Technical Notebook

Google Quest Q&A Labeling: Kaggle Competition

Mary Wall, January 15, 2020

In [36]:
```
import pandas as pd
import csv
import sys
from time import time

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from statsmodels.distributions.empirical_distribution import ECDF

import nltk
from nltk.tokenize import sent_tokenize, word_tokenize

from functools import reduce, partial
```

DATA

- Google Quest Q&A Labeling train and test data from https://www.kaggle.com/c/google-quest-challenge (https://www.kaggle.com/c/google-quest-challenge)
- Pre - trained word vectors, GloVe Data, publicly available through Stanford NLP Group Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

```
https://nlp.stanford.edu/projects/glove/
```

In [2]:
```
# read in Data
train = pd.read_csv('../google_stuff/data/google_quest/train.csv')
train = train.set_index('qa_id', drop=True)

test = pd.read_csv('../google_stuff/data/google_quest/test.csv')

sample_sub = pd.read_csv('../google_stuff/data/google_quest/sample_sub
mission.csv')

df_glove = pd.read_csv('../google_stuff/data/glove/glove.6B.50d.txt',
sep=' ', engine='python',
                 header=None, names=['word'] + list(range(1, 51)),
                 quoting=csv.QUOTE_NONE)
# set the index of the glove set by the word
df_glove = df_glove.set_index('word', drop=True)
```

Problem Statement

To match human classifications of google queries, using AI. The predictions are made as a percentage chance of having a particular tag(s) such as ‘conversational’, ‘fast speaking’, ‘having a commonly accepted answer’... and so on. Training an effective model will inform the way Q&A systems will be built at Google.

Features and Targets

- The features are the question title and answer, as well as question ID, Q&A user names, Q&A user page, url, category and host.
- There are 30 target related to the question and answer features.

Train Data Frame

There are 6079 question/answer pairings that have been tagged. The features used for modeling are: 'question_title', 'question_body', 'answer' The 30 targets are 'question_well_written' 'answer_helpful' 'answer_level_of_information' 'answer_plausible' 'answer_relevance' 'answer_satisfaction' 'answer_type_instructions' 'answer_type_procedure' 'answer_type_reason_explanation' and so on

In [3]:
```
def explore(df):
    print(df.shape)
    return df.head()

explore(train)
```

```
(6079, 40)
```

Out[3]:

qa_id	question_title	question_body	question_user_name	question_use
0	What am I losing when using extension tubes in...	After playing around with macro photography on...	ysap	https://photo.stackexchange.com/use
1	What is the distinction between a city and a s...	I am trying to understand what kinds of places...	russellpierce	https://rpg.stackexchange.com/use
2	Maximum protusion length for through-hole comp...	I'm working on a PCB that has through-hole com...	Joe Baker	https://electronics.stackexchange.com/user:
3	Can an affidavit be used in Beit Din?	An affidavit, from what i understand, is basic...	Scimonster	https://judaism.stackexchange.com/use
5	How do you make a binary image in Photoshop?	I am trying to make a binary image. I want mor...	leigero	https://graphicdesign.stackexchange.com/

5 rows × 40 columns

Testing Set

There are 476 question/answer pairings in the test data. The model will be trained using the train data frame and ultimately used to generate predictions of the targets for the test data. These predictions will be scored upon submission to the Kaggle competition.

In [4]:
```
explore(test)
```

```
(476, 11)
```

Out[4]:

	qa_id	question_title	question_body	question_user_name	que
0	39	Will leaving corpses lying around upset my pri...	I see questions/information online about how t...	Dylan	https://gaming.stackexchange
1	46	Url link to feature image in the portfolio	I am new to Wordpress. i have issue with Featu...	Anu	https://wordpress.stackexchange
2	70	Is accuracy, recoil or bullet spread affected ...	To experiment I started a bot game, toggled in...	Konsta	https://gaming.stackexchange
3	132	Suddenly got an I/O error from my external HDD	I have used my Raspberry Pi as a torrent-serve...	robbannn	https://raspberrypi.stackexchange
4	200	Passenger Name - Flight Booking Passenger only...	I have bought Delhi-London return flights for ...	Amit	https://travel.stackexchange

Sample submission example for kaggle competition

In [5]:
```
explore(sample_sub)
```

```
(476, 31)
```

Out[5]:

	qa_id	question_asker_intent_understanding	question_body_critical	question_conversational	qu
0	39	0.00308	0.00308	0.00308	
1	46	0.00448	0.00448	0.00448	
2	70	0.00673	0.00673	0.00673	
3	132	0.01401	0.01401	0.01401	
4	200	0.02074	0.02074	0.02074	

5 rows × 31 columns

Glove Data

Associates a 50 dimensional vector with text (400,000 tokens).
Disadvantage:

- model outputs only one vector embedding per word regardless of word sense.
- GloVE does not use neural networks - it is a log bilinear model.
- There are larger GLoVE data sets than this one.

