Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Научная работа

на тему

Нейросетевой подход к прогнозированию спортивных теннисных данных

Студент гр. 953504

А.П. Харкевич

Руководитель преподаватель кафедры информатики

Н.Л.Боброва

Минск 2022

**ОГЛАВЛЕНИЕ**

|  |  |
| --- | --- |
| [ВВЕДЕНИЕ](#page3) | [3](#page3) |
| 1. ИСТОРИЧЕСКАЯ СПРАВКА | 4 |

1. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ КОМПЬЮТЕРНОГО ЗРЕНИЯ 11

|  |  |
| --- | --- |
| 3. ОСНОВНЫЕ ЗАДАЧИ КОМПЬЮТЕРНОГО ЗРЕНИЯ | 15 |
| 3.1 OBJECT DETECTION | 15 |
| 3.2 SEMANTIC SEGMENTATION | 16 |
| 4. ПОДХОДЫ К РЕШЕНИЮ ЗАДАЧИ OBJECT DETECTION | 15 |
| 4.1 BRUTE FORCE | 15 |
| 4.2 R-CNN 13 | 16 |
| 4.3 FAST R-CNN 15 | 15 |
| 4.4 FASTER R-CNN 15 | 16 |
| 4.5 YOLO'15 | 15 |
| 4.6 СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОДХОДОВ | 16 |

1. МЕТОДЫ И АРХИТЕКТУРЫ РЕШЕНИЯ ЗАДАЧИ SEMANTIC

|  |  |
| --- | --- |
| SEGMENTATION | 15 |
| 5.1 BRUTE FORCE | 35 |
| 5.2 HOUR GLASS ARCHITECTURE | 16 |
| 5.3 U-NET 15 | 15 |
| ДАЛЬНЕЙШИЕ ПУТИ РАЗВИТИЯ ОБЛАСТИ | 20 |
| ЗАКЛЮЧЕНИЕ | 20 |
| СПИСОК ИСТОЧНИКОВ |  |

**ПЕРЕЧЕНЬ ОПРЕДЕЛЕНИЙ И СОКРАЩЕНИЙ**

Git (читается как «гит») — это система контроля версий, которая помогает отслеживать историю изменений в файлах. Git используют программисты для совместной работы над проектами.

Репозиторий — это все файлы, находящиеся под контролем версий, вместе с историей их изменения и другой служебной информацией.

Репозиторий Git можно создать, либо выбрав любую папку на компьютере, либо клонировав себе существующий репозиторий, например у работодателя.

GitHub — это самый популярный сайт для хранения git-репозиториев и работы с ними. Также GitHub является крупнейшей площадкой для размещения проектов с открытым исходным кодом. Для просмотра и загрузки общедоступных репозиториев не требуется ни регистрации, ни оплаты аккаунта

.

Датасет – это обработанная и структурированная информация в табличном виде. Строки такой таблицы называются объектами, а столбцы – признаками.

CSV-файлы (файлы данных с разделителями-запятыми) — это файлы особого типа, которые можно создавать и редактировать в Excel. В CSV-файлах данные хранятся не в столбцах, а разделенные запятыми. Текст и числа, сохраненные в CSV-файле, можно легко переносить из одной программы в другую.

Jupyter-ноутбук — это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. В такой среде разработки можно, например, написать функцию и сразу проверить её работу, без запуска программы целиком.

R — язык программирования для статистической обработки данных и работы с графикой, но в то же время это свободная программная среда с открытым исходным кодом. R применяется везде, где нужна работа с данными. Это не только статистика в узком смысле слова, но и «первичный» анализ (графики, таблицы), и продвинутое математическое моделирование.

RStudio — свободная среда разработки программного обеспечения с открытым исходным кодом для языка программирования R, который предназначен для статистической обработки данных и работы с графикой.

Devtools – одна из библиотек языка R. Цель devtools — облегчить вашу жизнь как разработчика пакетов, предоставляя функции R, которые упрощают многие распространенные задачи. Пакеты R на самом деле очень просты, и с правильными инструментами будет проще использовать структуру пакета, чем нет. Разработка пакетов в R может показаться пугающей, но devtools делает все возможное, чтобы сделать ее максимально удобной.

Парсинг — это процесс автоматического сбора данных и их структурирования. Парсинг обычно применяют, когда нужно быстро собрать большой объем данных. Его выполняют с помощью специальных сервисов — парсеров. Парсеры  «обходят» сайт и собирают данные, которые соответствуют заданному условию.

Очистка данных

Очистка данных позволяет исправить ошибки и заполнить недостающие данные, что является шагом к обеспечению их качества. После того как вы получили чистые данные, вам необходимо преобразовать их в соответственный читаемый формат. Этот процесс может включать изменение форматов полей, таких как форматы дат и валюты, изменение соглашений об именовании, а также корректировку значений и единиц измерения для обеспечения их соответствия.

Регуля́рные выраже́ния (англ. regular expressions) — формальный язык, используемый в компьютерных программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters). Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

Регулярные выражения используются некоторыми текстовыми редакторами и утилитами для поиска и подстановки текста. Например, при помощи регулярных выражений можно задать шаблоны, позволяющие:

найти все последовательности символов «кот» в любом контексте, как то: «кот», «котлета», «терракотовый»;

найти отдельно стоящее слово «кот» и заменить его на «кошка»;

найти слово «кот», которому предшествует слово «персидский» или «чеширский»;

убрать из текста все предложения, в которых упоминается слово кот или кошка.

