UQAC

Cours 8INF912 - Sujet spécial en informatique II

Kevin Bouchard

Rapport de Sprint 1

Benoît Manhes	Lilian Pattier	Pierre Scalzo
Manb22079601	Patl21039604	Scap15089506

11 juin 2019

Table des matières

Introduction **1**

1 Travail réalisé **1**

1.1 Conception du dataset 1

1.2 Implémentation du RNN 5

1.3 Elaboration d'un classificateur 11

2 Perspectives **12**

2.1 Construction avec thème imposé 12

2.2 Changement de classificateur 12

2.3 Élaboration d'un site web 13

Bilan **14**

Table des figures

1 Liste des sources 1

2 Exemple de site web de haikus 2

3 Code source du site précédent 3

4 Structure du site 3

5 Expression régulière 4

6 Récupération des datas 5

7 Stockage dans une BDD 5

8 Implémentation de l'import du fichier texte contenant le dataset 6

9 Elaboration du vocabulaire et suppression des doublons 7

10 Traduction du texte avec char2idx 7

11 Implémentation de l'entrainement 8

12 Visualisation de l'entrainement 8

13 Construction du modèle 9

14 Génération de texte 9

15 Paramètres utilisés 10

16 Exemple 1 d'haiku généré 10

17 Exemple 2 d'haiku généré 10

18 Résultats de classification 12

19 Projet de site web 13

Introduction

Un haiku est un petit poème avec une structure spécifique : il est composé de 3 lignes avec respectivement 5-7-5 syllabes. L'objectif est de créer une intelligence artificielle capable d'en produire par elle-même. Au cours de ce premier sprint nous avons conçu un dataset de haikus existants. Grâce à celui-ci nous avons entrainé un réseau de neuronnes récurrent fournissant après traitement un haiku. Un classificateur permettra par la suite d'accepter ou de réfuter le texte créé comme étant un haiku.

1 Travail réalisé

1.1 Conception du dataset

La première phase de notre travail a été de se constituer un ensemble de données en centralisant un maximum de haiku. Il n'existe pas de base de données en libre accès contenant des haikus en français. Nous avons donc dû extraire des données sur différents sites web de façon à nous constituer notre propre dataset. Dans un premier temps, nous avons dressé une liste de sites web sur lesquels nous pourrions extraire du contenu :

http://www.unhaiku.com
http://www.tempslibres.org
https://short-edition.com/fr/categorie/poetik/haiku-et-tanka
https://www.poetica.fr/categories/haikus/
https://www.eternels-eclairs.fr/haikus-basho-buson-issa-shiki-santoka.php
http://lieucommun.canalblog.com/archives/haikus___poesies_des_saisons/index.html
https://lundi.am/Esprit-zen-bol-vide
https://www.rosedesventseditions.com/haikus/haikus-de-printemps/
https://www.babelio.com/auteur/Basho-Matsuo/69220/citations

FIGURE 1 – Liste des sources

Le principal défaut de ces sites est qu'ils ne permettent pas de télécharger directement le contenu, qui est souvent réparti sur plusieurs pages web. Nous avons donc créé un script PHP permettant d'analyser les contenus desdits sites, puis d'en extraire seulement le nécessaire (dans notre cas, les haikus). Finalement, le script enregistre les haikus précédemment extraits dans une base de données.

Bien que le script est plus ou moins générique, les sites web sont fondamentalement différents par leur structure. Ils nous a donc fallu analyser ces différentes structures pour chaque site afin d'adapter notre script.

La plupart des sites ont des urls de la forme : `https://short-edition.com/fr/categorie/poetik/haiku-et-tanka?page=i`

Sur chacune des pages i se trouvent un certain nombre de haiku

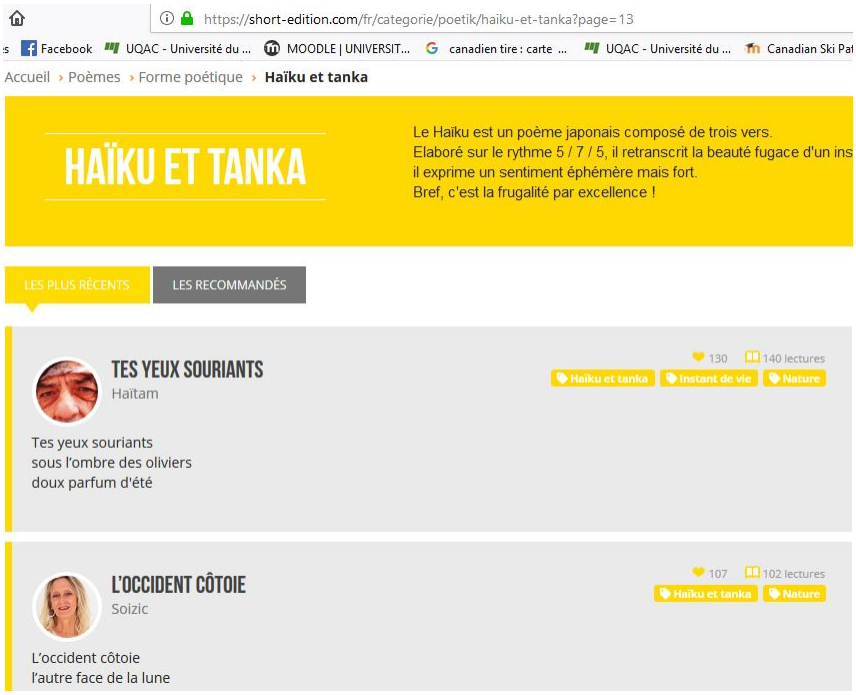

FIGURE 2 – Exemple de site web de haikus

Nous avons ainsi créé une variable i permettant d'incrémenter le nombre de pages puis nous avons utilisé la fonction PHP file_get_contents() pour "copier" le code source du site en question pour chaque url. Une fois le code source copié, il nous faut en extraire seulement le contenu des haikus. Nous pouvons en effet voir ci-dessous, une partie du code source avec en jaune la partie qu'il nous intéresse de récuperer.

Pour ce faire, nous avons utilisé des expressions régulières pour automatiser la récupération du contenu. Le but est de trouver un pattern de façon à savoir sur la page quand il est pertinent de récupérer le contenu. Nous pouvons par exemple voir sur le screen suivant, que les haikus sont contenus dans des div particulières, elles-mêmes contenues dans des div `<div class="chunk-content">`, à leur tour contenues dans des `<div class="js-infinite - scroll-content">`.

La regex suivante permet d'extraire chaque haiku sur la page : `<div class="js-infinite-scroll-content">(?s).*<div class="chunk-content"><div>(.+)<div>`

Finalement, une fois les haikus extraits, nous les passons à travers un dernier filtre permettant de supprimer les potentiels balises web (saut de ligne
 remplacé par saut de ligne universel) puis nous les ajoutons à notre base de données.

```
            </div></header><div class="title"><a href="/fr/auteur/soizic-4"
                  class="pull-left hidden-xs hidden-md" style="margin: 0 10px 5px 0"><img class="img-circ
  data-src="https://cdn.short-edition.com/uploads/f/soizic-4-image-2018-06-20-16-06-44.jpg" height="70"
  width="70"  alt="Image de Soizic"/></a><h2><a class="" href="/fr/oeuvre/poetik/paravent-retourne"
title="Lire l&#039;œuvre L'occident côtoie"
>L'occident côtoie</a></h2><p class="text-muted"><a href="/fr/auteur/soizic-4"
title="Toutes les œuvres de Soizic">Soizic</a></p></div><div class="clearfix"></div><div class="chunk-c
                    L'occident côtoie<br />l'autre face de la lune <br />l'orient apaisant<br /><br />
}     </span>

                                         
                                                <span class="post-lecture poetik">152 lectures</span><

            </div></header><div class="title"><a href="/fr/auteur/delo"
                  class="pull-left hidden-xs hidden-md" style="margin: 0 10px 5px 0"><img class="img-circ
  data-src="https://graph.facebook.com/932662373552059/picture?type=normal" height="70"
  width="70"  alt="Image de Delo"/></a><h2><a class="" href="/fr/oeuvre/poetik/les-baies-tendres-rouges-
title="Lire l&#039;œuvre Les baies tendres rouges"
>Les baies tendres rouges</a></h2><p class="text-muted"><a href="/fr/auteur/delo"
title="Toutes les œuvres de Delo">Delo</a></p></div><div class="clearfix"></div><div class="chunk-conte
                    Les baies tendres rouges<br />mûrissent en sucre épais<br />astres du soleil
            </div><br/></div><div class="clearfix"></div></div></article></div><div class="js-infinite-
}     </span>

                                         
                                                <span class="post-lecture poetik">290 lectures</span><
```

FIGURE 3 – Code source du site précédent

FIGURE 4 – Structure du site

FIGURE 5 – Expression régulière

Après, plusieurs essais, nous nous sommes rendu compte que la forme la plus adéquatte pour traiter nos données était un fichier texte contenant les différents haikus à la suite. C'est finalement, le format de dataset que nous avons retenu.

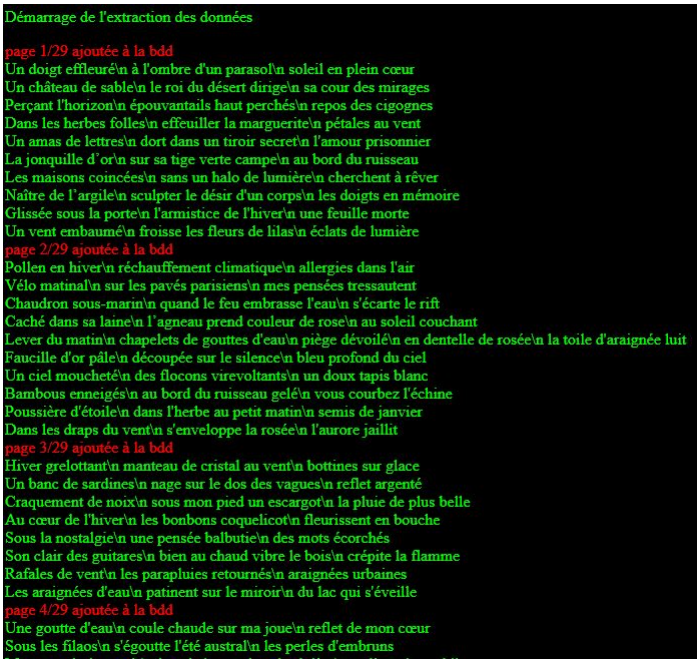

FIGURE 6 – Récupération des datas

	id	haiku	page
Éditer Copier Supprimer	126	Tombe la neige\n L'époque de Meiji\n Est déjà loin...	14
Éditer Copier Supprimer	127	Ah! mille flammes, un feu, la lumière,\n Une ombre...	15
Éditer Copier Supprimer	128	Au-dessus des feuillages\n S'élève gravement\n Le ...	15
Éditer Copier Supprimer	129	Quelque chose est perdue\n Quelque chose comme le ...	15
Éditer Copier Supprimer	130	Une table\n deux chaises\n blanches\n au fond du j...	15
Éditer Copier Supprimer	131	Comme un bras de désespoir,\n Parfois dans les vag...	15
Éditer Copier Supprimer	132	Dans la forêt verdoyante, mon ermitage.\n Seuls le...	15
Éditer Copier Supprimer	133	Soir de printemps\n Chose pénible entre toutes\n U...	15
Éditer Copier Supprimer	134	Front troué, sanglé dans la toile de tente,\n Sur ...	15
Éditer Copier Supprimer	135	La ronce\n N'est pas le pire	15
Éditer Copier Supprimer	136	Dans la première écluse\n La flûte pénètre\n Nupti...	16
Éditer Copier Supprimer	137	Soudain\n Une ombre passe\n Le vent.\n	16
Éditer Copier Supprimer	138	Dans un monde de rêve,\n Sur un bateau de passage,...	16
Éditer Copier Supprimer	139	Un rossignol chante -\n L'édifice de cette pension...	16
Éditer Copier Supprimer	140	S'il n'y avaient pas les ramiers,\n Les rochers\n...	16
Éditer Copier Supprimer	141	Pour connaître enfin la prune,\n Utiliser aussi to...	16
Éditer Copier Supprimer	142	la voix des roseaux\n bruit comme le vent d'automn...	16

FIGURE 7 – Stockage dans une BDD

1.2 Implémentation du RNN

L'ensemble des données obtenues est présenté dans un fichier texte, présent dans un dossier. La première étape est d'accéder au contenu du fichier depuis Python, sous une forme facile à traiter. La deuxième fonction getListOfProverbes() sera utilisée par la suite pour le classificateur :

```
############################
# ON CHARGE LES DONNEES TEXT
############################

from __future__ import absolute_import, division, print_function, unicode_literals

# ---------- Imports --------------
import os
import tensorflow as tf

# Juste pour enlever le warning AVX/FMA
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Conditions
tf.enable_eager_execution()

# Lien de la base de donnees
pathHaiku = os.path.expanduser("~/Desktop/Haiku.txt")
pathProverbes = os.path.expanduser("~/Desktop/Proverbes.txt")

# Retourne une liste avec chaque ligne du fichier texte
def getListOfHaikus():
    # Ouverture du fichier, remplissage de la liste, fermeture du fichier
    allHaikus = []
    with open(pathHaiku, 'r') as f:
        haiku = ""
        for line in f:
            if len(line)<3:
                allHaikus.append(haiku)
                haiku = ""
            else :
                haiku = haiku + line
    f.close()
    return allHaikus

def getListOfProverbes():
    allProverbes = []
    with open(pathProverbes, 'r') as f:
        for proverbe in f:
            allProverbes.append(proverbe)
    f.close()
    return allProverbes

```

FIGURE 8 – Implémentation de l'import du fichier texte contenant le dataset

Par la suite nous devons traiter les données du texte. Nous remarquons que celles-ci sont importées sous forme de liste, avec un haiku par case. La première idée était de faire fonctionner le RNN avec des données distinctes, et non un seul et unique texte. Hors il ne nous a pas été possible de trouver une solution satisfaisant cette attente. C'est pourquoi nous transformerons à nouveau cette liste en un texte entier. Malgré l'ajout de complexité en début de programme, qui n'a que peu d'incidence sur le temps de calcul par rapport au RNN, nous avons tout de même gardé cette implémentation car elle a l'avantage de compter le nombre de haiku présents dans le dataset.

La prochaine étape est de former un dictionnaire des caractères utilisés :

```
# Recuperer les donnees
datas = TextLoad.getListOfHaikus()

# Afficher la longueur du datas
print ('Longueur du datas : ' + str(len(datas)))
print()

# Detecter les caractères uniques de chaque haiku
listOfVocabsRaw = []
vocab = ''
for haiku in datas:
    vocab = sorted(set(haiku))
    listOfVocabsRaw.append(vocab)

# Supprimer les doublons des caracteres uniques de chaque haiku
listOfVocabs = []
i = 0
while i < len(listOfVocabsRaw) :
    for j in listOfVocabsRaw[i]:
        if j not in listOfVocabs:
            listOfVocabs.append(j)
    i += 1
print('Liste des caracteres uniques : ' + str(listOfVocabs))
vocab = listOfVocabs
print('Comparaison avec vocab : '+str(vocab))
print()
```

FIGURE 9 – Elaboration du vocabulaire et suppression des doublons

A chaque caractère identifié nous lui associons un numéro qui sera ensuite passé dans char2idx, construisant un dictionnaire compréhensible pour le RNN. Enfin l'ensemble des haikus, maintenant réunis en un seul texte, est traduit avec ce vocabulaire.

```
# Chaque caractere est associe a un numero
i = 0
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

allDatas = ''.join(datas)

texts_as_int = np.array([char2idx[c] for c in allDatas])
```

FIGURE 10 – Traduction du texte avec char2idx

L'étape suivante est l'entrainement qui se réalise sur un partie des données, séparée en entrées et cibles pour une première visualisation :

```python
# Taille maximum de la phrase
examples_per_epoch = len(allDatas)//seq_length

# Creer des exemples d'entrainement
char_dataset = tf.data.Dataset.from_tensor_slices(texts_as_int)

for i in char_dataset.take(5):
    print(idx2char[i.numpy()])
print()

# Slicer les exemples en sequences
sequences = char_dataset.batch(seq_length+1, drop_remainder = True)

for item in sequences.take(5):
    print(repr(''.join(idx2char[item.numpy()])))
print()

def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)
```

FIGURE 11 – Implémentation de l'entrainement

Ce qui fournit :

```
'Pansements durcis,\nVêtements flétris,\nVisages fermés.\nMon amour, viens dans ma bohème\nLe corps et le '
'cœur\nLibres\nLes enfants bavards\nNe l'attraperont jamais\nLa première luciole !\nCet homme\nEt sa suivant'
'e\nÉtaient dans l'air frais au temps des fleurs\nDu cerisier.\nElle fit\nDe son corps\nUn temple\nEt y mit\n'
'L'homme\nEn religion.\nArraché à la mort\nLe mince fil de ma vie\nRoseaux jaunissant de l'automne.\nAu mil'
'ieu des chrysanthèmes\nJe passe la main sur mes pommettes.\nQu'elles sont dures.\nLe blanc d'œuf\nDit au '

Input data:  'Pansements durcis,\nVêtements flétris,\nVisages fermés.\nMon amour, viens dans ma bohème\nLe corps et le'
Target data: 'ansements durcis,\nVêtements flétris,\nVisages fermés.\nMon amour, viens dans ma bohème\nLe corps et le '

Step    0
  input: 4 ('P')
  expected output: 6 ('a')
Step    1
  input: 6 ('a')
  expected output: 15 ('n')
Step    2
  input: 15 ('n')
  expected output: 17 ('s')
Step    3
  input: 17 ('s')
  expected output: 9 ('e')
Step    4
  input: 9 ('e')
  expected output: 14 ('m')
```

FIGURE 12 – Visualisation de l'entrainement

Le dataset est ensuite mélangé et nous pouvons construire le modèle à partir de la liste de vocabulaire utilisé. Notons que celui-ci ne devra plus être entrainé par la suite, étant donné que nous créons des points de sauvegarde des poids entre chaque epoch.

```
# Construction du modele
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    model = tf.keras.Sequential([
      tf.keras.layers.Embedding(vocab_size, embedding_dim,
                                batch_input_shape=[batch_size, None]),
      rnn(rnn_units,
          return_sequences=True,
          recurrent_initializer='glorot_uniform',
          stateful=True),
      tf.keras.layers.Dense(vocab_size)
    ])
    return model

model = build_model(
  vocab_size = len(vocab),
  embedding_dim=embedding_dim,
  rnn_units=rnn_units,
  batch_size=BATCH_SIZE)
```

FIGURE 13 – Construction du modèle

Nous utilisons également une fonction loss et un optimizer AdamOptimizer().

La génération de texte se fait à partir du modèle. Notons que la fonction de génération prend également en entrée un String. Cela pourra être une base pour la création d'haiku avec un thème imposé. Pour le moment n'ayant pas encore développé cette problématique nous indiquons une lettre aléatoire.

```
def generate_text(model, start_string):

    # Vectorisation de la premiere lettre
    input_eval = [char2idx[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    model.reset_states()
    for i in range(num_generate):
        predictions = model(input_eval)
        # Enleve la dimension du batch
        predictions = tf.squeeze(predictions, 0)

        # Dimension multinominale pour la prediction
        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()

        # On passe le mot predit en tant que prochaine entree du modele, ainsi que le prochain etat cache
        input_eval = tf.expand_dims([predicted_id], 0)

        text_generated.append(idx2char[predicted_id])

    return (start_string + ''.join(text_generated))
```

FIGURE 14 – Génération de texte

Pour finir, les résultats obtenus sont soumis au hyperparamètres utilisés. Les voici :

```
# --------- Parametres -------------
# Taille maximum de la phrase
seq_length = 100
# Batch size
BATCH_SIZE = 64
# Buffer size pour melanger le dataset
BUFFER_SIZE = 1000
# Embedding dimension
embedding_dim = 256
# Nombre d'unités RNN
rnn_units = 1024
# Nombre d'epochs
EPOCHS = 20
# Number of characters to generate
num_generate = 100
# Temperature -> faible pour un texte plus predictif -> important pour un texte plus surprenant
temperature = 0.2
```

FIGURE 15 – Paramètres utilisés

Les plus importants à noter sont le nombre d'epochs et la temperature. Nous avons imposé le premier à 20, nous rapprochant du surapprentissage. Cela est correct dans notre cas, car nous ne souhaitons pas la formation de nouveau mot ou l'apparition de structure trop exotique. Dans cette idée nous avons placé la température à 0.2, ce qui est bas et permet d'obtenir un texte plus prédictif. Néanmoins, plus bas engendrerait des répétitions dans les suites de caractères utilisés, ce qui n'est pas intéressant non plus.

Enfin nous limitons la séquence de caractères formés à 100. Néanmoins ce paramètres pourra être modifié par la suite suivant les besoins.

Le texte généré est donc d'une longueur de 100 caractères, correspondant à au moins 5 lignes dans la forme du haiku. L'intérêt est d'effectuer un post-traitement de ce résultat. En effet, comme indiqué précédemment, le premier mot est imposé par un caractère aléatoire, ce qui ne permet en général pas de former un vrai mot dans la première ligne, nous la supprimons donc. Puis la cinquième ligne est généralement incomplète, nous la supprimons également, ce qui nous laisse 3 lignes, nous rapprochant du format d'un vrai haiku.

Comme nous pouvons le voir, les résultats sont satisfaisants :

```
une petite feuille morte
le regard du matin
trouve la tête de l'aube
```

FIGURE 16 – Exemple 1 d'haiku généré

```
de la lune
le parfum des fleurs
du café noir
```

FIGURE 17 – Exemple 2 d'haiku généré

1.3 Elaboration d'un classificateur

La séquence du nombre de syllabes étant difficile à déterminer en français pour un simple programme, il parait judicieux d'utiliser un classificateur pour accomplir cette tâche. L'intérêt d'utiliser un classificateur est de pouvoir évaluer la précision et l'efficacité du générateur RNN. Pour cela il doit être en mesure de classer le texte en fonction de sa structure afin de déterminer si l'entrée correspond à un haïku ou non.

Le classificateur d'haïku comporte la structure d'un classificateur de texte classique. L'idée étant de voir dans un premier temps si un tel algorithme serait judicieux pour cette tâche. En effet après une observation de plusieurs haikus, une structure particulière des phrases semble se détacher. Avec la contrainte des syllabes, les phrases sont courtes, comportent essentiellement des noms et des adjectifs. On peut alors supposer que l'algorithme pourrait apprendre à différencier les haikus à partir de certains mots souvent utilisés. Pour ce classificateur on utilise la librairie sklearn.

Le dataset des données d'entrainement ou de test à la composition suivante :

- Un tableau de chaines de charactères Xtrain ou Xtest qui comporte les données à classifier. La moitié sont des haïkus et l'autre des proverbes ou citations quelconques. Ces derniers sont mélangés afin de ne pas fausser l'entrainement.
- Un tableau d'entier Ytrain ou Ytest qui correspond à la sortie associée pour chaque ième élément du tableau X. Avec comme valeur 1 si le texte est un haïku, 0 sinon.

Pour créer ces tableaux, on utilise le fichier TextLoad.py (annexe) où la méthode getTrainingData() renvoit les 4 tableaux X et Y à partir de 2 fichiers textes contenant les données (proverbes.txt et haikutrain.txt). Le nombre d'entités du dataset d'entrainement est de 5645, avec la moitié de haiku environ. Le dataset de set est de 200 entités avec la moitié d'haïkus également.

Pour le pré-traitement de texte, les données sont d'abord tokenisées, chaque donnée sera sous la forme d'un bag of world avec leur occurrence. Puis on convertit les occurrences en fréquences de terme.

Pour l'entrainement, on utilise dans un premier temps un classificateur Naïve Bayes que nous tenterons d'améliorer. Pour rendre le classificateur plus facile à utiliser nous utilisons ensuite un pipeline pour alléger le code par la suite. Avec ces classificateurs nous obtenons une performance de 89,5% ce qui est plutôt satisfaisant.

Afin d'améliorer la performance, nous allons utiliser un « support vector machine » (SVM), un peu plus long mais l'un des plus performants dans la classification de texte. La performance enregistrée est de 90% soit très peu en plus comparé au Naïve Bayes.

De nombreux paramètres sont à déterminer pour ce classificateur, pour optimiser ceux-ci nous allons utiliser un algorithme d'optimisation GridSearch. Avec celui-ci nous obtenons une performance de 90,5% ce qui est toujours très proche des méthodes vues précédemment.

Voici les résultats obtenus :

```
Performance avant Pipeline :  0.895
Performance apres pipeline :  0.895
Performance apres ajout SVM :  0.9
Performance avec GridSearch :  0.905
Best score gs :  0.882117703552723
clf__alpha: 0.0001
tfidf__use_idf: True
vect__ngram_range: (1, 2)
```

FIGURE 18 – Résultats de classification

2 Perspectives

2.1 Construction avec thème imposé

L'idée serait de créer des haikus à partir d'un mot donné en entrée. Dans cette optique plusieurs pistes s'offrent à nous.

Premièrement nous pensons au générateur qui comprend déjà une entrée. Nous pourrions simplement y indiquer le mot recherché et la conception se réaliserait à partir de celui-ci. Cependant le principal problème est que le mot demandé sera nécessairement au début du texte généré.

Une autre idée serait de générer autant de lignes nécessaires à ce que le mot apparaisse au hasard puis sélectionner les lignes l'entourant. Néanmoins, le processus peut être très long, demandant beaucoup trop de mémoire, et le mot pourrait peut-être même ne jamais apparaitre.

Enfin, nous pourrions combiner les deux idées pour obtenir un compromis. Nous pourrions obliger le générateur à insérer le mot à partir d'une condition, comme par exemple lorsque ses deux ou trois premiers caractères ont déjà été générés il devra les compléter avec l'entrée demandée. Il continuera de produire des caractères tant que cette condition n'est pas remplie.

2.2 Changement de classificateur

Nous avons réussi à obtenir un classificateur d'haïku plutôt performant néanmoins l'interprétation de ces résultats reste mitigée. Bien que la performance soit élevée, il faut prendre en compte le faible contenu du dataset de test. Il faudrait disposer d'un dataset de textes de type non-haïku plus divers pour avoir une meilleure précision de la performance.

Mais surtout, ce qui est à retenir c'est le très mince écart de performance entre les différents algorithmes. En réfléchissant plus profondément au fonctionnement de cette méthode de classification, il est possible qu'avec ce dataset trop faible, les résultats soient faussés : en effet on a classifié des données en fonction de l'occurrence des mots qu'ils les composaient. Or si une citation et un haïku parlent de la même chose, il peut s'avérer difficile de les différencier avec cette méthode. S'il existe un champ lexical relatif aux proverbes du dataset la classification peut être faussée.

Enfin il ne faut pas oublier la différence entre un haïku et un proverbe : la structure. Pour le prochain sprint nous allons développer un classificateur se basant sur la structure du texte et non pas sur son contenu : Le pré-traitement de texte se fera par séquençage de mot. Chaque mot sera remplacé par le nombre de caractères qui le compose. Ainsi, le classificateur pourra apprendre d'une manière similaire au nombre de syllabes à chaque ligne. Nous ferons également en sorte, si possible, de prendre en compte les sauts de lignes.

2.3 Élaboration d'un site web

Nous pourrions adapter notre RNN pour créer un site web qui génère des haikus. En effet, Tensorflow peut être implémenté en javascript. Il suffirait alors d'inclure une version entraînée de notre réseau de neuronne sur un serveur puis de laisser la possiblité à l'utilisateur de les générer aléatoirement en cliquant sur un bouton (voir figure suivante).

FIGURE 19 – Projet de site web

Bilan

Le travail effectué lors de ce premier sprint nous a permis d'obtenir des résultats très satisfaisants. Grâce à l'ensemble de haikus existants récoltés, notre RNN est capable de produire des textes de structure similaire. Le classificateur, bien que pas encore lié à notre RNN pour le moment, permettra de valider les résultats obtenus. Nous pourrions par la suite l'utiliser afin de préciser les hyperparamètres optimaux.