ERROR CORRECTION USING PROBABILISTIC LANGUAGE MODELS

A Dissertation

by

GOWRISHANKAR SUNDER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee	Anxiao (Andrew) Jiang
Committee Members	Yoonsuck Choe
	Tie Liu
Head of Department	Dilma M. Da Silva

May 2015

Major Subject: Computer Science & Engineering

ABSTRACT

Error Correction has applications in a variety of domains given the prevalence of errors of various kinds and the need to programmatically correct them as accurately as possible. For example, error correction is used in portable mobile devices to fix typographical errors while taking input from the keypads. It can also be useful in lower level applications - to fix errors in storage media or to fix network transmission errors. The precision and the influence of such techniques can vary based on requirements and the capabilities of the correction technique but they essentially form a part of the application for its effective functioning.

The research primarily focuses on various techniques to provide error correction given the location of the erroneous token. The errors are essentially *Erasures* which are missing bits in a stream of binary data, the locations of which are known. The basic idea behind these techniques lies in building up contextual information from an error-free training corpora and using these models, provide alternative suggestions which could replace the erroneous tokens. We look into two models - the topic-based LDA (Latent Dirichlet Allocation) model and the N-Gram model. We also propose an efficient mechanism to process such errors which offers exponential speed-ups. Using these models, we are able to achieve upto 5% improvement in accuracy as compared to a standard word distribution model using minimal domain knowledge.

NOMENCLATURE

HMM	Hidden Markov Model
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
SVD	Singular Value Decomposition
ER	Erasure Ratio
IID	Independent and Identically Dist1ributed
BOW	Bag of Words
POS	Parts-of-Speech

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Errors are present in all storage media and are one of the primary hurdles on the path to efficient Data Storage. They occur in different formats originating from a variety of sources. The characteristics of the errors vary depending on those of the source of the error. We look into one of the most common sources of errors - random noise. Errors originating from such random noise sources are difficult to fix since they do not follow any patterns. This makes the task of correcting the errors all the more difficult.

1.1 Why is Error Correction Important?

The ability to correct errors accurately will improve the reliability of the underlying system and so has numerous commercial and academic implications and is an active field of research. Providing such ability in an automated way provides a seamless experience to the end user wherein the presence of the error is totally hidden from them via the correction mechanism in between. Such error correction systems are present everywhere, right from the mobile devices which have text auto-correction capabilities to the numerous complex systems on the internet starting from the search engine which suggests corrections to our queries in order to get better results.

1.2 Natural Language Processing for Error Correction

There are multiple ways to look at the problem of error correction and Natural Language Processing techniques is one of the them, although it has been gaining traction pretty recently only. These techniques build models from the training corpora which can be used to make accurate inferences on possible corrections for an error. Models built this way utilize many attributes of the natural language, including the proximity of words, their role/part of speech in the sentences, their relationships with their neighbors

etc... and their accuracy depends both on their ability as well as the semantic similarities between the training data and the test data.

1.3 What are Erasures?

One of the major problems with error correction is identifying the presence and location of errors. For example, given a stream of valid English tokens out of which a select few are erroneous, it is hard to distinguish such errors from the correct ones as long as they too are valid English words. NLP does provide a few tools to deal with such errors which take into account the neighborhood of the words and identifies any outliers based on their relationships. Some of the information that can be used to identify such errors are

- Neighbourhood similarity - Reduced similarity with the closest neighbors of a given word indicates that it is a possible erroneous token
- Temporal Similarity - Another possible similarity metric would be time-based. For example, in a conversation happening during the day, if you come across a phrase "Good Night", this might probably point to something out of place.
- Topic Similarity - If a set of words are totally out of place wrt. the general topic of the rest of the document or conversation, it could possibly mean the presence of errors.

Although such means exist to narrow down on the possible error candidates, none of them are deterministic and by using such detection mechanisms, we run the risk of identifying false positives i.e. assume valid tokens to be errors and replace them with a possible alternative thereby introducing an error. Such a scenario is detrimental to the effective functioning of the error correction model.

In our research, we primarily focus on errors in binary data streams and only a specific type of error, namely *Erasures*. An erasure is essentially a bit in the stream that is unknown i.e. it could be either a zero or a one. Such errors provide both their presence as well as their location which makes the task of error detection obsolete letting us concentrate on the more important problem of fixing them. Erasures can happen during network transmission wherein one of the bit in a packet of binary data was lost and so, is filled with some sentinel value to be taken care of by the application. OCR software also provide similar inputs wherein they are unable to exactly identify a particular character and instead provide a list of possible characters with varying confidence.

1.4 Erasures in English Text

Erasures are more common in binary streams wherein a bit can go missing and needs to be replaced as explained above. Attempting to correct such erasures needs work at the binary level but we instead try to approach the same problem at a much higher level. Assuming that the binary stream actually represents English text, we can replace these erasures with multiple decodings of English tokens and using word distribution models as described below, deduce the most probable correction for the erasure. Without loss of generality, the English tokens are assumed to be built of only lowercase English alphabets (a-z) along with a few other special characters.

So, the primary assumption we make here is that the binary stream which is filled with erasures is a representation of English tokens in some form. One of the most commonly used binary codings for languages is the Huffman Code [9] It is an optimal prefix code which uses *variable-length code* to represent different alphabets of a language, usually based on their frequency of occurrence with the aim of reducing the overall length of the code. In our case, we built the Huffman Code based on the frequency of each alphabet in the training documents. A table consisting of the generated Huffman Codes is available

in the Appendix.

A stream of binary encoded English tokens with erasures among them could lead to numerous possible decodings. Replacing each missing bit with either a one or a zero would lead to its own possible decoding. So, for a binary stream with k erasures, we have upto 2^k possible decodings. Our aim is to select the most probable among them that can replace the erroneous token while fitting into the remaining text and possibly resemble the original text. Given the fact that Huffman coding is a variable-length code, the possibilities are huge if we do not assume the knowledge of word boundaries in the binary stream. To avoid this situation, it is assumed that the word boundaries in the binary stream are known [1].

1.5 The Erasure Model

In all the test data that were used here, we introduced **IID** (Independent and Identically Distributed) erasures at random positions in the original Huffman-coded binary stream. By this, we mean that the presence of an erasure at a particular position in the stream has no effect on the position of erasures in the remaining stream. This closely resembles how erasures are induced through random noise. The only constraint we place on the erasures in a stream is the proportion of erasures as compared to the entire stream. We denote this by ER - $\textsc{Erasure Ratio}$ which equals the ratio of erasures to the total bit-length of the stream. Higher the ER value, higher the number of erasures in the binary stream. With more erasures, we get more possible alternatives for each word which makes correction process that much harder. Hence, increasing ER values provides a way to test the effectiveness of each of the model as will be seen in future sections.

[1] Please note that it is possible to apply similar language models without knowledge of the word boundaries and the results would probably be similar across every known model and so, should not hugely affect the improvements we show here. The resulting explosion in possible corrections would make the models more complex though - so, we do assume this knowledge to keep the model simple

$$\text{Erasure Ratio, ER} = \frac{\text{number of erasures in the stream}}{\text{binary-length of the stream}}$$

Based on these facts, a specific problem definition that is being solved here follows.

1.6 Problem Definition

A text stream consisting of a sequence of tokens is given. Each of the tokens is constructed of a known alphabet set. The text is encoded as a binary stream, using a pre-determined set of Huffman Codes. This encoded binary stream contains erasures at random positions. The goal is to correct these erasures so that the resulting tokens obtained by decoding them matches the original text as much as possible.

1.7 Significance of Research and Its Challenges

As mentioned earlier, the ability to fix errors (or erasures in our case) programmatically goes a long way towards building a reliable data storage system. One of the challenges of building such a generic correction model is to avoid or at least limit the use of any prior knowledge of the data being corrected. This will go a long way in helping the correction system to be applied to a wide variety of use cases. Most prior work in this area use intrinsic knowledge of the data distribution in some form or another. For example, web-scale error correction mechanisms used in search engines currently in place take into account a lot of internet-related factors like the user's profile, his geographic location and temporal similarities etc. Although such knowledge is useful, it might not be available in all scenarios and so, what we need more is a general model that can be plugged into any domain with the least information. This is exactly what we are trying to achieve through our research.

Our model is formulated in a way that is most conducive for alternate models to be included. We have built a probability distribution based ranking model and the aggre-

gations performed are so aligned that including the results from a similarly normalized model is straight-forward. In fact, we ourselves have built and experimented with a hybrid model aggregating the results from two of our models and it has shown strong improvements. This ability makes the model more generic in the sense that it allows for more information to be included into the system as available. For example, if it is being applied in a domain wherein a different evaluation system could provide better results, it could directly be included into our model and the results should start reflecting the changes there.

1.8 Using Probabilistic Language Models

Error Correction can be precisely solved using probabilistic Language Model approaches. Such models build a word probability distribution based on the available training corpora and try to fit in unknown documents into this distribution. The initial distribution that is built drives the model in suggesting alternatives for potential errors. This model makes practical sense in that the way the words are distributed in a corpora pretty much defines the properties of the corpora and future documents which belong to similar data would inherently follow similar word distribution. So, in spite of being a primitive approach towards identifying alternatives, it works with great precision in many cases. Our research uses two such models to correct errors (or erasures in our case).

- The first one is the **Latent Dirichlet Allocation model** which is a complex generative model that can identify topics of similar text and classify newer documents accordingly. Using this model, we were able to classify newer text onto different topics and using the word distribution in each of those topics, we arrive at the best possible alternative. Although the underlying principle of correcting errors is mostly similar to that of using a global distribution model, the major difference is the use of topic-specific word distribution. The results have shown that this

method is more effective, probably due to the fact that it takes in more contextual information while arriving at the results.

- The second model is the **N-gram graph** model which builds a graph with N-grams as nodes. The N-grams use word relations wherein neighboring words in text form N-grams. The graph model is a slight variation of the regular N-gram model since neighbourhood exists between words which are not necessarily immediate neighbors in the text. This modification helps the model perform better and also allows us to generate the N-gram data from smaller datasets.

Both models depend on the word distribution in the training document as explained above and they try to overcome the inherent noise in such plain distribution models by using spatial similarity among words. The baseline to compare for each of the models is a general word distribution model which only accounts for the overall frequency of occurrence of each word across the entire training set.

As part of the research, we also came up with efficient methods to decode an erasure-filled Huffman coded binary stream. It provides exponential speed-ups as compared to regular decoding and has been helpful in testing the above to models on high ER values using a reduced amount of processing power and time.

1.9 Related Work

Language models have been in use to predict and correct grammatical, spelling or even contextual errors in multiple domains in various forms. For example, the auto-corrector available in word processing software or even on hand-held devices are mostly depending on an underlying language model that drives the results. Each such application has its own set of constraints under which it needs to work. Similarly, they also run on a few assumptions about the domain that they are being applied on.

1.9.1 Topic-based Language Model

Although global word distribution models work well in an abstract environment, they are less efficient in utilising any other inputs available in the system. For example, they do not take into account the spatial neighbourhood of other words when given a stream of words. This is where a topic-sensitive model is more helpful as it is able to leverage these signals to arrive at better results. Sophisticated topic models have been found to be extremely effective in relating seemingly unrelated semantic constructs and variations of such models have been extensively used in a wide variety of places.

Zweig et. al. [22] have tried using the Latent Semantic Analysis or LSA technique in combination with other techniques including N-gram models, Neural Network models etc. In fact, they have also used a linear combination of the results from two models, namely the LSA model and the N-gram model, to arrive at a much better performing model which is similar to what we have implemented towards the later part of the research. Yuret at. al. [23] have proposed solutions towards generating corrections making use of *Wordnet* training data and a thesaurus but it is pretty complicated, especially related to the Thesaurus part and is not entirely applicable in a practical sense. An *Entropy-based Language model* [5] provides a class-based abstraction helping in better predictions. In such a model, an hierarchy of grammatical constructs is developed over which the general N-gram models are run. This approach can provide marginally better results since the hierarchical abstraction can help in isolating different N-grams to the level they belong to thereby avoiding noise in other sections. This is very similar to the topic classification we aim to implement to avoid noise from other topics.

The LDA model has been efficiently used in [20] to correct errors in Optical Recognition Software. OCR software do not always detect characters deterministically and hence, are prone to errors based on similarity of characters (for example, hand-written versions

of characters 'i' and 'j' are pretty similar) But such software can associate a confidence percentage to their detection and using knowledge of commonly misinterpreted characters, [20] have made improvements in correcting errors in the system. Essentially, they have built an error model based on the probability of each character being misinterpreted as another using which they are able to detect all possible corrections for a given word. Armed with this information, they use the LDA model to rank such alternatives and choose the best among them. Using this model, they have been able to reach upto 7% improvement in accuracy as compared to global word distribution model, which reiterates the fact that a topic-sensitive model will significantly outperform a global model in most cases.

This model is in some ways similar to the approach we take to correct errors i.e. instead of correcting errors in OCR recognition using topic distributions, we are trying to correct erasures which are present in the binary representation. One of the drawbacks of their approach is that the error model they have used is restricted to the OCR domain. In other words, it is hard to arrive at such a probabilistic error model in every domain although it might suit well for OCR software. This forms the fundamental difference between our approach and theirs.

Topic-sensitive models are also being used in seemingly unrelated domains. For example, Wenget. al. [19] have used *TwitterRank* based on topic similarity between user tweets and their user-link topology information to measure influence among the users. Given the ability of the LDA model to identify hidden relations among tweets, their measures of influence outperform those developed using he usual algorithms like PageRank[16] and even Topic Sensitive Page Rank[7]. Similarly, a topic-based model has been used to help improve Collaborative Filtering in Recommender Systems in [13] More recently, [14] have used a modified version of LDA to classify Yelp user reviews by topics like "price" or "service" based on which the ratings can be weighted better. Similarly, [8] have used

LDA to identify hidden subtopics in Yelp reviews to identify customer demands more specifically.

These items depict the potential of the LDA model to identifying implicit relations among words efficiently using which we can arrive at better results for error correction. The first part of the research consists of using the LDA model for error correction. We can see that the model is able to perform pretty well without much interference once given sufficient training data.

1.9.2 N-gram based Language Models

As stated earlier, their primary advantage of N-grams over other simpler language models is their ability to expand the contextual range of information. This has been previously used by others for various purposes. [12] have introduced similarity measures for N-grams which could be used to identify the right alternative for a given erroneous word and since it is a similarity measure, it does not hugely suffer from the limitations of a plain N-gram model. They use a version of N-grams wherein the most granular item in their N-grams is characters rather than whole words. This way, they have used this similarity measure in parallel to other text similarity measures like **Edit Distance**.

[2] contains a few articles which work with web-scale N-grams to improve search quality. One possible example where they are used is to auto-suggest alternative search engine queries. These are handled using N-gram models built over previous valid search queries. Although they are able to arrive at useful approximations in large-scale test data as in the case of web queries, they are less effective when applied to errors in a much smaller scale. In most such cases, the primary source of noise that deteriorates the results is irrelevant words, which could be stop words, connectors or any other semantic construct that does not add any new information to the models.

A variation of this model, namely the N-gram graph model, has been used recently

to rank similarities between a text and its summary. [6] have used a framework called *AutoSummENG* that runs over such a graph to deduct similarities among a text and its summary. It basically evaluates each possible summary for the text by measuring its N-gram based similarity with the original text and the summary with the highest similarity can be chosen to replace the original text. Although the problem here does not exactly relate to effective text summarization, some ideas related to building the graph have been utilized. We aim to build similar N-gram graphs and use the relationships among the N-gram nodes to identify the most likely replacement for an error.

1.10 Experimental Data Used

We have used the English dataset from the Newsgroup Dataset [2] to test these models. It is a collection of around 18,500 newsgroup articles classified into 20 different topics. This specific dataset was chosen for the following reasons.

- The dataset is classified into 20 topics based on the original newsgroup topics, which also is suited well to test the topic-based LDA model.
- This dataset has been widely cited and hence gives a good benchmark to test the models. Specifically, the OCR-based paper [20] that uses LDA to correct errors have also used this dataset.

The dataset is split into a training set consisting of approximately 11,500 documents and the remaining documents forming the test set. This maintains an approximate training set to test set ratio of $60 : 40$ across all topics. All the models are preprocessed on the training set and the resulting data structures are stored onto the file system. On each invocation, the model is loaded onto memory and is instantly ready to start correcting errors in the document.

[2] `http://qwone.com/~jason/20Newsgroups/`

Given that the articles have been collected from newsgroups, they have a lot of email/web related metadata that needs to be filtered out so that we only have valid English words. This pre-parsing is performed on each document before inducing into the model so that the resulting text consists only of lower-case English alphabets and a few special characters as mentioned earlier. The resulting data consists of around 1.5 million words of which around 36,500 words are unique. This set only includes words that are larger than 3 characters. The reasoning is that

- Smaller words are primarily stop words - words which do not add any contextual information to either the LDA or the N-Gram model. Such words can be corrected using a more general word distribution model.

- Given that these words constitute most of the noise in the models generated, discarding them allows us to build more concentrated models which are able to perform better error correction.

The experiments have been performed using Python. More information on the choice of language and other tools can be found in the Appendix.

1.11 Brief Summary of Results

Below, we have listed the major improvements achieved through these models. We have used the global distribution model as the baseline for comparisons since it is the most widely used word distribution model.

- The LDA-model has been able to achieve upto 4% improvement in accuracy over the global model.

- The N-gram graph model, essentially an improvement over the regular N-gram model, has been experimented using various parameters. Although its performance is not better than the global model, it complements the LDA-model well and so, an

Hybrid model has been constructed using a linear combination of the results from the two models and this has shown upto 3% improvement over the LDA-model itself.

- An optimized approach to decode Huffman-coded binary streams with erasures has been developed. This has led to an exponential decrease in running time for the decoding process which has enabled more efficient experiments on the other models.

In the reminder of the thesis, we will describe each of these models in greater detail. Section 2 explains the LDA model and its results. Section 3 outlines the N-gram model and also the Hybrid model we developed which improves the previous model's performance. Finally, Section 4 explains the optimization over Huffman decoding with erasures which was used in the experiments on the above two models.

2. A TOPIC-SENSITIVE WORD DISTRIBUTION MODEL BASED ON LATENT DIRICHLET ALLOCATION

2.1 Word Distribution Models

In order to use NLP in error correction, one of the most commonly used models is the statistical language models [1]. Such models essentially provide a probability of any phrase or semantic construct in a language occurring together. These models are mostly built on training data that are assumed to closely match the semantic and contextual characteristics of test data as well. The assumption is that the word distribution or their frequencies remain the same across the documents which leads to the conclusion that the model built over the training set can be used as a generative model for any future document.

The preliminary version of the model consists of the frequency of each unique word w across the training document and so, any word W is assumed to have a probability of occurrence $P(W)$ in any document[2]. To apply this to the error correction domain, suppose we have a stream of words of which one of them is unknown, the probability that the missing word is w_i is given by $P(w_i)$ and so, of all the words in the dictionary, the one with the highest probability of occurrence is assumed to be the missing word. We should note here that in our erasure model, the erroneous word might have some of its bits missing and replacing them might lead to a set of possible alternatives. So, we only need to account for those words when comparing the probability and not the entire vocabulary. But still, we can see that such a simple model will not be accurate enough for larger corpora with a pretty large vocabulary since there might be more than

[1] A *Language Model* is a function that provides a probability of occurrence for each semantic construct in the language

[2] This is similar to the N-gram model that will be discussed later. Specifically, this can be assumed to represent a *unigram* model wherein the value of N is 1

a single word with similar frequency and the model does not provide much information on choosing one among them.

A simple extension of the model is to expand the length of the semantic construct from a single word to a stream of words[3]. For example, assuming a similar frequency model has already been constructed which provides the probability of occurrence for a word w_i as $P(w_i)$, and given a sentence consisting words t_0 through t_n of which t_j is known to be an error where $0 <= j <= n$, the probability that the word W is the right alternative is given below. The word that results in the highest probability of the phrase to be generated is taken as the alternative.

$$\prod_{i=0}^{j-1} P(t_i) \times P(W) \times \prod_{i=j+1}^{n} P(t_i)$$

2.2 Why Standard Word Distribution Models are not Sufficiently Accurate?

Although this model seems to take in more contextual information while arriving at a decision, it still suffers from the locality of its neighbourhood i.e. the neighbourhood of each word is immediate and hence, the model does not take in a holistic view of much larger neighborhoods like the whole document or even the class of documents. This results in myopic decisions which might not work well in many cases.

For example, consider a sentence like *the frog is* **swimming** *in the pond* where the emboldened text is missing. Going by the model above, the probability of the word **swimming** correctly replacing the error depends not only on the words *frog, pond* which help us deduce the correct word but it also takes into account the remaining words which are mostly stop words or connectors which do not add any valuable information in the correction process. But if we had taken a much larger semantic construct like the entire paragraph or the whole document and assuming the entire document was written about

[3]This resembles the N-gram model mostly

ponds or frogs, arriving at *swimming* as the right alternative would have been much easier. This is where a topic-sensitive model is much more accurate.

2.3 Topic-sensitive Distribution Models

A topic-sensitive distribution model can overcome the above problems by building the frequency model specific to a topic. This way, the model inherently takes into account the word distribution for the topic alone which helps in arriving at better corrections for an error. In the example above, if the sentence occurred in a document under a topic related to water bodies or frogs, deducing the correction to be *swimming* could have been much easier. Building such a topic-sensitive model is not very different from the general distribution model explained earlier. The only change is that the corpora is pre-classified into topics and the models are built individually for each of the topic. To correct an error, the topic of the entire document defines the model to use from where the rest of the steps remain the same.

When given the right classifications, this model might be able to work well but we might not always be able to clearly classify a training document into different topics. Even if we are provided with that information, assuming the topic of a given test document to be known significantly reduces the scope of the problem. For example, the sample Newsgroup data that was used for testing comes classified into different topics which might help in building the topic model but to be able to correct any given document, this topic based model does not provide means to identify the topic of the given document, which is required for the model to be effective.

2.4 An Illustrative Topic Model Example

To illustrate how a topic model could outperform a global word distribution model, we will take a look at the example below.

A computer is a general purpose device that can be **programmed** *to carry out a set*

of arithmetic or logical operations automatically. Since a sequence of operations can be readily changed, the computer can solve more than one kind of problem. Conventionally, a computer consists of at least one processing element, typically a central **processing** *unit (CPU), and some form of memory. The processing element carries out arithmetic and logic operations, and a sequencing and control unit can change the order of operations in response to stored information. Peripheral devices allow information to be retrieved from an external source, and the result of operations saved and retrieved.*[21] [4]

In the example above, assuming the words boldened are to be corrected, we know that any document related to computers will have the words "programmed" and "processing" with a much higher probability that any unrelated document. So, in an erroneous document, we have a word to be fixed whose alternatives are processing, writing, running based on the various possible decoding of the Huffman code and we know the probability that the remainder of the document is regarding computers, we would give a higher rank to "processing" which in this case would be the right option. The challenge lies in being able to identify the topic of the remainder of the document so as to make this prediction.

2.5 Why use LDA?

This is exactly where the Latent Dirichlet Allocation or the **LDA** is helpful. The formal paper on LDA explains it to be a *three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics*. More information on its structure and functioning can be found in [4]. It is fundamentally a generative model that allows a set of observations to be explained by hidden classifications which inherently explains why they are similar. This is closely related to our topic-based model definition earlier where the hidden classifications are the topics we talked about.

[4]Wiki source shared under the terms of Creative Commons Attribution Share-Alike license - http://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License

This is the reason why LDA can be used to classify text into various topics. It is able to build relationships among words across different documents based on their neighbourhood similarity to some extent. LSA is a similar model that compresses associations among words to be used as a distribution model. But it suffers from its inability to scale to larger datasets where LDA is more accurate.

The LDA model assumes each document to be a mixture of topics rather than belonging to a single topic. This applies to both the training as well as test documents. This relates well to practical cases since a document need not be specific to a single topic alone. For example, a scientific document could also have references to mathematics, economics and numerous other topics and so, classifying under the topic "Science" alone strips of other important information on the document. This is similar to the working of **Fuzzy Clustering Algorithms**. It has the following major advantages as compared to a plain topic-model.

- It does not require the documents to be classified into topics
- It is able to classify both the training and the test documents into different topics as configured. This way, it removes the need to be predetermined grouping for each topic
- It only uses a probability distribution of topics for a document and so, allows every document to be a part of multiple topics. Using such a model for error correction would allow words from multiple topics to influence the decision at varying levels which might probably lead to better results.
- The number of topics to be classified is also flexible. This way, many levels of abstractions of the topics can be created depending on the resources available as well as the granularity of the available documents and topic information.

2.6 Applying LDA to Error Correction

LDA model is self-generative in that it can identify and classify the documents into topics. Since data in general does not lend itself to be classified accurately into various topics, this property of LDA is very useful in classifying even unknown documents. In order to construct the model from the training documents, we only need to provide the number of topics to be classified into and the model takes care of associating each document to the various topics based on semantic relationships. Once the model is built, any document to be corrected for errors is evaluated by the model resulting in a probability of distribution across each topic. Using this information and a previously built topic-specific word distribution, the alternatives are ranked for a given error and the highest ranked alternative is used to replace the error.

2.6.1 Building the Topic-specific Word Distribution

To be able to use the LDA model to rank possible corrections for a given word, we first need to have a topic-specific word distribution. Building such a distribution would allow test documents to first be classified across the different topics with varying probabilities using which we can rank the corrections. But given the fact that there is no assumption of knowledge of topic classification, we first need to train the LDA model to be able to classify even the training documents across different topics.

The LDA model is basically built over a *bag-of-words, BOW* representation of each document. This means that the model does not concern itself with the spatial associations between words within a document[5]. So, we build an individual BOW-representation of each document and feed it into the trainer. Once all the documents are fed, the model is ready to classify any future document. This is explained in Algorithm 1

Using the model constructed above, we re-run the training documents and based

[5]The N-Gram section that follows later tries to leverage this information

Algorithm 1 LDA Model Construction

```
    procedure LDA(documents, num_topics)        ▷ documents is the list of training documents
        documents_to_bow = φ                    ▷ contains the documents as a bag-of-words
3:      for each doc ∈ documents do
            tokens = PARSE DOCUMENT(doc)
            documents_to_bow =
              documents_to_bow ∪ CONSTRUCT BOW(tokens)
6:      end for
        lda_model = BUILD LDA MODEL(documents_to_bow, num_topics)   ▷ build the LDA model using the BOW constructs
        return lda_model
9:  end procedure
```

on the topic distribution received, we build a topic specific word distribution model. This has been explained in Algorithm 2. The resulting distribution $topic\_spec\_dist = P[word|topic]$ can be constructed as follows where we first build a frequency distribution for the words among the topics and then normalize it to arrive at the required probability distribution.

$$C[word|topic] = \sum_{d \in D} P[topic|d] \times C[word|d], \forall topic \in T, \forall word \in V$$

$$P[word|topic] = \frac{C[word|topic]}{\sum_{w_i \in V} C[w_i|topic]}$$

2.6.2 Ranking the Alternatives of an Error

Once both parts of the model are built as described earlier, we can start using it to correct the errors. The first step towards correcting any document is to retrieve its topic distribution from the LDA model by feeding it the BOW representation of all the error-free words in the document. Let us represent this distribution as $P[topic_i|d]$ which

Algorithm 2 Building Topic-Specific Word Distribution

```
procedure TOPIC DISTRIBUTIONS(documents, lda_model)   ▷ documents is the list of training documents
                                                      ▷ lda_model is the model built from above topic_spec_dist = None
    for each doc ∈ documents do
        tokens = PARSE DOCUMENT(doc)
        bow_repr = CONSTRUCT BOW(tokens)
        dist = lda_model[bow_repr]
        for each topic ∈ dist do
            for each token ∈ tokens do
                topic_spec_dist[topic][token] = topic_spec_dist[topic][token] + dist[topic]   ▷ This builds a count distribution of the words
            end for
        end for
    end for
    NORMALISE(topic_spec_dist)                        ▷ This step structures it as a probability distribution from its current count distribution state
    return topic_spec_dist
end procedure
```

represents the probability that the document d belongs to the topic $topic_i$. Then, the erasures are run through a decoder which uses Huffman Codes and an English dictionary to identify a subset of possible corrections. Then, using the distribution above, each correction alt can be ranked with a probability of

$$P(alt) = \sum_{topic \in T} P[alt|topic] \times P[topic|d]$$

The highest ranked alternative is chosen to replace the erroneous token.

2.6.3 Enhancing LDA Topic Predictions Using Global Distribution Models

While building the topic distribution of a specific document to be corrected, we can only use those words which are error free. But with higher values of ER and the IID nature of the erasure distribution, the proportion of error-free words in a document

Algorithm 3 Error Correction Using LDA

procedure CORRECT ERRORS(doc, lda_model) ▷ doc is the document with erroneous tokens

▷ lda_model is the model built from above

tokens = PARSE DOCUMENT(doc)
errors = FILTER ERRORS(tokens)
clean_tokens = tokens - errors
bow_repr = CONSTRUCT BOW(clean_tokens)
$dist$ = lda_model[bow_repr] ▷ Probability of the document occurring under each topic
for each error ∈ errors **do**
possible_decodings = DECODE(error)
$best = 0$
for each token ∈ possible_decodings **do**
$rank = \sum_{topic_i \in dist} dist(topic_i) \times P(token|topic_i)$
if $rank < best$ **then**
$best = rank$
$alternative = token$
end if
end for
Replace error with $alternative$
end for
end procedure

reduces significantly to an extent where there are hardly a few words left error-free in a document of about 500+ words. Given such a test document, the LDA model will find it hard to associate the document with any topic and it will result in a more even distribution of topics for the document thereby reducing its accuracy. To overcome these issues, we include the following changes.

- **Erasures with Single Replacements** Given the fact that the errors we are dealing with are erasures over a valid binary stream, not all replacements of the missing bits would result in valid English words. For example, an erroneous binary stream with k erasures will effectively have 2^k possible bit replacements leading to that many English words. But in reality, most of them are not valid words and in many cases, we end up with a single valid alternative[6]. This can be directly included into the model to build the topic distribution discarding the fact the token originally had erasures.

- **Corrections from a Global Word Distribution Model** Another possible improvement is to use the correction based on a global word distribution model which is built over the training documents without any topic classification. Although this model has lesser accuracy compared to the topic model, it still provides a means to build the topic distribution for the document using the LDA model. The distributions obtained via using these suggestions have proven to be useful in achieving higher accuracies. This change might skew the overall topic distribution towards a global trend but applying it on the smaller scale of a document reduces this effect and is able to provide better results.

[6]This also forms the basis for the optimisation proposed later in the research which was used to efficiently prune possible decodings for a given token with erasures

2.7 Experimental Results

These models were tested out on the Newsgroup dataset described earlier. As part of the experiments, we used a global distribution model as the baseline for comparisons. This is similar to the baseline used by [20] and provides valuable insights into the functioning of the model. The Newsgroup data is by itself split into twenty different topics with documents split evenly across all of them, both in the test as well as the training dataset. Although LDA provides for using varying number of topics, since we know the abstract topics the data is split into, the experiments were run using twenty topics only.

To ensure that the LDA model was able to distinguish topics but still find connections between the related ones, we ran the documents from three different topics through the model and plotted the average probability of all the documents across each topic from each of the original topics. The results show direct correlation between the original topics and those generated by the LDA model. In Fig. 2.1 where the documents from the topic **"alt.atheism"** have been plotted, we can see that topics 1 and 3 are very active.

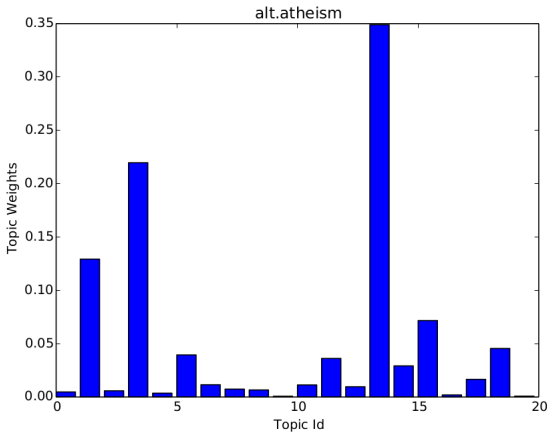


Figure 2.1: Topic Distribution for *alt.atheism* in LDA

Topic 13 is also very active but it is active for most folders (probably because it accounts for some subset of common words) and so, is less interesting. In Figures 2.2 and 2.3 however, two contrasting topics on **computers** have been plotted and we can see that they are less active around 1 and 3 and rather, their activity shifts around the topics 10to15.

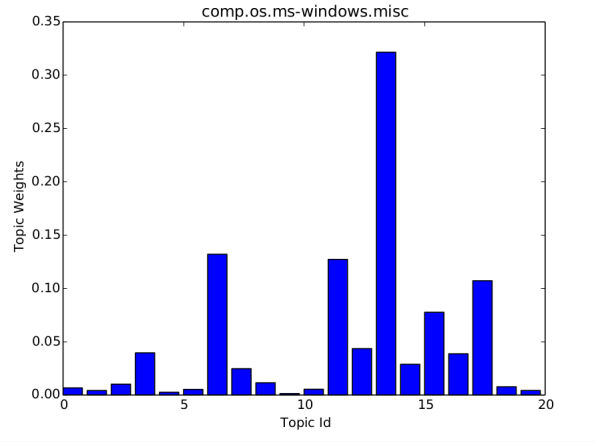


Figure 2.2: Topic Distribution for *comp.os.ms-windows.misc* in LDA

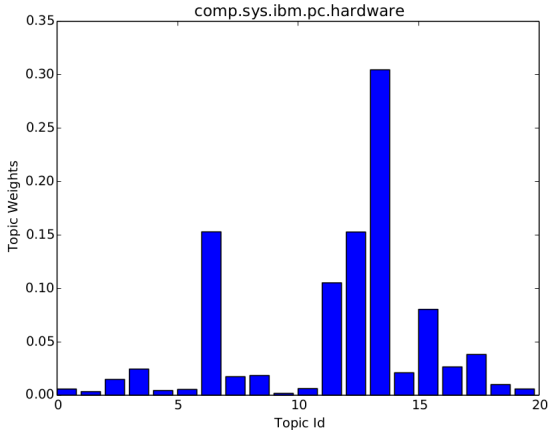


Figure 2.3: Topic Distribution for *comp.sys.ibm.pc.hardware* in LDA

It also shows that since the two topics are related, they have strong coherence among themselves. This shows how the LDA model is able to classify topics. Similarly, plotting different sets of topics from the original dataset shows coherence among related topics and discord among non-related topics. It is also to be noted that these clear demarcations arise only after removing most of the stop words and other shorted words (shorted than 4 characters) As expected, such words add little contextual information and in fact, add noise to the system thereby negatively impacting the functioning of the LDA model.

The primary metric we used to compare performance of the LDA model was the accuracy of the model. We define **Accuracy** as

$$\text{Accuracy} = \frac{\text{number of accurate corrections}}{\text{total number of erroneous tokens}}$$

While calculating the errors, we only consider words which have **more than one possible valid English decoding** This is because for words with no more than one possible decoding, no model is needed to correct them and so, does not add any value

to the results[7].

The tests have been conducted on a set of 50 files randomly chosen across the test dataset topics with erasures induced into their Huffman coded streams. We have tested the models across a wide range of ER values from 0.01 to 0.50 Please note that although any data with an ER value of more than 0.25 is mostly corrupted beyond proper recovery, we have tested such ER values to ensure that the model is able to perform under heavy stress and also understand the dynamics of the model with higher ER values. We were able to see around 2% to 4% improvement in accuracy on average over the global word distribution model as shown in Figure 2.4.

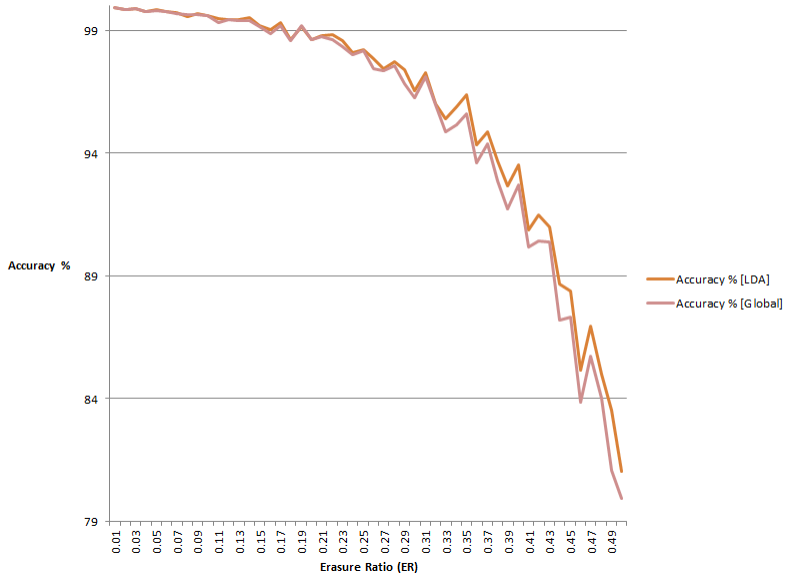


Figure 2.4: Comparison of Accuracy for LDA Topic-Model vs Global Model

We define D_{accuracy} as the difference between the accuracy of the LDA model from

[7]Please do note that such words are more common for lower ER values and are very useful in cases like 2.6.3

the global model i.e. $D_{\text{accuracy}} = accuracy_{lda} - accuracy_{global}$ We can see that as the ER value increases, this difference increases as well meaning that the LDA model becomes more efficient that the global model as ER increases. This can be explained by the fact that as ER increases, there are more erasures within the Huffman code of a single word which in turn increases the number of possible decodings. With more choices, the ability of the model to choose the right alternative is tested more and we can see that the LDA model is better suited in performing this task. This is supported by the plot inf Figure 2.5

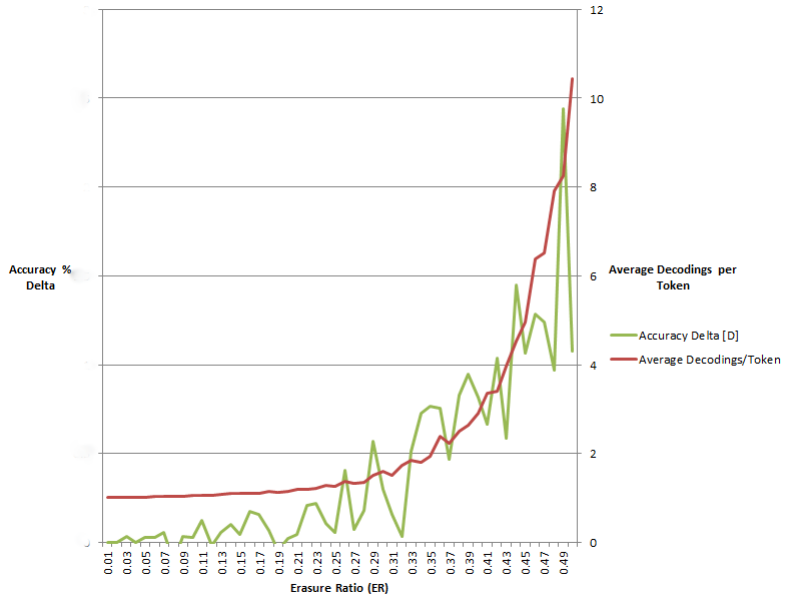


Figure 2.5: Accuracy Delta D vs Average Possible Decodings per Token

With regards to the running time or other resource constraints placed by using the LDA model, we have pre-loaded the LDA model into a serialized object which can be loaded onto memory at will and the reminder of the process does not add any significant resource requirement. All the tokens in the document are parsed twice, once to build the

topic distribution and finally to correct the errors and so, this model can be seamlessly used in applications to correct errors in a document.

2.8 Conclusion

The results have been very promising and shows that accounting for the topics in the document does improve performance of the model. The LDA is model is successful in classifying documents in an automated way and this has been used to correct errors in the documents accurately. Although an improvement of 3% to 4% does not seem significant, considering the fact that the original models themselves are able to achieve close to 85% accuracy at the bare minimum (refer to Figure 2.4), any improvement is significant. We should also note the fact that the model does not assume any knowledge of the characteristics of the text and can be ported to almost any domain with minimal modifications.

3. N-GRAM GRAPH MODEL

By definition, an **N-gram** is a contiguous sequence of **N** items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. When we build the distribution information of all such N-grams across the training data, we end up with a model that is able to tell us how frequent is a particular N-gram. Using this data, we can run a similar algorithm as with the topic model where we can form different N-grams using each of the possible corrections of the erroneous token with its neighboring text and depending on the most probable N-gram that can be formed, we can decide on the correction to use. Although this seems rather simple, using N-gram model has its own share of issues that we ran into. So, we had to switch to a much broader N-gram Graph model which has been explained later. Before going into all of them, we will look into how N-grams are currently being used for various language-related problems.

3.1 Standard N-gram Models

3.1.1 Error Correction using N-gram Models

We will now look into how a standard N-gram model can be used for error correction and the issues we faced using such a model. One of the most important decision that needs to be taken before building the model is the value of N. Higher the value of N, more complex the model will get and could possible be able to perform better. But blindly increasing N might not translate into better performance for the following reasons.

- While building an N-gram model with such an increased range, we will eventually include a lot of stop words or irrelevant words into the N-grams. This way,
- Increasing N will result in an exponential increase in the number of unique N-grams

to deal with. Besides requiring additional resources, this might reduce the relevance of important tokens in the text as there is simply a flooding of possible N-grams for every correction.

At the same time, a smaller value of N will not add enough information in the model which would mean lesser effectiveness of the model. So, arriving at the sweet spot for the value of N is one of the most important tasks. We tried building such an N-gram model with various values of N to see which works best but ran into a few issues.

- **Source Data** There are multiple sources of N-gram frequency data for English available online. Given that English text does not differ much in terms of the possible N-grams or their distributions, we could use them directly. I tried using the Google N-Gram Data (`http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`) but due to its sheer volume, I skipped it. As an alternative, I tried using the sample data provided by Linguistic Data Consortium (`https://www.ldc.upenn.edu/data-management/providing`) which provides a 5M N-gram sample. Both these sources had two issues, one being the presence of numerous stop words in a majority of N-grams which is understandable since they have been built over web data which is bound to have such noise. The second problem is that one we opt to work with trigrams or longer N-grams, we need to analyse a much larger dataset which might not be scalable.

- **Building Custom N-gram data** Given the issues mentioned above to leverage N-gram data available online, we built our own N-gram model using the training data we were using. In this case, it would be the same Newsgroup data. Building the N-gram model from the training documents will follow the same process as described later at 5 After the model is built, we can use this model in place of

the topic distribution for the document and the topic-specific word distribution as described in 3 A more detailed explanation is given in Algorithm 4

3.1.2 The Algorithm Explained

Algorithm 4 Error Correction using N-gram

```
procedure N-GRAM CORRECTION(doc, ngram_model, N)  ▷ doc is the document to be corrected
                                                  ▷ ngram_model is the model built from above
    tokens = PARSE DOCUMENT(doc)
    errors = FILTER ERRORS(tokens)
    clean_tokens = tokens - errors
    for each token ∈ errors do
        possible_decodings = DECODE(error)
        best = 0
        for each token ∈ possible_decodings do
            ngram = CONSTRUCT N-GRAM(token, N)        ▷ Construct an N-gram using the error's surroundings
            if P[ngram] < best then
                best = P[ngram]
                alternative = token
            end if
        end for
        Replace error with alternative
    end for
end procedure
```

But using this model of N-grams failed to improve the results as compared to the baseline of the global word distribution model. Even increasing the values of N in hopes of increasing context for each N-gram did not result in any improvement. So, we had to move to an alternative approach.

3.1.3 An Illustration of Why Standard N-grams' Inefficiency

One way to look into the possible reasons for the failure of the N-gram model is that when we construct an N-gram with each possible alternative replacing an error, not many samples exist from the training data for each such N-gram generated. This translates into failure during the ranking phase since none of the alternatives have a clear advantage and so, we end up choosing an alternative at random resulting in the reduced accuracy. For example, if we are fixing "super **fast** computers" using 2-grams where **fast** is the word to be fixed, the 2-gram training samples built from the data might probably have *"fast computers"* and *"super fast"* once but "super cute" a couple of times. In such a scenario, we are bound to choose "cute" as the replacement which is incorrect and there is no way to distinguish **fast** to be clear winner. The problem here is that the counts of very specific 2-grams are much lower which makes the decision-making tough and error-prone.

3.2 N-gram Graphs - an Enhanced Version of the N-gram Model

N-gram Graph provides an opportunity to avoid the problem of exponential increase in the possible N-grams when N is increased while at the same time, providing means to leverage the information available through such an increased window. The trick lies in not generating unique N-grams for each co-occurrence. Rather, the model accounts for the cumulative co-occurrences of two words to form an N-gram like construct. This way, there is no explosion in the state space of the N-gram model but still, it is able to account for a wider context, thereby being able to attain possibly better results. We need not worry about increasing the context range based on the resource constraints it will place. A formal definition of the graph model follows.

3.2.1 N-gram Graph Definition

The graph G consists of a set of vertices V each of which is the unique word in the vocabulary of the model. The graph also has a set of undirected edges, E each of which has an associated weight W connecting any of the two vertices in V. L_{win} is the size of the window within which all words are assumed to be related to each other. So, for any two vertices $v_1, v_2 \in V$, if there is an edge $\in E$ in between them, then the weight associated with the edge, $W_{1,2} = W_{2,1}$ is given by the total number of occurrences of the two words represented by $v_1$andv_2 within a distance of L_{win} across any of the documents in the training set. More formally,

$$W_{1,2} = W_{2,1} = \sum_{d \in D} \begin{cases} 1, & \text{if abs}(p_1 - p_2) \leq L_{win}. \\ 0, & \text{otherwise} \end{cases} \forall p_1 \in probe(d, v_1), \forall p_2 \in probe(d, v_2)$$

where D is the entire set of training documents and $probe(d, v_1)$ returns the list of positions in d where the word v_1 occurs. An example of a N-gram graph is explained below to give a better picture of its functioning.

3.2.2 Example of N-gram Graph

Figure 3.1 shows a sample N-gram graph constructed with a window size L_{win} of two for the stream of words **ABCACBADE** where each unique alphabet represents a unique word. In the actual implementation, words are stemmed to their appropriate roots so that the N-gram state space is reduced and more associations are allowed to form.

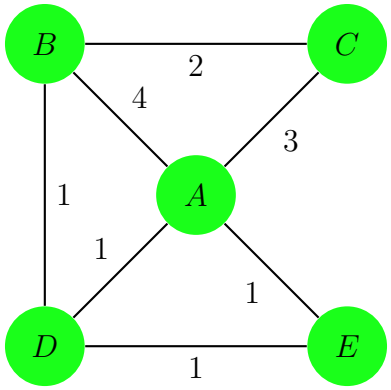


Figure 3.1: N-gram Graph for **ABCACBADE** and $L_{win} = 2$

Similarly, we can build the graph covering the entire dataset and the resulting graph denotes the associations among the various tokens. In the same example we used for the topic model (2.4), if we build the N-gram graph over this text after applying stemmers as explained below, we can see that we build strong associations between the words "computer" and "process" and "program" which will help increase the edge weights among those vertices resulting in a higher probability in the model below.

3.2.3 N-gram Graph Implementation

As mentioned earlier, the N-gram Graph model has two steps similar to the LDA model, the first being the construction of the model from the training data and the second being the correction of errors based on information from the model. A detailed algorithm explaining the two steps is given below. One can notice the strong similarity between the methods mentioned here and those under the LDA model. This way, we have the probability associated with each alternative as follows. As in the topic model, we choose the alternative which has the highest probability.

$P(alt) = \sum w \in DW_{w,alt} \forall w$within distance $L_{win}$from alt

Algorithm 5 N-gram Graph Construction

procedure N-GRAM GRAPH(documents, L_{win}) ▷ documents is the list of training documents
    ngram_graph = ϕ ▷ represents the N-gram Graph model
    for each doc ∈ documents **do**
        tokens = PARSE DOCUMENT(doc)
        for each token ∈ tokens **do**
            for each neighbor ∈ tokens **do**
                if abs(position[neighbor] - position[token]) $\leq$ L_{win} **then** ngram_graph[neighbor][token] += 1
                end if
            end for
        end for
    end for
    return ngram_graph
end procedure

Algorithm 6 Error Correction using N-gram Graph

procedure N-GRAM CORRECTION(doc, ngram_graph, L_{win}) ▷ doc is the document to be corrected
▷ ngram_graph is the model built from above
    tokens = PARSE DOCUMENT(doc)
    for each token ∈ tokens **do**
        if token == error **then**
            possible_decodings = DECODE(token)
            $best = 0$
            for each neighbors ∈ tokens **do**
                $rank = 0$
                if abs(position[neighbor] - position[token]) $\leq L_{win}$ **then** ▷ If the two tokens fall within the Window limit
                $rank+ = ngram\_graph[neighbor, token]$
                end if
                if $rank < best$ **then**
                $best = rank$
                $alternative = token$
                end if
            end for
            Replace token with $alternative$
        end if
    end for
end procedure

3.2.4 Stemming to Improve N-gram Graph Model Cohesion

Even though the N-gram graph does away with most resource constraints associated with a regular N-gram model, the number of unique words is still large. Apart from the resource constraint, having a larger vocabulary also reduces the accuracy of the model in many ways. For example, the same word could be expressed in different forms across the corpora - singular/plural mutations or different tenses of the verb etc. This would mean much stronger and more cohesive relationships between two words, which are actually related closely, would fail to develop since each might be in a different forms at their co-occurrences and so, their bond strength diffuses across these different forms in the N-gram graph.

To avoid this, we have used the Stemmed version of all the words in the vocabulary. **Stemming** essentially means reducing the words to their base or root forms. For example, all forms of a noun (singular/plural) would have the same stemmed root and the same applies to verbs and other semantic parts of speech. This effectively reduces the number of unique words or the nodes in the graph to a much more manageable 21,500 as compared to the 36,500 unique words in their raw form. Maintaining a graph with these many nodes is still challenging but is more probable now.

3.3 Experimental Results

- Initial tests based on the N-gram graph model revealed that the accuracy in most cases is comparable to the global word distribution model and some times even worse. All these experiments were run with L_{win} set to 5. This was pretty disappointing since pentagrams were hard to build in the plain N-gram version but we expected it to perform well. Setting the window size to 5 should have probably achieved the same effects but for some reason, it did not make any difference.

- Naturally, the next option was to try out different values for L_{win} Given that only the number of edges in the graph are bound to increase as L_{win} increases, we were able to try values ranging from 1 to 50, with the version with $L_{win} = 50$ taking up 60 MB of space on the memory when built fully. Again, the results did not match the expectation that a higher value of L_{win} would result in better performance. The accuracies were again comparable to the global distribution model only and in fact, higher L_{win} values resulted in even further deterioration. This can be seen in Figure 3.2 The graph also plots the results from Hybrid Model results which has been discussed in detail in a later section (Section 3.4).

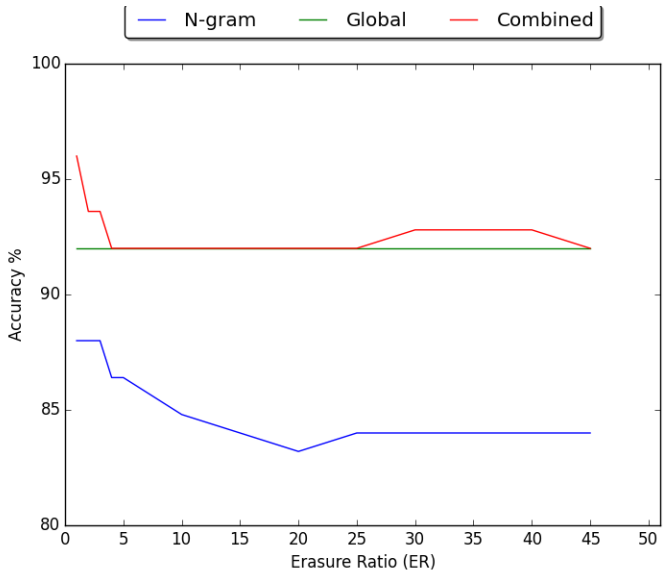


Figure 3.2: Accuracy vs. N-gram Window Size

One of the possible reasons for this poor accuracy of the N-gram graph model could be the purity of the training data. We are currently using the Newsgroup data to build the

N-gram graphs. We are using the same data for both the models so that it will allow us to compare the results directly. But the potential problem with this data is the significant presence of noise. Since this data has been extracted from the web, especially from newsgroup articles, it is almost impossible to completely remove all unwanted characters and headers.

Although such noise would affect the performance of all the models, its effect is more pronounced for the N-gram graph model since it takes into account the spatial neighbourhood of tokens more seriously which invariably means that such noise would limit the efficiency of the graph constructed. For the other models, the inferences are made at a global level which mitigates this problem. So, what was in fact perceived as a possible strong point of the N-gram graph model turns out to be its biggest drawback on such noisy data. Sample experiments with a much cleaner but smaller dataset showed that its performance does improve as the training data quality increases.

3.4 The Hybrid Model Explained

3.4.1 Why Combine Results?

With the above said restrictions, we wanted to check if the union of the accurate corrections from the LDA model and the N-gram graph model would provide better results. This way, a particular error not corrected by the LDA model could be corrected by the N-gram graph model or vice versa. This could be possible, even though the accuracy of the N-gram graph model is not matching the LDA, if the errors they are able to fix differ i.e. the two models supplement each other resulting in an enhanced result. For this to work, we require that

- The unified model, called the **Combined Model** from hereon, outperforms the LDA model - without this, there is no point in building the hybrid model. If the errors the two models correct are closely coupled and the uncorrected ones are

mostly same, this would result in the combined model not being any better than the LDA model.

- The second problem is that even if the union is found to be better, we still need to find a way to mathematically decide the best model for each case and choose the right one. Because choosing the wrong model could actually deteriorate the results further from the current LDA benchmarks.

3.4.2 Combining the Two Models

Given that both the LDA model and the N-gram graph model provide a ranking based on the probability of each alternative replacing the error, we decided to use the normalized ranking from each of the model and use a linear combination of the values to arrive at a new set of rankings. Our assumption is that this linear combination would be able to leverage the best results from both the models. So, we ended up using a model as follows:

$$P(alt) = \alpha P_{N-gram}(alt) + (1 - \alpha) P_{LDA}(alt)$$

Of course, the issue with this model is to be able to detect the weightage given to each of the models represented by α - the weight given to the results from the LDA model. Provided that we have sufficient data to train on, we could possibly run this model over different training sets and determine the best possible weight that gives the most accuracy.

3.4.3 Experimental Results

We ran the experiments on this hybrid model using training data generated with varying Erasure Ratios. The first part of the experiment was to identify the ideal value for α For this, we ran the tests on Erasure Ratios between 0.01 to 0.50, each ratio with

a set of 50 files containing erasures strewn over 3000 tokens. The best value for α was identified as the one which had the highest average accuracy over all Erasure Ratios.

Weights ranging from 0 (completely N-gram) to 1 (completely Topic Model) were tested in increments of 0.025 and it was identified that an α value of **0.475** would be ideal resulting in an average accuracy of 94.06%. The graph in Figure 3.3 shows the variation in the average accuracy across all the weights. It can be seen that a predominant change in the accuracy happens only towards the right end of the graph where the N-gram model is given the most weightage and so, the average accuracy dips as expected.

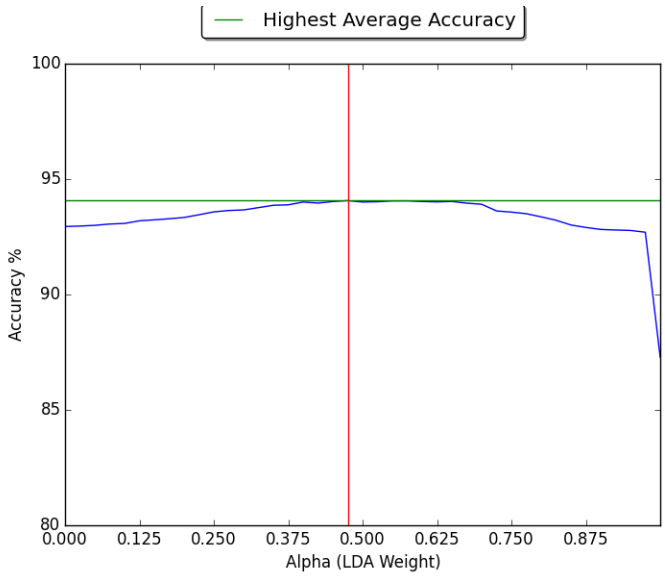


Figure 3.3: Average Accuracy vs. α

Having figured out the ideal value to assign for α, we used this value to combine results from the LDA model and the N-gram graph model and the results have been plotted below. As mentioned earlier, although this primitive aggregation model is not able to match the results from the combined model, it is still able to better the results

from the LDA model by atleast 2% Given the fact that this aggregation model is pretty naive and also the N-gram model could perform better if provided with cleaner data, such an hybrid model could well have another 2% to 3% improvement in accuracy. Compared to the global word distribution model, this model provides more than 4% more accuracy even including Erasure Ratios of upto 0.5. This can be seen from Table 3.1.

Table 3.1: Average Accuracy of all Error Correction Models

Global	89.86%
LDA	91.61%
N-gram	86.87%
Combined	95.80%
Hybrid	93.43%

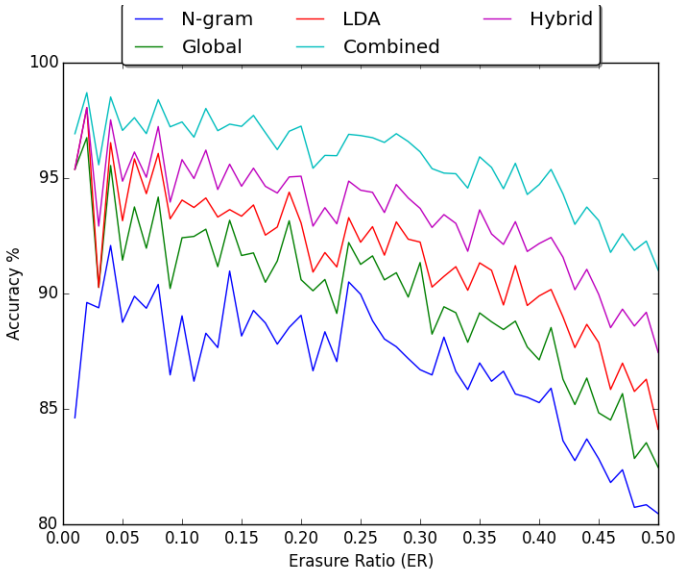


Figure 3.4: Accuracy vs. Erasure Ratio

3.5 Conclusion

The results were not as expected for the N-gram graph model. But we were still able to leverage useful information from the model to enhance the Topic-model. The fact that the N-gram model works better only with smaller window sizes indicates that the Topic model is using a broader contextual information and the N-gram model is supplementing it with more localised information derived from its smaller window size. An improvement of around 2% in accuracy over the Topic model is significant indeed considering the fact there is very little extra work going into the building the N-gram model itself.

4. BINARY STREAMS AND OPTIMIZED HUFFMAN DECODING

Both the models described earlier depend on efficiently decoding the alternatives for an erroneous token. The research also includes some improvements on this part so that the models can seamlessly integrate into other applications without placing any large resource constraints. So, this warrants a separate section. We will first look into a few interesting properties of the data we are dealing with. Then, we will see how we can leverage those properties to our advantage.

4.1 Huffman Coding

As mentioned earlier, Huffman Coding is a variable length coding scheme. We are assuming the input to be consisting of English words which will be encoded via Huffman coding. Since erasures in the binary stream can be replaced by all possibilities and since Huffman coding is a variable length coding, if the word boundaries are unknown, the number of possible decodings could explode and no meaningful assertion can be made for the corrections. Given that we are trying to correct entire documents which might consist of upto 500 words at times, erasures in a few words could lead to exponential number of possible decodings for the entire document.

To avoid this scenario, it is assumed that the word boundaries are known in the binary stream and no erasures are induced in those regions. This restricts the effects of erasures only to the corresponding words and does not affect the rest of the documents. Without this assumption, the entire decoding process would become much more complicated since an erasure in some token at the beginning of the document could affect the way the entire document is decoded and thereby lead to increased resource constraints and reduced accuracy of all the correction models in general.

We should also note that not all of the replacements of an erasure lead to valid

decodings. Given the variable length nature of Huffman codes, it is possible that some of the replacements of the erasure bits could lead to invalid decodings within the same token. This inherently reduces the number of possibilities thereby leading to better results from the correction models. This is more prevalent in smaller words which are induced with more erasures as many of the replacements would lead to invalid decodings as seen in an example in Section 4.2.1.

4.2 Prefix-based Elimination Technique Explained

We also use an English dictionary that was built using all the words in the training and the test data. Given the fact that both the models mentioned earlier do not have means to rank an alternative that was never seen before - both models would assign such an alternative a probability of occurrence of zero - we have taken the liberty of building a dictionary which will contain all the valid words. This makes the process of verifying the validity of a word much simpler.

As an added advantage, we use this information to effect a prefix-based elimination of the alternatives as they are generated. When an erasure is found, the algorithm tries to replace it with both a zero bit and a one bit. But as we saw earlier, not all of them lead to valid Huffman decodings. But we can effect an even more stricter filtering process if we take into account all the valid English words that are allowed.

We store the entire dictionary in a Trie data structure which allows prefix lookups. Then, while incrementally building the alternatives for a given word, if at any particular point, the prefix of the word built so far does not match any of the valid words, we know that any further processing of the stream with its current erasure replacements is not going to lead to any valid English word. This allows us to prune a large chunk of possibilities even before they are completely decoded.

More formally, while working with a partially decoded bit stream which has the bits

corrected for the first i erasures of the k erasures in all, if the partially decoded word is not a prefix of any valid English word stored in a pre-computed dictionary of valid words, further replacements of the remaining $k - i$ positions can be ignored thus saving 2^{k-i} operations. To implement this optimization, a Trie data structure that supports prefix operations, is used to hold the dictionary of English words. The prefix checks in such a structure can be performed in $O(L)$ where L is the length of the prefix. This optimization results in significant running time improvements.

4.2.1 Example

Figure 4.1 explains how this works with a very simple trie. Let the trie have the following words inserted into it - *a, ab, b, ba* The Huffman code for the characters are given by $\{\text{a} \rightarrow 0 \ \ \text{b} \rightarrow 10\}$ If the binary stream contains a token in the form **???** where **?** represents an erasure or a missing bit, we have $k = 3$ erasures leading to $2^3 = 8$ possible alternatives which are

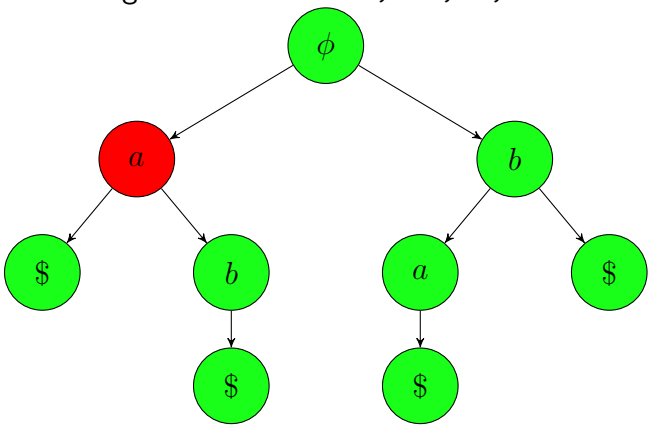
1. 000 $\rightarrow$ *aaa*
2. 001 $\rightarrow$ *INVALID*
3. 010 $\rightarrow$ *ab*
4. 011 $\rightarrow$ *INVALID*
5. 100 $\rightarrow$ *ba*
6. 101 $\rightarrow$ *INVALID*
7. 110 $\rightarrow$ *INVALID*
8. 111 $\rightarrow$ *INVALID*

Algorithm 7 Optimized Huffman Decoder

```
    procedure DECODE STREAM(stream, index, prefix, node, dict)
                                          ▷ stream is the Huffman coded binary stream
 3:                                       ▷ index represents the position in the stream
                                ▷ prefix represents the English text decoded so far from stream
                        ▷ node represents the node inside the English Trie that has been reached
 6:                                       ▷ dict represents the dictionary of all English words
        if node = φ then                  ▷ If the Trie node is invalid, no decoding possible
            return φ
 9:     end if
        if index = stream → length then                    ▷ End of binary stream reached
            if node → leaf = True then
12:             word = prefix + node → value
                if IS VALID(word, dict) = True then
                    return word
15:             end if
            end if
            return φ                              ▷ Return NULL if any of the conditions fail
18:     end if
        if node → leaf = True then
            prefix = prefix + node → value
21:         if IS PREFIX VALID(prefix, dict) = True then
                return DECODE STREAM(stream, index, prefix, node → root)
            end if
24:         return φ
        end if
        results = {}
27:     if stream → index = ERASURE then
            results = results ∪ DECODE STREAM(stream, index+1, prefix, node → chil-
    dren[0])
            results = results ∪ DECODE STREAM(stream, index+1, prefix, node → chil-
    dren[1])
30:     else if
             thenresults = results ∪
             DECODE STREAM(stream, index+1, prefix, node → children[stream → idx])
        end if
33:     return results
    end procedure
```

Using the prefix-elimination, we can see that one of the INVALID states 2 will never be reached as its prefix **00** is itself invalid. This has been shown in Figure 4.1 where the node in **RED** represents the state from which we can eliminate the INVALID state **001** even before decoding it completely. This is a rather simple example but similarly, the prefix-elimination optimisation will help prune out a majority of replacements even before they are traversed thereby leading to exponential speed-ups.

Figure 4.1: Trie for `a, ab, b, ba`



4.2.2 Improving Performance using Enhanced Trie Representations

As we can see, the algorithm essentially depends on the trie being able to retrieve query results efficiently, both for full word queries as well as prefix searches. But using a traditional implementation of Trie caused a few problems due to the scale of the dataset.

- It was using a lot of memory which impacted performance of the overall system.
- It was also not the best representation of the dictionary that was possible. As explained below, an alternate representation saves both memory as well as improves performance.

Given these issues, we switched to using **Directed Acyclic Word Graph** or **DAWG** [1] which is a finite state automaton that provides the same functionality as any other trie but does it more efficiently. Switching to this implementation improved the performance of the decoder by a factor of 10. We tested the improved decoder over a set of 3,000 tokens filled with erasures at an ER of 0.25 thus having more than 30,000 valid decodings. The decoder supported by a plain Trie took 4.2 seconds to complete the processing whereas the DAWG-based decoder took less than 0.5 seconds to complete the same task. The DAWG-based dictionary was also better compressed as it took less than 20 MB of memory when written to disk as compared to the 27 MB taken by the regular trie.

4.2.3 Experimental Results

To test the effectiveness of this approach, we ran the decoder using the optimisation over a set of 3,000 words encoded in Huffman code with erasure induced at an ER value ranging from 0.01 to 0.50. This leads to a significantly large number of erasure-filled words for higher ER values, each of which having numerous possible replacements based on the erasure replacements alone.

Running these different files against a decoder which uses the prefix-based elimination and one that does not (standard Huffman decoder), we can clearly see the positive impact of using the optimisation in Figure 4.2. In fact, for ER values greater than 0.28, the regular decoder was unable to complete the process within an hour and so, all experiments for the standard decoder with ER values greater than 0.28 were skipped from testing. For the same reason, the language models described in the earlier chapters could not have run for higher ER values without this optimisation. As expected, the prefix-elimination scheme leads to tremendous speed-ups.

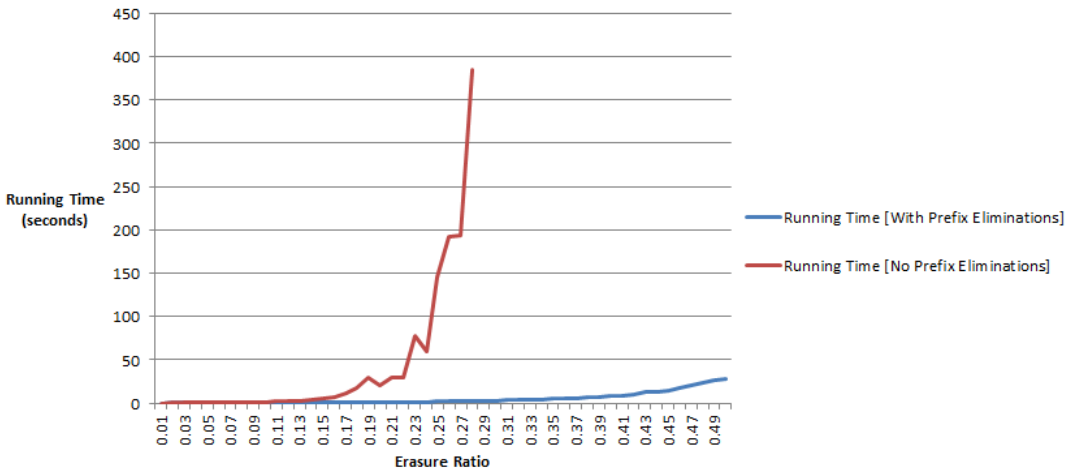


Figure 4.2: Running Time Comparisons

4.3 Conclusion

Although the enhancements is pretty straight forward and nicely fits into the Huffman Trie model itself, it does go a long way in helping to test larger datasets using lesser resources. The same idea could be extended to cases where the word boundaries are unknown in which case we need to use some form of memoization to cache previous results but still, the core idea will remain the same.

5. SUMMARY

Errors are pretty common in the domain of Data Storage and so, error correction is an active field of research. Many such ideas involve some assumption about the data which is used to build better results. For example, [20] have used the error model of the OCR software as the basis for their performance metrics. Our aim was to achieve an improved accuracy in such error correction using little or no assumption and we have made significant progress.

- We have achieved this goal to some extent, having displayed upto 5% improvement as compared to a global word distribution model with no major assumption on the characteristics of the data.
- We have also developed a model to efficiently decode Huffman codes with errors at random positions using a modified version of the Trie data structure. This optimisation can be used in any scenario wherein erroneous Huffman codes are to be decoded.
- We have seen that language models do not necessarily behave as expected as in the N-gram model. However, extensive experiments have proven that even such seemingly inefficient models can help complement a much better model resulting in improved accuracy. Given the characteristics of our model's output as probabilities, it also lends well to be combined with other language models.

However, there is definitely more room for improvement and we believe the models developed here can be expanded much more to achieve better results. Some of the possible improvements or future steps have been listed below.

5.1 Machine Learning Approach for Hybrid Model

The hybrid model is still, on average, 2% below in accuracy as compared to the best possible result combining the N-gram graph model and the LDA-model. One possible improvement would be to use a more complex underlying aggregator than the simple linear aggregator used for the hybrid model currently. One possible option is to use a learning model which when fed sufficient data could produce better results approaching closer to the best accuracy possible. Given the fact that generating errors is much easier with the available training data, such a model could be trained extensively before being tested.

Learning to Rank [18] is a supervised learning approach which fits well into our requirements. This approach is able to learn from trained rankings based on features provided and will be able to predict future rankings. In our case, the features are the weights given by the LDA-model as well as the N-gram graph model and the ranking involves binary classification - either right or wrong. *RankSVM*[10] is an application of the Learning to Rank approach using an underlying Support Vector Machine. This has been proved to work well in web-ranking and we tried using this model for our purpose. Initial experiments with a reduced set of training data has shown that such a trained model can achieve accuracy within the range of the global distribution model. We strongly believe that larger training sets and a well-tuned learner could perform even better, possibly overtaking the linear aggregator that has been used. There are also other available approaches towards classification like "Ensemble Learning" [15] using multiple decision trees which could be used.

5.2 Removing the Word Boundary Assumption

One of the key assumptions of the models used here is the knowledge of the word boundaries in an erasure-filled bit stream. The primary purpose of this restriction was

to reduce the complexity of the model. As mentioned earlier, if knowledge of the word boundary is not available, the variable length nature of Huffman Coding would lead to an explosion in the possible alternatives for a bit stream. We should note that such changes would mostly affect the performance of the global distribution model to the same extent as our models, if not worse. So, it does not offer any significant advantage for our models over the baseline model.

If this knowledge is not available, we could run a dynamic programming algorithm similar to the Viterbi algorithm for Hidden Markov Models, wherein the hidden states would be the alternatives for erroneous tokens (individual tokens are identified in such a sequence when a word delimiter character has been decoded). Although this might not scale for a really large sequence of words with a lot of erasures, it does offer improvement over the brute-force version of trying every possibility for K erasures, where K might be pretty large as it corresponds to the entire document.

5.3 Improving the N-gram Graph Model

The current version of the N-gram graph that has been constructed is pretty naive. This is one of the reasons for it being outdone by even the global distribution model in many cases. Although its combination with the LDA-model is able to give better results, there is still room for improvement of the model in both its construction as well as its usage. For example, it does not account for the distance between the tokens within the same window which might be an important signal that could be leveraged. Similarly, while applying the distribution from the graph model to the data, distance between the tokens could be used. Given that it has been separately used in effective "Similarity Detection in Text Summarisation"[6], it definitely holds potential to provide better results.

5.4 Parts-of-speech Tagging

Given that we are working with regular English tokens in the data, using Parts-of-Speech tags to determine better alternatives for an erroneous token is a good option. Specifically, the training data gives us insights into the various possible transformations of the POS (Parts-of-Speech) tags as we move across the sentences. So, each alternative would lend to a different transformation chain across the sentence and we could choose the one with the highest probability based on the data gathered from the training data. Given that the LDA-model and the N-gram model also work with probabilities, the POS tag based model could as well be included into the Hybrid model.

Although this does assume the presence of a robust tagger which is able to deduce tags of specific words using minimal contextual information from the sentence (since surrounding words could also be erroneous), it definitely lends well to include the results into the Hybrid model. In the worst case that the effects of such a model are mostly detrimental, both the linear combination as well as the ML-based versions of the Hybrid model could simply neglect its results after being trained.

REFERENCES

[1] Dawg-Python. `https://pypi.python.org/pypi/DAWG-Python`. Accessed: 2014-20-11.

[2] *SIGIR '10: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, 2010. ACM. 606100.

[3] Steven Bird. Nltk: The Natural Language Toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] Stanley F Chen. Performance Prediction for Exponential Language Models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 450–458. Association for Computational Linguistics, 2009.

[6] George Giannakopoulos and Vangelis Karkaletsis. Summarization System Evaluation Variations based on N-gram Graphs. *ACM Transactions on Speech and Language Processing Vol. 5(3)*, pages 1–39, 2008.

[7] Taher H Haveliwala. Topic-sensitive Pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, pages 517–526. ACM, 2002.

[8] James Huang, Stephanie Rogers, and Eunkwang Joo. Improving Restaurants by Extracting Subtopics from Yelp Reviews. *Yelp Dataset Challenge*, 2014.

[9] David A Huffman et al. A Method for the Construction of Minimum Redundancy Codes. *Proc. IRE*, 40(9):1098–1101, 1952.

[10] Thorsten Joachims. Optimizing Search Engines using Clickthrough Data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 133–142. ACM, 2002.

[11] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: Open Source Scientific Tools for Python. *http://www. scipy. org/*, 2001.

[12] Grzegorz Kondrak. N-gram Similarity and Distance. In *String Processing and Information Retrieval*, pages 115–126. Springer, 2005.

[13] Yehuda Koren. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434. ACM, 2008.

[14] Jack Linshi. Personalizing Yelp Star Ratings: A Semantic Topic Modeling Approach. *Yelp Dataset Challenge*, 2013.

[15] Richard Maclin and David Opitz. Popular Ensemble Methods: An Empirical Study. *arXiv preprint arXiv:1106.0257*, 2011.

[16] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The Pagerank Citation Ranking: Bringing Order to the Web. *Stanford InfoLab*, 1999.

[17] Radim Rehurek, Petr Sojka, et al. Software Framework for Topic Modelling with Large Corpora. *University of Malta*, 2010.

[18] Andrew Trotman. Learning to Rank. *Information Retrieval*, 8(3):359–381, 2005.

[19] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: Finding Topic-sensitive Influential Twitterers. In *Proceedings of the Third ACM International Conference on Web search and Data Mining*, pages 261–270. ACM, 2010.

[20] Michael L Wick, Michael G Ross, and Erik G Learned-Miller. Context-sensitive Error Correction: Using Topic Models to Improve OCR. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 1168–1172. IEEE, 2007.

[21] Wikipedia. Computer — wikipedia, the free encyclopedia, 2015. [Online; accessed 10-April-2015].

[22] Wen-tau Yih, Geoffrey Zweig, and John C Platt. Polarity Inducing Latent Semantic Analysis. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1212–1222. Association for Computational Linguistics, 2012.

[23] Deniz Yuret. Word Sense Disambiguation by Substitution. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 207–213. Association for Computational Linguistics, 2007.

APPENDIX A

HUFFMAN CODES USED

Here, we have listed the Huffman Code that was used for encoding the English text in the test data to a binary stream. The codes were built using the dataset used for the remaining sections. As stated earlier, we have only accounted for lowercase English alphabets to keep the code lengths smaller and so, all the text is assumed to be of lowercase. Apart from these characters, we also included two special characters, namely ' (apostrophe) and - (hyphen) which frequently occurs with compound words. Please note that we have not accounted for any other punctuation symbols or word delimiters like spaces or dots since the assumption is to work within word boundaries.

Table A.1: List of Huffman Codes Used

ASCII Character	**Huffman Code**
'	11101100
-	111111001
a	1001
b	1111111
c	00010
d	10100
e	001
f	111000
g	011101
h	11110
i	0110
j	0001101101
k	11101101
l	10101
m	111010
n	0101
o	1000
p	011111
q	0001101110
r	0000
s	0100
t	1011
u	111110
v	0111101
w	011100
x	000110101
y	000111
z	0001100011

APPENDIX B

IMPLEMENTATION DETAILS

The entire system was implemented using Python. Python does have its own drawbacks, the most important of them being the fact that it is comparatively slower than other languages, especially C/C++. The reasons to have chosen Python as the language are:

- Python is extremely flexible. Given that its not a strongly typed language, it allows for moving around the modules and modifying on the fly based on the changing dependencies.
- Python also has an extremely useful set of libraries - many of which were critical in building the models efficiently. If not for these libraries, extra hours would have been needed to implement the functionalities.

Following are the important list of libraries that were used in the development.

- *scipy and numpy [11]* was used primarily for holding the N-gram graph model using an efficient sparse-matrix representation. It was also used for other miscellaneous works including plotting the graphs etc...
- *Natural Language Toolkit - NLTK [3]* was used for language processing in various stages - tokenization, stemming, stopword removal etc...
- *ldagensim - Latent Dirichlet Allocation [17]* was used to build the LDA models. It is a handy library to build LDA models, including modules to generate bag-of-words representation and necessities for the LDA model.

- *DAWG-Python [1]* library was used to build the English dictionary, as mentioned earlier. It provided immense speedups to the word look up process and given the complexity of the underlying DAWG implementation, this library was very helpful in avoiding them.

As we can see, the rich library was one of the major motivating factors for choosing Python and it was definitely the right decision as many of these essential libraries did not have any equivalents in other languages, even including Java.