Building Predictive Text Models for Twitter, News, and Blogs Corpora

Mykola Steshenko

Saturday, October 11, 2014

Synopsis

The obsession of smartphones and other mobile devices has significantly affected our life, and we spend much time typing different kinds of texts in them. The typing still remains very unconvinient due to the size of devices, their virtual keyboards, sensor monitors and other factors. Smart keyboards can improve effeciency of this proccess, reducing the amount of typing. The main goal of this project is to develope a predictive text model for such keyboard, which will be able to predict the most probable next words for twitter, news, and blogs texts. Obtained model will be implemented into a demonstrational Web application.

Data

Data source

The data for this project is from HC Corpora. The particular dataset, that was used for analysis, was provided by Coursera and can be downloaded here:

Coursera-SwiftKey.zip

Zip archive contains files in format LOCALE.blogs.txt where LOCALE is the each of the four locales en_US, de_DE, ru_RU and fi_FI. We analysed only English corpora in this project, which consists of three files "en_US.twitter.txt", "en_US.blogs.txt", and "en_US.news.txt". As it mentioned in the names of the files, they contain raw textual data from twitter, blogs and news respectively. The files also might contain some foreign text and words of offensive and profane meaning. Here is basic information about size and counts of lines, words, and symbols for each file:

```
##                   line_counts word_counts sizeMb
## en_US.blogs.txt        899288    37334131  200.4
## en_US.news.txt        1010242    34372530  196.3
## en_US.twitter.txt     2360148    30373583  159.4
```

Additional description for this dataset is available in HC Corpora ReadMe.

Data processing

The data in each file was separated into two parts. The first part, which consists of roughly 80% of data, was used as training set for n-gram counts generation and for building the predictive model. The rest of the data was used as a test set for the model performance evaluation. Here is data distribution for all three files:

1. en_US.twitter.txt - 2,000,000 lines in training set, 360,148 lines in test set
2. en_US.blogs.txt - 750,000 lines in training set, 149,288 lines in test set
3. en_US.news.txt - 800,000 lines in training set, 210,238 lines in test set

Data Cleaning The raw data contains upper-case letters, punctuation marks, textual emotion expressions (different kinds of smiles), digits, e-mails, hash-tags, offensive and profane words, duplicated symbols, and some other special symbols that should be handled before further processing. It is expected that data from twitter have much more issues like these. Due to this fact, twitter predictive model also might have worse performance than other models, while data from blogs and news was written in much more clear way. Libraries "tm" and "stringi" were used for data cleaning. At the first step of cleaning process all documents in corpus were transformed to the lower case. After that symbols ">" and "<" were removed, because they might be useful later as special tags. At the next step all digits and combinations of digits with digits separated by one of special symbols (":,.-") were replaced with tag "<digit>", because the exact value of them can't be useful in prediction model. Also all e-mails were replaced with tag "<e-mail>" for the same reasons. After that some smiles like ":-P" that might produce single-letter tokens after the special symbols deletion were removed.

Another important procedure in a data cleaning process is handling of punctuation marks and other special symbols that might be meaningful. For instance, we can't just remove symbols like "-" and different kinds of apostrophes, because they play important role in some combined words like "I'm", "don't", "e-mail", etc. Also we can't remove all meaningful punctuation marks (",.?!"), which might cause chaos in corpus and produce n-grams that normally do not appear in sentence. Before handling punctuations, symbol ";" was replaced with "," due to the close meaning. We decided to left punctuation marks ",.?!" and treat them like separate words in our prediction model, because in this case they can be used also for the punctuation prediction, while they can be easily filtered at the final stage of the project. After handling these special symbols all other symbols "):#%$^\~}&+=@/"_]+" were removed.

We did not remove hash-tags but only hash-tag symbols, because they usually are meaningful. Also we decided not to remove stop words from corpus. For more convenience all the cleaning procedures described above were placed into one function, that was used in data parser.

Profanity Filtering Another important task in data cleaning is the removing of offensive and profane words, because we usually do not want to predict them. Just deleting these words might cause some problems with text consistency, so we replaced them with special tag "<badword>". The universal regular expression for the most common profane words was created using freely available bad-words dictionary.

White-space trim and empty string deletion At the next step of data processing the white-space correction was performed. Firstly, all white-space characters at the beginning and at the end of each document were removed, as well as duplicates of them. Secondly, empty strings were deleted from corpus. Finally, special tags "<s>",that means the start of a document, were placed at the beginning of each document in a corpus. We did this for the reason to be able to predict words at the beginning of the text, when there is no words to predict from. All these white-space processing procedures were placed into one function that was used in parser.

n-gram Counts

After we obtained a clean corpus of documents, the next important step in the process of data model creation is a word tokenization and n-gram counting. N-grams might be counted by making n-gram matrices using standard function from "tm" package. But there are some problems with this approach. Firstly, data matrices produced with it are very sparse and mostly consists of zeroes. Secondly, due to the previous fact, it is quite slow process. Thirdly, we have reasonable amount of data, and there are millions of n-grams there. It is not possible to store such a big matrix in PC memory. One way to handle this problem is to do not use matrices for counts storing. For a model creation we need counts only for n-grams that occurred in the data and there is no need in all zeroes in sparse matrix. The better way to store counts is a dictionary-like structures, for instance a list or named vector.

Another problem is in the size of the data. Even very simple operations like indexing slow down significantly when the size of an n-gram storage object increases. To handle this problem we decided to use "map-reduce like" approach. The corpus of data can be mapped to the small chunks very easy, so even without parallel

computing this might improve performance considerably, at the same time avoiding memory restrictions. During the analysis each corpus was mapped to the small chunks and 1, 2, 3 and 4-grams were counted. At the reduce stage all these counts were merged into the total 1, 2, 3 and 4-grams counts.

Map Phase N-grams was counted with the function that takes a small chunk of data as an input and writes 1, 2, 3 and 4-grams counts to global storage objects.

The higher level parser function that takes a name of file, the size of a chunk, start and end points as an input were developed. It uses n-gram counting function to count n-grams and store counts to a hard disk drive into a single file for each chunk.

We decided to count n-grams and to create the prediction model for each file separately. There are some reasons for that, and the main is that prediction models are good only in context of the data. While we still can use models from blog corpus to predict words in news text, we definitely shouldn't do this for twitter data. Data in them is very different and accuracy of this predictive model will be not very high. So we decided to create three different models for each context (news, blogs and twitter).

Reduce Phase In the reduce phase we merged all the n-grams counts from the all files created in the map phase. Three different functions was created to perform this. The first one splits files with 1, 2, 3 and 4-gram counts into four separate files. Second takes two files with n-gram counts of the same rank and merge them together. And the third is a higher level function that goes over all files and merge them for each given n.

n-gram Counts Optimization The final aim of this project is a Web application that might be used on mobile gadgets for the next word prediction and this fact produces some restrictions on our predictive model. It should be reasonably small, does not require a lot of resources, and to have a high performance, because it should work in real-time manner. One possible way to make optimization without affecting the accuracy of the model is to delete some very rare n-grams.

This might also cause some other positive effects. We did not even tried to filter words in corpora to find some misspelled words, words from other languages, and random sets of symbols like "dadasfga sfghasgdf" during the cleaning process. The intuition was that words like these should be very rare in the dataset and after n-gram counting we can just remove n-grams with the lowest counts. We found that the percentage of the most frequent words that cover most n-gram counts differ for different n (presented results are for a subset of original data):

- roughly top 0.6% 1-grams contain 50% of counts and top 30% 1-grams contain 90% of counts
- roughly top 16% 2-grams contain 50% of counts and top 83% 2-grams contain 90% of counts
- roughly top 45% 3-grams contain 50% of counts and top 89% contain 90% of counts

For 3-grams counts are distributed more uniformly than for 2 and 1-grams. This was expected because there are more possible combinations for n-grams with bigger n, and we do not have enough data to make a good coverage for all 3 and 4-grams. That is why removing n-grams with the count 1 might be not the best choice in the case of 4-grams. Despite this fact we decided to remove all n-grams with counts 1 for every n to improve performance.

Exploratory Data Analysis

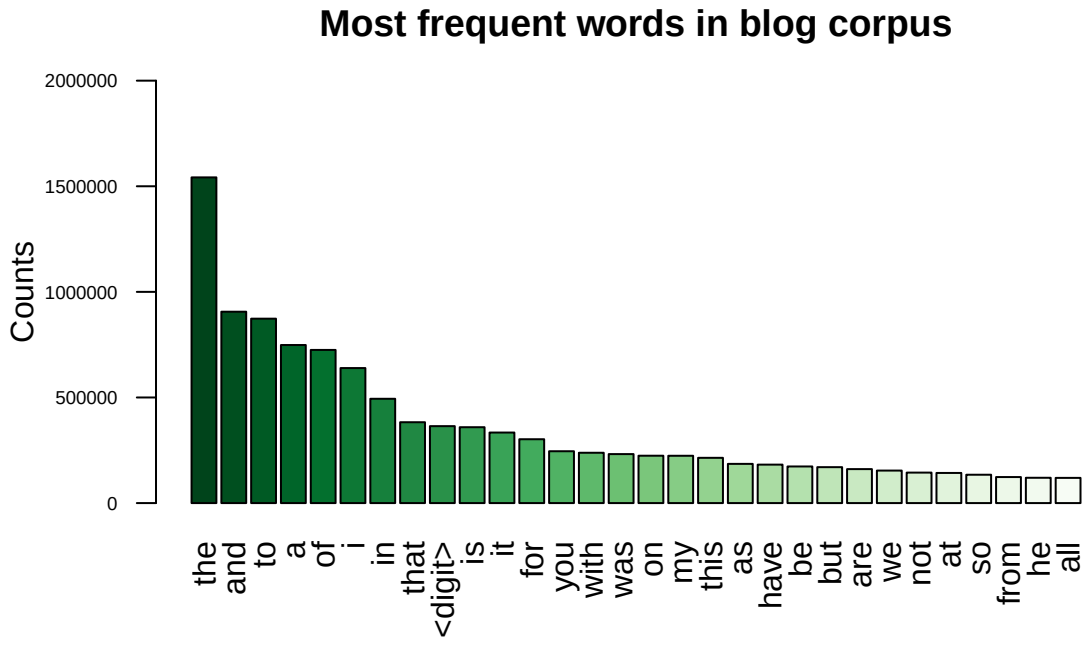
Blog Corpus

As the result of data cleaning and processing we obtained the n-gram counts for blog, twitter and news corpora. Here you can find information about the size of the different n-grams for blog corpus:

```
##            n.gram.size
## one.gram        173730
## two.gram       1559656
## three.gram     2254311
## four.gram      1271028
```

The largest n-gram count object is for 3-grams (as well as for the other corpora in our dataset). This is because we removed n-grams that happened only once in the corpus. 4-grams contain more such "rare" n-grams that other, because there are more possible 4-grams than n-grams with smaller n. We do not have enough of data to cover all common 4-grams, so there are a lot of 4-grams with count 1, even though some of them are not really rare.

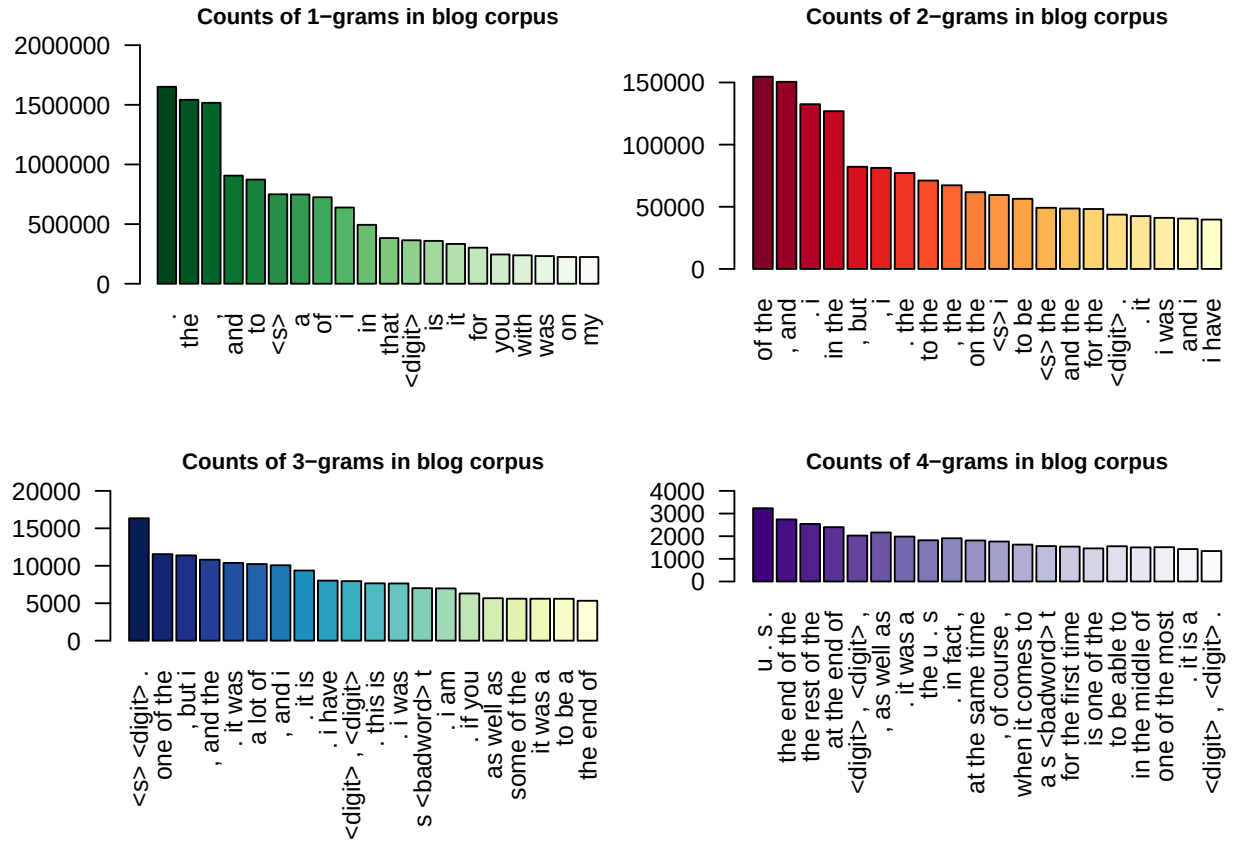
We analysed frequencies of different words in data. The 30 most frequent words in blog corpus are depicted on this graph:



As we can see, the most common words are the "stop-words". It was expected, because we consciously did not remove them from the corpus during the cleaning process. We did not do that because they might be useful for the word prediction Web application, which is the final goal of this project. We might ignore them for some other tasks but predicting the next word in the sentence require them in the prediction model.

Apart the "stop-words", the blog corpus contains many digits.

Here you can see the bar-charts of the most frequent n-grams in the blog corpus:



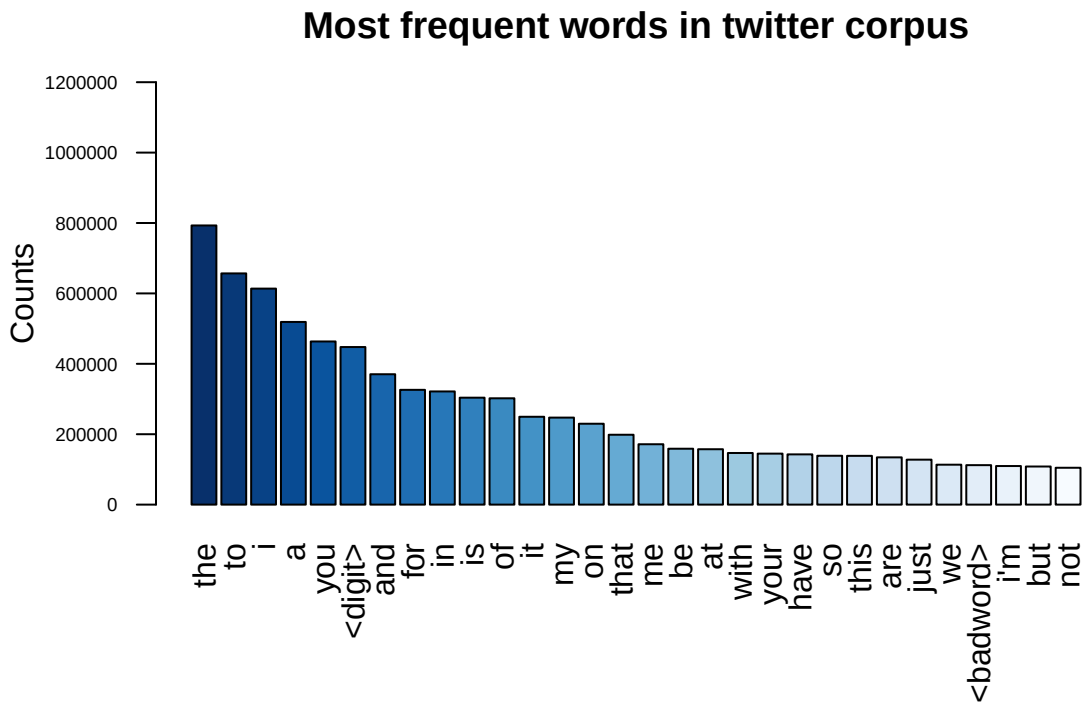
Punctuation marks were not removed from this graph, and we can see that the most frequent 1-gram is a ".". The most frequent 3-gram is "<s> <digit> ." where the tag "<s>" is the start of the document. This probably the specific feature of the blog corpus, because blog posts often have numeration, time and/or date at the beginning. Another interesting feature of the blog corpus is that the most frequent 4-gram in it is the abbreviation "U . S .", so we can conclude that either the data contains a lot of documents posted by bloggers from the United States or US-related discussions were among the hot topics in blogs at the time when the data was collected.

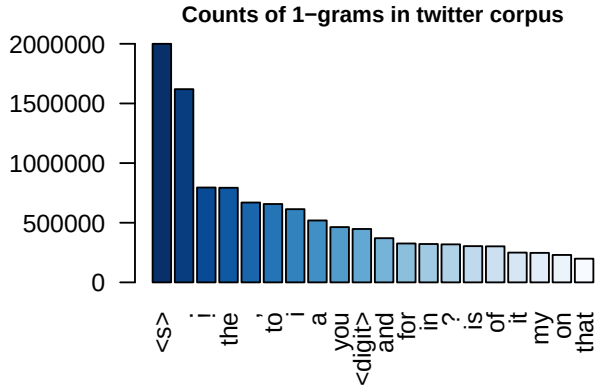
Twitter Corpus

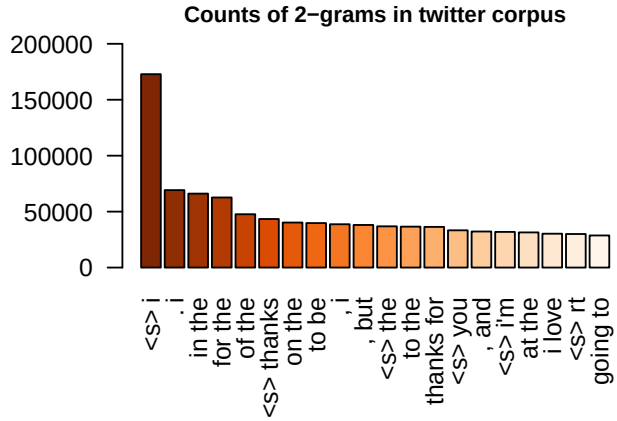
Twitter corpus consists of fewer 1-grams and 2-grams than blog or news corpus. This possibly because blogs and news usually have richer content with more sophisticated language and different contexts, while twits are generally short and consists of more common words. The number of 3 and 4-gram is roughly the same for twitter, blog, and news corpora.

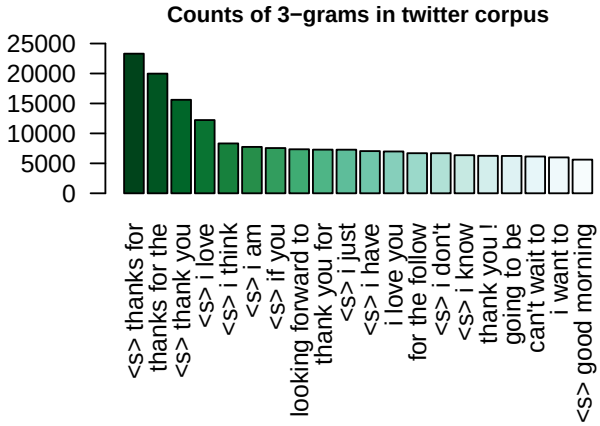
```
##            n.gram.size
## one.gram        145652
## two.gram       1245413
## three.gram     2208647
## four.gram      1312515
```

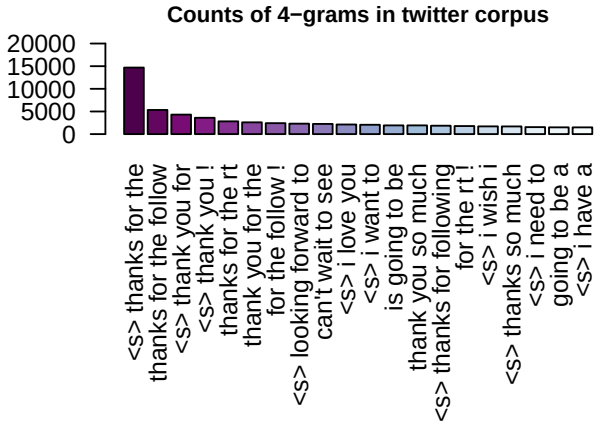
There are also some differences in the most common words frequencies. While the most frequent words are generally also "stop-words" as in the blogs and news data, twitter corpus contains many "bad-words".


Most frequent words in twitter corpus
Counts
1200000
1000000
800000
600000
400000
200000
0
the
to
i
a
you
<digit>
and
for
in
is
of
it
my
on
that
me
be
at
with
your
have
so
this
are
just
we
<badword>
i'm
but
not


Counts of 1−grams in twitter corpus
2000000
1500000
1000000
500000
0
<s>
.
,
the
,
to
i
a
you
<digit>
and
for
in
?
is
of
it
my
on
that


Counts of 2−grams in twitter corpus
200000
150000
100000
50000
0
<s> i
. i
in the
for the
of the
<s> thanks
on the
to be
, i
, but
<s> the
to the
thanks for
<s> you
, and
<s> i'm
at the
i love
<s> rt
going to


Counts of 3−grams in twitter corpus
25000
20000
15000
10000
5000
0
<s> thanks for
thanks for the
<s> thank you
<s> i love
<s> i think
<s> i am
<s> if you
looking forward to
thank you for
<s> i just
<s> i have
i love you
for the follow
<s> i don't
<s> i know
thank you !
going to be
can't wait to
i want to
<s> good morning


Counts of 4−grams in twitter corpus
20000
15000
10000
5000
0
<s> thanks for the
thanks for the follow
<s> thank you for
<s> thank you !
thanks for the rt
thank you for the
for the follow !
<s> looking forward to
can't wait to see
<s> i love you
<s> i want to
is going to be
thank you so much
<s> thanks for following
for the rt !
<s> i wish i
<s> thanks so much
<s> i need to
going to be a
<s> i have a

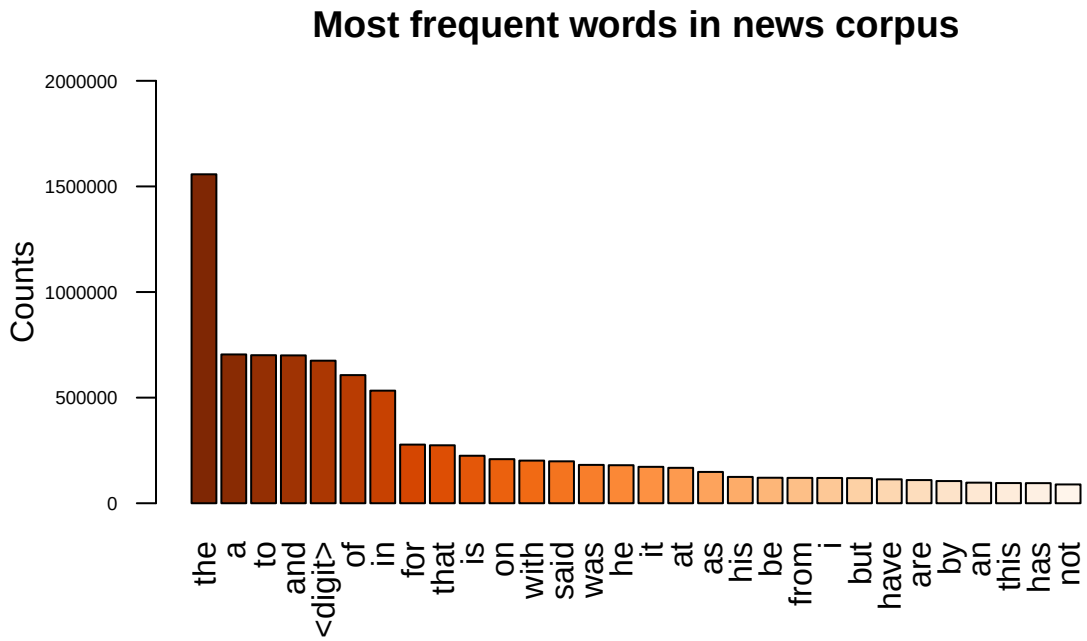
Common n-grams are also differ significantly from the blogs and news corpora. The most frequent 1 and 2-grams in the twitter corpus contain "<s>" tag that means start of the document. This is not surprising, because twits are usually short, and corpus from twitter will have more documents than a corpus of the same size with blogs or news.

Another feature of the twitter data is that most frequent 3 and 4-grams generally contain word "thank". It is probably due to the fact that there are a lot of twits like "thanks for the follow", "thanks for the rt", etc. in the corpus.

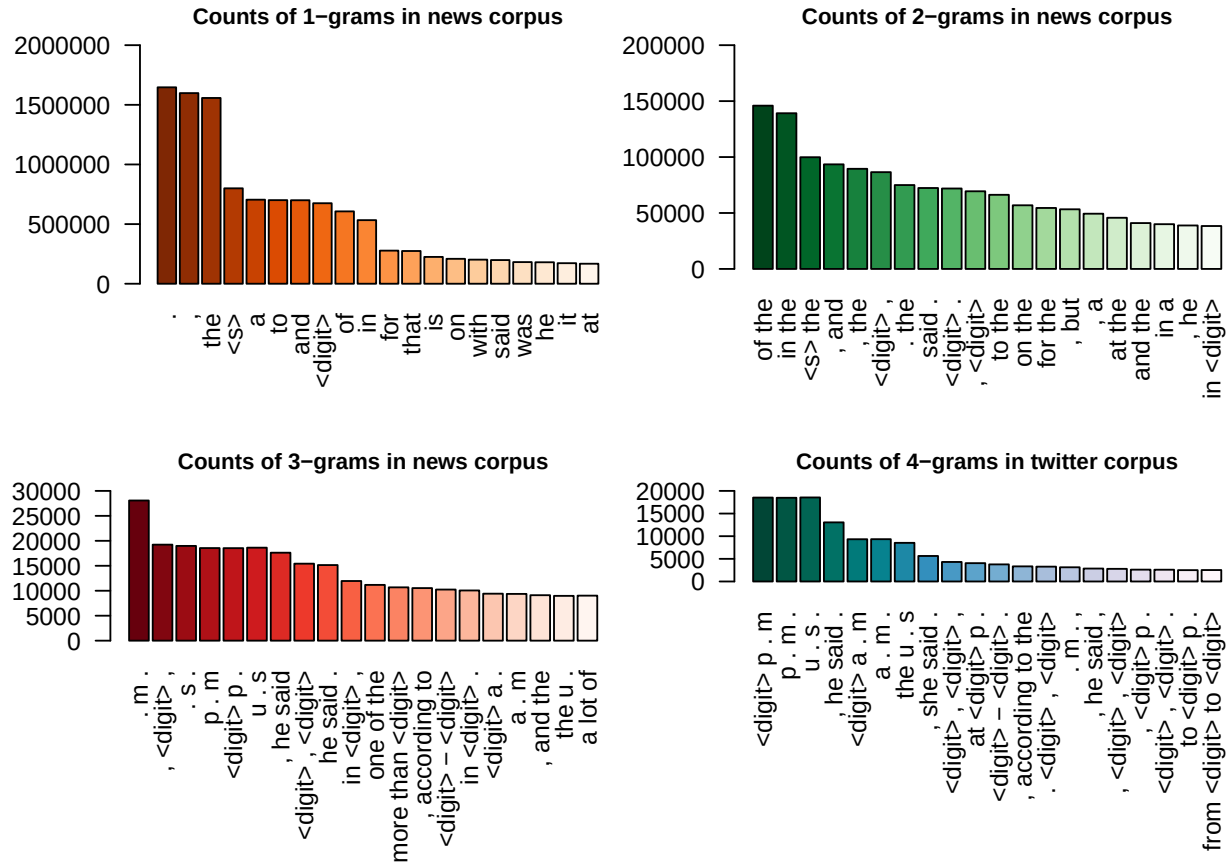
News Corpus

News corpus consists of roughly the same number of different n-grams and have approximately the same most frequent words distribution as corpus with blogs:

```
##            n.gram.size
## one.gram        158183
## two.gram       1560659
## three.gram     2061751
## four.gram      1467307
```



The most common 3 and 4 grams are different in the news corpus comparing to the other. They have a lot of digits and time abbreviations like "a . m .", "p . m ." or their combinations. Also the "u . s ." abbreviation frequent there as in the blogs corpus.



Predictive Model

Probabilistic model

The n-gram counts were transformed into probabilities to have the last word in a n-gram given the previous words. For each n-gram this probability might be calculated as a count of the particular n-gram divided on the count of (n-1)-gram (the same n-gram but without the last word).

After calculating probabilities for each n-gram, they were placed into the model objects (three objects, separately for twits, news, and blogs).

Optimization

Optimization is heavily dependant on the requirements for the final product. So we should describe how the final Web application is expected to work. It have to contain text input form, where user may type the text. Data from this form should be processed with predictive function that returns most probable next words using our n-gram model. It also might return more than one word, but some reasonable number of words to give a possibility to choose the right one by user itself. The goal of this application is to reduce the amount of typing, because it might be not very convenient process on mobile devices. So giving 5 the most probable next words to a user should be a reasonable number of words to choose from. Five items could be easily showed at a mobile device screen.

Taking to consideration provided description of the final product, we can think how our predictive models might be optimized. We do not require more than 5 predictions for each input text, so we can reduce the size of

models by deleting all predictions beside the top-5 for each n-gram (without removing the punctuation marks, they will be treated separately by the predictive function), because they will never appear in predictions. So at the end we should have predictive models with no more than 5 predictions for each n-gram, plus predictions of punctuation marks.

Here you can see the size of models before and after the optimization:

```
## [1] "Twitter model length before and after the optimization:"
##            before   after
## bigram    1560659  280272
## threegram 2061751 1175418
## fourgram  1467307 1227471
## [1] "Blogs model length before and after the optimization:"
##            before   after
## bigram    1559656  264785
## threegram 2254311 1147595
## fourgram  1271028 1004054
## [1] "News model length before and after the optimization:"
##            before   after
## bigram    1560659  280272
## threegram 2061751 1175418
## fourgram  1467307 1227471
```

Besides this, another optimization might be done to improve access to the data. Next word prediction should be almost instantaneous in our App, because potential user will not be waiting for predictions for even small amount of time. Created n-gram models are still have many elements in them, and searching for the right one might take some time if we will try to access data as is. So we decided to make hashing of the models by grouping similar records by the first two characters in each n-gram, which makes access to data much more faster.

Predictive Function

The predictive function uses n-gram models and indexes to predict the most probable next words. It produces a prediction that consists of 5 words at most.

When argument clean is TRUE, it also performs text preprocessing before generating predictions (similar to the corpus cleaning during the model-creation process). At the cleaning stage all digits, e-mails and bad-words are replaced with coresponding tokens, and special characters are removed (except the meaningful punctuation marks ".,?."), as well as excessive white-space characters. At the beginning of each document the special tag "<s>" is placed.

The predicting part of this function uses "backing-off" to the lower n models under conditions when the given n-gram model do not produce a good prediction. By doing this, it returns the most reliable prediction using the best performing model. The backing-of in the function goes from 4 to 3 and 2-gram models.

Smoothing wasn't used for the models, because during the tests it produced less accurate predictions than just using simple "backing-off" without smoothing.

Another feature of the particular predictive function is that it's behaviour depends on the last character in the text. When there is a "space" at this position, it includes punctuation mark into results, so it also predicts those four important punctuations - ".,!?".

Lets try to predict the next words for the 4-grams from the phrase:

"The alacrity of the brown fox contributes to its ability to perform a saltatory action over the lazy dog"

```
## [1] "The alacrity of the"
## [1] " most"  " world" " day"   " year"  " time"
##    user  system elapsed
##    0.38    0.00    0.38
## [1] "------------------------------------------------------"
## [1] "alacrity of the brown"
## [1] " sugar" " ale"   " and"   " paper"
##    user  system elapsed
##    1.22    0.00    1.27
## [1] "------------------------------------------------------"
## [1] "of the brown fox"
## [1] " news" " and"  " is"   " home" " was"
##    user  system elapsed
##    1.09    0.00    1.11
## [1] "------------------------------------------------------"
## [1] "the brown fox contributes"
## [1] " to"            " a"             " more"          " an"
## [5] " significantly"
##    user  system elapsed
##    0.23    0.00    0.23
## [1] "------------------------------------------------------"
## [1] "brown fox contributes to"
## [1] " the"  " a"    " high" " it"   " nbc"
##    user  system elapsed
##    0.37    0.00    0.38
## [1] "------------------------------------------------------"
## [1] "fox contributes to its"
## [1] " own"      " original" " name"     " former"   " knees"
##    user  system elapsed
##    0.36    0.00    0.36
## [1] "------------------------------------------------------"
## [1] "contributes to its ability"
## [1] " to" " is" " of"
##    user  system elapsed
##    0.52    0.00    0.52
## [1] "------------------------------------------------------"
## [1] "to its ability to"
## [1] " produce" " wrap"    " make"    " be"      " do"
##    user  system elapsed
##    0.20    0.00    0.21
## [1] "------------------------------------------------------"
## [1] "its ability to perform"
## [1] " the" " a"   " at"  " on"  " for"
##    user  system elapsed
##    0.31    0.00    0.32
## [1] "------------------------------------------------------"
## [1] "ability to perform a"
## [1] " specific" " certain"  " free"
##    user  system elapsed
##    0.47    0.00    0.47
## [1] "------------------------------------------------------"
## [1] "to perform a saltatory"
## list()
##    user  system elapsed
```

```
##    0.29    0.00    0.31
## [1] "------------------------------------------------------"
## [1] "perform a saltatory action"
## [1] " and"   " is"    " in"    " would"
##    user  system elapsed
##    0.34    0.00    0.35
## [1] "------------------------------------------------------"
## [1] "a saltatory action over"
## [1] " your" " with" " you"  " yet"
##    user  system elapsed
##    0.11    0.00    0.11
## [1] "------------------------------------------------------"
## [1] "saltatory action over the"
## [1] " years" " past"  " last"  " next"
##    user  system elapsed
##    0.06    0.00    0.06
## [1] "------------------------------------------------------"
## [1] "action over the lazy"
## [1] " days"  " eye"   ","      " river" " to"
##    user  system elapsed
##    1.00    0.00    1.02
## [1] "------------------------------------------------------"
## [1] "over the lazy dog"
## [1] " and"   " would" " you"   " years"
##    user  system elapsed
##    1.08    0.00    1.11
## [1] "------------------------------------------------------"
```

As we can see, the predictive function works, but to evaluate the accuracy of it a large scale accuracy test should be performed. It will be described in more details in the next segment of this report.

Performance Testing

To calculate how well the models perform, we used the parts of files that was considered as the test sets, and which have not been used for the models creation:

1. en_US.twitter.txt - last 360,148 lines at the end of the file
2. en_US.blogs.txt - last 149,288 lines at the end of the file
3. en_US.news.txt - last 210,238 lines at the end of the file

To obtain the accuracy of the word prediction algorithm the following steps were performed:

1. 4-grams and their counts were generated from test sets.
2. There were randomly sampled 1000 4-grams from each set for testing.
3. Each 4-gram was separated on the first 3 words (which were used for prediction) and a last word (to compare result with).
4. The prediction of at most 5 words was generated using the predicting function, and the real last word from 4-gram was searched in vector with predictions. This produced True or False (1 or 0) result for each 4-gram.
5. The results (0 or 1) for each particular 4-gram was multiplied by the counts of it. After doing this for all 4-grams, their counts were summed and compared to sum before running these tests. We did this because not every 4-gram have the same weight for our ability to predict, and their counts represents

frequencies of them in the real data. So counts were used as weights. For instance, a 4-gram with count 100 will happen 100 times more frequently than a 4-gram with count 1, and it at the same time will be 100 times more valuable for prediction.

Here are the results of prediction accuracy for all three models (models were tested in context of the data - twitter test set was used for the twitter model testing, etc.):

```
## [1] "Word prediction accuracy for news n-gram model:"
## [1] 0.2814
## [1] "Word prediction accuracy for blogs n-gram model:"
## [1] 0.2612
## [1] "Word prediction accuracy for twitter n-gram model:"
## [1] 0.3352
```

Web Application

Based on the obtained predictive function and optimized models, the demonstrational Web application was created. Here are some features of it's functionality:

- It predicts the most probable next words using backing-off n-gram model
- Prediction consists of 5 words or less to choose the right one by user itself
- The are three different modes for predicting (twitter, news, blog) for different kinds of texts
- When user types just a part of a word the application is trying to predict the rest of it
- While the application does not predict profane words, they still can be in the user's text and this does not affect the quality of prediction
- App includes punctuation marks into prediction results when the last character in the user's text is not a "space" character
- The predictions are not affected by the proximity of the document start. App will try to predict the next word even if the document is empty
- It can predict words that follow after e-mails or digits. Also it predicts abbreviations like U.S., p.m. and others.

The application can be accessed here: Word Prediction App It might take a few seconds for application to load.