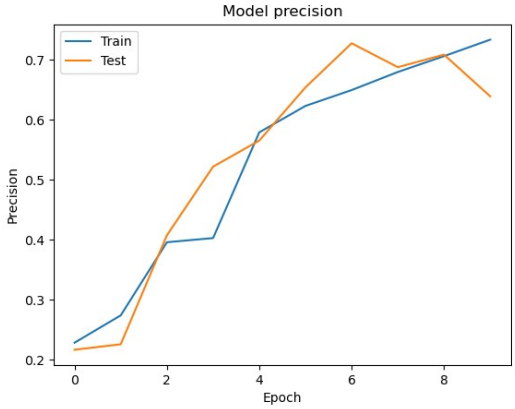


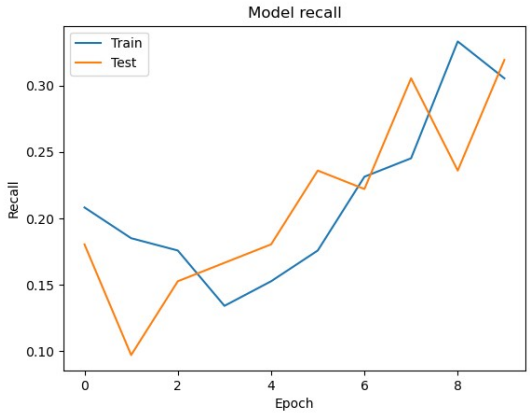




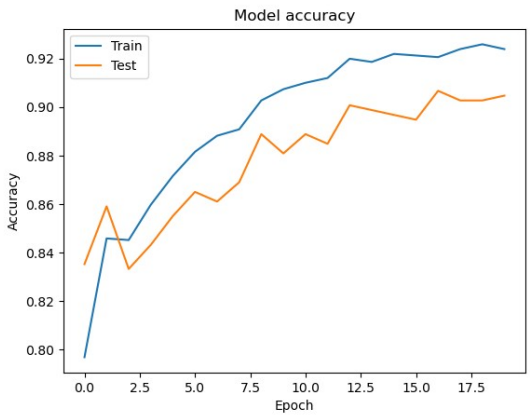
10 эпох:

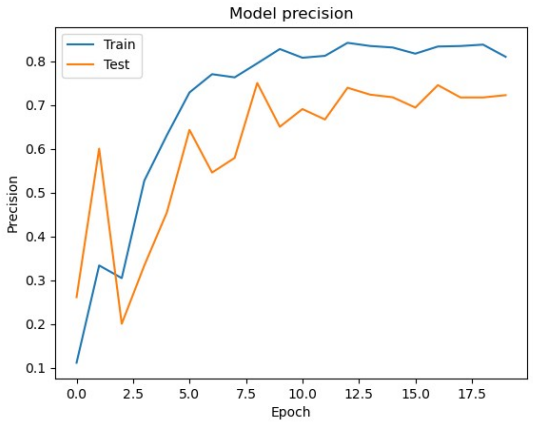


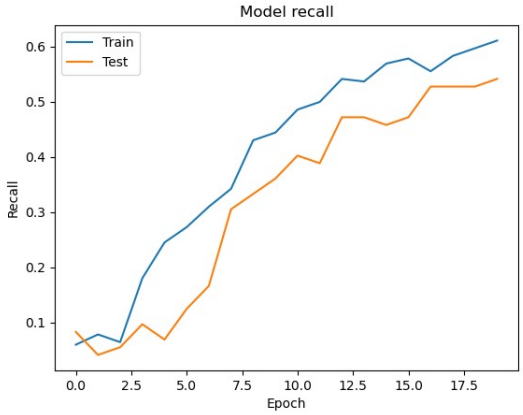




20 эпох:

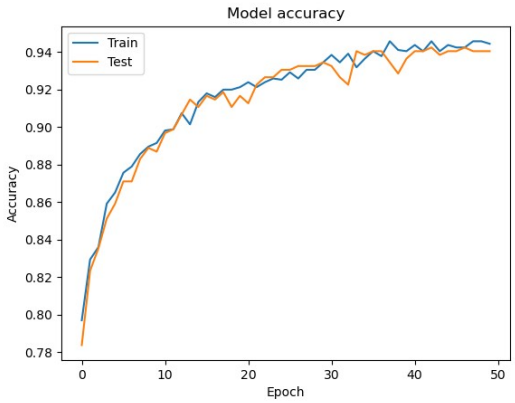


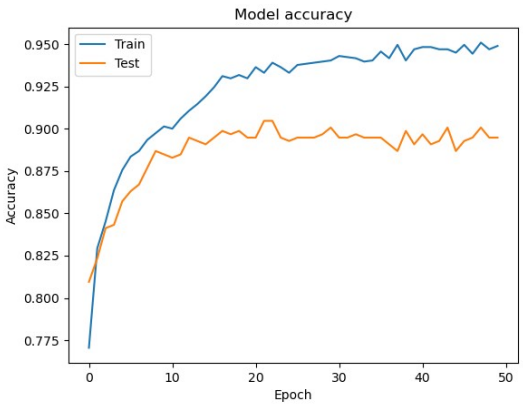


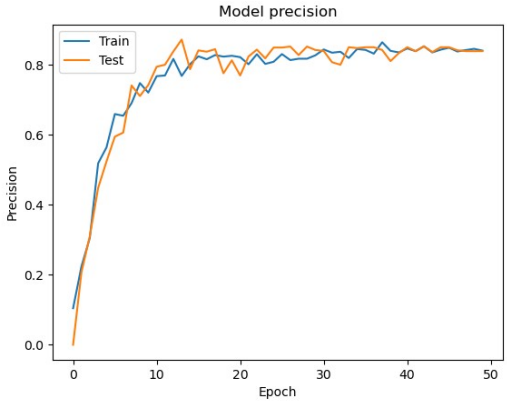


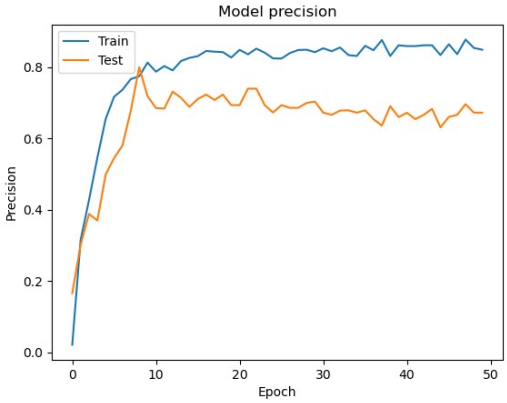
50 эпох (здесь рассмотрим две попытки обучения):

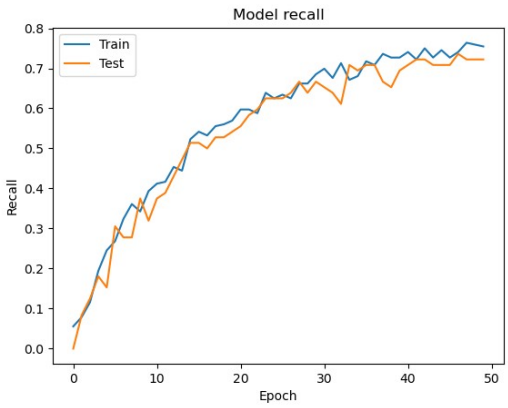
1 попытка *2 попытка*

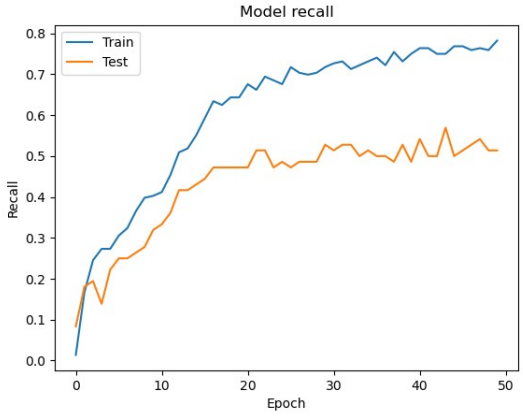












Из графиков следует, что ситуация с качеством обучения выправилась. Однако можно заметить 2 важные детали:

- Обнаружены скачки качества
- При большом кол-ве эпох возможно переобучение, но результаты дали два случая.

Причина и в том и в другом случае – относительно малый объём выборки. Мы смогли достигнуть высокого качества обучения, но получить более гладкие и постоянные результаты на используемом обучающем наборе возможности нет. Результат зависит от инициализации весов. В каких-то случаях этот процесс более удачный. Однако даже в этом случае однозначно расценивать показатели как очень высокие нельзя.

Нет гарантии, что на большем объёме данных они останутся на том же уровне (обучающая часть выборки объёмом 216 записей, тестовая часть выборки объёмом 72 записи).
Есть вероятность увеличения выборки через год (с учётом следующего сезона), что позволит провести более качественную процедуру обучения. Это даст ответ на вопрос, насколько алгоритм способен решать поставленную задачу. Но использование его на практике подразумевает обучение на объёме данных одного сезона (если обучение выполняется под конкретную тренерскую работу). Поэтому требуется найти способ сгладить показатели.

На графике *попытки 2* можно заметить резкое расхождение показателей при обучении и тестировании. Замечено, что такая ситуация встречалась наиболее часто при кол-ве эпох больше 20. Это может свидетельствовать о переобучение. Модель запоминает данные, на которых проводилось обучение, и теряет способность классифицировать неизвестные. Таким образом, можно сделать вывод, что на данный момент 20 эпох достаточно для качественного обучения.

4.2. Интерпретация результатов работы алгоритма

На выходе алгоритма мы имеем распределение вероятностей для каждого игрока и показателя отсутствия необходимости замены. Выбирается наиболее вероятный выход. Остальные данные можно использовать в качестве резервной информации с поправкой на то, что результат получен алгоритмом. Эти данные не учитывают дополнительные факторы, которым не обучалась модель.

Однако, если специалист не согласен с результатом (наиболее вероятным выводом), он может рассмотреть следующий по убыванию.

4.3. Необходимые шаги для дальнейшей работы по обучению ассистента

Первое, что необходимо сделать: найти способ сгладить результаты обучения. Возможным решением будет применение кросс-валидации (перекрёстной проверки). Этот метод применяется в аналогичных случаях, когда объём выборки сравнительно мал. Метод заключается в следующем:

- Мы фиксируем конфигурацию модели и кол-во эпох обучения
- Выборку делим случайным образом на тестовую и обучающую
- Выполняем процедуру обучения
- Считаем среднее арифметическое по оценкам качества обучения.

Таким образом, мы получим сглаженные показатели.

Второе, имеет смысл опробовать другие существующие алгоритмы классификации и сравнить результаты обучения. Это позволит выбрать наиболее удачный алгоритм. Учитывая, что задача разделена на две части (определить игрока для замены, определить необходимость замены в команде), мы можем получить два различных алгоритма удачных решений для каждой части. В таком случае имеет смысл в первую очередь решать задачу определения необходимости замены и в случае положительного ответа, определять игрока (каждая

подзадача решается своим алгоритмом). Планируется выполнить работу по реализации ассистента на основе алгоритма Random Forest (случайный лес). На данный момент, если посмотреть на введённые нами ранее оценки (доля правильных ответов в определении игрока на замену и доля правильных ответов в определении необходимости замены) можно сделать вывод, что многослойный персептрон обе подзадачи решает примерно одинаково (пренебрегаем скачками). Один из результатов обучения представлен в приложении (см. Приложение 6).

4.4. Переход к общему виду и другим задачам

Переход к общему виду задачи (описан на этапе формализации) предполагает расширение функционала. Данную работу следует проводить, когда будет получено оптимальное решение нашей упрощённой задачи. Переход заключается в формировании входного набора из всех игроков (с подробным указанием состояния игры) и добавлении к игрокам показателей по другим игровым элементам (подача, атака и т.д.). Персептрон чувствителен к изменению размерности входных данных. Однако расширение не затрагивает логику входного набора, следовательно, решение предполагает лишь изменение конфигурации модели и повторное обучение с сохранением подхода (перекрёстная проверка при обучении, алгоритмы, эпохи).

Задача «оптимальности замены» решается аналогичным способом, но принимает противоположную форму. Требуется определить игрока среди запасных, наиболее соответствующего заданной игровой ситуации.

Подбор оптимального состава команды включает задачу предматчевой подготовки: определить наиболее удачный состав. Понятие оптимального состава эквивалентно отсутствию необходимости замены. Если мы введём дополнительное значение показателю Part (см. формализацию) – состояние до игры, то задача подбора оптимального состава сводится к задаче «необходимости замены». В таком случае можно перебрать (использовать разумный перебор с учётом амплуа игроков) возможные составы и сделать прогноз, при каком наборе показателей замена не требуется. Если такой случай не обнаружен, выбрать версию состава, где вероятность отсутствия необходимости замены наибольшая.

Таким образом, общая схема решения проблемы подбора оптимального состава выглядит так:

- Используя математические преобразования по статистическим формулам, получить необходимые показатели
- Использовать машинное обучение и алгоритмы классификации для решения задачи о подборе оптимальной замены
- Использовать подзадачу «необходимости замены» (в общем виде) для подбора оптимального состава на игру.

5. Вывод

В ходе работы нам удалось получить первичную реализацию ассистента волейбольного статистика. Мы выбрали для реализации задачу о подборе оптимальной замены в условиях ограниченного времени (экспресс-анализ). Рассмотрели и обучили многослойный персептрон на простом случае с одним игровым элементом. Оценили результаты обучения и определили существующие проблемы. Рассмотрели переход к полному решению и общему виду. Сформулировали схему решения проблемы подбора оптимального состава команды.

В ходе дальнейшей работы необходимо решить проблему относительно малой выборки и выполнить действия по сглаживанию оценок качества обучения. Будут рассмотрены другие алгоритмы классификации для сравнения качества решения задачи и определено оптимальное решение. Необходимо рассмотреть более сложные задачи специалистов в сфере статистического анализа соревновательной деятельности волейболистов: комбинации и действия связующего, различные показатели, не используемые для экспресс-анализа.

6. Источники

1. Бабынин Ю.А., Кононов В.Н. Использование компьютерных программ для статистической обработки соревновательной деятельности волейболистов.
2. Проект Data Project https://www.dataproject.com/EN/en/Volleyball (дата обращения: 13.06.2020)
3. Data Volley 2007 Краткое руководство [electronic resource]. – URL: http://www.scoutman.net/uploads/1900/01/dvcodesrus_doc.pdf (дата обращения: 13.06.2020),
4. Задача классификации [electronic resource]. – URL: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8 (дата обращения: 13.06.2020)
5. НОУ ИНТУИТ | Основы теории нейронных сетей [electronic resource]. – URL: https://www.intuit.ru/studies/courses/88/88/info (дата обращения: 13.06.2020)
6. Scikit-learn Machine Learning in Python [electronic resource]. – URL: https://scikit-learn.org/stable/ (дата обращения: 13.06.2020)
7. Keras [electronic resource]. – URL: https://keras.io/ (дата обращения: 13.06.2020)
8. Машинное обучение на практике с Python и Keras [electronic resource]. – URL: https://pythonru.com/primery/mashinnoe-

obuchenie-na-praktike-s-python-i-keras (дата обращения: 13.06.2020)
9. Функции потерь библиотеки Keras [electronic resource]. – URL: http://www.100byte.ru/python/loss/loss.html (дата обращения: 13.06.2020)
10. Университет искусственного интеллекта. Keras Документация. Перевод на русский язык [electronic resource]. – URL: https://ru-keras.com/loss/ (дата обращения: 13.06.2020)
11. Оценка качества в задачах классификации и регрессии [electronic resource]. – URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9E%D1%86%D0%B5%D0%BD%D0%BA%D0%B0_%D0%BA%D0%B0%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%B0_%D0%B2_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0%D1%85_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8_%D0%B8_%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%B8 (дата обращения: 13.06.2020)

Приложение 1

Part = 1

Yaroslavich Yaroslavl | Player detail | Reception | Set Detail | ;0-8;

Player	Skill	Type	S	Set	Ind.	*E%	Tot	=	%	BP	pS	/	%	BP	pS	-	%	!	%	+	%	#	%	BP	pS
Team	Reception			Mat	4	26%	19	2	11%	2						3	16%	4	21%	6	32%	4	21%		
				1	4	33%	6									1	17%	2	33%	1	17%	2	33%		
				2	3		7	2	29%	2						1	14%	1	14%	1	14%	2	29%		
				3	4	50%	6									1	17%	1	17%	4	67%				
2 Dicarev	Reception			Mat	7	100%	1													1	100%				
				3	7	100%	1													1	100%				
9 Migel	Reception			Mat	2		8	2	25%	2						1	12%	2	25%	1	12%	2	25%		
				1	3	33%	3											2	67%			1	33%		
				2	1	50%	4	2	50%	2						1	25%					1	25%		
				3	7	100%	1													1	100%				
13 Piskarev	Reception			Mat	6	75%	8											2	25%	4	50%	2	25%		
				1	8	100%	2													1	50%	1	50%		
				2	6	67%	3											1	33%	1	33%	1	33%		
				3	5	67%	3											1	33%	2	67%				
18 Elichev	Reception			Mat	-1	100%	2									2	100%								
				1	-1	100%	1									1	100%								
				3	-1	100%	1									1	100%								

Part = 2

Yaroslavich Yaroslavl | Player detail | Reception | Set Detail | ;0-16;

Player	Skill	Type	S	Set	Ind.	*E%	Tot	=	%	BP	pS	/	%	BP	pS	-	%	!	%	+	%	#	%	BP	pS
Team	Reception			Mat	3	5%	39	7	18%	7		1	3%	1		7	18%	7	18%	6	15%	11	28%		
				1	3		14	4	29%	4		1	7%	1		1	7%	2	14%	1	7%	5	36%		
				2	2	15%	13	3	23%	3						3	23%	3	23%	1	8%	3	23%		
				3	5	33%	12									3	25%	2	17%	4	33%	3	25%		
2 Dicarev	Reception			Mat	7	100%	1													1	100%				
				3	7	100%	1													1	100%				
9 Migel	Reception			Mat	1	20%	15	5	33%	5						2	13%	4	27%	1	7%	3	20%		
				1	1	20%	5	2	40%	2								2	40%			1	20%		
				2	0	57%	7	3	43%	3						2	29%	1	14%			1	14%		
				3	6	67%	3											1	33%	1	33%	1	33%		
11 Jeltov	Reception			Mat	10	100%	1															1	100%		
				1	10	100%	1															1	100%		
13 Piskarev	Reception			Mat	5	46%	13	1	8%	1						1	8%	3	23%	4	31%	4	31%		
				1	5	33%	3	1	33%	1										1	33%	1	33%		
				2	5	60%	5											2	40%	1	20%	2	40%		
				3	5	40%	5									1	20%	1	20%	2	40%	1	20%		
18 Elichev	Reception			Mat	2	33%	9	1	11%	1		1	11%	1		4	44%					3	33%		
				1	3	20%	5	1	20%	1		1	20%	1		1	20%					2	40%		
				2	-1	100%	1									1	100%								
				3	3	33%	3									2	67%					1	33%		

Part = 3

Yaroslavich Yaroslavl | Player detail | Reception | Set Detail | ;0-26;

Player	Skill	Type	S	Set	Ind.	*E%	Tot	=	%	BP	pS	/	%	BP	pS	-	%	!	%	+	%	#	%	BP	pS
Team	Reception			Mat	3	2%	62	10	16%	10		1	2%	1		12	19%	15	24%	10	16%	14	23%		
				1	2	-5%	21	4	19%	4		1	5%	1		3	14%	6	29%	1	5%	6	29%		
				2	2	15%	20	4	20%	4						6	30%	3	15%	2	10%	5	25%		
				3	3	24%	21	2	10%	2						3	14%	6	29%	7	33%	3	14%		
2 Dicarev	Reception			Mat	7	100%	1													1	100%				
				3	7	100%	1													1	100%				
9 Migel	Reception			Mat	1	14%	22	7	32%	7						2	9%	7	32%	3	14%	3	14%		
				1	1	17%	6	2	33%	2								3	50%			1	17%		
				2	0	44%	9	4	44%	4						2	22%	1	11%	1	11%	1	11%		
				3	3	29%	7	1	14%	1								3	43%	2	29%	1	14%		
11 Jeltov	Reception			Mat	10	100%	1															1	100%		
				1	10	100%	1															1	100%		
13 Piskarev	Reception			Mat	4	29%	21	1	5%	1						4	19%	5	24%	5	24%	6	29%		
				1	3		8	1	12%	1						2	25%	2	25%	1	12%	2	25%		
				2	5	43%	7									1	14%	2	29%	1	14%	3	43%		
				3	5	50%	6									1	17%	1	17%	3	50%	1	17%		
18 Elichev	Reception			Mat	2	24%	17	2	12%	2		1	6%	1		6	35%	3	18%	1	6%	4	24%		
				1	2	17%	6	1	17%	1		1	17%	1		1	17%	1	17%			2	33%		
				2	2	50%	4									3	75%					1	25%		
				3	2	14%	7	1	14%	1						2	29%	2	29%	1	14%	1	14%		

Приложение 2

Примеры наборов входных данных

Part	Player	Ind	Eff		Result
1	2	7	0,5		0
	9	7	0,5		0
	13	5	0,67		0
	18	-1	0		0
	-1	0	0		0
	-1	0	0		0
					1

2	2	7	0,33		0
	9	6	0,44		0
	13	5	0,6		0
	18	3	0,11		1
	-1	0	0		0
	-1	0	0		0
					0

Приложение 3

Алгоритм преобразование отчёта в формат входных данных:

```
# Класс игрового элемента
class Element():
    def __init__(self, ind, tot, plus, plus_plus):
        self.ind = ind
        self.tot = tot
        self.plus = plus
        self.plus_plus = plus_plus

    # Округление показателя с указанной точностью
    @staticmethod
    def round_parameter(parametr, accuracy):
        t_num = parametr/accuracy

        return int(t_num + (0.5 if t_num > 0 else -0.5))*accuracy

from element_parameters import Element

# Класс игрового элемента "приём". Наследуется от базового класса
игрового элемента
class Reception(Element):
    def __init__(self, ind, tot, plus, plus_plus):
        super().__init__(ind, tot, plus, plus_plus)
        self.eff_by_max_team = None

    # Получение кол-ва позитивного приёма
    def get_positive_count(self):
        return self.plus + self.plus_plus

    # Вычисление показателя эффективности приёма по масимальному кол-
    ву позитивного приёма команды
```

```
    def set_eff_by_max_team (self, max_pos_cnt):
        pos_cnt = self.get_positive_count()

        if max_pos_cnt == 0:
            self.eff_by_max_team = Element.round_parameter(pos_cnt/self.tot,
0.01)
        else:
            self.eff_by_max_team = Element.round_parameter((pos_cnt*pos_cnt)/
                                                           (self.tot*max_pos_cnt),
                                                    0.01)

# Класс игрока
class Player:
    def __init__(self, number, reception):
        self.number = number
        self.reception = reception

# Класс эксперимента
class Experiment:
    # Параметры эксперимента
    experiment_players_cnt = 6
    experiment_parameters_cnt = 1
    player_parameters_cnt = 3

    def __init__(self, match_set, part):
        self.match_set = match_set
        self.part = part
        self.players = []
        self.players_count = 0
        self.__max_pos_cnt_reception = 0

    # Получение игрока из списка игроков эксперимента по индексу
    def get_player(self, index):
        return self.players[index]
```

```
    # Добавление игрока в список игроков эксперимента
    def add_player(self, player):
        self.players.append(player)
        pos_cnt_reception = player.reception.get_positive_count()
        self.players_count += 1

        if pos_cnt_reception > self.__max_pos_cnt_reception:
            self.__max_pos_cnt_reception = pos_cnt_reception

    # Вычисление показателя эффективности игрового элемента
    # по макисмальному кол-ву позитивного исполнения элемента в
команде
    # для каждого игрока эксперимента
    def calculate_eff_by_max_team(self):
        for player in self.players:
            player.reception.set_eff_by_max_team(self.__max_pos_cnt_reception)

import xlrd
import xlwt
from coach_models import Coach
from experiment_description import Experiment

# Класс выборки
class Sample:
    def __init__(self):
        self.experiments = []
        self.quantity = 0

    # Добавление эксперимента в список выборки
    def add_experiments(self, experiments):
        self.experiments.extend(experiments)
        self.quantity += len(experiments)
```

```
# Запись выборки в файл
def save_sample(self, path):
    sample_xls = xlwt.Workbook()
    sheet = sample_xls.add_sheet("Sample")
    sheet.write(0, 0, "Part")
    sheet.write(0, 1, "Player")
    sheet.write(0, 2, "Ind")
    sheet.write(0, 3, "Eff")
    sheet.write(0, Experiment.experiment_parameters_cnt
                    + Experiment.player_parameters_cnt + 1, "Result")

    row = 1

    for experiment in self.experiments:
        part = experiment.part
        coach = Coach(experiment)
        coach.evaluation_by_player()
        conclusion = coach.conclusion
        sheet.write(row, 0, part)
        cnt = 0

        while cnt < Experiment.experiment_players_cnt:
         sheet.write(row, Experiment.experiment_parameters_cnt +
                          Experiment.player_parameters_cnt + 1,
                    conclusion[cnt])

            if cnt < experiment.players_count:
                player = experiment.get_player(cnt)
                sheet.write(row, 1, player.number)

                reception = player.reception
                sheet.write(row, 2, reception.ind)
                sheet.write(row, 3, reception.eff_by_max_team)
            else:
```

```
                sheet.write(row, 1, -1)

                col = 2

                while col < Experiment.experiment_parameters_cnt +
                            Experiment.player_parameters_cnt:
                    sheet.write(row, col, 0)
                    col += 1

            if cnt == Experiment.experiment_players_cnt - 1:
                sheet.write(row + 1, Experiment.experiment_parameters_cnt +
                                    Experiment.player_parameters_cnt + 1,
                            conclusion[cnt + 1])
                row += 3
            else:
                row += 1

            cnt += 1

        sample_xls.save(path)

        print ("Save sample: successfully")

# Модуль обработки отчётов Data Volley 2007

import xlrd
from reception_description import Reception
from player_description import Player
from experiment_description import Experiment

# Получение выборки по отчётам Data Volley
def report_to_sample(matches, parts, sample, path):
    match = 1
```

```
    while match <= matches:
        part = 1

        while part <= parts:
            report = xlrd.open_workbook(path + "M" + str(match) + "P" +
str(part) + ".xls")
            reception_sheet = report.sheet_by_index(0)

            row = 5
            str_player = reception_sheet.cell_value(row, 0)
            report_experiment_list = []
            sets_in_report = set()

            while str_player != "0.0":
                if str_player != "" and str_player != "Team":
                    set_row = row + 1
                    actual_set = str(reception_sheet.cell_value(set_row, 4))

                    while actual_set != "Match" and actual_set != "":
                        if not (float(actual_set) in sets_in_report):
                            experiment = Experiment(float(actual_set), part)

                            reception = Reception(get_equal(reception_sheet, set_row,
5),
                                                  get_equal(reception_sheet, set_row, 7),
                                                      get_equal(reception_sheet, set_row,
20),
                                                  get_equal(reception_sheet, set_row, 22))

                            player = Player(get_player_number(str_player), reception)

                            experiment.add_player(player)

                            report_experiment_list.append(experiment)
```

```
                    sets_in_report.add(float(actual_set))

                else:
                    experiment =
get_experiment_from_list(report_experiment_list, float(actual_set))

                    reception = Reception(get_equal(reception_sheet, set_row,
5),
                                          get_equal(reception_sheet, set_row, 7),
                                          get_equal(reception_sheet, set_row, 20),
                                          get_equal(reception_sheet, set_row, 22))

                    player = Player(get_player_number(str_player), reception)

                    experiment.add_player(player)

                set_row += 1
                actual_set = str(reception_sheet.cell_value(set_row, 4))

            row += 1
            str_player = str(reception_sheet.cell_value(row, 0))

        part += 1

        for item in report_experiment_list:
            item.calculate_eff_by_max_team()

        sample.add_experiments(report_experiment_list)
    match += 1

print("Report convert to sample: successfully")
print("Quantity:", sample.quantity)

# Получение номера игрока в строке отчёта Data Volley
```

```
def get_player_number(str_player):
    i = 0
    number = 0

    while str_player[i] != ' ':
        number = number*10 + int(str_player[i])
        i += 1

    return float(number)

# Получение значения показателя отчёта Data Volley
def get_equal(sheet, row, col):
    equal = sheet.cell_value(row, col)

    if equal != "":
        return float(equal)
    else:
        return 0

# Получение эксперимента в списке экспериментов отчёта по сету
def get_experiment_from_list(report_experiment_list, actual_set):
    item = 0

    while report_experiment_list[item].match_set != actual_set:
            item += 1

    return report_experiment_list[item]
```

Приложение 4

Алгоритм модели тренерской оценки показателей игроков:

```
from experiment_description import Experiment

# Класс возможных моделей поведения тренера на основе показателей
class Coach:
    def __init__(self, experiment):
        self.experiment = experiment
        self.conclusion = []

        for i in range(Experiment.experiment_players_cnt + 1):
            self.conclusion.append(0)

    # Модель оценки показателей игроков.
    # Учитывает:
    #   *пороговые значения показателей,
    #   *особенности игроков
    def evaluation_by_player(self):
        # Определение претендентов на замену на основе показателей
игроков
        players_on_replace = []

        item = 0

        while item < self.experiment.players_count:
            player = self.experiment.get_player(item)

            if player.reception.eff_by_max_team < 0.3:
                if not self.__have_individual_approach(player):
                    players_on_replace.append(player)
            elif player.reception.ind < 5.0 and player.reception.eff_by_max_team <
0.5:
```

```
                if not self.__have_individual_approach(player):
                    players_on_replace.append(player)
            elif player.reception.ind < 4.0:
                if not self.__have_individual_approach(player):
                    players_on_replace.append(player)

            item += 1

        # Вывод тренера о необходимости замены игрока: 0 - замена не требуется, 1 - замена
         требуется.
        # Дополнительно указывается вывод об отсутствии необходимости замены в качестве
         мнимого игрока
        if len(players_on_replace) == 0:
            self.conclusion[len(self.conclusion) - 1] = 1
        else:
            worse = players_on_replace[0]

            for player in players_on_replace:
                worse = Coach.__get_worse_player(worse, player)

            item = 0

            while item < self.experiment.players_count:
                player = self.experiment.get_player(item)

                if player.number == worse.number:
                    self.conclusion[item] = 1

                item += 1

    # Индивидуальные особенности оценки игроков
    def __have_individual_approach(self, player):
```

```
        return player.number == 13.0 and self.experiment.part < 3.0 or
player.number == 18.0 and
                    self.experiment.part < 2.0

    # Проверка принадлежности игрока статусу худшего. Возвращает
худшего игрока
    @staticmethod
    def __get_worse_player(worse, player):
        if player.reception.ind < worse.reception.ind:
            return player
        elif player.reception.eff_by_max_team <
worse.reception.eff_by_max_team:
            return player
        else:
            return worse
```

Приложение 5

Реализация алгоритма ассистента, обучения и оценки качества обучения:

```
from keras.layers import Dense
from keras.models import Model, load_model
from keras.layers import Input
import keras.metrics as metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from experiment_description import Experiment
from data_description import Data

# Класс ассистента реализованного алгоритмом многослойного
персептрона.
# Включает:
#   *методы построения и обучения модели,
#   *методы получения результата работы алгоритма,
#   *метод проверки работы алгоритма.
class Assistant:
    def __init__(self):
        self.model = None
        self.history = None

    # Построение и обучение модели с выводом оценок обучения
    def create(self, data):
        train_x, test_x, train_y, test_y = train_test_split(data.data_x, data.data_y,
test_size = 0.25)

        input_size = Experiment.experiment_parameters_cnt +
                        Experiment.experiment_players_cnt*Experiment.player_para
                meters_cnt
        output_size = Experiment.experiment_players_cnt + 1
```

```
        # Модель имеет:
        #   *вход соответствующий набору игроков и параметров эксперимента,
        #   *два скрытых слоя по 29 нейронов
        #   *выводящий слой, соответствующий набору игроков эксперимента
        #    и дополнительному выходу, отвечающему за необходимость замены в команде
        visible = Input(shape = (input_size,), name = 'Input')
        hidden1 = Dense(29, activation = 'relu', name = 'Dense_1')(visible)
        hidden2 = Dense(29, activation = 'relu', name = 'Dense_2')(hidden1)
        output = Dense(output_size, activation = 'softmax', name = 'Output')(hidden2)

        self.model = Model(inputs=visible, outputs=output)

        self.model.summary()
        self.model.compile(loss = 'binary_crossentropy', optimizer = 'rmsprop',
                           metrics = ['accuracy', metrics.Precision(),
                    metrics.Recall()])

        self.history = self.model.fit(train_x, train_y, validation_data = (test_x, test_y), epochs = 20)

        # Проверка работы алгоритма на полном объёме выборки по результатам
        self.print_output_analisis(data, "Sample data")

        # Проверка работы алгоритма на тестовом наборе по результатам
        test_data = Data(data_x = test_x, data_y = test_y)
        self.print_output_analisis(test_data, "Test data")

        self.plot_accuracy()
        self.plot_precision()
        self.plot_recall()
```

```
        print("Create assistant: successfully")

    # Получение набора результатов работы алгоритма на входном наборе
    def get_predict_results(self, input_set):
        return self.model.predict(input_set.reshape((1, -1)))

    # Получение метки искомого значения в наборе результатов работы
алгоритма
    def get_result_label(self, input_set):
        predict_results = self.get_predict_results(input_set).ravel()
        label_result = 0
        max_result = predict_results[label_result]

        for label in range(Experiment.experiment_players_cnt + 1):
            if predict_results[label] > max_result:
                max_result = predict_results[label]
                label_result = label

        return label_result

    # Проверка работы алгоритма по результатам
    def print_output_analisis(self, data, name_data):
        ok_cnt = 0
        necessary_replacement_ok_cnt = 0
        not_require_replacement_ok_cnt = 0

        for i in range(data.quantity):
            label = self.get_result_label(data.data_x[i])

            if label == data.get_result_label(i):
                ok_cnt += 1

                if label == Experiment.experiment_players_cnt:
```

```
                not_require_replacement_ok_cnt += 1
            else:
                necessary_replacement_ok_cnt += 1

        necessary_replacement_cnt, not_require_replacement_cnt =
                                    data.get_split_necessary_not_require_replacemen
                                    t_count()

        print(f"{name_data} - Quantity: {data.quantity}, OK_count: {ok_cnt}")

        print(f"Replacement_count: {necessary_replacement_cnt},
                Replacement_OK_count: {necessary_replacement_ok_cnt}")
        print("Replacement_accuracy:",
necessary_replacement_ok_cnt/necessary_replacement_cnt)

        print(f"Replacement_count: {not_require_replacement_cnt},
                Replacement_OK_count: {not_require_replacement_ok_cnt}")
      print("Not_require_accuracy:",
                not_require_replacement_ok_cnt/not_require_replacement_cnt)

    # График динамики оценок после каждой эпохи по метрике
"Аккуратность"
    def plot_accuracy(self):
        plt.plot(self.history.history['accuracy'])
        plt.plot(self.history.history['val_accuracy'])
        plt.title('Model accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Test'], loc='upper left')
        plt.show()

    # График динамики оценок после каждой эпохи по метрике "Точность"
    def plot_precision(self):
        plt.plot(self.history.history['precision_1'])
```

```
    plt.plot(self.history.history['val_precision_1'])
    plt.title('Model precision')
    plt.ylabel('Precision')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

# График динамики оценок после каждой эпохи по метрике "Полнота"
def plot_recall(self):
    plt.plot(self.history.history['recall_1'])
    plt.plot(self.history.history['val_recall_1'])
    plt.title('Model recall')
    plt.ylabel('Recall')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

Приложение 6

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 19)                0
_________________________________________________________________
Dense_1 (Dense)              (None, 29)                580
_________________________________________________________________
Dense_2 (Dense)              (None, 29)                870
_________________________________________________________________
Output (Dense)               (None, 7)                 210
=================================================================
Total params: 1,660
Trainable params: 1,660
Non-trainable params: 0
_________________________________________________________________
Train on 216 samples, validate on 72 samples
Epoch 1/20
 32/216 [===>..........................] - ETA: 1s - loss: 1.0155 - accuracy: 0.8080 - precision_1: 0.2963 - recall_1: 0
216/216 [==============================] - 0s 1ms/step - loss: 0.8100 - accuracy: 0.8168 - precision_1: 0.3057 - recall_
1: 0.2222 - val_loss: 0.5525 - val_accuracy: 0.8333 - val_precision_1: 0.4062 - val_recall_1: 0.3611
