Auto detect question-answer pairs in email threads

Apoorva Bhatnagar
Computer Science
University of Washington
Bothell, WA
bhatnagar.apoorva12@gmail.com

ABSTRACT

Modern day email inboxes are a treasure trove of information that people exchange between each other either as email threads between two individuals or emails sent to large distribution groups. Very often this information is in the form of questions asked by one person which results in one or more responses containing answers to the asked question. In this paper, I present a classification approach to identify a question in paragraph of text and then examining all subsequent sentences in response emails for a potential match as an answer. Once a question-answer pair is found, it can be posted to a wiki with the email owner's consent and then can be referenced if a similar question is asked again by someone. Often, in large organizations employees answer a lot of work-related questions in emails and this information gets lost when the employee leaves the company and their mailbox gets deleted. Instead, if we could post this information to an internal wiki or a question-answer database, then it would prove useful even after an employee's departure.

CCS CONCEPTS

• Natural Language Processing • Machine Learning • Infersent • Sentence2Vector

KEYWORDS

NPS Chat dataset, Facebook Infersent, Sentence2Vector

ACM Reference format:

Apoorva Bhatnagar, 2019. In *Proceedings of Machine Learning class at University of Washington, 2 pages.*

1 Introduction

This paper presents the details of an approach used to solve the problem of detecting question answer pairs. The first section discusses the data sets used to train models and proposed ML libraries that are used.

The next section gives an overview of the sub-problems that need to be solved for achieving a solution. The following sections presents the results of the research. This includes the training methods used and an in-depth comparison of these training methods. Each of the training methods were trained and tested with k-folding of data. I compare, the confusion matrices, accuracy, f1-score and ROC curve and AUC to see analyze the best training method. The next section discusses the use Facebook's Infersent method. This is a pre-trained encoder that can encode a sentence into a vector. These vectors can then be used for multiple purposes to determine similarity. I discuss two approaches for similarity calculation – Cosine similarity and Soft Cosine Similarity. The subsequent section discusses the results achieved and the python code used for achieving the results. The final section discusses similar work done in this field and compares my results to those research pieces. At the end, we conclude with possible improvements that can be made to further enhance this system and possible futures usages of such an implementation.

2 Datasets and ML libraries

Dataset being used -

a) **NPS Chat corpus** - This data set can be found here http://faculty.nps.edu/cmartell/NPSChat.htm. The dataset contains 10,567 posts out of approximately 500,000 posts gathered from various online chat services in accordance with their terms of service. The data-set's sentences are labelled with tags such as *'Accept', 'Bye', 'Clarify', 'Continuer', 'Emotion', 'Emphasis', 'Greet', 'No Answer', 'Other', 'Reject', 'Statement', 'System', 'Wh-Question', 'Yes Answer', 'Yes/No Question'*. Because it's a dataset of human conversations in chat, this serves as a great training dataset for understanding and classifying email conversations because of their similar nature to chat conversations. In my implementation, I use the entire dataset to train 5 different classification models and then compare their results.

b) **Enron dataset** contains 500,000 emails made public by the Federal Energy Regulatory Commission during an investigation and contains emails from the senior management of Enron, organized into folders. This is an excellent data set to test the concept of this paper. Alternatively, I could also use my personal email as the test data for evaluating the performance of the code. However, I chose to use this dataset because its already extracted and cleansed for personal identifiable information.

Proposed ML libraries to be used:
NLTK, Text blob, Infersent, Spacy, Sklearn

3 Overview

The vision of this paper can be broken down to two sub problems

a) Detecting whether a sentence is a question or not. This is a classification problem where in a trained model is used to classify test sentences based on the learnt labels. I use 6 different training methods and compare the results. The model with the best ROC curve (highest area under the curve AUC) is chosen for the real predicting situation. Each of these trained models are serialized and persisted so that they can be loaded, and actual predictions can be done very quickly.
b) Given a question and a context that may contain an answer to the question, find the answer in the context.

