Traitement automatique du langage
TP 2 — Classification with Naive Bayes
Solutions

Yves Scherrer

Code

```
#! /usr/bin/python3
# -*- coding: utf-8 -*-

import os, math, sys, re

# Variables globales

# les deux classes à traiter
classes = (”pos”, ”neg”)

# contient tous les mots et leurs fréquences, tous les groupes
   confondus
wordFrequencies = {}

# contient tous les mots, distingués selon le groupe
wordFrequenciesPerClass = {}

# contient le nombre de mots par groupe
totalWordsPerClass = {}

# contient la matrice de confusion des résultats
confusionMatrix = {}

#---------------------------------------

def tokenizeMessage(l):
    # expression régulière de http://www.nltk.org/_modules/nltk/
       tokenize/regexp.html
    tokens = re.findall(r’\w+|[^\w\s]+’, l)
    return tokens
```

```
#--------------------------------------

def loadStopwords(language):
    f = open("stopwords_{}.txt".format(language), 'r', encoding="utf
       -8")
    stopwords = [x.strip() for x in f if not x.startswith("#")]
    return stopwords

#--------------------------------------

# lit tous les documents, les transforme en liste de mots et peuple
   les vocabulaires destinées à l'entraînement
def training(language, tokenize, lowercase, removeStopwords):
    if removeStopwords:
        stopwords = loadStopwords(language)

    for c in classes:
        wordFrequenciesPerClass[c] = {}
        totalWordsPerClass[c] = 0
        nbMsg = 0

        for msg in os.listdir(language + "/train/" + c):
            nbMsg += 1
            f = open(language + "/train/" + c + "/" + msg, encoding=
               "utf-8")
            if tokenize:
                tokens = tokenizeMessage(f.read())
            else:
                tokens = f.read().split()

            for token in tokens:
                if lowercase:
                    token = token.lower()
                if removeStopwords and token in stopwords:
                    continue
                wordFrequencies.setdefault(token, 0)
                wordFrequencies[token] += 1
                # incrémenter la fréquence du mot dans le
                   vocabulaire du groupe courant
                wordFrequenciesPerClass.setdefault(c, {})
                wordFrequenciesPerClass[c].setdefault(token, 0)
                wordFrequenciesPerClass[c][token] += 1
                totalWordsPerClass[c] += 1
            f.close()
        print("Entraînement effectué pour {0} ({1} fichiers, {2}
           mots)".format(c, nbMsg, totalWordsPerClass[c]))
```

```
#--------------------------------------

def testing(language, tokenize, lowercase, countFeatures, smoothing)
   :
    for trueClass in classes:
        confusionMatrix[trueClass] = {}
        # calculer le nombre total de mots distincts
        vocabularySize = len(wordFrequencies)
        for msg in os.listdir(language + "/test/" + trueClass):
            f = open(language + "/test/" + trueClass + "/" + msg,
               encoding="utf-8")
            if tokenize:
                tokens = tokenizeMessage(f.read())
            else:
                tokens = f.read().split()
            if lowercase:
                tokens = [token.lower() for token in tokens]
            if countFeatures == False:
                tokens = set(tokens)

            results = {}
            for predictedClass in classes:
                # déterminer la probabilité d'appartenance a priori
                   au groupe
                predictedProb = math.log(0.5)

                # pour chaque mot distinct qui apparait dans le
                   fichier test, on rajoute le log de la probabilité
                    d'appartenance au groupe donné
                for t in tokens:
                    # pas besoin de tester explicitement l'
                       appartenance aux stopwords, ce ne sera juste
                       pas dans la liste
                    if smoothing:
                        if t in wordFrequencies:
                            predictedProb += math.log((
                               wordFrequenciesPerClass[
                               predictedClass].get(t, 0) + 1) / (
                               totalWordsPerClass[predictedClass] +
                               vocabularySize))
                    else:
                        # si t apparaît dans wordFrequenciesPerClass
                           [pos] ainsi que dans
                           wordFrequenciesPerClass[neg]
                        if all([t in wordFrequenciesPerClass[pc] for
                            pc in classes]):
```

```
                            predictedProb += math.log(
                               wordFrequenciesPerClass[
                               predictedClass][t] /
                               totalWordsPerClass[predictedClass])
                    results[predictedClass] = predictedProb

            maxClass = max(results, key=lambda x: results[x])
            maxProb = results[maxClass]
            #print("{0}: vrai={1}, prédit={2}, logP={3:0.5f}".format
               (msg, trueClass, maxClass, maxProb))

            # ajouter le résultat à la matrice de confusion pour l'
               évaluation
            confusionMatrix[trueClass].setdefault(maxClass, 0)
            confusionMatrix[trueClass][maxClass] += 1
            f.close()

#---------------------------------------

def evaluation():
    print("\t".join(["", "pos", "neg", "Recall"]))
    rpos = confusionMatrix["pos"].get("pos", 0) / sum(
       confusionMatrix["pos"].values())
    rneg = confusionMatrix["neg"].get("neg", 0) / sum(
       confusionMatrix["neg"].values())
    print("\t".join(["pos", "{}".format(confusionMatrix["pos"].get("
       pos", 0)), "{}".format(confusionMatrix["pos"].get("neg", 0)),
        "{0:.2f}%".format(100*rpos)]))
    print("\t".join(["neg", "{}".format(confusionMatrix["neg"].get("
       pos", 0)), "{}".format(confusionMatrix["neg"].get("neg", 0)),
        "{0:.2f}%".format(100*rneg)]))
    ppos = confusionMatrix["pos"].get("pos", 0) / sum([
       confusionMatrix[x].get("pos", 0) for x in confusionMatrix])
    pneg = confusionMatrix["neg"].get("neg", 0) / sum([
       confusionMatrix[x].get("neg", 0) for x in confusionMatrix])
    print("\t".join(["Prec.", "{0:.2f}%".format(100*ppos), "{0:.2f}%
       ".format(100*pneg), ""]))
    print()
    acc = (confusionMatrix["pos"].get("pos", 0) + confusionMatrix["
       neg"].get("neg", 0)) / sum([sum(confusionMatrix[x].values())
       for x in confusionMatrix])
    print("Accuracy: {0:.2f}%".format(100*acc))


#---------------------------------------

if __name__ == "__main__":
```

```
    lg = sys.argv[1]
    # les features peuvent être ajoutés à la commande dans n'importe
        quel ordre
    # les variables ci-dessous sont booléennes
    tokenize = ("tokenize" in sys.argv[2:])
    lowercase = ("lowercase" in sys.argv[2:])
    removeStopwords = ("removeStopwords" in sys.argv[2:])
    countFeatures = ("countFeatures" in sys.argv[2:])
    smoothing = ("smoothing" in sys.argv[2:])
    training(lg, tokenize, lowercase, removeStopwords)
    testing(lg,tokenize, lowercase, countFeatures, smoothing)
    evaluation()
```

Results

Experiment 1

It is not clearly specified how one should proceed with words that have seen in one class but not in the other. Two variants are possible:

- Skip the word only in the class it has not been seen, but keep it in the other class:
  47.5% accuracy for English and 39.0% for French,
- Skip the word if it has not been seen in at least one class:
  83.5% accuracy for English and 68.5% for French.

Experiment 2

We apply a simple language-independent tokenization step using a regular expression:

- `'\w+|[^\w\s]+'`

We use stopword lists from the following sources:

- English: `http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words`
- French: `http://www.ranks.nl/stopwords/french`

The English messages are already lowercased, so no additional improvement is expected there.

Results (accuracies) are summed up in the following table:

System	English	French
Basic	83.5%	68.5%
Tokenization	86.5%	72.0%
Remove stopwords	84.0%	69.0%
Lowercase	83.5%	70.0%
T+S+L	85.5%	71.0%
T+S+L+Count features	83.0%	71.0%
T+S+L+Smoothing	84.5%	71.5%