Université de Montréal

TP 2

Université De Montréal

Informatique et Recherche Opérationnelle

IFT 3335 - Intelligence artificielle

Auteurs:
Martin El Mqirmi- ********

Date: March 29, 2022

Extraction des caractéristiques :

L'extraction des caractéristiques se fait dans le programme extract_data.py

- Fonction get_mot (path, longueur, dictX, dictY, vectorized) :
  1. **path** : chemin du fichier contenant les phrases
  2. **longueur** : nombre de mot avant et après le mot interest
  3. **dictX** : dictionnaire contenant tout les mots du fichier lu et associe à chaque mot un nombre différent
  4. **dictY** : dictionnaire contenant tout les sens du mot interest avec un chiffre associé à celui-ci
  5. **vectorized** : un booléen qui décide si la fonction retourne un tableau X contenant des tableaux de taille 2 × longueur avec les nombres associé à chaque mot (False), ou bien si il retourne un tableau X contenant des tableaux de taille du dictionnaire X et associe à chaque mot du dictionnaire la valeur 1 si il est dans le sac de mots autour du mot interest sinon la valeur 0 (True).

  Cette fonction retourne le contexte correspondant à l'ensemble des mots avant et les mots après (dans un sac de mots) sans ordre si vectorized = True sinon en ordre si vectorized = False.

- Fonction get_carac_mot (path, longueur, dictX, dictY, vectorized) :
  1. **path** : chemin du fichier contenant les phrases
  2. **longueur** : nombre de catégories avant et après le mot interest
  3. **dictX** : dictionnaire contenant toutes les catégories du fichier lu et associe à chaque catégorie un nombre différent
  4. **dictY** : dictionnaire contenant tout les sens du mot interest avec un chiffre associé à celui-ci
  5. **vectorized** : un booléen qui décide si la fonction retourne un tableau X contenant des tableaux de taille 2 × longueur avec les nombres associé à chaque catégorie (False), ou bien si il retourne un tableau X contenant des tableaux de taille du dictionnaire X et associe à chaque catégorie du dictionnaire la valeur 1 si il est dans le sac de catégories autour du mot interest sinon la valeur 0 (True).

  Cette fonction retourne le contexte correspondant à l'ensemble des catégories avant et les catégories après (dans un sac de catégories) sans ordre si vectorized = True sinon en ordre si vectorized = False.

- Fonction get_dictY (path) :

  1. **path** : chemin du fichier contenant les phrases

  Cette fonction retourne un dictionnaire contenant tous les sens du mot interest et à chaque sens un chiffre lui est associé. Par exemple si le mot interest_5 est présent dans le fichier, le dictionnaire contiendra la clé **interest_5** avec la valeur **5**

- Fonction get_dict_motX (path) :

  1. **path** : chemin du fichier contenant les phrases

  Cette fonction retourne un dictionnaire contenant tous les mots présent dans le fichier et à chaque mot un nombre unique lui est associé. Par exemple si le mot personal est présent dans le fichier, le dictionnaire contiendra la clé **personal** avec la valeur **1405** par exemple.

- Fonction get_dict_caraX (path) :

  1. **path** : chemin du fichier contenant les phrases

  Cette fonction retourne un dictionnaire contenant toutes les catégories présentent dans le fichier et à chaque catégorie un nombre unique lui est associé. Par exemple si la catégorie NNS est présente dans le fichier, le dictionnaire contiendra la clé **NNS** avec la valeur **10** par exemple.

Performances des différents algorithmes :

Algo Naives Bayes : naive_bayes.py

Les résultats suivant sont les résultats de 100 exécutions.

- **Algo Multinomial** :
  - Groupe de mots de taille 1 (taille 1 = 1 mot avant et 1 mot après) :
    - **vectorized** = True
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 388 (82 %)
    - **vectorized** = False
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 68 (14 %)
  - Groupe de mots de taille 2 :
    - **vectorized** = True
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 402 (85 %)
    - **vectorized** = False
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 108 (23 %)
  - Groupe de mots de taille 5 :
    - **vectorized** = True
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 401 (84.6 %)
    - **vectorized** = False
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474

            - **Bonne réponses** : 104 (22 %)
    - Groupe de catégories de taille 1 :
        - **vectorized** = True
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 269 (56.8 %)
        - **vectorized** = False
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 218 (46 %)
    - Groupe de catégories de taille 2 :
        - **vectorized** = True
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 285 (60 %)
        - **vectorized** = False
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 158 (33 %)

- **Algo Gaussien** :
    - Groupe de mots de taille 1 (taille 1 = 1 mot avant et 1 mot après) :
        - **vectorized** = True
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 347 (73 %)
        - **vectorized** = False
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 241 (50 %)

- Groupe de mots de taille 2 :
  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 380 (80 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 250 (53 %)
- Groupe de mots de taille 5 :
  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 368 (78 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 234 (50 %)
- Groupe de catégories de taille 1 :
  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 28 (6 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 245 (52 %)
- Groupe de catégories de taille 2 :
  - **vectorized** = True
    - **Nombre de train** : 1894

            - **Nombre de tests** : 474
            - **Bonne réponses** : 34 (7 %)
        * **vectorized** = False
            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 257 (54 %)

Algo Decision Tree : decision_tree.py

Les résultats suivant sont les résultats de 100 exécutions.

