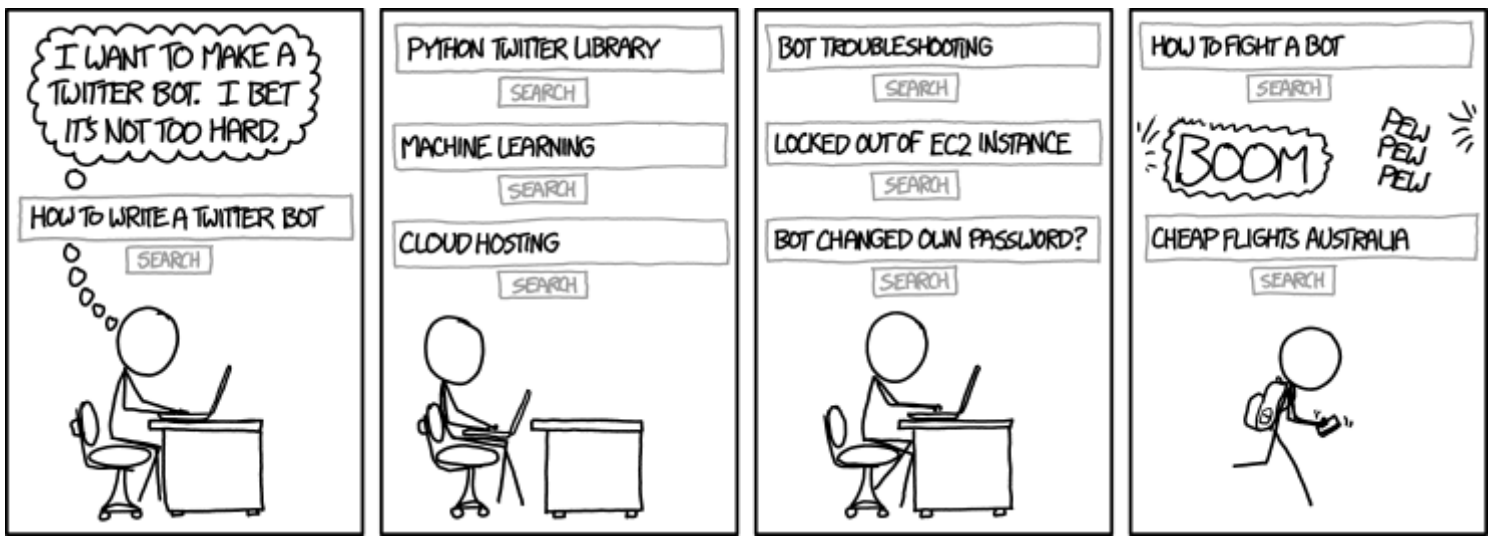
Final project: StackOverflow assistant bot

Congratulations on coming this far and solving the programming assignments! In this final project, we will combine everything we have learned about Natural Language Processing to construct a *dialogue chat bot*, which will be able to:

- answer programming-related questions (using StackOverflow dataset);
- chit-chat and simulate dialogue on all non programming-related questions.

For a chit-chat mode we will use a pre-trained neural network engine available from ChatterBot (https://github.com/gunthercox/ChatterBot). Those who aim at honor certificates for our course or are just curious, will train their own models for chit-chat.



©xkcd (https://xkcd.com)

Data description

To detect *intent* of users questions we will need two text collections:

- `tagged_posts.tsv` — StackOverflow posts, tagged with one programming language (*positive samples*).
- `dialogues.tsv` — dialogue phrases from movie subtitles (*negative samples*).

