Partially Distribution-Free Learning of Regular Languages from Positive Samples

Alexander Clark
ISSCO / TIM, University of Geneva
UNI-MAIL, Boulevard du Pont-d'Arve,
CH-1211 Genève 4, Switzerland
asc@aclark.demon.co.uk

Franck Thollard
EURISE, Université Jean Monnet,23,
Rue du Docteur Paul Michelon,
42023 Saint-Etienne Cédex 2, France
thollard@univ-st-etienne.fr

Abstract

Regular languages are widely used in NLP today in spite of their shortcomings. Efficient algorithms that can reliably learn these languages, and which must in realistic applications only use positive samples, are necessary. These languages are not learnable under traditional distribution free criteria. We claim that an appropriate learning framework is PAC learning where the distributions are constrained to be generated by a class of stochastic automata with support equal to the target concept. We discuss how this is related to other learning paradigms. We then present a simple learning algorithm for regular languages, and a self-contained proof that it learns according to this partially distribution free criterion.

1 Introduction

Regular languages, especially generated by deterministic finite state automata are widely used in Natural Language processing, for various different tasks (Mohri, 1997). Efficient learning algorithms, that have some guarantees of correctness, would clearly be useful. Existing algorithms for learning deterministic automata, such as (Carrasco and Oncina, 1994) have only guarantees of identification in the limit (Gold, 1967), generally considered not to be a good guide to practical utility. Unforunately the prospects for learning according to the more useful PAC-learning criterion are poor after the well known result of (Kearns and Valiant, 1989). Distribution-free learning criteria require algorithms to learn for every possible combination of concept and distribution. Under this worst-case analysis many simple concept classes are unlearnable. However in many situations it is more realistic to assume that there is some relationship between the concept and the distribution, and furthermore in general only positive examples will be available.

There are two ways of modelling this. The simplest is to study the learnability of distributions (Kearns et al., 1994; Ron et al., 1995). In this case the samples are drawn from the distribution that is being learned. The choice of error function then becomes critical – the most natural (and difficult) being the Kullback-Leibler divergence. This means that any successful algorithm must produce hypotheses that assign a non-zero probability to every string. If what we are interested in is learning the underlying non-probabilistic concept then these hypotheses will be useless. We have elsewhere proved (Clark and Thollard, 2004) a suitable result similar to that of (Ron et al., 1995), bounding the divergence, but that proof involves some more elaborate technical machinery.

The second way is to consider a traditional concept-learning problem, but to restrict the class of distributions to some set that only generates positive examples, and has some relation to the target concept. It is this latter possibility that we explore here.

In the particular case of learning languages we will have an instance space of Σ^* for some finite alphabet Σ, and we shall have a concept class, in this paper, corresponding to the class of all regular languages. In a distribution-free setting this is not learnable from positive and negative samples, nor *a fortiori* from positive samples alone. In our partially distribution-free framework however, we are able to prove learnability with an additional parameter in the sample complexity polynomial, that bounds a simple property of the distribution. We are able to present a simple stand alone proof for this well studied class of languages.

The rest of the paper is structured as follows. Section 2 argues for a modified version of PAC learning as being an appropriate learning framework for a range of NLP problems. After defining some notation in Section 3 we then define an algorithm that learns regular languages (Section 4) and then in Section 5 prove that it does so according to this modified PAC-learnability criterion. We conclude with a critical analysis of our results.

2 Appropriateness

Regular languages are widely used in a number of different applications drawn from numerous domains such as computational biology, robotics etc. In many of these areas, efficient learning algorithms are desirable but in each the exact requirements will be different since the sources of information, and the desired properties of the algorithms vary widely. We argue here that learning algorithms in NLP have certain special properties that make the particular learnability result we study here useful. The most important feature in our opinion is the necessity for learning from positive examples only. Negative examples in NLP are rarely available. Even in a binary classification problem, there will often be some overlap between the classes so that examples labelled with $-$ are not necessarily negative examples of the class labelled with $+$. For this reason alone we consider a traditional distribution-free PAC-learning framework to be wholly inappropriate. An essential part of the PAC-learning framework is a sort of symmetry between the positive and negative examples. Furthermore, there are a number of negative results which rule out distribution free learning of regular languages (Kearns et al., 1994).

A related problem is that in the sorts of learning situations that occur in practice in NLP problems, and also those such as first language acquisition that one wishes to model formally, the distribution of examples is dependent on the concept being learned. Thus if we are modelling the acquisition of the grammar of a language, the positive examples are the grammatical, or perhaps acceptable, sentences of the target language. The distribution of examples is clearly highly dependent on the particular language, simply as a matter of fact, in that the sentences in the sample are generated by people who have acquired the language.

It thus seems reasonable to require the distribution to be drawn from some limited class that depends on the target concept and generates only positive examples – i.e. where the support of the distribution is identical to the positive part of the target concept.

Our proposal is that when the class of languages is defined by some simple class of automata, we can consider only those distributions generated by the corresponding stochastic automata. The set of distributions is restricted and thus we call this partially distribution free. Thus when learning the class of regular languages, which are generated by deterministic finite-state automata, we select the class of distributions which are generated by PDFAs. Similarly, context free languages are normally defined by context-free grammmars which can be extended again to probabilistic or stochastic context free grammars.

Formally, for every class of languages, $\mathcal{L}$, defined by some formal device define a class of distributions, $\mathcal{D}$, defined by a stochastic variant of that device. Then for each language L, we select the set of distributions whose support is equal to the language:

$$D_L^+ = \{D \in \mathcal{D} : \forall s \in \Sigma^* \; s \in L \Leftrightarrow P_D(s) > 0\}$$

Samples are drawn from one of these distributions. There are two technical problems here: first, this doesn't penalise over-generalisation. Since the distribution is over positive examples, negative examples have zero weight – which would give a hypothesis of all strings zero error. We therefore need some penalty function over negative examples or alternatively require the hypothesis to be a subset of the target, and use a one-sided loss function as in Valiant's original paper (Valiant, 1984), which is what we do here. Secondly, this definition is too vague. The exact way in which you extend the "crisp" language to a stochastic one can have serious consequences. When dealing with regular languages, for example, though the class of languages defined by deterministic automata is the same as that defined by non-deterministic languages, the same is not true for their stochastic variants. Additionally, one can have exponential blow-ups in the number of states when determinizing automata. Similarly, with context free languages, (Abney et al., 1999) showed that converting between two parametrisations of models for stochastic context free languages are equivalent but that there are blow-ups in both directions.

It is interesting to compare this to the PAC-learning with simple distributions model (Denis, 2001). There, the class of distributions is limited to a single distribution derived from algorithmic complexity theory. There are a number of reasons why this is not appropriate.

First there is a computational issue: since Kolmogorov complexity is not computable, sampling from the distribution is not possible, though a lower bound on the probabilities can be defined. Secondly, there are very large constants in the sample complexity polynomial. Finally and most importantly, there is no reason to think that in the real world, samples will be drawn from this distribution; in some sense it is the easiest distribution to learn from since it dominates every other distribution up to a multiplicative factor.

We reject the identification in the limit paradigm introduced by (Gold, 1967) as unsuitable for three reasons. First it is only an asymptotic bound that says nothing about the performance of the algorithms on finite amounts of data; secondly because it must learn under all presentations of the data even when these are chosen by an adversary to make it hard to learn, and thirdly because it has no bounds on the amount of computation allowed.

An alternative way to conceive of this problem is to consider the task of learning distributions directly (Kearns et al., 1994), a task related to probability density estimation and language modelling, where the algorithm is given examples drawn from a distribution and must approximate the distribution closely according to some distance metric: usually the Kullback-Leibler divergence or the variational distance. We consider the choice between the distribution-learning analysis, and the analysis we present here to depend on what the underlying task or phenomena to be modelled is. If it is the probability of the event occurring, then the distribution modelling analysis is better. If on the other hand it concerns binary judgments about the membership of strings in some set then the analysis we present here is preferable.

The result of (Kearns et al., 1994) shows up a further problem. Under a standard cryptographic assumption the class of acyclic PDFAs over a two-letter alphabet are not learnable since the class of noisy parity functions can be embedded in this simple subclass of PDFAs. (Ron et al., 1995) show that this can be circumvented by adding an additional parameter to the sample complexity polynomial, the distinguishability, which we define below.

3 Preliminaries

We will write σ for letters and s for strings. We have a finite alphabet Σ, and Σ^* is the free monoid generated by Σ, i.e. the set of all strings with letters from Σ, with λ the empty string (identity). For $s \in \Sigma^*$ we define $|s|$ to be the length of s. The subset of Σ^* of strings of length d is denoted by Σ^d. A distribution or stochastic language D over Σ^* is a function $D : \Sigma^* \to [0, 1]$ such that $\sum_{s\in\Sigma^*} D(s) = 1$. The L_∞ norm between two distributions is defined as $\max_s |D_1(s) - D_2(s)|$. For a multiset of strings S we write $\hat{S}$ for the empirical distribution defined by that multiset – the maximum likelihood estimate of the probability of the string.

A probabilistic deterministic finite state automaton is a mathematical object that stochastically generates strings of symbols. It has a finite number of states one of which is a distinguished start state. Parsing or generating starts in the start state, and at any given moment makes a transition with a certain probability to another state and emits a symbol. We have a particular symbol and state which correspond to finishing.

A PDFA A is a tuple $(Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma)$, where

- Q is a finite set of states,
- Σ is the alphabet, a finite set of symbols,
- $q_0 \in Q$ is the single initial state,
- $q_f \notin Q$ is the final state,
- $\zeta \notin \Sigma$ is the final symbol,
- $\tau : Q \times \Sigma \cup \{\zeta\} \to Q \cup \{q_f\}$ is the transition function and
- $\gamma : Q \times \Sigma \cup \{\zeta\} \to [0, 1]$ is the next symbol probability function. $\gamma(q, \sigma) = 0$ when $\tau(q, \sigma)$ is not defined.

We will sometimes refer to automata by the set of states. All transitions that emit ζ go to the final state. In the following τ and γ will be extended to strings recursively in the normal way.

The sum of the output transition from each states must be one: so for all $q \in Q$

$$\sum_{\sigma\in\Sigma\cup\{\zeta\}} \gamma(q, \sigma) = 1 \tag{1}$$

Assuming further that there is a non zero probability of reaching the final state from each state: i.e.

$$\forall q \in Q \exists s \in \Sigma^* : \tau(q, s\zeta) = q_f \wedge \gamma(q, s\zeta) > 0 \tag{2}$$

the PDFA then defines a probability distribution over Σ^*, where the probability of generating a string $s \in \Sigma^*$ is $P^A(s) = \gamma(q_0, s\zeta)$. We will write $L(A)$ for the support of this distribution, $L(A) = \{s \in \Sigma^* : P^A(s) > 0\}$. We will also define $P_q(s) = \gamma(q, s\zeta)$ which we call the suffix distribution of the state q.

We say that two states q, q' are μ-distinguishable if $L_\infty(P_q, P_{q'}) > \mu$ for some $\mu > 0$. An automaton is μ-distinguishable iff every pair of states is μ-distinguishable. Since we can merge states q, q' which have $L_\infty(P_q, P_{q'}) = 0$, we can assume without loss of generality that every PDFA has a non-zero distinguishability.

Note that $\gamma(q_0, s)$ where $s \in \Sigma^*$ is the prefix probability of the string s, i.e. the probability that the automaton will generate a string that *starts* with s.

We will use a similar notation, neglecting the probability function for (non-probabilistic) deterministic finite-state automata (DFAs).

4 Algorithm

We shall first state our main result.

Theorem 1 *For any regular language L, when samples are generated by a PDFA A where $L(A) = L$, with distinguishability μ and number of states n, for any $\epsilon, \delta > 0$, the algorithm LearnDFA will with probability at least $1 - \delta$ return a DFA H which defines a language $L(H)$ that is a subset of L with $P_A(L(A) - L(H)) < \epsilon$. The algorithm will draw a number of samples bounded by a polynomial in $|\Sigma|, n, 1/\mu, 1/\epsilon, 1/\delta$, and the computation is bounded by a polynomial in the number of samples and the total length of the strings in the sample.*

We now define the algorithm LearnDFA. We incrementally construct a sequence of DFAs that will generate subsets of the target language. Each state of the hypothesis automata will represent a state of the target and will have attached a multiset of strings that approximates the distribution of strings generated by that state. We calculate the following quantities m_0 and N from the input parameters.

$$m_0 = \frac{8}{\mu^2} \log \frac{48n|\Sigma|(n|\Sigma| + 2)}{\mu\delta} \tag{3}$$

$$N = \frac{2n|\Sigma|m_0}{\epsilon} \tag{4}$$

We start with an automaton that consists of a single state and no transitions, and the attached multiset is a sample of strings from the target. At each step we sample N strings from the target distribution. This re-sampling ensures the independence of all of the samples, and allows us to apply bounds in a straightforward way. For each state u in the hypothesis automaton and letter σ in the alphabet, such that there is no arc labelled with σ out of u we construct a candidate node (u, σ) which represents the state reached from u by the transition labelled with σ. For each string in the sample, we trace the corresponding path through the hypothesis. When we reach a candidate node, we remove the preceding part of the string, and add the rest to the multiset of the candidate node. Otherwise, in the case when the string terminates in the hypothesis automaton we discard the string.

After we have done this for every string in the sample, we select a candidate node (u, σ) that has a multiset of size at least m_0. If there is no such candidate node, the algorithm terminates, Otherwise we compare this candidate node with each of the nodes already in the hypothesis. The comparison we use calculates the L_∞-norm between the empirical distributions of the two multisets and says they are similar if this distance is less than $\mu/4$. We will make sure that with high probability these empirical distributions are close in the L_∞-norm to the suffix distributions of the states they represent. Since we know that the suffix distributions of different states will be at least μ apart, we can be confident that we will only rarely make mistakes. If there is a node, v, which is similar then we conclude that v and (u, σ) represent the same state. We therefore add an arc labelled with σ leading from u to v. If it is not similar to any node in the hypothesis, then we conclude that it represents a new node, and we create a new node u' and add an arc labelled with σ leading from u to u'. In this case we attach the multiset of the candidate node to the new node in the hypothesis. Intuitively this multiset will be a sample from the suffix distribution of the state of the target that it represents. We then discard all of the candidate nodes and their associated multisets, but keep the multisets attached to the states of the hypothesis, and repeat.

5 Proof

We can now prove that this algorithm has the properties we claim. We use one technical lemma that we prove in the appendix.

Lemma 1 *Given a distribution D over Σ^*, for any $\mu' < 1/2$, when we independently draw a number of samples m more than $m_0 = \frac{1}{2\mu'^2} \log \frac{12}{\mu'\delta'}$, into a multiset S then $L_\infty(\hat{S}, D) < \mu'$ with probability greater than $1 - \delta'$.*

Let $H_0, H_1, \ldots, H_k$ be the sequence of finite automata, the states labelled with multisets, generated by the algorithm when samples are generated by a target PDFA A.

We will say that a hypothesis automaton H_i is *μ-good* if there is a bijective function Φ from a subset of states of A including q_0, to all the states of H_i such that $\Phi(q_0)$ is the root node of H_i, and if there is an edge in H_i such that $\tau(u, \sigma) = v$ then $\tau(\Phi^{-1}(u), \sigma) = \Phi^{-1}(v)$ i.e. if H_i is isomorphic to a subgraph of the target that includes the root. If $\Phi(q) = u$ then we say that u represents q. In this case the language generated by H_i is a subset of the target language. Additionally we require that for every state v in the hypothesis, the corresponding multiset satisfies $L_\infty(\hat{S_v}, P_{\Phi^{-1}(v)}) < \mu/4$. When a multiset satisfies this we will say it is *μ-good.*

We will extend the function Φ to candidate nodes in the obvious way, and also the definition of μ-good.