In [6]:
```
explore(df_glove)
```

```
(400000, 50)
```

Out[6]:

word	1	2	3	4	5	6	7	8	9	
the	0.418000	0.249680	-0.41242	0.12170	0.34527	-0.044457	-0.49688	-0.17862	-0.00066	·
,	0.013441	0.236820	-0.16899	0.40951	0.63812	0.477090	-0.42852	-0.55641	-0.36400	·
.	0.151640	0.301770	-0.16763	0.17684	0.31719	0.339730	-0.43478	-0.31086	-0.44999	·
of	0.708530	0.570880	-0.47160	0.18048	0.54449	0.726030	0.18157	-0.52393	0.10381	·
to	0.680470	-0.039263	0.30186	-0.17792	0.42962	0.032246	-0.41376	0.13228	-0.29847	·

5 rows × 50 columns

Initial Goal

To develop a model that can predict the correct tag greater than 50% of the time, and thus better than random guessing.

Approach and Process

- Obtain vector embeddings for words in the predictor columns.
- Tokenize each document and assign tokens vector embedding via = glove data.
$$\text{token} \mapsto v$$
- Assign each document to a vector sum over all words.
$$\text{document} \longmapsto \sum_{i=1}^{n} v_i = V_{document}$$
where n is the number of tokens in each document.
- Use the vector embeddings for each document to create a machine learning model.
- Use the model on the test data to predict targets and compare.

A note on encoding algorithm

The initial approach to encoding an individual document was computationally costly. An adjusted approach follows, taking into acount the following:

- there are 87,203 unique words in the train set, and 400,000 words in the glove data set, but only 26,163 of the train words also appear in the glove data set. To speed up computation, all words from the glove set that are not also in the train set are dropped
- associate words with an integer, so that the document encodings do not require a string comparison.

get_all_words function

- tokenizes each document of the column
- creates a set of all tokens from each document in a column
- this will be applied to the question body/title and answer text

In [7]:
```
# function used to create a set of words from each predictive column
def get_all_words(column, df):
    tokenize = lambda x: word_tokenize(x)
    tokenized = [tokenize(x) for x in df[column]]
    set_collect = lambda x, y: x.union(y)
    all_words = reduce(set_collect, tokenized[1:], set(tokenized[0]))
    return all_words
```

get_intersection_words function

- finds the intersection of the words from the predictive columns in the train data set with the glove data.
- the goal is to reduce the size of the glove data set making the document encoding algorithm run faster.

In [8]:
```
def get_intersection_words(column, df, df_glove):
    tokenize = lambda x: word_tokenize(x)
    tokenized = [tokenize(x) for x in df[column]]
    set_collect = lambda x, y: x.union(y)
    all_words = reduce(set_collect, tokenized[1:], set(tokenized[0]))
    glove_words = set(df_glove.index)
    intersection_words = glove_words.intersection(all_words)
    return intersection_words
```

encode_column

- associates each token in the intersection of glove and train documents with an integer
- pairs vector embeddings from glove data with tokens from documents, avoiding computational string comparisons

In [9]:
```
def encode_column(col_name, df, df_glove):
    intersection_words = get_intersection_words(col_name, df, df_glov
e)
    indx = pd.Index(list(intersection_words))
    df_ = df_glove.loc[indx]

    df2 = df_.reset_index()
    df3 = df2.reset_index()
    word_indexed_vecs = df3.set_index('word', drop=True)
    int_indexed_vecs = df3.set_index('index', drop=True)

    parse = lambda S: [x.lower() for x in word_tokenize(S) if x.lower
() in indx]
    X = df[col_name].apply(parse)

    coded = [[word_indexed_vecs.loc[x, 'index'] for x in E] for E in
X]
# if there are no common words between a document and the glove set,
returns 0's for the sum
# embed will return a 50-d vector for every document which is a sum on
the tokenized word to vector
# representations for those words that appear in glove
    vector_cols = list(range(1, 51))
    embed = lambda C: np.sum([np.zeros((50,))] +
                             [int_indexed_vecs.loc[i, vector_cols].val
ues for i in C], axis=0)

    data = np.vstack([embed(C) for C in coded])
    return pd.DataFrame(data, columns=vector_cols, index=df.index)
```

Time to embed documents as vectors

- encode documents in question body - 13 minutes
- encode documents in answer column - 13 minutes
- question title - 5 minutes

Previous approach - around 20 seconds per document

around 4.2 days!!!