Регулярные выражения позволяют задавать и гораздо более сложные шаблоны поиска или замены.

Результатом работы с регулярным выражением может быть:

проверка наличия искомого образца в заданном тексте;

определение подстроки текста, которая сопоставляется образцу;

определение групп символов, соответствующих отдельным частям образца.

Если регулярное выражение используется для замены текста, то результатом работы будет новая текстовая строка, представляющая собой исходный текст, из которого удалены найденные подстроки (сопоставленные образцу), а вместо них подставлены строки замены (возможно, модифицированные запомненными при разборе группами символов из исходного текста). Частным случаем модификации текста является удаление всех вхождений найденного образца — для чего строка замены указывается пустой.

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

**ВВЕДЕНИЕ**

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.

**НАЧАЛО РАБОТЫ**

Получение готовой нейросети для прогнозирования спортивных данных будет состоять в нескольких этапах:

1. Нахождение данных с матчами и спортсменами
2. Очистка данных
3. Построение нескольких моделей нейронной сети
4. Их обучение на части данных
5. Валидация моделей на другой части данных и на реальных данных
6. Сравнительный анализ моделей
7. Выбор наилучшей модели для задачи спортивного прогнозирования

Рассмотрим подробнее каждый из этих этапов.

**НАХОЖДЕНИЕ ДАННЫХ С МАТЧАМИ И СПОРСТМЕНАМИ**

На первом этапе необходимо произвести поиск нужно нам датасета.

После проведенного поиска было обнаружено, что для текущей задачи хорошо подойдёт гитхаб 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.

Одна 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 турниры 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. Одна стоит отметить, что для более узконаправленной задачи он может являться более гибким.

При рассмотрении второго варианта был найден гитхаб 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.

**ОЧИСТКА ДАННЫХ**

Приступим к подготовке данных. Для неё мы напишем скрипт на питоне используя библиотеку 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. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми 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 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 «родных» категориальных типов данных.

Пакет 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:

* Объект *DataFrame* для манипулирования индексированными массивами двумерных данных
* Инструменты для обмена данными между структурами в памяти и файлами различных форматов
* Встроенные средства совмещения данных и способы обработки отсутствующей информации
* Переформатирование наборов данных, в том числе создание сводных таблиц
* Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных
* Вставка и удаление столбцов данных
* Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *split-apply-combine*).
* Слияние и объединение наборов данных
* Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности
* Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее

Библиотека оптимизирована для высокой производительности, наиболее важные части кода написаны на [Cython](https://ru.wikipedia.org/wiki/Cython" \o "Cython) и [Си](https://ru.wikipedia.org/wiki/C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

Т. к. количество турниров достаточно большое и многие турниры уже не актуальны, т.к. 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 временно 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 и на разных кортах разные игроки будут показывать разные результаты. Мы же взяли самый распространенный тип, чтобы не вносить лишние помехи.

Следующим шагом мы переименуем названия колонок на более понятные, чтобы с ним было более удобно 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 выигравшей и проигравшей стороны 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 брэйк-поинт был выигран одним из игроков. Мы считаем также число удачных вторых подач игрока. Считаем мы также и такие метрики, как сколько подач было выигран игроков 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 будет 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 сыграет лучший из 3-х матчей против Кирьоса и обыграет его 6-4, 7-5. Всего Федерер выиграл 13 партий. Если Джокович выиграет эпическую 5-ю партию у Нишикори 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, так как мы сравниваем лучший матч из 3 партий с лучшим из 5. 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.

Для этого 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 + число выигранных вторых подач) / (общее число первых подач игрока + общее число вторых подач игрока + общее число двойных ошибок игрока).

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

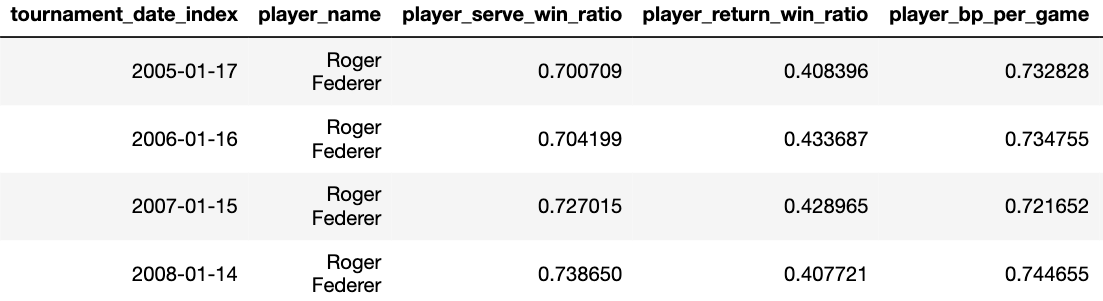
Третьей и четвертой метриками являются среднее число брэйкпоинтов за игру и процент реализации брэйкпоинта.

