УДК

НЕЙРОСЕТЕВОЙ ПОДХОД К ПРОГНОЗИРОВАНИЮ СПОРТИВНЫХ ТЕННИСНЫХ ДАННЫХ

Харкевич А.П., студент гр.953504

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Рыкова О. В. – канд. физ.-мат. наук

Аннотация. В данной работе рассмотрены основные этапы получения готовой нейросети для прогнозирования спортивных теннисных данных, анализа качества обучения и последующей оптимизации модели.

Ключевые слова. Python, pandas, xgboost, нейросеть, прогнозирование результатов, теннис.

Спортивное прогнозирование – процесс предсказания, необходимый для планирования в процессе спортивной подготовки и работы в сфере спорта.

Планирование в спорте на основе прогнозов используется различными специалистами в сфере физической культуры и спорта, тренерскими штабами, селекционерами, «скаутами» команд, аналитическими и букмекерскими агентствами.

Прогнозирование в спорте различается по срокам и может быть краткосрочным, среднесрочным, долгосрочным и сверхдолгосрочным. Вид и результат прогноза в спорте может быть различным и в зависимости, от цели, таким результатом может быть как определенный численный результат, процент вероятности исхода того или иного состязания, так и указание наступления конкретного события во временных рамках состязания. Однако, вне зависимости от вида прогноза и ожидаемого результата, спортивное прогнозирование требует анализа набора факторов, на него влияющих. Компьютерные технологии и современные методы искусственного интеллекта позволяют производить такой анализ факторов и осуществлять прогнозирование, получая при этом результат.

Получение готовой нейросети для прогнозирования спортивных данных будет состоять из нескольких этапов:

1) Нахождение данных с матчами и спортсменами
2) Очистка данных
3) Построение модели нейронной сети
4) Их обучение на части данных
5) Валидация моделей на другой части данных и на реальных данных
6) Оптимизация модели

Рассмотрим подробнее каждый из этих этапов.

На первом этапе необходимо произвести поиск нужно нам датасета.

После проведенного поиска было обнаружено, что для текущей задачи хорошо подойдёт github репозиторий с названием «tennis_atp» за авторством пользователя JeffSackmann.

Данный репозиторий содержит огромное количество данных о теннисных матчах более чем за 50 лет. Все данные в нём хранятся в формате csv файлов.

Однако с точки зрения нашей задачи в данном репозитории есть существенный недостаток – данные в нём разбиты по годам и по турнирам. С точки зрения нашей задачи хотелось бы иметь один файл, в которым все данные по годам и турнирам будут объединены. Очевидное решение данной проблемы – загрузить данные вручную или средствами библиотеки какого-либо из языков программирования.

Первый вариант был отвергнут в связи со своей неудобностью и длительностью ручных операций. Однако стоит отметить, что для более узконаправленной задачи он может являться более гибким.

При рассмотрении второго варианта был найден github репозиторий с названием «deuce» за авторством пользователя skoval.

Данный репозиторий содержит пакет, написанный на языке программирования R. Данный пакет парсит данные из репозитория Джеффа Сакмана преобразуя их в единый объект.

Для скачивания пакета была использована библиотека devtools, затем данные были загружены в код и преобразованы в csv файл.

На этом первый этап получения готовой нейросети для прогнозирования спортивных данных был завершен.

На втором этапе был написан скрипт на питоне с использованием 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.

Основная область применения — обеспечение работы в рамках среды Python не только для сбора и очистки данных, но и для задач анализа и моделирования данных, без переключения на более специфичные для статобработки языки (такие, как R и Octave).

Также активно ведётся работа по реализации в pandas «родных» категориальных типов данных.

Pandas прежде всего предназначен для очистки и первичной оценки данных по общим показателям, например среднему значению, квантилям и так далее; статистическим пакетом он в полном смысле не является, однако наборы данных типов DataFrame и Series применяются в качестве входных в большинстве модулей анализа данных и машинного обучения (SciPy, Scikit-Learn и других).

Основные возможности библиотеки pandas:

1) Объект DataFrame для манипулирования индексированными массивами двумерных данных
2) Инструменты для обмена данными между структурами в памяти и файлами различных форматов
3) Встроенные средства совмещения данных и способы обработки отсутствующей информации
4) Переформатирование наборов данных, в том числе создание сводных таблиц
5) Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных
6) Вставка и удаление столбцов данных
7) Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение» (англ. *split-apply-combine*).
8) Слияние и объединение наборов данных
9) Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности
10) Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее

Библиотека оптимизирована для высокой производительности, наиболее важные части кода написаны на Cython и Си.

Т. к. количество турниров достаточно большое и многие турниры уже не актуальны, т.е. на игроков, участвующих в них, данные прогнозироваться уже не будут, первым шагом будет подвыборка нужных нам турниров. Стоит задать временной диапазон порядка 20 лет и рассматривать турниры только в этом временном диапазоне. Иначе мы будем работать с устаревшими данными, не интересными нам для прогнозирования будущих матчей, однако стоит отметить, что для другой задачи эти данные могли бы сыграть определенную полезную роль.

Также, чтобы сделать прогнозы модели более точными, мы ограничим множество матчей матчами, которые происходили на кортах с жестким покрытием. Так как от типа покрытия сильно разнится стиль игры, то на разных кортах разные игроки будут показывать разные результаты. Мы же возьмём самый распространенный тип, чтобы не вносить лишние помехи.

Следующим шагом мы переименуем названия колонок на более понятные, чтобы с ним было удобнее работать и их названия были более осмыслены.

Далее для удобства мы преобразуем год и дату старта турнира к формату даты в питоне.

Затем с помощью регулярных выражений мы запоминаем сколько выигранных игр было у выигравшего и проигравшего игрока во всех сетах, а также общее количество игр во всех играх.

Затем запоминаем число игр, где подавал только один человек (победитель или побежденный). Запоминаем также число раз, когда в игре создавался брейк-поинт. Также запоминаем число раз, когда брейк-поинт был выигран одним из игроков. Мы считаем также число удачных вторых подач игрока. Считаем мы также и такие метрики, как сколько подач было выиграно игроком и сколько очков он выиграл, когда отбивался. Считаем также общее число выигранных очков для каждого игрока и просто общее число очков в матче.

Убираем лишние колонки данных, которые не будут нам полезны. Колонки, где данные были потеряны и не заполнены, мы заполняем дефолтными или средними значениями.

Следующим шагом будет преобразование данных к нужному формату, чтобы для конкретного матча у нас была статистика игроков как для победителя, так и для проигравшего, в отдельных строках.

Затем нам нужно будет преобразовать необработанную статистику матчей игроков из абсолютных значений в относительные соотношения. Это важно, так как абсолютное значение статистики игрока зависит от длительности матча.

Например, предположим, что Федерер выиграет Кирьоса со счётом 6-4, 7-5. Всего Федерер выиграет 13 партий. Если Джокович выиграет партию у Нишикори со счётом 7-6, 3-6, 5-7, 6-2, 7-6, то он выиграет в общей сложности 28 партий, более чем в два раза больше, чем Федерер. Это несправедливое сравнение. Больше смысла имеет сравнивать их коэффициенты выигрыша в игре. Для Федерера коэффициент выигрыша в игре (6+7)/(6+4+7+5) = 0.59, для Джоковича коэффициент выигрыша в игре 0.51. Сравнение этих двух коэффициентов более разумно, чем использование итоговых значений.

Далее мы создадим новые метрики, по которым удобно будет обучать нашу модель.

Первой такой метрикой станет процент выигранных подач. Она рассчитывается следующим образом: число выигранных первых подач плюс число выигранных вторых подач, деленное на общее число первых подач игрока плюс общее число вторых подач игрока плюс общее число двойных ошибок игрока.

Вторая метрика — это процент выигранных очков, когда игрок отбивался: число выигранных очков, когда игрок отбивался, деленная на общее число очков, когда игрок отбивался.

Третьей и четвертой метриками являются среднее число брэйкпоинтов за игру и процент реализации брэйкпоинта.

Далее пятой и шестой метриками будут процент выигранных игр и процент выигранных очков.

Затем мы подсчитаем логарифм ранга игрока и с помощью экспоненциальной функции от ранга его противника считаем вес его победы.

Затем мы подсчитаем взвешенный процент выигранных игр и взвешенный процент выигранных очков в матче.

Ещё одной важной метрикой является так называемый «клатч-фактор».

Клатч – напряженный момент в самом конце игры, от которого зависит исход матча. Умение игрока выложится на полную является важной метрикой.

Клатч-фактор мы будет рассчитывать, как разницу между процентов выигранных игр и процентом выигранных очков.

Следующим важным шагом будет суммирование статистики по данному игроку за предыдущие 10 матчей.

На рисунке 1 мы можем увидеть, как будет выглядеть часть результатов для Роджера Федерера:

tournament_date_index	player_name	player_serve_win_ratio	player_return_win_ratio	player_bp_per_game
2005-01-17	Roger Federer	0.700709	0.408396	0.732828
2006-01-16	Roger Federer	0.704199	0.433687	0.734755
2007-01-15	Roger Federer	0.727015	0.428965	0.721652
2008-01-14	Roger Federer	0.738650	0.407721	0.744655

Рисунок 1 – Часть результатов для Роджера Федерера

Эти показатели должны быть объединены с данными матча. Ключами, которые будут однозначно идентифицировать нашу группу данных, будут дата турнира и имя игрока как для player_1, так и для player_2. Пример данных после объединения с данными матча мы можем увидеть на рисунке 2.

player_1	player_2	tourney_start_date	player_serve_win_ratio_p1	player_serve_win_ratio_p2
Roger Federer	Marcos Baghdatis	2005-01-17	0.700709	0.634170
Roger Federer	Andre Agassi	2005-01-17	0.700709	0.692173
Roger Federer	Denis Istomin	2006-01-16	0.704199	0.660193
Roger Federer	Florian Mayer	2006-01-16	0.704199	0.614321
Roger Federer	Max Mirnyi	2006-01-16	0.704199	0.682761

Рисунок 2 – Пример данных после объединения с данными матча

Мы также возьмем различия между агрегатами player_1 и player_2, чтобы уменьшить количество функций и, следовательно, наше время вычисления. Интуитивно это работает, потому что, выиграет ли Федерер матч, зависит от того, насколько хорош его соперник по сравнению с ним. Пример данных после взятия разницы между агрегатами можно увидеть на рисунке 3.

player_1	player_2	tourney_start_date	player_serve_win_ratio_diff
Roger Federer	Marcos Baghdatis	2005-01-17	0.066539
Roger Federer	Andre Agassi	2005-01-17	0.008536
Roger Federer	Denis Istomin	2006-01-16	0.044006
Roger Federer	Florian Mayer	2006-01-16	0.089877
Roger Federer	Max Mirnyi	2006-01-16	0.021438

Рисунок 3 – Пример данных после взятия разницы между агрегатами

В разрезе последних 10 матчей для ранга игрока и логарифмического ранга игрока мы возьмём его наиболее повторяющееся значение по матчам. Для оставшихся метрик мы считаем среднее значение скользящим окном по матчам и берем наиболее часто повторяющееся значение среднего.

Далее мы добавляем полученные новые метрики к исходным данных игроков.

И затем считаем разницу в метриках двух соперников в матче.

Таким образом функции, которые мы будем использовать для наших прогнозов, будут представлять собой разницу в средней статистике каждого игрока за предыдущие 10 матчей. Например, если мы рассматриваем матч Александра Зверева как player_1 и Стефаноса Циципаса как player_2, то для Александра Зверева мы хотим усреднить его статистику (например, процент выигранных партий) по последним 10 матчам, допустим, это число 0.63. То же самое мы сделаем и для Стефаноса Циципаса, допустим, его средний коэффициент выигрыша в матчах - 0.68. Отметим, что это усредненное значение по матчам, в которых каждый игрок участвовал индивидуально, а не по их общим предыдущим поединкам.

Возьмем разницу между двумя характеристиками игроков, 0.63-0.68 = -0.05 и используем ее как характеристику для прогнозирования того, выиграет ли player_1 (Александр Зверев). Мы можем сделать это для множества других статистических данных игроков, таких как ранг игрока, процент выигранных первой и второй подачи или процент выигранных ответных очков.

Для генерации предсказаний была использована модель для классификации XGBClassifier на основе XGBOOST.

XGBoost — алгоритм машинного обучения, основанный на дереве поиска решений и использующий фреймворк градиентного бустинга. В задачах предсказания, которые используют неструктурированные данные (например, изображения или текст), искусственная нейронная сеть превосходит все остальные алгоритмы или фреймворки. Но когда дело доходит до структурированных или табличных данных небольших размеров, в первенстве оказываются алгоритмы, основанные на дереве поиска решений.

XGBoost разрабатывался как исследовательский проект Вашингтонского Университета. Tianqi Chen и Carlos Guestrin представили их работу на конференции SIGKDD в 2016 году и произвели фурор в мире машинного обучения. С момента его введения этот алгоритм не только лидировал в соревнованиях Kaggle, но и был основой нескольких отраслевых передовых приложений. В результате образовалось общество специалистов по анализу данных, вносящих вклад в проекты XGBoost с открытым исходным кодом с ~350 участниками и ~3,600 коммитами на GitHub.

Особенности фреймворка:

1) Широкая область применения: может быть использован для решения задач регрессии, классификации, упорядочения и пользовательских задач на предсказание.
2) Совместимость: Windows, Linux и OS X.
3) Языки: поддерживает большинство ведущих языков программирования, например, C++, Python, R, Java, Scala и Julia.
4) Облачная интеграция: поддерживает кластеры AWS, Azure и Yarn, хорошо работает с Flink, Spark

Дерево принятия решений — простой в визуализации и достаточно понятный алгоритм. Однако не так уж просто понять следующее поколение алгоритмов, основывающихся на деревьях. Поэтому для понимания обратимся к несложной аналогии.

Представьте, что вы специалист по подбору персонала и собеседуете нескольких отличных кандидатов. Каждый шаг эволюции алгоритмов, основанных на деревьях, может быть представлен как версия хода собеседования.

1) Дерево принятия решений: Каждый специалист по подбору персонала при собеседовании кандидата ориентируется по своему списку критериев: образование, опыт работы, успешность прохождения собеседования.
2) Бэггинг: Представьте, что вместо одного специалиста по подбору персонала теперь за каждым кандидатом наблюдают несколько, и каждый имеет возможность проголосовать. Этот алгоритм при принятии окончательного решения учитывает все высказанные мнения.
3) Случайный лес: Этот алгоритм основан на бэггинге. Отличается он тем, что выбирает случайные признаки. То есть, каждый специалист по подбору персонала может проверить знания кандидата лишь в какой-то одной случайно выбранной области.
4) Бустинг: Это альтернативный подход, в котором каждый специалист по подбору персонала основывается на оценке кандидата предыдущим специалистом. Это ускоряет процесс собеседования, так как не подходящие кандидаты сразу же отсеиваются.
5) Градиентный бустинг: Частный случай бустинга, в котором ошибка минимизируется алгоритмом градиентного спуска. То есть, наименее квалифицированные кандидаты отсеиваются как можно раньше.
6) XGBoost: Экстремальный градиентный бустинг. Это идеальная комбинация оптимизации ПО и железа для получения точных результатов за короткое время с минимальным использованием вычислительных ресурсов.

XGBoost и Gradient Boosting Machines (GBM) — ансамбли методов деревьев, которые используют принцип бустинга (чаще всего, алгоритм построения бинарного дерева решений) при помощи архитектуры градиентного спуска. В свою очередь, XGBoost — улучшение фреймворка GBM через системную оптимизацию и усовершенствование алгоритма.

Системная оптимизация:

1) Параллелизация: В XGBoost построение деревьев основано на параллелизации. Это возможно благодаря взаимозаменяемой природе циклов, используемых для построения базы для обучения: внешний цикл перечисляет листья деревьев, внутренний цикл вычисляет признаки. Нахождение цикла внутри другого мешает параллелизировать алгоритм, так как внешний цикл не может начать своё выполнение, если внутренний ещё не закончил свою работу. Поэтому для улучшения времени работы порядок циклов меняется: инициализация проходит при считывании данных, затем выполняется сортировка, использующая параллельные потоки. Эта замена улучшает производительность алгоритма, распределяя вычисления по потокам.
2) Отсечение ветвей дерева: В фреймворке GBM критерий остановки для разбиения дерева зависит от критерия отрицательной потери в точке разбиения. XGBoost использует параметр максимальной глубины max_depth вместо этого критерия и начинает обратное отсечение. Этот “глубинный” подход значительно улучшает вычислительную производительность.
3) Аппаратная оптимизация: Алгоритм был разработан таким образом, чтобы он оптимально использовал аппаратные ресурсы. Это достигается путём создания внутренних буферов в каждом потоке для хранения статистики градиента. Дальнейшие улучшения, как, например, вычисления вне ядра, позволяют работать с большими наборами данных, которые не помещаются в памяти компьютера.

Улучшения алгоритма:

1) Регуляризация: Он штрафует сложные модели, используя как регуляризацию LASSO (L1), так и Ridge-регуляризацию (L2) для того, чтобы избежать переобучения.
2) Работа с разреженными данными: Алгоритм упрощает работу с разреженными данными, в процессе обучения заполняя пропущенные значения в зависимости от значения потерь. К тому же он позволяет работать с различными уровнями разреженности.
3) Метод взвешенных квантилей: XGBoost использует его для того, чтобы наиболее эффективно находить оптимальные точки разделения в случае работы со взвешенным датасетом.
4) Кросс-валидация: Алгоритм использует свой собственный метод кросс-валидации на каждой итерации. То есть, нам не нужно отдельно программировать этот поиск и определять количество итераций бустинга для каждого запуска.

XGBoost может похвастаться лучшей комбинацией «производительность-время обучения» среди других алгоритмов.

Остается лишь грамотно настроить алгоритм путём подбора гиперпараметров.

В нашем случае мы используем логистическую регрессию для бинарной классификации и на выходе получаем вероятность победы одного из игроков.

Для обучения используем 300 деревьев. Для управления взвешиванием новых деревьев, добавленных в модель, используется параметр learning_rate, мы установили его равным 0.02. Параметр максимальной глубины дерева мы установили равным 6.

В качестве метрики оценки для проверки данных была выбрана AUC, что отлично подходит для бинарной классификации.

Чтобы лучше понять, что такое AUC метрика разберемся сначала, что из себя представляет ROC-кривая.

Кривая ROC представляет собой график, показывающий эффективность модели классификации при всех пороговых значениях классификации. Эта кривая отображает два параметра:

1) True Positive Rate
2) False Positive Rate

True Positive Rate (TPR) определяется следующим образом:

$$TPR=TP+FN \quad (1)$$

False Positive Rate (FPR) определяется следующим образом:

$$FPR=FP+TN \quad (2)$$

Кривая ROC отображает соотношение TPR и FPR при различных порогах классификации. Снижение порога классификации позволяет классифицировать больше элементов как положительные, тем самым увеличивая количество ложных срабатываний и истинных срабатываний. На рисунке 4 показан пример кривой ROC.

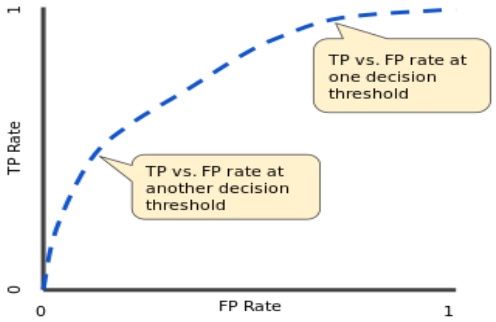

Рисунок 4 – Коэффициент TP и FP при различных порогах классификации

Чтобы вычислить точки на ROC-кривой, мы могли бы много раз оценивать модель логистической регрессии с разными порогами классификации, но это было бы неэффективно. К счастью, существует эффективный алгоритм, основанный на сортировке, который может предоставить нам эту информацию, называемый AUC.

AUC означает «Площадь под кривой ROC». То есть AUC измеряет всю двумерную область под всей кривой ROC (например, интегральное исчисление) от (0,0) до (1,1). Пример можно увидеть на рисунке 5.

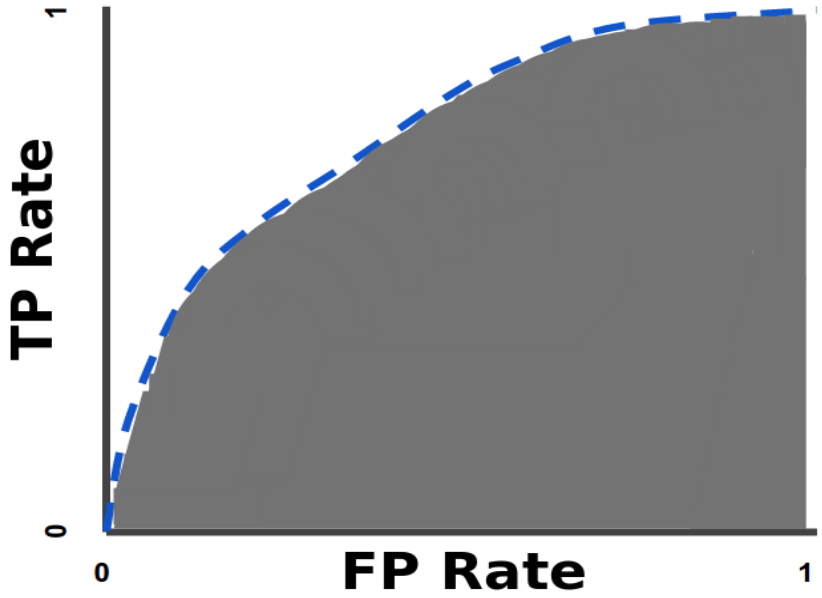

Рисунок 5 – AUC (площадь под кривой ROC)

AUC обеспечивает совокупный показатель производительности по всем возможным пороговым значениям классификации. Один из способов интерпретации AUC — это вероятность того, что модель ранжирует положительный прогноз выше, чем отрицательный прогноз.

Значение AUC варьируется от 0 до 1. Модель, чьи прогнозы на 100 % неверны, имеет значение AUC, равное 0,0; тот, чьи предсказания верны на 100%, имеет AUC 1,0.

AUC желательно использовать по следующим двум причинам:

1) AUC не зависит от масштаба. Он измеряет, насколько хорошо ранжируются прогнозы, а не их абсолютные значения.
2) AUC не зависит от порога классификации. Он измеряет качество прогнозов модели независимо от выбранного порога классификации.

Однако обе эти причины сопровождаются оговорками, которые могут ограничивать полезность AUC в определенных случаях использования:

1) Масштабная инвариантность не всегда желательна. Например, иногда нам действительно нужны хорошо откалиброванные по вероятности выходные данные, и AUC не скажет нам об этом.
2) Инвариантность порога классификации не всегда желательна. В тех случаях, когда существуют большие различия в стоимости ложноотрицательных и ложноположительных результатов, может быть важно минимизировать один тип ошибки классификации. Например, при обнаружении спама в электронной почте вы, вероятно, захотите свести к минимуму ложноположительные срабатывания (даже если это приведет к значительному увеличению ложноотрицательных результатов). AUC не является полезным показателем для этого типа оптимизации.

