Stage 1

April 19, 2021

1 Conventional Feature Based QA System

1.0.1 Structure

- import library
- initialize data file path
- functions of read dataset
  - read MCTest dataset
  - read DREAM dataset
  - read RACE dataset
- read dataset
- functions of predicting answer
  - get highest similarity choice
  - word tokenization
  - get synonyms of words
- predict MC answer, of each question
- predict MC answer, of whole dataset
- main function (starting point)
- analysis

1.0.2 Getting Started

Install required python package Execute by Jupyter notebook compiler Set data file path, default path as: /datasets/MCTest/MCTest/ /datasets/DREAM/ /datasets/RACE/RACE/ Results of each dataset will be exported to: /Stage 1 result/{dataset name}.csv

1.0.3 Package used

python 3.8.6, os, json, itertools pandas 1.2.2, numpy 1.19.5, gensim 3.8.3, nltk 3.5

[1]:
```
import pandas as pd
import numpy as np
import os
import json
import gensim       # for similarity
from nltk import word_tokenize, WordNetLemmatizer
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from nltk.corpus import wordnet     # get synonyms
```

```
from itertools import chain
```

1.1 Initialize data file path

[2]:
```
# file path, dataset stored in "/datasets" in the script directory
path = {
    "MC160":{
        "Train":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc160.train.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc160.train.ans")
        },
        "Dev":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc160.dev.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc160.dev.ans")
        },
        "Test":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc160.test.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/
↪MCTestAnswers/mc160.test.ans")
        }
    },
    "MC500":{
        "Train":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc500.train.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc500.train.ans")
        },
        "Dev":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc500.dev.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc500.dev.ans")
        },
        "Test":{
            "Question": os.path.join(sys.path[0], "datasets", "MCTest/MCTest/
↪mc500.test.tsv"),
            "Answer": os.path.join(sys.path[0], "datasets", "MCTest/
↪MCTestAnswers/mc500.test.ans")
        }
    },
```

```
    "DREAM":{
        "Train": os.path.join(sys.path[0], "datasets", "DREAM/train.json"),
        "Dev": os.path.join(sys.path[0], "datasets", "DREAM/dev.json"),
        "Test": os.path.join(sys.path[0], "datasets", "DREAM/test.json")
    },
    "RACE":{
        "high": {
            "Train": os.path.join(sys.path[0], "datasets", "RACE/RACE/train/
↪high"),
            "Dev": os.path.join(sys.path[0], "datasets", "RACE/RACE/dev/high"),
            "Test": os.path.join(sys.path[0], "datasets", "RACE/RACE/test/high")
        },
        "middle": {
            "Train": os.path.join(sys.path[0], "datasets", "RACE/RACE/train/
↪middle"),
            "Dev": os.path.join(sys.path[0], "datasets", "RACE/RACE/dev/
↪middle"),
            "Test": os.path.join(sys.path[0], "datasets", "RACE/RACE/test/
↪middle")
        }
    }
}
```

1.2 Read dataset file

[3]:
```
def readMCTest(questionPath, answerPath):
    question = pd.read_csv(questionPath,
        sep='\t',
        header=None,
        names=["id", "properties", "article",
               "q0", "q0_c0", "q0_c1", "q0_c2", "q0_c3",
               "q1", "q1_c0", "q1_c1", "q1_c2", "q1_c3",
               "q2", "q2_c0", "q2_c1", "q2_c2", "q2_c3",
               "q3", "q3_c0", "q3_c1", "q3_c2", "q3_c3",])
    answer = pd.read_csv(answerPath,
            sep='\t',
            header=None,
            names=['q0_ans', 'q1_ans', 'q2_ans', 'q3_ans', ])

    # pre-processing
    dataset = []
    for index, row in question.iterrows():
        # for each story
        for i in range(4):
            # for each question
            temp = {}
```

```
            temp["article"] = row["article"].replace("\\newline", " ")  # remove "\newline" char in article
            temp["question"] = row[f"q{i}"].split(":")[1]
            temp["answer sentence type"] = row[f"q{i}"].split(":")[0]
            for j in range(4):
                temp[f"choice {j}"] = row[f"q{i}_c{j}"]

            # answer choice = A/B/C/D, answer index = 0/1/2/3, answer = answer in string format
            temp["answer choice"] = answer.iloc[index][f"q{i}_ans"]
            temp["answer index"] = ord(temp["answer choice"]) - 65      # from "A" to 0
            temp["answer"] = temp["choice {}".format(ord(temp["answer choice"]) - 65)]

            dataset.append(temp)

    return pd.DataFrame(dataset)
```