In [10]:
```
(20*6079*3)/60/60/24
```

Out[10]:
```
4.221527777777777
```

Pickle Files

- all the vector embeddings for the train and test data were stored as pickle
- better format for Jupyter notebooks than csv (to see why unpickle and store as csv to compare which file size is larger)

In [14]:
```
embedded_titles = pd.read_pickle('pickle_files/embedded_titles.txt')
embedded_qbody = pd.read_pickle('pickle_files/embedded_qbody.txt')
embedded_answer = pd.read_pickle('pickle_files/embedded_answer.txt')
embedded_titles_test = pd.read_pickle('pickle_files/embedded_titles_te
st.txt')
embedded_qbody_test = pd.read_pickle('pickle_files/embedded_qbody_tes
t.txt')
embedded_answer_test = pd.read_pickle('pickle_files/embedded_answer_te
st.txt')
```

In [15]:
```
embedded_titles.to_csv('../google_stuff/data/embedding/csv/embedded_ti
tles.csv')
embedded_qbody.to_csv('../google_stuff/data/embedding/csv/embedded_qbo
dy.csv')
embedded_answer.to_csv('../google_stuff/data/embedding/csv/embedded_an
swer.csv')
embedded_titles_test.to_csv('../google_stuff/data/embedding/csv/embedd
ed_titles_test.csv')
embedded_qbody_test.to_csv('../google_stuff/data/embedding/csv/embedde
d_qbody_test.csv')
embedded_answer_test.to_csv('../google_stuff/data/embedding/csv/embedd
ed_answer_test.csv')
```

Example of question body vector embedding

- there are 6079 vector embeddings for each question body document.
- each vector is 50 dimensions
- the data frame is indexed by the question ID

In [16]:
```
explore(embedded_qbody)
```

```
(6079, 50)
```

Out[16]:

	1	2	3	4	5	6	7	8	9	
qa_id										
0	42.237	38.196	7.83203	-10.1146	40.6123	28.4822	-27.9094	-50.4085	-37.0975	-0.0:
1	37.0605	39.4014	-1.91795	-1.36748	60.977	20.6288	-43.4605	-50.595	-29.0595	-1
2	30.8807	14.1982	3.36902	-0.996975	44.7958	20.3453	-27.5108	-23.6854	-7.94469	-1
3	24.5323	12.6471	-6.27443	-3.86758	39.2465	20.9778	-25.0769	-6.75594	-11.9258	-3
5	20.3295	13.8852	6.92127	-5.5858	50.8067	12.8106	-19.8476	-22.835	-29.8199	1

5 rows × 50 columns

Model or Solution

The chosen model is linear regression using scikit - learn.

Import necessary models from scikit - learn

In [17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from itertools import product
```

Instantiate the model

In [18]:
```
model = LinearRegression()
```

Create X used in model.fit

- matrix of vector embeddings for the documents in question title/body and answer.
- each row contains three 50 dimensional vectors assigned to the title, body, and answer documents.

Collect feature data

In [19]:
```
data_array = np.concatenate([df.values for df in [embedded_titles, emb
edded_qbody, embedded_answer]], axis=1)
data_array
```

Out[19]:
```
array([[3.96663, 1.9142890000000001, 2.512292, ..., -19.39725360000001
7,
        -4.532179000000004, -4.770620400000003],
       [4.291580000000001, 6.520906400000001, -5.150425300000001, ...,
        -4.5680166, -3.2810134000000004, 1.4268052],
       [-0.19085200000000002, 2.0161499999999997, 2.74239, ...,
        -6.18466658, -6.326489999999997, -12.818838699999992],
       ...,
       [2.51028, -0.8507399999999999, 2.172953, ..., 3.922249999999998
7,
        2.1421360000000003, 9.783244999999997],
       [0.7699699999999998, 1.0116999999999998, -1.2954444, ...,
        -0.3164540000000049, -0.35453180000000106, 8.984216600000003],
       [3.57063, 3.47931, 1.2967939999999998, ..., -16.45439359999999
2,
        -7.636990999999994, 2.4526999999999908]], dtype=object)
```

Create column names for X

- T for question_title vector
- Q for question_body vector
- A for answer vector

In [20]:
```
explore(embedded_titles_test)
```

(476, 50)

Out[20]:

	1	2	3	4	5	6	7	8	9	
0	3.25036	1.08633	2.43576	-3.61049	4.85592	-1.53572	-2.37061	4.3371	0.005414	-3.!
1	3.13794	2.81698	0.39886	1.5972	2.2177	1.09869	-2.93891	-4.25514	0.205903	-0.0!
2	1.80895	-1.17933	4.56927	-2.36318	2.43233	6.0379	-0.184906	-2.55638	1.29049	0.5(
3	0.535785	0.62386	1.37852	-0.625634	2.86958	2.25575	-1.53351	0.67607	0.05376	-0.0:
4	1.31296	3.57585	-0.29093	0.80167	2.13275	1.97692	-6.47401	-2.15394	-1.41725	-1..

5 rows × 50 columns

In [21]:
```
col_names = [c + '_' + str(y).zfill(2) for c, y in product(['T', 'Q',
'A'], range(50))]
print(col_names)
```

