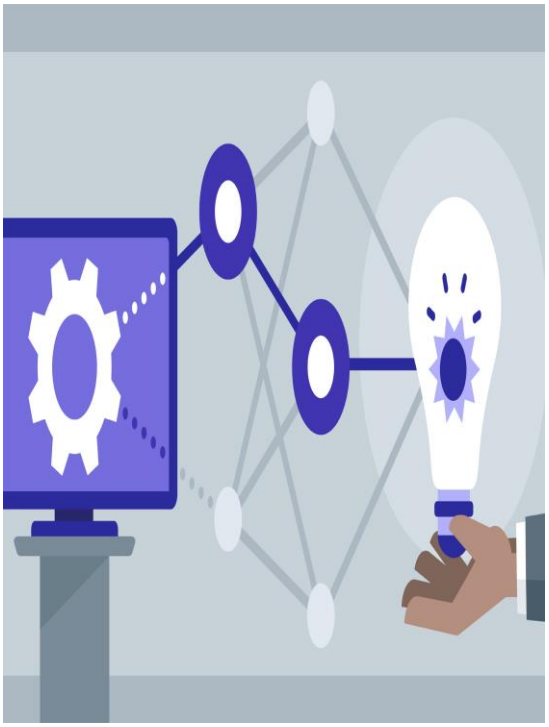

Skimind Use cases Interface

HOW TO USE SERVICES

Nathan Bangwa | IkalaWeb | 04 April 2019

Arborescence

- Skimind
  - checkPoint
  - gFunctions
  - interfacedb
  - learningModel
  - pdfManag
  - useCases
    - buildTicket
      - main.py
    - homeView
      - main.py

Description

`Skimind.useCases`

Ce module contient tous les modules qui permettent a un utilisateur de profiter des services fournis par skimindAI

HomeView

Cette interface permet a un utilisateur de voir les recentes predictions sur lesquelles il peut se decider de parier.

Skimind.useCases.homeView.main.py

Ce module contient une classe HomeView qui permet de recuperer les donnees predites et mettre à jour l'état des donnés de la vue principale.

```
Class HomeView(object)
     def __init__(self):
          Self.meta_data = pandas.DataFrame()

     def get_selected_columns(…):-> list

     def filter_datas(self,by_idmatch, by_datetime):->dict
```

cette method permet d'actualiser l'etat des matchs selon qu'ils ont deja commence ou pas

by_idmatch-> bool

filtrer les matchs selon leurs numeros respectifs

by_datetime-> bool

filtrer les matchs selon leurs dates respectives

Return -> cette methode returne les données à afficher dans un dictionnaire. (voir page 4)

Return : JSON

```
{
	numlist: // cette key est du type int et sa valeur est du type list
	[
		// chaque liste contient des valeurs correpondant a
		[
			idmatch->int, # le numero du match
			datetime->datetime, # la date et l'heure
			home->str, # le nom de l'equipe a domicile
			visitor->str, # le nom de l'equipe a l'exterieur
			# tache normal probabilities
			normal_cross->float, # probability normal cross
			normal_first->float, # probability normal first
			normal_second->float, # probability normal second
			normal_predict->int, # la prediction
			# tache first_mt probabilities
			first_mt_cross->float, # probability first_mt cross
			first_mt_first->float, # probability first_mt first
			first_mt_second->float, # probability first_mt second
			first_mt_predict->int, # la prediction
			# tache second_mt probabilities
			second_mt_cross->float, # probability second_mt cross
			second_mt_first->float, # probability second_mt first
			second_mt_second->float, # probability second_mt second
			second_mt_predict->int, # la prediction
			# tache second_mt probabilities
			mt_more_goal_cross->float, # probability mt_more_goal cross
			mt_more_goal_first->float, # probability mt_more_goal first
			mt_more_goal_second->float, # probability mt_more_goal second
			mt_more_goal_predict->int, # la prediction
			# status du matchs
			Matchs_status->str, # STARTED/NOSTARTED]]}
```

Utilisation

- Skimind
- test.py

```
from skimind.useCases.homeView import main as homeview

# HomeView -> vue principale ou global

viewer = homeview.HomeView()

# Recuperation des données a afficher
(actualization)

datas = viewer.filter_datas()

# affichage des données a la vue principale

# actualisation des donnees

datas = viewer.filter_datas()

# reaffichage des données a la vue
principale
```

BuildTicket

Cette interface permet à un utilisateur de construire un ou plusieurs ticket selon ses moyens.

