Rapport TP 2 - IFT3335-A-A22

Loïc Daudé Mondet
20243814 – Programmes d'échanges - 1er c.(Échange)

Francisco Pascoa
20160424

23/12/2022

Table des matières

I – Introduction

Le but de ce TP est de déterminer le sens du mot *interest* dans sont contexte. Dans notre corpus 2300 phrases, six sens ont été identifiés. Les classificateurs recevront en entrée un sac des mots entourant immédiatement *interest* dans une fenêtre donnée et la catégorie grammaticale des mots dans leur ordre d'apparition. L'apprentissage automatique sera accompli par la bibliothèque scikit-learn. Cinq types de classificateur seront étudiés : un classificateur bayésien naïf multinominal, un arbre de décision, une forêt aléatoire, une SVM et un perceptron multicouche.

II – Prétraitement des données

Nous avons pré-traité les données en deux temps. D'abord, nous avons nettoyé le corpus initial et nous avons séparé les mots, les catégories et l'étiquette de chacune des phrases dans des listes différentes. Ensuite, nous avons converti ces données vers une représentation numérique adéquate pour les méthodes d'apprentissage automatique.

II.1 Nettoyage et séparation

Les données fournies contenaient énormément d'informations que nous avons décidé d'ignorer par contraintes de temps et de complexité. Notamment, les groupes du nom des phrases étaient délimités par «[» et «]». Ces annotations ont toutes été retirées à cette étape. De plus, certaines phrases commençaient par plusieurs «=». Ne sachant pas leur signification, nous avons simplement décidé de les ignorer aussi. Nous n'avons pas appliqué de techniques plus avancées comme la lemmatisation ou l'utilisation de «stoplist».

Listing II.1 – Échantillion du corpus avant le nettoyage

```
[ yields/NNS ] on/IN [ money-market/JJ mutual/JJ funds/NNS ] continued/VBD [...]
$$
[ longer/JJR maturities/NNS ] are/VBP thought/VBN to/TO indicate/VB [...]
$$
nevertheless/RB ,/, said/VBD [ brenda/NP malizia/NP negus/NP ] ,/, [...]
$$
[ j.p./NP bolduc/NP ] ,/, [ vice/NN chairman/NN ] of/IN [ w.r./NP grace/NP ] [...]
$$
==================================== [ finmeccanica/NP ] is/VBZ [...]
$$
```

Listing II.2 – Échantillion du corpus après le nettoyage

```
yields/NNS on/IN money-market/JJ mutual/JJ funds/NNS continued/VBD to/TO [...]
$$
longer/JJR maturities/NNS are/VBP thought/VBN to/TO indicate/VB declining/VBG [...]
$$
nevertheless/RB ,/, said/VBD brenda/NP malizia/NP negus/NP ,/, editor/NN [...]
$$
j.p./NP bolduc/NP ,/, vice/NN chairman/NN of/IN w.r./NP grace/NP &/CC co./NP [...]
$$
```

Le nettoyage complété, nous avons découpé le corpus en ses différentes phrases et nous avons scindé chacune des phrases en une liste de ces binômes «mot/catégorie». À partir de cette liste de binômes, nous avons produit les trois listes ci-dessous.

Listing II.3 – Les trois listes

```
word_list_phrases = [
    ["yields", "on", "money-market", "mutual", "funds", "continued",  ...],
    ...
]
cat_phrases = [
    ["NNS", "IN", "JJ", "JJ", "NNS", "VBD",  ...],
    ...
]
labels = [6, ...]
```

II.2 Représentation numérique

Maintenant, que les mots, les catégories et les labels sont séparés, nous pouvons facilement les convertir en une représentation numérique. La représentation numérique du sac de mots est donnée par le `CountVectorizer` de scikit-learn. Les vecteurs générés par `CountVectorizer` représentent des sacs de mots. L'ordre des mots n'est pas conservé, seul leur présence l'est. La représentation numérique des catégories est donnée par un dictionnaire les contenant toutes et leur assignant à chacune un nombre. Par exemple, `cat_ids = {'NNS': 1, 'IN': 2, ...}`. Les labels sont déjà numériques.

III – Expériences

Il y a deux types de facteurs que nous voulions caractériser dans nos expériences. Tout d'abord, nous voulions observer l'effet de la taille de la fenêtre de contexte sur la performance des classificateurs. Pour ce faire, nous avons testé chacun des classificateurs avec une fenêtre de 1, 2 et 5 mots de chaque côté de *interest*. Ensuite, nous voulions observer l'impact de différents paramètres de chaque classificateur sur leur performance respective pour une fenêtre de 2 mots. Ces paramètres sont :

- pour l'arbre – la profondeur maximale (5, 20, 40) ;
- pour la forêt aléatoire – le nombre d'arbres dans la forêt (10, 100, 1000) ;
- pour la SVM – le noyau utilisé (linéaire, polynomiale, rbf, sigmoid) ;
- pour le perceptron – le nombre de couches cachées () ;

IV – Résultats

IV.1 Fenêtre de contexte variable

Les résultats suivants sont ceux des tests avec des largeurs de fenêtre variables. Le temps présenté est le temps combiné de l'entraînement et de l'évaluation de la performance du classificateur. Puisque l'arbre de décision, la forêt aléatoire et le perceptron sont sensibles au hasard dans leur apprentissage, les résultats présentés sont des moyenne±écart-type de cinq cycles apprentissage-évaluation. Voici les paramètres des différents classificateurs :

- Bayes naïf – par défaut ;
- Arbre de décision – par défaut ;
- Forêt aléatoire – par défaut ;
- SVM – `kernel=linear`, `decision_function_shape=ovo` ;
- Perceptron – `learning_rate=adaptive`, `learning_rate_init=0.025`, `hidden_layer_sizes=(175,)`

TABLE IV.1 – Score et temps des classificateurs pour différentes largeurs de fenêtre 1, 2, 5

Classificateurs	fenêtre = 1		fenêtre = 2		fenêtre = 5	
	Score	Temps (s)	Score	Temps (s)	Score	Temps (s)
Bayes naïf	0.535	0.212	0.571	0.174	0.596	0.211
Arbre de décision	0.846±0.003	0.168±0.002	0.817±0.003	0.202±0.002	0.761±0.004	0.280±0.001
Forêt aléatoire	0.866±0.003	3.69±0.07	0.868±0.007	2.94±0.05	0.803±0.006	2.46±0.05
SVM	0.863	26.056	0.888	30.297	0.863	44.071
Perceptron	0.872±0.009	216±75	0.871±0.012	178±27	0.855±0.006	234±30

IV.2 Paramètres des classificateurs variables

Les résultats suivants sont ceux des tests pour lesquels nous avons varié les paramètres des classificateurs pour une largeur de fenêtre constante à 2. Puisque l'arbre de décision, la forêt aléatoire et le perceptron sont sensibles au hasard dans leur apprentissage, les résultats présentés sont des moyenne±écart-type de cinq cycles apprentissage-évaluation.

TABLE IV.2 – Score et temps pour l'arbre de décision avec `max_depth` variable

`max_depth`	Score	Temps (s)
5	0.681±0.000	0.166±0.002
20	0.810±0.002	0.195±0.002
40 (défaut)	0.820±0.007	0.203±0.002

TABLE IV.3 – Score et temps pour la forêt aléatoire avec `n_estimators` variable

`n_estimators`	Score	Temps (s)
10	0.844±0.008	0.32±0.02
100 (défaut)	0.862±0.006	2.945±0.007
1000	0.871±0.003	28.61±0.16

TABLE IV.4 – Score et temps pour la SVM avec `kernel` variable et `decision_function_shape=ovo`

`kernel`	Score	Temps (s)
`linear`	0.888	30.217
`poly`	0.537	37.089
`rbf`	0.556	40.729
`sigmoid`	0.421	35.789

TABLE IV.5 – Score et temps pour le perceptron avec `hidden_layer_sizes` variable et `learning_rate=adaptive`, `learning_rate_init=0.025`

`hidden_layer_sizes`	Score	Temps (s)
`(125,)`	0.879±0.009	121±26
`(175,)`	0.873±0.014	193±16
`(225,)`	0.869±0.009	227±36

V – Comparaison des différentes méthodes et analyse

V.1 Analyse de l'impact de la largeur de la fenêtre et comparaison des classificateurs

Les classificateurs présentés dans le tableau IV.1 ont leurs paramètres optimaux. Leur performance est donc la meilleure possible dans ces tests et seule la fenêtre de contexte impacte leur score. Nous observons que dans presque tous les cas, une fenêtre de contexte plus large mène à une diminution marquée des scores sauf dans le cas du classificateur Bayes naïf. Dans presque tous les cas, il pire de regarder 5 mots de chaque côté que de n'en regarder qu'un seul. On remarque que l'arbre de décision seul semble préférer une fenêtre de 1. Notre hypothèse, pour expliquer ce comportement repose sur l'utilisation d'un sac de mots. Avec des fenêtres plus larges, il est possible que les sacs de mots se ressemblent davantage entre eux, rendant la tâche de trouver des différences utiles pour les discriminer plus difficile pour les classificateurs.

Ici, nous déterminons que le classificateur SVM est sur nos données dans la désambiguïsation de sens suivi de près par le perceptron. Notons cependant que le perceptron demande beaucoup plus de ressources calculatoires à entraîner.

V.2 Analyse de l'arbre de décision

Par défaut, sur notre corpus de données, l'arbre calculé avait une profondeur de 40. Nous avons voulu observer ce qu'il se passerait si nous limitions sa profondeur maximum à des valeurs inférieures. Un balayage quasi-logarithmique nous montre que le bénéfice d'avoir un arbre profond diminue plutôt rapidement puisque la différence de score entre 40 et 20 est bien moins grande qu'entre 20 et 5. Probablement qu'au-delà de 40 niveaux, nous aurions observé du surentraînement.

V.3 Analyse de la forêt aléatoire

Pour la forêt aléatoire, nous avons appliqué la même méthodologie que pour l'arbre de décision, mais pour le nombre d'arbres qu'elle contient seulement. Notre hypothèse était qu'au-delà d'un certain nombre, les gains seraient faible par rapport au temps requis pour générer la forêt. Nos tests ne nous contredisent pas. Nous remarquons que le temps d'entraînement augmente linéairement avec le nombre d'arbres dans la forêt, mais que le score semble approcher d'un plafond.

V.4 Analyse du classificateur SVM

Pour le classificateur SVM, nous étions curieux de voir quel rôle joue le noyau dans sa performance. Il s'avère que le noyau change tout, une fois le bon trouvé. Dans notre cas, le noyau linéaire est de loin supérieur à tous les autres.

V.5 Analyse du perceptron

Nos tests du perceptron montrent qu'il est plutôt efficace pour la tâche de désambiguïsation de sens. Il aurait été bien de testé des tailles de couche cachée encore plus petites puisque ses performances sont presqu'identiques entre 175 et 225, mais le temps d'entraînement est beaucoup plus long. Il aurait aussi été intéressant de voir s'il nous était possible de forcer le surentraînement.

VI – Conclusion

D'après nos résultats, sans prétraitement de données particulier, la fenêtre de contexte optimale est de deux mots de chaque côté de *interest* et le classificateur offrant la meilleure performance dans le délai le plus court est le classificateur SVM avec un noyau linéaire. cf. Bilan 1 Bilan 2.

Dans ce TP, les mots de contextes que nous donnions aux classificateurs étaient sous la forme de sacs de mots. Ceci veut dire que nous perdions l'ordre des mots qui pouvait nous informer sur la sémantique. Notre hypothèse est que c'est cette perte de sémantique qui menait à une diminution des performances avec une fenêtres plus large. Ainsi, nous proposons de donner aux classificateurs l'ordres des mots leur permettant potentiellement d'apprendre davantage avec des fenêtres plus grandes.

Annexes

A Utilisation des scripts

Notre TP est réalisé en huit fichiers.

- `main.py` – le script principal qui lance tous les tests ;
- `forest.py`, `naive.py`, `perceptron.py`, `svm.py`, `tree.py` – des «wrappers» pour les différents classificateurs ;
- `tests.py` – les fonctions de test des classificateurs ;
- `betterer_extract.py` – troisième itération du script de prétraitement des données.

Pour lancer les tests, il suffit d'exécuter `main.py`.

B Exemple d'exécution du programme

FIGURE 1 – Commande pour exécuter le programme. [1]

```
[01:02:32] loic@archlinux-loic-pcp /mnt/cloud/Nextcloud/Université/L3/UdeM
/IFT3335_IA/TPS/TP2/src
> time python main.py
```

1. `time` est une commande du shell qui permet d'afficher le temps total d'exécution lorsque que le programme termine, lancer `python main.py` suffit pour exécuter le programme.

FIGURE 2 – Tests avec une taille de fenêtre = 1

```
——— Tests avec jeu de paramètres 1 ———


——— Tests avec taille fenêtre = 1 ———

—— Tests Naive - paramètres : {} ——
Score naive: 0.535 - temps : 0.212s

—— Tests Tree - paramètres : {} ——
Score tree 1/5: 0.848 - temps : 0.172s
Score tree 2/5: 0.844 - temps : 0.168s
Score tree 3/5: 0.848 - temps : 0.169s
Score tree 4/5: 0.848 - temps : 0.165s
Score tree 5/5: 0.841 - temps : 0.166s
Moyenne des scores : 0.846 - Ecart type : 0.003
Temps moyen : 0.168s - Ecart Type : 0.002
Temps total tree: 0.840s

—— Tests Forest - paramètres : {} ——
Score forest 1/5: 0.865 - temps : 3.681s
Score forest 2/5: 0.865 - temps : 3.699s
Score forest 3/5: 0.865 - temps : 3.806s
Score forest 4/5: 0.863 - temps : 3.623s
Score forest 5/5: 0.871 - temps : 3.618s
Moyenne des scores : 0.866 - Ecart type : 0.003
Temps moyen : 3.686s - Ecart Type : 0.068
Temps total forest: 18.428s

—— Tests Svm - paramètres : {'kernel': 'linear', 'decision_function_shape':
'ovo'} ——
Score svm: 0.863 - temps : 26.056s

—— Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (175,)} ——
Score perceptron 1/5: 0.884 - temps : 181.558s
Score perceptron 2/5: 0.869 - temps : 279.399s
Score perceptron 3/5: 0.858 - temps : 135.445s
Score perceptron 4/5: 0.879 - temps : 328.916s
Score perceptron 5/5: 0.869 - temps : 154.621s
Moyenne des scores : 0.872 - Ecart type : 0.009
Temps moyen : 215.988s - Ecart Type : 75.116
Temps total perceptron: 1079.939s

Meilleur score :0.872 avec perceptron et paramètres : {'learning_rate':
'adaptive', 'learning_rate_init': 0.025, 'hidden_layer_sizes': (175,)}
Temps total d'exécution fenêtre = 1 : 1125.562s
```

FIGURE 3 – Tests avec une taille de fenêtre = 2

```
——— Tests avec taille fenêtre = 2 ———
—— Tests Naive - paramètres : {} ——
Score naive: 0.571 - temps : 0.174s

—— Tests Tree - paramètres : {} ——
Score tree 1/5: 0.820 - temps : 0.202s
Score tree 2/5: 0.820 - temps : 0.202s
Score tree 3/5: 0.816 - temps : 0.206s
Score tree 4/5: 0.812 - temps : 0.199s
Score tree 5/5: 0.818 - temps : 0.202s
Moyenne des scores : 0.817 - Ecart type : 0.003
Temps moyen : 0.202s - Ecart Type : 0.002
Temps total tree: 1.012s

—— Tests Forest - paramètres : {} ——
Score forest 1/5: 0.869 - temps : 2.987s
Score forest 2/5: 0.877 - temps : 2.965s
Score forest 3/5: 0.858 - temps : 2.951s
Score forest 4/5: 0.873 - temps : 2.963s
Score forest 5/5: 0.865 - temps : 2.851s
Moyenne des scores : 0.868 - Ecart type : 0.007
Temps moyen : 2.943s - Ecart Type : 0.048
Temps total forest: 14.718s

—— Tests Svm - paramètres : {'kernel': 'linear', 'decision_function_shape':
'ovo'} ——
Score svm: 0.888 - temps : 30.297s

—— Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (175,)} ——
Score perceptron 1/5: 0.863 - temps : 144.800s
Score perceptron 2/5: 0.888 - temps : 190.462s
Score perceptron 3/5: 0.865 - temps : 154.186s
Score perceptron 4/5: 0.858 - temps : 180.775s
Score perceptron 5/5: 0.882 - temps : 221.552s
Moyenne des scores : 0.871 - Ecart type : 0.012
Temps moyen : 178.355s - Ecart Type : 27.307
Temps total perceptron: 891.774s

Meilleur score :0.888 avec svm et paramètres : {'kernel': 'linear',
'decision_function_shape': 'ovo'}
Temps total d'exécution fenêtre = 2 : 938.064s
```

FIGURE 4 – Tests avec une taille de fenêtre = 5

```
——— Tests avec taille fenêtre = 5 ———
—— Tests Naive - paramètres : {} ——
Score naive: 0.596 - temps : 0.211s

—— Tests Tree - paramètres : {} ——
Score tree 1/5: 0.761 - temps : 0.280s
Score tree 2/5: 0.759 - temps : 0.282s
Score tree 3/5: 0.757 - temps : 0.278s
Score tree 4/5: 0.759 - temps : 0.280s
Score tree 5/5: 0.767 - temps : 0.280s
Moyenne des scores : 0.761 - Ecart type : 0.004
Temps moyen : 0.280s - Ecart Type : 0.001
Temps total tree: 1.401s

—— Tests Forest - paramètres : {} ——
Score forest 1/5: 0.793 - temps : 2.515s
Score forest 2/5: 0.810 - temps : 2.392s
Score forest 3/5: 0.801 - temps : 2.428s
Score forest 4/5: 0.810 - temps : 2.445s
Score forest 5/5: 0.803 - temps : 2.518s
Moyenne des scores : 0.803 - Ecart type : 0.006
Temps moyen : 2.460s - Ecart Type : 0.049
Temps total forest: 12.298s

—— Tests Svm - paramètres : {'kernel': 'linear', 'decision_function_shape':
'ovo'} ——
Score svm: 0.863 - temps : 44.071s

—— Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (175,)} ——
Score perceptron 1/5: 0.852 - temps : 277.066s
Score perceptron 2/5: 0.854 - temps : 240.527s
Score perceptron 3/5: 0.852 - temps : 184.525s
Score perceptron 4/5: 0.867 - temps : 245.780s
Score perceptron 5/5: 0.852 - temps : 219.744s
Moyenne des scores : 0.855 - Ecart type : 0.006
Temps moyen : 233.528s - Ecart Type : 30.613
Temps total perceptron: 1167.643s

Meilleur score :0.863 avec svm et paramètres : {'kernel': 'linear',
'decision_function_shape': 'ovo'}
Temps total d'exécution fenêtre = 5 : 1225.723s
```

FIGURE 5 – Bilan des tests sur les tailles de fenêtre

```
——— Bilan test des tailles de fenêtres ———
Meilleur score :0.888 avec svm et paramètres {'kernel': 'linear',
'decision_function_shape': 'ovo'} pour la fenêtre de taille 2
Temps total d'exécution : 3289.349s
——— Tests avec différents jeux de paramètres ———
```

FIGURE 6 – Tests en faisant varier les paramètres - Naive

```
——— Tests avec taille fenêtre = 2 ———
—— Tests Naive - paramètres : {} ——
Score naive: 0.571 - temps : 0.176s
```

FIGURE 7 – Tests en faisant varier les paramètres - Tree

```
── Tests Tree - paramètres : {'max_depth': 40} ──
Score tree 1/5: 0.831 - temps : 0.203s
Score tree 2/5: 0.810 - temps : 0.202s
Score tree 3/5: 0.816 - temps : 0.203s
Score tree 4/5: 0.818 - temps : 0.206s
Score tree 5/5: 0.825 - temps : 0.202s
Moyenne des scores : 0.820 - Ecart type : 0.007
Temps moyen : 0.203s - Ecart Type : 0.001
Temps total tree: 1.016s

── Tests Tree - paramètres : {'max_depth': 20} ──
Score tree 1/5: 0.812 - temps : 0.195s
Score tree 2/5: 0.808 - temps : 0.195s
Score tree 3/5: 0.810 - temps : 0.195s
Score tree 4/5: 0.808 - temps : 0.198s
Score tree 5/5: 0.812 - temps : 0.193s
Moyenne des scores : 0.810 - Ecart type : 0.002
Temps moyen : 0.195s - Ecart Type : 0.002
Temps total tree: 0.976s

── Tests Tree - paramètres : {'max_depth': 5} ──
Score tree 1/5: 0.681 - temps : 0.163s
Score tree 2/5: 0.681 - temps : 0.165s
Score tree 3/5: 0.681 - temps : 0.165s
Score tree 4/5: 0.681 - temps : 0.165s
Score tree 5/5: 0.681 - temps : 0.170s
Moyenne des scores : 0.681 - Ecart type : 0.000
Temps moyen : 0.166s - Ecart Type : 0.002
Temps total tree: 0.828s
```

FIGURE 8 – Tests en faisant varier les paramètres - Forest

```
── Tests Forest - paramètres : {'n_estimators': 10} ──
Score forest 1/5: 0.848 - temps : 0.304s
Score forest 2/5: 0.844 - temps : 0.301s
Score forest 3/5: 0.850 - temps : 0.354s
Score forest 4/5: 0.848 - temps : 0.325s
Score forest 5/5: 0.829 - temps : 0.307s
Moyenne des scores : 0.844 - Ecart type : 0.008
Temps moyen : 0.318s - Ecart Type : 0.020
Temps total forest: 1.591s

── Tests Forest - paramètres : {'n_estimators': 100} ──
Score forest 1/5: 0.869 - temps : 2.949s
Score forest 2/5: 0.860 - temps : 2.946s
Score forest 3/5: 0.869 - temps : 2.938s
Score forest 4/5: 0.858 - temps : 2.937s
Score forest 5/5: 0.854 - temps : 2.956s
Moyenne des scores : 0.862 - Ecart type : 0.006
Temps moyen : 2.945s - Ecart Type : 0.007
Temps total forest: 14.725s

── Tests Forest - paramètres : {'n_estimators': 1000} ──
Score forest 1/5: 0.869 - temps : 28.559s
Score forest 2/5: 0.873 - temps : 28.539s
Score forest 3/5: 0.873 - temps : 28.645s
Score forest 4/5: 0.875 - temps : 28.424s
Score forest 5/5: 0.867 - temps : 28.893s
Moyenne des scores : 0.871 - Ecart type : 0.003
Temps moyen : 28.612s - Ecart Type : 0.157
Temps total forest: 143.060s
```

FIGURE 9 – Tests en faisant varier les paramètres - Svm

```
── Tests Svm - paramètres : {'kernel': 'linear', 'decision_function_shape':
'ovo'} ──
Score svm: 0.888 - temps : 30.217s

── Tests Svm - paramètres : {'kernel': 'poly', 'decision_function_shape':
'ovo'} ──
Score svm: 0.537 - temps : 37.089s

── Tests Svm - paramètres : {'kernel': 'rbf', 'decision_function_shape':
'ovo'} ──
Score svm: 0.556 - temps : 40.729s

── Tests Svm - paramètres : {'kernel': 'sigmoid', 'decision_function_shape':
'ovo'} ──
Score svm: 0.421 - temps : 35.789s
```

FIGURE 10 – Tests en faisant varier les paramètres - Perceptron

```
── Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.1, 'hidden_layer_sizes': (125,)} ──
Score perceptron 1/5: 0.860 - temps : 144.579s
Score perceptron 2/5: 0.869 - temps : 75.616s
Score perceptron 3/5: 0.846 - temps : 172.296s
Score perceptron 4/5: 0.869 - temps : 141.137s
Score perceptron 5/5: 0.744 - temps : 321.624s
Moyenne des scores : 0.838 - Ecart type : 0.047
Temps moyen : 171.050s - Ecart Type : 81.703
Temps total perceptron: 855.252s

── Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (125,)} ──
Score perceptron 1/5: 0.888 - temps : 113.276s
Score perceptron 2/5: 0.871 - temps : 127.989s
Score perceptron 3/5: 0.882 - temps : 137.082s
Score perceptron 4/5: 0.888 - temps : 77.982s
Score perceptron 5/5: 0.865 - temps : 153.324s
Moyenne des scores : 0.879 - Ecart type : 0.009
Temps moyen : 121.931s - Ecart Type : 25.527
Temps total perceptron: 609.653s

── Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (175,)} ──
Score perceptron 1/5: 0.899 - temps : 215.931s
Score perceptron 2/5: 0.865 - temps : 208.876s
Score perceptron 3/5: 0.873 - temps : 172.649s
Score perceptron 4/5: 0.858 - temps : 191.391s
Score perceptron 5/5: 0.871 - temps : 179.903s
Moyenne des scores : 0.873 - Ecart type : 0.014
Temps moyen : 193.750s - Ecart Type : 16.513
Temps total perceptron: 968.751s

── Tests Perceptron - paramètres : {'learning_rate': 'adaptive',
'learning_rate_init': 0.025, 'hidden_layer_sizes': (225,)} ──
Score perceptron 1/5: 0.869 - temps : 226.231s
Score perceptron 2/5: 0.875 - temps : 270.807s
Score perceptron 3/5: 0.856 - temps : 262.661s
Score perceptron 4/5: 0.882 - temps : 199.377s
Score perceptron 5/5: 0.865 - temps : 177.853s
Moyenne des scores : 0.869 - Ecart type : 0.009
Temps moyen : 227.386s - Ecart Type : 35.691
Temps total perceptron: 1136.929s
```

FIGURE 11 – Bilan des tests sur les variations de paramètres

```
Meilleur score :0.888 avec svm et paramètres {'kernel': 'linear',
'decision_function_shape': 'ovo'}
Temps total d'exécution fenêtre = 2 : 3876.876s
```

FIGURE 12 – Temps total d'exécution