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import sys
sys.path.append("..")
from common.download_utils import download_project_resources

download_project_resources()
```

```
File data\dialogues.tsv is already downloaded.
File data\tagged_posts.tsv is already downloaded.
```

For those questions, that have programming-related intent, we will proceed as follow predict programming language (only one tag per question allowed here) and rank candidates within the tag using embeddings. For the ranking part, you will need:

- `word_embeddings.tsv` — word embeddings, that you trained with StarSpace in the 3rd assignment. It's not a problem if you didn't do it, because we can offer an alternative solution for you.

As a result of this notebook, you should obtain the following new objects that you will then use in the running bot:

- `intent_recognizer.pkl` — intent recognition model;
- `tag_classifier.pkl` — programming language classification model;
- `tfidf_vectorizer.pkl` — vectorizer used during training;
- `thread_embeddings_by_tags` — folder with thread embeddings, arranged by tags.

Some functions will be reused by this notebook and the scripts, so we put them into *utils.py* file. Don't forget to open it and fill in the gaps!

In [3]:
```
from utils import *
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Xiaowei\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Part I. Intent and language recognition

We want to write a bot, which will not only **answer programming-related questions**, but also will be able to **maintain a dialogue**. We would also like to detect the *intent* of the user from the question (we could have had a 'Question answering mode' check-box in the bot, but it wouldn't fun at all, would it?). So the first thing we need to do is to **distinguish programming-related questions from general ones**.

It would also be good to predict which programming language a particular question referees to. By doing so, we will speed up question search by a factor of the number of languages (10 here), and exercise our *text classification* skill a bit. :)

In [4]:
```
import numpy as np
import pandas as pd
import pickle
import re

from sklearn.feature_extraction.text import TfidfVectorizer
```

Data preparation

In the first assignment (Predict tags on StackOverflow with linear models), you have already learnt how to preprocess texts and do TF-IDF tranformations. Reuse your code here. In addition, you will also need to dump (https://docs.python.org/3/library/pickle.html#pickle.dump) the TF-IDF vectorizer with pickle to use it later in the running bot.

In [5]:
```
def tfidf_features(X_train, X_test, vectorizer_path):
    """Performs TF-IDF transformation and dumps the model."""

    # Train a vectorizer on X_train data.
    # Transform X_train and X_test data.

    # Pickle the trained vectorizer to 'vectorizer_path'
    # Don't forget to open the file in writing bytes mode.

    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    tfidf_vectorizer = TfidfVectorizer(min_df = 5, max_df = 0.9, ngram_range=(1,2), token_pattern = '(\S+)')

    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    tfidf_vectorizer = tfidf_vectorizer.fit(X_train)
    pickle.dump(tfidf_vectorizer, open(vectorizer_path, 'wb'))
    X_train = tfidf_vectorizer.transform(X_train)
    X_test = tfidf_vectorizer.transform(X_test)


    return X_train, X_test
```

Now, load examples of two classes. Use a subsample of stackoverflow data to balance the classes. You will need the full data later.

In [6]:
```
sample_size = 200000

dialogue_df = pd.read_csv('data/dialogues.tsv', sep='\t').sample(sample_size, random_state=0)
stackoverflow_df = pd.read_csv('data/tagged_posts.tsv', sep='\t').sample(sample_size, random_state=0)
```

Check how the data look like:

In [7]:
```
dialogue_df.head()
```

Out[7]:

	text	tag
82925	Donna, you are a muffin.	dialogue
48774	He was here last night till about two o'clock....	dialogue
55394	All right, then make an appointment with her s...	dialogue
90806	Hey, what is this-an interview? We're supposed...	dialogue
107758	Yeah. He's just a friend of mine I was trying ...	dialogue

In [8]:
```
stackoverflow_df.head()
```

Out[8]:

	post_id	title	tag
2168983	43837842	Efficient Algorithm to compose valid expressio...	python
1084095	15747223	Why does this basic thread program fail with C...	c_cpp
1049020	15189594	Link to scroll to top not working	javascript
200466	3273927	Is it possible to implement ping on windows ph...	c#
1200249	17684551	GLSL normal mapping issue	c_cpp

Apply *text_prepare* function to preprocess the data:

In [9]:
```
from utils import text_prepare
```

In [10]:
```
%%time
######### YOUR CODE HERE #############
dialogue_df['text'] = dialogue_df['text'].apply(text_prepare)

######### YOUR CODE HERE #############
stackoverflow_df['title'] = stackoverflow_df['title'].apply(text_prepare)
```

```
Wall time: 1min 22s
```

Intent recognition

We will do a binary classification on TF-IDF representations of texts. Labels will be either `dialogue` for general questions or `stackoverflow` for programming-related questions. First, prepare the data for this task:

