Bayesian-Filter for Ticket-Analysis

Contents

Introduction

At my company we have our own helpdesk, which helps our colleges whenever they encounter a technical problem they cannot solve themselves. In order to organize the workflow of helping our colleges we use a “Ticket-Management-System” called **Cherwell.** Whenever a customer (synonym for college in this context) has a problem, thy can either open up a ticket themselves and describe their problem in it and send it to the helpdesk team or thy can call the Hotline (Helpdesk) and they will open up a ticket for them. Either way all of our tickets are received by the First Level of Support, which brings us to the next topic, the helpdesk-structure. The helpdesk has 3 different kinds of levels, the First Level, which deals with “normal” issues and request such as, “I forgot my password”, Software-Installation or help and advice. The Second and Third Level deal with more difficult and time consuming issues, such as: bug fixes, change requests, hardware installation/maintenance and so on. In this case we will focus on the First Level, because this where we aim to make some improvements.

Currently we raise an evaluation of how many tickets have been solved on which Level each Month and will try to determine if it was a “good” or a “bad” month. The especially observant reader will notice, that this percentage of how many tickets of were solved in the First Level of Support is a very misleading number, because a rise or drop of this percentage can have a lot of reasons that have nothing to do with the quality of our First Level. For Example it could have just been that there were a lot of Bug Fixes to do which the First Level is not even supposed to handle and when the amount of “normal” issues drops at the same time (for instance, because a lot of people are on vacation) the percentage of **Tickets solved on First Level** is going to drop dramatically and you could think it is because this Team is lazy.

Because of this we are now trying to determine two different percentages:

- How many tickets **were solved** on First Level (same on as mentioned above)
- How many tickets **could have been solved** on the First Level (here it starts to get tricky)

In order to determine this second percentage we will use a statistical approach:

The Bayesian Filter Model.

I got the Inspiration from guy on Git called Browning and basically copied his algorithm, so please check him out: https://github.com/browning/comment-troll-classifier

He uses an Implementation of Paul Graham's Spam Filter which is very detailed and even more worth checking out: http://www.paulgraham.com/spam.html

What is a Bayesian Filter

A Bayesian Filter is basically a text classification algorithm, you train it witch **A LOT OF DATA,** for example witch 100.000 tickets that could be solved at the first level of support and 100.000 ones that could not. The Filter will then separate the text in so called tokens (for me I just consider words as tokens, but there are many approaches here…) and then it will calculate the **probability** of this token (so how probable is it, that this word appears in a Ticket that **could not be solved** on the First Level). The more training data you have the better, the Filter will get better the more data it gets. But here starts already one of the main problems: getting the right **Training Data** (later more on that).

How to use

To train the Filter create two .CSW-Files one with data of tickets that **could be solved** at First Level and one witch data of tickets that **could not be solved** at First Level and run the python file build_hashes.py (the program will prompt you to input the file Locations of those two .CSV-Files. The program will give you a little bit of information on how many probabilities (that is tokens, with a probability of not being solved on First Level) it calculated and a bit of other Information. The .JSON-File probabilities_dict.json is the place where all these probabilities are stored and the files positives_dict.csv and negatives_dict.csv is where the total amount of tokens of each corpus is stored. I suggest you do not edit these files, because it will only make the filter worse. The files probabilities.json, positives_dict.csv and negatives_dict.csv will be created anew with each run of the program build_hashes.py, so that the files are always built up clean. That means also that if you want to expand your Filter you need to add the new training data to the old and run the program again. Furthermore I recommend not putting in any blank lines in the training data files ticket_positives.csv and ticket_negatives.csv, as well as not blank line at the top or bottom of the file, the first line is also ignored every time because you will probably want a header for your data.

After you finished training the Filter you can start evaluating tickets. Just create a .CSV-File with the tickets you want to evaluate. Please keep in mind, that the file should not contain any blank lines at the beginning, the middle or end. In addition each line should contain **exactly** one ticket and nothing more, put the ticket number at the beginning and put a token separator (like ",”/”;”/”\t” and so on) between the ticket-number and the rest of the information (if you download a bunch of tickets directly from Cherwell it will already contain separators like tabstops, semicolons or commas -> I recommend that you only use the three separators mentioned above, because these are the only ones I tested. Then you just run the python file check_tickets.py, which will prompt you to enter the file path of your tickets, and you

will receive a short information about what happened and how many tickets have a probability of **not being solved** on the First Level, your Results are then printed in the **results.csv** file, for further use.

