Tag Suggestion for Stackoverflow

Group 21

Gathering data

- StackLite: a kaggle dataset consisting of only StackOverflow question-ids
- StackAPI: A python wrapper for the Stack Exchange API
- Use StackAPI to retrieve question body and its tags

```
# question_ids is a list containing all the question IDs to retrieve
# Only 100 question IDs can be queried in a single request

SITE = StackAPI('stackoverflow')
questions = SITE.fetch('questions', ids=question_ids[start:end], filter='withbody')
```

Basic approaches

- Approach 1: Predicted tags = set of tokens in the question body that appear in the tags dictionary
- Approach 2: Predicted tags = top-5 tokens in the question body ranked according to their tag popularity
- Approach 3: Prune the tags dictionary to remove the bottom 10% of the tags and then predict top-5 tokens in the question body

Previous results

Ground Truth	Predicted Tags - I	Top-5 occuring tags	Pruned tags dictionary
html html-form submit-button form-submit	html user wizard button markup	html user wizard button markup	html
winforms type-conversion c# decimal opacity	build using vb.net double opacity	build using vb.net double opacity	vb.net
file-type office-2007 mime	upload where types mime list find	upload mime types list find	list types
.net datetime c#	datetime	datetime	datetime
visual-c++ timer linux winapi unix	process linux discover api kernel compiled find porting port	process linux api kernel port	linux process api
email email-spam	using terminology	using terminology	
landscape ios objective-c	device iphone using interface landscape	device iphone using interface landscape	iphone interface
datetime time c# relative-time-span datediff	datetime time	datetime time	datetime
.net scripting c# compiler-construction	deployment .net c# each file function class trading interface assembly refresh database	deployment .net c# file database	deployment .net c# file database
sorting dictionary c#	dictionary key class map hash	dictionary key class map hash	dictionary hash class
Average precision	22%	26%	36%
Average recall	42%	38%	30%

With the basic token-in-tag-dictionary approach, we got a maximum f1-score of 0.30!

Preprocessing

- Preprocessing
  - BeautifulSoup to separate code snippets
  - nltk tokenizer and stopwords
  - sklearn TfidfVectorizer
- Pruning tags
  - Keep the top-k tags
- Pruning vocabulary
  - Keep words that are present in more than 0.2% of the total documents => every word in vocabulary is present in at least 70 documents.

Neural network

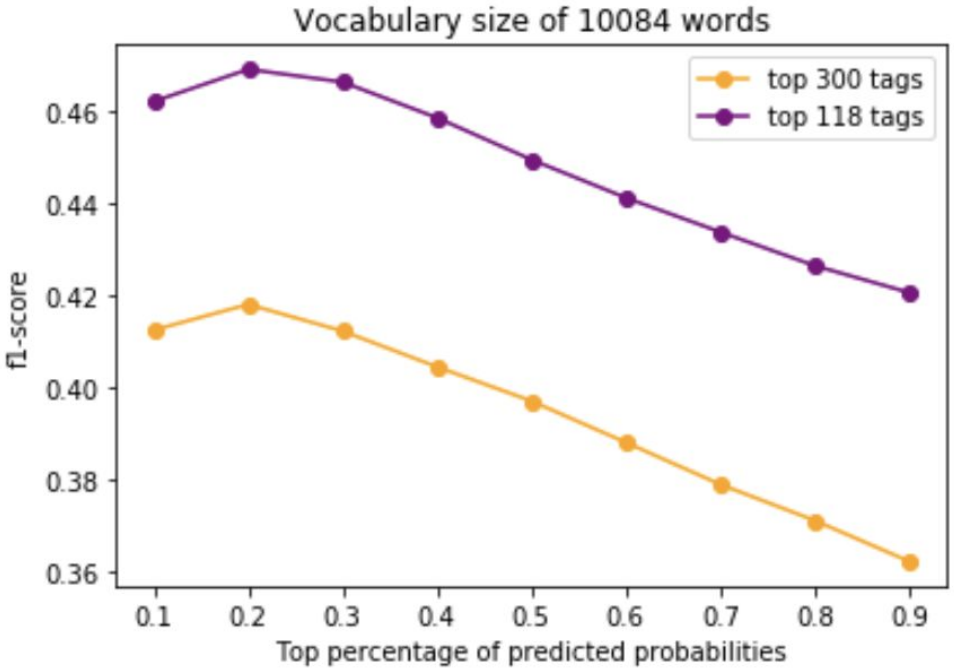
```
model = Sequential()
model.add(Dense(10500, activation='linear', input_dim=features))
model.add(LeakyReLU(alpha=.1))
model.add(Dense(2048, activation='linear'))
model.add(LeakyReLU(alpha=.1))
model.add(Dense(512, activation='linear'))
model.add(LeakyReLU(alpha=.1))
model.add(Dense(256, activation='linear'))
model.add(LeakyReLU(alpha=.1))
model.add(Dense(num_tags, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=25, batch_size=32, verbose=2)
```

Observations

- With code snippets vs without code snippets
  - Including the code snippets in the documents increases F1-score by 4%

- Activation function: Leaky Relu outperforms sigmoid, tanh and Relu
  - F1-score was 2% higher for Relu than for Sigmoid and tanh
  - It was further higher by 2% for Leaky Relu than for Relu
  - Model was stable at alpha = 0.1 for the Leaky Relu

Results



Experiment details:

- 35k stackoverflow posts
- Vocabulary size of 10084 after pruning words using mindf=0.0002
- Top 300 and 118 tags
- k = [0.1, 0.2, … , 0.9]

Supervised LDA

```
def inference(self):
    V = len(self.vocas)
    for m, doc, label in zip(range(len(self.docs)), self.docs, self.labels):
        for n in range(len(doc)):
            t = doc[n]
            z = self.z_m_n[m][n]
            self.n_m_z[m, z] -= 1
            self.n_z_t[z, t] -= 1
            self.n_z[z] -= 1

            denom_a = self.n_m_z[m].sum() + self.K * self.alpha
            denom_b = self.n_z_t.sum(axis=1) + V * self.beta
            p_z = label * (self.n_z_t[:, t] + self.beta) / denom_b * (self.n_m_z[m] + self.alpha) / denom_a
            new_z = numpy.random.multinomial(1, p_z / p_z.sum()).argmax()

            self.z_m_n[m][n] = new_z
            self.n_m_z[m, new_z] += 1
            self.n_z_t[new_z, t] += 1
            self.n_z[new_z] += 1
```

Supervised LDA

```
-- label 0 : common
like: 0.0128
use: 0.0123
would: 0.0115
way: 0.0113
code: 0.0084
know: 0.0076
want: 0.0072
need: 0.0066
work: 0.0064
file: 0.0058
net: 0.0052
server: 0.0049
best: 0.0048
anyone: 0.0048
user: 0.0047
```

```
-- label 8 : mysql
mysql: 0.1427
database: 0.0897
data: 0.0815
server: 0.0326
administrator: 0.0245
compatibility: 0.0204
backups: 0.0204
log: 0.0163
binary: 0.0163
replicate: 0.0122
matter: 0.0122
dump: 0.0122
```

```
-- label 28 : vim
vim: 0.1598
emacs: 0.0959
version: 0.0959
prime: 0.0320
editors: 0.0320
graphical: 0.0320
macvim: 0.0320
cocoa: 0.0320
carbonemacs: 0.0320
xemacs: 0.0320
aquamacs: 0.0320
tough: 0.0320
```

```
-- multithreading
static: 0.0468
reproduce: 0.0312
threads: 0.0234
queue: 0.0234
thread: 0.0234
variables: 0.0234
safe: 0.0234
ram: 0.0156
mutex: 0.0156
watson: 0.0156
alloc: 0.0156
cpu: 0.0156
threaded: 0.0156
```

Topic modelling looks like a promising approach but currently does not scale well with the amount of data

Github repository

https://github.com/bhvjain/stackoverflow-tag-suggestion