Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Project report TNM108 – Swedish FAQ Chatbot

Hi and welcome to our blog for a project in the course *TNM108:Machine Learning for Social Media.*

Today we are going to explore the question:

“Can a chatbot fulfill the same role of a FAQ section on a website?”

Now the easiest way to find an answer to this, is to try it out ourselves. So let’s go! Time to make a FAQ chatbot!

The first thing we need to do is to create a dataset of frequently asked questions and answers. We chose to create a FAQ dataset based on the FAQ on the *clas ohlson* website. Clas Ohlson is a swedish hardware store which also has an online shopping service, members club and more.

So, we went to their site www.clasohlson.se and transferred the questions and answers to an excel document. The dataset contains 2 columns, one for the questions and one for the answers, see figure 1.

Note: We decided to make the chatbot swedish, so all questions and answers are in swedish and the program is to be supported by the swedish language. However, this will limit the library functions that we can use.

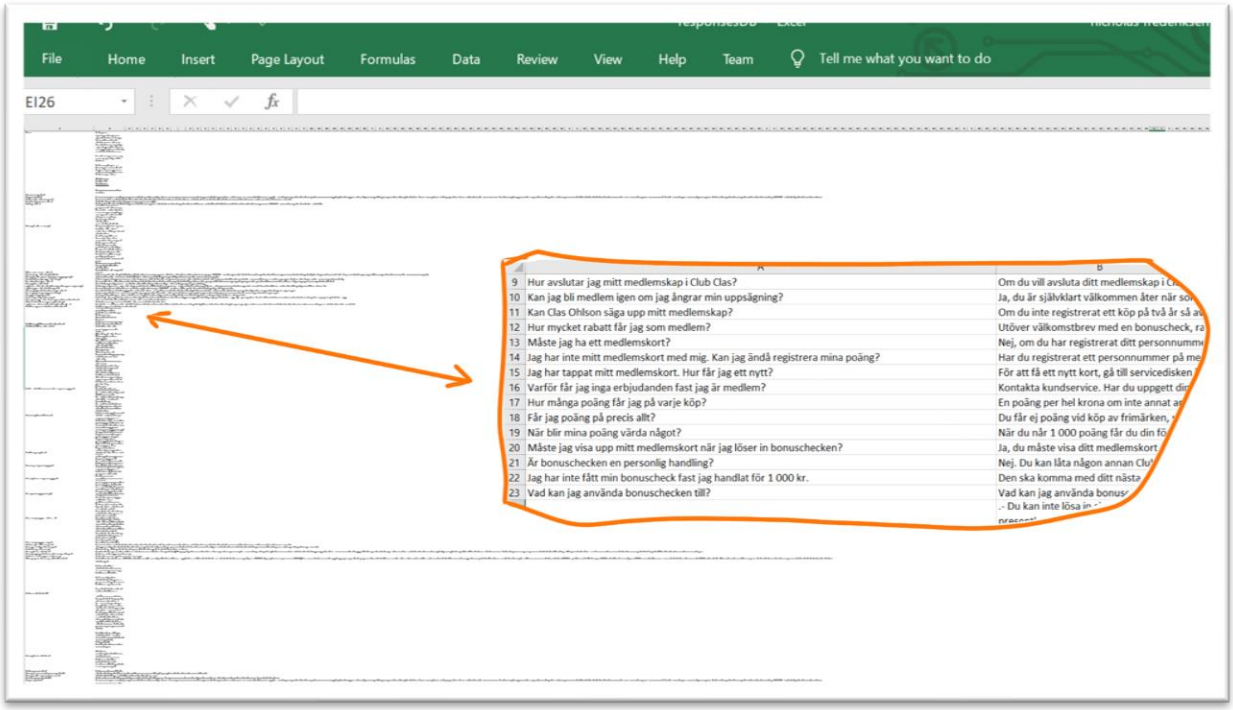


Figure 1. The data frame in Excel.

So what we want to achieve is a way for our chatbot to understand what the user is asking for, with the help of the dataset. To do this we are going to use a method called cosine similarity using the *Natural Language Toolkit*.
With cosine similarity we are basically going to compare the similarity between the users question and the questions in our dataset, and output whichever answer matches the question best.

