tp_text_mining_ipython

December 16, 2016

1 TP Text Mining, Naive Bayes Classifier

Mohamed AL ANI, le 09/12/2016

1.1 Imports

Importing packages that we will use

```
In [14]: import os.path as op
         import numpy as np
         import pandas as pd
         import vocabulary
         from scipy.sparse import csr_matrix
         from nltk.stem.snowball import SnowballStemmer
         import nltk
         import re

         #Sklearn
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.pipeline import Pipeline
         from sklearn.base import BaseEstimator, ClassifierMixin
         from sklearn.model_selection import GridSearchCV
         from sklearn.svm import LinearSVC
         from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import cross_val_score
         from sklearn.feature_extraction.text import TfidfVectorizer
```

1.2 Loading the data

```
In [15]: # Load data
         print("Loading dataset")

         from glob import glob
         filenames_neg = sorted(glob(op.join('data', 'imdb1', 'neg', '*.txt')))
         filenames_pos = sorted(glob(op.join('data', 'imdb1', 'pos', '*.txt')))
```

```
texts_neg = [open(f).read() for f in filenames_neg]
texts_pos = [open(f).read() for f in filenames_pos]

#On charge nos stop words dans une liste
stopWordsList = open("data/english.stop").read().split("\n")

texts = texts_neg + texts_pos
y = np.ones(len(texts), dtype=np.int)
y[:len(texts_neg)] = 0.
y[len(texts_neg):] = 1.

print("%d documents" % len(texts))
print("Nombre d'avis positifs : %d " %len(texts_neg))
print("Nombre d'avis négatifs : %d " %len(texts_pos))
```

```
Loading dataset
2000 documents
Nombre d'avis positifs : 1000
Nombre d'avis négatifs : 1000
```

1.3 Word count

1.3.1 Question 1

```
In [16]: #On crée une fonction qui récupère le vocabulaire, séparément de count_wor
         # à faire des calculs inutiles lorsque l'on veut que le vocabulaire et non

         def getVocabulary(texts, stopWords):
             '''getVocabulary : crée une liste de mots unique dans une liste de tex

             Parameters
             ----------
             texts : une liste de string
             stopWords : Un booléan permettant d'indiquer si l'on veut supprimer de

             Returns
             -------
             Liste de mots uniques
             '''
             words = set()
             if stopWords == True :
                 for text in texts:
                     text.replace("\n", "")
                     for item in text.split(" "):
                         if item not in stopWordsList and len(item)!=1:
                             words.add(item)
                 return words
```

```
            else :
                for text in texts:
                    text.replace("\n", "")
                    for item in text.split(" "):
                        words.add(item)
                return words

        def count_words(texts):
            """Vectorize text : return count of each word in the text snippets

            Parameters
            ----------
            texts : list of str
                The texts

            Returns
            -------
            vocabulary : dict
                A dictionary that points to an index in counts for each word.
            counts : ndarray, shape (n_samples, n_features)
                The counts of each word in each text.
                n_samples == number of documents.
                n_features == number of words in vocabulary.
            """

            #We us a csr_matrix for optimality reasons
            indptr = [0]
            indices = []
            data = []
            words = set()
            vocabulary = {}
            for text in texts:
                for term in text.split(" "):
                    index = vocabulary.setdefault(term, len(vocabulary))
                    indices.append(index)
                    data.append(1)
                    words.add(term)
                indptr.append(len(indices))
            return (csr_matrix((data, indices, indptr), dtype=np.int8).toarray(),
```

In [17]:
```
%%time
print("shape of created matrix data : ")
data, voc = count_words(texts)
print(data.shape)
```

```
shape of created matrix data :
(2000, 56199)
Wall time: 1.89 s
```

1.3.2 Question 2

based on ratings : - 5 stars system : - 3.5/5 and higher are considered positive - 2.5/5 and below are considered negative - 4 stars system : - 3/4 and higher are considered positive - 2/4 and below are considered negative - letter grade system : - B or above is considered positive - C- or below is considered negative

1.3.3 Question 3 : Implementation of NB :

```
In [18]: class NB(BaseEstimator, ClassifierMixin):
             ''' class NB : implémentation du modèle Naive Bayes Classifier

             attributs
             ------------
             stopWords (bool) : On indique si on veut un vocabulaire sans stop word
             prior : (c) np.array, probability of finding c value in y
             V : (v) list, vocabulary
             condprob : (c, v) matrix np.array, probability of finding a word for e
             dico_index_words : (String : int) dictionnary of words -> index


             methods
             -------
             fit : apprentissage
             predict : prédiction
             score : score d'accuracy du modèle
             '''


             def __init__(self, stopWords=True):
                 self.stopWords = stopWords



             def fit(self, X, y):
                 '''fit apprentissage du NB

                 Parameters
                 ----------
                 X : (n,p) matrix, numpy array
                 y : (n) list or numpy array

                 '''

                 #Getting the vocabulary
                 V = getVocabulary(texts, self.stopWords)
                 N = len(y)
                 prior = np.zeros(len(np.unique(y)))
```

```
        condprob = np.zeros((len(V), len(np.unique(y))))

        for c in np.unique(y):
            prior[c] = np.sum([y==c])/N
            textc = np.array([X])[:,y==c][0]

            #Concatening all texts
            text = " ".join(textc)

            #Word count
            textSplit = text.split(" ")
            dico = {}
            for word in textSplit:
                if word in dico:
                    dico[word] +=1
                else :
                    dico[word] = 0

            sumValues = float(np.sum(list(dico.values())))
            dico_index_words = {}

            #Computation of condprob matrix
            for i, word in enumerate(V):
                try :
                    condprob[i,c] = (dico[word]+1) / sumValues
                except :
                    condprob[i,c] = 1 / sumValues

                dico_index_words[word] = i

        self.V = V
        self.prior = prior
        self.condprob = condprob
        self.dico_index_words = dico_index_words

    def predict(self, X):
        prediction = np.zeros(len(X))

        #We iterate over all the documents in the list
        for i,doc in enumerate(X):
            V = set()
            doc = doc.replace("\n", "")

            #Creation of a dictionary for each doc
            for word in doc.split(" "):
                V.add(word)
            score = np.zeros(len(np.unique(y)))
            for c in np.unique(y):
```

```
                    score[c] = np.log(self.prior[c])
                    for word in V:
                        if word in self.dico_index_words:
                            score[c] += np.log(self.condprob[self.dico_index_w

                prediction[i] = np.argmax(score)
            return (prediction)

        def score(self, X, y):
            pred = self.predict(X)
            return np.mean( pred[:] == y[:])
```

1.3.4 Question 4 : Evaluation des performances en cross validation 5 folds :

Score brute, sans CV :

```
In [19]: %%time
         nb = NB(stopWords=False)
         nb.fit(texts[::2], y[::2])
         print("Accuracy : %0.3f" %nb.score(texts[1::2], y[1::2]))

Accuracy : 0.822
Wall time: 3.39 s
```

Avec CV = 5 :

```
In [14]: %%time
         scores = cross_val_score(nb, texts, y, cv=5)
         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

mean score and 95% conf intervalle :
Accuracy: 0.81 (+/- 0.02)
Wall time: 13.1 s
```

1.3.5 Question 5 : idem en ignorant les stop words :

```
In [15]: %%time
         nb = NB(stopWords=True)
         nb.fit(texts[::2], y[::2])
         print("Accuracy : ")
         print(nb.score(texts[1::2], y[1::2]))

Accuracy :
0.824
Wall time: 15.6 s
```

```
In [16]: %%time
         scores = cross_val_score(nb, texts, y, cv=5)
         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

mean score and 95% conf intervalle :
Accuracy: 0.83 (+/- 0.03)
Wall time: 1min 15s
```

On gagne ainsi 2% en accuracy

1.4 Text Mining and Scikitlearn

1.4.1 Question 1 :

We write a new tokenizer function to ignore useless characters, then we write a regex (inspired by the sources code) to replace all characters that don't correspond to words

```
In [6]: def my_tokenizer(s):
            prog = re.compile(r"(?u)\b\w\w+\b")
            return prog.findall(s.replace("\n", ""))
```

We use "countvectorizer" that basically will do a word count and some preprocessing on the text. We have tried with different parameters. First we try with an analysis on words with a 2-gram methods. Also we consider only words that have more than 10 occurences

```
In [92]: %%time
         MNB = MultinomialNB()

         #We use a Pipeline to sum up all the transformation :
         pipeline = Pipeline([('count', CountVectorizer(analyzer='word', min_df = 1
                                                        ngram_range=(1,2), stop_wor
                                                        tokenizer = my_tokenizer,\
                                                       )),\
                              ('MNB', MNB)])

         scores = cross_val_score(pipeline, texts, y, cv=5).mean()
         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.3f (+/- %0.4f)" % (scores.mean(), scores.std() * 2))

mean score and 95% conf intervalle :
Accuracy: 0.826 (+/- 0.0000)
Wall time: 22.5 s
```

We get a better accuracy and a better boundary confidence without even using the vocabulary.
** Now working on characters 3-7gram**

```
In [76]: %%time
         pipeline = Pipeline([('count', CountVectorizer(analyzer='char', min_df = 1
                                                        ngram_range=(3,7), stop_wor
                                                        tokenizer = my_tokenizer,\
                                                       )),\
                              ('MNB', MNB)])

         scores = cross_val_score(pipeline, texts, y, cv=5).mean()
         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.3f (+/- %0.4f)" % (scores.mean(), scores.std() * 2))
```

```
mean score and 95% conf intervalle :
Accuracy: 0.83 (+/- 0.00)
Wall time: 4min 53s
```

1.4.2 Question 2

We build a GridSearchCV on different values of C for a logistic regression and a LinearSVC

```
In [114]: %%time
          svc = LinearSVC()
          pipeline = Pipeline([('count', CountVectorizer(analyzer='word', min_df =
                                                         ngram_range=(1,2), stop_wo
                                                         tokenizer = my_tokenizer,\
                                                        )),\
                               ('clf', svc)])

          params = dict(clf=[LinearSVC(), LogisticRegression()],
                        clf__C=[0.0001, 0.001, 0.01])

          clf = GridSearchCV(pipeline, param_grid = params, cv=5)
          clf.fit(texts, y)
          print("best score : %0.3f" %clf.best_score_)
          print("best params : ")
          print(clf.best_params_ )
```

```
best score : 0.844
best param :
{'clf': LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0), 'clf__C': 0.001}
cv results
Wall time: 3min 10s
```

Here we got a better CV score and we print out the best model

We can try with TfidfVectorizer that will basically transform the wordcount into frequences of words in the text :

```
In [12]: %%time
         svc = LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

         pipeline = Pipeline([('count', TfidfVectorizer(analyzer='word', min_df = 1
                                                        ngram_range=(1,2), stop_wor
                                                        tokenizer = my_tokenizer,\
                                                       )),\
                              ('clf', svc)])



         scores = cross_val_score(pipeline, texts, y, cv=5).mean()

         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.3f (+/- %0.4f)" % (scores.mean(), scores.std() * 2))

mean score and 95% conf intervalle :
Accuracy: 0.856 (+/- 0.0000)
Wall time: 22.5 s
```

1.4.3 Question 3 + 4

Example

To understand how pos_tag and stemmer works, we tried on a small example :

```
In [72]: a = "Although when He was has liked liking big apples prettier pretty bigg
         print("old text : ")
         print(a)
         print()
         print("tags we get : ")
         print(nltk.pos_tag(a))
         print("\n new text : ")
         " ".join([stemmer.stem(word[0]) for word in nltk.pos_tag(a) if word[1] in

old text :
['Although', 'when', 'He', 'was', 'has', 'liked', 'liking', 'big', 'apples', 'prett

tags we get :
[('Although', 'IN'), ('when', 'WRB'), ('He', 'PRP'), ('was', 'VBD'), ('has', 'VBZ')

 new text :


Out[72]: 'was has like like big appl prettier biggest banana'
```

We build a list of pos_tags to keep and transform our old texts into new texts with the stemmer :

```
In [13]: %%time
         # nltk.help.upenn_tagset() to get list of different tags

         to_keep = ["NN", "NNP", "NNPS", "NNS", "VBG", "VBN", "VBD", "VBP", \
                    "VBZ", "VB", "JJ", 'JJR', 'JJS']

         print("****************")
         print("old text")
         print("****************")
         print(texts[1])

         stemmer = SnowballStemmer("english")
         newTexts = []
         for text in texts[:2]:
             textSplit = text.replace("\n", "").split()
             temp = " ".join([stemmer.stem(word[0]) for word in nltk.pos_tag(textSp
             newTexts.append(temp)
         print("****************")
         print("new text")
         print("****************")
         print(newTexts[1])

****************
old text
****************
the happy bastard's quick movie review
damn that y2k bug .
it's got a head start in this movie starring jamie lee curtis and another baldwin b
little do they know the power within . . .
going for the gore and bringing on a few action sequences here and there , virus st
we don't know why the crew was really out in the middle of nowhere , we don't know
here , it's just " hey , let's chase these people around with some robots " .
the acting is below average , even from the likes of curtis .
you're more likely to get a kick out of her work in halloween h20 .
sutherland is wasted and baldwin , well , he's acting like a baldwin , of course .
the real star here are stan winston's robot design , some schnazzy cgi , and the oc
so , if robots and body parts really turn you on , here's your movie .
otherwise , it's pretty much a sunken ship of a movie .

****************
new text
****************
happi bastard quick movi review damn y2k bug it got head start movi star jami lee c
Wall time: 582 ms
```

Test d'un modèle sur les nouveaux textes We create a new tokenizer with the stemmer and the pos_tag

```
In [70]: def my_new_tokenizer(s):
             s = s.replace("\n", "")
             temp = " ".join([stemmer.stem(word[0]) for word in nltk.pos_tag(s.spli
             prog = re.compile(r"(?u)\b\w\w+\b")
             return prog.findall(temp)
```

We try one last model of LinearSVC with the best model found previously and our new tokenizer :

```
In [71]: %%time
         lsvc = LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

         pipeline = Pipeline([('count', CountVectorizer(analyzer='word', min_df = 5
                                                        ngram_range=(1,2), stop_wor
                                                        tokenizer = my_new_tokenize
                                                       )),\
                              ('lsvc', lsvc)])

         scores = cross_val_score(pipeline, texts, y, cv=5).mean()

         print("mean score and 95% conf intervalle : ")
         print("Accuracy: %0.3f (+/- %0.4f)" % (scores.mean(), scores.std() * 2))

mean score and 95% conf intervalle :
Accuracy: 0.840 (+/- 0.0000)
Wall time: 9min 53s
```

Finally, we don't get a better score... Maybe we should recreate a GridSearchCV for the new data we get out of the tokenizer function. However, the fitting took almost 10 minutes so we won't do that here. Especially knowing that we already have quite a good score.