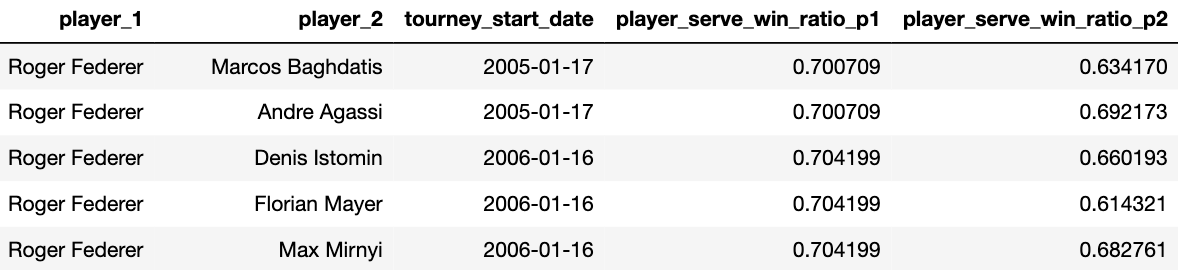
Далее пятой, шестой и седьмой метриками будут процент выигранных игр и процент выигранных очков, а также разницу между этими процентами.

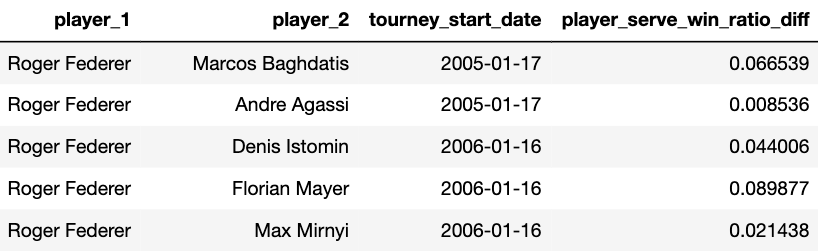
Далее считаем логарифм ранга игрока и с помощью экспоненциальной функции от ранга его противника считаем вес его победы.

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

Следующим важным шагом будет суммирование статистики по данному игроку за предыдущее х количество матчей.

Вот как будет выглядеть часть результатов для Роджера Федерера:  


Эти показатели должны быть объединены с данными матча, ключами, которые будут однозначно идентифицировать нашу группу данных, будет дата турнира и имя игрока как для player\_1, так и для player\_2:  


Заметьте, что у Федерера, его сводная статистика до данного турнира остаётся неизменной. Статистика для player\_2 также останется постоянной, но он будет в паре с разными игроками, которые влияют на изменение этих параметров в каждом матче. 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.  


Для суммирования статистик перейдём от метрик по матчам к метрикам в разрезе турниров.

В разрезе турнира 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 х 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). Так как это отрицательное число, то мы, естественно, ожидаем, что шансы будут слегка уравновешены с Зверем (все остальные равны). 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.

**ПОСТРОЕНИЕ НЕСКОЛЬКИХ МОДЕЛЕЙ НЕЙРОННОЙ СЕТИ**

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. На инфографике можно просмотреть эволюцию таких алгоритмов.

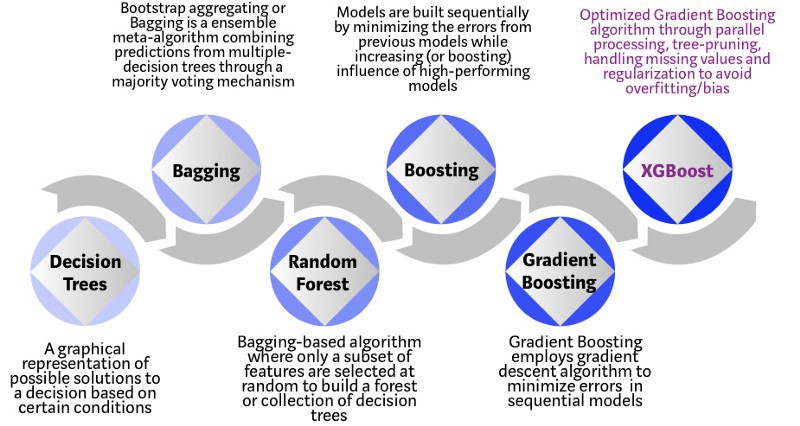


Рис.1 Эволюция алгоритмов, использующих деревья поиска решений

XGBoost разрабатывался как исследовательский проект Вашингтонского Университета. [Tianqi Chen и Carlos Guestrin](https://arxiv.org/pdf/1603.02754.pdf" \t "_blank) 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](https://github.com/dmlc/xgboost/" \t "_blank).

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 интуиция для построения следующего поколения алгоритмов, основывающихся на деревьях, не так уж и проста. 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:** Считайте его градиентным бустингом на стероидах (не зря, ведь он называется экстремальным градиентным бустингом). 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.

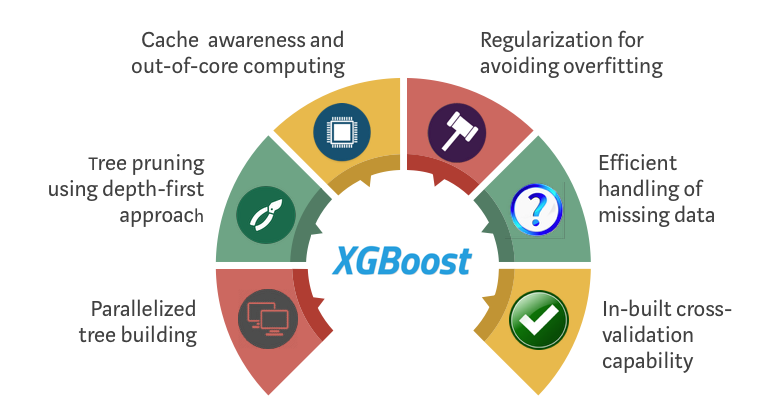


Рис.2 Как XGBoost оптимизирует стандартный алгоритм GBM

**Системная оптимизация:**

1. **Параллелизация:** В XGBoost построение деревьев основано на [параллелизации](http://zhanpengfang.github.io/418home.html" \t "_blank). 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 узорами разреженности.
3. [**Метод взвешенных квантилей:**](https://arxiv.org/pdf/1603.02754.pdf) 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.

Для примера был использован датасет Scikit-learn “[Make\_Classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html" \t "_blank)” для того, чтобы создать миллион точек с 20 признаками (из них 2 информативных и 2 излишних). Было протестировано несколько алгоритмов: логистическая регрессия, случайный лес, стандартный градиентный бустинг и XGBoost.

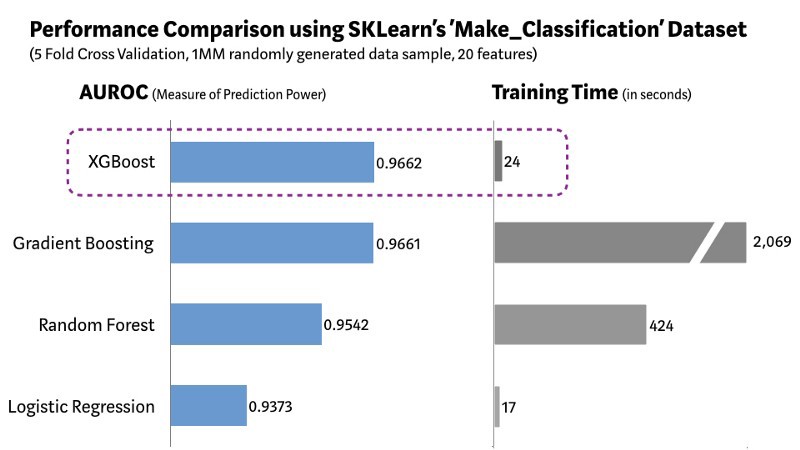


Рис.3 Сравнение производительности XGBoost и других алгоритмов на вышеупомянутом датасете

Как можно увидеть на графике выше, модель с XGBoost может похвастаться лучшей комбинацией “производительность-время обучения” среди других алгоритмов. Другие научные методы [измерения производительности](https://github.com/szilard/benchm-ml) показали похожий результат. Поэтому неудивительно, что XGBoost достаточно часто используется в соревнованиях по анализу данных.

**Значит, можно всё время использовать XGBoost?**

Когда дело доходит до машинного обучения (или даже жизни), не существует универсальных решений. Как специалисты по анализу данных, мы должны рассмотреть всевозможные алгоритмы для определённой задачи, чтобы выявить лучший из них. Но этого недостаточно. Необходимо настроить алгоритм путём подбора гиперпараметров. Также немаловажным в выборе алгоритма является его сложность, понятность и лёгкость имплементации. Именно здесь машинное обучение переходит грань между наукой и искусством, здесь происходит волшебство.

**ИХ ОБУЧЕНИЕ НА ЧАСТИ ДАННЫХ**

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

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 данных проверки была выбрана AUC, что отлично подходит для бинарной классификации.

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

**ROC-кривая**

**Кривая ROC (кривая** **рабочих характеристик приемника** ) представляет собой график, показывающий эффективность модели классификации при всех пороговых значениях классификации. Эта кривая отображает два параметра:

* Истинная положительная скорость
* Ложноположительный результат

**True Positive Rate** ( **TPR** ) является синонимом отзыва и поэтому определяется следующим образом:

TPR=TPTP+FN

**Ложноположительный показатель** ( **FPR** ) определяется следующим образом:

FPR=FPFP+TN

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 следующем рисунке показана типичная кривая ROC.

Изображение выглядит как текст

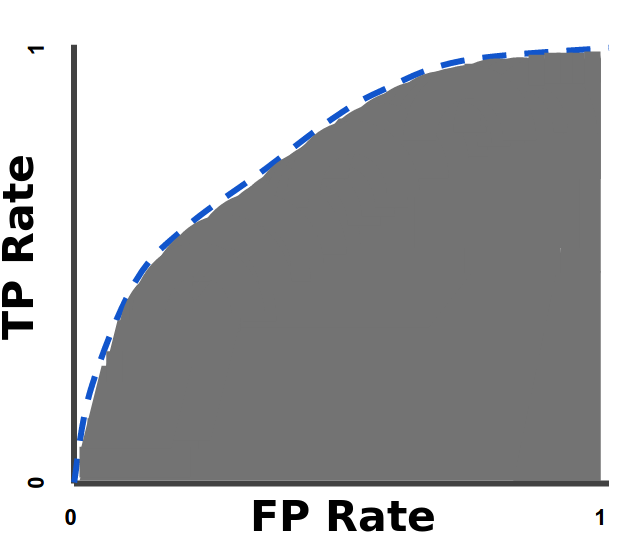
Автоматически созданное описание

**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: площадь под кривой ROC**

**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).



**Рисунок 5. AUC (площадь под ROC-кривой).**

AUC обеспечивает совокупный показатель производительности по всем возможным пороговым значениям классификации. Один из способов интерпретации AUC — это вероятность того, что модель ранжирует случайный положительный пример выше, чем случайный отрицательный пример. Например, учитывая следующие примеры, которые расположены слева направо в порядке возрастания прогнозов логистической регрессии:

Изображение выглядит как устройство, датчик

Автоматически созданное описание

**Рисунок 6. Прогнозы ранжированы в порядке возрастания оценки**

**логистической регрессии.**

AUC представляет собой вероятность того, что случайный положительный (зеленый) пример расположен справа от случайного отрицательного (красный) примера.

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 желателен по следующим двум причинам:

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

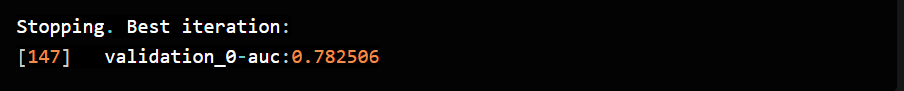
Однако обе эти причины сопровождаются оговорками, которые могут ограничивать полезность AUC в определенных случаях использования:

* **Масштабная инвариантность не всегда желательна.** Например, иногда нам действительно нужны хорошо откалиброванные выходные данные вероятности, и AUC не скажет нам об этом.
* **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 ложные срабатывания (даже если это приведет к значительному увеличению ложноотрицательных результатов). AUC не является полезным показателем для этого типа оптимизации.

Обучение модели досрочно заканчивать если в течение 20 раундов обучения не было обнаружено улучшения метрики.

**ВАЛИДАЦИЯ МОДЕЛИ**

Подключив наши тренировочные и валидационные сеты данных в модель xgboost, мы получаем окончательную валидацию AUC со значением в 0,78, имея приличное количество возможностей для прогнозирования.



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.

Изображение выглядит как текст, снимок экрана, черный, экран

Автоматически созданное описание

Неудивительно, что самой существенной характеристикой, определяемой xgboost, является разница в рангах игроков.

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

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

Один из самых основных вопросов, которые мы можем задать модели: какие функции оказывают наибольшее влияние на прогнозы?

Эта концепция называется важностью признаков.

Существует несколько способов измерения важности функции. Некоторые подходы отвечают на несколько разные версии поставленного выше вопроса. Другие подходы имеют задокументированные недостатки.

Мы сосредоточимся на важности перестановок. По сравнению с большинством других подходов важность перестановки:

• можно быстро рассчитать,

• она широко используется и понимается, и

• соответствует свойствам, которые мы хотели бы иметь для меры важности признаков.

Как это работает

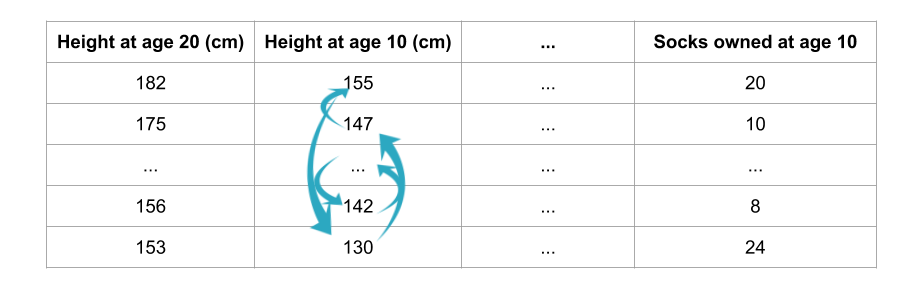
Важность перестановки использует модель не самым обычным способом, многие люди сначала находят это запутанным. Итак, мы начнем с примера, чтобы сделать его более конкретным.

Рассмотрим данные в следующем формате:

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 носками), а также некоторые другие функции, на которых мы не будем акцентировать внимание в этом объяснении.

Важность перестановки рассчитывается после подбора 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 предсказания. Если бы вместо этого мы перетасовали носки, полученные в результате, прогнозы не пострадали бы так сильно.

С этим пониманием 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.

Interpreting Permutation Importances[¶](https://www.kaggle.com/code/dansbecker/permutation-importance/tutorial#Interpreting-Permutation-Importances)

The values towards the top are the most important features, and those towards the bottom matter least.

The first number in each row shows how much model performance decreased with a random shuffling (in this case, using "accuracy" as the performance metric).

Like most things in data science, there is some randomness to the exact performance change from a shuffling a column. We measure the amount of randomness in our permutation importance calculation by repeating the process with multiple shuffles. The number after the **±** measures how performance varied from one-reshuffling to the next.

You'll occasionally see negative values for permutation importances. In those cases, the predictions on the shuffled (or noisy) data happened to be more accurate than the real data. This happens when the feature didn't matter (should have had an importance close to 0), but random chance caused the predictions on shuffled data to be more accurate. This is more common with small datasets, like the one in this example, because there is more room for luck/chance.

In our example, the most important feature was **Goals scored**. That seems sensible. Soccer fans may have some intuition about whether the orderings of other variables are surprising or not.

Рассмотрим важность перестановки для нашей модели:

Изображение выглядит как стол

Автоматически созданное описание

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.

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.

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

Дальнейшим улучшением модели будет служить внедрение elo.

**An Introduction to Tennis Elo**

Elo is a superior rating system to the ranking formulas used by the ATP and WTA. If you’ve spent much time reading this blog or listening to the podcast, you’ve probably heard me say that many times. But unless you’ve been exposed to Elo before, or done some research on your own, you might think of it as a sort of “magic” system. It’s worth digging in to understand better how it works.

**The basic algorithm**

The principle behind any Elo system is that each player’s rating is an *estimate* of their strength, and each match (or tournament) allows us to update that estimate. If a player wins, her rating goes up; if she loses, it goes down.

Where Elo excels is in determining the *amount* by which a rating should increase or decrease. There are two main variables that are taken into account: How many matches are already in the system (that is, how much confidence we have in the pre-match rating), and the quality of the opponent.

If you think about it for a moment, you’ll see that these two variables are a good approximation of how we already think about player strength. The more we already know about a player, the less we will change our opinion based on one match. [Novak Djokovic](http://www.tennisabstract.com/cgi-bin/player.cgi?p=NovakDjokovic)’s round-robin loss to [Dominic Thiem](http://www.tennisabstract.com/cgi-bin/player.cgi?p=DominicThiem) in London was a surprise, but only the most apocalyptic Djokovic fans saw the result as a disaster that should substantially change our estimate of his playing ability. Similarly, we adjust our opinion based on opponent quality. A loss to Thiem is disappointing, but a loss to, say, [Marco Cecchinato](http://www.tennisabstract.com/cgi-bin/player.cgi?p=MarcoCecchinato) is more concerning. The Elo system incorporates those natural intuitions.

**Elo rating ranges**

Traditionally, a player is given an Elo rating of 1500 when he enters the system–before any results come in. That number is completely arbitrary. All that matters is the *difference* between player ratings, so if we started each competitor with 0, 100, or 888, the end result of those differences would remain the same.

When I began calculating Elo ratings, I kept with tradition and started every player with 1500. Since then, I’ve expanded my view to Challengers (and the women’s ITF equivalent) and tour-level qualifying. If we started each new player at those levels with 1500 points, it re-scales the entire system, which would have been confusing. Instead, I replaced 1500 with a number in the low 1200s (it depends a bit on tournament level and gender) so that the ratings would remain approximately the same.

At the moment, the [ATP](http://tennisabstract.com/reports/atp_elo_ratings.html) and [WTA](http://tennisabstract.com/reports/wta_elo_ratings.html) top-ranked players are [Rafael Nadal](http://www.tennisabstract.com/cgi-bin/player.cgi?p=RafaelNadal) and [Ashleigh Barty](http://www.tennisabstract.com/cgi-bin/player.cgi?p=AshleighBarty), at 2203 and 2123, respectively. The best players are often in this range, and the *very* best often approach 2500. According to the most recent version of my algorithm, Djokovic’s peak was 2470, and [Serena Williams](http://www.tennisabstract.com/cgi-bin/player.cgi?p=SerenaWilliams)’s best was 2473.

The 2000-point mark is a good rule of thumb to separate the elites from the rest. At the moment, six men and seven women have ratings that high. 16 men and 18 women have Elo ratings of at least 1900, and a rating of 1800 is roughly equivalent to a place in the top 50.

**Era comparisons and Elo inflation**

Once we attach a single peak rating to every player, it’s only natural to start comparing across eras. While it’s always fun to do so, I’m not sure *any* rating system allows for useful cross-era comparisons in tennis. Elo doesn’t, either.

What you *can* do with Elo is compare how each player fared against her competition. In 1990, [Helena Sukova](http://www.tennisabstract.com/cgi-bin/player.cgi?p=HelenaSukova) achieved a rating of 2123–exactly the same as Barty’s today. That doesn’t mean that Sukova then was as good as Barty is now. But it does mean that their performance relative to their peers was similar. The second tier of players was considerably weaker thirty years ago, so in a sense it was easier to achieve such a rating. At the time, Sukova’s rating was only good for 11th place, far behind [Steffi Graf](http://www.tennisabstract.com/cgi-bin/player.cgi?p=SteffiGraf)’s 2600.

Thus, Elo doesn’t allow you rank players across eras unless you are confident that the level of competition was similar–or unless you have some other way of dealing with that issue, a minefield that many researchers have tried to cross, with little success.

A related issue is Elo inflation or deflation, which can also complicate cross-era comparisons. Every time a match is played, the winner and loser effectively “trade” some of their points, so the total number of Elo rating points in the system doesn’t change. However, every time a new player enters the system, the total number of points increases. And whenever a player retires, the total number of points decreases.

It would be nice if additions and subtractions canceled each other out, but for many competitions that use Elo, they don’t. Additions tend to outweigh subtractions, so Elo ratings increase over time. That doesn’t appear to be the case with my tennis ratings, at least in part because of the penalty I’ve introduced for injury absences, but it does serve as a reminder that the number of points in the system changes over time, for reasons unrelated to the strength of the top players. (I’ll have more to say about the absence penalty below.)

**Elo predictions**

Elo gives us a rating for every player, and we’re getting a sense of what we can and can’t do with them.

One of the main purposes of any rating system is to predict the outcome of matches–something that Elo does better than most others, including the ATP and WTA rankings. The only input necessary to make a prediction is the *difference* between two players’ ratings, which you can then plug into the following formula:

*1 – (1 / (1 + (10^((difference) / 400))))*

If we wanted to forecast a rematch of the last match of the Davis Cup Finals, we would take the Elo ratings of Nadal and [Denis Shapovalov](http://www.tennisabstract.com/cgi-bin/player.cgi?p=DenisShapovalov) (2203 and 1947), find the difference (256), and plug it into the formula, for a result of 81.4%, Nadal’s chance of winning. If we used the negative difference (-256), we’d get 18.6%, Shapovalov’s odds of scoring the upset.

My version of tennis Elo is based on the most common match format, best-of-three matches. In a best-of-five match, the favorite has a better chance of winning. The math for converting best-of-three to best-of-five is a bit complicated, but for those interested, I’ve posted [some code](https://github.com/JeffSackmann/tennis_misc/blob/master/fiveSetProb.py). The point is that an adjustment must be made. If the Nadal-Shapovalov rematch happens at the best-of-five Australian Open, Rafa’s 81.4% edge will increase to 86.7%.

**Adjusting Elo for surface**

For most sports, we could stop here. A match is a match, with only minor variations. In tennis, though, ratings and predictions should vary quite a bit based on surface.

My solution is a bit complicated. For each player, I maintain four separate Elo ratings: overall, hard court only, clay court only, and grass court only. I don’t differentiate between outdoor and indoor hard. For instance, Thiem’s ratings are 2066 overall, 1942 on hard, 2031 on clay, and 1602 on grass. (Surface ratings tend to be lower: Thiem’s clay rating is third-best on tour, miles ahead of everyone except for Nadal and Djokovic.)

These single-surface ratings tell us how we would rank players if we simply threw away results on every other surface. That’s not realistic, though. Single-surface ratings aren’t great at predicting match results. A better solution is to take a 50/50 blend of single-surface and overall ratings. If we wanted to predict Thiem’s chances in a clay-court match, we’d use a half-and-half mix of his 2066 overall rating and his 2031 clay-court rating. My [weekly Elo reports](http://tennisabstract.com/reports/atp_elo_ratings.html) show the single-surface ratings as “HardRaw” (and so on), and the blended ratings as “hElo,” “cElo,” and “gElo.”

There is no natural law that dictates a 50/50 blend. Every adjustment I’ve made to the basic Elo algorithm is determined solely by what works. (More on that below.) Initially, I suspected that a blend between single-surface and overall ratings would be appropriate, because a player’s success on one surface has some correlation with his success on others. I expected the blend to be different for each surface–perhaps using a higher percentage of the overall rating for grass, because there are [fewer matches](https://www.tennisabstract.com/blog/2017/06/23/unpredictable-bounces-predictable-results/) on the surface. In the end, my testing showed that 50/50 worked for each surface.

**Non-adjustments**

Ask some tennis fans which tournaments matches matter more–for rankings, for GOAT debates, whatever–and you can find yourself with a long, detailed list of what factors determine greatness. Maybe slams are more important than masters and premiers, though those are less important than tour finals and the Olympics, and of course finals are key, plus head-to-heads against certain players… you get the idea.

Elo provides for such adjustments. A coefficient usually referred to as the “k factor” allows us to give greater weight to certain matches. It’s common in Elo ratings for other sports, for example by using a higher k factor for postseason than regular season games. However, I’ve tested all sorts of different k factors for the likely types of “important” matches, and I’ve yet to find a tweak to the system that consistently improves its ability to predict match outcomes.

**The absence penalty**

There’s one exception. When players miss substantial amounts of time, I reduce their rating, and then increase the k factor for several matches after their return. I’ve explained more of the details in a [previous post](http://www.tennisabstract.com/blog/2018/05/15/handling-injuries-and-absences-with-tennis-elo/).

These steps are a logical extension of the Elo framework, especially when you consider our usual mental adjustments when a player misses time. If a player is injured for a few months, we never know quite what to expect when she returns. Maybe she’s as strong as ever; maybe she’s still a step slow. Perhaps she’ll return to normal quickly; she might never fully return to form. An extended absence raises a lot of questions. An injury player rarely returns in *better* form than when she left, while many players are worse upon return, giving us an average post-injury performance level that is worse than before the absence.

Therefore, when a player first returns, our *estimate* must be that she is worse. However, a few strong early results should be weighted more heavily–hence the higher k factor. The k factor reflects the fact that, immediately after an absence, we aren’t as confident as usual in our estimate.

The algorithm gets complicated, but the logic is simple. It’s basically just an attempt to work out a rigorous version of statements like, “I don’t know how well he’ll play when he comes back, but I’ll be watching closely.”

One side benefit of the absence penalty is that it counteracts Elo’s natural tendency toward ratings inflation. While more players enter the system than leave it, adding to the total number of available points, the penalty removes some points without re-allocating them to other players.

**Validating Elo and adjustments**

I’ve mentioned “testing” a few times, and I started this article with a claim that Elo is superior to the official ranking systems. What does that mean, and how do we know?

The simplest way to compare rating systems is a metric called “accuracy,” which counts correct predictions. There were 50 singles matches at the Davis Cup finals, and Elo picked the winner in 36 of them, for an accuracy rating of 72%. The ATP rankings picked the winner (in the sense that the higher-ranked player won the match) in 30 of them, for an accuracy rating of 60%. In this tiny experiment, Elo trounced the official rankings. Elo is also considerably better over the course of the entire season.

A better metric for this purpose is Brier score, which takes into account the confidence of each forecast. We saw earlier that Elo gives Nadal an 81.4% chance of beating Shapovalov. If Nadal ends up winning, 81.4% is a “better” forecast than, say, 65%, but it’s a “worse” forecast than 90%. Brier score takes the squared distance between the forecast (81.4%) and the result (0% or 100%, depending on the winner), and averages those numbers for all forecasted matches. It rewards aggressive forecasts that prove correct, but because it uses squared distance, it severely punishes predictions that are aggressive but wrong.

A more intuitive way to think about what Brier score is getting at is to imagine that Nadal and Shapovalov play 100 matches in a row. (Or, more accurately but less intuitively, imagine that 100 identical Nadals play simultaneous matches against 100 identical Shapovalovs.) A forecast of 81.4% means that we would expect Nadal to win 81 or 82 or those matches. If Nadal ends up winning 90, the forecast wasn’t Rafa-friendly enough. We’ll never get 100 simultaneous matches like this, but we do have thousands of individual matches, many of which share the same predictions, like a 60% chance of the favorite winning. Brier score aggregates all of those prediction-and-result pairs and spits out a number to tell us how we’re doing.

It’s tough to forecast the result of individual tennis matches. Any system, no matter how sophisticated, is going to be wrong an awful lot of the time. In many cases, the “correct” forecast is barely better than no forecast at all, if the evidence suggests that the competitors are equally matched. Thus, “accuracy” is of limited use–it’s more important to have the right amount of confidence than to simply pick winners.

All of this is to say: My Elo ratings have a much lower (better) Brier score than predictions derived from ATP and WTA rankings. Elo forecasts aren’t quite as good as betting odds, or else I’d be spending more time wagering and less time writing about rating systems.

Brier score is also the measure that tells us whether a certain adjustment–such as surface blends, injury absences, or tournament type–constitutes an improvement to the system. Assessing an injury penalty lowers the Brier score of the overall set of Elo forecasts, so we keep it. Decreasing the k factor for first-round matches has no effect, so we skip it.

|  |  |  |
| --- | --- | --- |
|  | **СПИСОК ИСТОЧНИКОВ** |  |
| 1. [Электронный | | ресурс]: |
|  | <https://shalaginov.com/2020/05/16/computer-vision-history/> |  |

1. Wikipedia [Электронный ресурс]: https://ru.wikipedia.org
2. [Электронный ресурс]: <https://medium.com/profisoft>
3. [Электронный ресурс]: <https://www.jetinfo.ru/computer-vision-technology-review/>
4. [Электронный ресурс]: https://dlcourse.ai/

https://science-engineering.ru/ru/article/view?id=1238

https://github.com/JeffSackmann/tennis\_atp

https://blog.skillfactory.ru/glossary/git/

<https://github.com/skoval/deuce>

<https://ru.wikipedia.org/wiki/RStudio#:~:text=RStudio%20%E2%80%94%20%D1%81%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F%20%D1%81%D1%80%D0%B5%D0%B4%D0%B0%20%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE,%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%20%D0%B8%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B%20%D1%81%20%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%BE%D0%B9>.

<https://ahmedushka7.github.io/R/scripts/hse_data_analysis/sem_1/introduction_to_R.html>

<https://blog.ringostat.com/ru/parsing-dannyh-s-saytov-chto-eto-i-zachem-on-nuzhen/>

<https://medium.com/nuances-of-programming/%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-xgboost-%D0%BF%D1%83%D1%81%D1%82%D1%8C-%D0%BE%D0%BD-%D1%86%D0%B0%D1%80%D1%81%D1%82%D0%B2%D1%83%D0%B5%D1%82-%D0%B4%D0%BE%D0%BB%D0%B3%D0%BE-dc8c4eca3fbc>

https://techcave.ru/posts/81-sozdaem-pervuyu-xgboost-model-na-python-s-ispolzovaniem-scikit-learn.html#:~:text=%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%20XGBoost%20%D0%B4%D0%BB%D1%8F%20%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%20%D0%BD%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F,%D0%B1%D1%8B%D1%82%D1%8C%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D0%BD%D1%8B%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%20%D0%B2%20%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80%D0%B5.

https://ru.wikipedia.org/wiki/Pandas

https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5\_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F

https://www.kaggle.com/code/dansbecker/permutation-importance/tutorial

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

http://www.tennisabstract.com/blog/2019/12/03/an-introduction-to-tennis-elo/

https://pythonworld.ru/numpy/1.html