```
['T_00', 'T_01', 'T_02', 'T_03', 'T_04', 'T_05', 'T_06', 'T_07', 'T_0
8', 'T_09', 'T_10', 'T_11', 'T_12', 'T_13', 'T_14', 'T_15', 'T_16', 'T
_17', 'T_18', 'T_19', 'T_20', 'T_21', 'T_22', 'T_23', 'T_24', 'T_25',
'T_26', 'T_27', 'T_28', 'T_29', 'T_30', 'T_31', 'T_32', 'T_33', 'T_3
4', 'T_35', 'T_36', 'T_37', 'T_38', 'T_39', 'T_40', 'T_41', 'T_42', 'T
_43', 'T_44', 'T_45', 'T_46', 'T_47', 'T_48', 'T_49', 'Q_00', 'Q_01',
'Q_02', 'Q_03', 'Q_04', 'Q_05', 'Q_06', 'Q_07', 'Q_08', 'Q_09', 'Q_1
0', 'Q_11', 'Q_12', 'Q_13', 'Q_14', 'Q_15', 'Q_16', 'Q_17', 'Q_18', 'Q
_19', 'Q_20', 'Q_21', 'Q_22', 'Q_23', 'Q_24', 'Q_25', 'Q_26', 'Q_27',
'Q_28', 'Q_29', 'Q_30', 'Q_31', 'Q_32', 'Q_33', 'Q_34', 'Q_35', 'Q_3
6', 'Q_37', 'Q_38', 'Q_39', 'Q_40', 'Q_41', 'Q_42', 'Q_43', 'Q_44', 'Q
_45', 'Q_46', 'Q_47', 'Q_48', 'Q_49', 'A_00', 'A_01', 'A_02', 'A_03',
'A_04', 'A_05', 'A_06', 'A_07', 'A_08', 'A_09', 'A_10', 'A_11', 'A_1
2', 'A_13', 'A_14', 'A_15', 'A_16', 'A_17', 'A_18', 'A_19', 'A_20', 'A
_21', 'A_22', 'A_23', 'A_24', 'A_25', 'A_26', 'A_27', 'A_28', 'A_29',
'A_30', 'A_31', 'A_32', 'A_33', 'A_34', 'A_35', 'A_36', 'A_37', 'A_3
8', 'A_39', 'A_40', 'A_41', 'A_42', 'A_43', 'A_44', 'A_45', 'A_46', 'A
_47', 'A_48', 'A_49']
```

Create Data Frame of embeddings with column names

- has 150 columns, to include the three 50 dimensional vector - document assignments
- has 6079 rows to to reflect the train data

In [22]:
```
X = pd.DataFrame(data_array, columns=col_names)
X.head()
```

Out[22]:

	T_00	T_01	T_02	T_03	T_04	T_05	T_06	T_07	T_08	
0	3.96663	1.91429	2.51229	-3.79679	2.6914	2.1388	-1.04762	-3.47599	-4.91889	
1	4.29158	6.52091	-5.15043	1.03506	6.50218	2.97328	-6.58223	-3.45823	-2.26175	-
2	-0.190852	2.01615	2.74239	-0.402239	0.953425	2.35235	3.80058	-2.84904	2.70342	-
3	3.02994	1.1532	0.377555	0.027771	2.61459	1.88286	-2.3593	-1.28309	-0.586269	0
4	2.78256	1.74746	1.89248	-0.0948963	4.0811	0.512911	-2.48701	-4.15973	-3.77595	

5 rows × 150 columns

Collect predictive test vector embeddings to dataframe, X_test

- index by question ID from the test data

In [23]:
```
data_array = np.concatenate([df.values for df in [embedded_titles_tes
t, embedded_qbody_test, embedded_answer_test]], axis=1)
data_array
X_test = pd.DataFrame(data_array, columns=col_names)
X_test['qa_id'] = test['qa_id']
X_test = X_test.set_index('qa_id', drop = True)
print(X_test.shape)
X_test.head()
```

```
(476, 150)
```

Out[23]:

qa_id	T_00	T_01	T_02	T_03	T_04	T_05	T_06	T_07	T_08
39	3.25036	1.08633	2.43576	-3.61049	4.85592	-1.53572	-2.37061	4.3371	0.005414
46	3.13794	2.81698	0.39886	1.5972	2.2177	1.09869	-2.93891	-4.25514	0.205903
70	1.80895	-1.17933	4.56927	-2.36318	2.43233	6.0379	-0.184906	-2.55638	1.29049
132	0.535785	0.62386	1.37852	-0.625634	2.86958	2.25575	-1.53351	0.67607	0.05376
200	1.31296	3.57585	-0.29093	0.80167	2.13275	1.97692	-6.47401	-2.15394	-1.41725

5 rows × 150 columns

Predict the tag targets for the test data

In [24]:
```
# create a list of the target column used to name
targets = [c for c in train.columns if train[c].dtype.kind == 'f']
frames = []
series = []

for i in range(0,30):
    frames.append(model.fit(X, train.loc[:, targets[i]]))
    frames[0].predict(X_test)
    series.append(pd.DataFrame(frames[i].predict(X_test), columns = [t
argets[i]]))
    sub = pd.concat(series,axis = 1)
    sub_clean=sub._get_numeric_data()
    sub_clean[sub_clean<0]=0  # Replace all negative values with 0's
    sub_clean.insert(0,'qa_id', value = test['qa_id'], allow_duplicate
s = True)
```

In [25]:
```
explore(sub_clean)
```

```
(476, 31)
```

Out[25]:

	qa_id	question_asker_intent_understanding	question_body_critical	question_conversational	qu
0	39	0.947582	0.550640	0.131618	
1	46	0.872197	0.574743	0.080178	
2	70	0.882917	0.530331	0.015422	
3	132	0.850352	0.196338	0.000000	
4	200	0.905601	0.631622	0.079933	

5 rows × 31 columns

Save to csv to match format of Kaggle sample submission - and then score!

In [26]:
```
#sub_clean.to_csv('../google_stuff/data/google_quest/submission_linre
g.csv')
```

Performance Evaluation

In [27]:
```
y = train.loc[:, targets]
```

In [28]:
```
y = train.loc[:, targets]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.
3)

scores_train = {}
scores_test = {}
models_tts = []

for t in targets:
    target = y_train.loc[:, t]
    model.fit(X_train, target)
    scores_train[t] = model.score(X_train, y_train.loc[:, t])
    score = model.score(X_test, y_test.loc[:, t])
    scores_test[t] = score
```

In [29]:
```
scores_train = {}
scores_test = {}
for t in targets:
    target = y_train.loc[:, t]
    model.fit(X_train, target)
    scores_train[t] = model.score(X_train, y_train.loc[:, t])
    score = model.score(X_test, y_test.loc[:, t])
    scores_test[t] = score

high_scores_train = {x: y for x, y in scores_train.items() if y > .1}
high_scores_test = {x: y for x, y in scores_test.items() if y > .1}
display(high_scores_train)
print()
display(high_scores_test)
```

```
{'question_asker_intent_understanding': 0.11634284587810508,
 'question_body_critical': 0.3455165850384759,
 'question_conversational': 0.17328828650657524,
 'question_fact_seeking': 0.14020459659056317,
 'question_has_commonly_accepted_answer': 0.200421809394006,
 'question_interestingness_others': 0.12004953073970827,
 'question_interestingness_self': 0.19640571563416553,
 'question_multi_intent': 0.14305264343392854,
 'question_opinion_seeking': 0.14929443855821756,
 'question_type_choice': 0.12608686522209445,
 'question_type_compare': 0.11636524716060026,
 'question_type_definition': 0.1855655724430426,
 'question_type_entity': 0.11822952279709531,
 'question_type_instructions': 0.35460254373598415,
 'question_type_reason_explanation': 0.20743935486165288,
 'question_well_written': 0.2284374889154016,
 'answer_level_of_information': 0.18096003428614005,
 'answer_type_instructions': 0.35226657554801166,
 'answer_type_reason_explanation': 0.23406940226327289}


{'question_body_critical': 0.2791184580809295,
 'question_conversational': 0.1096496015161621,
 'question_has_commonly_accepted_answer': 0.1291698140182299,
 'question_interestingness_self': 0.18586804012224356,
 'question_type_instructions': 0.28716177403103726,
 'question_type_reason_explanation': 0.11470618773625896,
 'question_well_written': 0.1482104417551393,
 'answer_level_of_information': 0.12144588259741818,
 'answer_type_instructions': 0.25164495736698467,
 'answer_type_reason_explanation': 0.15400541538361745}
```

Model performance: model performs best on the following targets (as seen from scores above):

- question_type_instructions
- answer_type_instructions
- question_body_critical

In some cases it random guessing may even have produced less error in target prediction.

In [30]:
```
high_scores_train = {x: y for x, y in scores_train.items() if y > .1}
high_scores_test = {x: y for x, y in scores_test.items() if y > .1}
display(high_scores_train)
print()
display(high_scores_test)
```

```
{'question_asker_intent_understanding': 0.11634284587810508,
 'question_body_critical': 0.3455165850384759,
 'question_conversational': 0.17328828650657524,
 'question_fact_seeking': 0.14020459659056317,
 'question_has_commonly_accepted_answer': 0.200421809394006,
 'question_interestingness_others': 0.12004953073970827,
 'question_interestingness_self': 0.19640571563416553,
 'question_multi_intent': 0.14305264343392854,
 'question_opinion_seeking': 0.14929443855821756,
 'question_type_choice': 0.12608686522209445,
 'question_type_compare': 0.11636524716060026,
 'question_type_definition': 0.1855655724430426,
 'question_type_entity': 0.11822952279709531,
 'question_type_instructions': 0.35460254373598415,
 'question_type_reason_explanation': 0.20743935486165288,
 'question_well_written': 0.2284374889154016,
 'answer_level_of_information': 0.18096003428614005,
 'answer_type_instructions': 0.35226657554801166,
 'answer_type_reason_explanation': 0.23406940226327289}

{'question_body_critical': 0.2791184580809295,
 'question_conversational': 0.1096496015161621,
 'question_has_commonly_accepted_answer': 0.1291698140182299,
 'question_interestingness_self': 0.18586804012224356,
 'question_type_instructions': 0.28716177403103726,
 'question_type_reason_explanation': 0.11470618773625896,
 'question_well_written': 0.1482104417551393,
 'answer_level_of_information': 0.12144588259741818,
 'answer_type_instructions': 0.25164495736698467,
 'answer_type_reason_explanation': 0.15400541538361745}
```

In [31]:
```
scores = {}
for t in targets:
    print(t)
    y = train.loc[:, t]
    model.fit(X, y)
    score = model.score(X, y)
    scores[t] = score
```

```
question_asker_intent_understanding
question_body_critical
question_conversational
question_expect_short_answer
question_fact_seeking
question_has_commonly_accepted_answer
question_interestingness_others
question_interestingness_self
question_multi_intent
question_not_really_a_question
question_opinion_seeking
question_type_choice
question_type_compare
question_type_consequence
question_type_definition
question_type_entity
question_type_instructions
question_type_procedure
question_type_reason_explanation
question_type_spelling
question_well_written
answer_helpful
answer_level_of_information
answer_plausible
answer_relevance
answer_satisfaction
answer_type_instructions
answer_type_procedure
answer_type_reason_explanation
answer_well_written
```

In [32]:
```
scores
```

Out[32]:
```
{'question_asker_intent_understanding': 0.1069451654854725,
 'question_body_critical': 0.33354545013345616,
 'question_conversational': 0.1625979603707105,
 'question_expect_short_answer': 0.08120119245926949,
 'question_fact_seeking': 0.12718961353174818,
 'question_has_commonly_accepted_answer': 0.1893239553065259,
 'question_interestingness_others': 0.11826741173052968,
 'question_interestingness_self': 0.20028703032875772,
 'question_multi_intent': 0.1311527452507083,
 'question_not_really_a_question': 0.01716573339678873,
 'question_opinion_seeking': 0.13923828062296495,
 'question_type_choice': 0.1198745484713245,
 'question_type_compare': 0.11029572161084966,
 'question_type_consequence': 0.048489432515462116,
 'question_type_definition': 0.17127916578136604,
 'question_type_entity': 0.10950275085402128,
 'question_type_instructions': 0.34304463993327305,
 'question_type_procedure': 0.0796901902303112,
 'question_type_reason_explanation': 0.19159150991307727,
 'question_type_spelling': 0.034133631049696245,
 'question_well_written': 0.21535752752572612,
 'answer_helpful': 0.050293646390362705,
 'answer_level_of_information': 0.17315952860748063,
 'answer_plausible': 0.03248974134338045,
 'answer_relevance': 0.03386901161107181,
 'answer_satisfaction': 0.07176583387437263,
 'answer_type_instructions': 0.3335244408386101,
 'answer_type_procedure': 0.07135019982762736,
 'answer_type_reason_explanation': 0.22074438809723418,
 'answer_well_written': 0.040650841206713584}
```

Competition Results

- 939 participants in the competition
- top 50 scores range from .398 to .456, (as of this writing)

Recommendations or next steps

Use a larger GLoVE corpus of pre-trained word vectors. If we consider the GloVE data we see that around 30% (see calculation below) of words in train documents can be found in the GloVe Data. Using a larger corpus might increase the accuracy of each document - vector representation, by including more tokens in the vector sum. Probably would be best to run on Amazon Web Services Cloud Computing.

In [33]:
```
# check how many unique words in the train set, and how many of those
words appear in the GloVE set
embed_cols = ['question_title', 'question_body', 'answer']
AIW = [get_intersection_words(c, train, df_glove) for c in embed_cols]
all_words = [get_all_words(c, train) for c in embed_cols]

all_intersection_words = reduce(lambda x, y: x.union(y), AIW, set())
print(len(all_intersection_words))
all_words = reduce(lambda x, y: x.union(y), all_words, set())
print(len(all_words))
```

```
26111
87203
```

In [34]:
```
26163 / 87203
```

Out[34]:
```
0.3000240817403071
```

Some additional considerations to improve the model:

- use a larger GLoVE corpus. (the smallest, 400,000 tokens, was used to construct embeddings)
- utilize different models - such as multi - layer perceptron, a supervised algorithm that learns a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- scrape the url where the answer appeared to see how it was rated, as well as any available info on how a user was rated.
- GloVE assigns a vector embedding for a word regardless of the context or position, may consider using a language model trained to differentiate word sense, such as BERT, a neural network and natural language process.
- may want to train a model to predict the question targets and another to predict the answer targets.

DEMO used in non - technical presentation

In [35]:
```
# DEMO for Slideshow
# create a list of the target column used to name
targets = [c for c in train.columns if train[c].dtype.kind == 'f']
targets

print(targets[0])
y = train.loc[:, targets[0]]
print(y)
type(y)

model.fit(X, y)

def intslope(X, y):
    model.fit(X,y)
    print("intercept", model.intercept_)
    print("coefficient", model.coef_)
    print("score", model.score(X,y))
   # print("linear model y =", int(model.coef_),"*x +",int(model.inter
cept_) )

intslope(X,y)
len(model.coef_)
# We see there are 150 coefficients. Each coefficient relates to one e
ntry of the of the 50 - d question title, question body
# and answer respectively.


print("intercept", model.intercept_)
print("coefficient", model.coef_)
print("score", model.score(X,y))

type(model.predict(X_test))
prediction = pd.DataFrame(model.predict(X_test), columns = [targets
[0]])
prediction.head()
type(prediction)

ans = model.predict(X_test)
print(ans.ndim)
type(ans)
#DEMO
print(targets[29])
```

```
question_asker_intent_understanding
qa_id
0       1.000000
1       1.000000
2       0.888889
3       0.888889
5       1.000000
          ...
9642    1.000000
9643    1.000000
9645    0.888889
9646    1.000000
9647    1.000000
Name: question_asker_intent_understanding, Length: 6079, dtype: float6
4
intercept 0.8709810138764886
coefficient [ 9.91000430e-03  1.27362163e-02 -1.03625186e-02 -2.719741
76e-03
 -1.43207843e-03 -6.27371284e-03 -6.02176139e-03 -6.70799185e-03
 -7.64205334e-03  1.43710262e-03  1.85641078e-02 -1.37014337e-02
 -7.33036704e-03 -2.96590095e-03 -1.03066391e-02 -6.76772350e-03
 -9.53619905e-04  2.21712969e-03  5.66342488e-03 -8.45638750e-03
  1.00244618e-02 -4.47949396e-03  1.34485772e-02 -7.64479124e-04
  8.29345313e-03 -1.14446069e-02  3.18597510e-03 -7.76451756e-03
  8.09693375e-03 -2.39134227e-03 -7.06655309e-03 -5.16489002e-03
  4.24773102e-03  8.85207551e-03 -7.02336891e-03  7.70238776e-03
  6.37179514e-03 -6.25148053e-03  1.14810271e-03 -1.37785532e-02
 -1.19258270e-02  5.99762182e-03  2.26673138e-03  4.00492762e-03
 -3.39234026e-03  3.77556438e-03  1.28833476e-04  6.82013632e-03
  3.34348914e-03  2.45734426e-03 -2.42159020e-04 -5.45533395e-04
  2.04558312e-04 -6.81798903e-05 -2.45291497e-04 -2.42599038e-04
  8.28149740e-04  2.80439662e-05  4.33384271e-04  1.64988302e-04
 -5.36619243e-04  8.78899501e-05  9.23202770e-04  3.46346097e-04
  1.08700325e-03  4.85829118e-04 -1.20369454e-04 -5.80841712e-04
 -6.67463384e-04 -2.03338189e-04 -7.94651553e-04  1.12723759e-03
 -2.52189633e-04  2.59126264e-04 -9.08678661e-04 -1.28987768e-04
 -2.42093124e-04 -2.96917818e-04 -1.33046035e-04 -2.03306192e-04
  6.95986392e-05  1.28525858e-04  4.37379410e-04 -1.58047326e-04
  4.44059744e-04  5.89974850e-04 -3.20137675e-04  4.60829920e-04
  4.26068562e-04  6.29118049e-04  5.34398723e-04 -9.90187747e-04
 -6.64159140e-04 -6.47406387e-04  3.17824314e-04 -4.01732730e-04
 -2.20877231e-04 -2.66494084e-04 -1.98072438e-05 -1.44313092e-04
 -2.75986596e-04  1.95115282e-04  7.56208939e-04 -5.26240129e-04
  2.95085646e-05  3.39926059e-04  2.84139809e-04 -1.59204808e-04
  7.91428620e-05  3.86672042e-04  2.35158361e-04  2.98336163e-04
 -6.99389522e-04 -1.23699535e-03 -7.47207019e-04 -2.42960355e-04
 -7.29603002e-04 -1.00950039e-03 -9.52246300e-04  6.19831608e-04
  8.07335954e-04 -7.39634349e-04 -5.06219933e-04  1.47167171e-03
  1.34353546e-03  1.14401055e-03 -2.85491766e-04  1.58039285e-03
 -3.36848800e-04  2.14689048e-04  3.10697165e-04 -1.51169975e-05
 -1.89276074e-03  2.47806115e-04 -2.72522652e-04  8.96731569e-04
 -7.53012956e-04  1.60275248e-04  1.45240579e-04 -4.98763328e-04
 -5.63853986e-04  7.19549251e-04  4.18144998e-04  6.52132052e-04
  5.18994646e-04 -3.07857037e-04 -1.43609418e-03 -1.60520304e-04
 -1.11527065e-03  1.90148419e-04]
score 0.1069451654854725
intercept 0.8709810138764886
```

```
coefficient [ 9.91000430e-03  1.27362163e-02 -1.03625186e-02 -2.719741
76e-03
 -1.43207843e-03 -6.27371284e-03 -6.02176139e-03 -6.70799185e-03
 -7.64205334e-03  1.43710262e-03  1.85641078e-02 -1.37014337e-02
 -7.33036704e-03 -2.96590095e-03 -1.03066391e-02 -6.76772350e-03
 -9.53619905e-04  2.21712969e-03  5.66342488e-03 -8.45638750e-03
  1.00244618e-02 -4.47949396e-03  1.34485772e-02 -7.64479124e-04
  8.29345313e-03 -1.14446069e-02  3.18597510e-03 -7.76451756e-03
  8.09693375e-03 -2.39134227e-03 -7.06655309e-03 -5.16489002e-03
  4.24773102e-03  8.85207551e-03 -7.02336891e-03  7.70238776e-03
  6.37179514e-03 -6.25148053e-03  1.14810271e-03 -1.37785532e-02
 -1.19258270e-02  5.99762182e-03  2.26673138e-03  4.00492762e-03
 -3.39234026e-03  3.77556438e-03  1.28833476e-04  6.82013632e-03
  3.34348914e-03  2.45734426e-03 -2.42159020e-04 -5.45533395e-04
  2.04558312e-04 -6.81798903e-05 -2.45291497e-04 -2.42599038e-04
  8.28149740e-04  2.80439662e-05  4.33384271e-04  1.64988302e-04
 -5.36619243e-04  8.78899501e-05  9.23202770e-04  3.46346097e-04
  1.08700325e-03  4.85829118e-04 -1.20369454e-04 -5.80841712e-04
 -6.67463384e-04 -2.03338189e-04 -7.94651553e-04  1.12723759e-03
 -2.52189633e-04  2.59126264e-04 -9.08678661e-04 -1.28987768e-04
 -2.42093124e-04 -2.96917818e-04 -1.33046035e-04 -2.03306192e-04
  6.95986392e-05  1.28525858e-04  4.37379410e-04 -1.58047326e-04
  4.44059744e-04  5.89974850e-04 -3.20137675e-04  4.60829920e-04
  4.26068562e-04  6.29118049e-04  5.34398723e-04 -9.90187747e-04
 -6.64159140e-04 -6.47406387e-04  3.17824314e-04 -4.01732730e-04
 -2.20877231e-04 -2.66494084e-04 -1.98072438e-05 -1.44313092e-04
 -2.75986596e-04  1.95115282e-04  7.56208939e-04 -5.26240129e-04
  2.95085646e-05  3.39926059e-04  2.84139809e-04 -1.59204808e-04
  7.91428620e-05  3.86672042e-04  2.35158361e-04  2.98336163e-04
 -6.99389522e-04 -1.23699535e-03 -7.47207019e-04 -2.42960355e-04
 -7.29603002e-04 -1.00950039e-03 -9.52246300e-04  6.19831608e-04
  8.07335954e-04 -7.39634349e-04 -5.06219933e-04  1.47167171e-03
  1.34353546e-03  1.14401055e-03 -2.85491766e-04  1.58039285e-03
 -3.36848800e-04  2.14689048e-04  3.10697165e-04 -1.51169975e-05
 -1.89276074e-03  2.47806115e-04 -2.72522652e-04  8.96731569e-04
 -7.53012956e-04  1.60275248e-04  1.45240579e-04 -4.98763328e-04
 -5.63853986e-04  7.19549251e-04  4.18144998e-04  6.52132052e-04
  5.18994646e-04 -3.07857037e-04 -1.43609418e-03 -1.60520304e-04
 -1.11527065e-03  1.90148419e-04]
score 0.1069451654854725
1

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-35-4ac94793a8fe> in <module>
     36 print(ans.ndim)
     37 type(ans)
---> 38 DEMO
     39 print(targets[29])

NameError: name 'DEMO' is not defined
```

In [ ]:
```
[model.fit(X,train.loc[:, targets[i]]) for i in range(0,29)]
```

In [ ]:
```
model.score(X,y)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: