Projet haikus_Notebook_partie analyse_20181209

December 9, 2018

1 Peut-on prédire le sexe des auteurs de haïkus ?

Python pour un data scientist - Décembre 2018

Auteurs : Samuel Allain, Maxime Bergeat, Gabriel Buresi

Dans ce notebook à exécuter après la récupération des haïkus disponibles sur le site de Temps Libres (http://www.tempslibres.org/tl/en/dbhk00.html), on cherche à prédire le sexe des auteurs de haïkus disponibles sur le site, en ne considérant que les poèmes écrits en français. Seuls les haïkus en français ont été sélectionnés. Le sexe des différents auteurs du site a été codé manuellement en fonction du prénom. Pour certains auteurs utilisant un pseudo, l'information n'est pas disponible est les haïkus en question ne sont pas utilisés dans l'analyse.

In [45]:
```
# Pour ne pas afficher les warnings en sortie... import warnings; warnings.simplefilter

## Import des librairies utilisées dans le notebook

# Manipulation algébriques et de données
import os
from os import path
import numpy as np
import pandas as pd
import time
from sklearn.pipeline import make_pipeline

# Statistiques descriptives
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Gestion du texte
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

# Machine learning
from sklearn import svm, datasets, model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_auc_score, roc_curve, auc, confusion_matrix

# Visualisation résultats
import seaborn as sn
import matplotlib.pyplot as plt
% matplotlib inline
```

1.1 Statistiques descriptives

Pour commencer, on donne quelques statistiques descriptives sur les haïkus de la base de données, notamment pour savoir les mots les plus fréquemment cités par les hommes et les femmes.

In [39]:
```
# Import des données
os.chdir("C:/Users/maxim/Documents/ensae-python-2018/Data/")
base_haikus = pd.read_csv("haikus_scraping_v2_20181110.csv","|")
base_haikus.head()
```

Out[39]:

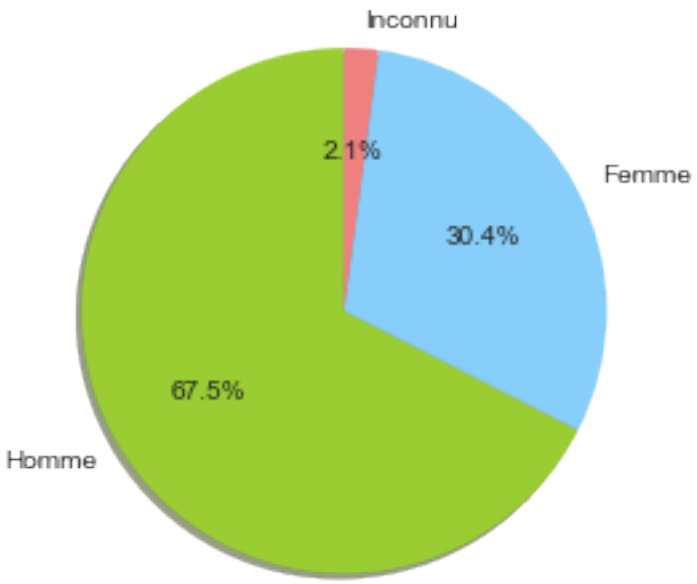
	url	id_auteur	auteur	sexe	pays	haiku
0	http://www.tempslibres.org/tl/tlphp/dbhk03.php...	quin-p	Philippe Quinta	H	France	Fut-elle pleine\\déjà, elle décroit\\la lune
1	http://www.tempslibres.org/tl/tlphp/dbhk03.php...	py-d	Daniel Py	H	France	Lune parfaitement ronde\\Buée sur la vitre
2	http://www.tempslibres.org/tl/tlphp/dbhk03.php...	alex-m	Marlène Alexa	F	Egypte	hiver\\dans les yeux de grand-mère\\une ombre ...
3	http://www.tempslibres.org/tl/tlphp/dbhk03.php...	sang-r	Rahmatou Sangotte	F	France	ciel gris -\\une odeur d'oignons\\caramélisés
4	http://www.tempslibres.org/tl/tlphp/dbhk03.php...	rais-c	Carol Raisfeld	F	USA	la fille du pasteur\\sa robe du dimanche\\de l...

In [37]:
```
print("On dispose de " +repr(len(base_haikus))+" haïkus dans la base de données scrapée
```

```
On dispose de 3477 haïkus dans la base de données scrapée.
```

In [21]:
```
# Nombre de haïkus selon le sexe de l'auteur.
labels = 'Homme', 'Femme', 'Inconnu'
```

```
freq = list(base_haikus['sexe'].value_counts())
colors = ['yellowgreen', 'lightskyblue', 'lightcoral']
plt.pie(freq, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90)
plt.axis('equal')
plt.title("Répartition des haïkus selon le sexe de l'auteur (N)")
plt.show()
```



Pour quelques haïkus, on ne connaît pas le sexe de l'auteur. Ces haïkus seront retirés de l'analyse par la suite pour coder le sexe de l'auteur. On constate que le jeu de données n'est pas complètement balancé (plus d'hommes que de femmes dans les données). Certaines méthodes utilisées ensuite pour la classification vont conduire à prédire beaucoup plus souvent que les auteurs sont des hommes que des femmes. Cet élément sera discuté dans la suite.

Par ailleurs, différents tests ont été réalisés pour construire un jeu de données balancé avec 50% de haïkus écrits par des femmes. Les résultats en termes de pouvoir prédictif n'améliorant pas significativement les modèles, ces résultats ne sont pas présentés, et on se base dans la suite sur le jeu de données initial après avoir retiré les haïkus pour lesquels le sexe de l'auteur est inconnu.

```
In [41]: # Nuage de mots pour indiquer les mots les plus cités par les auteurs de haïkus
         haikus = str(base_haikus["haiku"])
         wordcloud = WordCloud(background_color='white',
                               width=2500,
                               height=1800,
                              max_words=100).generate(haikus)

         # Affichage du nuage de mots
         plt.imshow(wordcloud, interpolation='bilinear')
```

```
plt.axis("off")
plt.show()
```

On constate ici que les mots les plus souvent cités par les auteurs de haïkus sont le plus souvent des mots vides de sens comme des articles. On va donc effectuer la représenation sous forme de nuage de mots en excluant les mots vides (*stopwords*).

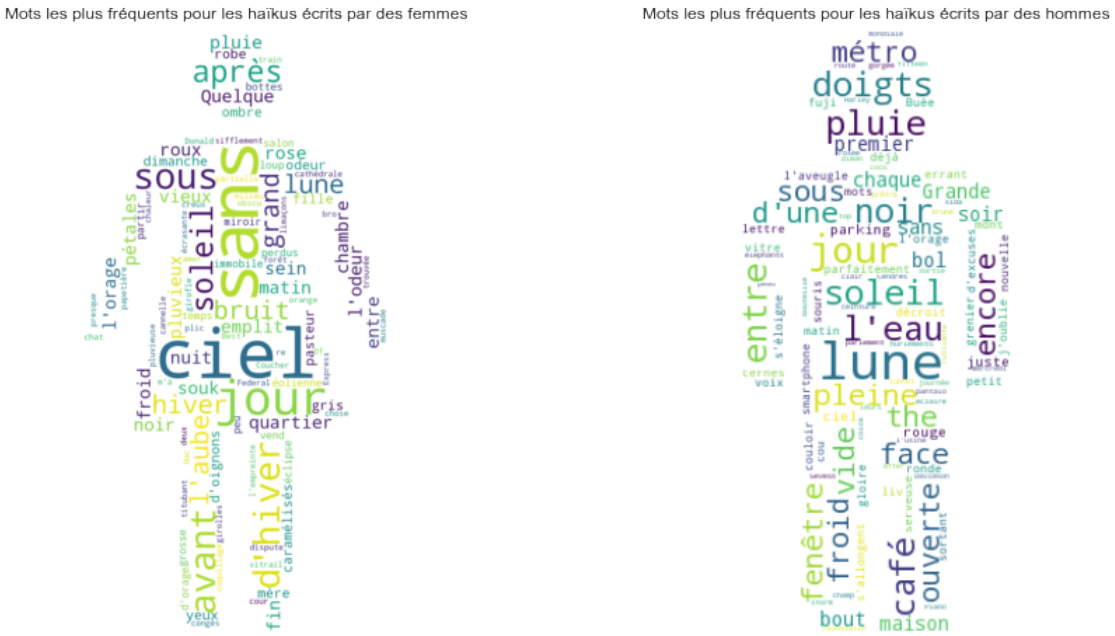
```
In [46]: # Création de la liste de stopwords. On utilise la liste du package stopwords en ajouta
         # dans la liste initiale.
         mots_vides = stopwords.words('french')
         mots_vides.append("les")
         stopwords=set(mots_vides)

         # Nuage de mots amélioré
         wordcloud2 = WordCloud(   stopwords=stopwords,
                                   background_color='white',
                                   width=2500,
                                   height=1800,
                                  max_words=100,
                                   ).generate(haikus)
         plt.imshow(wordcloud2, interpolation="bilinear")
         plt.axis('off')
         plt.show()
```

On note que les mots les plus cités font souvent référence à la nature (**ciel**, **lune**, **soleil**...). Cela est dû à la **règle du kigo** que doivent respecter les auteurs de haïkus. La règle du kigo indique en effet que chaque haïku doit comporter un kigo, c'est-à-dire un "mot de saison" faisant référence à la nature ou à l'une des quatre saisons. En effet,la représentation des saisons est très importante dans la culture japonaise (https://fr.wikipedia.org/wiki/Kigo). On s'intéresse maintenant aux mots les plus cités par les hommes et femmes auteurs de haïkus.

```
In [49]: # Jeux de données pour construires nuages de mots séparés pour les hommes et les femmes
         haikus_F = str(base_haikus[base_haikus['sexe'] == "F"]["haiku"])
         haikus_H = str(base_haikus[base_haikus['sexe'] == "H"]["haiku"])

         # Pour les femmes
         masqueF = np.array(Image.open("images/symbole-femme.png"))
         wordcloud_femmes = WordCloud(stopwords=stopwords,
                                      background_color="white",
                                      #mode="RGBA",
                                      max_words=100,
                                      mask=masqueF).generate(haikus_F)

         # Pour les hommes
         masqueH = np.array(Image.open("images/symbole-man2.png"))
         wordcloud_hommes = WordCloud(stopwords=stopwords,
                                      background_color="white",
                                      mode="RGBA",
                                      max_words=100,
                                      mask=masqueH).generate(haikus_H)
```

```
# Tracé sur un même graphique
fig = plt.figure(figsize = (15,8))
ax1 = fig.add_subplot(1,2,1)
plt.imshow(wordcloud_femmes, interpolation="bilinear")
plt.axis("off")
plt.title("Mots les plus fréquents pour les haïkus écrits par des femmes")
ax2 = fig.add_subplot(1,2,2)
plt.imshow(wordcloud_hommes, interpolation="bilinear")
plt.axis("off")
plt.title("Mots les plus fréquents pour les haïkus écrits par des hommes")
```

```
Out[49]: <matplotlib.text.Text at 0x1ce47653400>
```



On constate effectivement quelques différences. Par exemple, le mot **ciel** est plus souvent employé par des femmes, alors que les hommes utilisent plus souvent le mot **lune** pour ces données. Pour essayer d'exploiter ces différences, on met en place des modèles d'apprentissage supervisé (*machine learning*) afin de prédire le sexe des auteurs de haïkus.

1.2 Modèles de classification

Pour effectuer la classification, on considère l'ensemble des mots comme prédicteurs, et on teste différents modèles pour prédire le sexe. Plusieurs paramétrages ont été testés mais seulement certains sont présentés dans ce notebook : - En ce qui concerne l'inclusion des *stopwords* dans l'analyse, il a été décidé des les inclure étant donné que les textes étudiés sont très courts. Les résultats sont similaire si les *stopwords* sont utilisés dans les modèles. - Différents paramétrages ont été testés pour utiliser des *1*-grammes, *2*-grammes ou *3*-grammes dans les modèles de classification. Les résultats sont ici présentés en considérant comme régresseurs potentielles tous les *1*-grammes,

2-grammes et *3*-grammes dans les données. Toutefois, les résultats sont relativement similaires en ne prenant en compte que les *1*-grammes (mots). - Pour obtenir la matrice document-terme servant de base à l'analyse, nous avons testé l'utilisation de variables dichotomiques (présence ou non de chaque n-gramme dans le haïku) ou bien la mobilisation du score *tf-idf* pour pondérer chaque *n*-gramme en fonction de sa rareté dans le corpus de haïkus à disposition (en effet, la pondération en *tf* joue peu ici étant donné qu'il est rare qu'un mot soit répété plus d'une fois dans un même haïku !). Les résultats sont ici présentés avec l'application de la pondération *tf-idf*. Bien que cela soit théoriquement plus acceptable d'utiliser cette pondération (pour notamment sous-pondérer les *stopwords*), son application n'améliore pas de façon importante le pouvoir prédictif des modèles par rapport à la mobilisation de variables dichotomiques. - Le nombre de variables disponibles pour la prédiction est donc très important (plus de variables que d'observations). Une application de méthode d'analyse de données (Analyse en Composantes Principales) pour réduire la dimensionnalité des variables utilisées pour la classification est présentée en conclusion du notebook.

1.2.1 Création des jeux de données pour l'entraînement et la validation des modèles

```
In [51]: # On ne conserve que les auteurs pour lesquels le sexe est déterminé, et on crée
         # les jeux d'apprentissage et de validation.
         base_haikus_model = base_haikus[base_haikus.sexe != "?"]


         X_train, X_test, y_train, y_test = train_test_split(
                        base_haikus_model['haiku'], base_haikus_model["sexe"])
         print("On dispose de " +repr(len(base_haikus_model))+" haïkus dans la base de données u
```

```
On dispose de 3404 haïkus dans la base de données utilisée pour la classification.
```

```
In [52]: # Création des variables utilisées pour la classification.
         # On considère ici l'ensemble des 1-grammes, 2-grammes et 3-grammes, et on applique une

         pipe = make_pipeline(CountVectorizer(ngram_range=(1,3)), TfidfTransformer())

         pipe.fit(X_train)
         feat_train = pipe.transform(X_train)

         feat_test = pipe.transform(X_test)
         feat_test.shape
```

```
Out[52]: (2553, 39749)
```

On dispose donc en considérant l'ensemble des *3*-grammes des données qu'on dispose d'un nombre très important de variables potentiellement mobilisables pour la classification (près de 40 000). Pour la suite, on présente : - Des résultats comparatifs sur trois modèles simples utilisés en apprentissage supervisé forêts aléatoires, régression logistique, classifieur naïf bayésien, réseaux de neurones) - Une discussion sur l'affinage des hyper-paramètres des modèles. - Une extension concernant la réduction de la dimensionnalité pour les variables utilisées pour la prédiction

1.2.2 Comparaison entre trois modèles

On s'intéresse ici au pouvoir prédictif pour trois modèles pour coder le sexe des auteurs de haïkus. On s'intéresse également à un classifieur idiot qui prédit le sexe de l'auteur en fonction de la distribution observée sur les données initiales. Ce modèle joue le rôle de *benchmark* pour les modèles considérés.

Modèle idiot

```
In [65]: idiot = DummyClassifier(strategy="stratified")
         # Entraînement modèle
         idiot.fit(X=feat_train, y= y_train)
         # Prédiction sur le jeu test
         idiot.score(X=feat_test, y= y_test)
```

```
Out[65]: 0.5863689776733255
```

En moyenne avec le modèle idiot, on prédit correctement le sexe de l'auteur pour 58% des haïkus de la base de test.

Forêts aléatoires

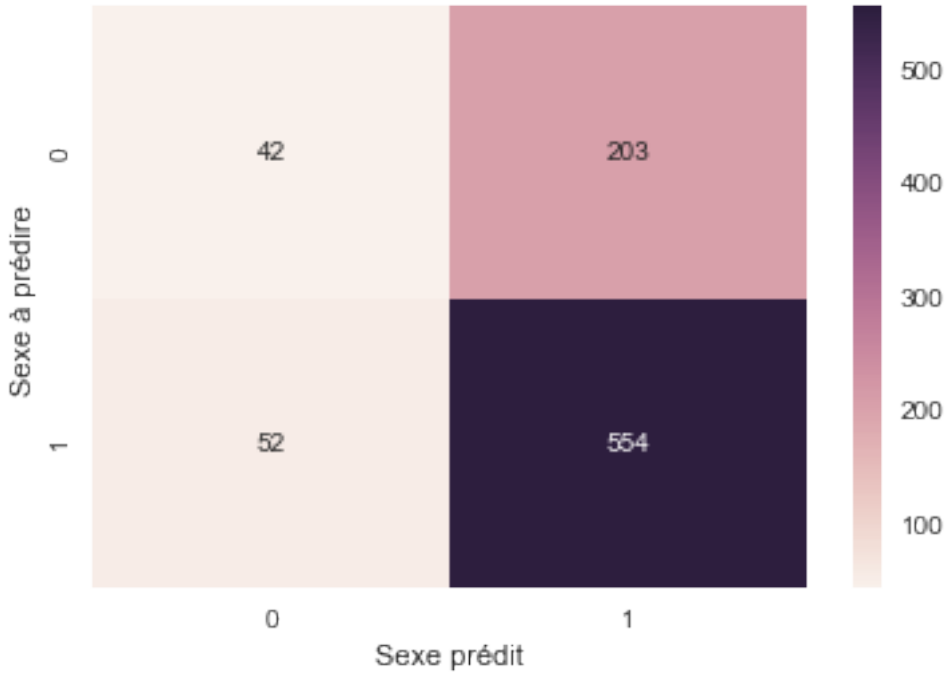
```
In [80]: ## Forêt aléatoire avec 50 arbres
         foret = RandomForestClassifier(n_estimators=50)
         # Entraînement modèle
         foret.fit(feat_train, y_train)
         # Prédiction sur le jeu test
         foret.score(feat_test, y_test)
```

```
Out[80]: 0.7015276145710928
```

Le score ici présenté est celui de l'exactitude. Sur la base de données de test, il correspond au nombre de haïkus dont le sexe de l'auteur est bien identifié. Pour préciser les résultats, on peut tracer la matrice de confusion.

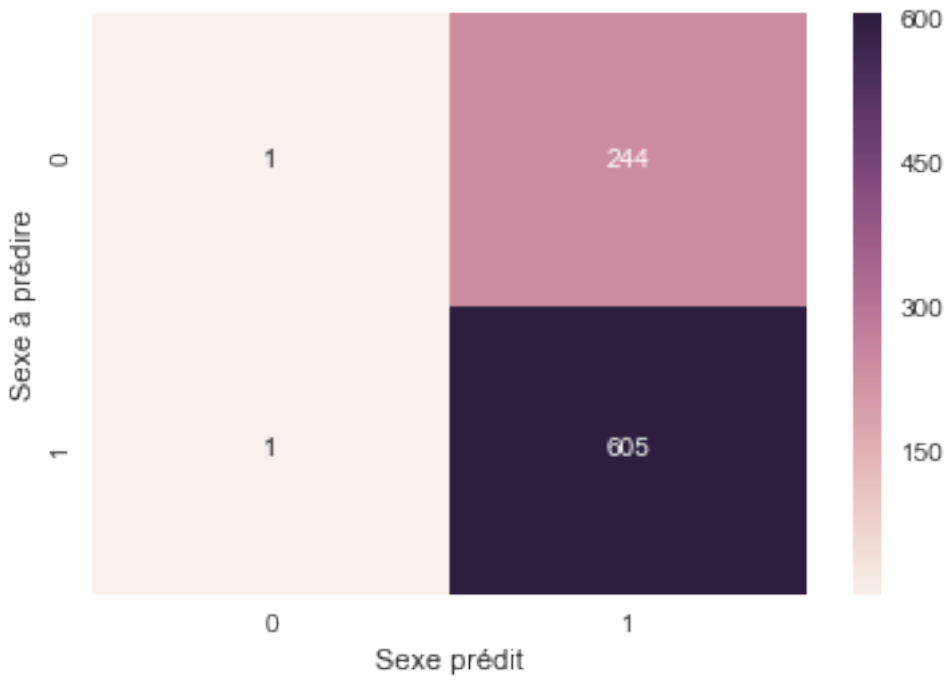
```
In [66]: ## Visualisation avec package sns (carte de chaleur)
         y_pred = clf.predict(feat_test)
         cnf_matrix = confusion_matrix(y_test, y_pred)
         matrice_a_tracer = pd.DataFrame(cnf_matrix)

         plt.figure()
         sn.heatmap(matrice_a_tracer, annot=True, fmt ="d")
         plt.xlabel("Sexe prédit")
         plt.ylabel("Sexe à prédire")
```

```
Out[66]: <matplotlib.text.Text at 0x1ce4dc41d30>
```



Régression logistique

```
In [76]: # On utilise ici la pénalité l1 (résultats similaires avec pénalité l2)
         lreg = LogisticRegression(penalty= "l2")
         # Entraînement modèle
         lreg.fit(X= feat_train, y=y_train)
         # Prédiction sur le jeu test
         lreg.score(X=feat_test, y=y_test)

Out[76]: 0.7121034077555817

In [77]: y_pred = lreg.predict(feat_test)
         cnf_matrix = confusion_matrix(y_test, y_pred)
         matrice_a_tracer = pd.DataFrame(cnf_matrix)

         plt.figure()
         sn.heatmap(matrice_a_tracer, annot=True, fmt ="d")
         plt.xlabel("Sexe prédit")
         plt.ylabel("Sexe à prédire")

Out[77]: <matplotlib.text.Text at 0x1ce4e307ba8>
```

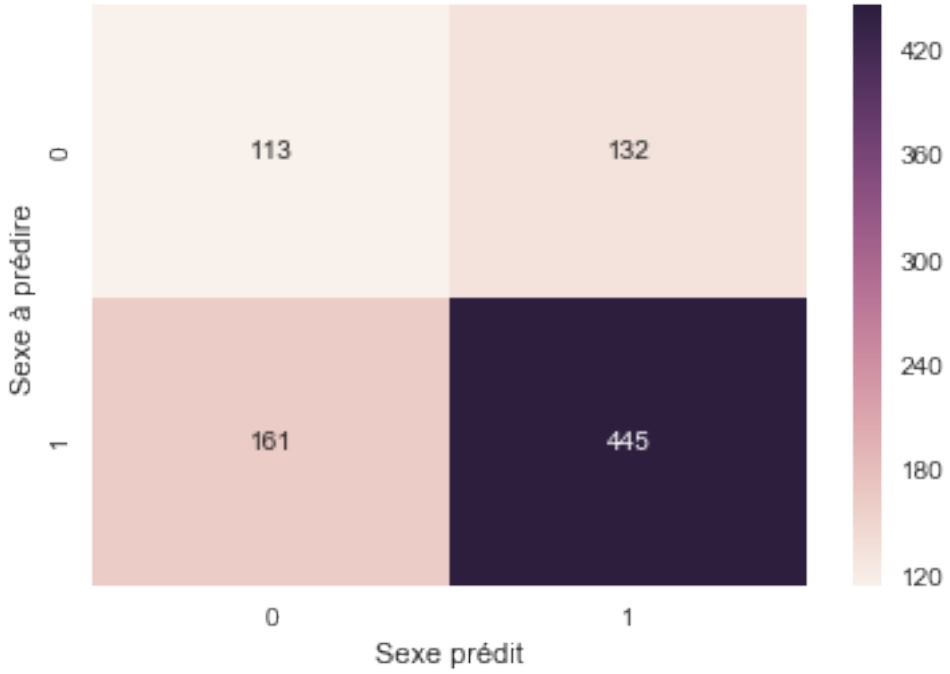


On constate ici que le modèle prédit presque exclusivement que les auteurs sont des hommes. Ce désavantage de la méthode de régression logistique est probablement dû au fait que le jeu de données comporte plus d'auteurs masculins que féminins.

Classifieur naïf bayésien

```
In [84]: ## NB : pour cette méthode, on doit convertir les matrices avec les régresseurs pour au
         feat_train_nosparse = feat_train.toarray()
         feat_test_nosparse =feat_test.toarray()

         bayes = GaussianNB()
         # Entraînement modèle
         bayes.fit(feat_train_nosparse, y_train)
         # Prédiction sur le jeu test
         bayes.score(feat_test_nosparse, y_test)

Out[84]: 0.6556991774383079

In [86]: y_pred = bayes.predict(feat_test_nosparse)
         cnf_matrix = confusion_matrix(y_test, y_pred)
         matrice_a_tracer = pd.DataFrame(cnf_matrix)

         plt.figure()
         sn.heatmap(matrice_a_tracer, annot=True, fmt ="d")
         plt.xlabel("Sexe prédit")
         plt.ylabel("Sexe à prédire")
```

Out[86]: <matplotlib.text.Text at 0x1ce00136ac8>



Contrairement au modèle de régression logistique, le modèle ne prédit pas que des auteurs masculins.

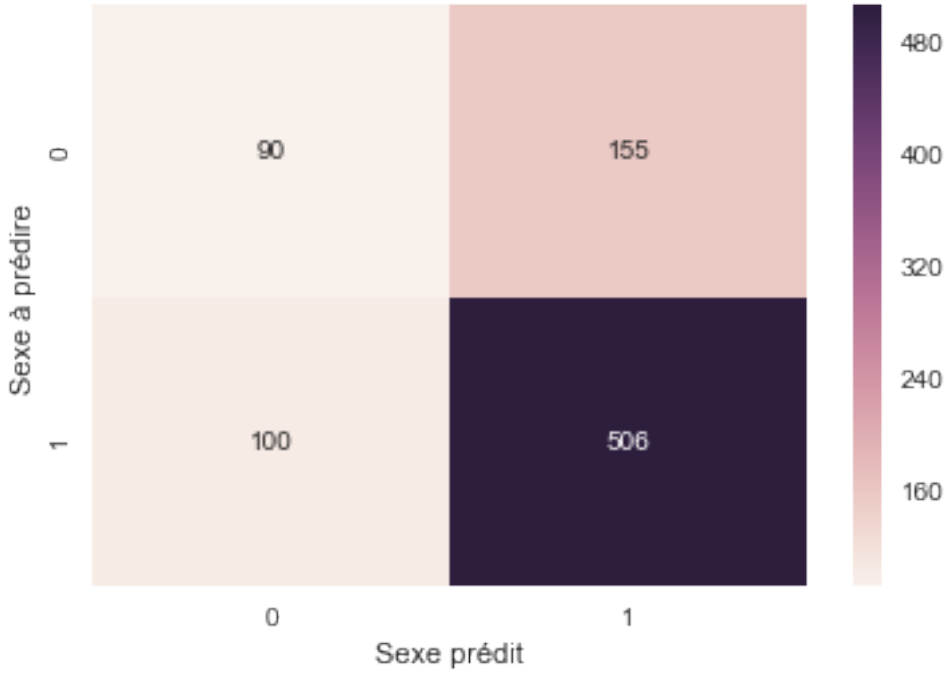
Réseau de neurones

```
In [78]: # Réseau de neurones - on utilise ici un modèle de réseau de neurones standard - multi-
         # 100 couches cachées.
         reseau_neur = MLPClassifier(hidden_layer_sizes=(100,))
         # Entraînement modèle
         reseau_neur.fit(X=feat_train, y= y_train)
         # Prédiction sur le jeu test
         reseau_neur.score(X=feat_test, y= y_test)
```

Out[78]: 0.700352526439483

```
In [79]: y_pred = reseau_neur.predict(feat_test)

         cnf_matrix = confusion_matrix(y_test, y_pred)
         matrice_a_tracer = pd.DataFrame(cnf_matrix)

         plt.figure()
         sn.heatmap(matrice_a_tracer, annot=True, fmt ="d")
         plt.xlabel("Sexe prédit")
         plt.ylabel("Sexe à prédire")
```

```
Out[79]: <matplotlib.text.Text at 0x1ce4e4279b0>
```

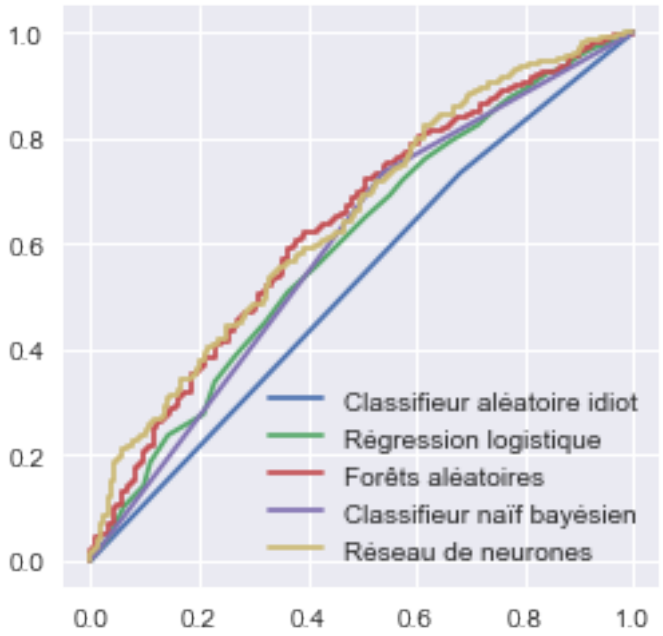


Un important avantage de la classification avec réseau de neurones est que le modèle prédit plus souvent d'auteurs féminins (dans des proportions similaires à la proportion observée dans les données). L'exactitude du modèle est aux alentours de 70%, soit un score proche des modèles précédents

Comparaison des modèles précédents avec courbes Roc

```
In [88]: ### Courbes ROC pour comparer les différentes approches.
         np_train = np.hstack([feat_train.todense()])
         np_test = np.hstack([feat_test.todense()])
         pmodelIdiot = idiot.predict_proba(np_test)[:, 1]
         pmodelFor = foret.predict_proba(np_test)[:, 1]
         pmodelreg = lreg.predict_proba(np_test)[:, 1]
         pmodelBayes = bayes.predict_proba(np_test)[:, 1]
         pmodelResNeur = reseau_neur.predict_proba(np_test)[:, 1]

In [92]: # Conversion de l'objet y en numérique (0/1 plutôt que homme/femme pour pouvoir tracer
         y_test_num= (y_test == "H")
         fpr1, tpr1, th1 = roc_curve(y_test_num, pmodelIdiot)
         fpr2, tpr2, th2 = roc_curve(y_test_num, pmodelFor)
         fpr3, tpr3, th3 = roc_curve(y_test_num, pmodelreg)
         fpr4, tpr4, th4 = roc_curve(y_test_num, pmodelBayes)
         fpr5, tpr5, th5 = roc_curve(y_test_num, pmodelResNeur)
```

```
In [94]: fig, ax = plt.subplots(1,1, figsize=(4,4))
         ax.plot(fpr1, tpr1, label='Classifieur aléatoire idiot')
         ax.plot(fpr2, tpr2, label='Régression logistique')
         ax.plot(fpr3, tpr3, label='Forêts aléatoires')
         ax.plot(fpr4, tpr4, label='Classifieur naïf bayésien')
         ax.plot(fpr5, tpr5, label='Réseau de neurones')

         ax.legend();
```



On constate que les modèles ici utilisés sont légèrement meilleurs que le modèle idiot qui reproduit la proportion observée dans les données initiales. Toutefois, les scores d'exactitude observées restent relativement faibles, aux alentours de 70%, ce qui laisse supposer qu'il est en fait difficile de prédire le sexe des auteurs de haïkus, ce qui semble intuitivement assez logique étant donné le peu de texte à disposition. Pour préciser les résultats, on peut comparer les modèles en construisant divers jeux d'apprentissage et de test (validation croisée). Par ailleurs, un écueil important de certains modèles testés (notamment régression logistique) est qu'ils prédisent que l'ensemble (ou quasiment) des auteurs étudiés sont des hommes. Pour pallier cette difficulté, on va utiliser une autre métrique pour mesurer la précision des résultats désormais. On utilisera désormais le score **f1_macro**, qui est une moyenne non pondéré du score F1 pour les hommes et pour les femmes. Le score F1 est défini pour une catégorie comme la moyenne géométrique de la précision (part de bien classés parmi les personnes codées comme une catégorie) et du rappel (proportion d'enregistrements retrouvés pour les observations de cette catégorie). Ainsi, le score F1 pour les femmes vaudra 0 dans le cas où le modèle ne prédit que des auteurs masculins ce qui sera donc très pénalisant.

Comparaison des modèles avec validation croisée

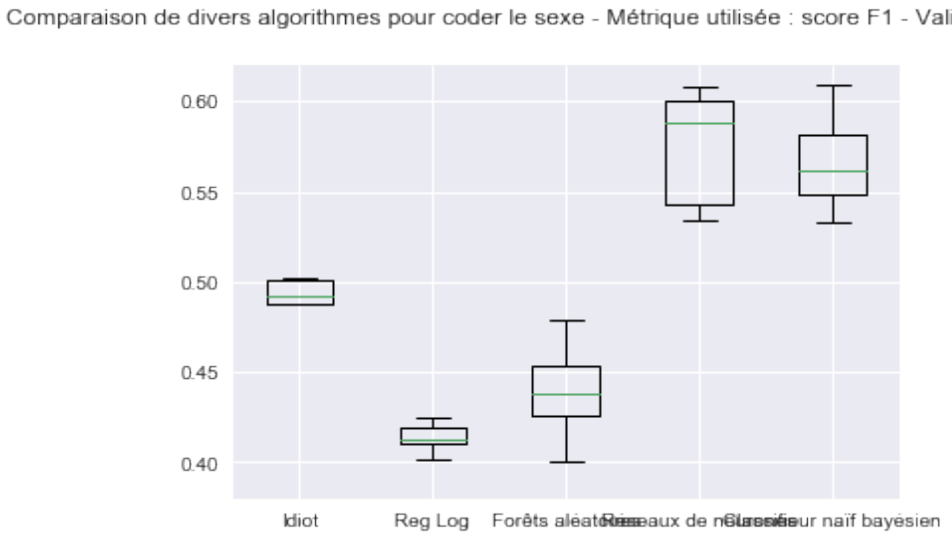
In [95]:
```
## On crée les jeux de covariables et de la variable à classifier à partir des données
# (le découpage apprentissage/test étant effectué plusieurs fois car on utilise la vali
X = pipe.transform(base_haikus_model['haiku']).toarray()
Y = base_haikus_model['sexe']

t_debut = time.time()

# Apprentissage des modèles
models = []
models.append(("Idiot", DummyClassifier(strategy="stratified")))
models.append(('Reg Log', LogisticRegression(penalty = "l1")))
models.append(('Forêts aléatoires', RandomForestClassifier(n_estimators=50)))
models.append(('Réseaux de neurones', MLPClassifier()))
models.append(('Classifieur naïf bayésien', GaussianNB()))
# evaluate each model in turn

results = []
names = []
scoring = 'f1_macro'

# Evaluation de chaque modèle à partir de validation croisée (découpage de l'échantillo
# Le score considéré ici est le score F1 macro comme indiqué.
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=None)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

t_fin = time.time()
temps_exec = t_fin - t_debut
print("Le fitting de l'ensemble de ces modèles a duré "+repr(temps_exec)+" secondes.")
```

```
Idiot: 0.492600 (0.019919)
Reg Log: 0.412211 (0.009390)
Forêts aléatoires: 0.442877 (0.029043)
Réseaux de neurones: 0.574517 (0.029617)
Classifieur naïf bayésien: 0.565178 (0.021956)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-95-cbc86cbf941d> in <module>()
 28
```

```
     29 t_fin = time.asctime()
---> 30 temps_exec = t_fin - t_debut
     31 msg_temps = "Temps d'exéccution en secondes de" % (temps_exec)
     32 print(msg_temps)


TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

In [96]:
```
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparaison de divers algorithmes pour coder le sexe - Métrique utilisée
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
## Peut encore (beaucoup...) s'améliorer... A voir avec les matrices de confusion si le
```



On constate ici que, lorsque la métrique F1 est utilisée pour comparer les modèles, pénalisant ainsi plus fortement les modèles ayant tendance à prédire seulement des auteurs masculins, la classification avec des réseaux de neurones ou des classificateurs naïfs bayésiens obtiennent de meilleures performances. Encore une fois, on constate que les meilleurs modèles obtiennent seulement des performances légèrement meilleures par rapport à un modèle fondé sur l'aléaoire. Cela laisse supposer qu'il est difficile de prédire le sexe des auteurs de haïkus seulement à partir du texte, ce dernier étant peut-être trop court pour permettre de déterminer le sexe de l'auteur.

1.2.3 Discussion : affinage des hyperparamètres

Dans la section précédente, la plupart des paramètres choisis pour les modèles étaient les paramètres par défaut des méthodes utilisées. On peut également tester une multitude de modèles

avec différents hyperparamètres de façon à optimiser les modèles. Cela a été fait pour l'ensemble des modèles présentés ci-dessus, mais on présente ici seulement quelques intuitions pour les forêts aléatoires. On fait varier trois types de paramètres : le nombre maximum de n-grammes utilisées pour un arbre, la profondeur maximale de l'arbre, et le nombre minimum d'observations par noeud terminal. D'autres paramètres peuvent également varier (comme par exemple le nombre d'arbres par forêt). L'optimisation se fait ici sur le score F1, pour pénaliser plus fortement les modèles où on prédit toujours des hommes.

```
In [105]: t_debut = time.time()

          parameters = {'max_features':("auto", "log2", 500, 1000, None),
                        'max_depth':[None, 100, 50, 30, 20, 10],
                         'min_samples_leaf':[1,2,5,10,20]}
          # On utilise comme score le score F1 pour une moyenne du rappel et de la précision.
          svc = RandomForestClassifier()
          clf = GridSearchCV(svc, parameters, cv=5, scoring = "f1_macro")
          clf.fit(feat_train, y_train)

          t_fin = time.time()
          temps_exec = t_fin - t_debut
          print("Le fitting de l'ensemble de ces modèles a duré "+repr(temps_exec)+" secondes.")
```

```
Le fitting de l'ensemble de ces modèles a duré 778.5757918357849 secondes.
```

```
In [106]: # Modèle sélectionné
          clf.estimator
```

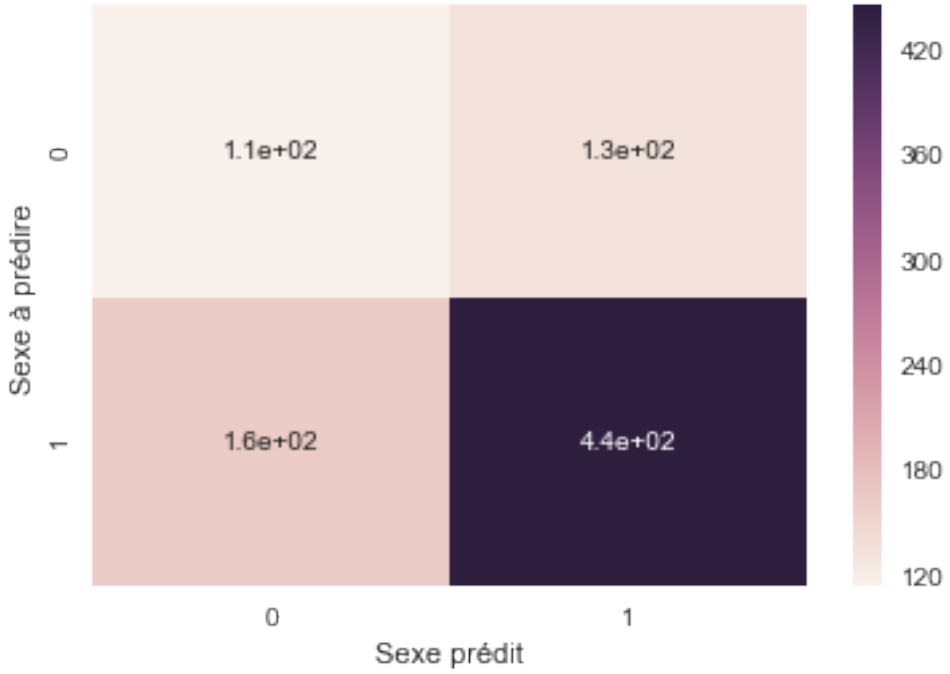
```
Out[106]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_split=1e-07, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

```
In [107]: clf.score(feat_test, y_test)
```

```
Out[107]: 0.5315298109222456
```

```
In [108]: # Matrice de confusior pour le "meilleur" modèle
          # (au sens du score F1)
          y_pred = clf.predict(feat_test)

          cnf_matrix = confusion_matrix(y_test, y_pred)

          plt.figure()
          sn.heatmap(matrice_a_tracer, annot=True)
          plt.xlabel("Sexe prédit")
          plt.ylabel("Sexe à prédire")
```

```
Out[108]: <matplotlib.text.Text at 0x1ce4e41c710>
```



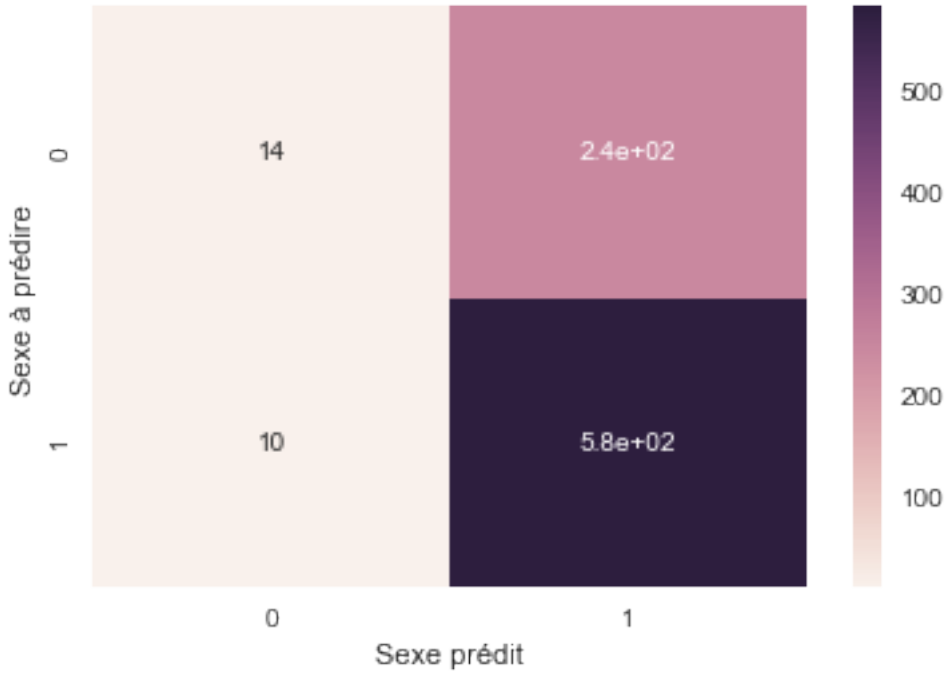
On constate ici que le modèle sélectionné n'a pas le défaut de coder trop souvent des hommes comme dans la partie précédente (optimisation opérée sur le critère du score F1). En revanche, le score du modèle mesuré par l'exactitude avec les prédictions réalisées est très mauvais (53% ici).

1.2.4 Extension : réduction de la dimensionnalité

Jusqu'ici, le nombre de variables (3-grammes) utilisées dans les modèles est très important (de l'ordre de 40000). En amont de la modélisation, il est également possible de réduire la dimensionnalité pour ces variables afin de limiter le nombre maximal de covariables. Même si l'interprétabilité des modèles résultants s'en trouve affaiblie, cette approche peut permettre d'améliorer le pouvoir prédictif des modèles. On présente ici quelques résultats après avoir effectué une Analyse en Composantes Principales pour les données. D'autres méthodes non linéaires peuvent également être utilisées (approche *Word2Vec*) mais elles n'ont pas été testées.

```
In [150]: pipe_svd = make_pipeline(CountVectorizer(ngram_range=(1,3)), TfidfTransformer(), Trunc
          pipe_svd.fit(X_train)
          feat_train_svd = pipe_svd.transform(X_train)
          feat_train_svd.shape

Out[150]: (2553, 300)

In [151]: clf_svd = RandomForestClassifier(n_estimators=50)
          clf_svd.fit(feat_train_svd, y_train)
```

```
Out[151]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                 max_depth=None, max_features='auto', max_leaf_nodes=None,
                                 min_impurity_split=1e-07, min_samples_leaf=1,
                                 min_samples_split=2, min_weight_fraction_leaf=0.0,
                                 n_estimators=50, n_jobs=1, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)

In [153]: feat_test_svd = pipe_svd.transform(X_test)
          clf_svd.score(feat_test_svd, y_test)

Out[153]: 0.700352526439483

In [159]: y_pred = clf_svd.predict(feat_test_svd)

          cnf_matrix = confusion_matrix(y_test, y_pred)


          matrice_a_tracer = pd.DataFrame(cnf_matrix)

          plt.figure()
          sn.heatmap(matrice_a_tracer, annot=True)
          plt.xlabel("Sexe prédit")
          plt.ylabel("Sexe à prédire")

Out[159]: <matplotlib.text.Text at 0x12400b66b38>
```



Pour conclure, on remarque donc qu'il est difficile de prédire le sexe des auteurs de haïkus à partir du seul texte de ces derniers. Plusieurs méthodes d'apprentissage supervisé ont été testées et donnent des résultats relativement similaires en termes d'exactitude, avec environ 70% de taux de bon codage. Toutefois, certaines méthodes ont tendance à ne prédire que des auteurs masculins, comme la régression logistique avec les paramètres choisis ici. Les méthodes fondées sur les réseaux de neurones ou la classificatio naïve bayésienne peuvent donc apparaître plus adaptées. Dans la suite, on considère le modèle XXX et on étudie plus précisément les résultats en fonctions des probabilités prédites que l'auteur soit un homme.