AI9000

Alex Lang

Conor Rogers

Confusion

```
# unzip and read story from zip file
def unzip_corpus(input_file, name):
    zip_archive = zipfile.ZipFile(input_file)
    contents = [zip_archive.open(fn, 'r').read().decode('utf-8')
        for fn in zip_archive.namelist() if fn == name]
    return ''.join(contents)

# get bag of words
# breaks sentences into set of tokenized sentences, removing stopwords
def get_bow(tagged_tokens, stopwords=""):
    if stopwords == "":
        return set([t[0].lower() for t in tagged_tokens])
    else:
        return set([t[0].lower() for t in tagged_tokens if t[0].lower() not in stopwords])

# The standard NLTK pipeline for POS tagging a document
def get_sentences(text):
    sentences = nltk.sent_tokenize(text)
    sentences = [nltk.word_tokenize(sent) for sent in sentences]
    sentences = [nltk.pos_tag(sent) for sent in sentences]

    return sentences

#returns lemma of word added
def lemmatizer(tokens):
    lem_tokens = []
    # this little bit is because wordnet lemmas don't play nicely with things verb infinitives..... [very rough fix
    second_form_same_vinfinitive = [('felt','feel'),('fell','fall'),('stood','stand'),('flattered','flatter'),('flattery','flatter')]

    vinfinitive_check = [a for (a,b) in second_form_same_vinfinitive]
    for token in tokens:
        for (a,b) in second_form_same_vinfinitive:
            if a == token:
                lem_tokens += [b]
        if token not in vinfinitive_check:
            lem_tokens += [WordNetLemmatizer().lemmatize(token,'v')]

    return lem_tokens

# qtokens: is a list of pos tagged question tokens with SW removed
# text: list of a list of pos tagged story sentences
# stopwords is a set of stopwords
# matches words to sentences for each text and returns the best answer
def baseline(qbow, text, stopwords):
    # Collect all the candidate answers
    answers = []
    qbow = set([nltk.LancasterStemmer().stem(word) for word in qbow])
    qbow.update(set(lemmatizer(qbow)))
    print(qbow)
    for f in text:
        for sent in f:
            # A list of all the word tokens in the sentence
            sbow = get_bow(sent, stopwords)

            # stem all questions and sentences for better results
            sbow = set([nltk.LancasterStemmer().stem(word) for word in sbow])
            sbow.update(set(lemmatizer(sbow)))

            # and then add the other
            print(sbow)

            # Count the # of overlapping words between the Q and the A
            # & is the set intersection operator
            overlap = len(qbow & sbow)
            print(c.OKGREEN + "overlap: " + c.ENDC + str(overlap))

            answers.append((overlap, sent))

    # Sort the results by the first element of the tuple (i.e., the count)
    # Sort answers from smallest to largest by default, so reverse it
    answers = sorted(answers, key=operator.itemgetter(0), reverse=True)
    #print(answers)

    # Return the best answer
    best_answer = (answers[0])[1]

    return best_answer

# reads file and finds best sentence
def find_best_sentence(question, fnames):
    print(c.OKGREEN)
    print(fnames)

    # raw story / sch
    dataset = "hw6_dataset"
    text = [unzip_corpus(dataset + ".zip", dataset + "/" + f) for f in fnames]

    # get words for every sentence in sentence
    stopwords = set(nltk.corpus.stopwords.words("english"))
    #stopwords = ""
```

55%

We did not understand
Dep. Parsing OR Con. Parsing

Rejection

We used REGEX
& a Dumb Luck Algorithm

Acceptance

```
import argparse, re, nltk, math, sys
from nltk.tree import Tree
import chunky
import process_questions

# Read the constituency parse from the line and construct the Tree
def read_con_parses(parfile):
    fh = open(parfile, 'r')
    lines = fh.readlines()
    fh.close()
    return [Tree.fromstring(line) for line in lines]

# See if our pattern matches the current root of the tree
def matches(pattern, root):
    # Base cases to exit our recursion
    # If both nodes are null we've matched everything so far
    if root is None and pattern is None:
        return root

    # We've matched everything in the pattern we're supposed to (we can ignore the extra
    # nodes in the main tree for now)
    elif pattern is None:
        return root

    # We still have something in our pattern, but there's nothing to match in the tree
    elif root is None:
        return None

    # A node in a tree can either be a string (if it is a leaf) or node
    plabel = pattern if isinstance(pattern, str) else pattern.label()
    rlabel = root if isinstance(root, str) else root.label()

    # If our pattern label is the * then match no matter what
    if plabel == "*":
        return root
    # Otherwise they labels need to match
    elif plabel == rlabel:
        # If there is a match we need to check that all the children match
        # Minor bug (what happens if the pattern has more children than the tree)
        for pchild, rchild in zip(pattern, root):
            match = matches(pchild, rchild)
            if match is None:
                return None
        return root

    return None

def pattern_matcher(pattern, tree):
    for subtree in tree.subtrees():
        node = matches(pattern, subtree)
        if node is not None:
            return node
    return None

def subtree_master(pattern,tree):
    subtree = pattern_matcher(pattern,tree)

    if subtree is None:
        # pattern = nltk.ParentedTree.fromstring("(NP (*) )")
        # subtree = pattern_matcher(pattern,tree)
        # print(subtree)
        # print(" ".join(subtree.leaves()))
        return 'nopes'


    print(subtree)
    print(" ".join(subtree.leaves()))
    return subtree


def q_determine(question,tree):
    search_words = question.lower().split()
    # print(search_words)
    if 'who' in search_words:
        print('who')
```

65%

We switched to
Constituency Parsing

Victory

68%

Held-Out Data

Recall

64.4%

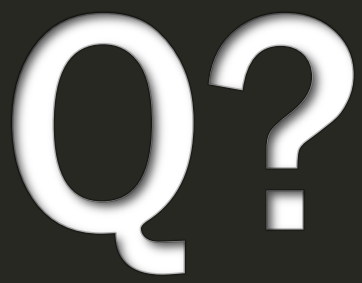
Precision

63.2%

F-score

63.8%

Reflections



We don’t ‘know’ what the Q is asking
We treat each ‘wh’ Q type accordingly

Observations

Recall

What Q is asking for &
Finding Best Sentence

Precision

Answering Q