I present my work on these two sub-problems with code snippets and examples -

3.1 Question Detection -

```
def mlp_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect):
    grid_search_parameters = {
        'hidden_layer_sizes': [(100,)],
        'activation': ['relu'],
        'solver': ['adam'],
        'alpha': [0.05],
        'learning_rate': ['adaptive']
    }

    mlp_classifier = MLPClassifier()
    classifier = GridSearchCV(
        estimator=mlp_classifier,
        param_grid = grid_search_parameters,
        cv=5, scoring='accuracy', verbose=0, n_jobs=-1)


    classifier.fit(training_data, training_labels)
    prediction = classifier.predict(testing_data)
    f1_score = metrics.f1_score(testing_labels, prediction, average='macro')
    print(f'F1 score for MLP ANN {f1_score}')
    # Prints Confusion matrix
    c_matrix = confusion_matrix(testing_labels, prediction)
    print(f'Confusion matrix {c_matrix}')

    # Prints Performance of the model (accuracy, f-score, precision, recall)
    print(classification_report(testing_labels, prediction))

    # classify a new sentence
    df= pd.DataFrame({'text': ['What is up', 'How do I find the results?'],'class': [1, 1]})
    print(classifier.predict(tfidf_vect.transform(df['text'].values).toarray()))

    actuals, predicts = plot_roc_curve(testing_labels, prediction, 'MLP ANN classifier')

    return classifier, actuals, predicts

def randomForest_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect):
    random_forest_classifier = RandomForestClassifier(n_estimators=10)

    random_forest_classifier.fit(training_data, training_labels)
    prediction = random_forest_classifier.predict(testing_data)

    f1_score = metrics.f1_score(testing_labels, prediction, average='macro')
    print(f'F1 score for Random Forest {f1_score}')
    # Prints Confusion matrix
    c_matrix = confusion_matrix(testing_labels, prediction)
    print(f'Confusion matrix {c_matrix}')

    # Prints Performance of the model (accuracy, f-score, precision, recall)
    print(classification_report(testing_labels, prediction))

    # classify a new sentence
    df= pd.DataFrame({'text': ['What is up', 'How do I find the results?'],'class': [1, 1]})
    print(random_forest_classifier.predict(tfidf_vect.transform(df['text'].values).toarray()))

    actuals, predicts = plot_roc_curve(testing_labels, prediction, 'Random Forest')

    return random_forest_classifier, actuals, predicts
```

figure 1a)

```
def plot_roc_curve( testing_labels, predictions, model):
    testing_labels = np.array([int(each) for each in testing_labels])
    predictions = np.array([int(each) for each in predictions])
    auc_score = roc_auc_score(testing_labels, predictions)
    print(f'AUC Score for {model} is {auc_score}')
    return testing_labels, predictions
    false_positive_rate, true_positive_rate, thresholds = roc_curve(testing_labels, predictions)
    plot.plot(false_positive_rate, true_positive_rate, marker='.', label=model)

def train_model():
    file_path = dirname(os.path.realpath(__file__))

    # Load nps chat data set
    # Fold the data between training and testing data K times
    # Train 5 different models with different grid search parameters
    # Compare results
    for i in range(10):
        #K-fold the dataset into different training and testing dataset
        training_data, training_labels, testing_data, testing_labels, tfidf_vect = load_nps_chat_data(randomSeed = 42*i)

        # 1.Random Forest classifier
        classifier, actuals, predictions = randomForest_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\randomForest.sav'
        pickle.dump(classifier, open(file_name, 'wb'))

        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='Random Forest')

        # 2. Naive Bayes classifier
        classifier, actuals, predictions = naiveBayes_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\naiveBayest.sav'
        pickle.dump(classifier, open(file_name, 'wb'))

        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='Naive Bayes')

        # 3. Decision Tree classifier
        classifier, actuals, predictions = decisionTree_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\decisionTree.sav'
        pickle.dump(classifier, open(file_name, 'wb'))
        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='Decision Tree')

        # 4. SVM classifier
        svm_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\svm.sav'
        pickle.dump(classifier, open(file_name, 'wb'))
        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='SVM')

        # 5. Logistic Regression classifier
        classifier, actuals, predictions = logisticRegression_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\logisticRegression.sav'
        pickle.dump(classifier, open(file_name, 'wb'))
        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='Logistic Regression')

        # 6. MLP ANN classifier
        classifier, actuals, predictions = mlp_TrainedModel(training_data, training_labels, testing_data, testing_labels, tfidf_vect)
        file_name = f'{file_path}\\models\\mlp.sav'
        pickle.dump(classifier, open(file_name, 'wb'))
        false_positive_rate, true_positive_rate, thresholds = roc_curve(actuals, predictions)
        plot.plot(false_positive_rate, true_positive_rate, marker='.', label='MLP ANN')
```

figure 1b)

Figure 1a: code snippet that shows model being trained on the NPS chat corpus using a multi-level perceptron-based ANN and a random forest classifier. The dataset is k folded and grid search is used to identify the best parameters for the ANN. Figure 1b: code snippet showing 6 different training models being trained using k-folding of the data. The trained models are persisted using pickle.

