Introduction:

Pour mon projet du cours Traitement Automatique de la Langue Naturelle, Madame Merlo et moi-même trouvons l’idée bonne de participer à une tâche partagée proposer par SemEval. Nous avons choisi une tâche proposer en 2017 qui se nomme “*Task 8: RumourEval: Determining rumour veracity and support for rumours*”. Ici nous nous intéresserons seulement à la sous-tâche A à savoir:

l'analyse du discours environnant est de déterminer comment les autres utilisateurs dans les médias sociaux considèrent la rumeur à partir d’un tweet source. Pour cette sous il nous ait fourni une conversation structurée par un arbre formée de tweets répondant aux tweets rumeurs originaux, où chaque tweet présente son propre type de soutien à l'égard de la rumeur. Ces tweets et ces tweets réponses sont classés en termes de soutien, de refus, d'interrogation ou de commentaire (SDQC). Donc la sous-tâche a pour objectif de marquer le type d'interaction entre une déclaration donnée (tweet rumeur) et un tweet de réponse (ce dernier peut être une réponse directe ou imbriquée).

Les données de SemEval 2017 task 8:

Pour chaque tweet source nous avons plusieurs informations:

- Un json contenant le tweet source et toutes ses données relatives.

- Les tweets réponses dans un sous dossier avec leurs données relatives.

- L’arborescence des tweets réponses.

- Les urls qui sont cités dans le tweet sources et ces réponses.

Les urls ne nous servent pas. En effet la sous-tâche A est pensé close, nous ne pouvons utiliser que les données fournie par SemEval. Donc on doit seulement utiliser les textes des tweets (et d’autres méta-données fournies) pour les classifier.

Mes structures de données:

Afin d’utiliser au mieux les données j’ai décidé des objets. En effet les objets permettent de faire une enveloppe simple contenant facilement des attributs et des méthodes spécifique aux tweets sources et aux réponses. J’ai donc trois objets : *Data, Tweet, Reply*.

*Tweet* et *Reply* sont des sous-classes qui hérite de la super-classe *Data*.

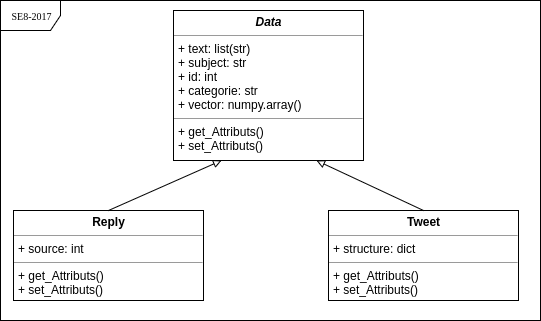
Chaque classe a une getter et un setter pour chacun de ses attributs. Les signatures des différentes classes sont les suivantes:

*- Data(data, subject, categorie)*

*- Tweet(data, subject, categorie, structure)*

*- Reply(data, subject, categorie, source\_tweet)*

Le paramètre *data* est un dictionnaire contenant le texte du tweet (*text*) et son identifiant (*id*) et les autres méta-données fournies. L’attribut *text* est pour le moment une liste de mot qui ont été décapitalisé, tokenisé puis lemmatisé. Le paramètre *subject* et *categorie* sont récupéré pour l’analyse de données et la classification. Le paramètre structure est un dictionnaire contenant l’arbre de réponses du tweet-source. Le paramètre *source\_tweet* permet de connaître d’où provient la réponses, à quel tweet source elle appartient; elle initialise l’attribut *source*. L’attribut *vector* et un tableau *numpy* infantilisé vide. Il accueillera lors de la vectorisation un vecteur créer à partir de *text*.



