Text Normalization Using Weighted Finite State Transducers

Mainak Biswas[1], Kushal Verma[2]

[1]Department of Information Technology, Jadavpur University, Kolkata-700106, India.

[2]Chief Engineer, Speech Recognition and Natural Language Processing, Samsung Research Institute, Bangalore-560037, India.

mainak.biswas.dbl@gmail.com, kushal.verma@samsung.com

Abstract

Although automatic speech recognition systems today are providing impressive results, one factor that still exacerbates it performance is the presence of transcripts in non-standard forms. Numbers, dates, scores, addresses, social media messages can all contain data that is not in their verbal form, which in turn may confuse ASR and NLP models for language translation, speech to transcript generator, transcript to speech generator, etc. To avoid this one of the most important component in any ASR pipeline is a text normalizer. A text normalizer converts all words present in a transcript that are in a non-standard (verbal) form to its corresponding standard form. Machine learning and deep learning models are popularly used in many ASR and NLP application and often produce outstanding results. Unfortunately, there isn't enough labelled available for all domains where text normalization is required. Thus supervised learning would not perform well. In this work we propose a Text Normalization model using mathematical tools like finite state transducers and weighted finite state transducers defined over a tropical semi-ring. The model recognizes the parts of an text input that is in its non-standard form and converts it to its standard form. Thus the sentence that the transducer outputs is in it standard form. A total of 10 domains are addressed in this work. The average accuracy obtained by the model is 99.00%.

Keywords: text normalization, ASR, automata, finite state transducers (FSTs), weighted FSTs, Semi-rings, tropical semi-ring, grammars, openfst, thrax

I. Introduction

Humans are continuously trying to increase and ameliorate human-machine communication. The most widely used mode of communication in humans is speech. Therefore, it would be extremely beneficial if computers could understand spoken languages and act according to verbal instructions given by users. Unfortunately, the major way of communicating with computers today is through a through text data typed into it through a keyboard. This is often deemed to be cumbersome by many. To make computers understand speech we would first need to convert it to its corresponding transcript. Automatic Speech Recognition (ASR) is the first step towards making computers understand spoken languages *(Samudravijaya)*. It is the field of research that deals with generating transcripts for a particular utterance, whose speech waveform is provided to the computer.

ASR and natural language processing (NLP) models are trained using waveform and their corresponding textual transcripts. One may notice that there are often dissimilarities between written and spoken languages. For example, when we say "I am five feet ten inches tall", a typical transcript of this speech could be "I am 5'10'' tall". Similar things can be seen with numbers, abbreviations, dates, addresses and many more. With the advent of social media, non-standard representations of words are becoming too common. For example, we often find people writing "thank you" as "thank u", "because" as "bcoz", "night" as "ni8" and so on. This is creating non-standard forms of languages more popular. When a ASR model is trained with transcripts as mentioned above, the performance of even the most state of the art ASR model drops drastically.

One of the most popular ways of overcoming this problem is by Text Normalization (TN). The objective of TN is to convert the non-standard parts of a transcript into its standard (verbal) form *(Veliz et al., 2019),* (Sproat et al., 2001). After text normalization is performed on the transcript, it becomes more uniform and the performance of the ASR model drastically increases. Hence TN is one of the most important component of an ASR pipeline.

Although Deep Learning models like the recurrent neural networks (RNN) are popularly used for various NLP and ASR application; such supervised learning algorithms require huge amount of labelled data to train these models. Unfortunately, there isn't enough labelled data for text normalization on wide variety of domains, and hence alternative mathematical models are needed to perform text normalization. One such alternative is by using transducers *(Hui, 2019)*. In this work, we propose a model for Text Normalization using Finite State Transducers (FSTs) and weighted Finite State Transducers (WFSTs) defined over semi-rings.

One of the reasons TN is challenging is because a single written form can have multiple verbal forms. For example: "26/3" can mean "twenty six over three" (if it is representing a number), "twenty sixth march" (if it is a short hand for a date), or "twenty six by three" (if it is a house number). For addressing such challenges we need weighted automata, and various operations on its weights to get the most appropriate normalized form of a transcript.

This paper is arranged as follows: Section II deals with the models that have been used for text normalization in this work. Section III describes the algorithms and domains addressed for text normalization. Section IV deals with the technologies used to generate the model. Section V shows the results obtained from various experiments conducted on our text normalization model. Section VI concludes our work and discusses its future scope.

II. Mathematical Models Used

(A) Finite Automata

Background:

(a) Finite State Automaton (FSA)

A general finite state automaton (i.e. non-deterministic FSA) is a 5-tuple, $F = (Q, \Sigma_\varepsilon, \delta, q_0, F)$, where:
Q: is the set of states,
$\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, where Σ is the set of input symbols, and ε is the empty string,
δ: $(Q\ x\ \Sigma_\varepsilon) \rightarrow\ 2^Q$, where 2^Q is the power set of Q (i.e. set of all possible combinations of the elements of Q). More intuitively speaking, it is a function that maps from a pair (q, a), where $q \in Q, and\ a \in \Sigma_\varepsilon$, to an arbitrary set of states $A \subseteq Q$.
q_0: is the set of start states, $q_0 \subseteq Q$.
F: is the set of accept states, $F \subseteq Q$.

Therefore, FSA can be thought as a machine, that takes a string as an input and outputs whether it belongs to a language L (i.e. a sequence of input symbols) or not. (i.e. it outputs a binary value) $L(F)$ is known as the language accepted by the automaton F. Therefore, it is also known as an acceptor. A graphical representation is the best way of representing an automaton, where the nodes represent the states and the directed edges represent the transition on an input symbol $a \in \Sigma_\varepsilon$.
Accepting a string: Due to the non-deterministic property of an FSA the machine can simultaneously exist in multiple state at a particular moment (i.e. after it has exhausted a certain length of the input string). So there may be multiple paths that the automaton may take when an input is fed to it. If one of the paths end at the a state $q_f \in F$, then the string is accepted by the automaton.

Example: Let $\Sigma = \{0,1\}$, we try to define an automaton F that accepts a language L of strings that either end with "01" or "10".
Automation that will accept is shown in Fig 1.

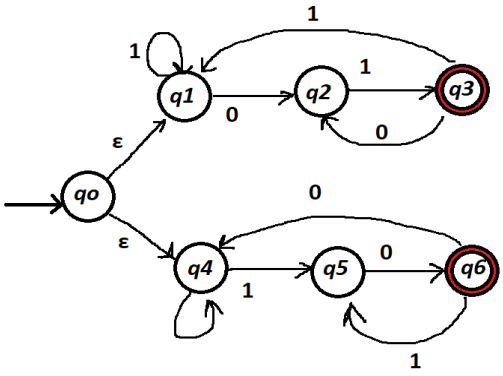


Fig. 1: q_0 is the start state, F = {q_3, q_6} is the set of accept states, and ε takes us to q_1 and q_4, which essentially serves as sub-automaton to accept strings ending with '01' and '10' respectively.

Algebraic representation of the automaton can be in terms of a regular expression. The regular expression for the above automaton is $(0 \mid 1)^*\ 01 \mid (0 \mid 1)^*\ 10$. It can be shown that an automaton and a regular expression are equivalent *(Aho & Ullman, 1977)*.

(b) Finite State Machine (FSM)

In digital electronic, a synchronous sequential circuit (whose output depends upon the present as well as sequence past inputs) is generally represented by an FSM, which has states (just like FSA) which captures the sequence of past inputs, and edges which represent transitions between states (in which the machine could be in). As a synchronous digital circuit must continuously produce outputs, in response to the present input and the current state, the machine produces outputs either when it reaches a state (Moore machine), or when it makes a transition (Mealy machine). Moreover, there isn't anything like the accept state. The machine keeps on transiting between states and produces output continuously (as long as there is an input given to the machine). Mathematically, speaking it can be expressed as a 6-tuple $M = (\Sigma, \Gamma, Q, q_0, \delta, \lambda)$ where:

Σ, Q, q_0 have their usual meaning (as in the case of an FSA).

Γ is the set of output symbols.

δ: $Q\ x\ \Sigma \rightarrow Q$, the way in which the transition function differs from that of a general FSA is that it is deterministic (i.e. exactly one transition can exist for a pair (q, a), $where\ q \epsilon Q, and\ a \epsilon \Sigma$). This is mainly due the fact that FSMs are widely used to simulate real-world objects (e.g. counters), where non-determinism is not a popular idea (as classical computers can exist at just one state at a particular time).

λ: $Q \rightarrow \Gamma$, for a Moore machine *("Moore Machines,")*, i.e. the machine gives an output when it reaches a state, therefore λ is a mapping from a state to an output symbol.

λ: $Q\ x\ \Sigma \rightarrow \Gamma$, for a Mealy machine *("Mealy Machines,")*, i.e. the machine gives an output when it makes a transition, and therefore λ, is a mapping from pair (q, a), to an output symbol $b \epsilon \Gamma$.

Figures 2 & 3 respectively show a Moore and a Mealy machine for a 'not gate'.

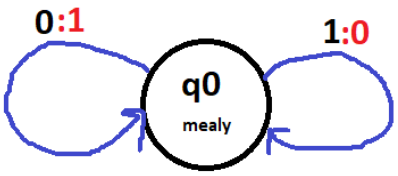


Fig. 2

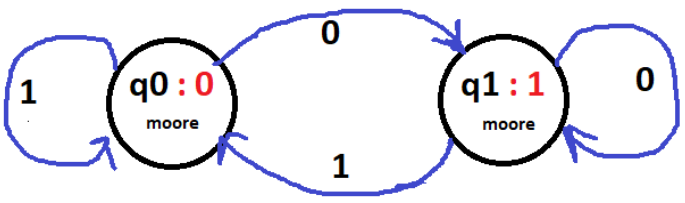


Fig. 3

Fig. 2 (left): It is the Mealy machine for a not gate, clearly the output (in red) depends on both the state as well as the input (inverting the input here). Fig.3 (right): Moore machine for the same, the output only depends on the state here, q_0 outputs 0, while q_1 outputs 1. Note: q_0 is the start state for both the machines.

(B) Finite State Transducers (FSTs)

Motivation:

In fields like Automatic Speech Recognition (ASR), just an automaton to accept a language is often not good enough. E.g. If we want to perform text normalization on dates of the form dd-mm-yyyy, just building an automaton would not be good enough. An automaton would only accept valid date (i.e. return true/false).

But our objective is not only accept a date like '15-08-1947' but also to convert to 'fifteenth august, nineteen forty seven' (i.e. to its verbal form). One may argue, that Moore/Mealy machine could help our cause; but it too has its limitations. An FSM doesn't accept a string, it produces outputs irrespective of whether the string is accepted or not. Moreover, in some cases non-determinism is essential for building intuitive/readable machines, or for converting a grammar to its corresponding automaton (directly building a deterministic automaton may often be very complex).
A finite state transducer somewhat combines both these properties. On one hand it allows non-determinism, while on the other it outputs a string for a particular input, iff the input is accepted. This enables us to build machines that can perform tasks as mentioned above.

FSTs Definition:

Mathematically an FST a 6-tuple $F = (Q,\ \Sigma_\varepsilon\ ,\ \Gamma_\varepsilon,\ \delta,\ q_0,\ F)$ where,
$\Sigma, Q,\ q_0,\ F$ have their usual meaning (as described above).
$\Gamma_\varepsilon = \Gamma\ \cup\ \{\varepsilon\}$, where $\Gamma\ and\ \varepsilon$ have their usual meaning.
$\delta \subseteq\ (Q\ x\ \Sigma_\varepsilon\ x\ Q\ x\ \Gamma_\varepsilon)$, is the transition relation.
A transducer is viewed as a directed graph. If $\exists\ (q_i, a, q_f, b)\ \epsilon\ \delta$, then one can conclude that there is an edge from q_i to q_f, on an input a. When the transition is made the machine outputs a symbol b. Here $q_i, q_f\ \epsilon\ Q, and\ a\ \epsilon\ \Sigma_\varepsilon, b\ \epsilon\ \Gamma_\varepsilon$.
The above transition function is non-deterministic. It allows the machine to make ε transitions as well as to output empty string (ε).
A string is accepted just as a FSA accepts a string. As the FST makes progress on the input string the output is constructed simultaneously too (concatenation of the output symbols). If at least one path ends up in an accept state, is the input string accepted and the corresponding output is returned. If none of the paths end up in an accept state the string is rejected and the output is not returned.
The example below will bring more clarity to the above discussion.

Consider building an automaton that will accept the numbers 1, 5, 11, 15, 51 and 55 only. Our objective is not only want to accept it but also write them in words (i.e. normalize them to their verbal form).
The transducer below is expected to perform the desired task.

The input and the output symbols are: $\Sigma = \{1, 5\}\ and\ \ \Gamma = \{"one", "five", "eleven", "fifteen",\ "fifty"\}$. q_0 is the start state.

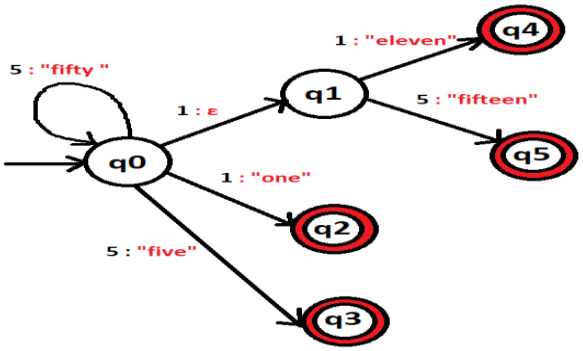


Fig. 4: It shows the transducer required to accept only the above mentioned numbers and return their verbal form. F = {q_2, q_3, q_4, q_5,} are the accept states. q_2, accepts both 1 and 51 and returns 'one' and 'fifty one' respectively. q_3 behaves similarly to accept 5, 55. The transducer on inputting 11 or 15 goes to q_4 and q_5 respectively through q_1. The transition q_0 to q_1 outputs ε, i.e. an empty string so that 'eleven' and 'fifteen' is outputted without any unnecessary prefix on getting another 1 or 5 respectively.

(C) Weighted Finite State Transducers

Introduction to WFSTs and Semi-rings:

One may be able to notice that there can be cases in which multiple paths lead to an accept state on a particular input. In such a case, each path would produce a different output (on the same input). We often need to decide what we want to do with all these possible outputs. Sometimes we may need to their union, while at other times we may just want the most appropriate one.
For solving these problems weighted FSTs are introduced. Weighted Finite State Transducers (WFSTs) are FSTs whose edges are augmented with another parameter, i.e. weights. It is analogous to a weighted-directed graph.
Just adding weights would have no meaning if operations were not performed on them. Moreover, the weights and need not always be real numbers. We need to define both these parameters according to what the question demands. This would in turn help us modify WFST algorithms according to our needs *(Mohri, 2009)*. So, a generic definition the weights and operation on them are defined in terms of semi-rings. These following example will help us build intuition about how a WFSTs over a semi-ring S, is defined.
Fig. 5 and 6, shows two WFSTs (designed to accept a string '01', and output 'xy' or 'yx'), with the same architecture (i.e. the same: $Q,\ \Sigma = \{0,1\},\ \Gamma = \{x, y\},\ \delta,\ q_0,\ F$). But what varies in both these cases are the weights and the operation on them. Therefore, these WFST would produce different strings as the outputs (i.e. they would select different paths as the most appropriate one to reach the accept state). "01" into both the transducers.

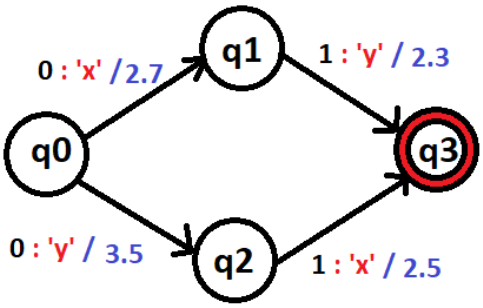


Fig. 5: WFST defined to return the path with the minimum cost.

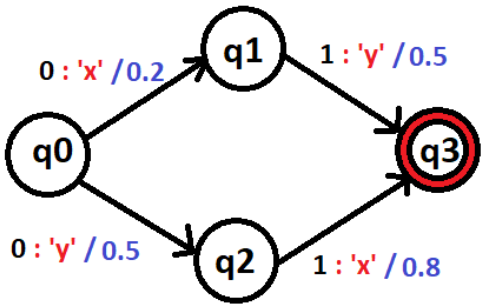


Fig. 6 : WFST defined to return the most likely path.

For the WFST in Fig. 5, we the weights represent the penalty the transducer must pay to make a transition. Thus the operation would be to find the path with the minimum cost, i.e. the path ($q_0 \rightarrow q_1 \rightarrow q_3$), and hence outputs "xy". The transducer performs the following computation to select the aforesaid path.

$$\min_{\pi \epsilon P(q_0, q_4)} (cost(\pi)) = \min\ ((2.7 + 2.3), (3.5 + 2.5)) = 5.0 \qquad (1)$$

Therefore the transducer returns "xy" as the output. Note: The cost function is the arithmetic sum.

In Fig. 6, the weights are probabilities and we want to select the path that is most likely (maximum probability). Considering the bits in the input stream to be independent, the transducer computes the following to select the path ($q_0 \rightarrow q_2 \rightarrow q_3$).

$$\max_{\pi \epsilon P(q_0, q_4)} (probability(\pi)) = \max\ ((0.2x0.5), (0.5x0.8)) = 0.4 \qquad (2)$$

Therefore the transducer returns "yx" as the output. Note: The probability function is the arithmetic product. The above example serves as the intuition to build up the formal definition of a semi-ring.

Semi-rings

In abstract algebra, a semi-ring is defined as a 5-tuple $S = (\kappa, \oplus, \otimes, \bar{0}, \bar{1})$ where,
κ is a set of elements on which the semi-ring is defined.
$\oplus\ and\ \otimes$ are the semi-ring operations that are performed on the set of elements κ. These operations are abstract and can be decided based on a certain problem. Casually they are often called the 'addition' and 'multiplication' operator of the semi-ring.
$\bar{0}\ and\ \bar{1}$ are the identities associated with $\oplus\ and\ \otimes$ respectively. Therefore they are casually called the additive and multiplicative identity.
Consider three elements $x, y, z\ \epsilon\ \kappa$, then the semi-ring must satisfy the following properties:
(i) **Under $\oplus$:**

- *Closure*: $x \oplus y\ \epsilon\ \kappa$. This means that the result of $\oplus$ remains in the semi-ring.
- *Commutative:* $x \oplus y = y \oplus x$
- *Associative:* $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
- *Additive* Identity: $x \oplus \bar{0} = x$

(ii) **Under $\otimes$:**

- *Closure*: $x \otimes y\ \epsilon\ \kappa$. This means that the result of $\otimes$ remains in the semi-ring.
- *Associative:* $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
- *Distributive:* $(i)\ x \otimes (y \oplus z) = x \otimes y \oplus x \otimes z$ (left distributive)
  $(ii)\ (x \oplus y) \otimes z = x \otimes z \oplus y \otimes z$ (right distributive)
- *Multiplicative Identity:* $x \otimes \bar{1} = x$

WFSTs Definition:

Mathematically, a WFST can be defined as a 9-tuple, $W = (Q, \Sigma_\varepsilon, \Gamma_\varepsilon, \delta, q_0, F, \lambda, \rho, \kappa)$ where:
$Q, \Sigma_\varepsilon, \Gamma_\varepsilon, q_0, F$ have the same meaning as in case of an ordinary FST mentioned above.
κ is the set of weights to be associated edges. A WFST is always defined over a semi-ring, and is same as the κ defined for the semi-ring.
$\delta \subseteq (Q\ x\ \Sigma_\varepsilon\ x\ Q\ x\ \Gamma_\varepsilon\ x\ \kappa)$ is the transition relation. If the weighted transducer is viewed as a directed graph, then $(q_i, a, q_f, b, w)\ \epsilon\ \delta$ can be considered as an edge between q_i and q_f, on an input symbol $a\ \epsilon\ \Sigma_\varepsilon$, and an output symbol $b\ \epsilon\ \Gamma_\varepsilon$. $w\ \epsilon\ \kappa$, is the weight associated with the edge.

ρ: $q_0 \rightarrow \kappa$, i.e. it is a mapping from an input state to a weight set κ. It can be considered as the *potential* of the transducer, as it has a penalty $p\ \epsilon\ \kappa$, whenever it starts at an initial state.
λ: $q_0 \rightarrow \kappa$, is mapping from the accept states to a weight set κ. It is the penalty the transducer has to pay when it ends up in an accept state.

Fig. 5 and 6 illustrates the above definition of a WFST. Note: $\lambda\ and\ \rho$ are null sets for these examples.

WFST over a semi-ring

We define a WFST over a semi-ring using the intuition that the example of the WFST provided above conveys. In the example we see that there are two operations performed over the weights of the WFST. The weights can be considered to belong to the set κ. Furthermore, the semi-ring has two operations too. Thus we can define a WFST over a semi-ring, as follows:
Let $P(q_0, x, y, F)$ be the set of all paths from a start state on an input string $x \epsilon \Sigma^*$ and an output string $y \epsilon \Gamma^*$. Let $\pi \epsilon P(q_0, x, y, F)$ be one path from the an initial to the a final state, then we define $w(\pi)$ as the total weight of the path, $\lambda(\pi)$ and $\rho(\pi)$ be the weight of the initial and final state respectively (of the path π). Then we define a transducer on a semi-ring S as:

$$T(x, y) = \bigoplus_{\pi \epsilon P(q_0, x, y, F)} \lambda(\pi) \otimes w(\pi) \otimes \rho(\pi) \quad (3)$$

If e ϵ π is an edge in the path, $q_i \epsilon q_0 \; and \; q_f \epsilon F$, be the initial and the final state in the path π, then $w(\pi), \lambda(\pi)$ and $\rho(\pi)$, can be mathematically written as:

$$w(\pi) = \bigotimes_{e \epsilon \pi} w_e(e) \quad \text{,where } w_e \text{ returns the weight of an edge} \quad (4)$$

$$\lambda(\pi) = \lambda(q_i) \quad (5)$$

$$\rho(\pi) = \rho(q_f) \quad (6)$$

When we carefully see the calculation involved in example with Fig. 5, we clearly see that we are exactly performing the same thing as described in equation 3. If we replace $\oplus$ with min function and $\otimes$ with the arithmetic addition we exactly get what is worked out in the example.
Continuing to use this level of abstraction would not be very beneficial for a practical domain text normalization. Thus, we need to define the commonly used semi-rings for all ASR applications.

Commonly Used Semi-rings in ASR

Table 1: Contains the most popularly used semi-rings in ASR applications

Semi-Ring	$\boldsymbol{\kappa}$	$\boldsymbol{x \oplus y}$	$\boldsymbol{x \otimes y}$	$\bar{\mathbf{0}}$	$\bar{\mathbf{1}}$
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min(x,y)	x+y	+∞	0
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$x \oplus_{log} y = -\ln\ (e^{-x} + e^{-y})$	x+y	+∞	0
Boolean	{0,1}	Boolean or (+ / v)	Boolean and (. /^)	0	1

An FST can be considered as a WFST defined over a boolean semi-ring (either an edge is present/not).
The semi-ring that has been used in this project is the tropical semi-ring and it is by far the most widely used semi-ring in ASR related applications *(Viterbi, 2019)*. Therefore, we shall prove that it is a semi-ring.

Proof of Tropical semi-ring being a semi-ring:

Let $x, y, z \; \epsilon \; \mathbb{R} \cup \{-\infty, +\infty\}$,
(i) Showing properties under $\oplus = \mathbf{min}$ holds:

- *Closure*: $min(x, y) = \begin{cases} x & x \leq y \\ y & y < x \end{cases}$, thus $min(x, y) \epsilon \; \mathbb{R} \cup \{-\infty, +\infty\}$. Hence closed under $\oplus$.

- *Commutative:* $min(x, y) = min(y, x)$, as the minimum function clearly is commutative.
- *Associative:* *Let* r = min(min(x, y), z).
  r is obtained by first finding the minimum between x *and* y and in turn comparing that result with z (to get the smallest of the three numbers). The same problem can also so be solved by finding the minimum between y *and* z first and then comparing it with x.
  Therefore, r = $min(x, min(y, z))$ also.
  - ⇨ $min(min(x, y), z) = min(x, min(y, z))$
  - ⇨ $(x \oplus y) \oplus z = x \oplus (y \oplus z)$, (hence associativity property holds)
- *Identity:* *Clearly, min(x, +∞)* = x.

(ii) Showing properties under $\otimes = +$ holds:

- *Closure:* let $x + y = m, then\ m \in \mathbb{R} \cup \{-\infty, +\infty\}$. Therefore its closed under $\otimes$.
- *Associativity:* $x + (y + z) = (x + y) + z$, always holds. It is a property of the 'arithmetic sum' operator.
- *Identity:* $x + 0 = x$, hence the identity property also holds.
- *Distributive:* Since $\otimes$ (= +) operator is a commutative in nature (i.e. $x + y = y + x$), proving left distributive property will also prove it for the right distributive case.
  Left Distributive property: $x \otimes (y \oplus z) = x + min(y, z)$
  $= min(x + y, x + z)$
  $= x \otimes y \oplus x \otimes z$, (hence left distributive property holds)
  Here, x can be treated as a constant that is added to the $min(y, z)$ to get the desired result. The same thing can be obtained if we add a constant x to both y *and* z before we find their minimum.

Hence the Tropical semi-ring has all the properties of a semi-ring. It's also called a commutative semi-ring as the $\otimes$ operation is commutative.
Properties of all the other semi-rings can also be shown in a similar way *(Hao, 2015)*.

Operation on Transducers

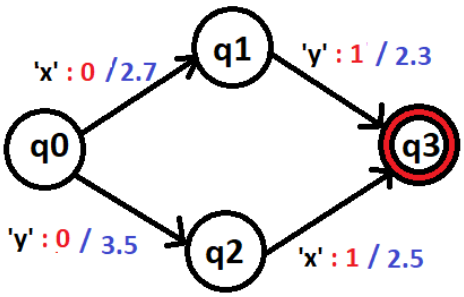
Mathematical properties like Kleene Closure, union, concatenation and reversal of FSTs / WFSTs are completely analogous to that of FSAs. *(Aho & Ullman, 1977)* discusses all these algorithms with respect to FSAs. There are 2 additional mathematical properties that a transducer has:

(i) Inverse: The inputs and outputs are swapped.

$$T^{-1}(x, y) = T(y, x), \quad (7)$$

(ii) Projection: All the output symbols are deleted and the corresponding acceptor is returned. If there are multiple edges on the same input in the original WFST, all of them but one are dropped and the weight is assigned according to the $\oplus$ function of the semi-ring that it is defined on.
Intuitively, for a tropical semi-ring the shortest path on an input is kept, and other paths on the same input are deleted. Fig. 8 provides the resultant transducer when projection is performed on the transducer in Fig. 5.

$$T(x) = \downarrow T(x, y) = \bigoplus_{y} T(x, y) \quad (8)$$



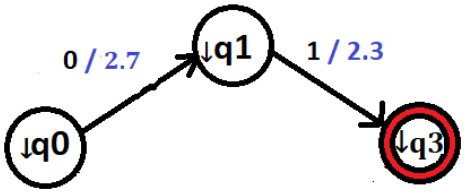


Fig. 7 (left): It is the inverse transducer of the one shown in Fig 5. Fig. 8 (right): It is the acceptor obtained on performing projection on the transducer in Fig 5.

WFSTs have other advanced operation like composition and optimization (ϵ-removal, determinization, weight pushing and minimization). *(Mohri)* discusses them in details. Optimization is performed to get the most efficient transducer. It is more of an automata theory topic and is not popularly used when it comes to applications in ASR. The most important operation that is required in designing transducers for Text Normalization is composition. It helps in reusing transducers that have been already defined.

Motivation behind defining Composition of two transducers

Say we have designed two transducers that will normalize cardinal and ordinal numbers respectively. Now we want to reuse them to build a transducer that will accept and normalize legal dates only. It would be cumbersome to define the transducers that normalize cardinal and ordinal numbers again. Instead, we may define an acceptor for accepting legal dates and in turn pass it to the above transducers. When that is done we would want the transducers already defined to normalize a legal date (accepted by the acceptor) only. Any illegal date will not be accepted and hence not normalized. This would make our work easier and the code clearer.

Composition: A High-Level Description

If we have two transducers, $T_A \; and \; T_B$, defined by the following 8-tuples:
$T_A = (Q_A, (\Sigma_\varepsilon)_A, (\Omega_\varepsilon)_A, \delta_A, (q_0)_A, F_A, \lambda_A, \rho_A, \kappa)$
$T_B = (Q_B, (\Omega_\varepsilon)_B, (\Gamma_\varepsilon)_B, \delta_B, (q_0)_B, F_B, \lambda_B, \rho_B, \kappa)$ are defined over a semi-ring $S = (\kappa, \oplus, \otimes, \bar{0}, \bar{1})$, then their composition $T_A \; o \; T_B$ is defined as ($x \; \epsilon \; \Sigma^* \; and \; y \; \epsilon \; \Gamma^*$):

$$T(x,y) = T_A \; o \; T_B \; (x,y) = \bigoplus_{z \; \epsilon \; \Omega^*} (T_A(x,z) \otimes T_B(z,y)) \qquad (9)$$

Intuitively, $T_A \; o \; T_B$ can be considered similar to a function composition. If T_A and T_B were functions defined on domains d_A and d_B , and had ranges range $r_A \; (= d_B \;)$ and r_B, then $T_A \; o \; T_B$ in case of a transducer would be analogous to $T_B \; (T_A(x))$ while dealing with functions.
Therefore, computing $T_A \; o \; T_B$ essentially means to produce a transducer T that would perform the same tasks as the following:
(i) Take an input x and pass it through Transducer T_A to produce an output z,
(ii) Feed z into Transducer T_B to get the required output y.

Note: The weights associated with the composed transducer T, would be calculated in accordance with the ⊗ operation of the semi-ring. Formally, if e_A and e_B are edges of transducer T_A and T_B respectively, then the effective edge weight e formed after valid composition, is defined as:

$$e = w(e_A) \otimes w(e_B) \quad (10)$$

The algorithm is discussed in details in *(Mohri)*.

The following example shows how composition can be used in Text Normalization and other ASR application.
Example: Say we want to normalize valid marks that a student can score in a 30 marks quiz. Consider that we already have defined a transducer B that normalizes cardinal number in the range [-10 billion, 10 billion], and we want to reuse it for normalizing the marks of a student (valid only, i.e. if 50 is entered, we want the effective transducer to reject the string).
The following diagram shows a block diagram how it can be solved using composition.

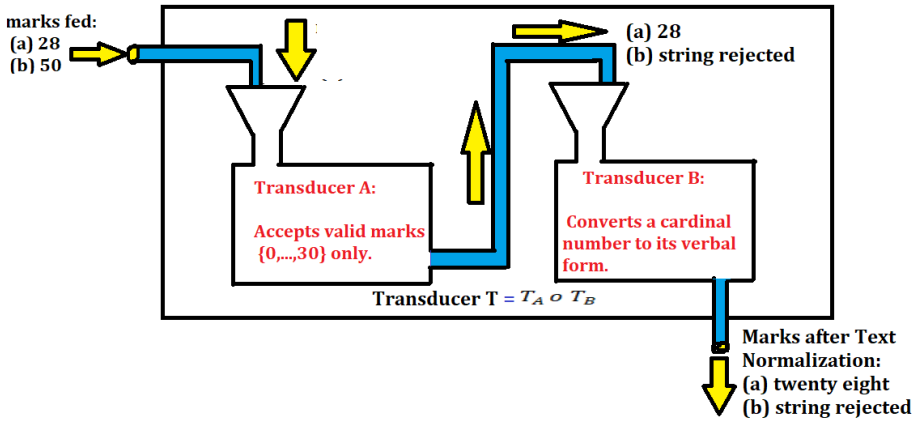


Fig. 9: Transducer A accepts valid marks in the range 0-30 only, and in turn passes it to a valid marks to transducer B which normalizes it to its verbal form. If it is not in the range transducer A rejects the string and hence the entire transducer rejects the string too.

III. Algorithms & Domains for Text Normalization

To build a robust TN model, we need to decide domains where text normalization is necessary. Once that is figured out, we need to define transducers that would normalize the respective domains. These domain specific transducers can later be integrated to generate a TN model.
We have selected 10 domains for perform text normalization on. The domains and their sub-domains are described in table 2.
Once the transducers for all these domains are defined, we generate a weighted transducer to integrate all of them. The approach used for integration is intuitive. Key intuitions behind the integration algorithm:

(i) Defining separators: while normalizing a sentence the models need to choose which substrings it needs to normalize, and for this we should define separators between words like: (‘,’ | ‘.’ | ‘;’ | ‘ ‘ | ‘\n’). The separators generate substrings that needs to passed into the transducers already defined. We augment a high weight to the edge accepting separators so that valid abbreviations having ‘.’ is not

treated as a separator. Assigning a high weight to it also makes the integrated transducer accept the longest normalizable string before going on to check the next substring.

(ii) High penalty for unrecognized string: This ensures that all substrings can be normalized, is indeed normalized. For accepting a substring that can't be normalized we incur a high penalty which again makes the transducer normalize as much as it can.

(iii) Understanding relative importance: Let $A\ and\ B$ be two transducers that normalizing languages $L(A)\ and\ L(B)$ respectively. Say, $L(A) \subseteq L(B)$, and $L(A)$ was designed to normalize some special strings in the $L(B)$. Therefore we might wish to give $L(A)$ more importance compared to $L(B)$, so the special cases are handled properly. For doing that, one must assign a lower weight to transducer A relative to transducer B.

Table 2. Describes the domains addressed for text normalization.

Srl. No.	Domain	Sub-domains
1.	Abbreviations	country names, english grammar, profession, sports, subjects, technical and social media abbreviations
2.	Address	house, street, flat numbers, common abbreviations used
3.	Currency	rupees, dollars, pounds, euros, yen, yuan, taka, dirham, etc.
4.	Dates	dd-mm-yyyy, dd-mm-yy, mm-dd-yyyy and other valid ones. Dates written in letters, like 12th Jan, 1999
5.	Web Related	Emails addresses, and all types of urls
6.	Maps	Directions, latitudes and longitudes, etc.
7.	Measurements	Length, weight, volume, area, temperature, time(duration), etc.
8.	Numbers	Cardinal, ordinal, comma-sensitive numbers in indian and western number systems, phone numbers and scientific numbers.
9.	Scores	Tennis, cricket, football & hockey, etc.
10.	Time	12 hrs, 24 hrs clocks, and o' clock formats

The Working of the Model

The flowchart of this algorithm is shown in figure 10. For example, the input to the integrated grammar is "My 1st internship was for 2 months; from 18-5-20 to 13-7-20". Then the model will normalize it as "My first internship was for two months; from eighteenth may, two thousand twenty to thirteenth july, two thousand twenty".
The spaces, commas will separate the string into substrings. The process of substring selection is shown in figure 11. The integrated transducer will now pass these substrings one at a time through the main transducer of all domains and see which domain is the most appropriate for a particular substring and normalize it accordingly. This is done by adding weights to the edges, so that the most likely path is selected.

If normalization is not required, it just passes the substring as entered. Words like 'my', 'for', 'months' don't need normalization, and hence will be kept as it is. Similarly, "1st" and "2" belongs to the domain of numbers (ordinal numbers and cardinal numbers subdomains respectively) and would be normalized accordingly. "18-

5-20” and “13-7-20” will be identified as a date (of the form dd-mm-yy) and normalized accordingly. We also make sure that the largest possible sequence gets normalized, i.e. we make the transducer incur a large penalty on choosing a separator. This makes sure that “13-7-20” is not normalized as “thirteen-seven-twenty”, gets identified as a date so that it can be normalized as “thirteenth july, two thousand twenty”.

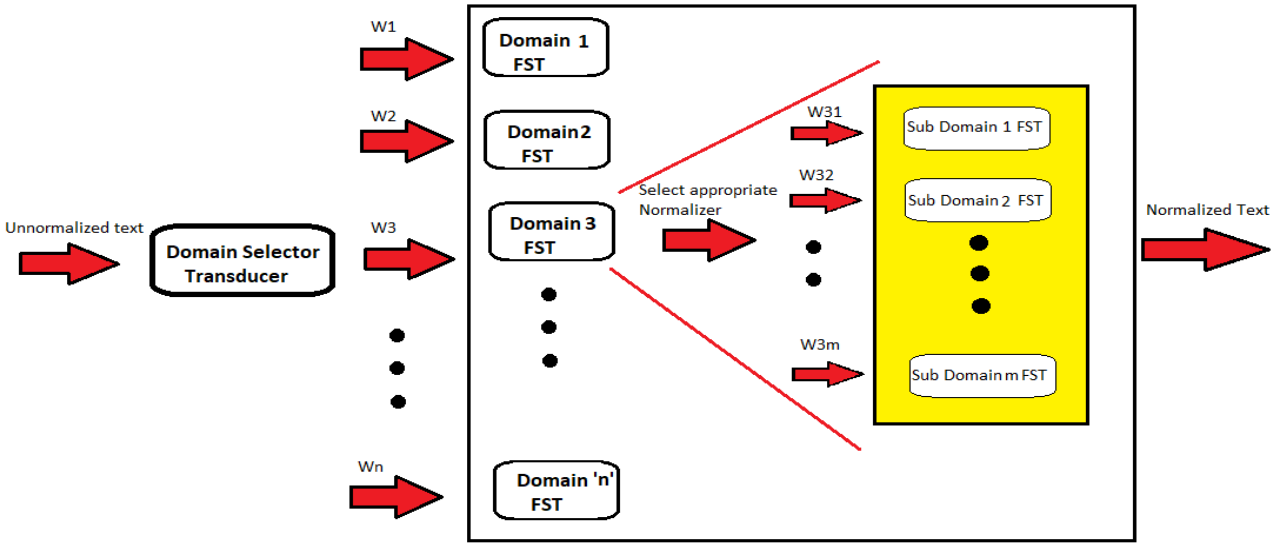


Fig 10: Showing the block diagram of the text normalization model. W_x represents the weight of an edge x.

Figure 11, shows the details of the domain selector transducer. It shows how the model separates substrings.

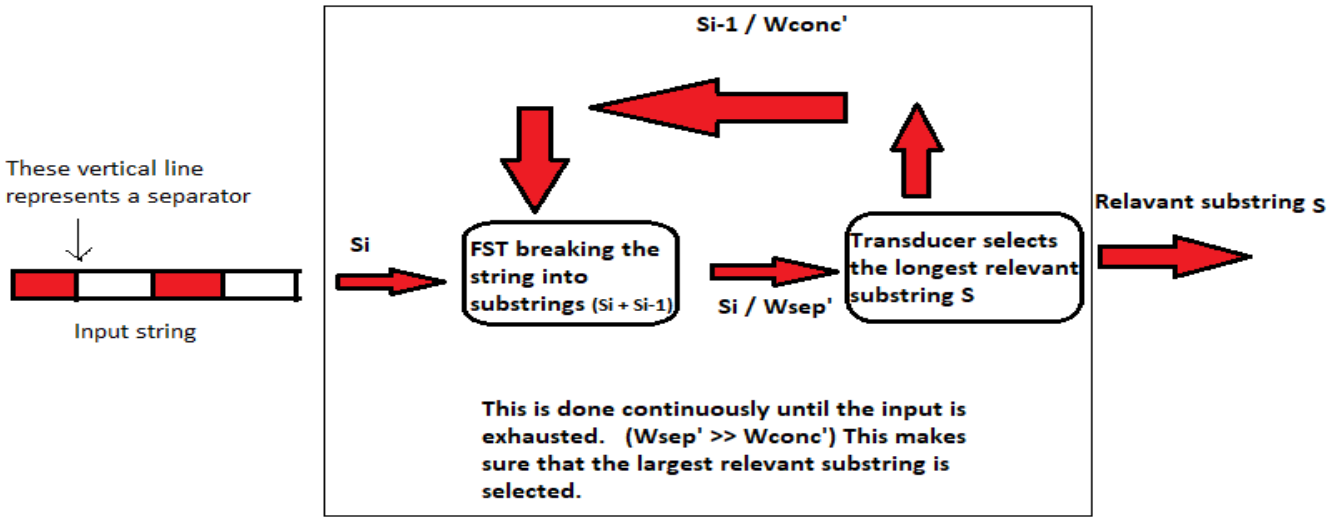


Fig 11: This describes the domain selector block in details. Which makes sure that the longest relevant string is passed to the domains which can in turn select the appropriate normalization scheme by using the weights on the edges and simulating the FST over a tropical semi-ring. This will return the most appropriate normalization.

Latency: When all the 500 test cases are run the latency of the model is found to be 2.19 s. Therefore, the average latency of a test case is around 438 µs.

IV. Technology Used to Write Grammars for Text Normalization

The main technology that has been used for this project is *OpenGrm Thrax Grammar Development* *(Gorman, 2020)* toolset. Using this we can write regular expressions / grammars which are in turn converted to the corresponding WFST using the *openFst* *(Tom Bagby, Dan Bikel, 2020)* library. Thrax library also allows us to run the code and test it. This library reduces our effort as it prevents us from writing code that would generate the WFST directly. Thrax also enables us to design acceptors. In case of acceptors if a string is accepted, the acceptor return whatever the original string was, while transducers return the resultant output string on an input.

The OpenFst is an open source library that has well optimized functions for constructing, optimizing, composing and searching WFSTs.

Openfst and thrax are tools that work on unix platforms only. Windows 10 users however can use the Ubuntu app (*Ubuntu Terminal for Windows 10*) for windows in order to use this libraries.

Our text normalizer has been coded with the above two technologies. One can refer to this *(Sproat, 2012)* open source grammar for getting better insights of how to use these libraries.

V. Results

Metric Used: $$Accuracy\ (in\ \%) = \frac{\#\ strings\ correctly\ normalised}{\#test\ cases} x100\%. \quad (11)$$

All the domains were separately tested and their accuracy was noted. Table 3, shows the accuracy of each domain on the test set.

Table 3: Shows the domains wise accuracy obtained by the model, when model of each domain is tested separately

Srl. No.	Domain	#test cases	#correctly normalized	Accuracy (in %)
1.	Abbreviations	50	50	100.00
2.	Address	50	50	100.00
3.	Currency	50	50	100.00
4.	Dates	50	50	100.00
5.	Web Related	50	50	100.00
6.	Maps	50	50	100.00
7.	Measurements	50	50	100.00
8.	Numbers	55	55	100.00
9.	Scores	50	50	100.00
10.	Time	45	45	100.00
Overall Accuracy Obtained:		**500**	**500**	100.00

Now the same test cases were tested using the integrated model and their accuracies were noted. Table 4, shows the result of this experiment.

Table 4: Shows the domains wise accuracy obtained by the model, when the global model was tested

Srl. No.	Domain	#test cases	#correctly normalized	Accuracy (in %)
1.	Abbreviations	50	50	100.00
2.	Address	50	48	96.00
3.	Currency	50	50	100.00
4.	Dates	50	49	98.00
5.	Web Related	50	50	100.00
6.	Maps	50	50	100.00
7.	Measurements	50	50	100.00
8.	Numbers	55	55	100.00
9.	Scores	50	49	98.00
10.	Time	45	44	97.78
Overall Accuracy Obtained:		**500**	**495**	99.00

One can observe that there is a slight dip in the accuracy when the same test set was tested using the integrated model. This is due to the ambiguities that arise when two or more paths accept the same string. This confuses the model and reduces accuracy. If the weights were not adjusted as described in section III the accuracy would have decreased drastically as more inputs would be ambiguously normalized. Thus a weighted FST is extremely essential while designing a Text Normalization model.

VI. Conclusion and Future Scope

This paper points out the importance of text normalization in most pipelines for generating an ASR model. It also describes why machine and deep learning tools is not a good choice for Text Normalization presently. It briefly outlines automata and transducer theory required for creating ASR models like that for Text normalization. It also shows why WFST defined over a tropical semi-ring is the best choice for our task. Although the results obtained when testing the domains individually is impressive (100.00%), the performance degrades a bit (99.00%) when the integrated transducer is used for normalization. This is mainly because the WFST is not that capable to fit highly non-linear functions.

Therefore, a future scope of this work could be to learn the weights of the transducer using supervised learning. Trying end-to-end deep learning can also be an option. If enough labelled data is available for each domain then, deep neural networks for sequence data (like the RNN and the LSTM) could be a very good choice for text normalization. These may even yield much better results. Apart from using ML techniques we can also try to concentrate on separating words (present in a dictionary) without spaces in between, like those in URLs. An example can be normalizing “www.getprogrammingtutorials.com”, to “www dot get programming tutorial dot com”. Therefore research in ‘text normalization’ is still rife with opportunities and there is a lot of scope for improvement.

References

Aho, A. V, & Ullman, J. D. (1977). *Principles of Compiler Design* (p. x + 604).

Gorman, K. (2020). *OpenGrm Thrax Grammar Development Tools*. http://www.openfst.org/twiki/bin/view/GRM/Thrax

Hao, L. Z. (2015). *WFSTs over semirings* (pp. 34–77). http://u.cs.biu.ac.il/~jkeshet/teaching/spr2017/spr2017_lecture7.pdf

Hui, J. (2019). *Speech Recognition — Weighted Finite-State Transducers (WFST)*. https://medium.com/@jonathan_hui/speech-recognition-weighted-finite-state-transducers-wfst-a4ece08a89b7

Mealy Machines. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Mealy_machine

Mohri, M. (n.d.). *Weighted Finite-State Transducers Algorithms*. *15*. https://www.gavo.t.u-tokyo.ac.jp/~novakj/wfst-algorithms.pdf

Mohri, M. (2009). *Weighted Automata Algorithms*. 213–254. https://doi.org/10.1007/978-3-642-01492-5_6

Moore Machines. (n.d.). In *Wikipedia*. https://en.wikipedia.org/wiki/Moore_machine

Samudravijaya, K. (n.d.). *Automatic speech recognition*. http://www.iitg.ac.in/samudravijaya/tutorials/asrTutorial.pdf

Sproat, R. (2012). *Grammars and software developed as part of a text normalization class taught at the Center for Spoken Language Understanding, Fall 2011.* http://www.openfst.org/twiki/pub/Contrib/ThraxContrib/export.tgz

Sproat, R., Black, A. W., Chen, S., Kumar, S., Ostendorf, M., & Richards, C. (2001). Normalization of non-standard words. *Computer Speech & Language*, *15*(3), 287–333. https://doi.org/https://doi.org/10.1006/csla.2001.0169

Tom Bagby, Dan Bikel, K. G. (2020). *OpenFst Library*. http://www.openfst.org/twiki/bin/view/FST/WebHome

Ubuntu Terminal for Windows 10. (n.d.). https://ubuntu.com/tutorials/tutorial-ubuntu-on-windows#1-overview

Veliz, C. M., De Clercq, O., & Hoste, V. (2019). Comparing MT approaches for text normalization. *International Conference Recent Advances in Natural Language Processing, RANLP*, *2019-Septe*, 740–749. https://doi.org/10.26615/978-954-452-056-4_086

Viterbi. (2019). *13 Symbolic MT 2 : Weighted Finite State Transducers*. *117*, 83–92. http://www.phontron.com/class/mtandseq2seq2019/assets/slides/mt-fall2019.chapter13.pdf