- **Groupe de mot de taille 1** :
  - **vectorized** = True
    * **Nombre de train** : 1894
    * **Nombre de tests** : 474
    * **Bonne réponses** : 387 (81.6 %)
  - **vectorized** = False
    * **Nombre de train** : 1894
    * **Nombre de tests** : 474
    * **Bonne réponses** : 378 (80 %)

- **Groupe de mot de taille 3** :
  - **vectorized** = True
    * **Nombre de train** : 1894
    * **Nombre de tests** : 474
    * **Bonne réponses** : 382 (81 %)
  - **vectorized** = False
    * **Nombre de train** : 1894
    * **Nombre de tests** : 474
    * **Bonne réponses** : 337 (71 %)

- **Groupe de mot de taille 5** :

  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 370 (78 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 326 (69 %)

- **Groupe de catégorie de taille 1** :
  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 266 (56 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 341 (72 %)

- **Groupe de catégorie de taille 3** :
  - **vectorized** = True
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 261 (55 %)
  - **vectorized** = False
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 331 (70 %)

Algo Multi Layer Perceptron : multi_layer.py

Les résultats suivant sont les résultats de 6 exécutions.
Pour cet algorithme, j'ai utilisé la fonction d'activation relu car d'après plusieurs lecture sur les fonctions d'activations, j'ai trouvé que relu était celle qui était très souvent la meilleure. J'ai fait des test avec les autres fonctions et je trouve bien que relu est la plus stable. Avec d'autre fonctions j'obtiens parfois de meilleure résultats mais j'obtiens plus souvent de moins bon résultats.

- **Groupe de mot de taille 1** :
  - **vectorized** = True
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 407 (86 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 408 (86 %)
    - **Nombre de neurones cachés** : 10
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 412 (87 %)
  - **vectorized** = False
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 213 (45 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 222 (47 %)
    - **Nombre de neurones cachés** : 10

            - **Nombre de train** : 1894
            - **Nombre de tests** : 474
            - **Bonne réponses** : 245 (52 %)

- **Groupe de mot de taille 3** :
  - **vectorized** = True
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 416 (88 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 412 (87 %)
    - **Nombre de neurones cachés** : 10
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 417 (88 %)
  - **vectorized** = False
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 192 (41 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 222 (47 %)
    - **Nombre de neurones cachés** : 10
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474

            - **Bonne réponses** : 220 (46 %)

- **Groupe de caractéristique de taille 1** :
  - **vectorized** = True
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 275 (58 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 274 (58 %)
    - **Nombre de neurones cachés** : 10
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 275 (58 %)
  - **vectorized** = False
    - **Nombre de neurones cachés** : 50
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 288 (61 %)
    - **Nombre de neurones cachés** : (30, 20, 30)
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 297 (63 %)
    - **Nombre de neurones cachés** : 10
      - **Nombre de train** : 1894
      - **Nombre de tests** : 474
      - **Bonne réponses** : 251 (53 %)

- **Groupe de caractéristique de taille 3** :

- **vectorized** = True
  - **Nombre de neurones cachés** : 50
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 287 (61 %)
  - **Nombre de neurones cachés** : (30, 20, 30)
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 283 (60 %)
  - **Nombre de neurones cachés** : 10
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 292 (62 %)
- **vectorized** = False
  - **Nombre de neurones cachés** : 50
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 286 (61 %)
  - **Nombre de neurones cachés** : (30, 20, 30)
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 309 (65 %)
  - **Nombre de neurones cachés** : 10
    - **Nombre de train** : 1894
    - **Nombre de tests** : 474
    - **Bonne réponses** : 262 (55 %)

De plus j'ai effectué des tests où je choisis aléatoirement la réponse et j'obtiens bien une probabilité d'environ $1/6 \simeq 17\%$. Comme tout les algorithmes donnent des réponses très supérieur à 1/6, nous pouvons conclure qu'ils apprennent.

Commentaires sur les résultats obtenus :

Naive Bayes :

Pour l'algorithme de Naive Bayes, j'observe que le plus concluant est d'utiliser l'algorithme multinomial de Naive Bayes puis c'est de prendre un groupe de mots autour (2 ou 3 mot avant et 2 ou 3 mot après). De plus il ne faut pas prendre en compte l'ordre des mots.

En prenant tous ces paramètres, j'obtiens environ 85% de bonnes réponses.
L'algorithme de Naive Bayes multinomial est donc assez efficace.

Arbre de décisions :

Pour l'algorithme de Decision Tree, j'observe que le plus concluant est de prendre un groupe de mots autour (1 ou 2 mot avant et 1 ou 2 mot après). De plus il ne faut pas prendre en compte l'ordre des mots.

En prenant tous ces paramètres, j'obtiens environ 81% de bonnes réponses.
L'algorithme de Decision Tree est donc assez efficace.

Autre remarque : Pour cet algorithme si nous prenons les catégories, des mots autour, sans tenir compte de l'ordre, l'algorithme donne un meilleur résultats que si on considère l'ordre des catégories.

Multi-Layer Perceptron :

Pour l'algorithme de Multi-Layer Perceptron, j'observe que le plus concluant est de prendre une seule ouche de neurones avec 10 neurones, car j'obtiens des résultats similaires tout en étant plus performant. De plus il faut prendre un sac de mots autour (2 ou 3 mot avant et 2 ou 3 mot après). Il faut aussi ne pas tenir compte de l'ordre des mots.

En prenant tous ces paramètres, j'obtiens environ 88% de bonne réponses.
L'algorithme de Multi-Layer perceptron est donc assez efficace.

Conclusion :

Tous les algorithmes sont assez efficaces, mais certains sont plus efficaces que d'autres. Je pense que le meilleur algorithme est l'algorithme de Multi-Layer Perceptron car il obtient les meilleurs résultats et je pense qu'avec plus de données d'entraînement il nous offrirait encore de meilleur résultats que les autres. Ensuite je pense que l'algorithme de Naive Bayes est meilleur que Decision Tree pour la désambiguïsation de sens de mots.