Yale University CPSC 683
Stability in Machine Learning: Generalization, Privacy & Replicability

Lecture 11: Language Identification and Generation

Instructor: Alkis Kalavasis, alkis.kalavasis@yale.edu

In Lecture 11, we present the Gold-Angluin online model for language identification in the limit (Gold, 1967; Angluin, 1980). Then we will discuss the online model for language generation in the limit by Kleinberg and Mullainathan (2024).

1 Countable Sets and Enumerations

Countable sets will be very important for this Lecture. So let us recall some basic examples.

1. Finite sets, naturals, integers, rationals are countable
2. Given a finite alphabet (like the letters of the English alphabet), the set of all finite-length words that can be formed using that alphabet is countable.
3. The set of all finite subsets of natural numbers: Each finite subset can be represented by a binary sequence, making this set countable.
4. (Assuming the axiom of countable choice) The union of countably many countable sets is countable.

Let us now recall some standard uncountable sets.

1. Cantor's Theorem implies that the powerset of $\mathbb{N}$ is uncountable.
2. The set of real numbers is uncountable.

Given a countable set S, an *enumeration* of S is a listing of all of its elements with the property that each $x \in S$ appears in a fixed finite position. For instance, the standard ordering of $\mathbb{N}$ is an enumeration since any x appears in the x-th position but the list $1, 3, 5, 7, ..., 2, 4, 6, ...$ is not an enumeration.

2 Language Identification in the Limit

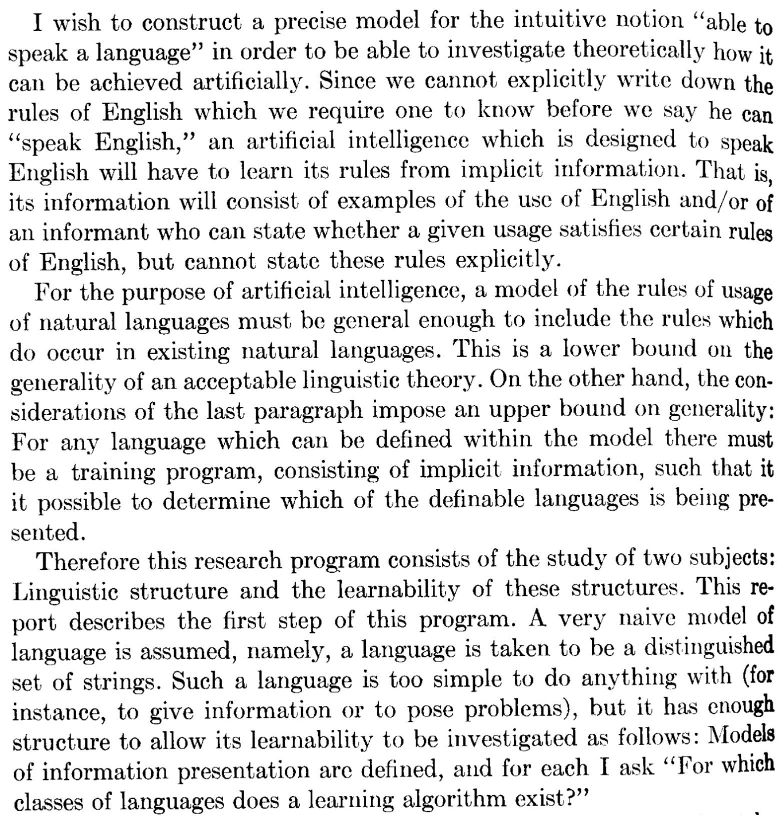
The problem of language identification in the limit from positive examples was introduced by Gold (1967) and further studied by Angluin (1979, 1980). The setting is specified by a countable collection of languages $\mathcal{L} = \{L_1, L_2, \dots\}$. For a fixed collection $\mathcal{L}$, an adversary and an identifier play the following game: The adversary chooses a language K from $\mathcal{L}$ without revealing it to the identifier, and it begins *enumerating* the strings of K (potentially with repetitions) $x_1, x_2, \dots$ over a sequence of time steps $t = 1, 2, 3, \dots$. The adversary can repeat strings in its enumeration, but the crucial point is that for every string $x \in K$, there must be at least one finite time step t at which it appears.

At each time t, the identification algorithm I, given the previous examples $x_1, x_2, \dots, x_t$, outputs an index i_t that corresponds to its guess for the true language K.

Definition 1: Language Identification in the Limit (Gold, 1967)

Fix some language K from the language collection $\mathcal{L}$. The identification algorithm I identifies K in the limit if there is some $t^* \in \mathbb{N}$ such that for all steps $t > t^*$, the identifier's guess i_t satisfies $i_t = i_{t-1}$

and $L_{i_t} = K$. The language collection $\mathcal{L}$ is identifiable in the limit if there is an identifier that identifies in the limit any $K \in \mathcal{L}$, for any enumeration of K.



I wish to construct a precise model for the intuitive notion "able to speak a language" in order to be able to investigate theoretically how it can be achieved artificially. Since we cannot explicitly write down the rules of English which we require one to know before we say he can "speak English," an artificial intelligence which is designed to speak English will have to learn its rules from implicit information. That is, its information will consist of examples of the use of English and/or of an informant who can state whether a given usage satisfies certain rules of English, but cannot state these rules explicitly.

For the purpose of artificial intelligence, a model of the rules of usage of natural languages must be general enough to include the rules which do occur in existing natural languages. This is a lower bound on the generality of an acceptable linguistic theory. On the other hand, the considerations of the last paragraph impose an upper bound on generality: For any language which can be defined within the model there must be a training program, consisting of implicit information, such that it it possible to determine which of the definable languages is being presented.

Therefore this research program consists of the study of two subjects: Linguistic structure and the learnability of these structures. This report describes the first step of this program. A very naive model of language is assumed, namely, a language is taken to be a distinguished set of strings. Such a language is too simple to do anything with (for instance, to give information or to pose problems), but it has enough structure to allow its learnability to be investigated as follows: Models of information presentation are defined, and for each I ask "For which classes of languages does a learning algorithm exist?"

Figure 1: Taken from Gold (1967). Gold's motivation for the model.

Gold (1967) showed that collections of finite cardinality languages, i.e., each language in the collection $\mathcal{L}$ is finite, can be identified in the limit from positive examples. This is true since in the limit, one will see all the elements of the target (finite) language, at which point it can be identified. The identification algorithm is the following: at time t, guess L to consist solely of the elements that have occurred in the sequence. Since L is finite, there will be a finite time after which all elements of L will have been revealed, so after that the algorithm will have identified the target.

A super-finite collection of languages denotes any collection which contains all languages of finite cardinality and at least one of infinite cardinality. Gold showed that super-finite collections of languages cannot be identified in the limit from positive examples. Further, he showed that negative examples help: any super-finite collection can be identified in the limit using positive and negative examples[1] (the idea is simple: keep guessing the infinite language until seeing a negative example; then it reduces to the finite case).

Theorem 1: (Gold, 1967)

Let $\mathcal{L} = \{L_\infty, L_1, L_2, \dots\}$ be the language collection with $L_1 \subset L_2 \subset \cdots \subset L_\infty = \cup_{i\geq 1} L_i$ and for each i, $|L_i| < \infty$. Then, there is no algorithm that identifies $\mathcal{L}$ in the limit from positive examples. Moreover, this collection can be identified in the limit when the algorithm has access to both positive and negative examples.

[1]This means that the adversary presents an enumeration of the whole domain $\mathcal{X}$, with a label indicating whether the example is in the target language.

For instance, let $L_i = [i]$ and $L_\infty = \mathbb{N}$.

Exercise 1

Show that the above language collection is not identifiable in the limit.

The above result already shows a separation in terms of identification between observing only positive examples and observing positive *and* negative examples in Gold's model. Moreover, it raises the question of whether there exist non-trivial collections of languages identifiable in the limit from positive examples. In that direction, Angluin (1979) studied pattern languages (whose definition is not important for our work) and showed that for that collection identification in the limit is possible only with positive examples.

Before proceeding with the characterization result, there are two important aspects that we have not paid extensive attention:

1. How does the algorithms access the collection $\mathcal{L}$? Usually, the only type of access the algorithm has is *membership oracle access*, i.e., given element x and index i, the oracle returns $1\{x \in L_i\}$. For instance, the algorithm for the finite cardinality languages works with just that type of access. However, it could e.g., be the case that the algorithm uses some other type of oracle, e.g., a *subset oracle*, where given indices i, j, the oracle returns $1\{L_i \subseteq L_j\}$.
2. Can the algorithm be collection-dependent? For instance, if the collection has some particular property, can the algorithm make use of it? As we will see, e.g., the algorithm of Kleinberg and Mullainathan (2024) is agnostic to the collection; it only uses membership oracles.

Let us consider another example. Let $\mathcal{L}$ be some *finite* collection of languages. Is this identifiable in the limit? Let $\mathcal{L} = \{L_1, L_2, ..., K, ..., L_T\}$ be the finite collection and let K be the unknown true language. Which languages can we eventually exclude easily?

For any $i \in [T]$ with $K \setminus L_i \neq \emptyset$, it means that eventually the algorithm will observe a point in K that is not in L_i (can check it with membership oracle); hence, in finite time, this language will become inconsistent.

However, we are not done: there is some possibility that $K \subset L_i$. In that case, any algorithm that starts predicting L_i will never exclude it since any element of K also belongs to L_i. If the algorithm could output from the minimal consistent, then we would be fine. For instance, if the algorithm is collection-dependent, then the pairwise subset relations can be encoded in the algorithm.

The next question is whether one can get a *characterization* of the language collections that can be identified from positive examples. Angluin (1980) resolved this problem.

Definition 2: Angluin's Condition (Angluin, 1980)

Fix a language collection $\mathcal{L} = \{L_1, L_2, \ldots\}$. Suppose there is a membership oracle which, given a string x and index i, answers $1\{x \in L_i\}$. The collection $\mathcal{L}$ is said to satisfy Angluin's condition if there is a *tell-tale* oracle that given an index i enumerates a set of *finite* strings T_i such that

$$T_i \subseteq L_i \text{ and for all } j \geq 1, \text{ if } T_i \subseteq L_j \text{ then } L_j \text{ is not a proper subset of } L_i.$$

The difficulty in trying to identify a language from positive examples is the problem of *over-generalization*. If while seeing positive examples the algorithm specifies a language that is a proper superset of the true answer K, then by only seeing positive examples it will never see a counterexample to that language. This would be avoided with positive and negative examples. Angluin's condition essentially ensures this over-generalization problem can be avoided by from just positive examples (without the help of negative examples).

Before proceeding to Angluin's result, we stress one important point: inspecting Angluin's definition, we can see that it requires access to a procedure that *finds* this set of strings T_i. This oracle is called a *tell-tale* oracle and is quite crucial for Angluin's algorithm to work.

Definition 2 led to the following characterization.

Theorem 2: (Angluin, 1980)

A countable language collection $\mathcal{L}$ is identifiable in the limit if and only if it satisfies Angluin's criterion.

Let us see why Angluin's condition is sufficient for identification in the limit. Assume that $\mathcal{L}$ satisfies Angluin's criterion.

For any $t \geq 1$:

1. Get x_t and let S_t be the training set observed until time t.
2. Find the least positive integer $g \leq t$ (if any) such that
   - $S_t \subseteq L_g$
   - $T_g^{(t)} \subseteq S_t$, where $T_g^{(t)}$ is the set of strings produced in the first t steps of the enumeration of T_g.
3. If no such index g exists, output a random index and go to the next iteration.
4. Otherwise output g.

To see why this works, let k be an index and L_k the true language. Let $s_1, s_2, ...$ be an enumeration of L_k. Let m be the index of the first appearance of L_k in the enumeration of $\mathcal{L}$. For each $0 < i < m$, either

1. $L_m \setminus L_i \neq \emptyset$. This means that there is an element in the enumeration that does not belong to L_i. Hence, there is a finite time that L_i will be rejected. Say n_i that time, which is associated with L_i.
2. If the above is not true, then $L_m \subset L_i$. In general, this case is hard when only positive examples are available, because L_i cannot be rejected. But, here is where Angluin's condition comes in. Let us pick n_i sufficiently large such that $T_i^{(n_i)} = T_i$ for all $t \geq n_i$.

Let n_m be sufficiently large such that $T_m \subseteq S_t$ for all $t \geq n_m$. Let $N = \max\{n_1, n_2, ..., n_m, m\}$. Now for all $t \geq N$, we have $m \leq t, S_t \subseteq L_m$, and $T_m \subseteq S_t$.

Now if for $i < m$ and L_i does not belong to Case 1, then $L_m \subset L_i$, then T_i cannot be a subset of L_m, and so $T_i^{(t)}$ is not a subset of L_m. Hence, the tell-tale for L_i serves a counterexample for L_i. This means that the index converges to m and stabilizes at that point.

Finally, let us consider the case of language identification with both positive and negative examples, i.e., when the adversary provides an enumeration of the whole domain $\mathcal{X}$ and every example has a label indicating whether it is in the true language K. We mention that focusing on algorithms equipped with membership oracle, the following result appears in Gold (1967).

Theorem 3: (Gold, 1967)

Any countable language collection is identifiable in the limit from positive and negative examples.

To see how the algorithm works, let $\mathcal{L} = \{L_1, L_2, \ldots\}$ and denote by L_z the smallest indexed language in $\mathcal{L}$ for which $L_z = K$. The algorithm observes an enumeration of the form $(x_t, y_t) \in \times\{0, 1\}$ for $t \geq 1$. Recall this means that $1\{x_t \in K\} = y_t$. The algorithm works as follows: in every timestep $t \in \mathbb{N}$, it predicts the lowest index of a consistent language, i.e., the smallest $j \in \mathbb{N}$ for which $1\{x_\tau \in L_j\} = y_\tau$ for all $\tau \leq t$. Consider two cases: if $z = 1$, then the algorithm will never predict any language $L_{z'}, z' \geq 2$, so it will be correct from the first step. If $z > 1$, then for all $L_{z'}, z' < z$, that come before L_z in the enumeration of $\mathcal{L}$, there is a finite time $t_{z'}$ when the example $(x_{t_{z'}}, y_{t_{z'}})$ contradicts the language $L_{z'}$.

3 Language Generation in the Limit

We now move to language generation in the limit from positive examples, introduced by Kleinberg and Mullainathan (2024). The setup is exactly the same as in the Gold-Angluin model (the adversary provides an enumeration of K), but now the goal of the learner is to *generate unseen examples* from K instead of identifying the index of K. Their formal definition is the following.

Definition 3: Language Generation in the Limit (Kleinberg and Mullainathan, 2024)

Fix some language K from the collection $\mathcal{L} = \{L_1, L_2, \ldots\}$ and a generating algorithm $\mathcal{G}$. At each step t, let $S_t \subseteq K$ be the set of all strings that the algorithm $\mathcal{G}$ has seen so far. $\mathcal{G}$ must output a string $x_t \notin S_t$ (its guess for an unseen string in K). The algorithm $\mathcal{G}$ consistently generates from K in the limit if, for all enumerations of K, there is some $t^* \in \mathbb{N}$ such that for all steps $t \geq t^*$, the algorithm's guess a_t belongs to $K \setminus S_t$. The collection $\mathcal{L}$ allows for consistent generation in the limit if there is an algorithm $\mathcal{G}$ that, for any choice of the target language $K \in \mathcal{L}$, it consistently generates from K in the limit.

Definition 3 straightforwardly generalizes to randomized algorithms; consider the same setup as before except that now the output string a_t may be randomized. The definition of generation is also the same except that instead of requiring $a_t \in K \setminus S_t$ one requires that the support A_t of the distribution from which a_t is sampled is non-empty and satisfies $A_t \subseteq K \setminus S_t$.

Insight 1

One way to think of the elements of some language L is as valid *sentences* and not just words. In this way, the above model aims to abstract the whole training process of LLMs. In the real world, first a huge collection of texts (say Wikipedia articles) are collected; these correspond to the elements of the true language. Then in practice, the way that pre-training is applied is via next-token prediction: each article is separated into tokens, and the model is trained to predict the next token given the previous one. In the generation phase, where the weights are frozen, the model also generates in a next-token based manner: given an input prompt, it samples the next token and then recursively generates the whole sentence (which is another element from the language in our formulation). In this way, the above model abstracts all these steps and simply gets training samples (which are sentences from a language) and aims to learn how to generate new sentences from that language.

Observe that language generation requires that the algorithm's outputs are *consistent* with K (in the limit), but allows the algorithm to not generate certain strings from K. For instance, if K is the set of all strings, then the algorithm that always outputs even length strings (not in S_t), generates from K in the limit but also misses infinitely many strings in K (namely, all strings of odd length). Consistency is clearly a desirable notion: without consistency, algorithms may keep outputting strings outside the target language K which, when K is the set of all meaningful and true strings, inevitably leads to hallucinations.

A trivially consistent generator is one that outputs data already seen in the training set. As we already mentioned, we count such outputs as mistakes. This form of predicting unseen positive examples makes the task of generation interesting. At first sight, it seems that there is an easy strategy that achieves generation in the limit: given an enumeration of all hypotheses $L_1, L_2, \ldots$, we sequentially generate from L_i ($i = 1, 2, \ldots$) until it becomes inconsistent with the sample S_n; then we move to L_{i+1}. This strategy seems natural for generation because we know that there is some index k such that the true language $K = L_k$. This idea has a fundamental issue, already reported by Kleinberg and Mullainathan (2024): if there exists an index i such that $i < k$ and $L_k \subsetneq L_i$, then the generator will get stuck at L_i and never update.

3.1 Pick the First Consistent

In thinking about approaches to generation in the limit, there is a natural strategy that at first seems to solve the problem directly, but in fact does not work. Its failure is useful to discuss, since it motivates the more involved solution that follows.

The strategy is to move through the list of languages $\mathcal{L} = \{L_1, L_2, L_3, \ldots\}$ in order, treating each language L_i as a hypothesis for K until the sample S_t proves otherwise. That is, we start with L_1, and we generate strings from $L_1 - S_t$ until we encounter (if ever) a step t in which S_t is not a subset of L_1. At this point we know that L_1 cannot be the true language K, and so we continue the process with L_2.

The nice idea that underpins this strategy is that the true language K is equal to L_z for some index z. So if our process were to reach L_z at some step t_*, it would never move on from L_z, and so we would be generating from $K - S_t$ for all $t \geq t_*$.

Unfortunately, there is a deep issue with this approach: there may be a language $L_i \in \mathcal{L}$ with the property that L_i comes before L_z and L_i properly contains L_z (that is, $i < z$, and $L_z \subsetneq L_i$). In this case, our procedure would stop at the first such L_i forever: since it is only ever shown samples in S_t that come from the language L_z, and since $L_z \subseteq L_i$, it would never encounter a string in S_t that didn't belong to L_i, and so it we would never move on to L_{i+1}. And when this procedure generated from $L_i - S_t$, there is no guarantee that it would choose strings from L_z.

Note that if this approach worked, we could also identify in the limit. This is exactly where Angluin's criterion was used.

The important observation is that if the algorithm is maintaining hypotheses for the true language K over time, it can provably *never know whether its current hypothesis is correct*; instead, it must be always moving further down the collection of languages, potentially considering languages that are not K, but in such a way that it is eventually always generating from $K - S_t$.

3.2 The Algorithm

A non-trivial solution to this problem was given by Kleinberg and Mullainathan (2024). They show that all countable sets of languages in countable domains allow for generation in the limit from positive examples; this is in stark contrast with identification in the limit from positive examples.

Theorem 4: Theorem 1 in Kleinberg and Mullainathan (2024)

There is an algorithm with the property that for any countable collection of languages $\mathcal{L} = \{L_1, L_2, \ldots\}$, any target language $K \in \mathcal{L}$, and any enumeration of one of these languages K, the algorithm generates from K in the limit with positive examples.

We say that a language L_i is consistent with the sample S_t at time t if S_t is contained in L_i.

Definition 4

A language L_n is critical at step t if L_n is consistent with S_t, and for every language $L_i \in \{L_1, ..., L_n\}$ that is consistent with S_t, we have $L_n \subseteq L_i$.

Note that at any time t, there is at least one critical language: the language with the smallest index that is consistent with S_t.

1. Let $\mathcal{L} = \{L_1, L_2, ...\}$ be the enumeration of the language class. For any step $t \geq 1$, we proceed as follows:
2. Let $\{L_1, ..., L_t\}$ be the first t languages.

3. Given the examples S_t until time t, identify all the critical languages in $\{L_1, ..., L_t\}$ at time t.
4. Among those, let L_{n_t} be the critical language with the largest index $(n_t \leq t)$.
5. Output some element from $L_{n_t} \setminus S_t$.

We now provide some intuition on how this algorithm works. Let $L_1, L_2, \ldots$ be an enumeration of the collection of languages and K be the true language. Let z be an index such that $L_z = K$. Now assume that we have two languages L_i and L_j with $L_i \subseteq L_j$ which are both consistent with S_t. Then, it is clear that the generating algorithm should prefer to generate from L_i rather than L_j: any $w \in L_i \setminus S_t$ satisfies $w \in L_j \setminus S_t$. This property inspired Kleinberg and Mullainathan (2024) to define the notion of a *critical language*, which we introduced before. Let $C_n = \{L_1, L_2, \ldots, L_n\}$.

Recall that a language L_n is critical at step t if L_n is consistent with S_t and for every $L_i \in C_n$ that is consistent with S_t, it must be $L_n \subseteq L_i$. There are some key properties upon which the generating algorithm is built:

- At any time, there is at least one language consistent with S_t, the true one $L_z = K$. Also, there is at least one critical language at any step t: for any t, the consistent language L_i with the lowest index i must be critical at step t, as it is the only consistent language in C_i.
- There exist times t for which L_z (which is K) is not critical. But eventually, L_z will become critical at some step and then remain critical forever after that. Also, any critical language coming after L_z must be a subset of L_z, thus it is safe to generate from it.
- Hence the algorithm, roughly speaking, keeps track of a list of critical languages and generates from the last one in the list; this is because, after some finite index, all the critical languages are subsets of L_z and, hence, it is safe to generate from any of them.

We remark that the above algorithm requires, apart from membership oracle access, access to an oracle that checks where $L_i \subseteq L_j$?. This is called a subset oracle. A natural question is whether a generation in the limit algorithm can be solely based on membership queries. The answer is affirmative. One can relax the notion of criticality by considering *projections* of languages:

Definition 5

Let t and m be positive integers. A language L_n is (t, m)-critical if L_n is consistent with S_t, and for every language $L_i \in \{L_1, ..., L_n\}$ such that L_i is consistent with S_t, we have $L_n[m] \subseteq L_i[m]$.

Here $L[m]$ are the first m elements in the enumeration of L. Note that in the above, as m increases, (t, m)-criticality tends to become the actual t-critical definition, since larger and larger projections of the languages are considered.

The idea is again to generate a new string from the highest-indexed critical language. The difficulty now is that we have to find a new string. This is also doable, see Kleinberg and Mullainathan (2024) for details.

References

Angluin, D. (1979). Finding patterns common to a set of strings (extended abstract). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, page 130–141, New York, NY, USA. Association for Computing Machinery.

Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135.

Gold, E. M. (1967). Language identification in the limit. *Information and control*, 10(5):447–474.

Kleinberg, J. and Mullainathan, S. (2024). Language generation in the limit. In *Advances in Neural Information Processing Systems*, volume 37.