Definition 1 (Good sample) *We say that a sample of size N is μ-ϵ-good given a good hypothesis DFA H and a target A if all the candidate nodes with multisets larger than the threshold m_0 are μ-good, and that if $P_A(L(A) - L(H)) > \epsilon$ then the number of strings that exit the hypothesis automaton is more than $\frac{1}{2} N P_A(L(A) - L(H))$.*

5.1 Approximately Correct

We will now show if all the samples are good, that for all $i - 0, 1, \ldots, k$, the hypothesis H_i will be good, and that when the algorithm terminates the final hypothesis will have low error. We will do this by induction on the index i of the hypothesis H_i. Clearly H_0 is good. Suppose H_{i-1} is good, and we draw a good sample. Consider a candidate node (u, σ) with multiset greater than m_0.

Since the previous hypothesis was good, this will be a representative of a state q and thus the multiset will be a sequence of independent draws from the suffix distribution of this state P_q. Thus $L_\infty(\hat{S_{u,\sigma}}, P_q) < \mu/4$ by the goodness of the sample. We compare it to a state in the hypothesis v. If this state is a representative of the same state in the target v, then $L_\infty(\hat{S_v}, P_q) < \mu/4$ (by the goodness of the multisets), the triangle inequality shows that $L_\infty(\hat{S_{u,\sigma}}, \hat{S_v}) < \mu/2$, and therefore the comparison will return true. On the other hand, let us suppose that v is a representative of a different state q_v. We know that $L_\infty(\hat{S_{u,\sigma}}, P_q) < \mu/4$ and $L_\infty(\hat{S_v}, P_{q_v}) < \mu/4$ (by the goodness of the multisets), and $L_\infty(P_q, P_{q_v}) \geq \mu$ (by the μ-distinguishability of the target). By the triangle inequality $L_\infty(P_q, P_{q_v}) \leq L_\infty(\hat{S_{u,\sigma}}, P_q) + L_\infty(\hat{S_{u,\sigma}}, \hat{S_v}) + L_\infty(\hat{S_v}, P_{q_v})$, which implies that $L_\infty(\hat{S_{u,\sigma}}, \hat{S_v}) > \mu/2$ and the comparison will return false. In these cases H_i will be good. Alternatively there is no candidate node above threshold in which case the algorithm terminates, and $i = k$. The total number of strings that exit the hypotheis must then be less than $n|\Sigma|m_0$ since there are at most $n|\Sigma|$ candidate nodes each of which has multiset of size less than m_0. By the definition of N and the goodness of the sample $P_A(L(A) - L(H)) < \epsilon$. Since it is good and thus defines a subset of the target language, this is a suitably close hypothesis.

5.2 Probably Correct

We must now show that by setting m_0 sufficiently large we can be sure that with probability greater than $1 - \delta$ all of the samples will be good. We need to show that with high probability a sample of size N will be good for a given hypotheis G. We can assume that the hypothesis is good at each step. Each step of the algorithm will increase the number of transitions in the active set by at least 1. There are at most $n|\Sigma|$ transitions in the target; so there are at most $n|\Sigma| + 2$ steps in the algorithm since we need an initial step to get the multiset for the root node and another at the end when we terminate. So we want to show that a particular sample will be good with probability at least $1 - \frac{\delta}{n|\Sigma|+2}$.

There are two sorts of errors that can make the sample bad. First, one of the multisets could be bad, and secondly too few strings might exit the graph. There are at most $n|\Sigma|$ candidate nodes, so we will make the probability of getting a bad multiset less than $\delta/2n|\Sigma|(n|\Sigma| + 2)$, and we will make the probability of the second sort of error less than $\delta/2(n|\Sigma| + 2)$.

First we bound the probability of getting a bad multiset of size m_0. This will be satisfied if we set $\mu' = \mu/4$ and $\delta' = \delta/2n|\Sigma|(n|\Sigma| + 2)$, and use Lemma 1.

We next need to show that at each step the number of strings that exit the graph will be not too far from its expectation, if $P_A(L(A) - L(H)) > \epsilon$. We can use Chernoff bounds to show that the probability too few strings exit the graph will be less than $\delta/2(n|\Sigma| + 2)$

$$\begin{aligned} e^{-N(G)P_A(L(A)-L(H))/4} &< e^{-N\epsilon/4} \\ &< \delta/2(n|\Sigma| + 2) \end{aligned}$$

which will be satisfied by the value of N defined earlier, as can be easily verified.