Обучение нашей модели досрочно заканчивается, если в течение 20 раундов обучения не было обнаружено улучшения метрики.

После выбора модели и значений гиперпараметров нужно будет разделить данные на тренировочные и данные для тестирования (валидации) модели, чтобы предотвратить перегрузку xgboost.

Разбивать данные мы будем следующим образом: первые взятые 20 лет будут использоваться для тренировки, 20 + 1 год будет использоваться для валидации и 20 + 2 год будет использоваться для тестирования.

Подключив наши тренировочные и валидационные сеты данных в модель xgboost, мы получаем окончательную валидацию AUC со значением в 0,78, которое можно увидеть на рисунке 6.

```
[164]   validation_0-auc:0.78196
```

Рисунок 6 – Значения метрики AUC

Для анализа вклада функций в предсказательную мощность мы можем использовать встроенный метод feature_importances_. Он, по сути, вычисляет долю времени, в течение которого функция появляется в дереве решений. Чем больше она появляется, тем больше вероятность того, что она будет сильным фактором точности прогнозирования. Значение анализа приведены на рисунке 7.

```
pd.Series(model.feature_importances_, index=X_train.columns).sort_values(ascending=False)
```

```
player_log_rank_diff                  0.619010
player_game_win_ratio_diff            0.108943
player_point_win_ratio_weighted_diff  0.080545
player_serve_win_ratio_diff           0.075001
player_rank_diff                      0.060340
player_return_win_ratio_diff          0.056161
dtype: float32
```

Рисунок 7 – Значения метрик модели при анализе с помощью feature_importances

Неудивительно, что самой существенной характеристикой, определяемой xgboost, является разница логарифмов рангов игроков.

Это также подтверждается ещё одним фактором, а именно важностью перестановки.

Рассмотрим детальнее вопрос о том, что представляет собой важность перестановки.

По сравнению с большинством других подходов важность перестановки:

1) можно быстро рассчитать,
2) она широко используется
3) соответствует свойствам, которые мы хотели бы иметь для меры важности признаков.

Важность перестановки использует модель не самым обычным способом, многие сначала находят это запутанным. Для понимания мы начнем с примера. Рассмотрим данные в следующем формате на рисунке 8:

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...	...	...	...
156	142	...	8
153	130	...	24

Рисунок 8 – Данные о человеке, доступные в возрасте, когда ему было 10 лет.

Мы хотим предсказать рост человека, когда ему исполнится 20 лет, используя данные, доступные в возрасте, когда человеку было 10 лет.

Наши данные включают полезные функции (рост в возрасте 10 лет), функции с небольшой прогностической силой (владение количеством носков), а также некоторые другие функции, на которых мы не будем акцентировать внимание в этом примере.

Важность перестановки рассчитывается после обучения модели. Таким образом, мы не будем менять модель или прогнозы, которые мы получим для заданного значения роста, количества носков и т. д.

Вместо этого мы зададим следующий вопрос: если я случайным образом перетасую один столбец проверочных данных, оставив целевой столбец и все остальные столбцы на месте, как это повлияет на точность прогнозов в этих уже перемешанных данных?

Случайное изменение порядка одного столбца должно привести к менее точным прогнозам, поскольку полученные данные больше не соответствуют ничему, наблюдаемому в реальном мире. Точность модели особенно страдает, если мы перемешиваем столбец, на который модель сильно опиралась при прогнозировании. В этом случае перетасовка роста в 10 лет вызовет ужасные последствия. Если бы вместо этого мы перетасовали столбец владения носками, полученные в результате прогнозы не пострадали бы так сильно.

Таким образом процесс выглядит следующим образом:

1) Получаем обученную модель.

2) Перемешиваем значения в одном столбце, делаем прогнозы, используя полученный набор данных. Используем эти прогнозы и истинные целевые значения, чтобы вычислить, насколько функция потерь пострадала от перетасовки. Это ухудшение производительности измеряет важность переменной, которую мы только что перетасовали.

3) Возвращаем данные в исходный порядок (отменив перетасовку с шага 2). Теперь повторяем шаг 2 со следующим столбцом в наборе данных, пока не рассчитаем важность каждого столбца.

Рассмотрим важность перестановки для метрик нашей модели на рисунке 9:

```
perm = PermutationImportance(model).fit(X_val, y_val)
eli5.show_weights(perm, feature_names = X_val.columns.tolist())
```

Weight	Feature
0.1689 ± 0.0195	player_log_rank_diff
0.0132 ± 0.0132	player_rank_diff
0.0098 ± 0.0062	player_game_win_ratio_diff
0.0039 ± 0.0100	player_point_win_ratio_weighted_diff
0.0039 ± 0.0050	player_return_win_ratio_diff
-0.0065 ± 0.0051	player_serve_win_ratio_diff

Рисунок 9 – Значения важности перестановки для метрик модели

Значения вверху являются наиболее важными характеристиками, а значения внизу имеют наименьшее значение.

Первое число в каждой строке показывает, насколько снизилась производительность модели при случайном перемешивании (в данном случае с использованием «точности» в качестве показателя производительности).

Иногда можно увидеть отрицательные значения важности перестановок. В этих случаях предсказания перетасованных (или зашумленных) данных оказывались более точными, чем реальные данные. Это происходит, когда случайность делает прогнозы на перетасованных данных более точными. Это чаще встречается с небольшими наборами данных, такими как в этом примере, потому что здесь больше места для удачи/случайности.

Важность перестановки, по сути, заключается в том, что она включает в себя перестановку в случайном порядке очередности измерений, а также в том, как она влияет на точность прогнозирования. Если точность резко снижается, то это хороший индикатор того, что данная функция была действительно важна, если она не сильно меняется, то эта функция, вероятно, не важна для вашей модели.

Таким образом проанализировав с помощью различных способов вклад метрик в обучение приходим к выводу, что наиболее влиятельной метрикой оказалась разница логарифмов рангов игроков.

Учитывая, что большая часть инфраструктуры уже заложена, делать прогнозы теперь относительно просто.

К примеру, для практического использования мы можем использовать список игроков, с помощью itertools создать множество всех перестановок между парами игроков и для каждого из них вычислить результат их матча. Таким образом каждый раз не нужно будет вычислять один и тот же результат.

При прогнозах особое внимание стоит уделять игрокам, которые не имели до этого большого количества серьезных матчей, по которым есть данные. Такие игроки вносят непредсказуемость, т. к. не всегда понятно, чего от них можно ожидать. В дальнейшем в качестве развития работы можно разработать использование одного из алгоритмов «холодного старта» для таких игроков.

Также можно углубить анализ и посчитать, к примеру, среднее значение вероятностей победы игрока в матчах со всеми его потенциальными соперниками.

Дальнейшим улучшением модели будет служить внедрение elo(Эло).

Для начало рассмотрим, что такое Эло.

Эло — это более совершенная рейтинговая система по сравнению с формулами рейтинга, используемыми ATP и WTA.

Принцип любой системы Эло заключается в том, что рейтинг каждого игрока является оценкой его силы, и каждый матч (или турнир) позволяет нам обновлять эту оценку. Если игрок выигрывает, его рейтинг повышается; если она проигрывает, он падает.

Отличие Эло заключается в определении величины, на которую должен увеличиваться или уменьшаться рейтинг. Учитываются две основные переменные: сколько матчей уже есть у игрока в системе (то есть насколько мы уверены в предматчевом рейтинге), и рейтинг соперника.