- concatenate `dialogue` and `stackoverflow` examples into one sample
- split it into train and test in proportion 9:1, use *random_state=0* for reproducibility
- transform it into TF-IDF features

In [11]:
```
from sklearn.model_selection import train_test_split
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Xiaowei\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [12]:
```
X = np.concatenate([dialogue_df['text'].values, stackoverflow_df['title'].values])
y = ['dialogue'] * dialogue_df.shape[0] + ['stackoverflow'] * stackoverflow_df.shape[0]

######### YOUR CODE HERE ##########
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
print('Train size = {}, test size = {}'.format(len(X_train), len(X_test)))

######### YOUR CODE HERE ###########
X_train_tfidf, X_test_tfidf = tfidf_features(X_train, X_test, RESOURCE_PATH['TFIDF_VECTORIZER'])
```

```
Train size = 360000, test size = 40000
```

Train the **intent recognizer** using LogisticRegression on the train set with the following parameters: *penalty='l2'*, *C=10*, *random_state=0*. Print out the accuracy on the test set to check whether everything looks good.

In [13]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [14]:
```
######################################
######### YOUR CODE HERE #############
######################################
intent_recognizer = LogisticRegression(penalty='l2', C=10, random_state=0).fit(X_train_tfidf, y_train)
```

```
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lb
fgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.h
tml)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modu
les/linear_model.html#logistic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [15]:
```
# Check test accuracy.
y_test_pred = intent_recognizer.predict(X_test_tfidf)
test_accuracy = accuracy_score(y_test, y_test_pred)
print('Test accuracy = {}'.format(test_accuracy))
```

```
Test accuracy = 0.9916
```

Dump the classifier to use it in the running bot.

In [16]:
```
pickle.dump(intent_recognizer, open(RESOURCE_PATH['INTENT_RECOGNIZER'], 'wb'))
```

Programming language classification

We will train one more classifier for the programming-related questions. It will predict exactly one tag (=programming language) and will be also based on Logistic Regression with TF-IDF features.

First, let us prepare the data for this task.

In [17]:
```
X = stackoverflow_df['title'].values
y = stackoverflow_df['tag'].values
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print('Train size = {}, test size = {}'.format(len(X_train), len(X_test)))
```

```
Train size = 160000, test size = 40000
```

Let us reuse the TF-IDF vectorizer that we have already created above. It should not make a huge difference which data was used to train it.

In [19]:
```
vectorizer = pickle.load(open(RESOURCE_PATH['TFIDF_VECTORIZER'], 'rb'))

X_train_tfidf, X_test_tfidf = vectorizer.transform(X_train), vectorizer.transform(X_test)
```

Train the **tag classifier** using OneVsRestClassifier wrapper over LogisticRegression. Use the following parameters: *penalty='l2'*, *C=5*, *random_state=0*.

In [20]:
```
from sklearn.multiclass import OneVsRestClassifier
```

In [21]:
```
######################################
######### YOUR CODE HERE #############
######################################
tag_classifier = OneVsRestClassifier(LogisticRegression(penalty='l2', C=5, random_state=0), n_jobs=-1) \
                                     .fit(X_train_tfidf, y_train)
```

In [22]:
```
# Check test accuracy.
y_test_pred = tag_classifier.predict(X_test_tfidf)
test_accuracy = accuracy_score(y_test, y_test_pred)
print('Test accuracy = {}'.format(test_accuracy))
```

```
Test accuracy = 0.80025
```

Dump the classifier to use it in the running bot.

In [23]:
```
pickle.dump(tag_classifier, open(RESOURCE_PATH['TAG_CLASSIFIER'], 'wb'))
```

Part II. Ranking questions with embeddings

To find a relevant answer (a thread from StackOverflow) on a question you will use vector representations to calculate similarity between the question and existing threads. We already had `question_to_vec` function from the assignment 3, which can create such a representation based on word vectors.