But before we do this we are going to need some text preprocessing.
Let's look at an overview of the workflow/pipeline, see figure 2.

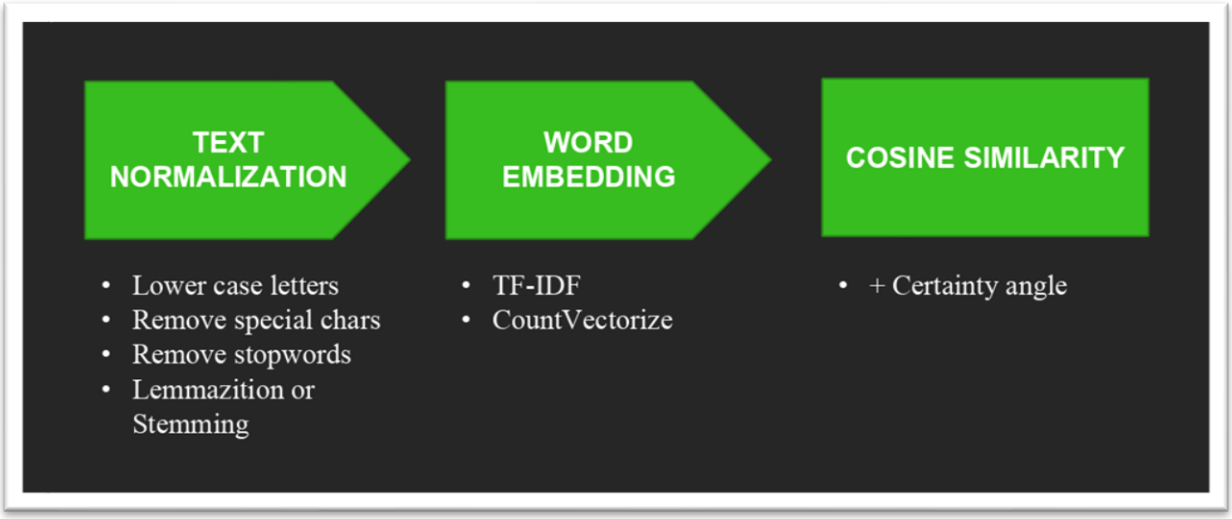


Figure 2. Pipeline used for the chatbot script.

The very first thing to do is to import all the libraries we are going to use, including the Natural Language Toolkit.

```
1  import pandas as pd
2  import nltk
3  import re
4  import textwrap
5  from nltk.stem import SnowballStemmer # to perform stemmer in swedish
6  from sklearn.feature_extraction.text import TfidfVectorizer  # to perform tfidf
7  from sklearn.metrics import pairwise_distances  # to perfrom cosine similarity
8  from nltk.corpus import stopwords  # for stop words
9
```

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Step 1: Text Normalization

Following our workflow image, we start by normalizing the text. The process of normalizing text will include making everything to lowercase letters, removal of special characters (with an exception for ‘ääö’), Stopword removal, and Stemming.

To make everything lowercase we need define a function, see below, where we iteratively replace each letter with its lowercase version. *Note: Since we are going to make a swedish chatbot special characters ‘ääö’ are needed to be incorporated in the re.sub() function.*

```
13  # function that converts text into lower case and removes special characters
14  def toLower(x):
15      for i in x:
16          a = str(i).lower()
17          p = re.sub(r'[^a-z0-9ääö]', ' ', a)
18          print(p) # Print to confirm.
19
```

Stemming the text is a process where you take a word and try to transform it into its base form (*i.e. Singing -> Sing or in swedish Sjunga -> Sjung*). However stemming is not that sophisticated of a process since all it does is remove the endings of a word according to a list of potential words ending, in hopes of obtaining the base word. Drawbacks with this is that it may create a made up word rather than the base word.
(*i.e. swedish for runners Springskor -> Springsk*).
Now may that be the case, for our purpose, this will not have an effect on the final output since we apply stemming on both the input question and the database, word similarities can still be found.
An improvement would be Lemmatization.

“**Lemmatization** usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.”

However, no open source lemmatization function for the swedish language exists at the moment. One is required to make one from scratch which will take a lot of time. Time which we did not have unfortunately.
The following function presented below handles lower case transforming, special character removal and word stemming all in one.

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

```
22  # function that performs text normalization steps
23  def text_normalization(text):
24      text = str(text).lower()  # text to lower case
25      spl_char_text = re.sub(r'[^ a-zåäö]+', '',
26                             text)  # removing special characters
27      tokens = nltk.word_tokenize(spl_char_text)  # word tokenizing
28      stemmer = SnowballStemmer("swedish")
29      stem_words = []
30      for token in tokens:
31          stem_token = stemmer.stem(token)
32          stem_words.append(stem_token)
33
34      return " ".join(stem_words)  # returns the lemmatized tokens as a sentence
35
```

Now let's get rid of useless stopwords by writing this function:

```
36
37  def remove_stopwords(text):
38      stop = stopwords.words('swedish')
39      text = text_normalization(text)
40      t = []
41      new_txt = ""
42      splitted_txt = text.split()
43      for word in splitted_txt:
44          if word in stop:
45              continue
46          else:
47              t.append(word)
48          new_txt = " ".join(t)
49
50      return new_txt
```

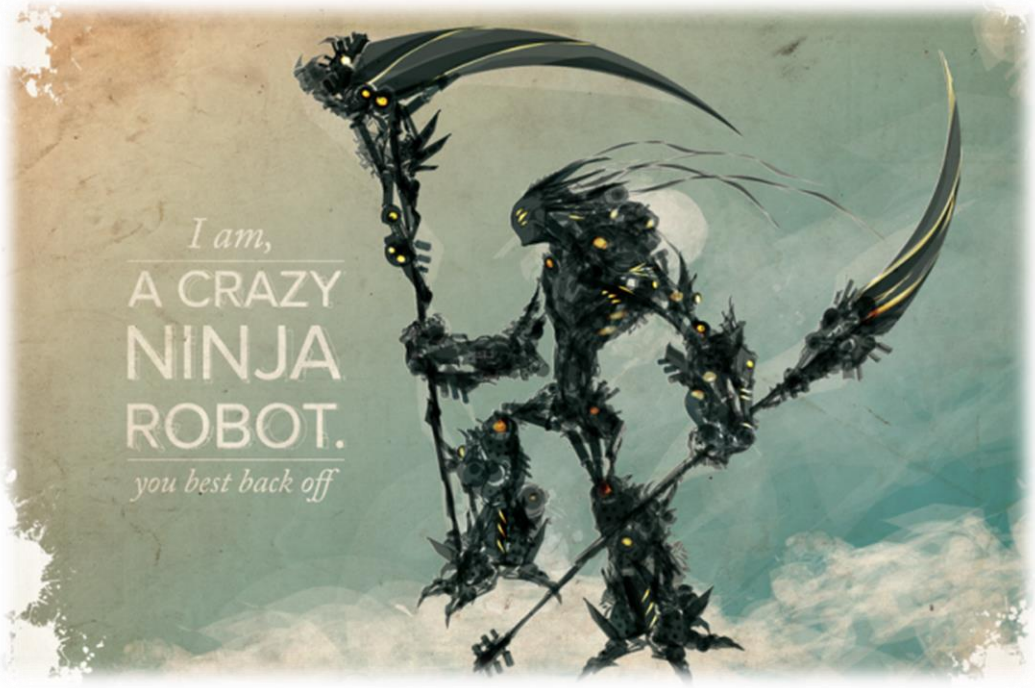
Note: In this function we also call our previous text_normalization function. So when calling this function we effectively stem the text and remove stopwords.
Stop words are extremely common words that would appear to be of little value in matching a user's need and hence they are excluded from the vocabulary entirely. Below are the predefined swedish stop words.

```
36  stopwords.words('swedish')
37
38  ['och', 'det', 'att', 'i', 'en', 'jag', 'hon', 'som', 'han', 'på', 'den', 'med', 'var', 'sig', 'för', 'så', 'till', 'är',
    'men', 'ett', 'om', 'hade', 'de', 'av', 'icke', 'mig', 'du', 'henne', 'då', 'sin', 'nu', 'har', 'inte', 'hans', 'honom',
    'skulle', 'hennes', 'där', 'min', 'man', 'ej', 'vid', 'kunde', 'något', 'från', 'ut', 'när', 'efter', 'upp', 'vi', 'dem',
    'vara', 'vad', 'över', 'än', 'dig', 'kan', 'sina', 'här', 'ha', 'mot', 'alla', 'under', 'någon', 'eller', 'allt',
    'mycket', 'sedan', 'ju', 'denna', 'själv', 'detta', 'åt', 'utan', 'varit', 'hur', 'ingen', 'mitt', 'ni', 'bli', 'blev',
    'oss', 'din', 'dessa', 'några', 'deras', 'blir', 'mina', 'samma', 'vilken', 'er', 'sådan', 'vår', 'blivit', 'dess',
    'inom', 'mellan', 'sådant', 'varför', 'varje', 'vilka', 'ditt', 'vem', 'vilket', 'sitta', 'sådana', 'vart', 'dina',
    'vars', 'vårt', 'våra', 'ert', 'era', 'vilkas']
39
```

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

This is how to the document text in the data frame looks like after the text normalization.

```
In[3]: df.head()
Out[3]:
                                  Context  ... stemmed_text_no_stopwords
0                    Returnera en produkt  ...           returner produk
1                       Fel på en produkt  ...                fel produk
2  Hur kan jag se min beställningsstatus?  ...     se beställningsstatus
3       Kostar det något att vara medlem?  ...                kost medlem
4                    Hur blir jag medlem?  ...                     medlem

[5 rows x 3 columns]
```



Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Step 2: Word Embedding

Our next step is word embedding, it is representation for text where words that have the same meaning have a similar representation. Next we use the natural language toolkits tf-idf functions to vectorize our sentences.

```
74    # Using Tf-IDF
75    tfidf = TfidfVectorizer()
76    tfidf_array = tfidf.fit_transform(df['stemmed_text_no_stopwords']).toarray(
77    )  # transforming data into array.
78    df_tfidf = pd.DataFrame(tfidf_array, columns=tfidf.get_feature_names())
```

To clarify, *tf* is Term Frequency, which means the frequency of the term in the current text, and *idf* is Inverse Document Frequency, which reveals how unusual the word is across documents. Here the document represents a single text, i.e. row 0 or row 1, etc., where the documents refer to all rows in the dataset, see figure 3.

	lös	lösenord	medlem	medlemskap	medlemskort	medlemsvillk
0	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.61125	0.00000	0.00000	0.00000
4	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.61125	0.00000	0.00000	0.00000
6	0.00000	0.00000	0.46208	0.00000	0.00000	0.00000
7	0.00000	0.00000	0.00000	0.51874	0.00000	0.00000
8	0.00000	0.00000	0.38662	0.00000	0.00000	0.00000
9	0.00000	0.00000	0.00000	0.52799	0.00000	0.00000
10	0.00000	0.00000	0.49957	0.00000	0.00000	0.00000
11	0.00000	0.00000	0.00000	0.00000	0.75212	0.00000
12	0.00000	0.00000	0.00000	0.00000	0.42026	0.00000
13	0.00000	0.00000	0.00000	0.00000	0.48936	0.00000
14	0.00000	0.00000	0.37415	0.00000	0.00000	0.00000
15	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
16	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
17	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
18	0.45661	0.00000	0.00000	0.00000	0.36785	0.00000
19	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

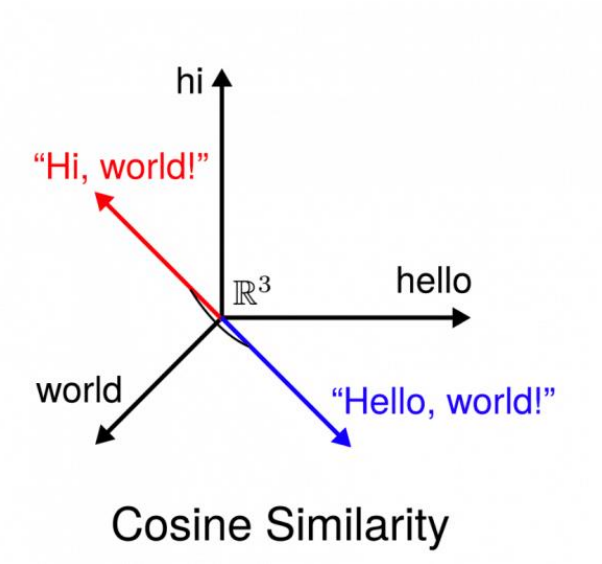
Figure 3. The data frame after tf-idf. Each row is a vectorization of text from the original data frame, the columns are now every word in the document in alphabetical order. The value in each item is the tf-idf value for that word in the row, i.e. it is a kind of score on how important each word is relative to all other words.

Now that the text has been vectorized and scoring values have been obtained by tf-idf we can compare the user inputed question with our data frame questions with Cosine Similarity.

Step 3: Cosine Similarity

Cosine similarity is a measure of similarity between two vectors. It returns a value that is computed by taking the dot product and dividing that by the product of their norms between two vectors.

`Cosine Similarity (a, b) = Dot product(a, b) / ||a|| * ||b||`



To do this, the function below was defined.

```
52
53  def match_answer(array, matrix):
54      cos = 1 - pairwise_distances(
55          matrix, array, metric='cosine')  # Applying cosine similarity
56      index_value = cos.argmax()
57      return index_value, cos.max()
58
```

Notice that the function returns 2 values. An index value and a cosine value. The index value corresponds to the index in the data frame which most matches the input question in text similarity. However, to perceive just how similar they are the best resulting cosine angle is also looked at.

This is done so that we can, through a predefined **certainty angle** (Figure 4), evaluate the quality of the output answer. If the angle between two vectors are to large then it would mean that there is a strong likelihood that the two vectors (though similar relatively to the rest) are too different and would result in the chatbot giving a bad/unrelated response.

In other words, no matter the input a match would be assigned to that input depending on the similarity and if the input is silly or off topic then the chatbot would look foolish when responding with one of the FAQ answers.

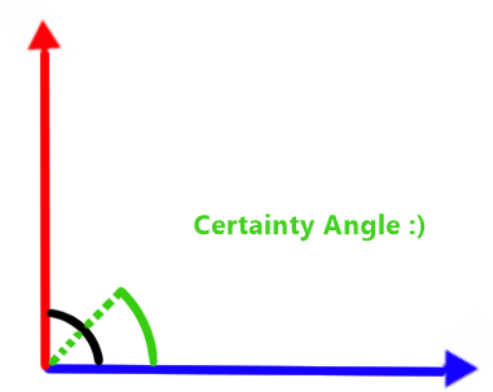


Figure 4. An illustration on the concept of a certainty angle.

Now, what to do if the resulting angle is outside of the certainty angle? Well, we decided to have our chatbot give a pre-written response reflecting its uncertainty of the answer.

The uncertain chatbot response is this:

“I’m afraid I do not have a good answer for that. You may try to reformulate your question or ask me about something else.”

But in swedish of course. This message should convey to the user the limitations of the answers that the chatbot can give, in a less embarrassing fashion.

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Lastly, all we need to do now is to put everything together, add some if-statements to have the chatbot perform some useful scripted dialogs and then the chatbot will be ready to interact with Clas Ohlson customers.

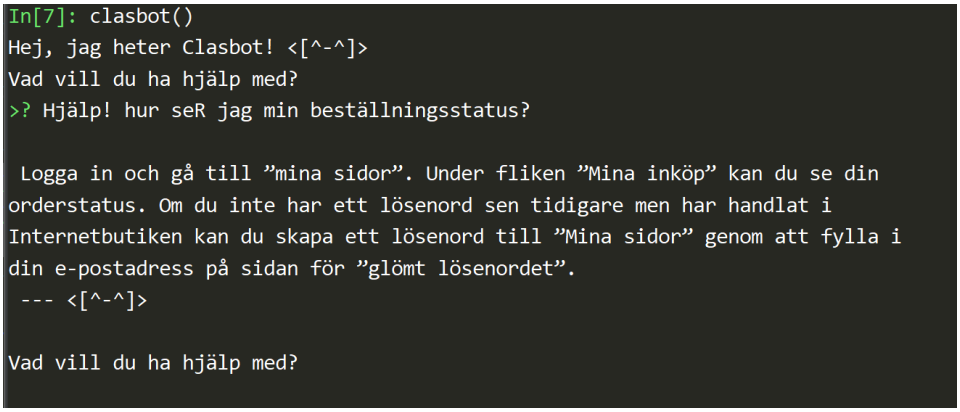
```
def clasbot():
    # FIXA TILL DATABASEN ------------------------------------------------------------------
    toLower(df['Context'])

    df['stemmed_text_no_stopwords'] = df['Context'].apply(
        remove_stopwords
    )  # applying the fuction to the dataset to get clean text

    # Using Tf-IDF
    tfidf = TfidfVectorizer()
    tfidf_array = tfidf.fit_transform(df['stemmed_text_no_stopwords']).toarray(
    )  # transforming data into array.
    df_tfidf = pd.DataFrame(tfidf_array, columns=tfidf.get_feature_names())
    # ------------------------------------------------------------------------------------------
```

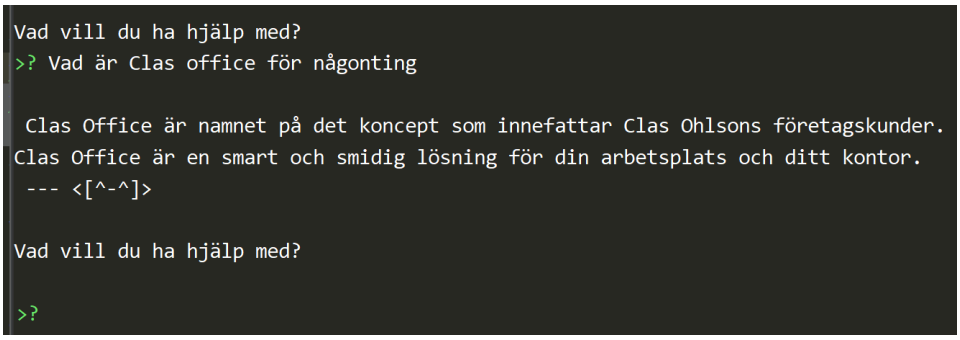
```
    wrapper = textwrap.TextWrapper(width=80)

    print("Hej, jag heter Clasbot! <[^-^]>")
    while True:
        # VAD SÄGER KUNDEN?
        while True:
            try:
                question = input("Vad vill du ha hjälp med? \n")

            except ValueError:
                print("Va?, please try again")
                continue

            if question.lower() == "hej då":
                print("Hej då!")
                return

            no_stop = remove_stopwords(question)
            stemmed = text_normalization(no_stop)
            # Using Tf-IDF
            q_tfidf = tfidf.transform(
                [stemmed]).toarray()  # transforming data into array.

            idx, certainty = match_answer(q_tfidf, df_tfidf)

            if certainty > 0.1:
                # Clasbot says this.
                answer = df['Text Response'].loc[idx]
                print("\n", wrapper.fill(answer), "\n --- <[^-^]> \n")
            else:
                print(
                    "\n Jag har tyvärr inget bra svar på det. \n Du kan prova att formulera om frågan eller ställ en
                    ny.\n --- <[x_x]> \n"
                )


# Kör Clasbot
clasbot()
```

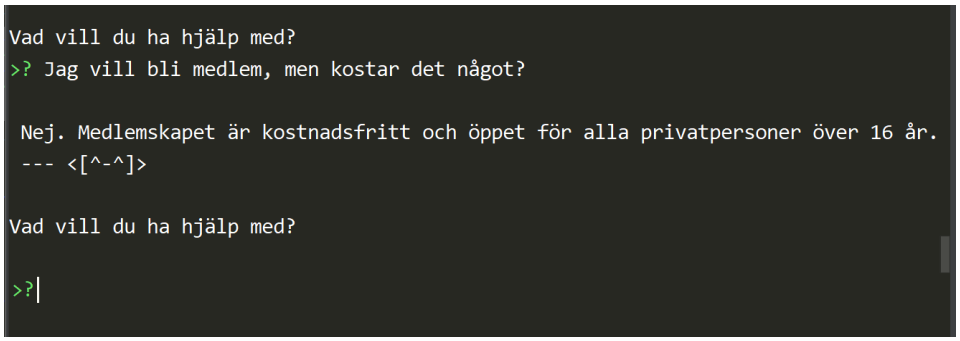
Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Results