5.3 Polynomial complexity

Since we need to draw at most $n|\Sigma| + 2$ samples of size N the overall sample complexity will be $(n|\Sigma| + 2)N$, which ignoring log factors gives a sample complexity of $\mathcal{O}(n^2|\Sigma|^2\mu^{-2}\epsilon^{-1})$, which is quite benign. It is easy to see that the computational complexity is polynomial. Producing an exact bound is difficult since it depends on the length of the strings. The precise complexity also depends on the relative magnitudes of μ, $|\Sigma|$ and so on. The complexity is dominated by the cost of the comparisons. We can limit each multiset comparison to at most m_0 strings, which can be compared naively with m_0^2 string comparisons or much more efficiently using hashing or sorting. The number of nodes in the hypothesis is at most n, and the number of candidate nodes is at most $n|\Sigma|$, so the number of comparisons at each step is bounded by $n^2|\Sigma|$ and thus the total number of multiset comparisons by $n^2|\Sigma|(n|\Sigma|+2)$. Construction of multisets can be performed in time linear in the sample size. These observations suffice to show that the computation is polynomially bounded.

6 Discussion

The convergence of these sorts of algorithms has been studied before in the identification in the limit framework, but previous proofs have not been completely convincing (Carrasco and Oncina, 1999), and this criterion gives no guide to the practical utility of the algorithms since it applies only asymptotically. The partially distribution free learning problem we study here is novel. as is the extension of the results of (Ron et al., 1995) to cyclic automata and thus to infinite languages.

Before we examine our results critically, we would like to point out some positive aspects of the algorithm. First, this class of algorithms is in practice efficient and reliable. This particular algorithm is designed to have a provably good worst-case performance, and thus we anticipate its average performance on naturally occurring data to be marginally worse than comparable algorithms. We have established that we can learn an exponentially large family of infinite languages using polynomial amounts of data and computation. Mild properties of the input distributions suffice to guarantee learnability. The algorithm we present here is however not intended to be efficient or cognitively plausible: our intention was to find one that allowed a simple proof.

The major weakness of this approach in our opinion is that the parameter n in the sample complexity polynomial is the number of states in the PDFA generating the distribution, and not the number of states in the minimal FA generating the language. Since determinisation of finite automata can cause exponential blow ups this is potentially a serious problem, depending on the application domain. A second problem is the need for a distinguishability parameter, which again in specific cases could be exponentially small. An alternative to this is to define a class of μ-distinguishable automata where the distinguishability is bounded by an inverse polynomial in the number of states. Formally this is equivalent, but it has the effect of removing the parameter from the sample complexity polynomial at the cost of having a further restriction on the class of distributions. Indeed we can deal with the previous objection in the same way if necessary by requiring the number of states in the generating PDFA to be bounded by a polynomial in the minimal number of states needed to generate the target language. However both of these limitations are unavoidable given the negative results previously discussed.

References

S. Abney, D. McAllester, and F. Pereira. 1999. Relating probabilistic grammars and automata. In *Proceedings of ACL '99*.

R. C. Carrasco and J. Oncina. 1994. Learning stochastic regular grammars by means of a state merging method. In R. C. Carrasco and J. Oncina, editors, *Grammatical Inference and Applications, ICGI-94*, number 862 in LNAI, pages 139–152, Berlin, Heidelberg. Springer Verlag.

R. C. Carrasco and J. Oncina. 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, 33(1):1–20.

Alexander Clark and Franck Thollard. 2004.

Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5:473–497, May.

F. Denis. 2001. Learning regular languages from simple positive examples. *Machine Learning*, 44(1/2):37–66.

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

M. Kearns and G. Valiant. 1989. Cryptographic limitations on learning boolean formulae and finite automata. In *21st annual ACM symposium on Theory of computation*, pages 433–444, New York. ACM, ACM.

M.J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R.E. Schapire, and L. Sellie. 1994. On the learnability of discrete distributions. In *Proc. of the 25th Annual ACM Symposium on Theory of Computing*, pages 273–282.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

D. Ron, Y. Singer, and N. Tishby. 1995. On the learnability and usage of acyclic probabilistic finite automata. In *COLT 1995*, pages 31–40, Santa Cruz CA USA. ACM.

L. Valiant. 1984. A theory of the learnable. *Communications of the ACM*, 27(11):1134 – 1142.

Appendix

Proof of Lemma 1.

We write $p(s)$ for the true probability and $\hat{p}(s) = c(s)/m$ for the empirical probability of the string in the sample – i.e. the maximum likelihood estimate. We want to bound the probability over an infinite number of strings, which rules out a naive application of Hoeffding bounds. It will suffice to show that every string with probability less than $\mu'/2$ will have empirical probability less than μ', and that all other strings will have probability within μ' of their true values. The latter is straightforward: since there are at most $2/\mu'$ of these frequent strings. For any given frequent string s, by Hoeffding bounds:

$$Pr[|\hat{p}(s) - p(s)| > \mu'] < 2e^{-2m\mu'^2} < 2e^{-2m_0\mu'^2} \tag{5}$$

So the probability of making an error on a frequent string is less than $4/\mu' e^{-2m_0\mu'^2}$.

Consider all of the strings whose probability is in $[\mu' 2^{-(k+1)}, \mu' 2^{-k})$.

$$S_k = \{s \in \Sigma^* : \gamma(q, s\zeta) \in [\mu' 2^{-(k+1)}, \mu' 2^{-k})\} \tag{6}$$

We define $S_{rare} = \bigcup_{k=1}^{\infty} S_k$. The Chernoff bound says that for any $\delta > 0$, for the sum of n bernouilli variables with prob p and

$$Pr(X > (1+\delta)np) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{np} \tag{7}$$

Now we bound each group separately, using the binomial Chernoff bound where $n = m\mu' > mp$ (which is true since $p < \mu'$)

$$Pr\left[\hat{p}(s) \geq \mu'\right] \leq \left(\frac{mp}{n}\right)^n e^{n-mp} \tag{8}$$

This bound decreases with p, so we can replace this for all strings in S_k with the upper bound for the probability, and we can replace m with m_0.

$$\begin{aligned} Pr\left[\hat{p}(s) \geq \mu'\right] &\leq \left(\frac{m_0\mu' 2^{-k}}{m_0\mu'}\right)^{m_0\mu'} e^{m_0\mu' - m_0\mu' 2^{-k}} \\ &\leq \left(2^{-k} e^{1-2^{-k}}\right)^{m_0\mu'} < 2^{-km_0\mu'} \end{aligned}$$

Assuming that $m_0\mu' > 3$

$$Pr\left[\hat{p}(s) \geq \mu'\right] < 2^{-2k} 2^{2-m_0\mu'}$$

$$\begin{aligned} Pr\left[\exists s \in S_k : \hat{p}(s) \geq \mu'\right] &\leq |S_k| 2^{-2k} 2^{2-m_0\mu'} \\ &\leq \frac{8}{\mu} 2^{-k} 2^{-m_0\mu'} \end{aligned}$$

Using the factor of the form 2^{-k}, we can sum over all of the k.

$$\begin{aligned} Pr[\forall s \in S_{rare} : \hat{p}(s) \geq \mu'] &< \frac{8}{\mu'} 2^{-m_0\mu'} \sum_{k=1}^{\infty} 2^{-k} \\ &< \frac{8}{\mu} 2^{-m_0\mu'} \end{aligned}$$

Putting these together we can show that the probability of the bound being exceeded will be

$$\frac{4}{\mu'} e^{-2m_0\mu'^2} + \frac{8}{\mu} 2^{-m_0\mu'} < \frac{12}{\mu'} e^{-2m_0\mu'^2} \tag{9}$$

This will be less than δ' if

$$m_0 = \frac{1}{2\mu'^2} \log \frac{12}{\mu'\delta'} \tag{10}$$

which establishes the result.