IFT 3335 - Introduction à l'intelligence artificielle
Devoir 2

Étienne Ameye
Calin Popa

1 mai 2022

Rapport

L'objectif de ce travail est d'entrainement différents modèles d'apprentissage automatique sur un ensemble de phrase pour prédire le sens d'un mot cible. Les modèles à entrainer sont un classifieur de Bayes naïf, un arbre de décision et un perception multicouche. Pour chacun des modèles, nous allons tester plusieurs choix d'hyper-paramètres et de caractéristiques et les comparer pour chercher à savoir quel sont les choix optimaux.

Préparation des données

La première chose à faire, avant de commencer à entrainer les différentes modèles, est de préparer les données. Notre ensemble de données d'entrainement est un ensemble de phrase, dans lesquelles chaque mot a été séparé et identifié par sa catégorie. Voici un exemple :

```
[ yields/NNS ] on/IN [ money-market/JJ mutual/JJ funds/NNS ] continued/VBD to/TO slide/VB ,/,
amid/IN [ signs/NNS ] that/IN [ portfolio/NN managers/NNS ] expect/VBP [ further/JJ declines/NNS
] in/IN
[ interest_6/NN rates/NNS ] ./.

$$

==================================== [ finmeccanica/NP ] is/VBZ [ an/DT italian/JJ state-owned/JJ
holding/VBG company/NN ] with/IN [ interests_5/NNS ] in/IN [ the/DT mechanical/JJ engineering/NN

industry/NN ] ./.
```

Le mot dont on doit trouver le sens est `interest`, son sens est indiqué juste après, par un numéro entre 1 et 6.

Sous cette forme, les données ne sont pas utilisables pour nos modèles d'apprentissage automatique. Il faudrait placer chaque ligne dans une matrice, et séparer les mots et leur catégories. Il faut aussi se demander comment représenter les mots et les catégories. Toutes ces décisions font parti du processus d'optimisation des modèles ; c'est ce qu'on appelle l'extraction des caractéristiques. Idéalement, on voudrait avoir une méthode facilement modifiable pour l'extraction des caractéristiques afin de tester différents choix de caractéristiques.

Pour cette raison, nous avons créé deux classes paramétrisables pour l'extraction des caractéristiques et une structure de donnée pour représenter ces caractéristiques. Le processus d'extraction se fait en deux temps : d'abord l'extraction des mots, puis l'extraction des caractéristiques. Comme chacune de ces étapes est important dans le processus d'optimisation, elles sont détaillées ci-dessous.

Tout le code relatif à la préparation des données ce trouve dans le fichier `extractors.py`.

Extraction des mots

L'extraction des mots consiste à lire les phrases et séparer les mots et les catégories. On veut aussi extraire le mot cible (`interest`) et sa classe (son sens). Les mots sont représentés par des tuples `[mot,catégorie]`. Dans le cas du mot cible, le tuple est `[classe,catégorie]`.

Les tuples sont enregistrés dans des matrices ; chaque ligne est une phrase et chaque colonne, un tuple. Nous avons le choix d'utiliser 2 matrices pour les mots de la phrase ; une matrice pour les mots avant la cible et une matrice pour les mots après. L'idée est qu'à cette étape nous voulons garder l'ordre des mots dans la phrase. Finalement, nous avons une troisième matrice pour le tuple de la cible dans chaque phrase.

Nettoyage des données

La lecture des phrases se fait assez simplement à l'aide d'expressions régulières. Cependant, nous profitons de cette étape pour effectuer un certain nettoyage des données. Les différentes étapes du nettoyage sont les suivantes :

- Éliminer les groupes du nom : Les groupes du nom sont identifiés dans chaque phrase, mais nous ne savons pas comment représenter cette information. Nous préférons donc l'éliminer à cette étape.
- Éliminer les groupes sans catégories : Cette étape est faite plus par précautions. Nous voulons éviter qu'un mot sans catégorie apporte de la confusion dans nos caractéristiques.
- Éliminer la ponctuation : La ponctuation ne contient que très peu d'information sur le sens d'un mot dans une phrase (à noter que ce n'est pas toujours le cas).
- Éliminer certaines catégories : Certaines catégories représentent des symboles de ponctuation, mais ne sont pas détecter par nos expressions régulières de ponctuation. Nous les éliminons donc à cette étape.

En plus du nettoyage des données, nous avons aussi la possibilité de faire du stemming et d'éliminer les stopwords. Pour le stemming, nous utilisons le module de NLTK. Pour les stopwords, nous utilisons une liste (disponible ici) à laquelle nous avons rajouté quelques mots. La liste complète se trouve dans le fichier `stopwords.txt`.

Nous avons du faire plusieurs choix pour décider du nettoyage des données. Ces choix ont une influence sur l'entrainement des modèles, et seront donc détaillé plus tard, lorsque nous parlerons des performances des modèles. Pour l'instant, notons quand même que nous avons fait le choix de ne pas éliminer les nombres présents dans les phrases.

Extraction des caractéristiques

Une fois les mots extraits des phrases, nous pouvons extraire les caractéristiques.

La première étape de l'extraction des caractéristiques consiste à sélectionner les mots qui se trouve dans une fenêtre autour de la cible. On peut faire varier la taille de cette fenêtre pour choisir plus ou moins de mots. Tous les mots en dehors de la fenêtre seront ignorés. S'il n'y a pas assez de mots dans la phrase pour couvrir toute la fenêtre, nous ajoutons des mots vides aux extrémités.

Nous enregistrons les mots dans la fenêtre, leur catégorie et les informations sur la cible dans une structure de données `Features`. Nous n'effectuons rien d'autre pour l'extraction des caractéristiques parce que nous voulons nous laisser le plus de flexibilité possible lors de la représentation des caractéristiques.

Représentation des caractéristiques

Une fois les caractéristiques extraites de chaque phrase, elles sont enregistrées dans une structure de données `Features`. L'avantage de cette structure de données est qu'elle nous donne beaucoup de flexibilité pour la représentation des caractéristiques. Nous avons le choix entre les représentations suivantes :

- Sac de mot : La représentation en sac de mot se base sur la classe `CountVectorizer` de `Scikit-Learn`.
- Vecteur onehot : La représentation en vecteur onehot utilise le vocabulaire calculé par `CountVectorizer`.
- Par index : La représentation par index assigne un index à chaque mot/catégorie, en fonction du vocabulaire calculé par `CountVectorizer`.

Nous détaillerons plus tard les choix que nous avons fait pour chaque modèle.

Analyse des résultats

Arbre de décision

Le modèle d'arbre de décision est un modèle d'apprentissage automatique utilisé dans la classification et la désambiguation du sens de mots. Pour ce travail le paramètre qui a été passé a l'arbre était que l'arbre devait être ¨balancé¨. Ce qui veut dire que pour chaque nœud, le sous-arbre droit et gauche ont une grandeur qui diffère de 1.

Les attributs utilisées pour ce modèle sont ¨BagOfWords¨ et ¨CatOneShot¨.

Sans Stemming, on remarque que la precision est faible au début (fenetre=1), elle monte rapidement quand la fenetre devient 2 et elle redescend ensuite. Donc le resultat pour la fenetre 2 est très bon comparé aux autres résultats.
Avec Stemming, la valeur de précision pour la fenetre 1 augmente juste un peu mais la précision pour la fenetre 2 diminue de beaucoup et les autres valeurs restent soit égales soit elles diminuent a leur tour, donc le stemming nous nuit plus qu'il nous aide.

En conclusion, la meilleure fenêtre est de 2 peu importe si on a du stemming ou non et il est mieux de faire notre modele sans stemming. On pourrait essayer d'améliorer le modèle en enlevant la contrainte qui dit l'arbre doit etre balancé et en limitant la profondeur de l'arbre.

Classifieur de Bayes naïf

Notre modèle du classifieur de Bayes naïf sera entrainé en utilisant la représentation « Bag of words » pour les mots autour de la cible et la représentation « Onehot » pour les catégories. On utilise le modèle de Bayes naïf multinomial.

Nous testerons l'extractions des mots avec et sans stemming et stopwords. Dans chaque, nous testerons aussi plusieurs tailles de fenêtre de contexte pour trouver la taille optimale.

Afin d'éviter le sur-apprentissage, nous avons séparé notre jeu de données en ensemble d'entrainement, de validation et de test. L'ensemble d'entrainement sera utilisé pour entrainer le modèle. L'ensemble de validation sera utilisé pour l'optimisation des hyper-paramètres. Finalement, nous évaluerons les performances du modèle optimal avec l'ensemble de test. L'ensemble d'entrainement contient 50% des données, et ceux de validation et de test contiennent 25% chacun.

Tout le code relatif au classifieur de Bayes naïf se trouve dans le fichier `NaiveBayes.ipynb`.

Cas de base

Pour le cas de base, nous n'utilisons ni stopwords, ni stemming.

La figure 1 montre la précision du modèle pour différentes tailles de fenêtre. La taille optimale est de 2, avec une précision sur l'ensemble de validation de 74.669%.

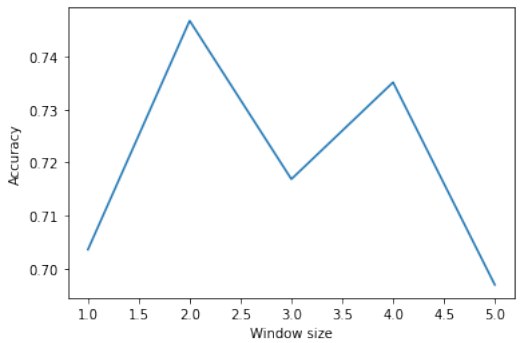

FIGURE 1 – Précision en fonction de la taille de fenêtre

La précision sur l'ensemble de test pour ce modèle est de 75%, soit mieux que la précision sur l'ensemble de validation (quoique très peu différente). La figure 2 montre la matrice de confusion sur les prédictions avec l'ensemble de test. On remarque que le modèle a tendance à trop souvent prédire la classe 6.

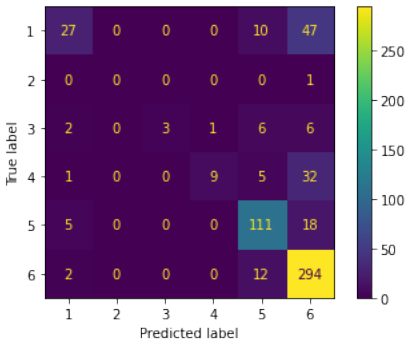

FIGURE 2 – Matrice de confusion sur l'ensemble de test

Avec stopwords

Ensuite, nous effectuons les mêmes tests, mais en ajoutant une liste de stopwords à l'extraction des mots.

La figure 3 montre la précision du modèle pour différentes tailles de fenêtre. La taille optimale est de 4 et la précision sur l'ensemble de validation est de 74.338%. Il y a donc une différence sur la taille de fenêtre optimale lorsqu'on utilise des stopwords, mais la précision reste sensiblement la même.

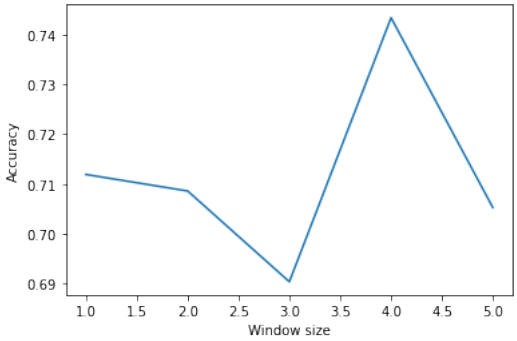

FIGURE 3 – Précision en fonction de la taille de fenêtre

La précision de ce modèle sur l'ensemble de test est de 73.818%. L'utilisation des stopwords semble rendre le modèle moins performant. La figure 4 montre la matrice de confusion des prédictions sur l'ensemble de test. On remarque qu'encore une fois le modèle prédit trop souvent la classe 6.

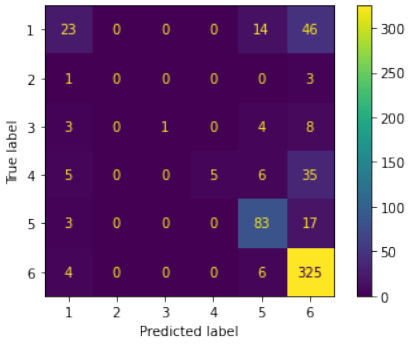

FIGURE 4 – Matrice de confusion sur l'ensemble de test

Avec stemming

Nos derniers tests sont effectués en faisant du stemming sur les mots.

La figure 5 montre la précision du modèle pour différentes tailles de fenêtre. La taille optimale est de 3. Pour cette taille, la précision sur l'ensemble de validation est de 75.993%.

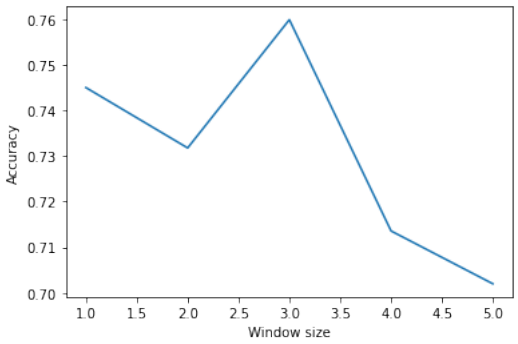

FIGURE 5 – Précision en fonction de la taille de fenêtre

Pour ce modèle, la précision sur l'ensemble de test est de 75.338%. C'est donc notre modèle le plus performant. La figure 6 montre la matrice de confusion sur les prédictions de l'ensemble de test.

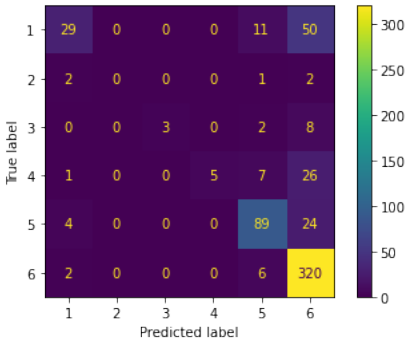

FIGURE 6 – Matrice de confusion sur l'ensemble de test

Conclusion

Le modèle du classifieur de Bayes naïf semble être trop sensible au déséquilibre de l'ensemble de données. La précision optimale est de 75.338%, mais toutes les erreurs commises sont sur la classe 6, qui est la classe la plus présente dans notre ensemble de données. Les features utilisent n'ont qu'un impact marginale sur les performances du modèle.

Pour avoir une meilleure idée des performances du modèle, il refaire les tests avec un ensemble de données plus équilibré.

Perceptron multicouche

Notre modèle du perceptron multicouche sera entrainé en utilisant la représentation « Bag of words » pour les mots autour de la cible et la représentation « Onehot » pour les catégories. On utilise la descente de gradient stochastique sur une fonction de cout logistique avec un taux d'apprentissage fixé à [0,1].

Nous testerons plusieurs tailles de réseaux afin de trouver la taille optimale. Nous testerons avec une profondeur (nombre de couche) de 1 à 5 et pour chaque cas, nous testerons avec une largeur de 1 à 10. Nous

testerons aussi plusieurs tailles de fenêtre de contexte, de 1 à 5, et nous essayerons plusieurs configurations de l'extraction des mots.

Afin d'éviter le sur-apprentissage, nous avons séparé notre jeu de données en ensemble d'entrainement, de validation et de test. L'ensemble d'entrainement sera utilisé pour entrainer le modèle. L'ensemble de validation sera utilisé pour l'optimisation des hyper-paramètres. Finalement, nous évaluerons les performances du modèle optimal avec l'ensemble de test. L'ensemble d'entrainement contient 50% des données, et ceux de validation et de test contiennent 25% chacun.

Tout le code relatif au perceptron multicouche se trouve dans le fichier `MultiLayerPerceptron.ipynb`.

Cas de base

Pour le cas de base, nous n'utilisons ni stopwords, ni stemming.

La figure 7 montre la précision optimale du modèle pour différente taille de fenêtre. La taille optimale est 3. On remarque cependant que la variation de précision est très faible, environ 3%.

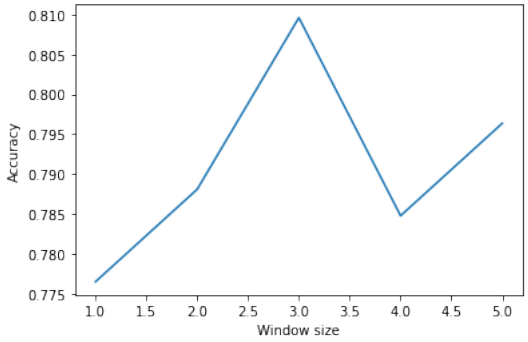

FIGURE 7 – Précision optimale en fonction de la taille de fenêtre

La figure 8 montre la précision optimale du modèle en fonction de la taille du réseaux de neurones. L'axe des x donne la largeur de chaque couche et chaque courbe représente une profondeur de réseau. La taille optimale du réseaux est d'une seule couche caché avec 5 neurones.

On remarque qu'avec une profondeur ≥ 3, la précision du modèle est beaucoup plus faible. De plus, la précision est presque toujours meilleure avec 1 seule couche caché qu'avec 2. Il semble donc que les réseaux plus petits soient les plus performants.

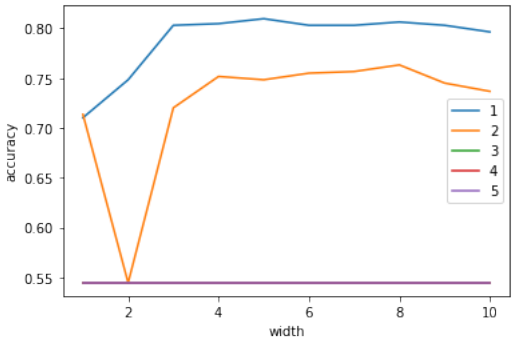

FIGURE 8 – Précision optimal en fonction de la taille du réseau

La précision optimale du modèle sur l'ensemble de test est de 78,885%, soit légèrement inférieure à celle sur l'ensemble de validation (de 80,96%). La figure 9 montre la matrice de confusion des prédictions sur l'ensemble de test. On remarque que le modèle prédit trop souvent la classe 6, probablement parce qu'elle est beaucoup plus représentée dans nos données.

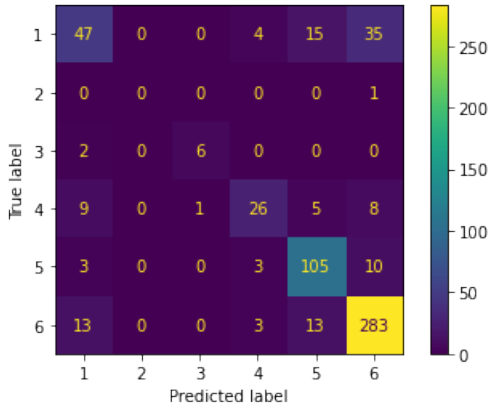

FIGURE 9 – Matrice de confusion sur l'ensemble de test

Avec stopwords

Ensuite, nous effectuons les mêmes tests, mais en ajoutant une liste de stopwords à l'extraction des mots.

La figure 10 montre la précision optimale du modèle pour différentes tailles de fenêtre. La taille optimale est 3. Encore une fois, la variation de la précision est assez faible, environ 4%.

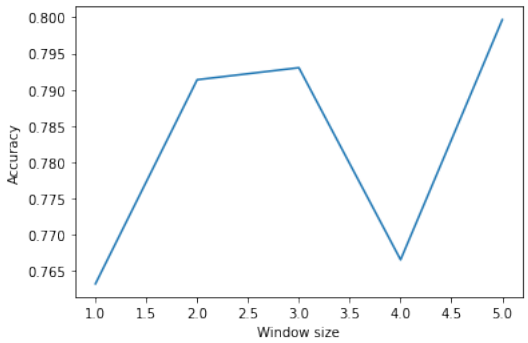

FIGURE 10 – Précision optimale en fonction de la taille de fenêtre

La figure 11 montre la précision optimale en fonction de la taille du réseaux de neurones. L'axe des x donne la largeur de chaque couche et chaque courbe représente une profondeur de réseau. La taille optimale du réseaux est d'une seule couche caché avec 7 neurones.

Les résultats sont très similaire à ceux obtenus sans l'utilisation des stopwords. Les réseaux les plus simples semblent être les plus performants.

La précision optimale sur l'ensemble de test est de 80,743%. Cette fois-ci, la précision est plus élevé sur l'ensemble de test que sur l'ensemble de validation. La figure 12 montre la matrice de confusion des prédictions sur l'ensemble de test. On remarque que le modèle prédit trop souvent la classe 6, probablement parce qu'elle est beaucoup plus représentée dans nos données.

Avec stemming

Nos derniers tests sont effectués en faisant du stemming sur les mots.

La figure 13 montre la précision optimale du modèle pour différentes tailles de fenêtre. La taille optimale est 4. La variation de la précision est encore plus faible que pour les deux autres cas, environ 2%.

La figure 14 montre la précision optimale en fonction de la taille du réseaux de neurones. L'axe des x donne la largeur de chaque couche et chaque courbe représente une profondeur de réseau. La taille optimale du réseaux est d'une seule couche caché avec 4 neurones.

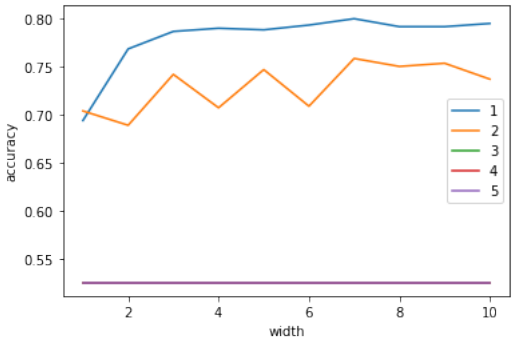

FIGURE 11 – Précision optimal en fonction de la taille du réseau

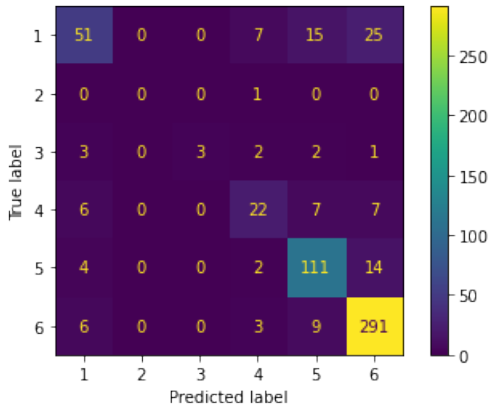

FIGURE 12 – Matrice de confusion sur l'ensemble de test

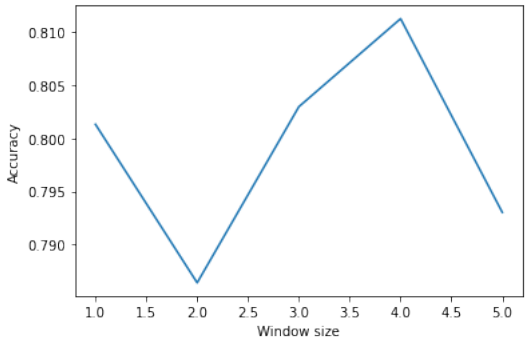

FIGURE 13 – Précision optimale en fonction de la taille de fenêtre

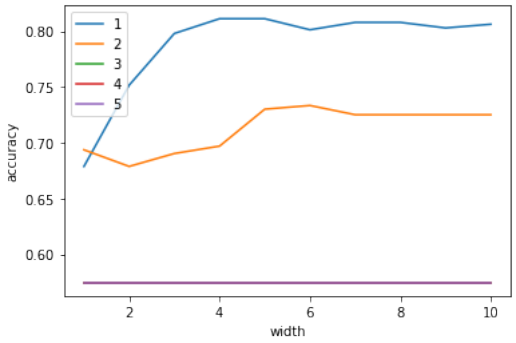

FIGURE 14 – Précision optimale en fonction de la taille du réseau

Les résultats sont toujours très similaire aux deux autres cas.

La précision optimale sur l'ensemble de test est de 79,561%. La figure 15 montre la matrice de confusion des prédictions sur l'ensemble de test. On remarque que le modèle prédit trop souvent la classe 6, probablement parce qu'elle est beaucoup plus représentée dans nos données.

Conclusion

Les résultats des tests avec et sans stemming et stopwords sont tous très similaires. Les précisions sur l'ensemble de test des différents tests tournent tous autour de 79%, donc on peut les considérer équivalente. Tous les meilleurs modèles n'ont aussi qu'une seule couche avec peu de neurones. En ce qui concerne la taille

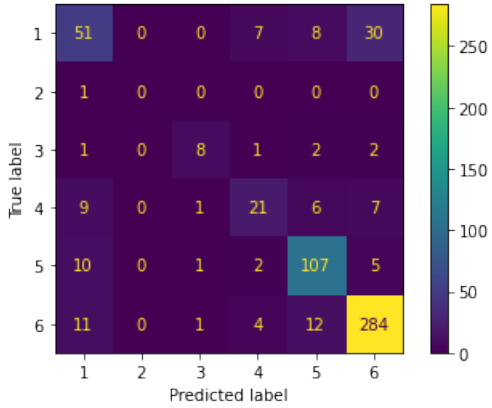

FIGURE 15 – Matrice de confusion sur l'ensemble de test

optimale de fenêtre, elle ne semble pas avoir tant d'influence sur les performances des modèles.

Il y a cependant certains paramètres que nous n'avons pas testé. Par exemple, il serait intéressant de voir si le word embedding pour la représentation des mots a une influence. On pourrait aussi regarder la différence avec les performances d'un modèle n'utilisant pas les catégories des mots comme attribut.