The Algorithm

I wrote this Filter in Paython 3.62 and used .CSV- and .JSON-Files for the Input and storing of the probabilities. Furthermore I use the libraries pdb, sys, json, csv, re, time, datetime and os. In addition to them I wrote two more libraries: progress_bar (shows you the progress of the current script) and Chinese_japanese_chars (finds Chinese or Japanese tokens in chars and splits them).

build_hashes.py

At the start of the program I Initialize a bunch of variables and build up the most important Regular Expressions.

Then I start building the dictionaries (I use the same method for building those two dictionaries (build_dict())), each one at a time, first the negative_dict.json. I loop over the whole corpus of training data and initialize the variable “more_tokens” this is very important for the splitting of Chinese/Japanese characters (later more on that). Then I remove all the characters that are numbers special signs and so on. Then I split the ticket into its tokens and remove every empty token via the filter command.

```
#build_dictionaries
def build_dict(filename):
    """
    pass this method training data and it will create dictionaries, one for the ammount of each negative token,
    one for the amount of each positve token and on probability dictionary which states the probability of each token beeing positive( positive = over 0.5)

    Posiive means that the ticket/token has been solved on the first level of support in this cenario

    Parameters:
    filename    - Requiered : name of the csv-file with the training data (string)
    """
    dictionary = {}
    regex_noword = re.compile(r"\W", re.IGNORECASE) #everything that is not a word
    regex_digit = re.compile(r"\d", re.IGNORECASE)  #all numbers

    #start processing training data
    csv_file = open(filename, 'r', encoding='utf-8', errors='ignore')
    ticketreader = csv.reader(csv_file, delimiter=',', quotechar='|')
    tickets = list(ticketreader)

    #progress
    l = len(tickets)
    i = 0
    progress_bar.printProgressBar(0, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
    for i, ticket in enumerate(tickets):
        progress_bar.printProgressBar(iteration = i + 1, total = l, prefix = 'Progress:', suffix = 'Complete', length = 50)

        more_tokens = []

        # deleting all the unwanted characters
        line = str(ticket)
        line = line.replace('\\ufeff', ' ') #this is a so called byte order mark (BOM, not relevant anymore)
        line = line.replace(';', ' ')
        line = line.replace(',', ' ')
        line = line.replace('_', ' ')
        line = line.replace('\\n', ' ')
        line = line.replace('\\xa0', ' ')
        line = line.replace('\\t', ' ')
        line = line.replace('\\r', ' ')
        line = line.replace('\\', ' ')
        line = re.sub(regex_noword, ' ', line)
        line = re.sub(regex_digit, ' ', line)

        tokens = line.split()
        tokens = list(filter(None, tokens))
```

After that I check for Chinese/Japanese characters, because these need special attention. For that I use the method **check_for_cj_chars()** from my **chinese_japanese_chars** module. Most of the time Chinese/Japanese characters are written without whitespaces between them, so my program will recognize a whole chunk of Chinese/Japanese information as one token and this token will probably never appear again exactly like that so it is not useful to us, in fact it could jeopardize our evaluation because it will assign a 0.9 or 0.1 probability to this token which will make the ticket very hard to evalutate. Furthermore these “strings” of Chinese/Japanese characters often contain some English tokens as well (like: “申請VPN帳密”), so we need to separate the Chinese/Japanese tokens and keep

the English ones intact. So basically I check if a token contains a Chinese character with the RegEx we created at the beginning and if it does I will iterate through it and check for each character weather or not it is Chinese/Japanese . If it is I append it to the total list of tokens and replace the Chinese/Japanese character I just find in the token I am iterating through with a whitespace.This means at the End of this loop we have just the English tokens left (if there are any) and there are separated by n whitespaces. So we just split the remaining English tokens at each whitespace and append the result to or total list of tokens. Then we just need to remove the original token (the one we iterated through) from our token list. I know it sounds very complex but is actually quite simple:

```
# check for chinese/japanese tokens
for t in tokens:
    if re.search(regex_chinese, t) or re.search(regex_japanese,t):
        token_string = t
        for x in t:
            if re.search(regex_chinese, x) or re.search(regex_japanese,t):
              tokens.append(x)
              token_string = token_string.replace(x,' ')

        more_tokens = token_string.split()

        for mt in more_tokens:
            tokens.append(mt)

        tokens.remove(t)
```

Now all that is left to do is counting the tokens, like this:

```
for token in tokens:
    token = token.lower()
    if token in positive_tokens:
        positive_tokens[token] = positive_tokens[token] + 1
    else:
        positive_tokens[token] = 1
```

Then we do the same for the for the ticket_positives.csv file, so that we end up with two dictionaries of amounts of tokens. In order to calculate the probabilities we loop over the negative tokens, get the number that is stored there and if there is the same token in the positive tokens we get this number aswell (otherwise we set it to 0). Then we just divide the negative number by the sum of the negative and positive number to get our probability. Notice how I double the amount of the positive number, this is to better distinguish between tokens that occasionally occur in positive tickets and the ones that almost never do (again, there is probably room for tuning here). If the probability equals 1 then we set it to 0.9, because there can never really be a 100% chance for this token (there is probably room for tuning here). After we iterated through the whole threshold of negative tokens we iterate through the threshold of positive to catch those that weren't in the list of negative tokens and set them to 0.1 (again,

because there can never really be a 0% chance).

```
#calculate porbabilities
for k in negative_tokens:
    num_in_negatives = negative_tokens[k]
    if k in positive_tokens:
        num_in_positives = positive_tokens[k]
    else:
        num_in_positives = 0

    p = ( num_in_negatives) / ( (num_in_negatives ) + (num_in_positives * 2  ))
    if p == 1:
        p = 0.9
        probabilities[k] = p
    else:
        probabilities[k] = p


for k in positive_tokens:
  if not (k in negative_tokens):
    probabilities[k] = 0.1
```

Then we will save our results in the corresponding files and at the end there is a lot of counting the tokens and probabilities and tokens to give the user a quick overview.

check_tickets.py

Before you start the program you need to enter the tickets you want to check in the **tickets-.CSV-File** like I mentioned in the section “How to use”. First the probability dictionary and the tickets are loaded into the program and the tickets are prepared for analyzation. That means I split the ticket number from the rest of the ticket body and put them into a list, then all the special characters are removed so that all that is left is just the plain text.

```
#get tickets / prepare tickets
print("preparing tickets...")
with open('test.csv', newline='', encoding='utf8') as csvfile:
  ticketreader = csv.reader(csvfile, delimiter=',', quotechar='|')
  for row in ticketreader:
    ticket = str(row)
    ticket_split = ticket.split(',', 1)
    #get ticket number
    ticket_no = ticket_split[0]
    ticket_no = ticket_no.replace('\\ufeff', '') #this is a so called byte order mark (BOM, not relevant anymore)
    ticket_no = re.sub(regex_justdigit, '', ticket_no)

    #get ticket body without ticket number
    ticket_body = ticket_split[1]

    ticket_body = ticket_body.replace(';', ' ')
    ticket_body = ticket_body.replace(',', ' ')
    ticket_body = ticket_body.replace('_', ' ')
    ticket_dody = ticket_body.replace('\\ufeff', '') #this is a so called byte order mark (BOM, not relevant anymore)
    ticket_body = ticket_body.replace('\\n', ' ')
    ticket_body = ticket_body.replace('\\xa0', ' ')
    ticket_body = ticket_body.replace('\xa0', ' ')
    ticket_body = ticket_body.replace('\\t', ' ')
    ticket_body = ticket_body.replace('\\r', ' ')
    ticket_body = ticket_body.replace('\\', ' ')
    ticket_body= re.sub(regex_noword, ' ', ticket_body)
    ticket_body = re.sub(regex_nodigit, ' ', ticket_body)
    tickets[ticket_no] = ticket_body

print("finished preparing tickets...")
```

Then we iterate over our corpus of tickets and then again spilt the ticket into tokens. After that we need to check for Chinese tokens again, like I explained earlier.

```
# calculation of the probabilites
print("calculating probabilities...")
#progress
l = len(tickets)
i = 0
printProgressBar(0, l, prefix = 'Progress:', suffix = 'Complete', length = 50)
for i, ticket in enumerate(tickets):
  printProgressBar(iteration = i + 1, total = l, prefix = 'Progress:', suffix = 'Complete', length = 50)

  more_tokens = [] # additional tokens if the current token is a string of chinese/japanese characters
  token_probs = {} # array of the probabilities which occour in the current ticket

  tokens = tickets[ticket].split()

# check for chinese/japanese tokens
  pdb.set_trace()
  for t in tokens:
    if re.search(regex_chinese, t) or re.search(regex_japanese,t):
      token_string = t
      for x in t:
        if re.search(regex_chinese, x) or re.search(regex_japanese,t):
          tokens.append(x)
          token_string = token_string.replace(x,' ')

      more_tokens = token_string.split()

      for mt in more_tokens:
        tokens.append(mt)

      tokens.remove(t)
```

Once we have separated all the tokens we can begin calculating the probability of the current ticket. We do that by selecting the corresponding probabilities from our **probabilities_dict.json** (that we created with in the first step: **build_hashes.py**). If we don't have a probability for one token we set it to 0.4, because unfirmiliar tokens tend to be rather innocent. Then we get the 15 most interesting tokens, where interesting is measured by how far their probability is from a neutral (0.5).

```
for token in tokens:
  token = token.lower()
  if token in probabilities:
    token_probs[token] = probabilities[token]
  else:
    unknown_probs = unknown_probs + 1
    token_probs[token] = 0.4


# sort token probabilities by distance from .5
# and pull out the top X ones to use to calculate total probability
max_tokens = 15
interesting_tokens = []
c=0
for w in sorted(token_probs, key=sort_func, reverse=True):
  interesting_tokens.append(token_probs[w])
  if c >= max_tokens:
    break
  c = c+1
```

Calculating the Probabilities is rather easy as you will notice. At the End there is just a bit of error handling in case there is going to be a division by zero and some counting of probabilities for quick evaluation purposes for the user.

```
# calculate real probabilitiy
a = 1
b = 1
for token in interesting_tokens:
  a = a * token
  b = b * ( 1 - token )

if a + b == 0:
  result[ticket_no] = 'no probabilites determined'
  no_prob = no_prob + 1
else:
  final_prob = a / ( a + b )
  result[ticket] = final_prob
  total_prob = total_prob + 1
  if final_prob == 0.1:
    final_prob_0 = final_prob_0 + 1
  if final_prob == 0.9:
    final_prob = final_prob_1 + 1
  if final_prob == 0.5:
    final_prob_5 = final_prob_5 + 1
  if final_prob > 0.5:
    final_prob_gt = final_prob_gt + 1
  if final_prob < 0.5:
    final_prob_lt = final_prob_lt + 1
```

Then the results are saved and some Information is printed for the user and that's it.

Training Data

The most important thing about this method is the training data. If this is chosen poorly your Filter will not work properly at all. This is also the most difficult part because we don't really have a criteria on which we can decide if a ticket actually could have been solved on the First Level. Due to that it is quite hard to find appropriate training data, because I am trying to find data das this filter should find for me, if that makes any sense. Another rule of thumb is that you cannot have enough data so don't hesitate to use all the data that's there even if it seems pointless.

Furthermore it seems to be important that both of the corpuses are about the same size. If not the dictionary of one of them gets "to advanced" and falsifies the probabilities.

Ideas for Improvement

I want the algorithm to improve itself with each time you execute it. But in order to do that I would need some to check if the algorithm was right, and that's the whole point of what I am doing here so I feel like chasing my own tail the whole time. The other thing would be that we gather better training data, so for example we could get some guys and classify a couple of thousand tickets, but I fear that won't be enough…..

Another idea of mine was, to leave certain fields intact and don't split them. But that just resulted in decreasing the performance and it didn't make a difference at all….

The next Improvement I am going to try out is to incorporate tuples. By that I mean a second probability dictionary where all the tokens that are written next to each other stand. I think this will make the Filter "smarter" and it can recognize patterns in writing. Furthermore it is probably especially useful for Chinese/Japanese characters, since they change their meaning when written in a certain order.