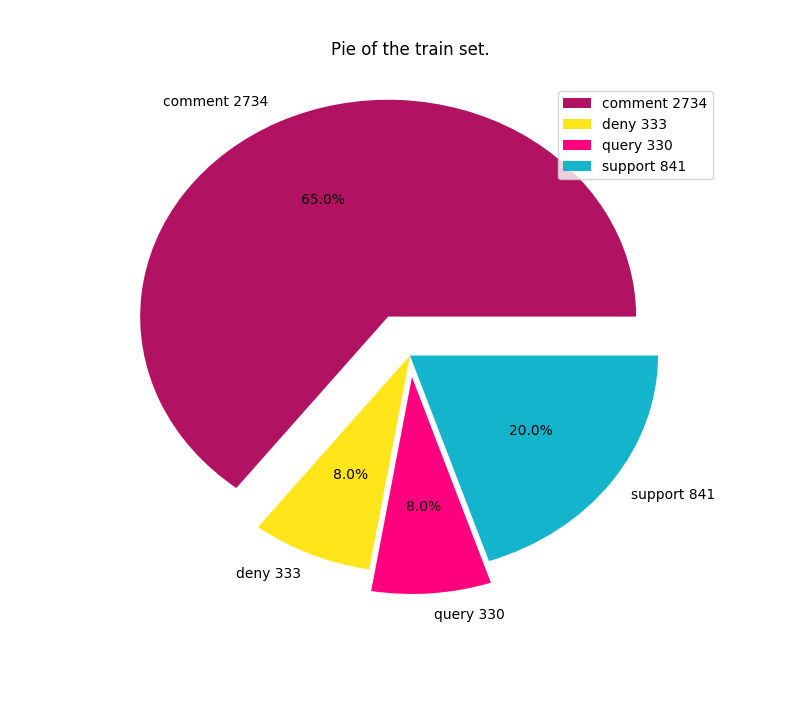
Analyse de données:

Pour effectuer à bien cette classification nous devons avoir une vue exhaustive des différentes dimensions de ce corpus. J’ai fait des analyses à partir du corpus d’entraînement sur trois niveaux:

- La répartition des catégories en général (qui inclut la répartition des des catégorie par sujet).

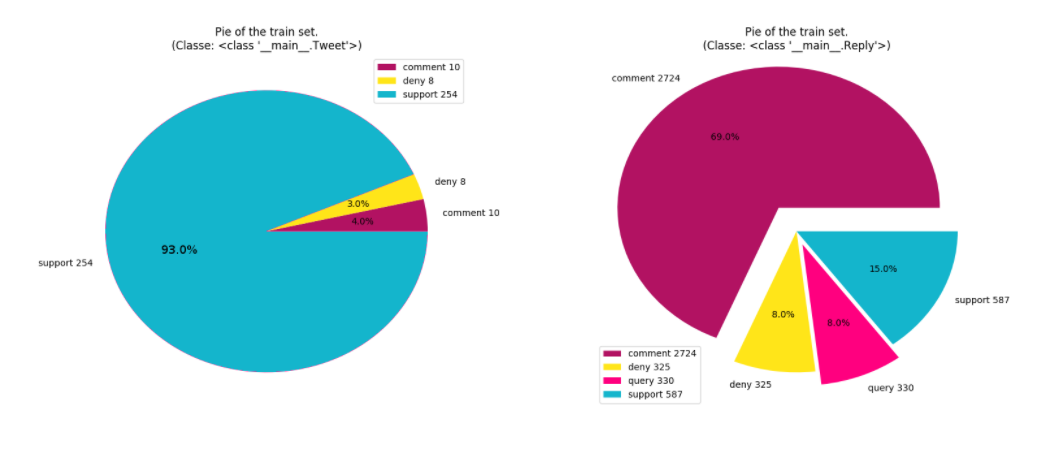
- La répartition de la fréquence des mots par catégorie

- La répartition des catégories par classe d’objet

Figure 1 Répartition des catégories dans le corpus d'entraînement

La répartition des catégories en général nous a permis de voir que la classe majoritaire était les commentaires. Ils sont suivis par les supports à 1/5 des données qui sont eux-même suivis par les refus et les questionnement aux même niveaux. Il y a donc un biais important en ce qui concerne la réparation des classes. Il faudra prendre en compte que les commentaires sont majoritaire et peut-être en faire notre catégorie par défaut. À partir de cette analyse nous faisons notre première baseline sur cette classe majoritaire. Il y a 65% de commentaires donc nos prochains buts seront de les dépasser en classant plus de tweet correctement. Pour les répartitions par sujet cela suit le tableau ci-dessus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Comment | Deny | Query | Support |
| charliehebdo | 67 | 22 | 5 | 5 |
| ebola-essein | 62 | 18 | 3 | 18 |
| ferguson | 66 | 16 | 9 | 8 |
| ottawashooting | 61 | 21 | 8 | 10 |
| prince-toronto | 71 | 22 | 11 | 7 |
| putinmissing | 53 | 29 | 8 | 10 |
| sydneysiege | 63 | 20 | 9 | 8 |



