Master 1 Cryptis/Isicg

Rapport de Projet IA

TP1/TP2

Réalisé par :
JAMOUSSI Houcein
TRABELSI Ayoub

Enseignant :
Monsieur TAMINE Karim

2022-2023

Table des matières

1 Travail demandé **1**

2 Introduction **1**

3 Partie 1 **2**

3.1 Algorithme KPPV 2

3.1.1 Principe 2

3.1.2 Réalisation 2

3.2 Algorithme de Perceptron 5

3.2.1 Principe 5

3.2.2 Réalisation 5

4 Partie 2 **8**

4.1 Réseaux de neurones 8

4.1.1 Modèle basé sur une architecture à base de réseau de neurones sans couches cachées . . . 8

4.1.2 Modèle basé sur une architecture à base de réseaux de neurones avec une couche cachée . 9

4.2 Classifieur Bayésien naif 10

4.2.1 Principe 10

4.2.2 Réalisation 10

4.3 Comparaison des modèles 11

5 Bibliographie **11**

Table des figures

1 L'intelligence artificielle 1
2 Fonction KPPV 2
3 Affichage des données classées et des données de test 3
4 Affichage des données de test classées 3
5 Affichage de la précision et de l'erreur 4
6 Affichage de la précision et de l'erreur (K=3) 4
7 Affichage de la précision et de l'erreur (K=2) 4
8 Perceptron 5
9 Fonction Perceptron 6
10 Fonction Apprentissage 6
11 L'erreur 6
12 Affichagede l'ensemble de test classifiés 7
13 Architecture du modèle M1 8
14 Précision et perte du modèle M1 en fonction des époques 8
15 Valeurs de la précision et de la perte du modèle M1 9
16 Architecture du modèle M2 9
17 Précision et perte du modèle M2 en fonction des époques 9
18 Valeurs de la précision et de la perte du modèle M2 10
19 Précission du modèle Bayésien naif 10
20 Performances des trois modèles 11

1 Travail demandé

Le travail demandé sera effectué selon les deux parties suivantes :
-Partie 1 : Implémentation des méthodes des **K Plus Proches Voisins** et du **Perceptron** en Python.
-Partie 2 : Construction d'un **Anti-Spam** basé sur :

- Un réseau de neurones sans couches cachées
- Un réseau de neurones avec une couche cachée
- Un classifieur de type Bayes naif

2 Introduction

L'intelligence artificielle (IA) est un procédé d'imitation de l'intelligence humaine qui s'appuie sur la création et l'application d'algorithmes exécutés dans un environnement de calcul dynamique. Son but est de doter les machines par la capacité de penser et de se comporter comme des êtres humains.[1]
L'intelligence artificielle englobe l'apprentissage automatique, qui lui-même englobe l'apprentissage profond, comme le montre la figure .
L'apprentissage automatique

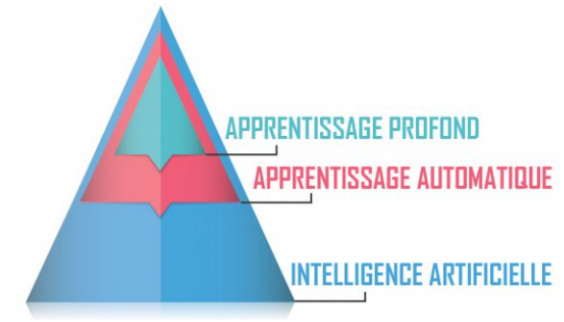

FIGURE 1 – L'intelligence artificielle

L'apprentissage automatique est une application de l'IA qui permet aux systèmes d'apprendre et de s'améliorer à partir de l'expérience sans être explicitement programmés.
L'apprentissage automatique se concentre sur le développement de programmes informatiques qui peuvent accéder à des données et les utiliser pour apprendre par eux-mêmes. Il peut être supervisé, non supervisé, ou par renforcement[2].
-L'apprentissage supervisé
L'apprentissage supervisé est fondé sur une base de données étiquetées, dans laquelle les échantillons sont définis par des entrées et des sorties.
Il tente de répondre à deux problématiques : la classification et la régression.
La classification présente des sorties discrètes qui correspondent à des classes auxquelles peut appartenir un échantillon à l'entrée.[3]
Quant au problème de régression, le modèle tente de prédire une fonction à valeurs réelles et continues.
-L'apprentissage non supervisé
les données ne fournissent pas de signal explicite et le modèle doit extraire de l'information uniquement à partir de la structure des entrées.
-L'apprentissage par renforcement
L'algorithme apprend quoi faire à partir d'expériences répétées dans lesquelles il doit atteindre son but.
Cela implique de laisser un algorithme apprendre de ses erreurs afin d'atteindre un objectif[3].

Tout au long de notre travail, on s'intéressera plutôt aux méthodes supervisées.

3 Partie 1

3.1 Algorithme KPPV

3.1.1 Principe

L'algorithme des k plus proches voisins, connu sous le nom de KPPV ou KNN, est un algorithme d'apprentissage supervisé, qui utilise la proximité pour effectuer des classifications ou des prédictions sur le regroupement d'un point de données individuel : il est nécessaire d'avoir des données labellisées et à partir de cet ensemble E de données labellisées, il sera possible de classer (déterminer le label) d'une nouvelle donnée (donnée n'appartenant pas à E).
Soit un ensemble E contenant n données labellisées et X une entrée :

- On calcule les distances entre la donnée X et chaque donnée appartenant à E pour trouver les K plus proches voisins de X.
- On attribue à X la classe qui est la plus fréquente parmi les K données les plus proches.

Le succès de cet algorithme va dépendre de deux facteurs :
-La quantité de données d'entrainement
-La qualité de la mesure de distance

3.1.2 Réalisation

On a choisi de travailler sur google collaboration puisque cette plateforme nous offre non seulement la simplicité et la clarté mais aussi la possibilité de contribuer au code sur plusieurs machines.
Vous trouverez dans **la première section** de notre code la partie concernant l'algorithme KPPV.
Voici le lien du code :
å https://colab.research.google.com/drive/1jMGyXtW6PdJ6iWq_i8ZgP6Bz72YSk7ze#scrollTo=wkLog5QspR7H
.

```
def kppv(x,appren,oracle,K):
    clas=[]
    v=len(x[0])
    for i in range(0,v):
        dis=[]
        clss=[]
        kppp=[]
        v2=len(appren[0])
        j=0
        while j<v2:
           dis.append(distance_euclidienne(appren[0][j],x[0][i],appren[1][j],x[1][i]) )
           j=j+1
        clss.append(dis)
        clss.append(oracle)
        clss=np.transpose(clss)
        clss = clss[clss[:,0].argsort()]
        clss=np.transpose(clss)
        kppp=clss[1][:K]
        ms=Counter(kppp).most_common()[0][0]
        clas.append(ms)
    res = np.array(clas)
    return res
```

FIGURE 2 – Fonction KPPV

Pour estimer la sortie associée à une nouvelle entrée x, on utilisera notre fonction kppv qui fonctionnera comme suit :

— Stocker la distance entre X et chaque autre point dans une liste appelée dis.

— Trier les points selon la distance dans une autre liste appelée clss.

— Choisir les K premiers points.

— Attribuer à notre donnée X la classe majoritaire (à l'aide de la fonction prédéfinie most-common).

Dans un premier temps, on affiche les deux classes prédéfinies accompagné de l'ensemble de test :

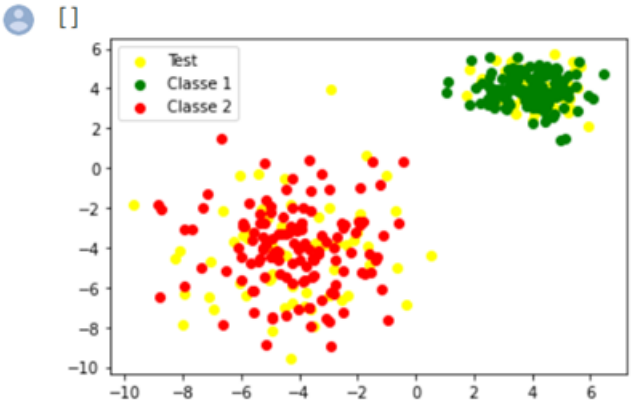

FIGURE 3 – Affichage des données classées et des données de test

Après l'application de l'algorithme KPPV sur notre ensemble de test, on obtient cette visualisation de la classification de données en deux classes :

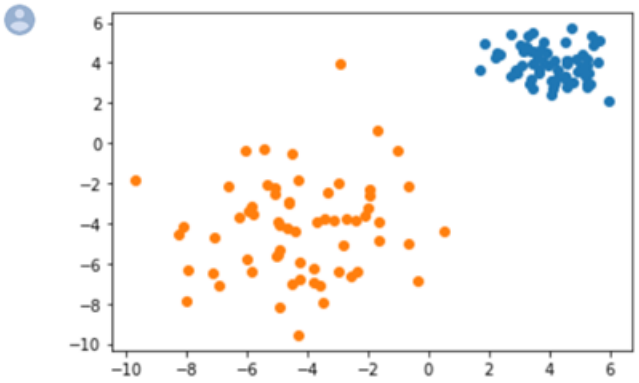

FIGURE 4 – Affichage des données de test classées

Puisque les données de test sont suffisamment éloignées, la classification est assez simple pour l'algorithme, ce qui donne le résultat suivant :

```
[201] test_erreur = np.concatenate((np.zeros(64),np.ones(64)))
      print('Précision',metrics.accuracy_score(test_erreur, clas))
      print('Erreur',1-metrics.accuracy_score(test_erreur, clas))

      Précision 1.0
      Erreur 0.0
```

FIGURE 5 – Affichage de la précision et de l'erreur

On a donc essayé de rapprocher les points de test pour introduire un peu d'ambiguïté sur notre modèle, ainsi on observe le changement des résultats suite à chaque changement de la variable K :

K=3

```
[216] test_erreur = np.concatenate((np.zeros(64),np.ones(64)))
      print('Précision',metrics.accuracy_score(test_erreur, clas))
      print('Erreur',1-metrics.accuracy_score(test_erreur, clas))

      Précision 0.9453125
      Erreur 0.0546875
```

FIGURE 6 – Affichage de la précision et de l'erreur (K=3)

K=2

```
test_erreur = np.concatenate((np.zeros(64),np.ones(64)))
print('Précision',metrics.accuracy_score(test_erreur, clas))
print('Erreur',1-metrics.accuracy_score(test_erreur, clas))

Précision 0.921875
Erreur 0.078125
```

FIGURE 7 – Affichage de la précision et de l'erreur (K=2)

3.2 Algorithme de Perceptron

3.2.1 Principe

Un Perceptron est un neurone artificiel, et donc une unité de réseau de neurones. Il effectue des calculs pour détecter des caractéristiques ou des tendances dans les données d'entrée.

Il s'agit d'un algorithme pour l'apprentissage supervisé de classificateurs binaires. C'est cet algorithme qui permet aux neurones artificiels d'apprendre et de traiter les éléments d'un ensemble de données.

Le Perceptron joue un rôle essentiel dans les projets de Machine Learning. Il est massivement utilisé pour classifier les données, ou en guise d'algorithme permettant de simplifier ou de superviser les capacités d'apprentissage de classificateurs binaires.[4]
Le principe de fonctionnement de perceptron c'est modéliser la décision à l'aide d'une fonction linéaire suivi d'une fonction d'activation.

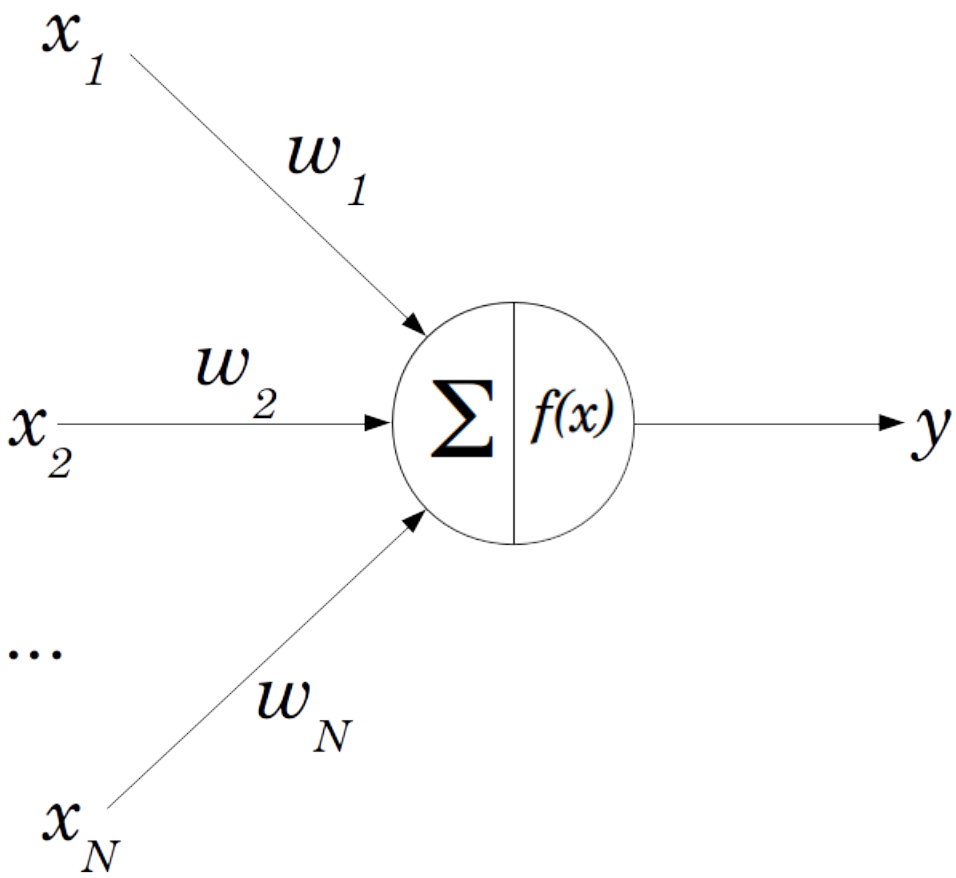

FIGURE 8 – Perceptron

Y=La fonction d'activation(Xi*Wi+ b).
Si les deux classes sont linéairement séparables, donc on peut utiliser le perceptron comme un classifcateur.

3.2.2 Réalisation

Vous trouverez dans **la deuxième section** de notre code la partie concernant l'algorithme Perceptron. Voici le lien du code :
`https://colab.research.google.com/drive/1jMGyXtW6PdJ6iWq_i8ZgP6Bz72YSk7ze#scrollTo=wkLog5QspR7H`

En premier lieu, on prépare la fonction perceptron qui appliquera à la lettre le principe de l'algorithme du perceptron en calculant la somme w*x+b et en appliquant la fonction d'activation indiquée en entrée sur cette somme.

```
[ ]  def perceptron(x,w,active):
       somme=(w[0]+(w[1]*x[0]+w[2]*x[1]))
       if(active==0):#fonction d'activation
         z=(np.sign(somme))
       else:          #z=sign(w1*x1+w2*x2+b)
         z=(np.tanh(somme))
       return z       #z=tanh(w1*x1+w2*x2+b)
```

FIGURE 9 – Fonction Perceptron

En second lieu, on prépare la fonction apprentissage qui se chargera de mettre à jour les valeurs des poids w et de calculer la valeur de l'erreur cumulée.

```
[ ]  def apprentissage(x,yd,active):
       erreur=[] #tableau contient l'erreur de chaque itération
       w=[0.5,1,1]   #initialisation de w1=1 w2=1 b=0.5
       mdiff=0       #initialiser a 0 mdiff (somme des erreurs de chaque exemples)
       for i in range (0,100): #100 itération
         for j in range (256): #Parcourir toutes les entrées
           y_a=perceptron([x[0][j],x[1][j]],w,active)
           if (y_a!=yd[j]):
             w[1]=w[1]+(0.1*(yd[j]-y_a)*x[0][j])  #Mise a jour des poids
             w[2]=w[2]+(0.1*(yd[j]-y_a)*x[1][j])
             w[0]=w[0]+(0.1*(yd[j]-y_a))          #Mise a jour de bias
             mdiff= mdiff + (yd[i]-y_a)**2
         print('Itération ',i)
         print('W1=',w[1])
         print('W2=',w[2])
         print('b=',w[0])
         erreur.append(mdiff)
       print(erreur)

       return w,erreur
```

FIGURE 10 – Fonction Apprentissage

Ci-contre se trouve la courbe de l'erreur cumulée :

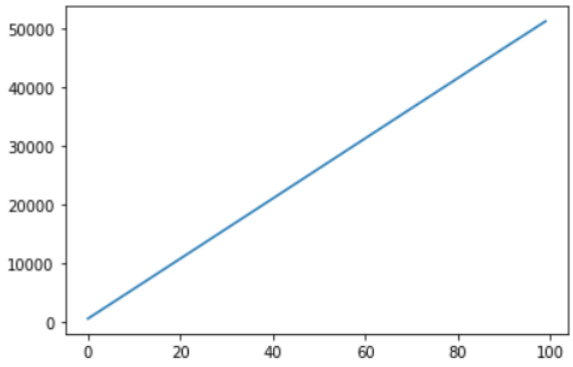

FIGURE 11 – L'erreur

Enfin, la fonction affiche-classe nous aidera à évaluer la séparation associée au neurone.

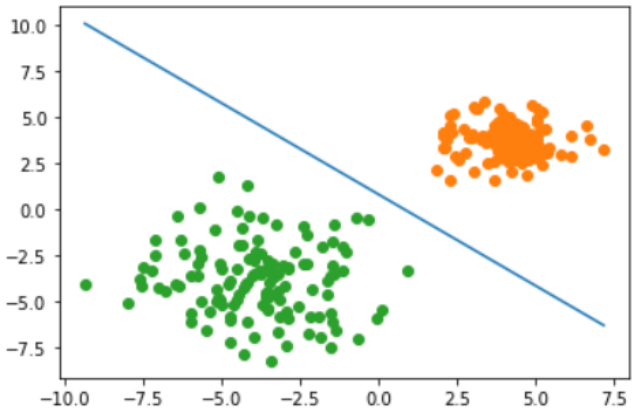

FIGURE 12 – Affichagede l'ensemble de test classifiés

Même si la méthode KPPV utilise plus de ressources, il s'avère qu'elle est plus précise que la méthode Perceptron dans notre exemple.

4 Partie 2

4.1 Réseaux de neurones

Les réseaux de neurones artificiels sont des modèles de calcul mathématique et informatique dont la conception est inspirée du fonctionnement des neurones biologiques capables de traiter les entrées reçues d'un ensemble de connexions caractérisées par des poids synaptiques.[5]

Il existe différents types de réseaux de neurones :

— **RN sans couches cachées.**

— **RN avec couches cachées.**

Vous trouverez dans ce lien colab la partie concernant la construction d'un Anti-Spam. Voici le lien du code : å
https://colab.research.google.com/drive/1SL9DZVdMowc9vP7YwMfBqjNCEmzQIx4D#scrollTo=h26jHVwUOkJB

4.1.1 Modèle basé sur une architecture à base de réseau de neurones sans couches cachées

Pour débuter, on a utiliser la fonction files.upload pour importer les fichiers contenant les données d'apprentissage et de test.
En second lieu, on a réparti les données d'entrainement sur deux listes : train-x contenant les entrées et train-y contenant les étiquettes (classe 0 ou 1).
De même, on a réparti les données de test sur deux listes : test-x contenant les entrées et test-y contenant les étiquettes (classe 0 ou 1).
On fait appel à la bibliothèque keras pour créer notre modèle.

```
M1= tf.keras.models.Sequential()
M1.add(tf.keras.layers.Dense(units=1, activation='sigmoid', input_dim=57, ))
M1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = M1.fit(train_x, train_y, epochs=150 )
```

FIGURE 13 – Architecture du modèle M1

Pour suivre l'évolution du modèle, on affichera la perte et la précision par rapport aux époques :

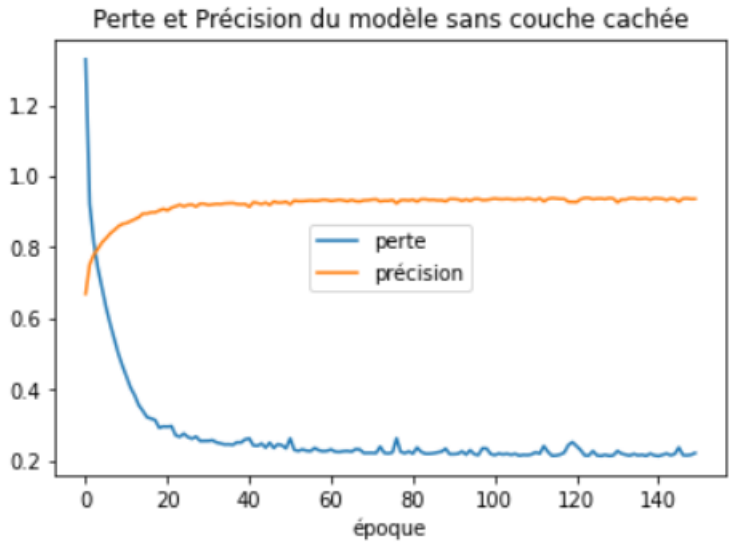

FIGURE 14 – Précision et perte du modèle M1 en fonction des époques

Finalement, on exécute notre modèle sur les données de test pour voir comment il se comporte. On obtient les valeurs suivantes de la précision et de la perte :

-Perte=0.1898

-Précision=0.9372

```
accuracy1 = M1.evaluate(test_x, test_y)
print('Perte',accuracy1[0])
print('Précision',accuracy1[1])
```

```
40/40 [==============================] - 0s 2ms/step - loss: 0.1898 - accuracy: 0.9372
Perte 0.1898142248392105
Précision 0.9372056722640991
```

FIGURE 15 – Valeurs de la précision et de la perte du modèle M1

4.1.2 Modèle basé sur une architecture à base de réseaux de neurones avec une couche cachée

Dans ce modèle, on est demandé d'ajouter une couche cachée. Pour ce faire, on utilisera la commande suivante : **M2.add(tf.keras.layers.Dense(1, activation='sigmoid'))** pour aboutir à la création de notre modèle comme suit :

```
M2 = tf.keras.models.Sequential()
M2.add(tf.keras.layers.Dense(57, input_dim=57, activation='sigmoid'))
M2.add(tf.keras.layers.Dense(1, activation='sigmoid'))
M2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = M2.fit(train_x, train_y, epochs= 150)
```

FIGURE 16 – Architecture du modèle M2

Pour suivre l'évolution du modèle, on affichera la perte et la précision par rapport aux époques :

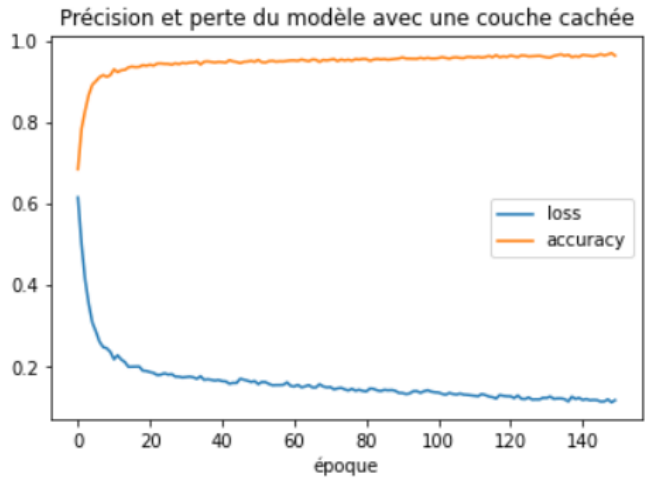

FIGURE 17 – Précision et perte du modèle M2 en fonction des époques

De même, on exécute notre modèle sur les données de test pour voir comment il se comporte. On obtient les valeurs suivantes de la précision et de la perte :
-Perte=0.1301
-Précision=0.9521

```
[11] accuracy2 = M2.evaluate(test_x, test_y)
     print('Perte',accuracy2[0])
     print('Précision',accuracy2[1])

     40/40 [==============================] - 0s 1ms/step - loss: 0.1301 - accuracy: 0.9521
     Perte 0.13014498353004456
     Précision 0.9521192908287048
```

FIGURE 18 – Valeurs de la précision et de la perte du modèle M2

4.2 Classifieur Bayésien naif

4.2.1 Principe

La classification naïve bayésienne s'apparente à une classification bayésienne probabiliste simple (dite naïve). Elle repose sur le théorème de Bayes, qui n'est autre qu'un modèle de probabilités.
L'algorithme a pour principe de prédire une classe étant donné un ensemble de caractéristiques, autrement dit la probabilité qu'un événement se produise étant donné la probabilité qu'un autre événement se soit déjà produit.[6]

4.2.2 Réalisation

On commence par séparer les données en Spam et NonSpam, pour ensuite calculer respectivement leurs probabilités et leurs espérances qu'on utilisera enfin dans la fonction prediction qui nous classifiera l'entrée X en Spam ou NonSpam. Toutes ces étapes sont bien décrites dans le code.

Pour tester la précision, on calculera la fréquence des prédictions correctes par rapport au total des prédictions comme suit : **-Précision=0.7166**

```
vrai = 0
all = 0
for i in range(test_x.shape[0]):
  prediction_result = prediction(test_x[i])
  if (prediction_result == test_y[i]):
    vrai+=1
  all +=1
pres =  vrai / all
print("vrai = ",vrai)
print("total = ",all)
print("précision = ",pres)
accuracy3 = [(1-pres), pres]
```

```
vrai =  913
total =  1274
précision =  0.716640502354788
```

FIGURE 19 – Précission du modèle Bayésien naif

4.3 Comparaison des modèles

On affiche la précision des trois modèles :

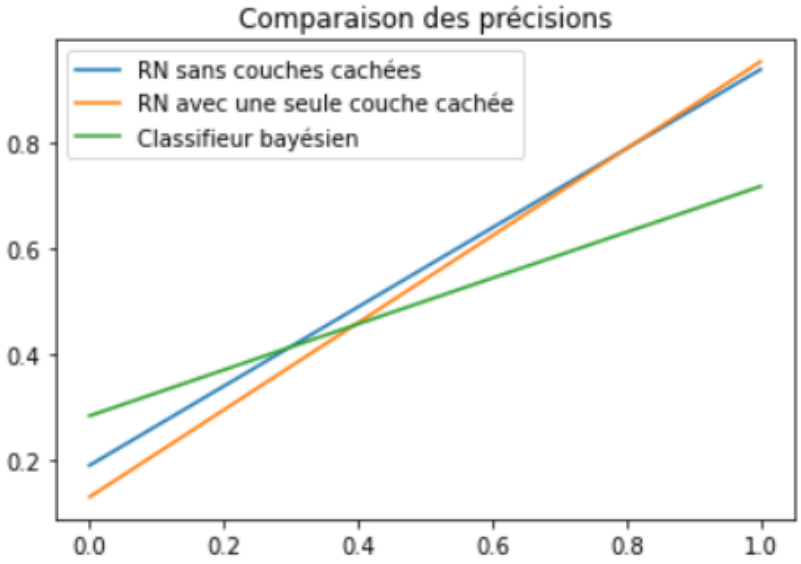

FIGURE 20 – Performances des trois modèles

Le modèle RN avec une seule couche cachée est le plus précis dans notre exemple.

5 Bibliographie

Ci-dessous une collection de références vers les différentes ressources nous ayant servi lors de la réalisation de ce projet

Références

[1] NETAPP. QU'EST-CE QUE L'INTELLIGENCE ARTIFICIELLE ?
https://www.netapp.com/fr/artificial-intelligence/what-is-artificial-intelligence/

[2] WHAT IS MACHINE LEARNING ? A DEFINITION. https://www.expert.ai/blog/machine-learning-definition/?

[3] L. BASTIEN. MACHINE LEARNING : DÉFINITION, FONCTIONNEMENT, UTILISATIONS.
https://datascientest.com/machine-learning-tout-savoir

[4] *Perceptron : qu'est-ce que c'est et à quoi ça sert ?*
https://datascientest.com/perceptron

[5] *Artificial neuron*
https://www.techtarget.com/searchcio/definition/artificial-neuron

[6] *Classification naïve bayésienne : définition et principaux avantages*
https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1501321-classification-naive-bayesienne-definition/