betbot

Release 0.5.3

Hermann Krumrey

Sep 18, 2020

CONTENTS

Contents:

CHAPTER

ONE

BETBOT PACKAGE

1.1 Subpackages

1.1.1 betbot.api package

Submodules

betbot.api.ApiConnection module

class betbot.api.ApiConnection.**ApiConnection**(*username: str*, *password: str*, *url: str*)

Bases: object

Class that handles API calls to the bundesliga-tippspiel instance

__init__(*username: str*, *password: str*, *url: str*)

Intitializes the API connection :param username: The username on bundesliga-tippspiel :param password: The password for that user :param url: The base url to the bundesliga-tippspiel instance

property auth_headers

Returns Authorization headers for API calls

authorized() → bool

Checks if the stored API key is valid :return: True if valid, False if not (for example because it expired)

execute_api_call(*endpoint: str*, *method: str*, *authorization_required: bool = False*, *json_data: Optional[Dict[str, Any]] = None*) → Dict[str, Any]

Executes an API call :param endpoint: The API endpoint :param method: The request method :param authorization_required: Whether authoirzation is required :param json_data: The JSON data to send :return: The response JSON

get_current_matchday_matches() → List[*betbot.api.Match.Match*]

Retrieves a list of matches for the current matchday :return: The list of matches

login() → bool

Retrieves an API Key using a username and password if no key has been retrieved yet or if the existing key has expired :return: True if the login was successfulle

place_bets(*bets: List[*betbot.api.Bet.Bet*]*)

Places a list of bets :param bets: The bets to place :return: None

betbot.api.Bet module

class betbot.api.Bet.**Bet**(*match_id: int*, *home_score: int*, *away_score: int*)
Bases: object

Class that encapsulates Bet information

__init__(*match_id: int*, *home_score: int*, *away_score: int*)
Initializes the Bet :param match_id: The ID of the associated match :param home_score: The score bet on the home team :param away_score: The score bet on the away team

to_dict() → Dict[str, int]

Returns A dictionary that canbe used to place the bet using the API

betbot.api.Match module

class betbot.api.Match.**Match**(*_id: int*, *matchday: int*, *home_team: str*, *away_team: str*, *finished: bool*)
Bases: object

Class that encapsulates information about a Match

__init__(*_id: int*, *matchday: int*, *home_team: str*, *away_team: str*, *finished: bool*)
Initializes the Match :param _id: The ID of the match :param matchday: The matchday of the match :param home_team: The name of the home team :param away_team: The name of the away team :param finished: Whether the match is already finished or not

classmethod from_json(*json_data: Dict[str, Any]*)
Generates a Match object from JSON data :param json_data: The JSON data :return: The generated Match

Module contents

1.1.2 betbot.neural package

Subpackages

betbot.neural.data package

Subpackages

betbot.neural.data.vector package

Submodules

betbot.neural.data.vector.InputVector module

class betbot.neural.data.vector.InputVector.**InputVector**(*bet_odds: Dict[*betbot.neural.data.enums.Bookmakers*, Tuple[float, float, float]], average_goals: Dict[str, Dict[str, Dict[int, float]]], average_odds: Dict[str, Dict[*betbot.neural.data.enums.Bookmakers*, Dict[str, Dict[int, float]]]]*)

Bases: *betbot.neural.data.vector.Vector.Vector*

Class that models an input vector for use as training data for a neural network

__init__(*bet_odds: Dict[*betbot.neural.data.enums.Bookmakers*, Tuple[float, float, float]], average_goals: Dict[str, Dict[str, Dict[int, float]]], average_odds: Dict[str, Dict[*betbot.neural.data.enums.Bookmakers*, Dict[str, Dict[int, float]]]]*) → None

Initialize self. See help(type(self)) for accurate signature.

average_goals: Dict[str, Dict[str, Dict[int, float]]]

average_odds: Dict[str, Dict[*betbot.neural.data.enums.Bookmakers*, Dict[str, Dict[int,

bet_odds: Dict[*betbot.neural.data.enums.Bookmakers*, Tuple[float, float, float]]

classmethod from_data(*match:* betbot.neural.data.Match.Match, *home_history: List[*betbot.neural.data.Match.Match*], away_history: List[*betbot.neural.data.Match.Match*]*) → *betbot.neural.data.vector.InputVector.InputVector*

Generates an input vector based on match and historical match data :param match: The match data :param home_history: Historical match data for the home team :param away_history: Historical match data for the away team :return: The generated input vector object

classmethod legend() → List[str]

Returns Strings describing the individual parts of the vectors

property vector

Returns A vector of floats that can be used to train neural networks

betbot.neural.data.vector.OutputVector module

class betbot.neural.data.vector.OutputVector.**OutputVector**(*home_goals: int, away_goals: int*)

Bases: *betbot.neural.data.vector.Vector.Vector*

Class that models an output vector for a match

__init__(*home_goals: int, away_goals: int*) → None

Initialize self. See help(type(self)) for accurate signature.

away_goals: int

home_goals: int

classmethod legend() → List[str]

Returns Strings describing the individual parts of the vectors

property vector

Returns The vector as a list of float values

betbot.neural.data.vector.Vector module

class betbot.neural.data.vector.Vector.**Vector**
Bases: object

Class that models a vector

classmethod legend() → List[str]

Returns Strings describing the individual parts of the vectors

property vector

Returns The vector as a list of float values

Module contents

Submodules

betbot.neural.data.DataFetcher module

class betbot.neural.data.DataFetcher.**DataFetcher**(*no_cache: bool = False*)
Bases: object

Class that specifies common methods used to fetch data

__init__(*no_cache: bool = False*)
Initializes the DataFetcher Automatically loads data from the internet if it does not yet exist locally. :param no_cache: Whether or not to use local data (if it exists)

get_current_matchday_vectors(*country: str*, *league: int*) → List[Tuple[*betbot.neural.data.vector.InputVector.InputVector*, *betbot.neural.data.Match.Match*]]
Retrieves input vectors for the current matchday :param country: The country for which to fetch vectors :param league: The league for which to fetch vectors :return: The input vectors and the corresponding match objects

get_training_data() → List[Tuple[*betbot.neural.data.Match.Match*, *betbot.neural.data.vector.InputVector.InputVector*, *betbot.neural.data.vector.OutputVector.OutputVector*]]

Returns All training data from previous matches

load() → Dict[str, Dict[int, Dict[int, List[Dict[str, str]]]]]
Loads the match data from the internet :return: The data as dictionaries representing matches,

mapped to their respective countries, leagues and seasons

load_training_vectors(*target: str*) → List[Tuple[List[float], List[float]]]
Loads the raw training vectors from a CSV file :param target: The target file :return: The vectors

save(*target_file: str*)
Saves the currently loaded data to a file :param target_file: The file to which to save :return: None

write_training_vectors_to_csv(*target: str*)
Writes the training vectors to a CSV file :param target: The path to the CSV file :return: None

betbot.neural.data.FootballDataFetcher module

class betbot.neural.data.FootballDataFetcher.**FootballDataFetcher**(*no_cache: bool = False*)

Bases: *betbot.neural.data.DataFetcher.DataFetcher*

Class that uses football-data.org to fetch data

get_current_matchday_vectors(*country: str*, *league: int*) → List[Tuple[*betbot.neural.data.vector.InputVector.InputVector*, *betbot.neural.data.Match.Match*]]

Retrieves input vectors for the current matchday :param country: The country for which to fetch vectors :param league: The league for which to fetch vectors :return: The input vectors and the corresponding match objects

get_training_data() → List[Tuple[*betbot.neural.data.Match.Match*, *betbot.neural.data.vector.InputVector.InputVector*, *betbot.neural.data.vector.OutputVector.OutputVector*]]

Returns All training data from previous matches

load() → Dict[str, Dict[int, Dict[int, List[Dict[str, str]]]]]

Loads the match data from the internet :return: The data as dictionaries representing matches,

mapped to their respective countries, leagues and seasons

load_matches() → Dict[str, List[*betbot.neural.data.Match.Match*]]

Loads all matches segmented by country :return: {country: [matches]}

static load_oddsportal_matches() → List[*betbot.neural.data.Match.Match*]

Loads match data from oddsportal :return: THe match data from oddsportal

segment_matches_by_team(*all_matches: Dict[str, List[*betbot.neural.data.Match.Match*]]*) → Dict[str, Dict[str, List[*betbot.neural.data.Match.Match*]]]

Segments matches by the teams involved in them :param all_matches: Data on all matches :return: The segmented match data

betbot.neural.data.Match module

class betbot.neural.data.Match.**Match**(*country: str*, *league: int*, *season: int*, *date: datetime.datetime*, *finished: bool*, *home_team: str*, *away_team: str*, *home_ht_score: Optional[int]*, *away_ht_score: Optional[int]*, *home_ft_score: Optional[int]*, *away_ft_score: Optional[int]*, *bet_odds: Dict[*betbot.neural.data.enums.Bookmakers, *Tuple[float, float, float]]*)

Bases: object

Models the required attributes of a Match

__init__(*country: str*, *league: int*, *season: int*, *date: datetime.datetime*, *finished: bool*, *home_team: str*, *away_team: str*, *home_ht_score: Optional[int]*, *away_ht_score: Optional[int]*, *home_ft_score: Optional[int]*, *away_ft_score: Optional[int]*, *bet_odds: Dict[*betbot.neural.data.enums.Bookmakers, *Tuple[float, float, float]]*) → None

Initialize self. See help(type(self)) for accurate signature.

away_ft_score: Optional[int]

away_ht_score: Optional[int]

away_team: str

bet_odds: Dict[*betbot.neural.data.enums.Bookmakers*, Tuple[float, float, float]]

country: str

date: datetime.datetime

finished: bool

classmethod from_football_data(*data: Dict[str, str]*) → Optional[*betbot.neural.data.Match.Match*]

Generates a match based on data from football-data.co.uk :param data: The data to use :return: The generated Match

home_ft_score: Optional[int]

home_ht_score: Optional[int]

home_team: str

league: int

season: int

betbot.neural.data.enums module

class betbot.neural.data.enums.**Bookmakers**(*value*)

Bases: enum.Enum

Class that defines the various bookmakers

B365 = 'b365'

BWIN = 'bw'

WILLIAM_HILL = 'wh'

class betbot.neural.data.enums.**Countries**(*value*)

Bases: enum.Enum

Class that defines the various countries

BELGIUM = 'belgium'

ENGLAND = 'england'

FRANCE = 'france'

GERMANY = 'germany'

GREECE = 'greece'

ITALY = 'italy'

NETHERLANDS = 'netherlands'

PORTUGAL = 'portugal'

SCOTLAND = 'scotland'

SPAIN = 'spain'

TURKEY = 'turkey'

betbot.neural.data.names module

Module contents

betbot.neural.keras package

Submodules

betbot.neural.keras.BetPredictorTrainer module

class betbot.neural.keras.BetPredictorTrainer.**BetPredictorTrainer**(*model_dir: str*)

Bases: *betbot.neural.keras.Trainer.Trainer*

Trainer Class that specifies an evaluation for bets

betbot.neural.keras.MatchHistoryTrainer module

class betbot.neural.keras.MatchHistoryTrainer.**MatchHistoryTrainer**(*model_dir: str*)

Bases: *betbot.neural.keras.BetPredictorTrainer.BetPredictorTrainer*

Trainer Class that specifies a neural network for use with historical match data

load_training_data(*force_refresh: bool*) → List[Tuple[List[float], List[float]]]

Loads data for training the model :param force_refresh: Forces a refresh of data :return: The training data, with the input and output vectors separate

betbot.neural.keras.Trainer module

class betbot.neural.keras.Trainer.**Trainer**(*model_dir: str*)

Bases: object

Class that models common functions of a neural network trainer

__init__(*model_dir: str*)

Initializes the trainer :param model_dir: The directory in which to store models

load_trained_model(*iterations: int = 4, epochs: int = 32, batch_size: int = 64, force_retrain: bool = False, minimum_accuracy: float = 0.0, custom_model_fn: Optional[Callable[], tensorflow.python.keras.engine.training.Model]] = None, custom_compile_fn: Optional[Callable[[tensorflow.python.keras.engine.training.Model], None]] = None*) → tensorflow.python.keras.engine.training.Model

Generates a trained keras model that can be used to predict output immediately :param iterations: The amount of iterations to train :param epochs: The amount of epochs to train :param batch_size: The batch size to use when training :param force_retrain: Whether or not to force retraining :param minimum_accuracy: Optional value for minimum accuracy :param custom_model_fn: Allows using a custom model to be trained :param custom_compile_fn: Allows using a custom compile function :return: The trained model

load_training_data(*force_refresh: bool*) → List[Tuple[List[float], List[float]]]

Loads data for training the model :param force_refresh: Forces a refresh of data :return: The training data, with the input and output vectors separate

train(*iterations: int*, *epochs: int*, *batch_size: int*, *custom_model_fn: Optional[Callable[], tensorflow.python.keras.engine.training.Model]] = None*, *custom_compile_fn: Optional[Callable[[tensorflow.python.keras.engine.training.Model], None]] = None*) → Tuple[tensorflow.python.keras.engine.training.Model, float, float, float, float]

Trains the model using the training data for a specified amount of times and returns the best performing model Iterations and epochs are different things! Each iteration starts from scratch. :param iterations: The amount of iterations :param epochs: The amount of epochs to train :param batch_size: The batch size to use :param custom_model_fn: Allows using a custom model to be trained :param custom_compile_fn: Allows using a custom compile function :return: The best trained model and its score + accuracy

as well as average score and accuracy stats

Module contents

Module contents

1.1.3 betbot.prediction package

Submodules

betbot.prediction.DrawPredictor module

class betbot.prediction.DrawPredictor.**DrawPredictor**

Bases: *betbot.prediction.Predictor.Predictor*

Class that always predicts 0:0, 1:1, 2:2 or 3:3

classmethod name() → str

Returns The name of the predictor

predict(*matches: List[*betbot.api.Match.Match*]*) → List[*betbot.api.Bet.Bet*]

Performs the prediction :param matches: The matches to predict :return: The predictions as Bet objects

betbot.prediction.MatchHistoryNNPredictor module

class betbot.prediction.MatchHistoryNNPredictor.**TableHistoryNNPredictor**

Bases: *betbot.prediction.Predictor.Predictor*

Class that predicts results based on historical match data

__init__()

Initializes the Neural Network. Loads existing weights if they exist, otherwise will train new weights

classmethod name() → str

Returns The name of the predictor

predict(*matches: List[*betbot.api.Match.Match*]*) → List[*betbot.api.Bet.Bet*]

Performs the prediction :param matches: The matches to predict :return: The predictions as Bet objects

betbot.prediction.Predictor module

class betbot.prediction.Predictor.**Predictor**
Bases: object

Class that specifies required methods for predictor objects

__init__()
Initializes the model directory if it does not exist

classmethod name() → str

Returns The name of the predictor

predict(*matches: List[*betbot.api.Match.Match*]*) → List[*betbot.api.Bet.Bet*]
Performs the prediction :param matches: The matches to predict :return: The predictions as Bet objects

betbot.prediction.RandomPredictor module

class betbot.prediction.RandomPredictor.**RandomPredictor**
Bases: *betbot.prediction.Predictor.Predictor*

Class that always predicts random results

classmethod name() → str

Returns The name of the predictor

predict(*matches: List[*betbot.api.Match.Match*]*) → List[*betbot.api.Bet.Bet*]
Performs the prediction :param matches: The matches to predict :return: The predictions as Bet objects

betbot.prediction.TipicoOddsPredictor module

class betbot.prediction.TipicoOddsPredictor.**TipicoOddsPredictor**
Bases: *betbot.prediction.Predictor.Predictor*

Class that determinalistically predicts matches based on Tipico quotes

generate_bet(*match_id: int*, *home_odds: float*, *draw_odds: float*, *away_odds: float*) → *betbot.api.Bet.Bet*
Generates a Bet based on the quote data :param match_id: The ID of the match to vote on :param home_odds: The odds that the home team wins :param draw_odds: The odds that the match ends in a draw :param away_odds: The odds that the away team wins :return: The generated bet

load_tipico_data(*matchday: int*) → List[Tuple[str, str, float, float, float]]
Loads tipico quote data for the 2020/21 bundesliga season :param matchday: The matchday for which to retrieve the data :return: The quote data as a list of tuples of the form:

home team name, away team name, home win odds, draw odds, away win odds

classmethod name() → str

Returns The name of the predictor

predict(*matches: List[*betbot.api.Match.Match*]*) → List[*betbot.api.Bet.Bet*]
Performs the prediction :param matches: The matches to predict :return: The predictions as Bet objects

Module contents

1.1.4 betbot.test package

Module contents

1.2 Submodules

1.3 betbot.main module

betbot.main.**main**(*predictor_name: str*, *username: str*, *password: str*, *url: str*)
The main function of the betbot Periodically predicts results and places bets accordingly :param predictor_name: The name of the predictor to use :param username: The username for bundesliga-tippspiel :param password: The password for bundesliga-tippspiel :param url: The base URL for the bundesliga-tippspiel instance :return: None

1.4 Module contents

betbot.**sentry_dsn = 'https://bf9dc130ba4c4284a42813a0032b63fe@sentry.namibsun.net/21'**
The sentry DSN used for exception logging

CHAPTER

TWO

BETBOT

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

W