Skimind.useCases.buildTicket.main.py

> Ce module contient une classe BuildTicket dont les attribut sont des données fournies par l'utilisateur excepté l'attribut meta_data. Et c'est cette classe qui construit un plusieurs meilleurs tickets en fonction des parametres utilisateurs.

```
Class BuildTicket(object)
     def __init__(self, bet_money, win_money)->
          Self.bet_money = win_money
          Self.win_money = bet_money
          Self.meta_data = pandas.DataFrame()
     def save_ticket(self, idticket, bet_money, uid)->
     def data_to_display(self)-> -> dict()
     def find_tickets(…)-> -> data_to_display()
```

def find_tickets(task_list, margin_time, check_numlist)

Cette methode permet de trouver les meilleurs tickets en fonction des attributs d'instanciation.

ARGS ->

task_list -> list-> la liste des taches en prendre en compte lors de la selection des matchs. Si elle est vide, on considere toutes les taches lors du filtrage.

margin_time -> time->

ce parametre definit l'heure a laquelle l'utilisateur veut imprimer son ticket.

Ex -> avec skimind l'user construit un ticket à 11h alors qu'il a l'intention d'aller l'imprimer a 13h. alors il se peut que son ticket ne soit pas validé parce que certains matchs trouvés par skimindAI auraient seraient deja entrain de se jouer. Alors pour eviter cela, il vaut mieux que l'user puisse dire à quel moment il compte aller imprimer son ticket. Et tous les matchs qui se derouleront avant l'heure de l'impression seront ignores par l'algorithme.

check_numlist -> bool->

ce parametre est un simple indicateur booleen. Ce qui signifie ->

True -> l'agorithme doit grouper les meilleurs matchs par numlist (numero de la liste).

False -> l'algorithme considere tous les matchs comme appartenant à une meme liste. Dans ce cas, le numero de la liste (numlist) par defaut est utilisé.
default_numlist = 7

Return -> cette methode returne les données à afficher dans un dictionnaire en appellant la fonction data_to_display. (voir page 9)

Return -> JSON

```
{
	idticket-> // cette key est du type int et sa valeur est du type dict
	{
		“numlist”->int,
		“probability”->float, # la probabilité globale du ticket
		“bet_money”->int, # le montant parier par l’user
		“win_money”->int # le montant a gagner
		“matchs”->list# la liste des matchs
		[
			// chaque liste contient des valeurs correpondant a
			[
				idmatch->int, # le numero du match
				home->str, # le nom de l’equipe a domicile
				visitor->str, # le nom de l’equipe a l’exterieur
				cote->float, # cote attribué
				predict->int # la prediction
			]
		]
	}
}
```

`def save_ticket(idticket, bet_money, uid)`

cette fonction permet de sauvegarder un ticket validé par un user et toutes les infos qui vont avec.

ARGS ->

`idticket` -> int

le numero du ticket que l'utilisateur veut ajouter a sa liste des tickets

`bet_money` -> int

le montant qu'il pari pour ce ticket (si ce parameter n'est pas fourni, c'est le montant fourni lors de l'instanciation qui sera pris en compte -> self.bet_money)

`uid` -> str->

l'id de l'user dans la data base (si ce parameter n'est pas fourni, c'est le uid par defaut qui sera utilisé -> "skimindAI")

Utilisation

- Skimind
- test.py

```
from skimind.useCases.buildTicket import main as buildticket

# BUILD TICKET -> donnees
# ces donnees seront fournies par l'user
# via une interface html
bet_money = 1000
win_money = 50000
task_list = ["SECOND_MT", "FIRST_MT", "SECOND_MT", "MT_MORE_GOAL"]
margin_time = "15->00->45"

# instantiation de la classe BuildTicket
builder =  buildticket.BuildTicket(bet_money, win_money)
# le json : donnees a afficher
datas = builder.find_ticket(task_list,margin_time,check_numlist)
# puis les donnees sont envoyées a l'interface
#pour permettre a l'user de choisir les tickets a sauver

# sauvegarder un ticket choisi
idticket_choice = 1234 # l'user choisi le ticket n0 1234
bet_money = 5000 # l'user change sa mise

uid = "jeanluc243" # ceci n'est pas demandé a l'user
builder.save_ticket(idticket_choice, bet_money, uid)
```