La répartition des catégories en fonction des classes objets marque une significative différence. Pour la classe des Reply la répartition est assez proche de la répartition générale. Par contre en ce qui concerne la classe des Tweet la classe grandement majoritaire sont les supports (le contraire aurait été bizarre vu que c’est le tweet qui lance la discussion sur la rumeur). Cela pourra être une information très utile pour classer les Tweets. Les Tweets source qui sont des support représente 30% du corpus entier.

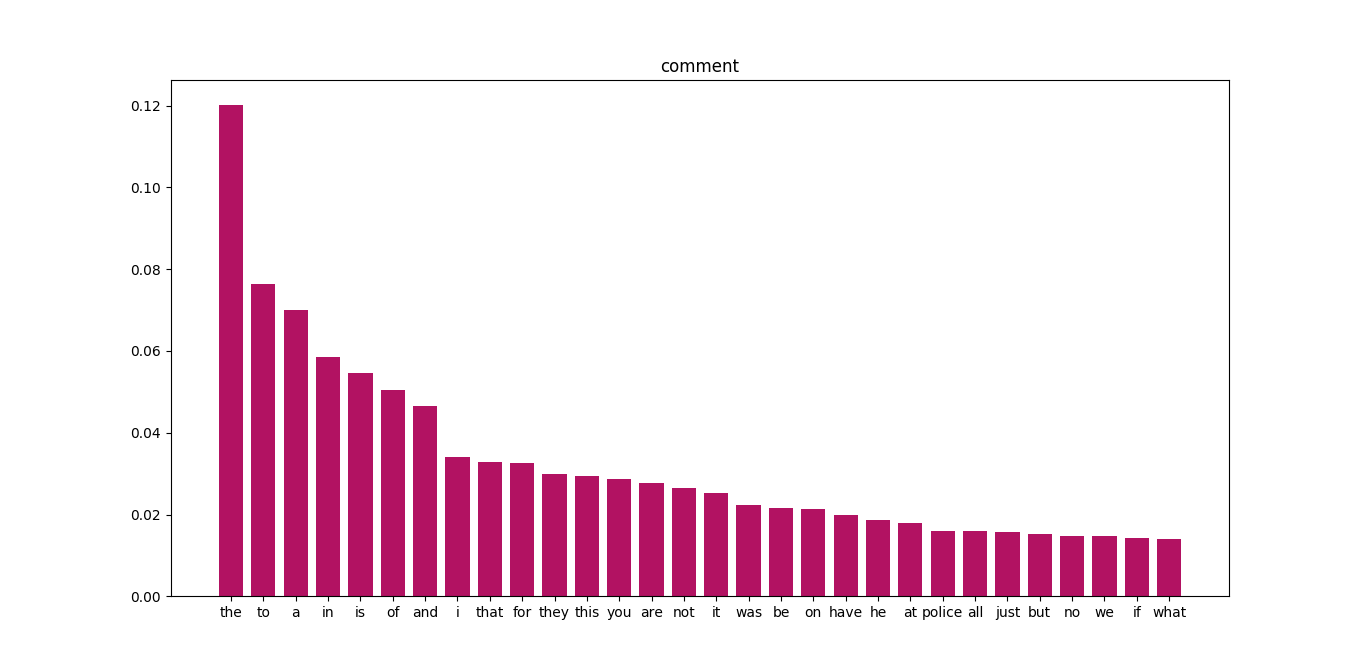
Ici nous avons représenter les distributions zipfienne de la fréquence des mots

par catégories ordonné du plus au moins fréquent. Sans grande surprise “the”

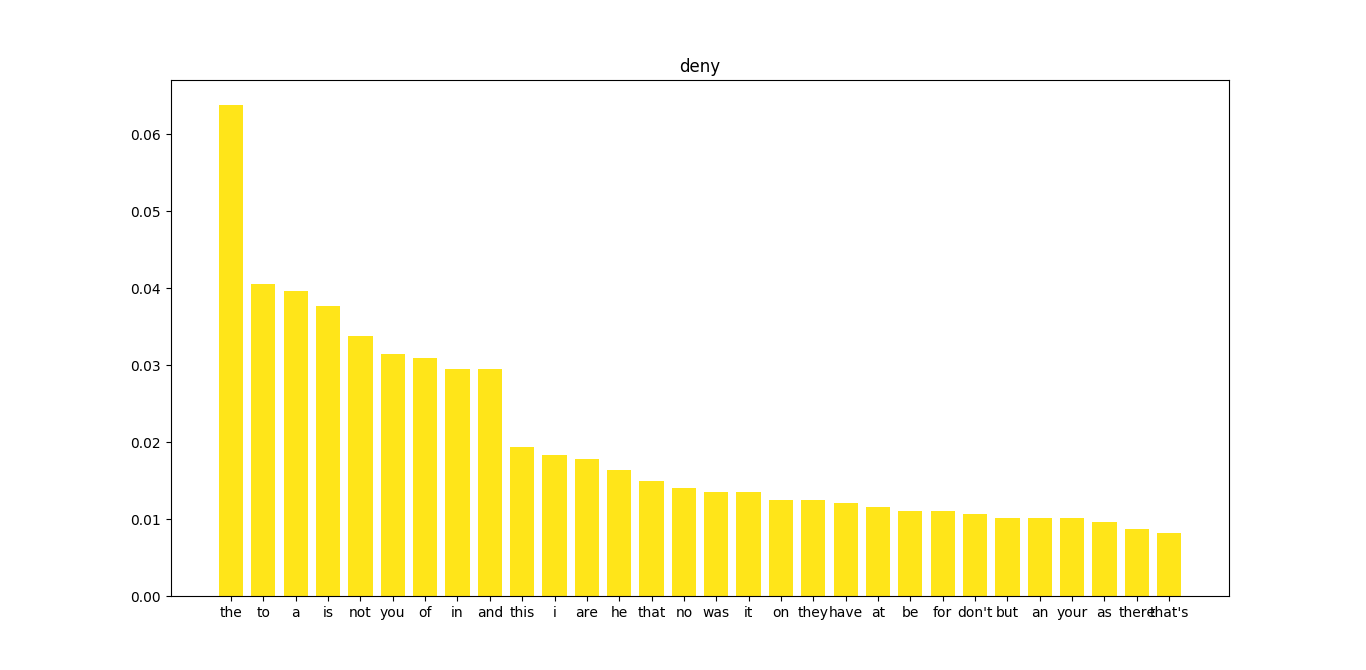
est le mot au premier rang toutes catégories confondues. “the” n’est pas un

trait distinctif, par contre tous les mots suivants sont différents ou n’ont pas le

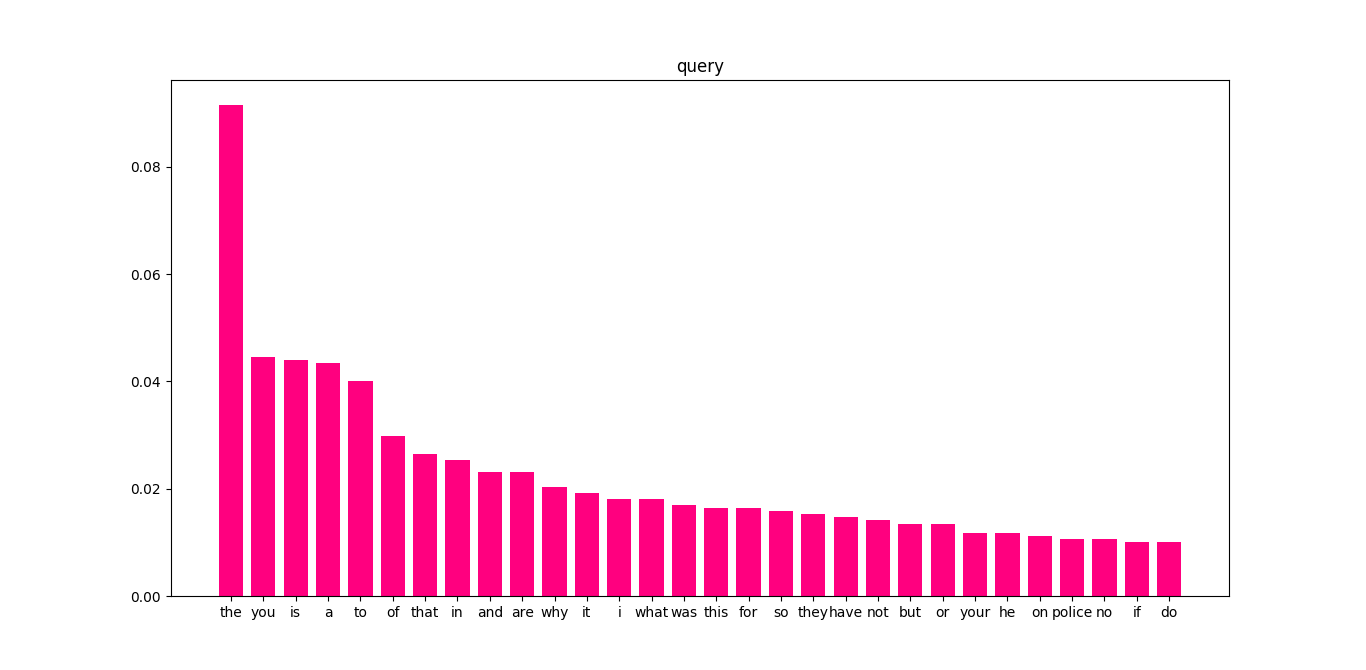
même rang selon la catégories.



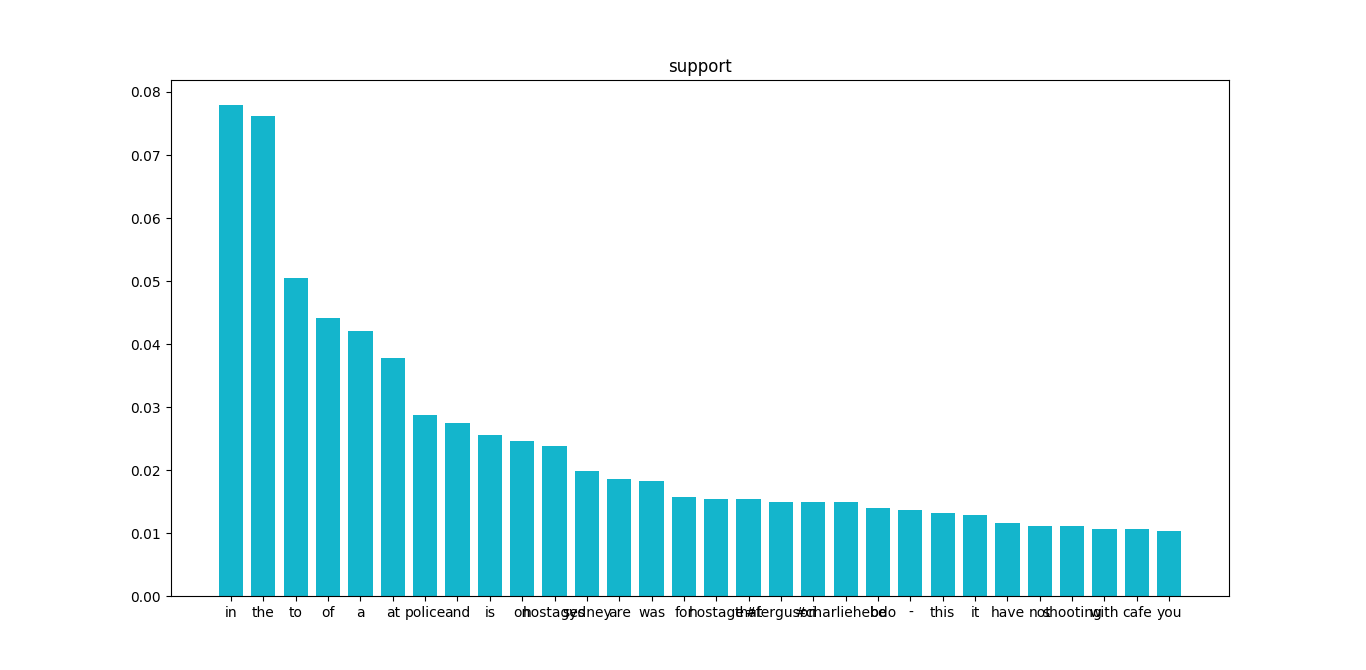
Le vocabulaire qui ressort en majorité des comment est un vocabulaire assez usuel.



