Text Analysis of Technical Support Documents

Stephen Kappel (spk2131)

May 5, 2014

Motivation

I work in the Customer Success & Support organization for CA Technologies (an enterprise software firm). CA's technical support answers over 200,000 questions from customers each year. These questions come in primarily as support tickets and forum posts. Efficient support – that is, answering these questions quickly and accurately – drives higher customer satisfaction and lowers costs. My goal is to drive efficient support by gaining insight and creating operational tools that take advantage of our textual data. The textual data at our fingertips includes support tickets, forums posts, log files, KB search queries, and technical documentation.

From an insight perspective, understanding the recurring themes in customer questions, log files, documentation, etc. could (a) highlight where there are product quality or functionality gaps that need to be addressed by development, and (b) identify areas where more documentation is needed. In terms of operational tools:

- Issues could be intelligently routed to engineers based upon the content of the current issue. The routing algorithm could judge the strengths and weaknesses of engineers based on past issues, and route incoming issues accordingly.
- Previous issues similar to a new issue could be raised to an engineer's attention. This would be done by analyzing the text of a current support ticket and log file with previous suuport tickets and log files.
- When a new question is posted to a forum, email alerts could be sent do engineers that answered a similar issue in the past, or the individual posting the question may be pointed to similar forum questions posted in the past.

Overview

In this project, I use the text from questions posted to MSDN's SharePoint and SQL Server forums during 2013. I previously scraped this data to an SQL database for another project. (See `https://github.com/StephenKappel/dataology/tree/master/MsdnForums` for more information about the data set and my other project.) This corpus consists of the title and question text from 73,822 threads spanning 48 distinct forums. I believe I can extend the algorithms I build using this corpus to other technical issue text, such as the text in support tickets or CA Technologies' forums text. The benefit of using the MSDN corpus for this project is that the data is alread publically available and I already have it conveniently organized in a database.

I start by looking at a supervised learning case, where I try to classify threads into a forum category (product) or forum based upon their text content. Being that all the threads are already in a forum, I have labels that allow me to easily check how well my algorithms perform. This type of algorithm could have applications in areas like issue routing and categorizing knowledge base search queries by product.

I then turn my attention to the unsupervised case, where I try to find other threads similar to a given thread. Also, I use LDA in an attempt to find the latent topics within the corpus.

Supervised Learning of Forum Categories and Forums

For supervised learning, each *document* I consider is the combined title and question text from a forum post. The *labels* are the forum to which the thread was posted and the forum category (SharePoint or SQL Server) to which that forum belongs. My objective is to find a algorithm that is able to use a document's text to predict its lables with a high degree of accuracy. Below, I outline the process I have gone through in search of such an algorithm.

Identifying the Vocabulary

After collecting the text data to analyze, the next step in building a bag-of-words classification algorithm is identifying the words that will be considered as features. My approach for producing the vocabulary was as follows:

1. Read the complete corpus of text from my database into Python.
2. Tokenized the text into words, being sure to remove periods, hyphens, and other punctuation.
3. Print into a file a list of all the words appearing in corpus and the number of time each of these words appeared.
4. Manually identify a set of frequently-occuring, non-value-add "stop words" from this list and created a file listing these words.
5. Find all the words in the corpus that appear at least ten times and are not in the list of stop words. Let this be the vocabulary. (I accomplish this in the Python script *1_create_vocab.py*, and my vocabulary is printed out to *wordlist.csv*.)

Identifying Training, Validation, and Test Data Sets

Conveniently, each thread in my database is tagged with a randomly chosen GUID. Therefore, I can get a random division of the corpus by simply using the first character of the GUID. I have identified my data sets as follows:

- **Training** The 46,127 threads having a GUID beginning with a numerical digit.
- **Validation** The 13,769 threads having a GUID beginning with D, E, or F.
- **Test** The 13,926 threads having a GUID beginning with A, B, or C.

Classifying Threads by Forum Category

I began with the easiest classification problem – dividing the threads into the two forum categories – SQL Server and Sharepoint. The high-level approach I took for classifying documents into categories was:

1. For a set of training documents, create a *document-vocab matrix* with a row for each document and a column for each word in the vocabulary. By parsing each training example, populate the *document-vocab matrix* with counts of the number of times each word appears in each training document.
2. For each forum category, sum all the rows in the *document-vocab matrix* for which the row is associated with a document belonging to the document category. This produces a *category vector* for each category, where the *category vector* has one entry for each word in the vocabulary.
3. For each document that is to be classified, build a *document vector* with each entry being a count of the number of time a particular word in the vocabulary appears in the document.
4. Classify each document by finding the dot product between the *document vector* and each *category vector*. (This is the numerator of the traditional cosine simiarity comparator.) The document should be assigned to the category for which this product is greatest.