Here are some example conversations with the chatbot, newly named to ***Clasbot.***







Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

```
Vad vill du ha hjälp med?
>? Vad äter apor?

 Jag har tyvärr inget bra svar på det.
 Du kan prova att formulera om frågan eller ställ en ny.
 --- <[x_x]>

Vad vill du ha hjälp med?

>?
```

```
Vad vill du ha hjälp med?
>? kan jag veub efn doeod dtet detd etd edetdet vararororor

 Jag har tyvärr inget bra svar på det.
 Du kan prova att formulera om frågan eller ställ en ny.
 --- <[x_x]>

Vad vill du ha hjälp med?
>? kan jag ser mina beställnigar från tidigare?

 Ja. Logga in och gå till ”Mina sidor”. Under fliken “Mina beställningar” kan du
se tidigare inköp som gjorts i internetbutiken. Om du inte har ett lösenord sen
tidigare men har handlat i Internetbutiken kan du skapa ett lösenord till ”Mina
sidor” genom att fylla i din e-postadress på sidan för ”glömt lösenordet”.
 --- <[^-^]>

>?
```

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Discussion

So… what is the answer to our question?
Can a chatbot fulfill the same role of a FAQ section on a website?
Well, based on the results our chatbot gave us we can conclude that it is entirely possible that a chatbot can indeed fulfill such a role. Our chatbot does have some limitations but if these could be addressed then we'd argue that a chatbot is actually a preferable customer experience.

Things that are good with Clasbot

Clasbot can understand input questions of the same meaning but different phrasing, which is good because it means that we can re-use the data in the database, i.e. the database doesn't need to include every single possible way of phrasing a certain question, it finds similar ones. Thanks to cosine similarity.

Clasbot is also fast and gives correct or helpful answers if the topical question is part of the FAQ.

An advantage with interacting with Clasbot rather than a websites FAQ is that an answer can be given to the user much faster, as in the current FAQ section on the Clas Ohlson website, has all the frequently asked questions listed and categorized by topic. They also 'hide' the answers (you must click on the question to reveal them) in an attempt to reduce clutter in the site, which ultimately costs the user some time.

Comparatively, when using Clasbot, if you know what you want the answers to you can just type it in and an answer will be provided immediately. No time wasted on scrolling and clicking around searching.

Things that are less good with Clasbot

The database is rather specific with topics regarding specific functions on the website and fail to answer "general" questions a customer might want answered regarding the products the company sell.

Moreover, due to the relatively small database and few variations of the same questions, by sometimes replacing just one word in the question we get an entirely different response.

For instance, if one asks Clasbot about "how to see one's past purchases" we get a fitting answer, namely that the user can log in and find it under the history tab. But if one instead asked, "how to see one's past billing information" (information that can also be found out by logging in) we get an answer for another question regarding billing, which fails to answer the initial question.

Nicholas Frederiksen, nicfr426
Pontus Söderqvist, ponso669

Things that could improve Clasbot

- Lemmatization would probably improve Clasbots ability to pick out more relevant answers between similar questions, but it does not seem necessary with the current database.
- Extending the database in the form of adding more variants of the same questions.
- Adding a spell checker would also benefit the bot, since we must anticipate eventual user errors when typing.

Conclusion

Yes, in conclusion, an FAQ chatbot can indeed fulfill the role of a website FAQ and be implemented in python with the NLTK library.

Ways to measure the effectiveness of a chatbot could be to ask users to evaluate it after interacting with it, measure click through rate after interacting with the chatbot (we assume that a successful action after the interaction means the user got the help they needed) and/or measure how often the same users interact with the chatbot (we assume that repeat interactions indicate that the user experience is positive and the chatbot is helpful).

Lastly, further improvements to the bot can be made in the form of expanding the database, adding lemmatization, adding spell checking and more customer friendly interaction dialog.

Thanks for reading!