On voit des mots de négation et un vocabulaire d’interpellation se démarquer.



Encore plus que la catégorie “deny”: On voit clairement des mots interrogatifs et un vocabulaire d’interpellation se démarquer.



Le support n’ont pas l’air d’avoir un vocabulaire clairement propre. Il

y a beaucoup de mots usuels (comme pour les “comment”).

Première approche du problème:

Vous retrouverez sous le répertoire “*old”* la première version du code que j’avais fait pour gérer le problème. À cette époque j’avais mal compris l’énoncé du problème et je ne pensais qu’il ne fallait que trier uniquement les réponses aux tweets et non les tweets sources eux-mêmes. J’avais donc un manque dans ma classification lors le *scorer* m’indiquait qu’il me manquait 272 classifications; ce qui correspond au nombre de tweets sources. Nous n’allons pas représenter les résultats que l’on avait obtenu avec notre corpus incomplet; car cela ne fait pas de sens de se baser sur des résultats incorrectes. Par contre certaines observations demeurent vraies. En effet dans cette solution nous utilisons simplement la fréquence des mots par catégorie. Ceci est pour calculer l’argmax sur la liste des produit des fréquences de mots par catégorie. On remarque entre autre que plus la probabilité des mots inconnus est lissé avec une valeur basse plus les prédictions de classes sont bonnes. Ce modèle est un algorithme naïve bayes en soit.

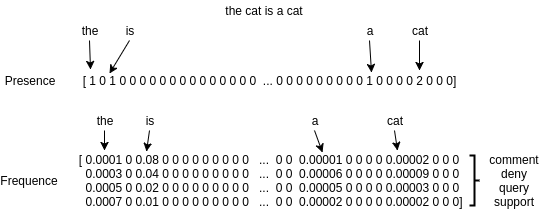
Avec un numérateur de 10-11 on obtient ces résultats là:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C:pred | comment | deny | query | support |
| comment | 124 | 18 | 18 | 13 |
| deny | 5 | 1 | 1 | 4 |
| query | 12 | 3 | 9 | 4 |
| support | 45 | 11 | 3 | 10 |

Soit 51.25% d’exactitude cdqs.

Deuxième approche du problème:

Dans cette seconde approche nous tentons des solutions de machine learning après s’être rendu compte de la complexité de la tâche dans cette première approche. Nous avons donc vectorisé informations textuelles de deux manières pour le moment:



Les vecteurs de présence contienne le nombre de fois que le mot est vus dans le tweet. Chaque dimension du vecteur représente un mot du vocabulaire total des fichiers d’entraînement.

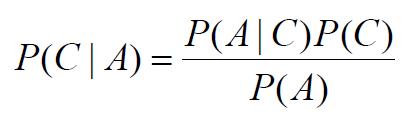
Les vecteurs de fréquence contiennent 4 fois plus d’informations que les vecteurs de présences chaque quart du vecteur représente la fréquence relative par catégories de chaque mot du tweet.

