Second order hidden markov model for typos correction

Ahmed Mazari Hafed Rhouma

December 12, 2016

Contents

List of Figures

Abstract

The goal of this project is to design a model to correct typos in texts without a dictionary.

1 First order hidden markov model

We instantiate the HMM with 26 states and 26 observations. Then, we train the HMM model and we run Viterbi algorithm.
As a result we get 7.41% of error for tag predictions and 92.59% of accuracy.
For this task a random classifier makes 96.15% errors (our classifier makes 7.41%). Since we have 26 states than the random classier has $1/26 = 3.85\%$ to get the appropriate letter and $25/26 = 96.15\%$ errors.
We conclude that our classifier drastically outperform the random classifier.

2 Second order hidden markov model

In order to train second order HMM, we need to operate a variable change. To do so, We change the format of the corpus by concatenating all 2 neighbors. For example $[('o','o'),('t','t'),('h','h'),('e','e'),('f','r')]$ becomes $[('ot','ot'),('th','th'),('he','he')]$. Then to instantiate the HMM, we need to provide a list of all possible combinations of 2 letters of the alphabet for the *stateList* and *observationList* parameters. By doing this, we guarantee that every possible observation in the *testSet* can find an entry in the transition matrix. In the worst case the value would be 0 because never encountered in the training step.
After we trained the hmm with the *testSet* and run the viterbi algorithm we got 8% of error for the prediction.

3 Hidden markov model for noisy insertion

We describe a way to extend our model to handle noisy insertion of characters. The following example illustrates this noisy insertion :
('t','t'), ('h','h'), ('e','e'), ('m','m'), ('E','l') such that E is an empty letter.
When dealing with noisy insertion, we add a state/observation E (empty letter). We would then have a new set of observations: all the alphabet letters + the empty letter.
for any particular insert state, we may have different transition probabilities for entering it for the first time vs. staying in the insert state [10]. The figure below illustrates how we handle noisy insertion of characters.

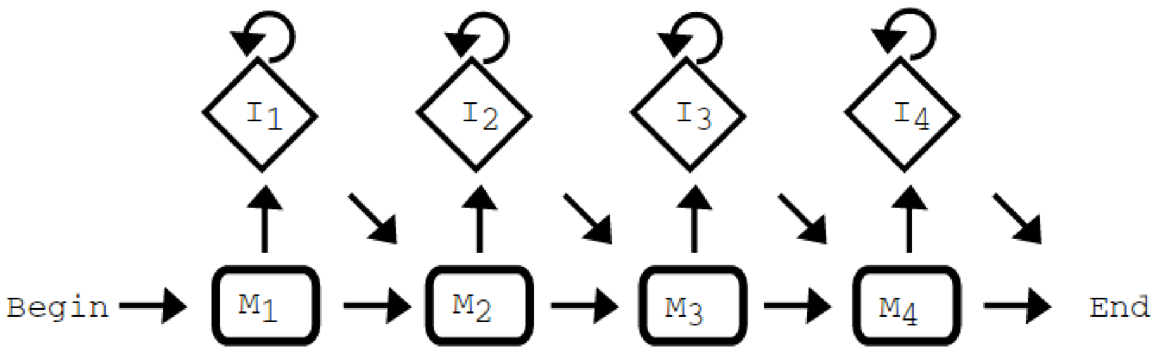


Figure 1: Insert states

Such that M represent the states (observations) and I are the insert states we add to handle noisy insertion.

4 Hidden Markov model for deletion

We describe a way to extend our model to handle deletion of characters. The following example illustrates the structure of the dataset with omitted characters :
('t','t') , ('h','h') , ('e','E'), ('m','m') such that E : is an empty letter.
When dealing with deletion, we add a state/observation E (empty letter). The delete states we add don't emit any symbols. We would then have all the initial states (all alphabet letters) + the empty letter as depicted in the figure [10] below. Such that M represent the states (observations) and D are the deletion states we add to handle omitted characters.

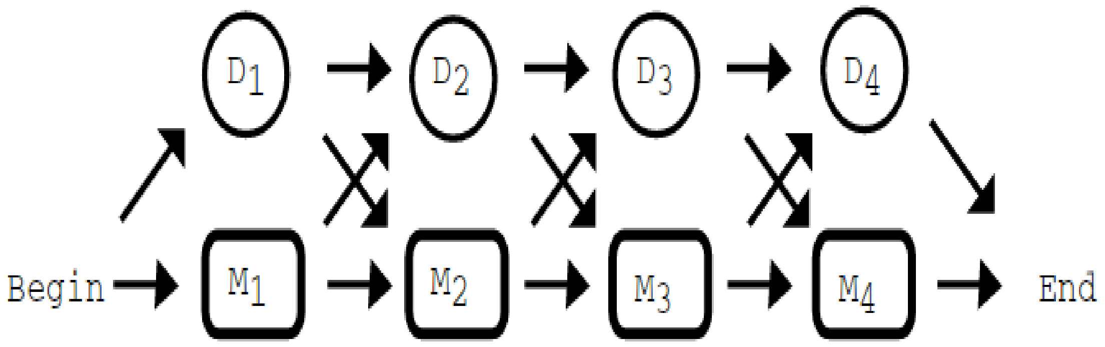


Figure 2: Deletion states

5 Unsupervised training for typos correction

In this section we discuss different ideas to do unsupervised training for typos correction. First, we present Deep Boltzmann machine. Secondly, we introduce Viterbi treillis. Finaly, We show a variant algorithm for hidden markov model which is Baum-Welch

5.1 Deep Boltzmann Machine

The idea consists of a stacked boltzmann machine which is a generative stochastic networks that learn probability distribution over inputs. The stacked boltzmann machine model for spelling correction is based on the idea of treating words as 'documents' and spelling correction as a form of document retrieval (retrieve the best matching).
The strings are represented as a bags of character n-grams. In the stacked boltzamann machine bit strings are learned from training words which are correctly spelled and a corresponding dictionary of bit strings paired with correct spellings is derived. The aim of the model is to predict a bit string with closest Hamming distance.
The words are represented as a bag of n-grams character. The benefit of this representation comes from the fact that it approximate the matching of strings because we rely on the hypothesis that partial matches are possible [5]. So, the bit string with the closest hamming distance to the predicted bit string produces the proposed spelling for the query word as it is depicted the following figure.

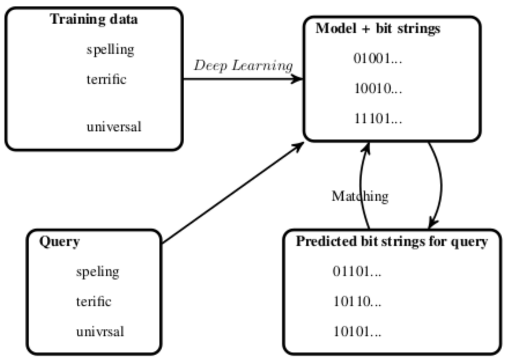


Figure 3: Model for the prediction of bit strings is learned from correctly spelled

5.1.1 Model of Stacked boltzmann machine for typos correction

Three restricted Boltzmann machine with binary hidden units are stacked. It has the following structure : $(N(input) \times N/2(hidden)) - (N/2(input) \times N/2(hidden)) - (N/2(input) \times 128(output))$ where N is the dimension of the input layer (the number of features) and $N = min(N_f, 1000)$ Nf represents the amount of different features in the training data. In order to produce better binary codes, deterministic Gaussian noise was added to the inputs. We generate bit strings of length 128(experimental results). These RBM's are connected to their inverse counterparts in order to reconstruct the original training data from the bit string predictions as it is illustrated in the following figure

The three first Boltzmann machines (RBM 1, RBM 2, RBM 3) are pre-trained for 500 epochs. Then, the chain is put in the reverse mode (RBM' 3, RBM' 2, RBM' 1) and finally the training data is reconstructed (input') from the predicted bit strings. A gradient descent backprogagation of 100 epochs is followed to feed back the measured

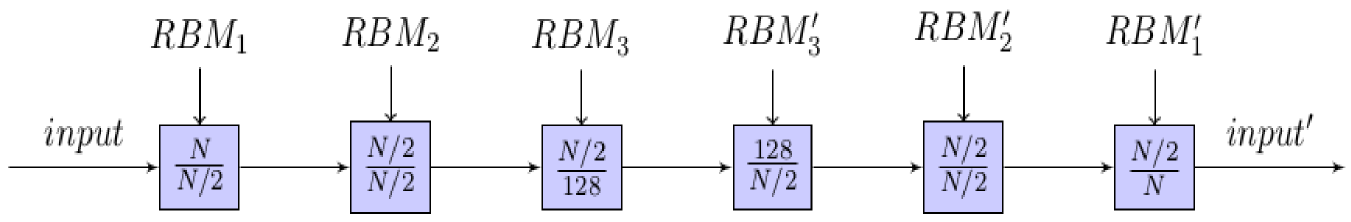


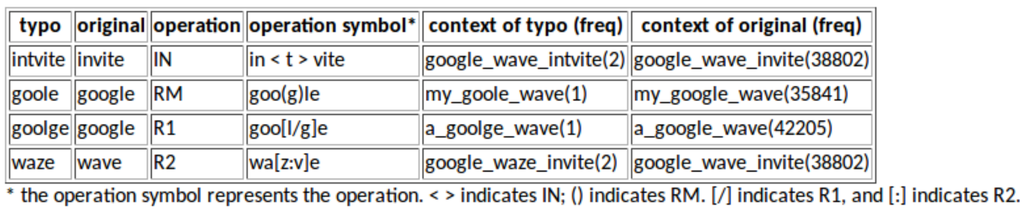
Figure 4: Three RBM's are pre-trained, and then reversed

reconstruction error into the chain and optimizes the various weights. Reconstruction error is the error between original training data and reconstructed, predicted training data. Finally we get a trained and optimized RBMs that represents our final model.
Since Restricted boltzmann machine returns a probability distribution, the predicted values by the model are discretized to binary values by simply thresholding them : values greater or equal to 0.1 becoming 1 [1]. The intuitive explanation of discretization is that RBM'S have asymmetric energy functions that tend to produce values closer to zero than 1. The parameters of training are as follow : learning rate = 0.1, momentum and weight decay were not used. Whereas, momentum = 0.9 was used for back-propagation.
To prepare the input layer, We indexed every word (tweet, wikipedia) for the presence of character bigrams and trigrams, on the basis of a feature lexicon of character n-grams derived from the training data. This lexicon consists of the 1,000 top character n-grams ranked by inverse document frequency, treating the bag of character n-grams of a word as a 'document'. The error correction is more accurate for words with more than 6 characters.

5.1.2 Data for the model

- Tweet data : Corpus of spelling errors in Tweets 39, 172 spelling errors, their correction, and the edit steps to be performed to transform the error to the correct word form (replacement, insertion, deletion). The corpus has 2, 466 different correct word types, which implies on average 15.9 errors per word [2].
  This corpus is a collection of typos in tweets. The corpus consists of pairs of a typo and its original form (such as gogle:google). The typos dealt with this research are classified into four types:
  - INSERT (IN): a character is added to the original word.
  - REMOVE (RM): a character is removed from the original word.
  - REPLACE1 (R1): the order of character is different from the original word (the number of differences is one.
  - REPLACE2 (R2): a character is different from the original word.

  The format of data is illustrated in the following figure.



typo	original	operation	operation symbol*	context of typo (freq)	context of original (freq)
intvite	invite	IN	in < t > vite	google_wave_intvite(2)	google_wave_invite(38802)
goole	google	RM	goo(g)le	my_goole_wave(1)	my_google_wave(35841)
goolge	google	R1	goo[l/g]e	a_goolge_wave(1)	a_google_wave(42205)
waze	wave	R2	wa[z:v]e	google_waze_invite(2)	google_wave_invite(38802)

* the operation symbol represents the operation. < > indicates IN; () indicates RM. [/] indicates R1, and [:] indicates R2.

Figure 5: Format of tweet data

- The Wikipedia corpus : [3] consists of 2,455 errors coupled to correct word forms (a total of 1,922 different correct word types), which means on average 1.3 errors per word.

5.1.3 Implementation details

Fast implementation of this model can be done relying on Reservoir computing [4]. The Oger toolbox is rapid for training and evaluating modular learning architectures on large data-sets. It provides

- Easily building, training and using modular structures of learning algorithms
- Wide variety of state-of-the-art machine learning methods, such as PCA, ICA, SFA, RBMs. . .

It also builds functionality on top of the Modular toolkit for Data Processing (MDP) such as :

- Cross-validation of datasets
- Grid-searching large parameter spaces
- Processing of temporal datasets
- Gradient-based training of deep learning architectures
- Interface to the Speech Processing, Recognition, and Automatic Annotation Kit (SPRAAK)
- Conditional Restricted Boltzmann Machine (CRBM) node
- Perceptron node

5.1.4 Presentation of Deep Boltzmann Machine

Restricted boltzmann machine is a generative model used to model high-dimensional temporal sequences such as video motion capture data (Taylor et al. 2006) or speech (Mohamed and Hinton, 2010). They are trained using contrastive divergence learning procedure (Hinton, 2002) where we have to fine tune hyper-parameter such as the learning rate, the momentum (Method for increasing the speed of learning), the weight-cost, weigh-decay(regularization term such as L2), the initial values of the weights and the number of hidden units and the size of each mini-batch [6]. RBM has one visible layer (v) and one hidden layer (h). The links are only between the hidden units and visible units as depicted in the following figure [7]

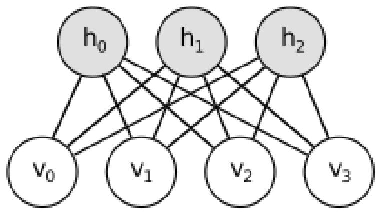


Figure 6: Structure of rbm

5.2 Unsupervised Viterbi Treillis

The idea relies on the Unsupervised spelling correction for Slovak [8]. The Slovak language is characterized by many possible morpho-logical word forms. As a consequence, every operation that considers statistical information about sequences of words, such as word spelling correction, needs to have a proper method of training to improve the performance of the system.
The training corpus contains 49592554 tokens in 2910180 sentences. To do automatic correction the model is based upon Viterbi treillis under certain restrictions. If the word is out-of-vocabulary (OOV), it can be one of :

- Word tokens : regular word, proper name, foreign word
- Non-word tokens : number, web links, incorrectly typed words

The first step of the correction should decide if the out-of-vocabulary word is feasible for automatic word correction. It ensures that possibly useful OOV word are not destroyed and incorrectly replaced by one the vocabulary words. To do so, we define a heuristic to identify an incorrect word which probably contains typing error if :

- It is not in the vocabulary of the correct words
- It is lowercase
- It does not contain features characteristic for foreign words, such as strange letters, numerals, uncommon letter combinations

Once this verification is over, the spell checking can be described as a process of finding the best possible sequence of correct words, given to the list of possibly incorrect words. The words distorted by errors in the sentence can be described as a sequence of observation “o”. Possible corrections “w” for these words “o” are hidden states.
The Viterbi algorithm assign the most probable sequence of the corrected words to the given list of possible incorrect words (marked by the bold lines in the following figure). Word o_t t in a given sentence has some possible corrections $w_i(t)$. Then a value $V_i(t)$ is assigned for every possible correction and using backtracking from the last value it is possible to derive the best sequence of corrections for the given sentence. After all nodes in the Viterbi trellis are evaluated, the best path (the best sequence of correct words) can be found using backtracking, tacking paths with the best Viterbi evaluation as illustrated in the following figure

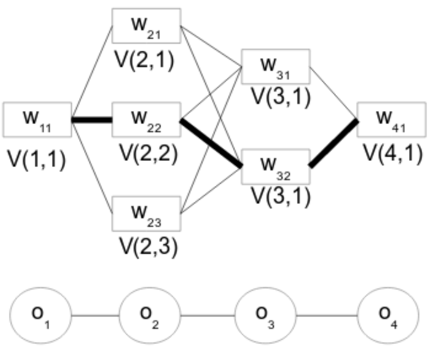


Figure 7: Structure of Viterbi treillis for typos correction

5.2.1 Component of viterbi treillis for HMM

Observation matrix $P(o|w)$: probability of the occurrence of the incorrect form of the word according to its correct form. Since the number of incorrect words is infinite, then the method of maximum likelihood estimation should be estimated. We define a heuristic to estimate $P(o|w_j)$ of observation o according to the state w_j :
$P(o|w_j) = C - j / \sum_{i=0}^{C} (C - i)$
J is order of the word proposed by the spell-checking module, C is a number of proposed correction by spell-checking and $sum()$ is a normalization constant.
The proposed word by the spelling checking module has the highest probability of match with the current observation.
The State transition matrix $P(w|h)$: is a language model of the target language and expresses probability of occurrence of a word according to the given context.
If incorrect words, then state and observation probabilities can be estimated, the most probable sequence of the corrections can be calculated using Viterbi algorithm. For every possible correct word form a trellis is constructed and every node has assigned a certain value $V(t,i)$ that is used to find the best possible sequence as a path in this trellis. The Viterbi value of a node, representing a transition a correct word form to another (a state-transition) can be calculated using a recursive formula:
$V(t,i) = P(o_t|w_{t_i}) max_{j \in S_{t-1}} V(t-1,j) P(w_{t_i}|w_{t-1,j})$
where $V(t,i)$ is Viterbi value for word w_i in time $t, P(o_t|w_{t_i})$ is observation probability.
The steps of the algorithm are as follows :

- Applies rule based corrections in order to deal with the most simple errors
- identify possibly incorrect words
- For every incorrect word proposes a list of corrections
- Constructs a Viterbi trellis, evaluating all possible transitions between correct states
- Using backtracking proposes the best sequence of correct states

The following figure recapitulate the viterbi treillis architecture :

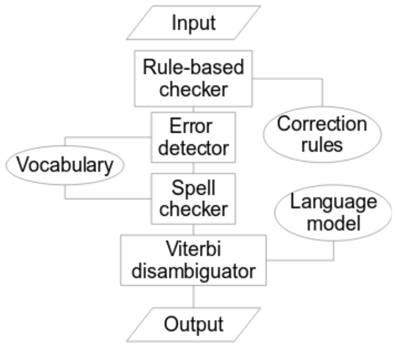


Figure 8: Architecture of Viterbi treillis for typos correction

5.3 Unsupervised Baum Welch algorithm for HMM

As an improvement of our HMM (first and second order), We want to find the most likely model parameters given the data using maximum likelihood estimation $argmax_l P(X_{training}|l)$. This would let us learn model probabilities from raw data. However, we can't determine these probabilities analytically because the state paths are hidden and the equations cannot be solved analytically. In general, iterative hill-climbing algorithm is used (try) to find a good model but can get stuck in local maxima.
Baum-Welch algorithm comes up with a trick to automatically estimate the parameters of an HMM. It can be considered as a special case of the Expectation Maximization (EM) algorithm. It estimates the transition probability $P(l_t^k = 1|l_{t-1}^k = 1)$ and emission probability $P(x_t|l_t^k = 1)$ such that l is a hidden variable (1 to K variables) given multiple sequences of observations. This unsupervised algorithm for HMM works as follow :

1. Start with initial probability estimates
2. Compute expectations of how often each transition / emission is used
3. Re-estimate the probabilities based on those expectations
4. Repeat until convergence

Once we have estimated these parameters of the HMM using complete sequences, we notice that the HMM is a generative model: letting l_t and x_t be the missing entries, we can do the following using the preceding state:
$l_t^* = argmax_{l_t} P(l_t|l_{t-1})$
$x_t^* = argmax_{l_t} P(x_t|l_t^*)$
We choose l_t that maximizes the combination of $P(l_t|l_{t-1})$ and $P(l_{t+1}|l_t)$
Provides a maximum likelihood estimates : attempts to find the model that assigns the training data the highest likelihood. Each iteration of Baum-Welch is guaranteed to increase the log-likelihood of the data but convergence to the optimal solution is not guaranteed [9].

This algorithm allows HMM to do typos correction in unsupervised way. It can be applied directly to our data where first and second order HMM are trained.

References

[1] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. Int. J. Approx. Reasoning,

[2] Tweet corpus.http://luululu.com/tweet/

[3] Wikipedia corpus. http://www.dcs.bbk.ac.uk/ ROGER/corpora.html

[4] Reservoir computing . http://reservoir-computing.org/organic/engine

[5] Stephan Raaijmakers.A Deep Graphical Model for Spelling Correction

[6] Geoffrey Hinton.A practical guide for training restricted boltzmann machine

[7] Boltzmann Machine tutorial. http://deeplearning.net/tutorial/rbm.html

[8] Daniel HLADEK, Jan STAS, Jozef JUHAR.Unsupervised spelling correction for Slovak

[9] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics) .2006

[10] Mona Singh,Jordan Parker.Profile Hidden Markov Models. Topics in Computational Molecular Biology