Initially, I trained a model on the NPS chat corpus using a Naïve Bayes Classifier [7]. This model had an accuracy score of 60% which isn't good enough for classifying sentences as questions or not. To overcome this, I tried training the model on 6 other ML models such as SVM, Random Forest, Logistic Regression and multi-level perceptron-based ANN. These models significantly improved performance of the classifier with the highest accuracy being achieved by SVM, Logistic Regression and MLP. I calculated the Confusion matrix, F1 score, precision, recall, ROC and AUC for each of these models. These are shown in figure 2 and 3. The high performance of some of these models is clear from these graphs. Once trained, the models were persisted so that they can be loaded from storage and then used to make predictions on sentences detected in an email. If the model predicted a sentence to be a question, we proceed to section 3.2.

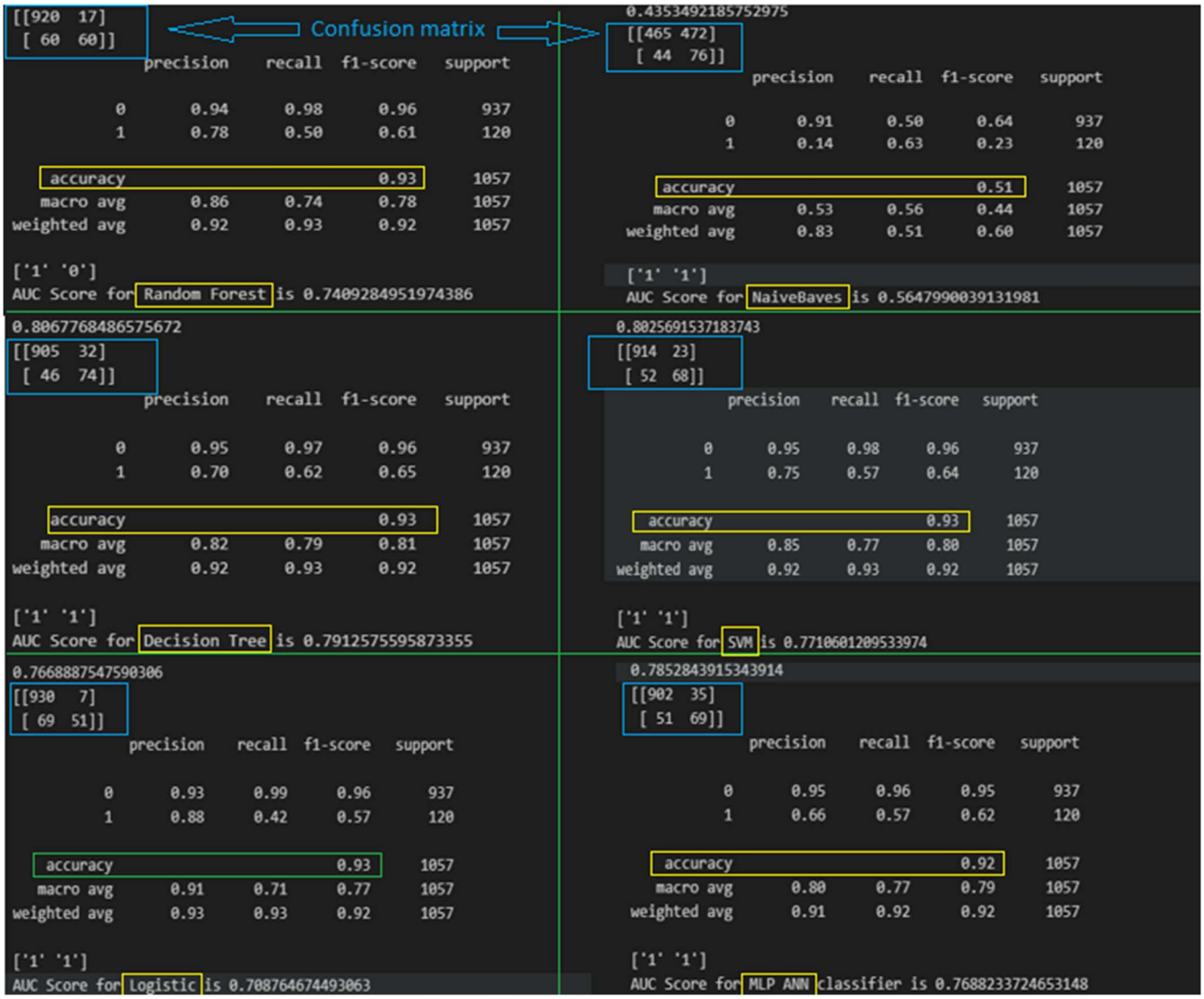

Figure 2 (top). Comparison of the 6 ML models and their accuracy, f1-score, confusion matrices and AUC scores

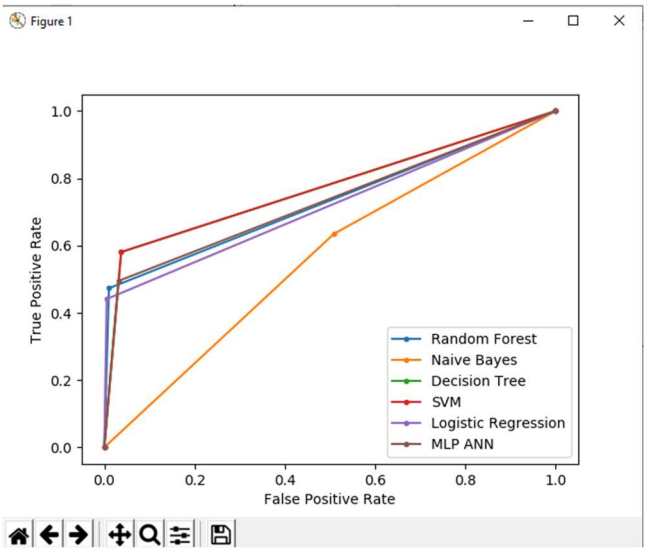

Figure 3 (right). ROC curves for each of the 6 models. After 100 randomly generated folds of the data, the SVM model had a high accuracy rate as seen in the graph on the right. We use this for our problem of detecting questions.

3.2 Answer Detection -

For this, I decided to use Facebook's recently released pre-trained model called Infersent which can provide vector representations of whole sentences. This method uses *fastText* to train itself using a deep neural network (the implementation of which is outside the scope of this paper). This model provides a sentence2vec representation and is the next step from previously used techniques such as word2vec. Given a sentence, this can help me retrieve a vector representation of the sentence. An approach to detect whether a given sentence is an answer to a question or not, I evaluate the Euclidean distance and the Cosine distance between the vector representation of the question and answer. The hypothesis is that a question and answer usually contain similar words and hence would not be very far away from each other in the vector space. My results indicate that Cosine distance has a higher accuracy than Euclidean and that's intuitive because Cosine distance takes the direction of the vector into account as well

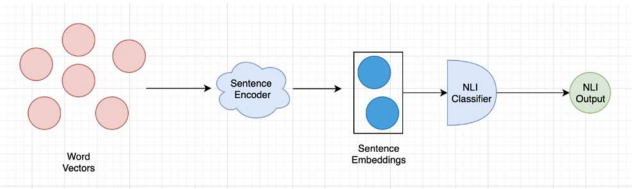

Figure 4: Infersent - a sentence embeddings method that provides semantic representations for English sentences. It is trained on natural language inference data and generalizes well to many different tasks (explained by [4])

4. Implementation

For testing both sub-problems I am using the Enron dataset as a test data set. It's an unlabeled data set that contains email messages sent between employees of Enron. The data contains a collection of email threads. An email thread consists of email messages sent back and forth between different people. Each message contains the sender and receiver's email address, a subject and body of text. I am mostly interested in the text body of these messages for the purpose of this paper. First step I did here was to extract text from the email and then break down the text into a list of sentences. For this part I used **Textblob**[8] to break the paragraph into sentences. Next, I fed each sentence from the list to the trained model from 3.1:

a) Each sentence from the list was fed to the trained model to classify it as a question or not. The mode has an accuracy of 95% on the test dataset and does a good job in detecting questions in the email set. This is illustrated in figure 5 which shows the questions are generally accurately detected. There are some outliers that can be highlighted. Long and twisted sentences are hard to classify. The model also isn't as accurate as it was on the test data set because of the difference in the domain of training data set and the dataset being evaluated. The training dataset was based on chat conversations, but the evaluation is done on email conversations. While the hypothesis is that these are very similar human interactions, there are still some nuances between them and that can induce some more inaccuracies in the prediction of the model

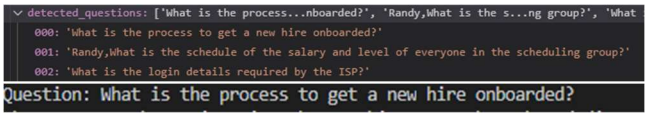

Figure5: The sentences detected as questions are reasonable questions asked by people in emails

b) After a sentence is detected as a question, I use infersent to find out its vector form representation. See Figure 6

```
import nltk
import os
import torch

from os.path import dirname, join as pjoin
from models import InferSent
from sklearn.metrics.pairwise import cosine_similarity, euclidean_distances

def load_infersent_model():
    file_path = dirname(os.path.realpath(__file__))
    MODEL_PATH = f'{file_path}\\encoder\\infersent2.pkl'
    params_model = {'bsize': 64, 'word_emb_dim': 300, 'enc_lstm_dim': 2048,
                    'pool_type': 'max', 'dpout_model': 0.0, 'version': 2}
    infersent = InferSent(params_model)
    infersent.load_state_dict(torch.load(MODEL_PATH))

    W2V_PATH = f'{file_path}\\fastText\\crawl-300d-2M.vec'
    infersent.set_w2v_path(W2V_PATH)

    return infersent

def build_encoding(sentences):
    infersent = load_infersent_model()
    infersent.build_vocab(sentences, tokenize=True)
    embeddings = infersent.encode(sentences, tokenize=True)
    # infersent.visualize('Yes I can call you?.', tokenize=True)

    cos_similarity = cosine_similarity(embeddings)
    euclidean_distance_similarity = euclidean_distances(embeddings)

    return cos_similarity, euclidean_distance_similarity
```

Figure 6: Code snippet showing loading of the infersent dataset and creation of a vector representation (sentence embedding) for a list of candidate sentences

Next, we look at the next 20 sentences from the response message and convert each sentence to a vector form as well. Each of these sentences are candidates for being the answer to the asked question. I then compare vector forms of the question and the candidate sentences to find out their cosine similarity. Sentences with lowest cosine angle relative to the question is selected as an answer to the question (see figure 7 for an illustrative example of cosine similarity). I also check the Euclidean distance between the question and the candidate answers. This is done to understand the difference between the performance of the prediction system based on the directionality of sentence vectors. Euclidean distance does not consider the direction between the two vectors and hence can result in higher false positives.

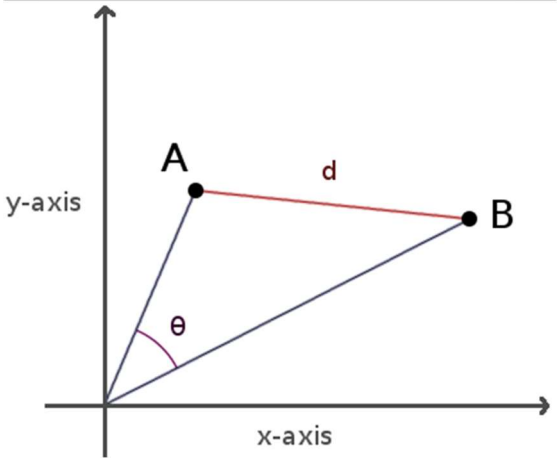

Figure 7: Cosine distance between vectors A and B where A represents the question and B represents the candidate answer. The correct answer is the sentence with vector B such that theta is minimized.

The boxes in green highlight the good examples found in this research. The question detected in these cases are real reasonable questions asked by a person and the answer detected is also accurate and contains information that can be useful to more than 1 person.

The boxes in yellow highlight cases where a good question was detected but the captured answer either lacks details and is vague or is incorrect altogether. Because the detected question was correct, these are still considered success because this information can be used by the process to nudge the user to provide valid answers

The boxes in red are the bad cases. In these cases, the model was inaccurate and caused a false positive. The sentence marked as a question is not really a question and hence the candidate answers detected also add no further value to the problem being solved.

Overall, while the system does detect real questions and answers and while there are a few false positives, they can be filtered out to present useful results to a user.

Figure 8. Real results of the implementation.

```
c:\work>"C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe" c:\Users\akbhatna\.vscode\extensions\ms-python.python-2019.11.5
Found 102(/103) words with w2v vectors
Vocab size : 102
Question: What is the process to get a new hire onboarded?
Chosen answer by cosine sim: The new hire goes through a dedicated weeklong onboarding process with HR - see details here <link redacted>, the process
gets started with them getting welcomed with a breakfast and coffee, They then go through a series of speaker sessions and the company executives prese
nt them their vision for the company and its future; The group then breaks for lunch  provided by HR; The post lunch sessions focus on benefits of work
ing at the company and post those sessions, the new hires are processed by IT to get their desk set up and ready for work.
Chosen answer by eucledean sim: (Patti S for example)Phillip
Question: Randy,What is the schedule of the salary and level of everyone in the scheduling group?
Chosen answer by cosine sim: Greg,Please find attached the schedule of the salary and level for all folks in the scheduling group, I believe there are
Chosen answer by eucledean sim: (Patti S for example)Phillip
Found 58(/62) words with w2v vectors
Vocab size : 58
Question: Could you share the details of collar/floor price for your power generation deal?
Chosen answer by cosine sim: Phillip,As discussed  during our phone conversation, In a Parallon 75 microturbinepower generation deal for a national ac
Chosen answer by eucledean sim: These are the complete list of trades fromEnron Online (EOL), Enron's direct phone conversations, and three brokeragefi
Found 25(/29) words with w2v vectors
Vocab size : 25
Question: I have forwarded your request to Zarin Imam at EES.
Chosen answer by cosine sim: Follow these steps so you don'tmisplace these files.1.
Chosen answer by eucledean sim: Follow these steps so you don'tmisplace these files.1.
Found 112(/125) words with w2v vectors
Vocab size : 112
Question: Jim,Is there going to be a conference call or some type of weekly meeting about all the regulatory issues facing California this week?
Chosen answer by cosine sim: Can you make sure the gas desk is included.Phillip
Chosen answer by eucledean sim: George,Below is a list of questions that Keith and I had regarding the WestgateOwnership StructureWhat will be the own
Found 73(/83) words with w2v vectors
Vocab size : 73
Question: General partner?What are all the legal entities that will be involved and in whatcapacity(regarding ownership andliabilities)?Who owns the I
Chosen answer by cosine sim: Now or for the land closing?
Chosen answer by eucledean sim: improvements?Who holds the various loans?Is the land collateral?InvestmentWhat happens to initial investment?Is it use
Found 108(/114) words with w2v vectors
Vocab size : 108
Question: By whom?What type of bank account?
Chosen answer by cosine sim: Do you getpaid a markup on subcontractors as ageneral contractor and paid gain out of profits?Do you or Larry receive any
l investors have in selecting design and materials for units?What level of investor involvement will be possible during constructionplanning and permi
edules, or reference checking?Are there any specific companies or individuals that you already plan touse?
Chosen answer by eucledean sim: Internet viewingfor investors?Reports to track expenses vs plan?Bookkeeping procedures to record actual expenses?What
Found 60(/70) words with w2v vectors
Vocab size : 60
Question: Jeff,What is up with Burnet?Phillip
Chosen answer by cosine sim: Jeff,I need to see the site plan for Burnet.
Chosen answer by eucledean sim: Remember I must get writtenapproval from Brenda Key Stone before I can sell this property and she hasconcerns about th
Found 14(/14) words with w2v vectors
Vocab size : 14
Question: Check 1411  Lucy      What is this?4.
Chosen answer by cosine sim: Check 1415  Papes      Detail description and units?5.
Chosen answer by eucledean sim: Check 1415  Papes      Detail description and units?5.
Found 31(/35) words with w2v vectors
Question: Checks 1416, 1417, and 1425  Why overtime?6.
Chosen answer by cosine sim: Check 1438    Walmart?
Chosen answer by eucledean sim: Check 1428    Ralph's   What unit?7.
Found 38(/43) words with w2v vectors
Vocab size : 38
Question: Check 1428    Ralph's   What unit?7.
Chosen answer by cosine sim: Description and unit?Try and pull together the support for these items and get back to me.Phillip
Chosen answer by eucledean sim: ---------------------- Forwarded by Phillip K Allen/HOU/ECT on 09/12/2000Michael EtringerPhillip,Attached is the list.
```

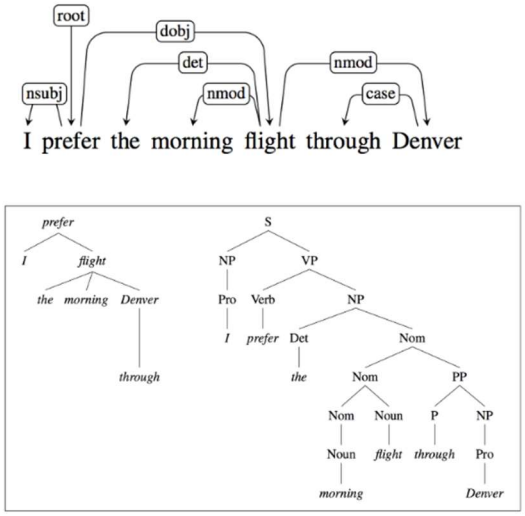

Figure 9: Spacy Dependency parse tree for a sample sentence

To boost the results further, I added spacy dependency tree analysis to the question-answer pair as well. The assumption here is that question and answers usually share the same root verb in a sentence and if the root of the question and answer are the same, then we can say with higher confidence that the selected answer is the actual answer to the question.

5. Comparison with similar work

Most of the work that I found in this domain was around finding the answer to a given known question using a context of text where the answer might be located. SQUAD 2.0 dataset is a labelled dataset that contains questions posed by crowdworkers on a set of Wikipedia articles where the answer to every question is a segment of text from the corresponding reading passage. Most implementations that create a supervised model use this data set to train their model and then apply it on real world problems. There are no published cases where someone has attempted to apply this to the data contained in people's inbox and I believe this might be the first attempt at solving this problem on user's mailboxes outside of private companies such as Google and Microsoft who own the majority share of emails being exchanged. Gmail and Outlook both have capabilities to nudge users based on understanding the content in a user's mailbox. See figure for example. There are no known publications from either company on this is achieved.

6. Possible extensions and improvements

The question detection classifier that I initially used was a Naïve Bayes classifier and it performed very poorly, giving me no more than 60-70 percent accuracy. When I switched to using better training models such as MLP based neural network with stochastic gradient decent and support vector machines the accuracy of the model improved to 95% with low recall rates as well. A more accurate question detector is obviously the key to this problem.

Once a question is detected, we evaluate the next 20 sentences for whether they contain the answer or not. This is an arbitrary choice I made for this research and can be fine-tuned based on the results of the model. Another extension and possible improvement for sub-problem B would be to use a supervised learning problem and then train a model using the SQUAD data set and then apply that to the test data

7. Conclusion

The focus of the paper is to solve the problem of detecting question-answer pairs in emails. These pairs usually represent knowledge shared between people such as co-workers in an office. This information can be extracted out of the mailbox and with the user's consent automatically posted to a wiki page as documentation. This way when a third person has the same question, they needn't send out an email and can first search the wiki page for answers. It also helps in making this information available to an organization in case an employee left the company and their mailbox were to be deleted.

Detecting questions in a person's emails can have other revolutionary outcomes. Imagine being asked a question but then forgetting to answer it because of your busy schedule. A question-answer detector can identify when a question has been asked but there is no answer and can help remind the user to respond back to that email. Another cool application could be to augment existing search capabilities of email-search. Today these are just text-based searches and can be made better if the search is question based. Modern human interactions with machines is increasing becoming conversational and hence information indexed as questions and answers can be easily searched and retrieved by a search program.

REFERENCES

[1] NPS Chat corpus - http://faculty.nps.edu/cmartell/NPSChat.htm.
[2] Squad Dataset - https://rajpurkar.github.io/SQuAD-explorer/
[3] Enron Dataset - https://www.kaggle.com/wcukierski/enron-email-dataset
[4] Understanding Infersent - https://medium.com/analytics-vidhya/sentence-embeddings-facebooks-infersent-6ac4a9fc2001
[5] Understanding Squad Dataset - https://rajpurkar.github.io/mlx/qa-and-squad/
[6] Question detection - https://medium.com/@sid.ghodke/finding-the-questions-in-customer-chat-messages-9f5f7db00881
[7] NLTK Question classifier - http://www.nltk.org/book/ch06.html
[8] Textblob - https://textblob.readthedocs.io/en/dev/