Если вы задумаетесь об этом на мгновение, то увидите, что эти две переменные являются хорошим приближением того, как мы обычно думаем о силе игрока. Чем больше мы уже знаем об игроке, тем меньше мы изменим свое мнение на основании одного матча. Проигрыш Новака Джоковича по круговой системе Доминику Тиму в Лондоне стал неожиданностью, но только самые скептично настроенные фанаты Джоковича увидели в этом результате катастрофу, которая должна существенно изменить нашу оценку его игровых способностей. Точно так же мы корректируем наше мнение в зависимости от рейтинга оппонента. Поражение от Тима разочаровывает, но поражение, скажем, от Марко Чеккинато вызывает большее беспокойство. Система Эло включает в себя эту естественную интуицию.

Традиционно игроку при входе в систему присваивается рейтинг Эло 1200 — до того, как появятся какие-либо результаты. Это число само по себе совершенно произвольно. Все, что имеет значение, — это разница между рейтингами игроков, поэтому, если мы начнем считать рейтинга каждого участника с 0, 100 или 888, конечный результат этих различий останется прежним.

На данный момент лучшими игроками ATP и WTA являются Рафаэль Надаль и Эшли Барти с 2203 и 2123 очками соответственно. Лучшие игроки часто находятся в этом диапазоне, а лучшие из лучших часто приближаются к 2500. Согласно последней версии алгоритма, пик Джоковича был 2470, а лучший результат Серены Уильямс — 2473.

Отметка в 2000 баллов — это хорошее практическое правило, позволяющее отделить элиту от остальных. На данный момент такие высокие рейтинги имеют шесть мужчин и семь женщин. 16 мужчин и 18 женщин имеют рейтинг Эло не ниже 1900, а рейтинг 1800 примерно эквивалентен месту в топ-50.

Как только мы присвоим каждому игроку единый пиковый рейтинг, вполне естественно начать сравнивать его по эпохам.

Можно сравнить по эпохам, как каждый игрок справлялся с его конкурентами. В 1990 году Хелена Сукова достигла рейтинга 2123 — точно такого же, как сегодня у Барти. Это не значит, что Сукова тогда была так же хороша, как сейчас Барти. Но это означает, что их показатели по сравнению с их сверстниками были одинаковыми. Второй эшелон игроков был значительно слабее тридцать лет назад, поэтому добиться такого рейтинга в каком-то смысле было проще. В то время рейтинг Суковой был хорош только для 11-го места, намного отставая от 2600 Штеффи Граф.

Таким образом, Elo не позволяет вам ранжировать игроков по эпохам, если вы не уверены, что уровень конкуренции был одинаковым, или если у вас нет другого способа решить эту проблему.

Связанный с этим вопрос - инфляция Эло, которая также может усложнить сравнения между эпохами. Каждый раз, когда играется матч, победитель и проигравший эффективно «торгуют» некоторыми из своих очков, поэтому общее количество рейтинговых очков Эло в системе не меняется. Однако каждый раз, когда в систему заходит новый игрок, общее количество очков увеличивается. И всякий раз, когда игрок уходит из игры, общее количество очков уменьшается.

Было бы неплохо, если бы сложения и вычитания компенсировали друг друга, но для многих соревнований, в которых используется Эло, этого не происходит. Добавления, как правило, перевешивают вычитания, поэтому рейтинг Эло со временем увеличивается. Следовательно количество очков в системе меняется со временем, по причинам, не связанным с силой топ-игроков.

Эло дает нам рейтинг для каждого игрока, и мы получаем представление о том, что мы можем и не можем делать с ними.

Одной из основных целей любой рейтинговой системы является прогнозирование исхода матчей, что Эло делает лучше, чем большинство других, включая рейтинги ATP и WTA. Единственным входом, необходимым для составления прогноза, является разница между рейтингами двух игроков, которую затем можно подставить в следующую формулу:

$$1 – (1/(1 + (10^((\text{разница})/400)))) \quad (3)$$

Если бы мы хотели спрогнозировать матч-реванш последнего матча финала Кубка Дэвиса, мы бы взяли рейтинги Эло Надаля и Дениса Шаповалова (2203 и 1947), нашли разницу (256) и подставили ее в формулу, получив результат 81,4% для Надаля на победу. Если бы мы использовали отрицательную разницу (-256), мы бы получили в результате 18,6%, для Шаповалова на победу.

В теннисе рейтинги и прогнозы должны сильно различаться в зависимости от покрытия.

Для каждого игрока нужно вести четыре отдельных рейтинга Эло: общий, только корт с твердым покрытием, только корт с грунтовым покрытием и только корт с травяным покрытием. Например, рейтинг Тима составляет 2066 в целом, 1942 на харде, 2031 на грунте и 1602 на траве.

(Поверхностные рейтинги, как правило, ниже: грунтовый рейтинг Тима занимает третье место, намного опережая всех, кроме Надаля и Джоковича.)

Рейтинги на одной поверхности говорят нам, как бы мы ранжировали игроков, если бы просто отбрасывали результаты на всех остальных поверхностях. Однако это неверно. Односторонние рейтинги не очень хороши для прогнозирования результатов матчей. Лучшим решением будет взять смесь 50/50 одноповерхностных и общих рейтингов. Если бы мы хотели предсказать шансы Тима в матче на грунтовом корте, мы бы использовали половинную смесь его общего рейтинга 2066 и его рейтинга на грунтовом корте 2031.

Однако стоит отметить, что не существует естественного закона, который диктует смесь рейтингов именно 50/50.

Спросите у фанатов тенниса, какие матчи турниров имеют большее значение для рейтингов — и вы получите длинный подробный список факторов, определяющих это значение. Например, финалы или олимпийские игры являются более важными, чем обычные личные встречи. Для таких матчей нужно вносить определенные корректировки.

Эло предусматривает такие корректировки. Коэффициент, обычно называемый «коэффициентом k», позволяет придавать больший вес определенным матчам. Это распространено в рейтингах Эло и для других видов спорта, например, при использовании более высокого коэффициента k для постсезонных игр, чем для игр регулярного сезона.

Обычно этот коэффициент полагают равным 10 для сильнейших игроков (рейтинг 2400 и выше), 20 (было 15) — для игроков с рейтингом меньше, чем 2400 и 40 (было 30) — для новых игроков (первые 30 партий с момента получения рейтинга ФИДЕ), а также для игроков до 18 лет, рейтинг которых ниже 2300.

В нашем случае этот коэффициент будет зависеть от числа выигранных и проигранных матчей игрока.

Когда игроки пропускают значительное количество времени, нужно снижать их рейтинг, а затем повышать коэффициент k на несколько матчей после их возвращения.

Эти шаги являются логическим продолжением системы Эло. Если игрок травмирован на несколько месяцев, мы никогда не знаем, чего ожидать, когда он вернется. Может быть, он так же силён, как и прежде; может быть, он уже на шаг медленнее. Возможно, что он быстро вернется в норму, но также он может и никогда полностью не вернуться в форму. Длительное отсутствие вызывает много вопросов. Игрок с травмой редко возвращается в лучшей форме, чем до ухода, в то время как многие игроки возвращаются хуже, что дает нам средний уровень производительности после травмы, который хуже, чем до её отсутствия.

Следовательно, когда игрок впервые возвращается, наша оценка должна составляться таким образом будто бы он играет немного хуже. Однако некоторым сильным ранним результатам следует придать больший вес – отсюда и более высокий коэффициент k. Коэффициент k отражает тот факт, что сразу после отсутствия игрока мы не так уверены в своей оценке, как обычно.

Алгоритм усложняется, но логика проста. По сути, это просто попытка выработать строгую версию утверждений вроде: «Я не знаю, насколько хорошо он будет играть, когда вернется, но я буду внимательно следить».

Одним из дополнительных преимуществ штрафа за отсутствие является то, что он противодействует естественной склонности Эло к завышению рейтингов. В то время как больше игроков входит в систему, чем выходит из нее, штраф удаляет некоторые очки, не перераспределяя их другим игрокам.

Самый простой способ сравнить рейтинговые системы — это показатель под названием «точность», который подсчитывает правильные прогнозы. В финале Кубка Дэвиса было 50 одиночных матчей, и Эло правильно выбрал победителя в 36 из них с рейтингом точности 72%. Рейтинг ATP правильно выбрал победителя (в том смысле, что игрок с более высоким рейтингом выиграл матч) в 30 из них с рейтингом точности 60%. В этом маленьком эксперименте Эло превзошел официальные рейтинги. Эло был также значительно лучше в течение всего сезона.

Лучшей метрикой для сравнения рейтинговых систем является оценка Брайера, которая учитывает достоверность каждого прогноза. Ранее мы видели, что Эло дает Надалю шанс победить Шаповалова в 81,4%. Если Надаль в конечном итоге выиграет, 81,4% — это более хороший прогноз, чем, скажем, 65%, но это хуже прогноза 90%. Оценка Брайера представляет собой квадрат расстояния между прогнозом (81,4%) и результатом (0% или 100%, в зависимости от победителя) и усредняет эти числа для всех прогнозируемых матчей. Он вознаграждает близкие к 100 прогнозы, которые оказываются верными, но поскольку он использует квадрат расстояния, он сурово наказывает близкие к 100, но неверные прогнозы.

Более интуитивный способ понять, к чему ведет счет Брайера, — представить, что Надаль и Шаповалов играют 100 матчей подряд. (Или, более точно, но менее интуитивно, представьте, что 100 одинаковых Надалей играют одновременные матчи против 100 одинаковых Шаповаловых.) Прогноз 81,4% означает, что мы ожидаем, что Надаль выиграет в 81 матче. Если Надаль в итоге

выиграет 90, прогноз не был достаточно благоприятным для него. Мы никогда не получим 100 таких одновременных матчей, но у нас есть тысячи отдельных матчей, многие из которых имеют одинаковые прогнозы, например, 60-процентный шанс на победу фаворита. Оценка Брайера объединяет все эти пары «прогноз-результат» и выдает число, сообщающее нам, какой у нас результат.

Трудно прогнозировать результат отдельных теннисных матчей. Любая система, какой бы сложной она ни была, в большинстве случаев будет ошибаться. Во многих случаях «правильный» прогноз едва ли лучше, чем отсутствие прогноза вообще, если данные свидетельствуют о том, что конкуренты равны. Таким образом, «точность» имеет ограниченное применение — более важно иметь правильную степень уверенности в победе, чем просто выбирать победителей.

Рейтинги Эло имеют гораздо более низкие (лучшие) баллы Бриера, чем прогнозы, полученные на основе рейтингов ATP и WTA.

Оценка Брайера также является мерой, которая говорит нам о том, является ли определенная корректировка, такая как смешивание поверхностей, отсутствие травм или тип турнира, улучшением системы. Штраф за травму снижает оценку Бриера в общем наборе прогнозов Эло, поэтому мы сохраняем эту корректировку. А уменьшение k-фактора для матчей первого раунда не имеет никакого эффекта, поэтому мы не используем эту корректировку.

Теперь зная информацию об оценке Эло мы сможем внедрить её в качестве метрики в нашу модель. Проведем предрасчёт рейтинга Эло для каждого из спортсменов. Затем проведем очистку данных, такую же как проводили для прошлой модели, добавив рейтинг Эло в качестве дополнительного параметра для обучения.

Обучим нашу модель аналогично тому, как делали это до этого и посмотрим на результаты.

Проанализировав вклад функций в обучение, получим следующий результат на рисунке 10.

```
pd.Series(model.feature_importances_, index=X_cols_subset_1).sort_values(ascending=False)
```

```
player_old_elo_diff                  0.440281
player_win_weight_diff               0.120508
player_log_rank_diff                 0.079821
player_rank_diff                     0.079811
player_game_win_ratio_diff           0.078194
player_point_win_ratio_weighted_diff 0.068934
player_return_win_ratio_diff         0.066744
player serve win ratio diff          0.065708
```

Рисунок 10 – Значения метрик модели с Эло при анализе с помощью feature_importances

Как мы видим, Эло сыграло очень важно роль в обучении модели в несколько раз обойдя остальные метрики.

Рассмотрим теперь важность перестановок.

Weight	Feature
0.1173 ± 0.0320	player_old_elo_diff
0.0528 ± 0.0242	player_win_weight_diff
0.0291 ± 0.0162	player_log_rank_diff
0.0213 ± 0.0118	player_game_win_ratio_diff
0.0157 ± 0.0100	player_rank_diff
-0.0016 ± 0.0094	player_point_win_ratio_weighted_diff
-0.0024 ± 0.0107	player_return_win_ratio_diff
-0.0039 ± 0.0141	player_serve_win_ratio_diff

Рисунок 11 – Значения важности перестановки для метрик модели с Эло

И здесь Эло показывает прекрасный результат, обойдя остальные метрики.

Таким образом исходя из анализа метрик можно утверждать, что логарифм ранга игрока оказывает большее влияние на обучение, чем просто ранг игрока. Но Эло в свою очередь обходит как ранга игрока, так и логарифм ранга игрока по важности влияния на обучение.

Возьмём определенный список игроков, между которыми были сыграны матчи, и они известны. Пример части игроков приведен на рисунке 12.

```
    'Rafael Nadal',
    'Hugo Dellien',
    'Federico Delbonis',
    'Joao Sousa',
    'Christopher Eubanks',
    'Peter Gojowczyk',
    'Jozef Kovalik',
    'Pablo Carreno Busta',
    'Nick Kyrgios',
    'Lorenzo Sonego',
```

Рисунок 12 – Пример части игроков взятых для валидации

Введем вручную результаты этих матчей. Пример части результатов матчей приведен на рисунке 13.

```
['Rafael Nadal', 'Hugo Dellien', 1],
['Hugo Dellien', 'Rafael Nadal', 0],
['Federico Delbonis', 'Joao Sousa', 1],
['Joao Sousa','Federico Delbonis',  0],
['Christopher Eubanks', 'Peter Gojowczyk', 0],
['Peter Gojowczyk','Christopher Eubanks', 1],
['Jozef Kovalik', 'Pablo Carreno Busta', 0],
['Pablo Carreno Busta','Jozef Kovalik', 1],
```

Рисунок 13 – Пример части результатов матчей взятых для валидации

Таким образом мы имеем список игроков и известные результаты матчей, на которых мы сможем провалидировать качество наших обученных моделей.

Для валидации предсказаний моделей будем использовать следующие метрики accuracy, AUC, logloss.

Метрика AUC была описана выше.

Рассмотрим две другие метрики.

Логарифмическая потеря – метрика оценки эффективности Модели Бинарной классификации.

Для того, чтобы разобраться в том, что это, обратимся к концепции бинарной классификации. Такой алгоритм сначала предсказывает вероятность того, что Наблюдение будет отнесено к классу 1, а затем причисляет его к одному из двух классов (1 или 0) на основе того, пересекла ли вероятность пороговое значение, которое устанавливается по умолчанию равным 0,5. На рисунке 14 приведены результаты классификации email (1 – "спам").

ID	ТЕКСТ ПИСЬМА	РЕАЛЬНЫЙ КЛАСС	ПРЕДСКАЗАННАЯ ВЕРОЯТНОСТЬ	ПРЕДСКАЗАННЫЙ КЛАСС
64247	Попробуйте бесплатную защиту от телефонного спама и выиграйте один	1	0,95	1
26591	Привет! Прости пожалуйста что пропал – очень много было работы. В целом, ничего не меняет	0	0,2	0
70282	Здравствуйте, Елена Александровна! Запрошенная вами справка "О наличии счетов"	0	0,65	1

Рисунок 14 – Результаты классификации email

Итак, прежде чем предсказывать класс записи, модель должна спрогнозировать вероятность того, что запись будет отнесена к классу 1. Помните, что именно от этой вероятности предсказания записи данных зависит значение логарифмической потери.

Логарифмическая потеря указывает, насколько близка вероятность предсказания к соответствующему истинному значению (0 или 1 в случае бинарной классификации). Чем больше прогнозируемая вероятность отклоняется от фактического значения, тем выше значение логарифма потерь. Формула расчета Log-Loss будет приведена чуть позже.

Для примера рассмотрим задачу классификации электронных писем. Давайте представим спам как класс 1, а класс "нормальных" писем как 0. Давайте изучим настоящее спам-письмо (фактическое значение равно 1) и статистическую модель, которая классифицирует это письмо как спам с вероятностью 1. Поскольку вероятность предсказания равна почти 1, то и разность между предсказанной вероятностью и фактическим классом равна почти 0. Нулю равен, следовательно, и логарифм этой разности. Пример такого предсказания приведен на рисунке 15:

ID	ТЕКСТ ПИСЬМА	РЕАЛЬНЫЙ КЛАСС	ПРЕДСКАЗАННАЯ ВЕРОЯТНОСТЬ	РАЗНОСТЬ	ЛОГАРИФМИЧЕСКАЯ ПОТЕРЯ
1318	Попробуйте бесплатную защиту от телефонного спама и выиграйте один из пяти iPhone 12 на 256 ГБ...	1	1,00	0,00	0

Рисунок 15 – Результаты предсказания с вероятностью 1

Рассмотрим на рисунке 16 еще одно спам-письмо, классифицированное как спам с вероятностью 0,9. Вероятность прогноза модели на 0,1 отличается от фактического значения 1, и, следовательно, значение логарифмической потери больше нуля (равно 0,105).

ID	ТЕКСТ ПИСЬМА	РЕАЛЬНЫЙ КЛАСС	ПРЕДСКАЗАННАЯ ВЕРОЯТНОСТЬ	РАЗНОСТЬ	ЛОГАРИФМИЧЕСКАЯ ПОТЕРЯ
64247	Привет. Надеюсь, у тебя было время посмотреть, какие услуги по строительству предлагает моя компания...	1	0,90	0,10	0,105

Рисунок 16 – Результаты предсказания с вероятностью 0.9

А теперь давайте посмотрим на рисунке 17 на обычное электронное письмо. Модель классифицирует его как спам с вероятностью 0,2, то есть считает нормальным письмом (при условии, что порог по умолчанию равен 0,5). Абсолютная разница между вероятностью предсказания и фактическим значением, равным 0 (так как это нормально), составляет 0,2, что больше, чем то, что мы наблюдали в предыдущих двух наблюдениях. Значение логарифма потерь, связанное с прогнозом, составляет 0,223.

ID	ТЕКСТ ПИСЬМА	РЕАЛЬНЫЙ КЛАСС	ПРЕДСКАЗАННАЯ ВЕРОЯТНОСТЬ	РАЗНОСТЬ	ЛОГАРИФМИЧЕСКАЯ ПОТЕРЯ
2101	Скажи, когда тебе необходимо настроить рабочий ПК. У меня есть...	0	0,20	-0,20	0,223

Рисунок 17 – Результаты предсказания с вероятностью 0.2

Обратите внимание, как теперь значение Log-Loss худшего прогноза (удаленного от фактического значения) выше, чем у лучшего прогноза (ближе к фактическому значению).

Теперь предположим, что существует набор из 5 различных спам-писем, прогнозируемых с широким диапазоном вероятностей 1.0, 0.7, 0.3, 0.009 и 0.0001. Обученная статистическая модель неидеальна и, следовательно, выполняет (действительно) плохую работу по последним трем наблюдениям (классифицирует их как нормальные, поскольку значения вероятности ближе к 0, чем к 1). На рисунке 18 видно, что значение логарифмических потерь экспоненциально возрастает по мере того, как растет разность между реальным классом и предсказанной вероятностью:

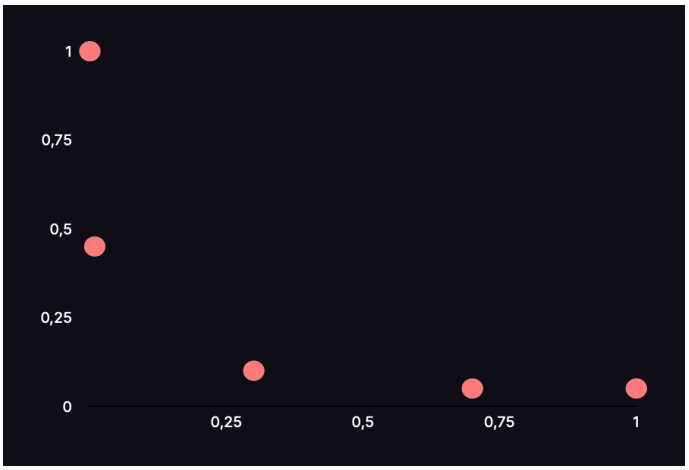

Рисунок 18 – Значения логарифмических потерь при прогнозах с разной вероятностью

Если мы построим график логарифмических потерь для перечня спам-писем со всеми возможными видами вероятностей, график будет выглядеть как на рисунке 19.

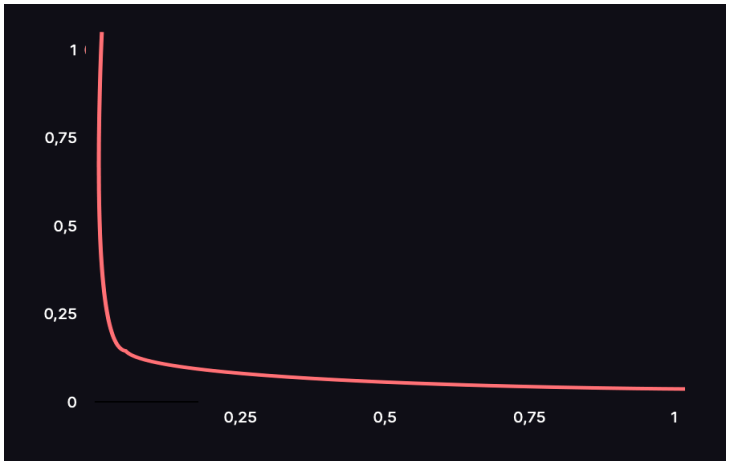

Рисунок 19 – Значения логарифмических потерь при всех возможных вероятностях прогнозов

В случае с нормальными письмами график будет зеркальным отображением приведенного выше и будет изображаться как на рисунке 20.

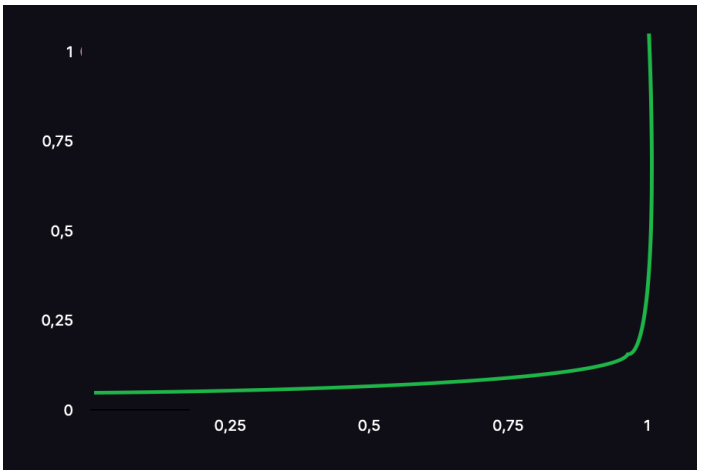

Рисунок 20 – Значения логарифмических потерь при всех возможных вероятностях прогнозов для нормальных писем

Подводя итог, можно сказать, что чем дальше вероятность предсказания от фактического значения, тем выше значение логарифмических потерь. При обучении модели классификации мы хотели бы, чтобы наблюдение предсказывалось с вероятностью, максимально приближенной к фактическому значению (0 или 1). Следовательно, Log-Loss – хороший выбор в качестве функции

потери для обучения и оптимизации. Чем дальше вероятность предсказания от ее истинного значения, тем выше штраф.

Теперь, когда мы понимаем логику, лежащую в основе метрики, мы можем посмотреть на конечную формулу на рисунке 21.

$$Log - Loss = -[y_i \times \ln(p_i) + (1 - y_i) \times ln(1 - p_i)]$$

y_i– истинный класс наблюдения,

p_i – предсказанная вероятность

Рисунок 21 – Формула логарифмических потерь

Чтобы оценить модель в целом, вычисляется среднее арифметическое логарифмических потерь всех наблюдений. Модель с совершенными предсказаниями имеет логарифм потерь, равным нулю. Другими словами, идеальная модель предсказывает вероятность каждого наблюдения как фактическое значение.

Логарифмическая потеря для бинарной классификации – тоже, что и среднеквадратическая ошибка для регрессии. Обе метрики показывают, насколько хороши или плохи результаты прогнозов, указывая на дистанцию между прогнозом и фактическим значением.

Логарифмические потери можно рассчитать с помощью SkLearn. Для начала импортируем функцию:

import matplotlib.pyplot as plt

Применим функцию "на бегу", передав аргументы-списки:

log_loss(["спам", "нормальное письмо", "нормальное письмо", "спам"],

[[.1, .9], [.9, .1], [.8, .2], [.35, .65]])

Система здесь уже вынесла свой вердикт, и во втором списке находятся, как можно догадаться, пары значений, описывающих вероятности "спама" и "нормального письма". Функция сама определяет, что первый элемент внутреннего списка – число 0,1, описывает вероятность письма быть нормальным письмом, а второй – соответственно, спамом, и применив такой паттерн ко всем остальным парам значений, вычисляет разность между реальным классом и предсказанной вероятностью. Следуя формуле, описанной выше, она находит значение Log-Loss для каждого наблюдения и усредняет полученный результат получая в итоге 0.21616.

Accuracy — это показатель, который описывает общую точность предсказания модели по всем классам. Это особенно полезно, когда каждый класс одинаково важен. Он рассчитывается как отношение количества правильных прогнозов к их общему количеству.

Метрика acc содержит результат деления суммы True Positive и True Negative прогнозов на количество всех прогнозов. Таким образом, accuracy, равная 0.5714, означает, что модель с точностью 57,14% делает верный прогноз.

В модуле sklearn.metrics есть функция precision_score(), которая также может вычислять accuracy. Она принимает в качестве аргументов достоверные и предсказанные метки.

Стоит учесть, что метрика accuracy может быть обманчивой. Один из таких случаев — это несбалансированные данные. Предположим, у нас есть всего 600 единиц данных, из которых 550 относятся к классу Positive и только 50 — к Negative. Поскольку большинство семплов принадлежит к одному классу, accuracy для этого класса будет выше, чем для другого.

Если модель сделала 530 правильных прогнозов из 550 для класса Positive, по сравнению с 5 из 50 для Negative, то общая accuracy равна (530 + 5) / 600 = 0.8917. Это означает, что точность модели составляет 89.17%. Полагаясь на это значение, вы можете подумать, что для любой выборки (независимо от ее класса) модель сделает правильный прогноз в 89.17% случаев. Это неверно, так как для класса Negative модель работает очень плохо.

Для вычисления значения метрик возьмём следующие модели: нашу первую модель, модель в которой мы ввели новую метрику эло для анализа и “наивную” модель, где вероятность победы каждого из игроков равна 0.5.

Получаем следующие результаты, которые можем увидеть на рисунке 22

	logloss	AUC	accuracy
model	0.529107	0.800347	0.712598
elo_model	0.515281	0.825118	0.763780
true_naive	9.654653	0.720486	0.720472

Рисунок 22– Результаты валидации моделей

На основе результатов можно сказать следующее: первичная модель и модель на основе Эло показали сравнимые хорошие показатели. Однако Эло модель по всем показателям обошла первичную.

В то же время модель, всегда отдающая результат 0.5 сильно уступила лишь в logloss метрике. Это подчеркивает важность выбора метрики для валидации модели обучения. Некоторые метрики являются неинформативными, в другое время как остальные помогают качественно провалидировать прогнозы полученной модели.

Дальнейшими перспективами развития данной работы можно назвать вычисление оптимального размера скользящего окна по матчам для агрегирования метрик, оптимальной длительности обучения модели и обучения модели для различных типов кортов.

В ходе данной работы было получено понимание того, что выбор правильных метрик для обучения зачастую важнее подбора гиперпараметров для обучения.

А также выявлена необходимость подбора правильной метрики для валидации, чтобы она была информативной.

Список использованных источников:

1. Forecasting the results of the world winner universiade [Electronic Resource] /ScienceReview. – Mode of access: https://science-engineering.ru/ru/article/view?id=1238. Date of access: 07.04.2023.

2. ATP Tennis Rankings, Results, and Stats [Electronic Resource] / JeffSackmann. – Mode of access: https://github.com/JeffSackmann/tennis_atp. Date of access: 07.04.2023.

3. Deuce [Electronic Resource] / Skoval. – Mode of access: https://github.com/skoval/deuce. Date of access: 07.04.2023.

4. Introduction to R [Electronic Resource] / Ahmedushka7. – Mode of access: https://ahmedushka7.github.io/R/scripts/hse_data_analysis/sem_1/introduction_to_R.html. Date of access: 07.04.2023.

5. Parsing data from sites [Electronic Resource] / Ringostat. – Mode of access: https://blog.ringostat.com/ru/parsing-dannyh-s-saytov-chto-eto-i-zachem-on-nuzhen/. Date of access: 07.04.2023.

6. Nuances of Programming [Electronic Resource] / Medium. – Mode of access: https://medium.com/nuances-of-programming. Date of access: 07.04.2023.

7. XGBoost Model [Electronic Resource] / Techcave. – Mode of access: https://techcave.ru/posts/81-sozdaem-pervuyu-xgboost-model-na-python-s-ispolzovaniem-scikit-learn.html. Date of access: 07.04.2023.

8. Permutation importance [Electronic Resource] / Kaggle. – Mode of access: https://www.kaggle.com/code/dansbecker/permutation-importance/tutorial. Date of access: 07.04.2023.

9. ROC and AUC metrics [Electronic Resource] / Google. – Mode of access: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Date of access: 07.04.2023.

10. Elo rating ranges [Electronic Resource] / TennisAbstract. – Mode of access: http://www.tennisabstract.com/blog/2019/12/03/an-introduction-to-tennis-elo/. Date of access: 07.04.2023.

11. Numpy [Electronic Resource] / PythonWorld. – Mode of access: https://pythonworld.ru/numpy/1.html. Date of access: 07.04.2023.

12. Log-loss Metric [Electronic Resource] / Helenkapatsa. – Mode of access: https://www.helenkapatsa.ru/logharifmichieskaia-potieria. Date of access: 07.04.2023.

13. Accuracy Metric [Electronic Resource] / PythonRu. – Mode of access: https://pythonru.com/baza-znanij/metriki-accuracy-precision-i-recall. Date of access: 07.04.2023.

UDC

NEURAL NETWORK APPROACH TO PREDICTION OF SPORTS TENNIS DATA

Kharkevich A.P.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Rykova O.V. – PhD in Physics and Mathematics

Annotation. In this paper, we consider the main stages of obtaining a ready-made neural network for predicting sports tennis data, analyzing the quality of training, and then optimizing the model.

Keywords. Python, pandas, xgboost, neural network, predicting results, tennis.