Modèles:

Pour chacune des vectorisations nous l’avons tester sur deux modèles. Nous allons ici vous les présenter:

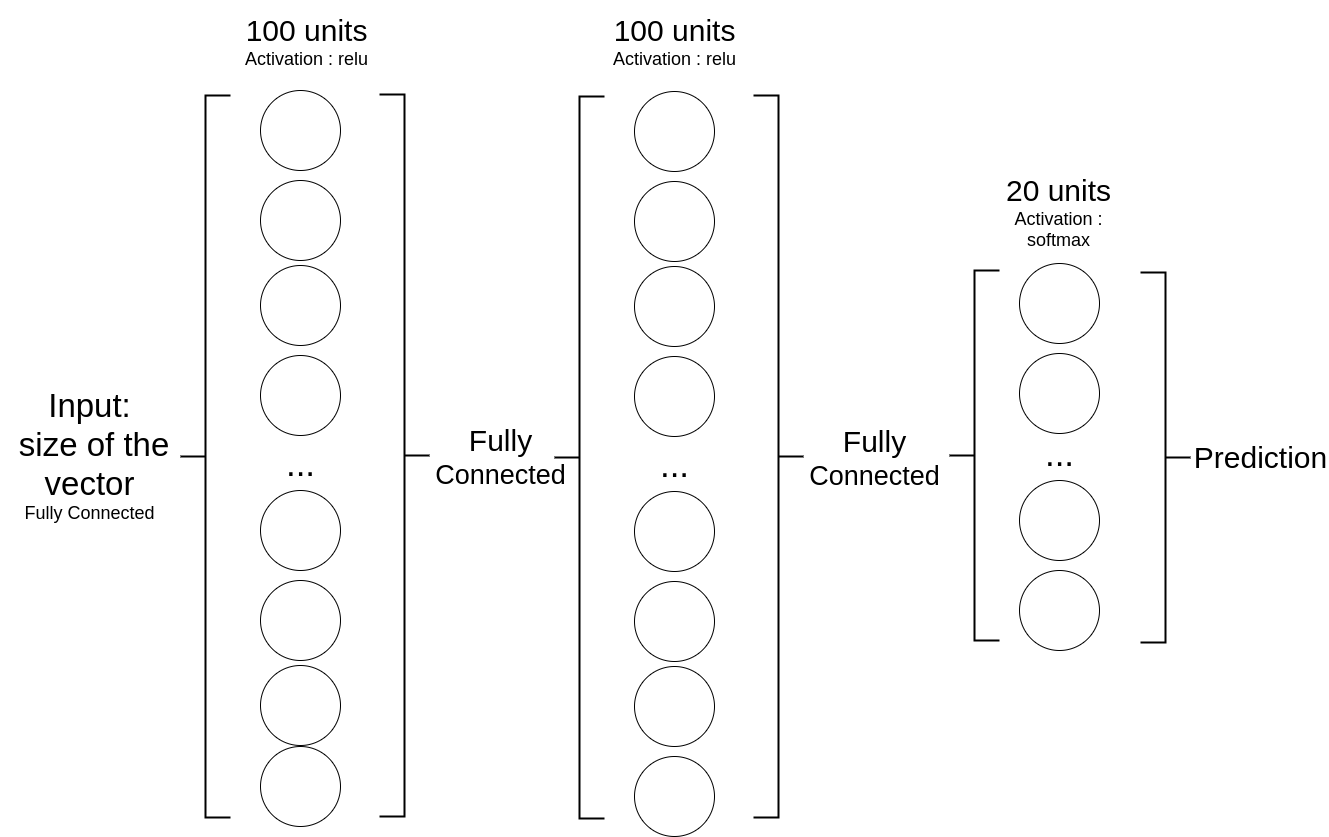
Naïve Bayes:

En utilisant la librairie Scikit-Learn nous avons créer un modèle naïve bayes utilisant une gaussienne pour la classification. Ce modèle prend des vecteur en entrée et ressort une classification en sortie.



Neural network:

Notre second modèle est un réseau neuronal. Celui-ci prend les mêmes entrées et donne les mêmes classification que le modèle précédent. Ce modèle neuronal contient 3 couche donc une couche cachée. Les 2 premières couches ont 100 unités et s’active avec “relu” et la dernière a 20 unités et s’active avec un “softmax”.



Résultats: