Bianca Bica (20161056)
Chaima Boussora (20159909)
Karine Nguyen (20160036)

IFT 3335 - Utiliser la classification pour la désambiguïsation de sens de mots

Tout d'abord, à priori d'évaluer la performance de différents algorithmes de classification, il a fallu effectuer un pré-traitement des données. Les objectifs de cette étape sont de préparer les données pour nos multiples algorithmes de classification, ce qui peut nécessiter la correction ou le retrait de certaines données selon un contexte donné, et de comprendre intuitivement certaines relations entre les caractéristiques et les étiquettes. Dans notre cas, cette étape préliminaire était nécessaire afin d'implémenter un programme permettant d'extraire les caractéristiques à partir des textes annotés, *interest.acl94.txt.* Lorsque nous parlons de caractéristiques, nous faisons référence aux mots ainsi qu'aux catégories.

```
def remove_line_filler(line):
    current_line = line.replace('=', '')
    current_line = current_line.replace(',/,', '')
    current_line = current_line.replace('[', '')
    current_line = current_line.replace(']', '')
    current_line = current_line.replace('\n', '')
    current_line = current_line.replace('/', ' ')
    return list(filter(None, current_line.split(' ')))  # removes all the whitespaces in line
```

Figure 1 : Les caractères retirés du texte annoté dus à un manque de pertinence et d'information

```
'geopolitical', 'interests_4', 'that', 'may', 'be', 'at', 'risk', "''", 'says', 'winston', 'lord', 'a', 'former', 'kissinger', 'protege', 'who', 'was',
'ambassador', 'to', 'china', 'from', '1985', 'to', '1989', '.'], ['mr.', 'lord', 'adds', 'that', 'although', 'he', 'does', "n't", 'fully', 'share', 'mr.',
'kissinger', "'s", 'sympathy', 'for', 'the', 'deng', 'regime', 'he', 'is', 'convinced', 'that', 'mr.', 'kissinger', "'s", 'views', 'are', "n't", 'influenced', 'by',
 'money', ':', '``', 'if', 'he', 'did', "n't", 'have', 'a', 'cent', 'of', 'commercial', 'interest_5', 'in', 'china', 'he', 'would', 'have', 'taken', 'the', 'same',
'position', '.']], [['NNS', 'IN', 'JJ', 'JJ', 'NNS', 'VBD', 'TO', 'VB', 'IN', 'NNS', 'IN', 'NN', 'NNS', 'VBP', 'JJ', 'NNS', 'IN', 'NN', 'NNS', '.'], ['JJR', 'NNS',
'VBP', 'VBN', 'TO', 'VB', 'VBG', 'NN', 'NNS', 'IN', 'PP', 'VBP', 'NN', 'NNS', 'TO', 'VB', 'RB', 'JJR', 'NNS', 'IN', 'DT', 'JJR', 'NN', '.'], ['RB', 'VBD', 'NP',
'NP', 'NP', 'NN', 'IN', 'NP', 'NP', 'NP', 'NNS', '``', 'MD', 'VB', 'RP', 'RB', 'IN', 'PP', 'VBP', 'RP', "''", 'IN', 'IN', 'JJ', 'NNS', 'IN', 'JJ', 'NN', 'NNS',
'.'], ['NP', 'NP', 'NN', 'NN', 'IN', 'NP', 'NP', 'CC', 'NP', 'WDT', 'VBZ', 'DT', 'AB', 'NN', 'NN', 'IN', 'DT', 'JJ', 'NN', 'VBD', 'VBN', 'DT', 'NN', '.'], ['NP',
```

Figure 2 : Liste contenant 2 sous-liste : une sous-liste de tous les mots et une sous-liste de toutes les catégories dans le texte annoté (un grossissement d'image est nécessaire)

En ce qui a trait à l'implémentation de notre pré-traitement, il peut être divisé en deux étapes. En premier lieu, de manière générale, nous apportons des modifications au texte annoté et nous séparons les mots aux catégories. Pour chaque phrase, nous retirons les paires (mot, catégorie) les moins intéressantes, c'est-à-dire des caractères qui sont peu informatifs et ne permettent pas de déterminer le sens du mot *interest*. La figure ci-dessus (Figure 1) montre les caractères qui seront retirés du texte annoté. Une fois que les retraits nécessaires ont été effectués, nous avons créé deux dictionnaires, soit un qui contient les mots associés à des chiffres en ordre croissant (clé) et soit un autre qui contient les catégories associées à des chiffres en ordre croissant. Par exemple, la clé 6 correspond au 6ème mot La première sous-liste contient des listes de tous les mots dans chaque phrase. De la même façon, la deuxième sous-liste contient des listes de toutes les catégories dans chaque phrase. La figure ci-dessus (Figure 2) illustre une partie de la sortie de la première étape du prétraitement. Nous avons déterminé le nombre de listes en fonction du nombre de groupes de caractéristiques. Nous avons décidé d'avoir deux listes, car dans l'énoncé, nous mentionnons que nous devons utiliser au minimum les deux groupes de caractéristiques suivant : les mots et les catégories.

L'objectif de ce travail est de tester la performance de différents algorithmes de classification, tels que le Naive Bayes, l'arbre de décision, la forêt aléatoire, le SVM et la Multilayer Perceptron. Pour évaluer la performance du Naive Bayes, nous avons créé une fonction qui prend en argument le nom du fichier annoté et

le type de caractéristique voulu, soit ‘words’ ou ‘categories’. Tout d’abord, nous effectuons l’extraction des données dépendamment de la caractéristique choisie. Les données que nous avons utilisées sont les sorties du prétraitement sur le texte original. L’extraction consiste d’une part, à ajouter les index des deux mots/catégories qu’on trouve avant et après le mot ambigu pour chaque phrase dans une liste, et d’autre part, de mettre toutes les instances du mot ambigu dans une liste. Les index de ces mots/catégories ont été déterminés en créant un dictionnaire. Étant donné que nous avons deux caractéristiques, par conséquent, nous avons deux dictionnaires. Pour chaque dictionnaire, la valeur correspond à un mot/une catégorie du texte annoté et la clé correspond à l’index de cet élément. En se basant sur l’implémentation du Naive Bayes fournie dans la documentation de Scikit-learn[1], nous définissons les paramètres X et Y pour les tests et l’entraînement. Étant donné qu’il n’est pas précisé dans l’énoncé du TP, 30% de l’échantillon est assigné au test (X_test, Y_test) et son complément est assigné à l’entraînement (X_train, Y_train). Ensuite, nous appelons le module ‘GaussianNB()’ de la librairie ‘sklearn’ avec les variables de test et d’entraînement en argument et nous utilisons la métrique ‘*Accuracy*’ afin de comparer ses variables. Nous avons choisi cette métrique, car elle est la plus répandue pour des problèmes de classification[2]. Notre accuracy doit se trouver entre les valeurs 0 et 1, où la valeur représente la fraction des échantillons classifiée correctement. On constate que le *accuracy* change dépendamment du nombre d'éléments qu’on retient avant et après le mot ambigu lors de l’étape d’extraction. Lorsque nous retenons deux mots avant et après le mot ambigu, on obtient une accuracy d’environ 49 % pour la caractéristique mots et 51 % pour la

[1] https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes
[2] https://realpython.com/train-test-split-python-data/

caractéristique catégories. La figure ci-dessous (Figure 3) illustre l'accuracy dans le cas où nous retenons 1 mot/catégorie, 2 mots/catégories et 3 mots/catégories avant et après le mot ambigu.

```
***Classifier: Naive Bayes with 1 word/category***
Feature: words
Accuracy: 48.65 %
Feature: categories
Accuracy: 46.23 %
***Classifier: Naive Bayes with 2 words/categories***
Feature: words
Accuracy: 49.21 %
Feature: categories
Accuracy: 51.42 %
***Classifier: Naive Bayes with 3 words/categories***
Feature: words
Accuracy: 41.59 %
Feature: categories
Accuracy: 48.50 %
```

Figure 3 : Le accuracy du Naive Bayes lorsque nous testons les différentes tailles de fenêtres de contexte (taille 1, 2 et 3)

Pour les algorithmes de classification qui suivent, pour des fins cohérentes, nous évaluons leurs performances en utilisant la métrique d'*accuracy* et nous retenons deux éléments avant et après le mot ambigu pour l'extraction des données.

Pour évaluer la performance de l'arbre de décision, les mêmes étapes ont été appliquées que le Naive Bayes, c'est-à-dire, l'extraction des données et la répartition des variables de tests et d'entraînements (30% de l'échantillon et 70%

de l'échantillon de assignés, respectivement, au test et à l'entraînement). Ensuite, on appelle le module 'DecisionTreeClassifier()' de la librairie 'sklearn' avec ses variables en argument pour par la suite calculer son accuracy. On constate que lorsqu'on utilise les mots comme caractéristique, son accuracy est environ 65% et son accuracy lorsqu'on utilise les catégories comme caractéristique est environ 67%.

```
*** Classifier: Decision Tree ***
Feature: words
Accuracy: 65.14 %
Feature: categories
Accuracy: 66.88 %
```

Figure 4 : L'accuracy de l'algorithme de l'arbre de décision

Pour évaluer la performance d'une forêt aléatoire, nous procédons de la même façon que les algorithmes de classifications précédents et on appelle le module 'RandomForestClassifier(), de la librairie 'sklearn' avec les variables de tests et d'entraînements en paramètre. Nous avons décidé que le nombre d'arbres dans notre forêt correspond à la valeur de défaut du module, soit 100 arbres. En conséquence, les valeurs d'*accuracy* obtenues pour la caractéristique des mots et des catégories sont d'approximativement 71% et 70% respectivement.

```
***Classifier: Random Forest***
Feature: words
Accuracy: 71.29 %
Feature: categories
Accuracy: 69.56 %
```

Figure 5 : L'accuracy de l'algorithme de la forêt aléatoire

Pour le classificateur SVM, nous procédons de la même façon que les autres classificateurs. Pour évaluer sa performance, nous avons décidé d'utiliser la classe 'SVC' et parmi les plusieurs types de *kernel*, nous avons décidé les kernels 'RBF' et 'Polynomial'. Le degré de la fonction de kernel polynomial prend la valeur défaut de 3. En appliquant le module 'SVC()' aux paramètres d'entraînement et de tests, l'*accuracy* du 'RBF' kernel pour les caractéristiques mots et catégories sont de 59% et 55% respectivement. L'*accuracy* du kernel polynomial de degré 3 pour les mêmes caractéristiques est de 57% et 55% respectivement.

```
***Classifier: SVM***
Feature: words
Accuracy (RBF Kernel):  55.08
Accuracy (Polynomial Degree 2 Kernel):  57.21
Accuracy (Polynomial Degree 3 Kernel):  56.26
Accuracy (Polynomial Degree 5 Kernel):  57.21
Feature: categories
Accuracy (RBF Kernel):  56.97
Accuracy (Polynomial Degree 2 Kernel):  55.08
Accuracy (Polynomial Degree 3 Kernel):  55.79
Accuracy (Polynomial Degree 5 Kernel):  58.16
```

Figure 6 : L'accuracy de l'algorithme de SVM pour les kernels 'RBF' et 'Polynomial'

Bianca Bica (20161056)
Chaima Boussora (20159909)
Karine Nguyen (20160036)

Pour la performance de l'algorithme Multilayer Perceptron, nous procédons aux étapes de l'extraction des données et à l'assignation des différents types de variables similaires aux autres classificateurs mentionnés ci-haut. Pour évaluer la performance de ce classificateur, nous avons utilisé le module 'MLPClassifier()' avec les variables d'entraînement et de tests en argument. Dans ce module, nous avons défini le nombre de couches cachées et le nombre de neurones ainsi que le nombre maximal d'itérations. Selon les ressources disponibles en ligne, le nombre de neurones cachés idéal est déterminé de la manière suivante[3]:

- Le nombre de neurones cachés se trouve entre la taille de la couche d'entrée et la taille de la couche de sortie
- Le nombre de neurones cachés correspond à la somme du 2/3 de la taille de la couche d'entrée et la taille de la couche de sortie
- Le nombre de neurones cachés est inférieur au double de la taille de la couche d'entrée

Pour déterminer le nombre de couche cachés idéal pour notre réseau de neurones, la manière de procéder est la suivante :

- 0 couches cachées : seulement capable de représenter les fonctions et les décisions de façon linéaire
- 1 couche cachée et plus : peut approximer toute fonction contenant un mapping continu d'un espace fini à un autre

Nous avons testé si nos données sont séparables linéairement avec le classificateur SVM. Nous avons utilisé le module 'LinearSVC()' de la librairie 'sklearn' et nous avons obtenu une valeur d'*accuracy* très basse pour les caractéristiques, mots et catégories, soit 20% pour les deux. En conséquence, étant donné que nos données ne sont pas séparables linéairement, le

[3] https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3

classificateur Multilayer Perceptron prend au moins une couche cachée. Afin d'observer sa performance, nous avons testé ce classificateur sur plusieurs réseaux de neurones. Par exemple, nous avons un réseau de neurones comportant 3 couches cachées, chacune contenant 10 neurones (on a l'attribut 'hidden_layer_sizes=(10, 10, 10)').. Nous avons aussi un réseau de neurones de 1 couche cachée comportant 10 neurones sur cette couche (on a l'attribut 'hidden_layer_sizes=10'). et un autre réseau de neurones de 2 couches cachées, chacune contenant 10 neurones (on a l'attribut 'hidden_layer_sizes=(10, 10)'). Pour évaluer la performance de ce classificateur, nous avons utilisé le module 'MLPClassifier()' avec les variables d'entraînement et de tests en argument. Dans ce module, nous avons défini le nombre de couches cachées et le nombre de neurones ainsi que le nombre maximal d'itérations. À 3 couches cachées, nous avons un *accuracy* de 55% et 63% pour les caractéristiques mots et catégories. À 2 couches cachées, nous avons un accuracy de 54% et 65% et à 1 couche cachée, nous avons un accuracy de 49% et 55%.

```
***Classifier: Multi Layer Perceptron with 3 hidden layers***
Feature: words
Accuracy: 55.05 %
Feature: categories
Accuracy: 62.62 %
***Classifier: Multi Layer Perceptron with 2 hidden layers***
Feature: words
Accuracy: 54.26 %
Feature: categories
Accuracy: 64.83 %
***Classifier: Multi Layer Perceptron with 1 hidden layers***
Feature: words
Accuracy: 48.90 %
Feature: categories
Accuracy: 55.21 %
```

Bianca Bica (20161056)
Chaima Boussora (20159909)
Karine Nguyen (20160036)

Figure 7 : L'accuracy de l'algorithme de Multilayer Perception

Lorsque nous comparons la valeur d'*accuracy* de tous les classificateurs, au niveau du Naive Bayes, on constate qu'on obtient un meilleur *accuracy* lorsque le nombre de mots/catégories qu'on retient avant et après le mot ambigu est de 2. On obtient une valeur d'*accuracy* fort supérieur de cette manière comparativement à lorsqu'on retient 1 mot/catégorie ou 3 mots/catégories avant et après le mot ambigu. De plus, on constate que la valeur d'*accuracy* ne dépasse pas 51%, ce qui signifie que seulement la moitié des échantillons sont classifiés correctement. Pour le classificateur de l'arbre de décision, on obtient une valeur d'accuracy similaire peu importe le type de caractéristique qu'on prend en compte, soit aux alentours de 65%-67%. Ses valeurs étant au-dessus de 50% signifie que 65%-67% de l'échantillon est correctement classifiés. Pour la forêt aléatoire, similaire à l'arbre de décision, les 2 caractéristiques ont un accuracy similaire, soit 70%-71%. Cela signifie que 70%-71% de l'échantillon est classifiées correctement. Pour le classificateur SVM, on constate que les 2 kernels choisis, soit le RBF kernel et le Polynomial kernel, retournent une valeur d'accuracy similaire pour une caractéristique donnée. En observant la figure 6, on est en mesure de classifier correctement 59% de l'échantillon avec un kernel RBF et 57% avec un kernel Polynomial dans le cas où la caractéristique choisie est les mots. De plus, on arrive à classifier correctement 55% de l'échantillon avec les kernels RBF et Polynomial si la caractéristique choisie est les catégories. Pour l'algorithme Multilayer Perception, dans le cas où nous choisissons les mots comme caractéristique, on obtient le meilleur accuracy lorsque nous avons un réseau de neurones de 3 couches cachées, soit 55% de

l'échantillon est classifié correctement. Dans le cas où nous choisissons les catégories comme caractéristiques, on obtient le meilleur accuracy lorsque nous avons un réseau de neurones de 2 couches cachée, soit 65% de l'échantillon est classifié correctement. Pour résumer, d'une part, si on considère la caractéristique des mots et des catégories, l'algorithme de forêt aléatoire a la valeur d'accuracy la plus élevée, à 0.7129 et 0.6956 (Figure 5). Ces résultats peuvent s'expliquer par les différents avantages que proposent le classificateur de forêt aléatoire. Celui-ci est très puissant et précis et a une très bonne performance sur tout type de problèmes, notamment les modèles non linéaires. Ce classificateur a l'avantage de pouvoir attribuer des poids spécifiques (équilibrer la pondération des classes) aux ensembles de données en fonction de la fréquence des classes. Un désavantage par rapport à cette méthode est le fait que nous devons choisir le nombre d'arbres générés manuellement, et ceci peut impacter le score d'*accuracy*. Dans notre cas, nous avons seulement testé une forêt qui contient 10 arbres de décision, mais nous aurions pu améliorer ceci en ajoutant davantage d'arbres, étant donné que plus il y a d'arbres, plus on augmente la performance de l'algorithme. De plus, le classificateur de la forêt aléatoire est plus précis que le classificateur Decision Tree en raison de son agrégation de plusieurs arbres de décision, contrairement à un seul arbre. Cet algorithme permet de limiter le surentraînement ainsi que de réduire le biais. D'ailleurs, nous pouvons bien observer avec nos résultats que Random Forest a généré de meilleurs résultats que Decision Tree (Figure 4). D'un autre côté, nous avons aussi implémenté le classificateur Naive Bayes, qui nous a retourné les meilleurs résultats pour 2 mots avant et après le mot cible (Figure 3). Théoriquement, un modèle qui prend en considération les 3 mots avant et après

le mot cible devrait avoir de meilleurs résultats qu'un modèle à 2 mots. Cependant, en pratique, ce n'est pas toujours le cas car lorsqu'on considère 3 mots avant et après, il y a beaucoup plus de sens et d'interprétations possibles à traiter, rendant le problème de classification plus complexe. Il est à noter que ce classificateur n'a pas été aussi performant que Decision Tree et Random Forest. Évidemment, le problème le plus commun du Bayes Naïf part de l'hypothèse d'indépendance que l'on fait afin d'exécuter l'algorithme: il se peut que les caractéristiques ne soient pas indépendantes l'une de l'autre pourvu une certaine étiquette. C'est d'ailleurs notre cas puisque les catégories dépendent des mots, soit nos 2 caractéristiques. Cela explique donc bien pourquoi nous avons obtenu un résultat plus faible pour ce classificateur. Par rapport au classificateur SVM, nous avons tout d'abord essayé de déterminer si nos données sont séparables linéairement et ce n'était pas le cas. Nous avons tout de même obtenu de différents résultats pour des noyaux polynomiaux de degré 2, 3 et 5 (Figure 6), le dernier ayant un *accuracy* supérieur aux autres. Nous pouvons expliquer ceci par le fait que nous avons 6 classes et notre fonction de classification risque d'être de degré élevé. Cependant, nous n'avons pas pu tester pour un degré supérieur à 5 puisque celui-ci prenait trop de temps à s'exécuter, soit un désavantage commun associé au SVM polynomial. Comme le classificateur SVM nous a permis de déterminer que nos données ne sont pas linéairement séparables, nous avons implémenté plusieurs variations de couches cachées et de neurones par couches cachées. Lorsqu'on évalue seulement la caractéristique des mots, nous obtenons un meilleur résultat avec 3 couches cachées (Figure 7) et pour la caractéristique des catégories, nous obtenons un meilleur résultat avec 2 couches cachées. Le problème avec ce classificateur est le fait qu'il est difficile

de déterminer le bon nombre de couches cachées ainsi que le nombre de neurones que celles-ci doivent contenir. Une amélioration possible serait d'effectuer des calculs plus spécifiques à notre problème afin de déterminer des valeurs qui maximisent nos résultats.