Данные, которые используются:

1. История матчей основных европейских лиг
2. Информация о командах
3. Информация о игроках связанная с информацией из футбольной игры(рейтинг игрока и его приблизительные характеристики)

Преобразования, совершаемые над данными:

Данные изначально представлены в виде БД SQLite, реализованы специальные методы для парсинга.

- Поиск команды по id
- Поиск матча по id
- Поиск команды по названию
- Последние 5 матчей команды(по id)
- Статистика 2 команд по 2 id
- Средний рейтинг состава команд (по Fifa_api)

Основные преобразования:

На вход каждому нейрону подается число, всего 3 входных параметра:

1. отношение среднего состава команд
2. статистика команд (отношение)
3. последние 5 матчей (отношение)

Описание кода

Скрипт Information_parser.py

Описание: класс обладает всеми необходимыми методами для получения основной информации из БД. Используемые таблицы: Match, Player, Fifa_Players, Team. Каждый из методов представляет из себя select'ы разной сложности с разными параметрами.

Классы:

1)Team_Dao

Класс для работы с сущностью Team.

Методы класса:

1. get_team_by_id(self, team_id) - на вход метод получает id команды (api_id в таблице Team), далее обращается к БД с селектом с соответствующим условием и выводит на выходе сущность Team(класс Entity.Team). Если таких объектов из БД несколько, выберется первый.
2. get_team_by_fullname(self, fullname) - метод выводит сущность Entity.Team, основываясь на полном имени команды(считается, что оно уникально). Представляет из себя селект с соответствующим условием.

Использование:

Класс необходим при обучении нейронной сети для доступа к необходимым параметрам каждой из команд, участвующих в матче(например fifa_api_id), также для взаимодействия с пользователем(в перспективе).

2) Match_Dao

Класс для работы с сущностью Match

Методы класса

1.get_match_by_teams_season(self, home_team_id, away_team_id, season) - на входе api_id домашней команды, api_id гостевой, сезон (например 2006/2007)

Описание: метод выводит матч между двумя командами(порядок важен) в определенном сезон.

2.get_by_id(self, match_id) - результатом будет сущность Match, полученная по id в бд(порядковый номер, не путать с api_id).

Используется для случайной выборке при обучении сети.

3.get_last5_matches_of_team(self, team, date) - team-api_id команды, date-дата, крайняя дата последнего матча. Возвращает лист из 5 матчей команды. Используется для работы нейронной сети как один из входных параметров.
4.get_players_func(self, match_id)-по match_api_id, указанным на входе, функция сначала из таблицы Match достает api_id всех игроков, участвовавших в матче с этим match_id, далее делит данные на игроков гостей и хозяев. Полученный лист итерируется, формируется select сначала к таблице Player, где получается fifa_api_id, по которому из таблицы Player_attribute достается средний рейтинг каждого игрока, который добавляется в новый лист. Получаются два листа из среднего рейтинга команды гостей и хозяев. Если хотя бы один из них пустой-выводится 0, что означает “этот параметр не дает преимущества ни одной из сторон”. Иначе выводится среднее арифметическое 1ого листа, деленное на ср. арифм. второго листа.
5.get_statistics(self, team_home_id, team_away_id) - метод по api_id гостевой команды и домашней(порядок не важен) выводит массив из 1, 0 или -1, где 1-победа первой команды, 0-ничья, -1-победа второй. Используется как один из входных параметров(после преобразования) нейронной сети.

Скрипт Entity.py
Класс Team
Сущность команда
Поля:
team_name - полное название команды
api_id - id команды(не путать с id в бд, что есть порядковый номер)
team_fifa_id - id команды в таблицы Team_attribute

Класс Match
Сущность матч
m_id - id - матча в БД
api_id - id матча, используемое для связей с другими таблицами
away_team - api_id команды гостей
home_team - api_id команды хозяев
season - сезон, в который игрался матч

home_Team_goal - кол-во голов хозяев
away_Team_goal - кол-во голов гостей
date - дата, когда проводился матч

Скрипт Neuron.py

Класс EntryNeuron

Поля:
parameter - название входного параметра
parameter_value = его значение в формате float
hidden_neurons = лист из Внутренних нейронов класса HiddenNeuron.
hidden_neurons_way = лист из float чисел, веса синапсов до внутр. нейронов

Методы:
set_input_value(self, value) - метод вводит во входной нейрон соотв. параметр(его значение).
send_to_hidden(self) - отправляет данные в следующий слой(по листу hidden_neurons). Вызывается метод из класса HiddenNeuron.add_input(value, cost), где value - значение параметра, а cost - ребро до этого нейрона

Класс HiddenNeuron

Поля:
output_result - выходной результат нейрона после функции активации, используется для обучения нейрона методом обратного распространения ошибки, по нему высчитывается дельта ошибки.
input_data - входные параметры из EntryNeuron(лист)
way_to_entry_neuron - величины синапсов до входных нейронов(лист)
way_to_out_neuron - путь до конечного нейрона(float)
outer_neuron - ссылка на конечный нейрон (OuterNeuron)

Методы:
add_input(self, stat, cost) - добавление данных и входных нейронов
execute(self) - по входным данным(линейная комбинация) получается число, которое подставляется в функцию активации(гиперболический тангенс, math.tanh(пакет math)), результат записывается в output. После этого вызывается метод outer_neuron.set_from_hidden(self.output_result,

self.way_to_out_neuron), то есть результат отправляется в конечный нейрон.

Класс OuterNeuron
Поля:
output - результат работы нейрона после функции активации(как и в hiddenNeuron, гиперболический тангенс, где x-линейная комбинация из данных от скрытого слоя и путей до них)
final_data - данные из скрытого слоя(лист, результат работы каждого скрытого нейрона)
way_to_hidden - пути до скрытых нейронов(лист)
Методы:
set_from_hidden(self, value, cost) - добавление данных из скрытых нейронов
execute(self) - данные от скрытых нейронов передаются в функцию активации, результат которой записывается в output
get_error(self, must_result) - ошибка результата(output) по отношению к must_result по MSE, используется при обучении нейронной сети

Класс NeuronNet
Конструктор - принимает количество входных параметров, их названия и количество скрытых нейронов
Поля:
out_neuron - конечный нейрон(класс OuterNeouron)
hidden_neurons - лист из скрытых нейронов(класс HiddenNeuron, размер листа из конструктора)
input_neurons - лист из входных нейронов(класс EntryNeuron, размер листа из конструктора)
Методы:
init_net_ways() - случайным образом задает начальные синапсы(ребра) нейронной сети, малым числом(от 0 до 1)
execute(self, input_values):
input_values - входные данные нейронной сети. Сначала они добавляются во входные нейроны(по листу hidden_neurons, у каждого вызывается set_input_value), далее такая же итерация, но уже вызывается set_to_hidden(), после чего у каждого из hidden_neourons вызывается

execute. После этого тот же метод вызывается у out_neuron, и результат его возвращается return-ом.

back_propagation()

реализация метода обратного распространения ошибки:

1. Инициализировать $\{w_{ij}\}_{i,j}$ маленькими случайными значениями, $\{\Delta w_{ij}\}_{i,j} = 0$
2. Повторить NUMBER_OF_STEPS раз:

   .Для всех d от 1 до m:

   1. Подать $\{x_i^d\}$ на вход сети и подсчитать выходы o_i каждого узла.
   2. Для всех $k \in Outputs$

      $$\delta_k = -o_k(1 - o_k)(t_k - o_k)\,.$$

   3. Для каждого уровня l, начиная с предпоследнего:

      Для каждого узла j уровня l вычислить

      $$\delta_j = o_j(1 - o_j) \sum_{k \in Children(j)} \delta_k w_{j,k}.$$

   4. Для каждого ребра сети {i, j}

      $$\Delta w_{i,j}(n) = \alpha \Delta w_{i,j}(n-1) + (1-\alpha)\eta \delta_j o_i\ .$$
      $$w_{i,j}(n) = w_{i,j}(n-1) + \Delta w_{i,j}(n).$$

3. Выдать значения w_{ij}.

Вывод из работы:

Нейронная сеть-простой для обучения, но при этом мощный инструмент для задач данного типа. Основные проблемы, с которыми можно столкнуться-отсутствие данных или их малое количество, при котором невозможно обучение с учителем.

Также сложно преобразовывать нейронную сеть, т.к. необходимо каждый раз исправлять синапсы, что может только ухудшить результат(сеть подходит для этих данных, но не для общих).

Однако, несмотря на это, нейронные сети-отличная тема для изучения машинного обучения, т.к. способна решать далеко не самые простые задачи.

В данном проекте используются только 3 основных параметра, но также можно оптимизировать сеть, используя большее число параметров, но проиграть в скорости(из-за больших вычислений).