There are many small adjustments that can be made to the above algorithm, each of which can significantly impact the accuracy of the classification. Not knowing what would work best on my data set, I experimented with the following:

- Dividing the product in step (4) by the product of the vectors' norms. This gives us the cosine similarity. Plots where I use this adjustment have "Cosine" in the title.
- Normalizing the *category vectors* (by dividing by the sum of entries in the vector) so that the categories with more training documents did not get a boost in cosine similarity. Plots where I use this adjustment have "Normalized" in the title.
- Using binary word occurrence indicators rather than word frequency counts when building the *document-vocab matrix* and/or *document vectors*. Plots where I use this adjustment have "Binary Word Counts" in the title.
- Using inverse document frequency to transform the *category vectors* and give more influence to those words that appear less often. Plots where I use this adjustment have "tf-idf" in the title.
- Incorporating prior knowledge of forum category probabilities into the cosine similarity calculations. Plots where I use this adjustment have "with Priors" in the title.

In order to play with these parameters, I wrote my code (see *2_classify.py*) in a highly parameterized way, so that each of these parts could easily be turned on and off. I plotted accuracy vs. number of training examples for a numbers of variants of the algorithm. A few of these plots are shown in Appendix A, and a full set is in the submission zip. My conclusions from my experimentations are:

- Without normalization of the *category vectors*, the more frequenct category is increasingly favored by the algorithm as the training set increasing in size, yielding poor results.
- Compared to word counts, binary word indicators perform better.
- tf-idf performs better than both binary word indicators and word counts, yielding over 95% accuracy for the data in my validation set.
- In nearly all scenarios, the accuracy of classification of SharePoint threads is greater than the accuracy of classification of SQL Server threads. Perhaps this is because there are more SharePoint threads than SQL Server threads in the population.
- Including prior knowledge/probabilities does not seem to help the model. I tried several ways to incorporate the prior knowledge into the model with varying strengths, and none produced notable improvements in accuracy.

Running the best parameterization I found (normalized tf-idf classification without the use of priors), I ran my algorithm agains the test set, and got these results, where the column indicates the true label of the example:

	SharePoint	SQL Server	Overall
Correct class	7,304	5,865	13,169
Incorrect class	450	307	757
Accuracy	94.2%	95.0%	94.6%

Classifying Threads by Forum

Being that classifying threads into forum categories works well, the logical next step is to classify threads into specific forums within those categories. To make the data cleaner, I grouped SharePoint forums with the same topic ("Search", "Development and Programming", etc.) but for different releases of SharePoint into labels that mainted the topic but removed the distinction between releases. In many cases the same question could apply to multiple releases of the product, so having separate lables for different releases would only serve to make the algorithm less accurate/useful. This brings the total number of forums down from 48 to 38. Even so, I expected the algorithm to take a significant accuracy hit when being scaled from two classes to 38 classes. And, indeed, using the same normalized tf-idf (without priors) methodology that yielded 95% accuracy for forum categories, we get approximately 23% accuracy with 6,000 training examples.

After some experimentation, I found that the accuracy could be improved by adding more training examples. As seen in Appendix B, there is a sudden rise in accuracy after 6,000 training examples. Also, priors have a more beneficial impact for forums classification than they did for forum categories. With

9,000 training examples, the normalized tf-idf algorithm achieved an accuracy of approximately 33% without using priors and an accuracy of over 40% with priors.

However, normalized tf-idf no longer seems to be the best solution. Using cosine similarity on binary vectors, the algorithm is able to achieve an accuracy of over 50% using 5,000 or more training examples. It is interesting to note that SQL Server threads are classfied correctly more often than SharePoint threads. This is surprising to me, because there are more SharePoint threads in the data than SQL Server threads, and there are fewer SharePoint forums than SharePoint forums. Product complexity may be a factor here. Using the entire training set and testing against the entire test set, I got these results:

	SharePoint	SQL Server	Overall
Correct class	3,378	3,529	6,907
Incorrect class	4,376	2,643	7,019
Accuracy	43.6%	57.2%	49.6%

Not quite satisfied with this performance, I decided to shift gears. Instead of using cosine similarity between a document and each of the forums in aggregate, I implemented a KNN-style approach, where each document is compared directly to all other documents and the nearest neighbors vote for the classification. To compare documents, I use cosine similarity between the *document vectors* (where the training *document vectors* are constructed with binary values and the test *document vector* is constructed using word frequency).

I tried several values of k and training set sizes when testing my algorithm against my validation data set. Plots from these tests are shownn in Appendix C. My code for KNN classfication is in *3_classify_knn.py*. My conclusions about using a KNN approach to text classification are:

- The performance of the KNN algorithm can significantly vary based on the choice of k. For my use case, a k of around 9 seems to work best.
- Because the KNN algorithm must compare each document to all the training documents, it is extremely slow and the run-time is sensitive to the size of the training set.
- The KNN accuracy is slightly worse than my first approach. This in combination with speed would deter me from using this for classification.

The results from testing the KNN algorithm (with $k = 9$ and 3,000 training documents) against $1,000$ documents from the the test set are shown below:

	SharePoint	SQL Server	Overall
Correct class	244	221	465
Incorrect class	313	222	535
Accuracy	43.8%	49.9%	46.5%

Unsupervised Learning of Issue Similarity and Latent Topics

Considering a use case where we want to find past/resolved questions similar to a new one, I adpated my KNN algorithm to, given the text from a thread, find the three threads in the training and validation sets that are the most similar based on cosine similarity and tf-idf. This is implemented in *4_find_similar.py*. From my test runs, the results were mixed; while the similar documents identified by the algorithm definitely had similar vocabularies, they weren't always related to the same problem, so one's resolution had nothing to do with the other. A sample output from the algorithm is shown in Appendix D.

In pursuit of algorithms that can be used to discover topics/themes in textual data, I researched the work being done by David Blei and others in the area of Latent Dirichlet Allocation (LDA). Instead of implementing my own LDA algorithm, I leveraged the work of Matthew Hoffman, who has published an open-source Python implementation of LDA topic modelling at `http://www.cs.princeton.edu/~blei/downloads/onlineldavb.tar`. I adapted this code to use text from my database, rather than text scraped from Wikipedia. The algorithms outputs the top words associated with each topic generated by the model. The topics identified by the model, when run with a k parameter (number of topics) of 7 can be found in Appendix E. My adapted code is in *5_discover_topics.py*.

Opportunities for Further Exploration & Improvement

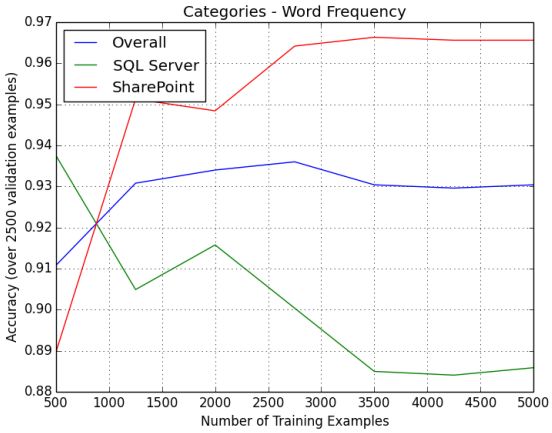
This project has improved my understanding of text analysis and laid the foundation for future work, but has not yet produced an algorithm that is ready to be applied to my real-world use cases. As I move forward toward these real use cases, some of the things I would like to attempt to improve my algorithms include:

- Using a layered approach to (1) identify the forum category (product) that some text is related to, then (2) identify the forum (topic) under the assumption that the forum must be part of the forum category already determined.
- Applying stemming to the words so that terms like "install" and "installation" are considered as the same feature, rather than two unrelated features. This will likely require a special stemmer geared toward technical terminology. Optimally, abbreviations (and typos) would also be mapped to their related words. For example, db $\rightarrow$ database and SP $\rightarrow$ SharePoint.
- Creating a vocabulary weighting methodology (similar to tf-idf) that adds weight to terms that are strongly concentrated in a specific class.
- Deriving insight from the topics from the LDA algorithm. I want to look at the concentration of each of the LDA topics in each forum/forum category. I will likely make use of the LDA package in R, which provides additional capabilities for easily working with LDA. Viewing LDA topic trending over time may show interesting and actionable insight.
- Incorporate other non-textual features into models with text. For example, when trying to find similar previous problems, we may be able to incorporate information like the date or the user asking the question.
- Giving extra weight to the words that appear in title of a thread, as these words are likely more valuable than the other words in the post.
- Improve the speed of the KNN algorithm by only searching within a particular set of documents, which may be identified using a faster classification algorithm.

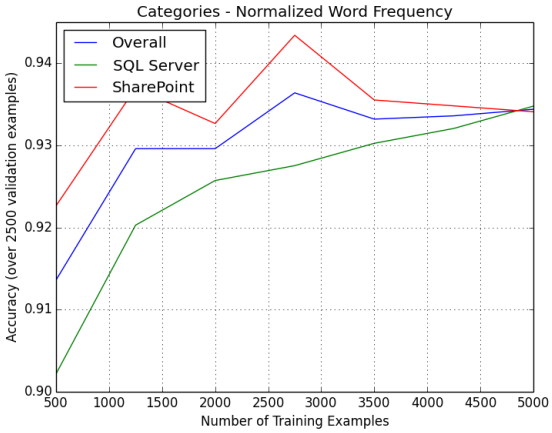
References

Blei, D. (2011). Probabilistic topic models. doi:10.1145/2107736.2107741

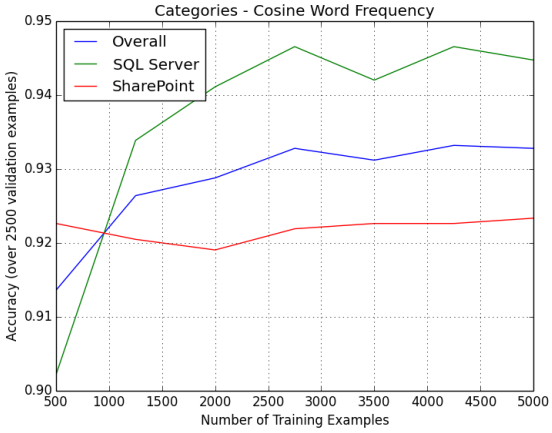
Appendix A: Classification between Forum Categories



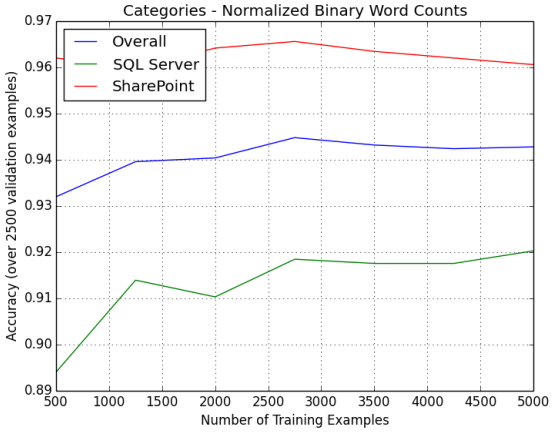
(a)



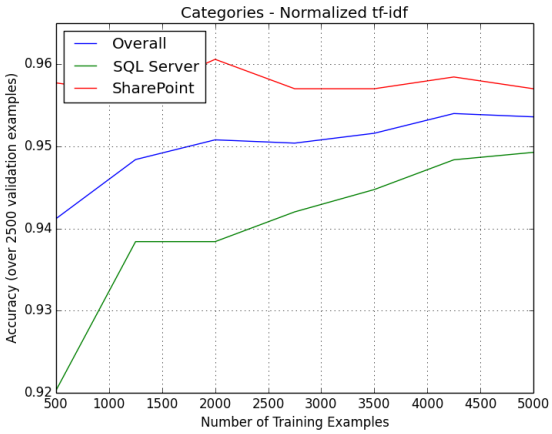
(b)



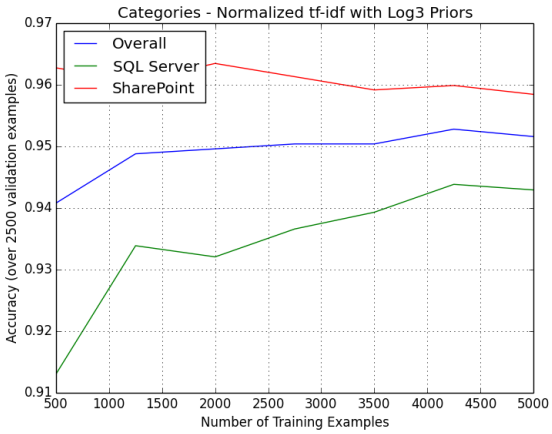
(c)



(d)



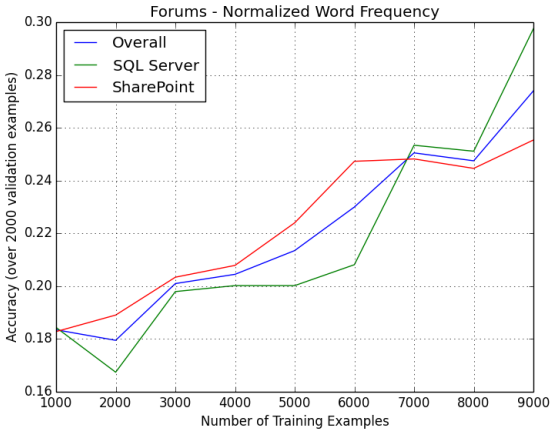
(e)



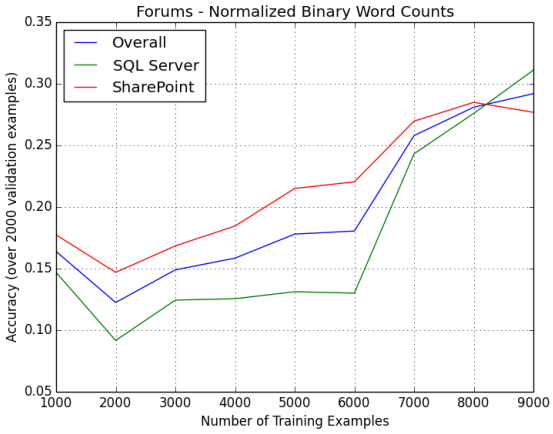
(f)

A sampling of plots constructed to show accuracy differences between variants of the classification algorithm for splitting threads between forum categories. (a) shows the first attempt, and (e) shows the best-performing variant.

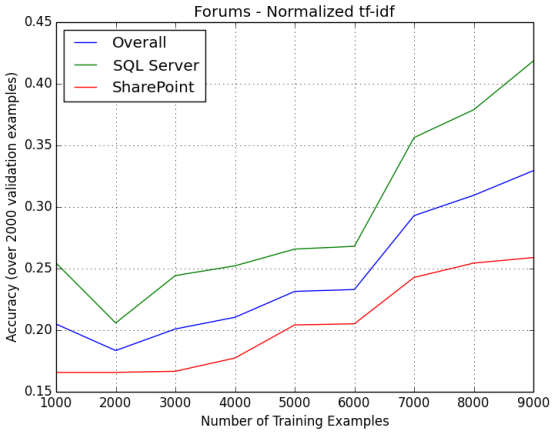
Appendix B: Classification between Forums



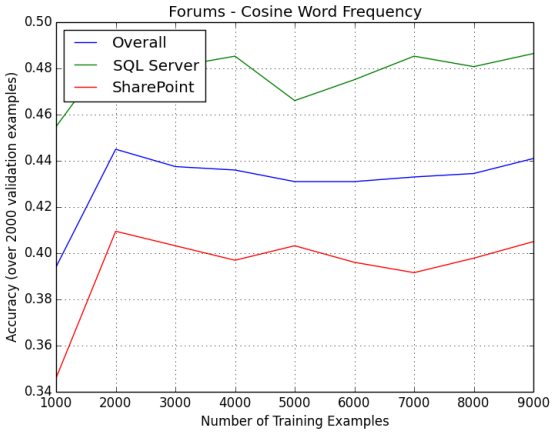
(a)



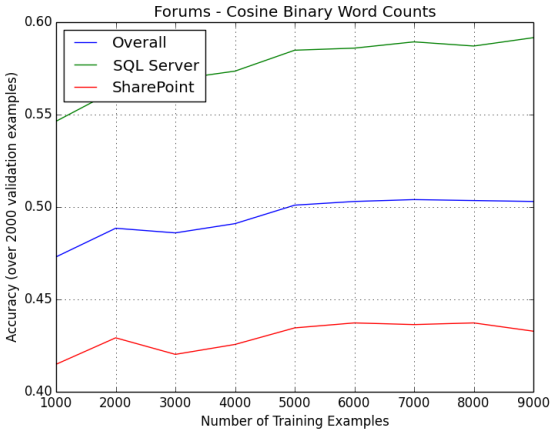
(b)



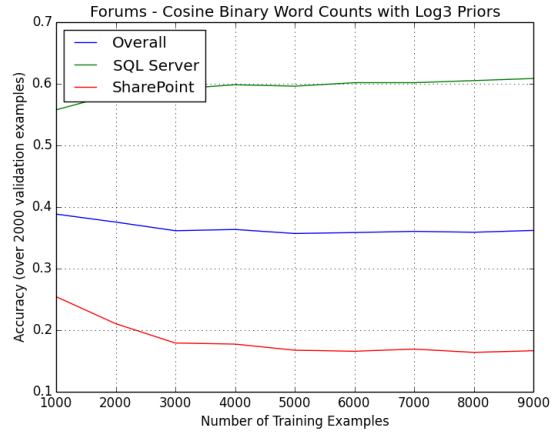
(c)



(d)



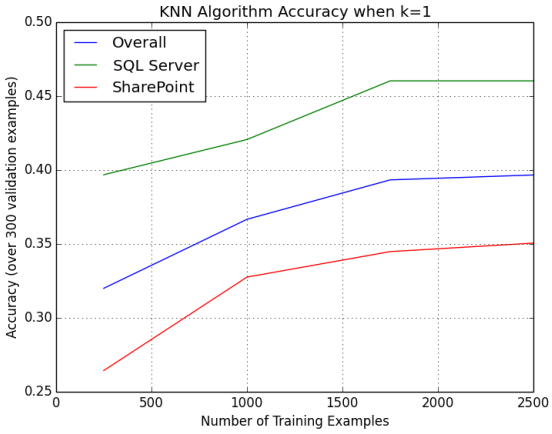
(e)



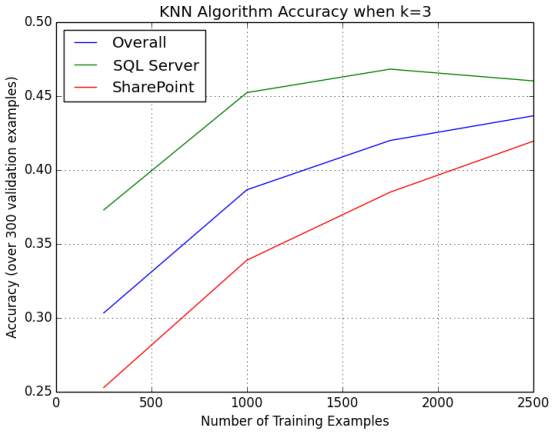
(f)

A sampling of plots constructed to show accuracy differences between variants of the classfication algorithm for splitting threads between forums. (c) shows the algorithm that had performed best for forum categories. (e) shows the best performing variant in this case. Notice that the cosine similarity variants perform better than the non-cosine similarity versions.

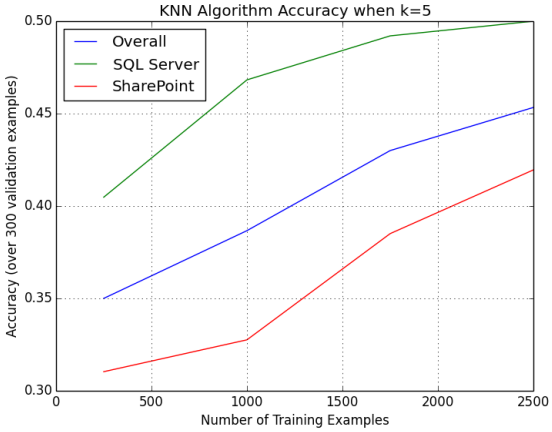
Appendix C: KNN Classification between Forums



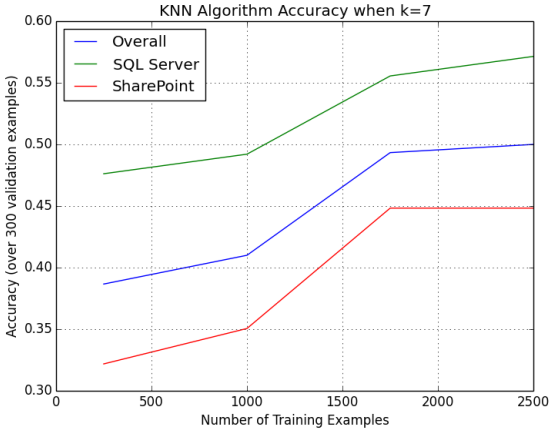
(a)



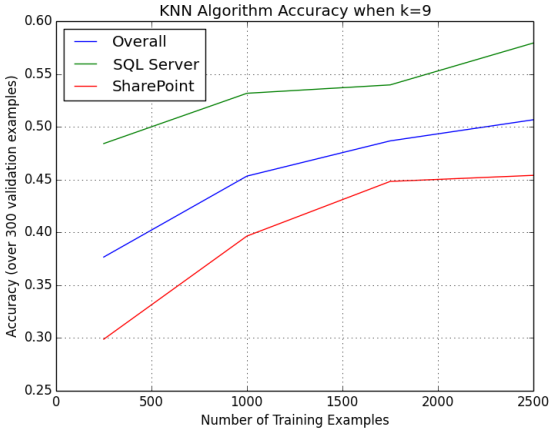
(b)



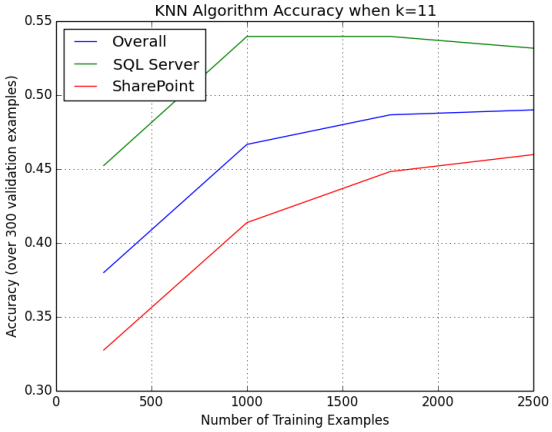
(c)



(d)



(e)



(f)

A series of plots showing how different values of the k parameter affects the accuracy of the KNN algorithm for classifying thread text into forums.

Appendix D: Nearest Neighbors Text Review

This appendix shows an example result produced by the code found in the *4_find_similar.py* file.

The algorithm was asked to find the threads most similar to the thread with this text:

```
Parsing XML file with conditional INSERT

Hi,

I'm a beginner trying to do something smart. I've been reviewing other answers
but I haven't been able to find a solution.

What I'm trying to do is create a stored procedure which will read an XML file
and insert this in a database table. So far so good. Since this job will run
every day and the XML file will change, I need to make sure that the procedure
is not terminating on duplicate keys.

I have managed to use test with a declared variable successfully, but I don't
know how to do this with a dynamic value from the XML. Basically I need to
compare the project IDs in the XML with the project IDs in the database table,
and if they match, skip the INSERT.

The XML file:
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<projects>
  <project ID="99000" description="Project X" />
  <project ID="99010" description="Project Y" />
  <project ID="99020" description="Project Z" />
</projects>

My code is:
DECLARE @HANDLE INT
DECLARE @RESULTS TABLE (RESULT XML)
DECLARE @SQLSTR VARCHAR(MAX)
DECLARE @XMLFILENAME VARCHAR(1000)
DECLARE @XMLDOC XML
DECLARE @PROJECT_ID INT
DECLARE @PROJECT_DESC VARCHAR(100)
SET @XMLFILENAME = '\\ARC\Public\projects.xml'
SET @SQLSTR = 'SELECT * FROM OPENROWSET(BULK ''' &#43; @XMLFILENAME &#43; ''',
SINGLE_CLOB) AS XMLDATA'
INSERT INTO @RESULTS EXEC (@SQLSTR)
SELECT @XMLDOC = RESULT FROM @RESULTS
EXEC SP_XML_PREPAREDOCUMENT @HANDLE OUTPUT, @XMLDOC
/*
Need to run through the project IDs in the XML file and compare it to the
project IDs which are already in the database table.
If it matches skip the INSERT and continue to the next.
 */
IF NOT EXISTS (SELECT * FROM PROJECT_G WHERE PROJECT_G.GPR_PROJECTID =
@PROJECT_ID)
BEGIN
  INSERT INTO PROJECT_G (GPR_PROJECTID, GPR_BESK)
SELECT *
FROM OPENXML(@HANDLE, 'projects/project')
WITH(PROJECT_ID [INT] '@ID', PROJECT_DESC [VARCHAR](100) '@description')
END
EXEC SP_XML_REMOVEDOCUMENT @HANDLE
```

```
Thank you for taking the time to help me!
Greetings!
```

The algorithm identified the following as the closest matches.

Best match:

```
SQL get values from XML URL

Hi my frinds,

I want to get values from the XML URL. but when I execute the SP (Stored
Procedure) and I get ERROR

this is the code:
ALTER PROCEDURE [dbo].[SP_Last_Hadashot_Tozaot] AS
BEGIN
 SET NOCOUNT ON;
Declare @xml XML
DELETE FROM [ONE].[dbo].Last_Hadshot_Tozaot
Select  @xml  = CONVERT(XML,bulkcolumn,2) FROM OPENROWSET(BULK
'http://www.one.co.il/XML/Articles/format/Last',SINGLE_BLOB) AS X
SET ARITHABORT ON
--Insert into Basketballisrael table
Insert into  dbo.Last_Hadshot_Tozaot
        ([LastUpdate],[ID],[Version],[ArticleBy],[Title],[InnerImages])
    Select
        CONVERT(VARCHAR(max),GETDATE()),
        P.value('ID[1]','VARCHAR(max)') ,
        P.value('Version[1]','VARCHAR(max)'),
        P.value('ArticleBy[1]','VARCHAR(max)'),
        P.value('Title[1]','VARCHAR(max)'),
        P.value('InnerImages[1]','VARCHAR(max)'),
From @xml.nodes('//Article') as PropertyFeed(P)
END

and I get this ERROR
"Msg 4861, Level 16, State 1, Procedure SP_Last_Hadashot_Tozaot, Line 12
Cannot bulk load because the file "http://www.one.co.il/XML/Articles/format/Last"
could not be opened. Operating system error code 123(failed to retrieve text for
this error. Reason: 15105)."

Thank you for your Help
```

Second best match:

```
How to pass the table name and database name through variable in sql query

Hi,
i have written below query for dynamically create the table and insert the
records from source table.

Declare @Sql Varchar(Max)
Declare @x int
declare @value varchar(100),@schema varchar(100)='aaa'
Declare @f1 Varchar(100)
Declare @table Table(Idt Int Identity(1,1),grp_id numeric(18,0),id int)
Insert Into @table Select Distinct grp_id,0 from abc..table1
Select @x = Max(Idt) from @table
```

```
select @value=(select id from cde..table2 where Shortname='&#43;@schema&#43;')
While @x>0
Begin
Select @f1= grp_id From @table Where Idt = @x
Set @Sql =
'If Exists( Select 1 from sys.objects where name = ''condition_'&#43;@f1&#43;'''
And Type_Desc = ''USER_TABLE'')

Insert Into condition_'&#43;@f1&#43; ' Select * from abc..table1
Where grp_id = '''&#43;@f1&#43;'''

Else

Select * Into condition_'&#43;@f1&#43;' from abc..table1
Where grp_id = '''&#43;@f1&#43;'''
alter table condition_'&#43;@f1&#43;' add id int
update condition_'&#43;@f1&#43;' set id = '&#43;@value&#43;'
'
 Set @x = @x-1
 --print @sql
 Exec (@Sql)
End

how to pass the table name , database name through variable for this query
because this query is used for 5 tables so every time change the table name.
so will use the variable and pass the value for table name and database name
only.

please modify this query.
Regards,
Abdul .
```

Third best match:

```
SQL get values from XML (No get all data)

Hi,

i have this XML:

and we insert this xml in a table via SP.
ALTER PROCEDURE [dbo].[SP_Last_Hadashot_Tozaot] AS
BEGIN
SET NOCOUNT ON;
Declare @xml XML
DELETE FROM [ONE].[dbo].Last_Hadshot_Tozaot
Select  @xml  = CONVERT(XML,bulkcolumn,2) FROM OPENROWSET(BULK
'C:\Users\yonia\Desktop\ONE\XML\Last_hadashot_tozaot.xml',SINGLE_BLOB) AS X
SET ARITHABORT ON
--Insert into Basketballisrael table
Insert into  dbo.Last_Hadshot_Tozaot
        ([LastUpdate],[ID] ,[Version],[ArticleBy],[Title],[InnerImages])
    Select
        CONVERT(VARCHAR(max),GETDATE()),
        P.value('ID[1]','VARCHAR(max)') ,
        P.value('Version[1]','VARCHAR(max)'),
        P.value('ArticleBy[1]','VARCHAR(max)'),
        P.value('Title[1]','VARCHAR(max)'),
        P.value('InnerImages[1]','VARCHAR(max)'),
```

```
From @xml.nodes('//Article') as PropertyFeed(P)
END

The problem is that all the data fall less "InnerImages" (see table)

How I do to solve this problem?
```

Appendix E: LDA Results

This appendix shows the output of the *5_discover_topics.py* LDA script. The text within the square brackets has been manually added by me as my interpretation of the topic. Also, in interest of space, I have truncated the topic lists to show only the top 20 words.

```
topic 0:   [SQL SERVER ADMINISTRATION]
                sql    ---  0.0281
           database   ---  0.0243
             server   ---  0.0205
              table   ---  0.0168
               data   ---  0.0135
                 db   ---  0.0099
                 id   ---  0.0091
              query   ---  0.0078
             backup   ---  0.0076
             select   ---  0.0072
                log   ---  0.0069
              error   ---  0.0066
               time   ---  0.0064
                dbo   ---  0.0063
                set   ---  0.0060
               null   ---  0.0056
             tables   ---  0.0056
                job   ---  0.0048
                 sp   ---  0.0046
             create   ---  0.0044


topic 1:   [REPORTING/BI]
             report   ---  0.0291
               date   ---  0.0189
               data   ---  0.0157
              value   ---  0.0150
               ssrs   ---  0.0142
              table   ---  0.0108
             column   ---  0.0105
           measures   ---  0.0074
               time   ---  0.0072
             values   ---  0.0071
            reports   ---  0.0070
             select   ---  0.0069
              query   ---  0.0068
               cube   ---  0.0068
          parameter   ---  0.0064
               help   ---  0.0064
            columns   ---  0.0059
          dimension   ---  0.0058
             number   ---  0.0055
              month   ---  0.0054
```

```
topic 2:   [SHAREPOINT FRONT-END USAGE]
              list   ---  0.0403
         sharepoint   ---  0.0222
           workflow   ---  0.0164
               item   ---  0.0157
               page   ---  0.0138
              field   ---  0.0113
               type   ---  0.0094
                new   ---  0.0092
             custom   ---  0.0092
               view   ---  0.0087
                 id   ---  0.0086
              items   ---  0.0079
             column   ---  0.0075
               text   ---  0.0075
              value   ---  0.0075
               user   ---  0.0074
               form   ---  0.0073
                web   ---  0.0073
            content   ---  0.0072
                add   ---  0.0072


topic 3:   [SHAREPOINT ADMINISTRATION]
         sharepoint   ---  0.0453
               site   ---  0.0298
                web   ---  0.0157
               user   ---  0.0144
             server   ---  0.0118
             search   ---  0.0102
        application   ---  0.0102
             access   ---  0.0092
               http   ---  0.0086
            service   ---  0.0082
              users   ---  0.0080
               page   ---  0.0078
                new   ---  0.0077
                 sp   ---  0.0073
              error   ---  0.0069
            content   ---  0.0067
                com   ---  0.0065
         collection   ---  0.0060
               farm   ---  0.0059
                app   ---  0.0056


topic 4:   [FILE SYSTEM]
               file   ---  0.0336
           document   ---  0.0273
              error   ---  0.0247
               data   ---  0.0221
            library   ---  0.0199
               ssis   ---  0.0185
            package   ---  0.0153
               task   ---  0.0138
               form   ---  0.0127
              excel   ---  0.0118
```

```
              source   ---  0.0113
               files   ---  0.0112
            infopath   ---  0.0094
              folder   ---  0.0094
                code   ---  0.0090
          connection   ---  0.0083
           documents   ---  0.0076
                open   ---  0.0072
                 new   ---  0.0067
                help   ---  0.0059


topic 5:   [INSTALLATION/CONFIGURATION]
              server   ---  0.0929
                 sql   ---  0.0740
               error   ---  0.0234
           microsoft   ---  0.0230
             windows   ---  0.0177
             install   ---  0.0115
           installed   ---  0.0104
             version   ---  0.0103
            database   ---  0.0098
             service   ---  0.0095
            services   ---  0.0095
            instance   ---  0.0087
               setup   ---  0.0082
       configuration   ---  0.0077
                  sp   ---  0.0073
              failed   ---  0.0071
             machine   ---  0.0067
              studio   ---  0.0066
           following   ---  0.0065
             edition   ---  0.0065


topic 6:   [SHAREPOINT PROGRAMMING/DEVELOPMENT]
              system   ---  0.0411
           microsoft   ---  0.0390
              string   ---  0.0303
          sharepoint   ---  0.0207
               error   ---  0.0158
             boolean   ---  0.0156
                 web   ---  0.0149
              object   ---  0.0141
           exception   ---  0.0126
                data   ---  0.0102
                 int   ---  0.0085
                  ui   ---  0.0068
                type   ---  0.0066
                http   ---  0.0066
              server   ---  0.0064
             version   ---  0.0058
                 exe   ---  0.0057
            assembly   ---  0.0056
             office   ---  0.0053
                  id   ---  0.0053
```