Report on Programming Assignment Naive Bayesian Text Classification

Amy de Lange - s1534041

November 2019

Introduction

The goal of the assignment is to program a classifier that can distinguish emails in the categories 'spam' and 'ham' (not-spam). The tools that were provided the assignment decription manual and a corpus of emails to train and test with. I installed Python 2 and Atom (code editor) myself and followed an online course to retrieve the basic knowledge about the language of Python.

The purpose of this assignment is to learn the basic concepts of programming and machine learning. This will be done by building a Naive Bayesian Text Classifier, that implements a machine learning method for classifying texts based on the statistics about the occurrences of words in texts of the different categories. The first step would be to prepare the corpus into readable tokens where the classifier can be trained with. In the spam classifier, the occurrence of specific words in a text will be leading in classifying the text into one of the two classes. The specific words will be labelled in the training part where words will be counted in both classes to calculate later on their probability of being either spam or ham.

To improve the classifier, chi-square calculations will be first done on all the unique words of all the emails combined. These will generate the most distinguishing words from the corpus. This will be provided to the classifier to look for these specific words to calculate, at the end, the probability of a certain email for being spam or ham.

Structure of the model

The model consists out of 6 parts. They will be all called upon in the main module. The parts are: Preparing the corpus for usage (1), creating two dictionaries with all the unique words from every class email set (2), counting word occurrences in the corpus per classified email (3), building a chi-square list (4), calculating word probabilities (5) and finally testing the the classifier by classifying the emails from the test corpus (6).

Reading, Normalizing, Tokenizing

The corpus received for this assignment consist out of 10 parts, the first 9 for training the classifier and part 10 to test the classifier. The corpus consisted out of spam and ham emails. They could be distinguished by their filenames; spam consisted out of 'spms' and ham consisted out of 'msg'. To read the corpus, I made use of the module 'glob' to import files from my laptop directory. The categorize the right files to spamfiles and hamfiles, I made use of regular expressions to add upon the directory 'dir' that matches the filename characteristics of ham and spam.

By tokenizing the words in every email and normalizing them, we will get a dictionary of only lowercase words. The dictionaries of spam and ham will be filled up by a for-in loops their all files of that category.

```
"""READING, NORMALIZING, TOKENIZING"""
def reader(dir):
    ham_files = glob.glob(dir+"/*msg?.txt")
    spam_files = glob.glob(dir+"/spms*.txt")

    ham_dictionary = {file: tokenizer(open(file).read()) for file
    in ham_files} # make a dictionary of all ham files
    ham_tokens = ham_dictionary.values()
    for ham_token in ham_tokens:
        for e in ham_token:
            ham_data.append(e)

    spam_dictionary = {file: tokenizer(open(file).read()) for file
    in spam_files} # make a dictionary of all spam files
    spam_tokens = spam_dictionary.values()
    for spam_token in spam_tokens:
        for e in spam_token:
            spam_data.append(e)

    return ham_data, spam_data

def normalise(word):
    return re.sub(r'\W+', '', word.lower())

def tokenizer(str):
    tokens = [normalise(word) for word in str.split(" ") if
    normalise(word).isalpha()]
    allTokens = tokens
    return allTokens
```

Listing 1: Making dictionaries of all appearing words for each category

Unique dictionaries

Later on in the assignment, we would like to retrieve a chi-square word list to train and test the classifier with. To generate that list, we would like to loop through a list of all unique words that occur in the corpus. Therefore, we make unique words dictionaries for each category. This will be done by an easy if statement to get a dictionary with the keys as words and the values as the times of occurrences in that class (ham/spam).

```
"""UNIQUE WORDS"""

def UniqueWordsMaker():
    print "=== spam_data: " + str(len(spam_data)) + " ==="
    for word in spam_data:
        if word not in spam_data_pr.keys():
            spam_data_pr.update({word:1})

        if word in spam_data_pr.keys():
            value = spam_data_pr.get(word) + 1
            spam_data_pr.update({word: value})

    print "=== ham_data: " + str(len(ham_data)) + " ==="

    for word in ham_data:
        if word not in ham_data_pr.keys():
            ham_data_pr.update({word:1})

        if word in ham_data_pr.keys():
            value = ham_data_pr.get(word) + 1
            ham_data_pr.update({word: value})
```

Listing 2: Making dictionary with unique words

Chi-square calculations

For the classifier to better distinguish between two classes, specific words are more distinguishing than others. For example, the word "a" will be probably used in both classes because most sentences are made up with that word without any specific meaning attached. To know which words are a good base to train the model with, we made use of the chi-square test.

To structure the calculations, I made use of pandas which generates a table with columns and rows. The retrieved value will be saved in the "chi-square" columns. The data is saves as "data" and will be sorted ascending to view the most distinguishing words on top of the list.

```
"""BUILDING CHI SQUARE LIST"""

def chi_square():
    print "chi square calculations starting..."
    #data = np.array([ham_data_pr.keys(), "h",0])
    spam_data_pd = pd.DataFrame(columns=["data", "n_words", '
    chi_score', "label" ])
    spam_data_pd["data"] = spam_data_pr.keys()
    spam_data_pd["label"] = "s"
    spam_data_pd["chi_score"] = 0
    spam_data_pd["n_words"] = spam_data_pr.values()

    ham_data_pd = pd.DataFrame(columns=["data", "n_words", '
    chi_score', "label" ])
    ham_data_pd["data"] = ham_data_pr.keys()
    ham_data_pd["label"] = "h"
    ham_data_pd["chi_score"] = 0
    ham_data_pd["n_words"] = ham_data_pr.values()
    data = spam_data_pd.append(ham_data_pd)

    n_ham_data = len(data[data['label']=="h"])
    n_spam_data = len(data[data['label']=="s"])
    data['1_1'] = np.where(data["data"].isin(data[data['label']=="h
    "]["data"]),data["n_words"],0)
    data['1_2'] = np.where(data["data"].isin(data[data['label']=="s
    "]["data"]),data["n_words"],0)
    data['2_1'] = n_ham_data - data['1_1']
    data['2_2'] = n_ham_data - data['1_2']
    data['C1'] = data['1_1'] + data['2_1']
    data['C2'] = data['1_2'] + data['2_2']
    data['W1'] = data['1_1'] + data['1_2']
    data['W2'] = data['2_1'] + data['2_2']
    data['N'] = data['C1'] + data['C2']
    data['E1_1'] = (data['W1'] * data['C1']) / data['N']
    data['E1_2'] = (data['W1'] * data['C2']) / data['N']
    data['E2_1'] = (data['W2'] * data['C1']) / data['N']
    data['E2_2'] = (data['W2'] * data['C2']) / data['N']

    cols = ['1_1','1_2','2_1','2_2']
    cols2 = ['E1_1','E1_2','E2_1','E2_2']
    data['chi_score'] = sum(((data[c] - data[e])**2)/data[e] for c
    in cols for e in cols2)

```

```
    data = data.sort_values(by=["chi_score"], ascending=True)

    data = data.reset_index()
    return data
```

Listing 3: Chi-square calculations

Discussion

During meeting with Randy, we agreed to finalize this assignment by this report. We had several meetings where Randy and I worked on bugs that appeared in the code. Those meetings were really helpful and taught me a lot about coding and how to build a programming structure.

I came quite far in the assignment. However still there is a part missing in the final part of this assignment. I struggled quite a lot with the probability part of the assignment (part 5 and 6). First couldn't fix the counter for the words to be able to calculate the word probabilities. After that, I already planned that the word probabilities would be saved into a list that would be called upon in the probability calculations of the emails. The classification would than be done by simply looking at which probability is higher, spam or ham. To be able to conclude things upon the accuracy of the classifier, I would compare the filename of the testing emails to the outcome of the classification. This would give me a final percentage of the accuracy. By using different sizes of chi-square lists, I would have expected different outcomes of the accuracy.

Listings