However, it would be costly to compute such a representation for all possible answers in *online mode* of the bot (e.g. when bot is running and answering questions from many users). This is the reason why you will create a *database* with pre-computed representations. These representations will be arranged by non-overlaping tags (programming languages), so that the search of the answer can be performed only within one tag each time. This will make our bot even more efficient and allow not to store all the database in RAM.

Load StarSpace embeddings which were trained on Stack Overflow posts. These embeddings were trained in *supervised mode* for duplicates detection on the same corpus that is used in search. We can account on that these representations will allow us to find closely related answers for a question.

If for some reasons you didn't train StarSpace embeddings in the assignment 3, you can use pre-trained word vectors (https://code.google.com/archive/p/word2vec/) from Google. All instructions about how to work with these vectors were provided in the same assignment. However, we highly recommend to use StartSpace's embeddings, because it contains more appropriate embeddings. If you chose to use Google's embeddings, delete the words, which is not in Stackoverflow data.

In [24]:
```
starspace_embeddings, embeddings_dim = load_embeddings('textsimilarity.tsv')
```

Since we want to precompute representations for all possible answers, we need to load the whole posts dataset, unlike we did for the intent classifier:

In [25]:
```
posts_df = pd.read_csv('data/tagged_posts.tsv', sep='\t')
```

Look at the distribution of posts for programming languages (tags) and find the most common ones. You might want to use pandas groupby (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html) and count (https://pandas.pydata.org/pandas-

docs/stable/generated/pandas.DataFrame.count.html) methods:

In [26]:
```
posts_df
```

Out[26]:

	post_id	title	tag
0	9	Calculate age in C#	c#
1	16	Filling a DataSet or DataTable from a LINQ que...	c#
2	39	Reliable timer in a console application	c#
3	42	Best way to allow plugins for a PHP application	php
4	59	How do I get a distinct, ordered list of names...	c#
...	...	...	...
2171570	45887455	What is the difference between node.js and ayo...	javascript
2171571	45887857	Why do sequential containers have both size_ty...	c_cpp
2171572	45892983	why 1 + + "1" === 2; +"1" + + "1" === 2 and "1...	javascript
2171573	45893693	Why does the first line work but the second li...	javascript
2171574	45898184	Can I safely convert struct of floats into flo...	c_cpp

2171575 rows × 3 columns

In [27]:
```
######### YOUR CODE HERE #############
counts_by_tag = posts_df.groupby('tag').count()['title']
counts_by_tag
```

Out[27]:
```
tag
c#            394451
c_cpp         281300
java          383456
javascript    375867
php           321752
python        208607
r              36359
ruby           99930
swift          34809
vb             35044
Name: title, dtype: int64
```

Now for each `tag` you need to create two data structures, which will serve as online search index:

- `tag_post_ids` — a list of post_ids with shape `(counts_by_tag[tag],)`. It will be needed to show the title and link to the thread;
- `tag_vectors` — a matrix with shape `(counts_by_tag[tag], embeddings_dim)` where embeddings for each answer are stored.

Implement the code which will calculate the mentioned structures and dump it to files. It should take several minutes to compute it.

In [28]:
```
import os
os.makedirs(RESOURCE_PATH['THREAD_EMBEDDINGS_FOLDER'], exist_ok=True)

for tag, count in counts_by_tag.items():
    tag_posts = posts_df[posts_df['tag'] == tag]

    ######### YOUR CODE HERE #############
    tag_post_ids = tag_posts['post_id']

    print(count, embeddings_dim)
    tag_vectors = np.zeros((count, embeddings_dim), dtype=np.float32)
    for i, title in enumerate(tag_posts['title']):
        ######### YOUR CODE HERE #############
        tag_vectors[i, :] = question_to_vec(title, starspace_embeddings, embeddings_dim)

    # Dump post ids and vectors to a file.
    filename = os.path.join(RESOURCE_PATH['THREAD_EMBEDDINGS_FOLDER'], os.path.normpath('%s.pkl' % tag))
    pickle.dump((tag_post_ids, tag_vectors), open(filename, 'wb'))
```

```
394451 100
281300 100
383456 100
375867 100
321752 100
208607 100
36359 100
99930 100
34809 100
35044 100
```

In [ ]:
```

```