[4]:
```
def readDREAM(docPath):
    with open(docPath) as f:
        data = json.load(f)

    dataset = []
    for story in data:
        temp = {}

        # pre-processing of article
        # for sentence spoke by "M:", add prefix "Men:" to every sentence
        # for sentence spoke by "W:", add prefix "Women:" to every sentence
        temp["article"] = ""
        for sentence in story[0]:
            if "M:" in sentence:
                sentence = sentence.replace("M: ", "")
                for sent in sent_tokenize(sentence):
                    temp["article"] += "Men: " + sent + " "
            elif "W:" in sentence:
                sentence = sentence.replace("W: ", "")
                for sent in sent_tokenize(sentence):
                    temp["article"] += "Woman: " + sent + " "
            else:
                temp["article"] += sentence

        temp["question"] = story[1][0]["question"]
        for i in range(len(story[1][0]["choice"])):
            temp[f"choice {i}"] = story[1][0]["choice"][i]
```

```
        # answer choice = A/B/C/D, answer index = 0/1/2/3, answer = answer in␣
↪string format
        temp["answer"] = story[1][0]["answer"]
        for i in range(len(story[1][0]["choice"])):
            if story[1][0]["choice"][i] == story[1][0]["answer"]:
                temp["answer choice"] = chr(i + 65)      # from 0 to "A"
                temp["answer index"] = i
                break

        dataset.append(temp)

    return pd.DataFrame(dataset)
```

[5]:
```
def readRACE(docPath):
    dataset = []

    for filename in os.listdir(docPath):
        with open(os.path.join(docPath, filename), 'r') as f: # open in␣
↪readonly mode
            story = json.load(f)

        temp = {}
        temp["article"] = story["article"]

        tempPerQuestion = {}
        for i in range(len(story["questions"])):
            temp = {"article": temp["article"]}

            temp["question"] = story["questions"][i]
            for j in range(len(story["options"][i])):
                temp[f"choice {j}"] = story["options"][i][j]

            # answer choice = A/B/C/D, answer index = 0/1/2/3, answer = answer␣
↪in string format
            temp["answer index"] = ord(story["answers"][i]) - 65      # from␣
↪"A" to 0
            temp["answer"] = temp[f"choice {temp['answer index']}"]
            temp["answer choice"] = story["answers"][i]

            dataset.append(temp)

    return pd.DataFrame(dataset)
```

[6]:
```
def getDataset(dSet, purpose):
    """
    Inputs:
        dSet[String] = MC160 / MC500 / DREAM / RACE-middle / RACE-high
```

```
        purpose[String] = Train / Test / Dev
    Return:
        dataset[pd.DataFrame]: dataset from selected data file
    """
    print("{} {}".format(dSet, purpose))
    if dSet == "MC160" or dSet == "MC500":
        pathQuestion = path[dSet][purpose]["Question"]
        pathAnswer = path[dSet][purpose]["Answer"]

        dataset = readMCTest(questionPath=pathQuestion, answerPath=pathAnswer)
    elif dSet == "DREAM":
        dataset = readDREAM(path[dSet][purpose])
    elif dSet == "RACE-high":
        dataset = readRACE(path["RACE"]["high"][purpose])
    elif dSet == "RACE-middle":
        dataset = readRACE(path["RACE"]["middle"][purpose])

    return dataset

# for quick view only
dataset = getDataset("MC160", "Dev")
print(dataset.shape)
dataset.head(2)
```

```
MC160 Dev
(120, 10)
```

[6]:

	article	question	answer	sentence type	choice 0	choice 1	choice 2	choice 3	answer choice	answer index	answer
0	It was Jessie Bear's birthday. She was having …	Who was having a birthday?	one	one	Jessie Bear	no one	Lion	Tiger	A	0	Jessie Bear
1	It was Jessie Bear's birthday. She was having …	Who didn't come to the party?	multiple	multiple	Lion	Tiger	Snake	Jessie Bear	C	2	Snake

1.3 Sub function of getting prediction

[7]:
```
def getClosestSentence(choices, answerString):
    """
    Find the highest similarity choice by a given query
    Inputs:
        choices[List]: list of sentences as document
        answerString[String]: sentence as query
    Return:
```

```
        index[Int]: index of choice with highest similarity
    """
    # for choices
    gen_docs = []   # 1 item = 1 choice

    # word lemmatize, to lower case, and filter stopword of each choice
    for choice in choices:
        gen_docs.append([w.lower() for w in wordTokenize(choice)])

    # dictionary of choices, convert to BOW for each choice
    dictionary = gensim.corpora.Dictionary(gen_docs)
    corpus = [dictionary.doc2bow(gen_doc) for gen_doc in gen_docs]

    # build TFIDF model
    tfidf = gensim.models.TfidfModel(corpus)

    # build similarity model, using TFIDF of choices
    sims = gensim.similarities.MatrixSimilarity(tfidf[corpus],␣
↪num_features=len(dictionary))


    # for answerString
    # word lemmatize, to lower case, and filter stopword of each choice
    tokenizedAnswer = wordTokenize(answerString.lower())

    # convert to BOW
    bowAnswer = dictionary.doc2bow(tokenizedAnswer)

    # convert to TFIDF
    tfidfAnswer = tfidf[bowAnswer]

    # get similarity, select argmax, return closest sentence index
    return np.argmax(sims[tfidfAnswer])
```

[8]:
```
wnl = WordNetLemmatizer()
stopWords = set(stopwords.words("english"))

def wordTokenize(sentence):
    """
    Word lemmatize, to lower case, and filter stopword of each choice
    Input:
        sentence[String]: a sentence
    Return:
        tokenized word[List]: list of tokenized word
    """
    return [wnl.lemmatize(w.lower()) for w in word_tokenize(sentence) if w not␣
↪in stopWords]
```

```
def getSynonyms(lemmatized):
    """
    Getting sysnonyms of words by NLTK wordnet
    Input:
        lemmatized[List]: list of tokenized word
    Return:
        tempString[String]: joining all synonyms of all tokenized input word␣
 ↪into one string
    """
    tempString = ""
    for word in lemmatized:
        # get synonyms
        synonyms = wordnet.synsets(word)
        lemmas = set(chain.from_iterable([word.lemma_names() for word in␣
 ↪synonyms]))
        if lemmas:
            # if word have synonyms
            tempString += " ".join(lemmas) + " "
        else:
            # if word doesn't have synonyms, e.g. wh-words, stopwords
            tempString += "".join(word) + " "

    return tempString
```

1.4 Main function of getting prediction

[9]:
```
def predictMC(article, question, options, answerSentenceType=""):
    """
    Predict answer of each question
    Inputs:
        article[String]
        question[String]
        options[List]: MC choices
        answerSentenceType[String]: "one"/"multiple", only usable for MCTest
    Return:
        closestOption[Int]: predicted answer in index
    """
    # article
    sentences = sent_tokenize(article)      # from article to list of sentences
    synonymsSentences = []
    for sent in sentences:
        lemmatizedSent = wordTokenize(sent)
        synonymsSent = getSynonyms(lemmatizedSent)
        if synonymsSent:
            # if sentence has synonyms
            synonymsSentences.append(synonymsSent)
```

```
    # question
    lemmatizedQuestion = wordTokenize(question)
    synonymsQuestion = getSynonyms(lemmatizedQuestion)

    # get closest sentence(s) in article given the question
    closestSentenceIndex = getClosestSentence(synonymsSentences,␣
↪synonymsQuestion)
    if (answerSentenceType == "one"):
        # for MCTest, only get 1 sentence if answer sentence type = "one"
        closestSentence = synonymsSentences[closestSentenceIndex]
    else:
        # get 2 sentences before and after and the closest sentence
        closestSentence = synonymsSentences[max(closestSentenceIndex - 2, 0)]
        closestSentence += synonymsSentences[max(closestSentenceIndex - 1, 0)]
        closestSentence += synonymsSentences[closestSentenceIndex]
        closestSentence += synonymsSentences[min(closestSentenceIndex + 1,␣
↪len(synonymsSentences)-1)]
        closestSentence += synonymsSentences[min(closestSentenceIndex + 2,␣
↪len(synonymsSentences)-1)]

    # options
    synonymsOptions = []
    for option in options:
        # for each option
        tempLemmatizedOption = wordTokenize(option)
        tempSynonymsOption = getSynonyms(tempLemmatizedOption)
        synonymsOptions.append(tempSynonymsOption)

    # get closest answer in choices given the closest sentence(s) in article
    closestOption = getClosestSentence(synonymsOptions, closestSentence)

    return closestOption
```

[10]:
```
def predict(dataset):
    """
    Predict answer of whole dataset
    Input:
        dataset[pd.DataFrame]
    Return:
        accuracy info.[Dict]
        dataset[pd.DataFrame]: with the predicted answer as new column
    """
    # foc accuracy calculation purpose
    count = 0
    correct = 0
    wrong = 0
```

```
    for index, row in dataset.iterrows():
        # for each question

        # get MC options of question
        options = row[dataset.columns[dataset.columns.str.
↪startswith('choice')]].tolist()

        # predict answer by article, question, choices
        if "answer sentence type" in row:
            # for MCTest
            predictedAnswer = predictMC(row["article"], row["question"],␣
↪options, row["answer sentence type"])
        else:
            # for non MCTest
            predictedAnswer = predictMC(row["article"], row["question"],␣
↪options)

        # concate predicted answer to dataset
        dataset.loc[index,'predicted answer'] = predictedAnswer

        # for accuracy calculation purpose
        if(row["answer index"] == predictedAnswer):
            correct += 1
        else:
            wrong += 1
        count += 1

        # for executing information
        if count % 100 == 0:
            print(f"correct= {correct}  wrong= {wrong}  count= {count} ␣
↪accuracy= {correct/count}")
    # for executing information
    print(f"correct= {correct}  wrong= {wrong}  count= {count}  accuracy=␣
↪{correct/count}")

    return {"correct":correct, "wrong":wrong, "count":count, "accuracy":␣
↪correct/count}, dataset
```

1.5 Main function (starting point)

[11]:
```
# define dataset
purpose = "Train"
purpose = "Dev"
purpose = "Test"
```

```
dSetList = ["MC160", "MC500", "DREAM", "RACE-middle", "RACE-high"]

accuracy = []
dataset = {}

for dSet in dSetList:
    temp = []
    dataset[dSet] = getDataset(dSet, purpose)
    temp, dataset[dSet] = predict(dataset[dSet])
    temp["dSet"] = dSet
    temp["purpose"] = purpose
    accuracy.append(temp)
```

```
MC160 Test
correct= 48  wrong= 52  count= 100  accuracy= 0.48
correct= 111  wrong= 89  count= 200  accuracy= 0.555
correct= 131  wrong= 109  count= 240  accuracy= 0.5458333333333333
MC500 Test
correct= 53  wrong= 47  count= 100  accuracy= 0.53
correct= 102  wrong= 98  count= 200  accuracy= 0.51
correct= 153  wrong= 147  count= 300  accuracy= 0.51
correct= 203  wrong= 197  count= 400  accuracy= 0.5075
correct= 249  wrong= 251  count= 500  accuracy= 0.498
correct= 307  wrong= 293  count= 600  accuracy= 0.5116666666666667
correct= 307  wrong= 293  count= 600  accuracy= 0.5116666666666667
DREAM Test
correct= 30  wrong= 70  count= 100  accuracy= 0.3
correct= 61  wrong= 139  count= 200  accuracy= 0.305
correct= 95  wrong= 205  count= 300  accuracy= 0.31666666666666665
correct= 142  wrong= 258  count= 400  accuracy= 0.355
correct= 178  wrong= 322  count= 500  accuracy= 0.356
correct= 225  wrong= 375  count= 600  accuracy= 0.375
correct= 274  wrong= 426  count= 700  accuracy= 0.3914285714285714
correct= 313  wrong= 487  count= 800  accuracy= 0.39125
correct= 354  wrong= 546  count= 900  accuracy= 0.3933333333333333
correct= 391  wrong= 609  count= 1000  accuracy= 0.391
correct= 426  wrong= 674  count= 1100  accuracy= 0.38727272727272727
correct= 475  wrong= 725  count= 1200  accuracy= 0.3958333333333333
correct= 506  wrong= 781  count= 1287  accuracy= 0.39316239316239315
RACE-middle Test
correct= 37  wrong= 63  count= 100  accuracy= 0.37
correct= 77  wrong= 123  count= 200  accuracy= 0.385
correct= 114  wrong= 186  count= 300  accuracy= 0.38
correct= 147  wrong= 253  count= 400  accuracy= 0.3675
correct= 181  wrong= 319  count= 500  accuracy= 0.362
correct= 219  wrong= 381  count= 600  accuracy= 0.365
correct= 256  wrong= 444  count= 700  accuracy= 0.3657142857142857
correct= 304  wrong= 496  count= 800  accuracy= 0.38
```

```
correct= 338  wrong= 562  count= 900  accuracy= 0.37555555555555553
correct= 381  wrong= 619  count= 1000  accuracy= 0.381
correct= 410  wrong= 690  count= 1100  accuracy= 0.37272727272727274
correct= 445  wrong= 755  count= 1200  accuracy= 0.37083333333333335
correct= 473  wrong= 827  count= 1300  accuracy= 0.3638461538461538
correct= 511  wrong= 889  count= 1400  accuracy= 0.365
correct= 520  wrong= 916  count= 1436  accuracy= 0.362116991643454
RACE-high Test
correct= 29  wrong= 71  count= 100  accuracy= 0.29
correct= 68  wrong= 132  count= 200  accuracy= 0.34
correct= 98  wrong= 202  count= 300  accuracy= 0.32666666666666666
correct= 125  wrong= 275  count= 400  accuracy= 0.3125
correct= 157  wrong= 343  count= 500  accuracy= 0.314
correct= 185  wrong= 415  count= 600  accuracy= 0.30833333333333335
correct= 219  wrong= 481  count= 700  accuracy= 0.31285714285714283
correct= 246  wrong= 554  count= 800  accuracy= 0.3075
correct= 269  wrong= 631  count= 900  accuracy= 0.29888888888888887
correct= 295  wrong= 705  count= 1000  accuracy= 0.295
correct= 327  wrong= 773  count= 1100  accuracy= 0.2972727272727273
correct= 361  wrong= 839  count= 1200  accuracy= 0.30083333333333334
correct= 392  wrong= 908  count= 1300  accuracy= 0.30153846153846153
correct= 416  wrong= 984  count= 1400  accuracy= 0.29714285714285715
correct= 443  wrong= 1057  count= 1500  accuracy= 0.29533333333333334
correct= 477  wrong= 1123  count= 1600  accuracy= 0.298125
correct= 506  wrong= 1194  count= 1700  accuracy= 0.29764705882352943
correct= 536  wrong= 1264  count= 1800  accuracy= 0.29777777777777775
correct= 560  wrong= 1340  count= 1900  accuracy= 0.29473684210526313
correct= 590  wrong= 1410  count= 2000  accuracy= 0.295
correct= 621  wrong= 1479  count= 2100  accuracy= 0.2957142857142857
correct= 653  wrong= 1547  count= 2200  accuracy= 0.2968181818181818
correct= 688  wrong= 1612  count= 2300  accuracy= 0.2991304347826087
correct= 718  wrong= 1682  count= 2400  accuracy= 0.2991666666666667
correct= 749  wrong= 1751  count= 2500  accuracy= 0.2996
correct= 780  wrong= 1820  count= 2600  accuracy= 0.3
correct= 808  wrong= 1892  count= 2700  accuracy= 0.2992592592592593
correct= 840  wrong= 1960  count= 2800  accuracy= 0.3
correct= 876  wrong= 2024  count= 2900  accuracy= 0.3020689655172414
correct= 911  wrong= 2089  count= 3000  accuracy= 0.30366666666666664
correct= 942  wrong= 2158  count= 3100  accuracy= 0.3038709677419355
correct= 974  wrong= 2226  count= 3200  accuracy= 0.304375
correct= 1007  wrong= 2293  count= 3300  accuracy= 0.3051515151515152
correct= 1046  wrong= 2354  count= 3400  accuracy= 0.3076470588235294
correct= 1078  wrong= 2420  count= 3498  accuracy= 0.3081761006289308
```

1.6 Analysis

[12]:
```
# performance
pd.DataFrame(accuracy).set_index(["dSet", "purpose"])
```

[12]:

dSet	purpose	correct	wrong	count	accuracy
MC160	Test	131	109	240	0.545833
MC500	Test	307	293	600	0.511667
DREAM	Test	506	781	1287	0.393162
RACE-middle	Test	520	916	1436	0.362117
RACE-high	Test	1078	2420	3498	0.308176

[13]:
```
# export result to csv
for ds in dataset:
    dataset[ds].to_csv(f"Stage 1 result/{ds}.csv", index=False)
```

[14]:
```
# for MCTest only
dataset["MC500"][dataset["MC500"]["answer index"] !=␣
 ↪dataset["MC500"]["predicted answer"]].groupby(["answer sentence type"]).
 ↪count()

dataset["MC500"][dataset["MC500"]["answer index"] ==␣
 ↪dataset["MC500"]["predicted answer"]].groupby(["answer sentence type"]).
 ↪count()
```

[14]:

answer sentence type	article	question	choice 0	choice 1	choice 2	choice 3	answer choice	answer index	answer	predicted answer
multiple	152	152	152	152	152	152	152	152	152	152
one	155	155	155	155	155	155	155	155	155	155

[15]:
```
dataset["MC500"]
```

[15]:

	article
0	It was Sally's birthday. She was very excited…
1	It was Sally's birthday. She was very excited…
2	It was Sally's birthday. She was very excited…
3	It was Sally's birthday. She was very excited…

```
4    On the farm there was a little piggy named And…
..                                                 …
595  Greg and his mother were building a racing car…
596  Joey went to a baseball game during the winter…
597  Joey went to a baseball game during the winter…
598  Joey went to a baseball game during the winter…
599  Joey went to a baseball game during the winter…

                                          question answer sentence type  \
0                      What time did the party start?                one
1                          Who got hurt at the party?             multiple
2                               Whose birthday is it?                one
3           What time did Jennifer arrive to the party?             multiple
4     What did the piggies do when Andy got back fr…             multiple
..                                                 …                  …
595                      Where was the race happening?             multiple
596   Who went to the baseball game and with how ma…             multiple
597             what kind of store did Joey turn into?                one
598   Which team won the game Joey went to and by h…             multiple
599                      What dessert did Joey choose?                one

                       choice 0                          choice 1  \
0                            10                                 2
1               Erin and Jennifer                  Cathy and Erin
2                           Cathy                           Jessica
3                               1                                 2
4      play games and eat dinner  play in the mud and go for a walk
..                            …                                 …
595                 At the park.      On the track near his school.
596                Joey, nobody.                Mark, nobody
597                 Garden store                Grocery store
598     Home team, by two runs.            Away team, by one run
599  warm brownie with ice cream                Vanilla shake

                             choice 2                          choice 3  \
0                                  11                                 1
1                  Jennifer and Sally                 Erin and Sally
2                               Sally                           Jennifer
3                                   8                                10
4    swim in the river and play games  go for a walk and look at flowers
..                                  …                                 …
595                        In a river.               In their backyard.
596                    Sam, two others              Joey, three others.
597                          Car store                     Coffee store
598           Away team, by two runs.            Home team, by one run.
599          apple pie with ice cream     Marshmallow and chocolate cake
```

```
    answer choice  answer index                          answer  \
0               D             3                               1
1               C             2             Jennifer and Sally
2               C             2                           Sally
3               B             1                               2
4               A             0     play games and eat dinner
..            …           …                                 …
595             B             1  On the track near his school.
596             A             0                  Joey, nobody.
597             D             3                   Coffee store
598             D             3        Home team, by one run.
599             D             3  Marshmallow and chocolate cake

     predicted answer
0                 0.0
1                 3.0
2                 2.0
3                 0.0
4                 3.0
..                …
595               1.0
596               1.0
597               3.0
598               3.0
599               0.0

[600 rows x 11 columns]
```

[ ]: