Lambert 2021: Grammar Interpretations and Learning TSL Online

Kenneth Hanson

LIN 629, Fall 2022

Overview

- The tier-based strictly local (TSL) languages model some long-distance generalizations in natural languages (Heinz et al. 2011).
  - Factor-based, like strictly local (SL) and strictly piecewise (SP)
  - Factors based on relativized adjacency
  - Two parameters: k (factor size) and T (tier-alphabet)
- TSL patterns are easy to learn if you know T, hard if you don't.
- Two algorithms for learning TSL (including T) already exist:
  - Jardine and Heinz (2016) — 2-TSL only
  - Jardine and McMullin (2017) — arbitrary k, batch only
- This paper: a new **online** algorithm, arbitrary k.

Roadmap

- Background on:
  - SL and TSL
  - Online Gold learning
  - String extension learning
- The algorithm
  - Salience
  - Augmented subsequences
  - Pointwise string extension learning
  - Optimization
- Interacting constraints
- Other subregular classes

SL and TSL

k-SL (strictly *k*-local) — set of forbidden substrings of length *k*

Example

No *aa* or *bb*

Σ = {a,b,c,d} G = {aa, bb}

ex. valid words: abaca, abada

ex. invalid words: aaca, bbb

SL and TSL (2)

k-TSL (tier-based strictly *k*-local) — forbidden substrings of length *k* on a tier of salient symbols (relativized adjacency)

Example

No *aa* or *bb*, ignoring *d*

Σ = {a,b,c,d} T = {a,b,c} G = {aa, bb}

ex. valid words: abaca, abab

ex. invalid words: ada, bdb, adda, bddb

- Banning *aa* effectively bans ad^*a

Online Gold learning

Using an *online* variant of identification in the limit from positive text (Gold 1967).

- $L \subseteq \Sigma^*$ is the target language (stringset)
- $L_\circledcirc$ is L plus an adjoined element $\circledcirc$ that represents the lack of any string
- $t : \mathbb{N} \rightarrow L_\circledcirc$ is the **text**, an infinite sequence of strings drawn from L
  - Must contain each word in L at least once
  - May at some points return $\circledcirc$
- t_n is the word in t at time point n

Online Gold learning (2)

- $\mathbb{G}$ is a set of grammars
- $\mathcal{L} : \mathbb{G} \to \mathcal{P}(\Sigma^*)$ is the function from a grammar to the corresponding language
- Two grammars G_1 and G_2 are **equivalent** ($G_1 \equiv G_2$) iff they are extensionally equal, i.e. $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.
- $\varphi : \mathbb{G} \times L_{\circledcirc} \to \mathbb{G}$ is an **online learning function**, taking the previous guess and a single data point, and returning a new guess

Online Gold learning (3)

- **Convergence**
  Given a text t and a learner φ, φ **converges** on t iff there exists some $i \in \mathbb{N}$ and some grammar G such that for all $j \geq i$, it holds that $\varphi(t_j) \equiv G$.
- **Identification in the limit of a language**
  If given any text t for a stringset L, φ converges on t to a grammar G such that $\mathcal{L}(G) = L$, then we say that φ identifies L in the limit
- **Identification in the limit of a class of languages**
  For any class of stringsets $\mathbb{L} \subseteq P(\Sigma^*)$, we say that φ identifies $\mathbb{L}$ in the limit iff it identifies L in the limit for all $L \in \mathbb{L}$.

String extension learning (Heinz 2010) with Gold

For any class of languages based on sets of factors, we can learn languages in that class by memorizing the factors that we observe.

$f : \Sigma^* \to \mathcal{P}(\Sigma^k)$ is a function that maps a string to the factors it contains. For TSL, these are factors on the tier.

We can easily plug this into Gold's paradigm:

$$\varphi_f(G, w) \triangleq \begin{cases} G & \text{if } w = \circledcirc \\ G \cup f(w) & \text{otherwise} \end{cases}$$

If T is fixed, then learning k-TSL is just like learning k-SL. But we want to learn T. So G here includes both T and the forbidden factors on the tier.

Salience

T contains only the salient symbols — those not ignored on the tier.

- Any symbol not in *T* is freely insertable and deletable – adding it to a string results in a string in L, and removing it also results in a string in L.

We can test for salience using substring factors (Jardine and McMullin 2017).

- A symbol *x* is freely insertable iff for each attested factor of width $\leq$ k, inserting *x* at each possible point results in an attested factor one symbol wider.
- A symbol *x* is freely deletable iff for each attested factor of width k + 1 that contains *x*, the removal of each instance of x in turn results in an attested factor one symbol narrower.

Salience – formalization

Let $\mathcal{F}_k$ be the factors of length $\leq$ k, and $\mathcal{F}_{k+1}$ the factors of length $\leq$ k + 1.

Define for each symbol $\sigma \in \Sigma$ two sets:

- $\sigma_\oplus$ = set of factors obtained by adding a single instance of σ to $\mathcal{F}_k$
- $\sigma_\ominus$ = set of factors obtained by deleting a single instance of σ from $\mathcal{F}_{k+1}$.

Then the set of salient symbols is $T = \{\sigma : \sigma_\oplus \not\subseteq \mathcal{F}_{k+1} \vee \sigma_\ominus \not\subseteq \mathcal{F}_k\}$.

Salience – efficiency

This portion of the algorithm is efficient in terms of the size of the input, *n*. The space complexity is exponential w.r.t. $|\Sigma|$.

- Space complexity is $\mathcal{O}(|\Sigma|^{k+1})$
- Time complexity is $\mathcal{O}(nk \log |\Sigma|)$

Batch learning for TSL

Jardine and McMullin (2017) give a batch-learning algorithm:

- Find the set of salient symbols.
- Erase the non-salient symbols from the input.
- Reprocess the input with an SL learner.

This requires all input to be retained. We want to avoid this for online learning.

Towards online learning

Factors on a tier use relativized adjacency.

This means that the factors are subsequences with a special property: if a symbol is included once in the subsequence, it can not be excluded later in the same subsequence.

Example

string: lokalis

valid 3-factor: lkl

invalid 3-factor: lki

Augmented subsequences

To learn the tier factors, we only need to keep track of valid subsequences and their intervening symbols.

Example Augmented subsequences for “cabacba”

Factor	Valid Intervener Sets	Invalid Intervener Sets
aa	{b, bc}	{abc}
ab	{∅, c}	{abc}
ac	{∅}	{ab}
ba	{∅}	{abc}
bb	{ac}	{}
bc	{a}	{}
ca	{∅, b}	{ab, abc}
cb	{a, ∅}	{abc}
cc	{ab}	{}

Augmented subsequences – efficiency

Keeping track of all augmented subsequences is beyond exponential in space w.r.t. $|\Sigma|$ and k.

We can save space by taking advantage of the following fact: if a factor is attested with intervener set I, then it can also be assumed to be attestable for any superset of I for which it remains valid.

The space needed is still exponential in the worst case w.r.t. $|\Sigma|$, but will often be much smaller, since for many symbols we only need to store a single set ($\varnothing$).

Augmented subsequences – revisited

Example Augmented subsequences for “cabacba”, minus redundant and invalid items

Factor	Valid Intervener Sets	Invalid Intervener Sets
aa	{b, ~~bc~~}	{~~abc~~}
ab	{∅, ~~c~~}	{~~abc~~}
ac	{∅}	{~~ab~~}
ba	{∅}	{~~abc~~}
bb	{ac}	{}
bc	{a}	{}
ca	{∅, ~~b~~}	{~~ab~~, ~~abc~~}
cb	{~~a~~, ∅}	{~~abc~~}
cc	{ab}	{}

Pointwise Extension Learning

To learn TSL, we can combine the two sources of information discussed above.

- The substrings of length $\leq k + 1$ allow us to determine which symbols are salient.
- The augmented subsequences of length $\leq k$ allow us to select a set of permitted factors once salience has been determined.

So, we can do something very much like ordinary string extension learning, just with two grammars at once.

The result can then be used directly as an acceptor: a string is accepted iff it has only permitted substrings and each of its valid augmented subsequences is attested or entailed by something that is attested.

Optimization

Actually, just the first grammar (substrings of length $\leq$ k + 1) has all the information we need.

- Let $s : \mathbb{G} \rightarrow \mathcal{P}(\Sigma)$ be the function that finds the tier (salient symbols).
- Let $\pi_T(w)$ be the tier-projection function (erasing non-salient symbols).
- Then $\mathcal{L}(G) \triangleq \{w \in \Sigma^* : f(\pi_{s(G)}(w)) \subseteq G\}$

In this case, learning the grammar requires no more time and space beyond that needed for determining salience. The time complexity of interpreting the grammar as k-TSL rather than $(k{+}1)$-SL is deferred until later.

Example: Latin liquid dissimilation

Sequential *r*'s and *l*'s are banned unless a non-coronal consonant intervenes.

This pattern is 2-TSL.

Σ = {a,l,k,r} T = {l,k,r} G = {ll, rr}

So we need to collect a 3-SL grammar.

Example: Latin liquid dissimilation

n	t_n	Factors
0	akkalkak	{akk, kka, kal, alk, lka, kak}
1	klark	{kla, lar, ark}
2	kralk	{kra, ral, alk}
3	karlakalra	{kar, arl, rla, lak, aka, kal, alr, lra}
4	akrala	{akr, kra, ral, ala}
5	aklara	{akl, kla, lar, ara}
6	rakklarkka	{rak, akk, kkl, kla, ark, rkk, kka}
7	arkralkla	{ark, rkr, kra, ral, alk, lkl, kla}
8	laarlraalr	{laa, aar, arl, rlr, lra, raa, aal, alr}
9	kaaakkrka	{kaa, aaa, aak, akk, kkr, krk, rka}
10	klkkklrk	{klk, lkk, kkk, kkl, klr, lrk}
11	krlkrkl	{krl, rlk, lkr, krk, rkl}
12	alrla	{alr, lrl, rla}

Note: Factors already seen are grayed out.

Interacting constraints

Many of the words in the previous example are phonologically implausible.

A more realistic language would combine SL constraints for local (syllable structure) constraints with TSL constraints for liquid dissimilation.

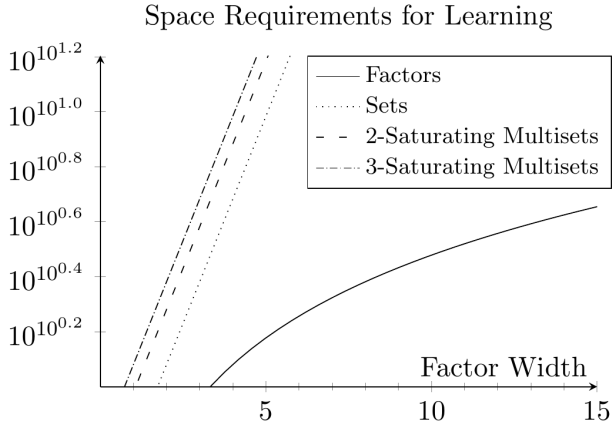
But algorithms of this sort cannot generally handle conjoined constraints. Trying to learn TSL + SL results in an SL approximation of the grammar.

On the other hand, algorithms that attempt to learn a grammar for every possible tier require much more space, exponential in the size of the alphabet.

Aside: higher subregular classes

This method can also be used for the tier-based equivalents of higher subregular classes. We just need to keep track of factors of size $k + 1$, in addition to sets (LT) or multisets (LTT) of k-factors.

Again, space is the problem — tracking factor (multi)-sets is super-exponential.



Conclusion

We can learn both the tiers and constraints for *k*-TSL languages using a variant of string extension learning.

There are some practical limitations:

- Worst case space complexity is exponential, even if typical complexity might not be.
- All possible (k+1) factors must be attested — this is not realistic.
- Learning is not possible under interaction with SL constraints.

References

Gold, E. Mark (1967). “Language identification in the limit”. In: *Information and control* 10.5, pp. 447–474.

Heinz, Jeffrey (July 2010). “String Extension Learning”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 897–906.

Heinz, Jeffrey et al. (2011). “Tier-based strictly local constraints for phonology”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human language technologies*, pp. 58–64.

Jardine, Adam and Jeffrey Heinz (2016). “Learning Tier-based Strictly 2-Local Languages”. In: *Transactions of the Association for Computational Linguistics* 4, pp. 87–98. DOI: 10.1162/tacl_a_00085. URL: https://aclanthology.org/Q16-1007.

Jardine, Adam and Kevin McMullin (2017). “Efficient Learning of Tier-Based Strictly k-Local Languages”. In: *Language and Automata Theory and Applications*. Ed. by Frank Drewes et al. Cham: Springer International Publishing, pp. 64–76. ISBN: 978-3-319-53733-7.