Epoch 2/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.6295 - accuracy: 0.8393 - precision_1: 0.4333 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.6077 - accuracy: 0.8221 - precision_1: 0.3432 - recall
_1: 0.2685 - val_loss: 0.4632 - val_accuracy: 0.8274 - val_precision_1: 0.3171 - val_recall_1: 0.1806
Epoch 3/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.5104 - accuracy: 0.8080 - precision_1: 0.2381 - recall_1: 0
216/216 [==============================] - 0s 30us/step - loss: 0.4990 - accuracy: 0.8254 - precision_1: 0.3235 - recall
_1: 0.2037 - val_loss: 0.3932 - val_accuracy: 0.8294 - val_precision_1: 0.3250 - val_recall_1: 0.1806
Epoch 4/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.5050 - accuracy: 0.8214 - precision_1: 0.2143 - recall_1: 0
216/216 [==============================] - 0s 145us/step - loss: 0.4193 - accuracy: 0.8360 - precision_1: 0.3689 - recal
l_1: 0.2083 - val_loss: 0.3391 - val_accuracy: 0.8492 - val_precision_1: 0.4375 - val_recall_1: 0.1944
Epoch 5/20
```

```
 32/216 [===>.....] - ETA: 0s - loss: 0.3967 - accuracy: 0.8170 - precision_1: 0.2000 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.3600 - accuracy: 0.8419 - precision_1: 0.3736 - recall_1: 0.1574 - val_loss: 0.2985 - val_accuracy: 0.8750 - val_precision_1: 0.5957 - val_recall_1: 0.3889
Epoch 6/20
 32/216 [===>.....] - ETA: 0s - loss: 0.3404 - accuracy: 0.8616 - precision_1: 0.5200 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.3231 - accuracy: 0.8578 - precision_1: 0.5057 - recall_1: 0.2037 - val_loss: 0.2851 - val_accuracy: 0.8790 - val_precision_1: 0.6341 - val_recall_1: 0.3611
Epoch 7/20
 32/216 [===>.....] - ETA: 0s - loss: 0.3174 - accuracy: 0.8661 - precision_1: 0.5417 - recall_1: 0
216/216 [==============================] - 0s 145us/step - loss: 0.2999 - accuracy: 0.8724 - precision_1: 0.6186 - recall_1: 0.2778 - val_loss: 0.2658 - val_accuracy: 0.8770 - val_precision_1: 0.7778 - val_recall_1: 0.1944
Epoch 8/20
 32/216 [===>.....] - ETA: 0s - loss: 0.3209 - accuracy: 0.8527 - precision_1: 0.4286 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.2804 - accuracy: 0.8750 - precision_1: 0.7288 - recall_1: 0.1991 - val_loss: 0.2531 - val_accuracy: 0.8869 - val_precision_1: 0.7273 - val_recall_1: 0.3333
Epoch 9/20
 32/216 [===>.....] - ETA: 0s - loss: 0.3182 - accuracy: 0.8839 - precision_1: 0.7143 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.2682 - accuracy: 0.8869 - precision_1: 0.7711 - recall_1: 0.2963 - val_loss: 0.2433 - val_accuracy: 0.8849 - val_precision_1: 0.7917 - val_recall_1: 0.2639
Epoch 10/20
 32/216 [===>.....] - ETA: 0s - loss: 0.2643 - accuracy: 0.8839 - precision_1: 0.7500 - recall_1: 0
216/216 [==============================] - 0s 73us/step - loss: 0.2510 - accuracy: 0.8922 - precision_1: 0.8118 - recall_1: 0.3194 - val_loss: 0.2291 - val_accuracy: 0.9127 - val_precision_1: 0.7917 - val_recall_1: 0.5278
Epoch 11/20
 32/216 [===>.....] - ETA: 0s - loss: 0.2351 - accuracy: 0.9152 - precision_1: 0.8421 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.2424 - accuracy: 0.9008 - precision_1: 0.8173 - recall_1: 0.3935 - val_loss: 0.2238 - val_accuracy: 0.9107 - val_precision_1: 0.8000 - val_recall_1: 0.5000
Epoch 12/20
 32/216 [===>.....] - ETA: 0s - loss: 0.2055 - accuracy: 0.9375 - precision_1: 0.9500 - recall_1: 0
```

```
216/216 [==============================] - 0s 102us/step -
loss: 0.2307 - accuracy: 0.9028 - precision_1: 0.8000 - recal
l_1: 0.4259 - val_loss: 0.2179 - val_accuracy: 0.9127 - val_precision_1: 0.8333 -
val_recall_1: 0.4861
Epoch 13/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.2318 - accuracy: 0.8929 -
precision_1: 0.7222 - recall_1: 0
216/216 [==============================] - 0s 72us/step -
loss: 0.2233 - accuracy: 0.9061 - precision_1: 0.7984 - recall
_1: 0.4583 - val_loss: 0.2122 - val_accuracy: 0.9107 - val_precision_1: 0.8293 -
val_recall_1: 0.4722
Epoch 14/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.2264 - accuracy: 0.9107 -
precision_1: 0.7727 - recall_1: 0
216/216 [==============================] - 0s 72us/step -
loss: 0.2160 - accuracy: 0.9074 - precision_1: 0.7969 - recall
_1: 0.4722 - val_loss: 0.2044 - val_accuracy: 0.9187 - val_precision_1: 0.8444 -
val_recall_1: 0.5278
Epoch 15/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.2248 - accuracy: 0.8973 -
precision_1: 0.7368 - recall_1: 0
216/216 [==============================] - 0s 145us/step -
loss: 0.2101 - accuracy: 0.9061 - precision_1: 0.7721 - recal
l_1: 0.4861 - val_loss: 0.2062 - val_accuracy: 0.9127 - val_precision_1: 0.8333 -
val_recall_1: 0.4861
Epoch 16/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.1670 - accuracy: 0.9286 -
precision_1: 0.8077 - recall_1: 0
216/216 [==============================] - 0s 72us/step -
loss: 0.2026 - accuracy: 0.9120 - precision_1: 0.7943 - recall
_1: 0.5185 - val_loss: 0.1935 - val_accuracy: 0.9206 - val_precision_1: 0.8333 -
val_recall_1: 0.5556
Epoch 17/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.1702 - accuracy: 0.9509 -
precision_1: 0.9565 - recall_1: 0
216/216 [==============================] - 0s 102us/step -
loss: 0.1966 - accuracy: 0.9180 - precision_1: 0.8194 - recal
l_1: 0.5463 - val_loss: 0.1984 - val_accuracy: 0.9107 - val_precision_1: 0.8000 -
val_recall_1: 0.5000
Epoch 18/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.1798 - accuracy: 0.9286 -
precision_1: 0.8077 - recall_1: 0
216/216 [==============================] - 0s 72us/step -
loss: 0.1923 - accuracy: 0.9140 - precision_1: 0.7867 - recall
_1: 0.5463 - val_loss: 0.1929 - val_accuracy: 0.9187 - val_precision_1: 0.8444 -
val_recall_1: 0.5278
Epoch 19/20
 32/216 [===>..........................] - ETA: 0s - loss: 0.1710 - accuracy: 0.9196 -
precision_1: 0.7917 - recall_1: 0
216/216 [==============================] - 0s 72us/step -
loss: 0.1881 - accuracy: 0.9167 - precision_1: 0.7812 - recall
```

_1: 0.5787 - val_loss: 0.1847 - val_accuracy: 0.9147 - val_precision_1: 0.7843 - val_recall_1: 0.5556
Epoch 20/20
 32/216 [===>.....] - ETA: 0s - loss: 0.1672 - accuracy: 0.9464 - precision_1: 0.9167 - recall_1: 0
216/216 [==============================] - 0s 72us/step - loss: 0.1803 - accuracy: 0.9253 - precision_1: 0.8239 - recall
_1: 0.6065 - val_loss: 0.1883 - val_accuracy: 0.9147 - val_precision_1: 0.7959 - val_recall_1: 0.5417
Sample data - Quantity: 288, OK_count: 218
Replacement_count: 230, Replacement_OK_count: 176
Replacement_accuracy: 0.7652173913043478
Replacement_count: 58, Replacement_OK_count: 42
Not_require_accuracy: 0.7241379310344828
Test data - Quantity: 72, OK_count: 52
Replacement_count: 62, Replacement_OK_count: 45
Replacement_accuracy: 0.7258064516129032
Replacement_count: 10, Replacement_OK_count: 7
Not_require_accuracy: 0.7