COMP 474 UU,COMP 6741 UU 2214

Lab Session #09

Introduction

Welcome to Lab #09. This week, we'll start developing two types of chatbots: pattern-based (using AIML) and search-based (using cosine similarity on tf.idf vectors).

Follow-up Lab #08

Solution Task #1 (KNN Regression)

Here's a sample program for the kNN regression exercise from lecture Worksheet #6.

Solution Task #2 (KNN Classification)

Here's a solution for running kNN Classification on the *scikit-learn* wine dataset.

Task #1: Building your first Chatbot

The oldest but perhaps still most widely used technique for building a chatbot is using question-answer patterns. Basically, regular expressions are matched against the user's input and rewrite/output patterns generate a response (see the lecture slides and material for more details). As discussed, this technique has its limitations, but this type of bot can be easily combined with the other methods discussed in class.

We'll start with the technology mentioned in the lecture, using the Artificial Intelligence Markup Language (AIML). This has the advantage of being an open standard, but with the downside that active development has largely moved to other platforms (see below for notes on other bot frameworks).

To get started, you need one of the AIML-compatible libraries, e.g., AIML-Bot. Install the library and create your first bot:

```
import aiml_bot
bot = aiml_bot.Bot(learn="mybot.aiml")
while True:
    print(bot.respond(input("> ")))
```

You'll need an AIML file (which is in XML format) to define the question/answer patterns. Here is a first one to test:

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1">
  <category>
    <pattern>HELLO *</pattern>
    <template>Hi Human!</template>
</category>
<category>
  <pattern>HELLO TROLL</pattern>
  <template>Good one, human.</template>
</category>
</aiml>
```

Experiment with generating `<random>` responses as shown in the lecture.

To avoid duplicating patterns for the same kind of interaction, you can use the `<srai>` tag:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<aiml version="1.0.1">
   <category>
      <pattern>WHAT IS COMP 474</pattern>
      <template>COMP 474 is the Intelligent Systems course.</template>
   </category>

   <category>
      <pattern>WHAT IS COMP 6721</pattern>
      <template>COMP 6721 is the Applied Artificial Intelligence course.</template>
   </category>

   <category>
      <pattern>DO YOU KNOW WHAT * IS</pattern>
      <template>
         <srai>WHAT IS <star/></srai>
      </template>
   </category>
</aiml>
```

As you can see, questions matching the pattern `DO YOU KNOW WHAT * IS` are now redirected to the patterns for `WHAT IS *`.

You can find various AIML files online; for example, there is a std-65% AIML file that covers "65% of all standard questions" that a bot could be asked. Try some of them out, but remember that the Python AIML implementations only support AIML v1, so AIML v2 files will not work.

Where to go from here

As mentioned in the lecture, bot development languages & frameworks have become very fragmented, with numerous vendors, all using their own non-standard extension or proprietary bot language. They all offer convenient cloud development and deployment frameworks, but using them locks your bot to their platform (e.g., you cannot develop a bot with Amazon Lex and move it to Google's Dialogflow):

- Pandorabots are based on AIML2.0, but using their proprietary cloud framework

- Amazon's Lex started out with a modified, JSON-based version of AIML that has evolved into its own language
- Google's Dialogflow has its own bot language, originally developed by api.ai
- Microsoft has its own Bot Framework and specialized offers like QnA Maker
- IBM has of course Watson Assistant

Most of these make it easy to connect to other existing services running on the same platform and mix multiple bot techniques (patterns, retrieval, grounding, generation). They also offer convenient cloud deployment to multiple platforms (e.g., Slack, Skype, Twitter, Facebook Messenger). Their downside is the mentioned "vendor lock-in".

Some open source frameworks that are under more active development than the AIML-based ones are Hubot, Rasa, Chatterbot, Errbot, and Will.

Note: In the next lab, we will start working with *Rasa*, which we'll also use for Part #2 of the course project.

Task #2: Search-based chatbots

In this task, we'll use another technique discussed in the lecture: Search-based bots, which look for an answer that is similar to an existing question in a corpus.

To compute the similarity (*user question* vs. *corpus question* or *user question* vs. *corpus answer*), we'll use the same techniques that you already developed: Vectorization using tf.idf and similarity computation using cosine. So, your task is to write a program that:

- takes a user's question as input and converts it into a tf.idf vector
- finds the closed matching question in the dataset by using cosine similarity
- and prints out the existing answer for that question from the dataset.

For our experiments, we'll use the Amazon Q&A dataset that you can download at https://jmcauley.ucsd.edu/data/amazon/qa/

To keep things simple, start with a single category from the "Per-category files" list. Download one of the zip files from the link given above and load it into your program using the function below:

```
import gzip
def read_file(path):
    g = gzip.open(path, 'r')
    for l in g:
        yield eval(l)
dataset = read_file('qa_Appliances.json.gz')
```

From the dataset, we need to read only the questions, i.e., we need to extract the data that have "question" as the key value.

```
question_list = []
for i in dataset:
    question_list.append(i['question'])
    answer_list.append(i['answer'])
```

Now convert the list to an array using Numpy's `asarray()` function:

```
import numpy as np
question_dataset = np.asarray(question_list)
```

Then, just like in previous exercises, use a `TfidfVectorizer()` to convert the natural language data into a vector.

Now you can ask the user to input a question:

```
input_question = input("What is your question: ")
```

Read the user's input and find similar questions that are present in the dataset. For finding the best match, use cosine similarity. Make sure you also vectorize the user's input.

For example, the highest cosine similarity question for the question *"Is the blender powerful?"* (in the appliances data set) should be:

```
Similar question to user's question: Is the Genuine OEM FSP Whirlpool Kitchen Aid Blender Rubber Seal part number 9704204 suitable for Kitchenaid blender KSB560CU1?
```

```
Answer given to the similar question: Not sure, but it did work fine for me. FYI - It is inexpensive enough to order it and try.
If it does not fit, send it back. As an alternative you should be able to contact the company to get a better answer to your question.
```

You can also try printing out the top 5 highest similar questions for the user's question.

Task #3: Classification with k-Nearest-Neighbor (kNN)

For more sophisticated question-answering systems, a first step is to classify the question by type to make sure the generated answer makes sense (there are a lot of possible questions, but most fall into a few similar patterns). The approach shown in the lecture classifies question by the expected answer type, like *Person*, *Location*, *Date*, or *Definition*. Giving enough labeled examples, we can train a machine learning algorithm to classify a new (unseen) question in one of the predefined types.

For our first experiments, we will use the kNN algorithm as discussed in the lecture. We'll again use the implementation from scitkit-learn:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
```

To see how this works, let's start with some simple data:

```
corpus = np.array([
        'Who is Bill Gates',
        'Where is Concordia located',
        'What is AI',
        'What city is McGill located in',
        'Who is McGill'
])
```

As always, we need to convert our text input into a feature vector we can use for machine learning. To keep it simple for now, we'll use the representation in form of tf-idf vectors as before:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
```

However, for supervised learning we also need the labels for (ground truth) for each question. We'll encode this in a vector as follows:

```
# Encode labels: 0="Person", 1="Location", 2="Definition"
y = np.array([0,1,2,1,0])
```

Now you can "train" a classifier (for kNN, this simply stores the vectors with their labels):

```
clf = KNeighborsClassifier(3)
clf.fit(X, y)
```

Here, "3" is *k*, the number of neighbors voting when classifying unseen data (see the documentation). Note that this is a standard pattern when creating a ML model with scikit-learn, you can use other algorithms (e.g., Naive Bayes, Decision Tree, SVM) in the same way (here is a nice cheat sheet for working with scitkit-learn from Datacamp).

Once you have a trained classifier (again, other ML algorithms involve building a model, whereas kNN falls into the "lazy learner" category), you can use it to classify unseen data:

```
q = 'What city is Concordia located in'
q_vec = vectorizer.transform([q])
predict = clf.predict(q_vec)
print('Predicted class = ', predict)
```

This question is close enough to find the right answer (1 = "Location"), but you'll see for most questions you'll get a wrong results: We simply do not have enough training data and additionally use a representation (tf-idf vectors) that have a high dimensionality.

From here, you have two ways to improve:

- Use different features for the machine learning algorithm, like we did on the worksheet. A more realistic solution could NLP techniques like, POS-tagging, to pre-process extract the features (we'll cover how to do this with NLP libraries in the next lab).
- Get more training data; for example, here is a dataset you could use with questions and their types, which look like this:

  ```
  NUM:count How many people in the world speak French ?
  LOC:other Where is the Orinoco ?
  DESC:def What is a Canada two-penny black ?
  ```

That's all for this lab!

Last modified: Tuesday, 22 March 2022, 12:57 PM

◄ Worksheet #08

Jump to...

Lecture Slides #10 ►

Français (fr)
Italiano (it)
العربية (ar)