Restricting Probability Distributions to Increase the Class of Learnable Languages

Alexander Clark[1] and Shalom Lappin[2]

[1]Department of Computer Science
Royal Holloway College London

[2]Department of Philosophy
King's College London

RANLP 2009
September 15, 2009

Outline

Goal of Computational Learning Theory Applied to Natural Languages

- Formal learnability results can give us insight into the nature of natural language.
- Learning language is a computational process.
- The learnability of a class of languages depends on
  1. the formal properties of its members, and
  2. the data available to the learning algorithm.

Goal of Computational Learning Theory Applied to Natural Languages

- Formal learnability results can give us insight into the nature of natural language.
- Learning language is a computational process.
- The learnability of a class of languages depends on
  1. the formal properties of its members, and
  2. the data available to the learning algorithm.

Goal of Computational Learning Theory Applied to Natural Languages

- Formal learnability results can give us insight into the nature of natural language.
- Learning language is a computational process.
- The learnability of a class of languages depends on
    1. the formal properties of its members, and
    2. the data available to the learning algorithm.

Goal of Computational Learning Theory Applied to Natural Languages

- Formal learnability results can give us insight into the nature of natural language.
- Learning language is a computational process.
- The learnability of a class of languages depends on
  1. the formal properties of its members, and
  2. the data available to the learning algorithm.

Negative results in PAC learning

Negative results in modern statistical learning theory come in two types.

1. Information theoretic bounds on the amount of data required (sample complexity):
    - Given an infinite hypothesis space, the Vapnik-Chervonenkis (VC) dimension characterizes whether something can be learned from bounded amounts of data.
2. Complexity problems concerning the amount of computation required to complete a learning task:
    - There may be enough information available, but it may not be possible to process it efficiently.

Negative results in PAC learning

Negative results in modern statistical learning theory come in two types.

1. Information theoretic bounds on the amount of data required (sample complexity):
   - Given an infinite hypothesis space, the Vapnik-Chervonenkis (VC) dimension characterizes whether something can be learned from bounded amounts of data.
2. Complexity problems concerning the amount of computation required to complete a learning task:
   - There may be enough information available, but it may not be possible to process it efficiently.

Negative results in PAC learning

Negative results in modern statistical learning theory come in two types.

1. Information theoretic bounds on the amount of data required (sample complexity):
   - Given an infinite hypothesis space, the Vapnik-Chervonenkis (VC) dimension characterizes whether something can be learned from bounded amounts of data.
2. Complexity problems concerning the amount of computation required to complete a learning task:
   - There may be enough information available, but it may not be possible to process it efficiently.

Negative results in PAC learning

Negative results in modern statistical learning theory come in two types.

1. Information theoretic bounds on the amount of data required (sample complexity):
   - Given an infinite hypothesis space, the Vapnik-Chervonenkis (VC) dimension characterizes whether something can be learned from bounded amounts of data.
2. Complexity problems concerning the amount of computation required to complete a learning task:
   - There may be enough information available, but it may not be possible to process it efficiently.

Negative results in PAC learning

Negative results in modern statistical learning theory come in two types.

1. Information theoretic bounds on the amount of data required (sample complexity):
   - Given an infinite hypothesis space, the Vapnik-Chervonenkis (VC) dimension characterizes whether something can be learned from bounded amounts of data.
2. Complexity problems concerning the amount of computation required to complete a learning task:
   - There may be enough information available, but it may not be possible to process it efficiently.

Finite Languages

- Valiant (1984) introduced the PAC learning paradigm as a distribution free model.
- Nowak, Komarova and Niyogi (2002) point out that the class of finite languages is not PAC learnable.
- This result follows from the fact that the hypothesis space for this class exhibits infinite VC dimension.
- For any data sample from the vocabulary of a finite language there will be a set in the class of finite languages that includes it, and one that excludes it.
- Therefore any data sample from the vocabulary of a finite language will be shattered by the hypothesis space for this class of languages.

Finite Languages

- Valiant (1984) introduced the PAC learning paradigm as a distribution free model.
- Nowak, Komarova and Niyogi (2002) point out that the class of finite languages is not PAC learnable.
- This result follows from the fact that the hypothesis space for this class exhibits infinite VC dimension.
- For any data sample from the vocabulary of a finite language there will be a set in the class of finite languages that includes it, and one that excludes it.
- Therefore any data sample from the vocabulary of a finite language will be shattered by the hypothesis space for this class of languages.

Finite Languages

- Valiant (1984) introduced the PAC learning paradigm as a distribution free model.
- Nowak, Komarova and Niyogi (2002) point out that the class of finite languages is not PAC learnable.
- This result follows from the fact that the hypothesis space for this class exhibits infinite VC dimension.
- For any data sample from the vocabulary of a finite language there will be a set in the class of finite languages that includes it, and one that excludes it.
- Therefore any data sample from the vocabulary of a finite language will be shattered by the hypothesis space for this class of languages.

Finite Languages

- Valiant (1984) introduced the PAC learning paradigm as a distribution free model.
- Nowak, Komarova and Niyogi (2002) point out that the class of finite languages is not PAC learnable.
- This result follows from the fact that the hypothesis space for this class exhibits infinite VC dimension.
- For any data sample from the vocabulary of a finite language there will be a set in the class of finite languages that includes it, and one that excludes it.
- Therefore any data sample from the vocabulary of a finite language will be shattered by the hypothesis space for this class of languages.

Finite Languages

- Valiant (1984) introduced the PAC learning paradigm as a distribution free model.
- Nowak, Komarova and Niyogi (2002) point out that the class of finite languages is not PAC learnable.
- This result follows from the fact that the hypothesis space for this class exhibits infinite VC dimension.
- For any data sample from the vocabulary of a finite language there will be a set in the class of finite languages that includes it, and one that excludes it.
- Therefore any data sample from the vocabulary of a finite language will be shattered by the hypothesis space for this class of languages.

Regular Languages

- The hypothesis space for the class of regular languages also has infinite VC dimension.
- A string from the vocabulary of any regular language will be shattered by the hypothesis space of this class of languages, which contains an infinite set of FSAs (FSGs).
- Therefore the class of regular languages is also not PAC learnable.

Regular Languages

- The hypothesis space for the class of regular languages also has infinite VC dimension.
- A string from the vocabulary of any regular language will be shattered by the hypothesis space of this class of languages, which contains an infinite set of FSAs (FSGs).
- Therefore the class of regular languages is also not PAC learnable.

Regular Languages

- The hypothesis space for the class of regular languages also has infinite VC dimension.
- A string from the vocabulary of any regular language will be shattered by the hypothesis space of this class of languages, which contains an infinite set of FSAs (FSGs).
- Therefore the class of regular languages is also not PAC learnable.

Imposing an Upper Bound on the Size of the Language

- As Nowak et al. (2002) observe, by imposing an upper bound k on the cardinality of the sets of finite languages in $\mathcal{H}_{\mathcal{L}}$, one achieves finite VC dimension for this hypothesis space.
- The VC-dimension of such an $\mathcal{H}_{\mathcal{L}}$ whose elements are bounded in size is at most k.
- In this case the class of languages in $\mathcal{H}_{\mathcal{L}}$ is PAC learnable.

Imposing an Upper Bound on the Size of the Language

- As Nowak et al. (2002) observe, by imposing an upper bound k on the cardinality of the sets of finite languages in $\mathcal{H}_{\mathcal{L}}$, one achieves finite VC dimension for this hypothesis space.
- The VC-dimension of such an $\mathcal{H}_{\mathcal{L}}$ whose elements are bounded in size is at most k.
- In this case the class of languages in $\mathcal{H}_{\mathcal{L}}$ is PAC learnable.

Imposing an Upper Bound on the Size of the Language

- As Nowak et al. (2002) observe, by imposing an upper bound *k* on the cardinality of the sets of finite languages in $\mathcal{H}_{\mathcal{L}}$, one achieves finite VC dimension for this hypothesis space.
- The VC-dimension of such an $\mathcal{H}_{\mathcal{L}}$ whose elements are bounded in size is at most *k*.
- In this case the class of languages in $\mathcal{H}_{\mathcal{L}}$ is PAC learnable.

Imposing an Upper Bound on the Size of the Grammar

- Similarly, the class of regular languages generated by FSAs with an upper bound of *k* states (FSGs with not more than *k* rules) is PAC learnable.
- This result recalls Shinohara's (1994) theorem stating that length bounded EFSs can be inferred from positive evidence for the class of context sensitive grammars, in the Gold paradigm.
- However, unlike Shinohara's theorem, the PAC framework requires that learning is tractable.

Imposing an Upper Bound on the Size of the Grammar

- Similarly, the class of regular languages generated by FSAs with an upper bound of k states (FSGs with not more than k rules) is PAC learnable.
- This result recalls Shinohara's (1994) theorem stating that length bounded EFSs can be inferred from positive evidence for the class of context sensitive grammars, in the Gold paradigm.
- However, unlike Shinohara's theorem, the PAC framework requires that learning is tractable.

Imposing an Upper Bound on the Size of the Grammar

- Similarly, the class of regular languages generated by FSAs with an upper bound of k states (FSGs with not more than k rules) is PAC learnable.
- This result recalls Shinohara's (1994) theorem stating that length bounded EFSs can be inferred from positive evidence for the class of context sensitive grammars, in the Gold paradigm.
- However, unlike Shinohara's theorem, the PAC framework requires that learning is tractable.

Obtaining an Upper Bound on a Grammar

- In many cases, we don't need to specify the upper bound as a condition on learning.
- For example, for particular finite languages we can learn without having a prior bound.
- We could posit an upper bound on the size of possible grammars as a learning prior.
- Alternatively, it may be possible to estimate an upper bound from learning samples, or we could have a gradually increasing bound as a function of the amount of data we have seen.
- The size of the representation of a language is normally a parameter for the sample complexity polynomial.

Obtaining an Upper Bound on a Grammar

- In many cases, we don't need to specify the upper bound as a condition on learning.
- For example, for particular finite languages we can learn without having a prior bound.
- We could posit an upper bound on the size of possible grammars as a learning prior.
- Alternatively, it may be possible to estimate an upper bound from learning samples, or we could have a gradually increasing bound as a function of the amount of data we have seen.
- The size of the representation of a language is normally a parameter for the sample complexity polynomial.

Obtaining an Upper Bound on a Grammar

- In many cases, we don't need to specify the upper bound as a condition on learning.
- For example, for particular finite languages we can learn without having a prior bound.
- We could posit an upper bound on the size of possible grammars as a learning prior.
- Alternatively, it may be possible to estimate an upper bound from learning samples, or we could have a gradually increasing bound as a function of the amount of data we have seen.
- The size of the representation of a language is normally a parameter for the sample complexity polynomial.

Obtaining an Upper Bound on a Grammar

- In many cases, we don't need to specify the upper bound as a condition on learning.
- For example, for particular finite languages we can learn without having a prior bound.
- We could posit an upper bound on the size of possible grammars as a learning prior.
- Alternatively, it may be possible to estimate an upper bound from learning samples, or we could have a gradually increasing bound as a function of the amount of data we have seen.
- The size of the representation of a language is normally a parameter for the sample complexity polynomial.

Obtaining an Upper Bound on a Grammar

- In many cases, we don't need to specify the upper bound as a condition on learning.
- For example, for particular finite languages we can learn without having a prior bound.
- We could posit an upper bound on the size of possible grammars as a learning prior.
- Alternatively, it may be possible to estimate an upper bound from learning samples, or we could have a gradually increasing bound as a function of the amount of data we have seen.
- The size of the representation of a language is normally a parameter for the sample complexity polynomial.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and

(2) every distribution D on the data samples, and

(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,

(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and

(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and

(2) every distribution D on the data samples, and

(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,

(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and

(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and

(2) every distribution D on the data samples, and

(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,

(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and

(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and

(2) every distribution D on the data samples, and

(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,

(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and

(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and
(2) every distribution D on the data samples, and
(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,
(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and
(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and

(2) every distribution D on the data samples, and

(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,

(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and

(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and
(2) every distribution D on the data samples, and
(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,
(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and
(6) the algorithm runs in polynomial time.

Distribution Free Learning

An algorithm $\mathcal{A}$ PAC-learns a class of languages $\mathcal{L}$ if and only if, there is a polynomial q, such that

(1) for every $L \in \mathcal{L}$ and
(2) every distribution D on the data samples, and
(3) $\epsilon, \delta > 0$,

whenever $\mathcal{A}$ sees a number of samples greater than $q(1/\epsilon, 1/\delta)$,

(4) it returns a hypothesis H such that with probability greater than $1 - \delta$,
(5) the error of the hypothesis $P_D((H - L) \cup (L - H)) < \epsilon$, and
(6) the algorithm runs in polynomial time.

Modifying the Distribution Free Learning Assumption

- The PAC learning model requires that if a class of languages is learnable, then it is learnable for all probability distributions on data samples from that class.
- By modifying this assumption and restricting the set of possible distributions available for PAC learning in a specified hypothesis space $\mathcal{H}$, it is possible to significantly alter the class of PAC learnable languages.
- This approach uses properties of the probability distributions for a class of languages to facilitate learning of that class, and this can solve computational complexity problems.

Modifying the Distribution Free Learning Assumption

- The PAC learning model requires that if a class of languages is learnable, then it is learnable for all probability distributions on data samples from that class.
- By modifying this assumption and restricting the set of possible distributions available for PAC learning in a specified hypothesis space $\mathcal{H}$, it is possible to significantly alter the class of PAC learnable languages.
- This approach uses properties of the probability distributions for a class of languages to facilitate learning of that class, and this can solve computational complexity problems.

Modifying the Distribution Free Learning Assumption

- The PAC learning model requires that if a class of languages is learnable, then it is learnable for all probability distributions on data samples from that class.
- By modifying this assumption and restricting the set of possible distributions available for PAC learning in a specified hypothesis space $\mathcal{H}$, it is possible to significantly alter the class of PAC learnable languages.
- This approach uses properties of the probability distributions for a class of languages to facilitate learning of that class, and this can solve computational complexity problems.

Specifying the Set of Distributions for a Class of Languages

- Clark and Thollard (2004) (C&T) characterize the set of distributions $\mathcal{D}$ for a class of languages $\mathcal{L}$ through the stochastic variant of the automata that generate the elements of $\mathcal{L}$.
- For each $L \in \mathcal{L}$, a distribution for L is the set of probability values for the strings constructed from the vocabulary of L, where the stochastic automata that generates L assigns a probability greater than 0 to each string L.
- If Σ^* is the set of strings on the vocabulary Σ of L, then the set of distributions for L is $D_L = \{D \in \mathcal{D} : \forall s \in \Sigma^*(s \in L \Leftrightarrow P_D(s) > 0)\}$

Specifying the Set of Distributions for a Class of Languages

- Clark and Thollard (2004) (C&T) characterize the set of distributions $\mathcal{D}$ for a class of languages $\mathcal{L}$ through the stochastic variant of the automata that generate the elements of $\mathcal{L}$.
- For each $L \in \mathcal{L}$, a distribution for L is the set of probability values for the strings constructed from the vocabulary of L, where the stochastic automata that generates L assigns a probability greater than 0 to each string L.
- If Σ^* is the set of strings on the vocabulary Σ of L, then the set of distributions for L is $D_L = \{D \in \mathcal{D} : \forall s \in \Sigma^*(s \in L \Leftrightarrow P_D(s) > 0)\}$

Specifying the Set of Distributions for a Class of Languages

- Clark and Thollard (2004) (C&T) characterize the set of distributions $\mathcal{D}$ for a class of languages $\mathcal{L}$ through the stochastic variant of the automata that generate the elements of $\mathcal{L}$.
- For each $L \in \mathcal{L}$, a distribution for L is the set of probability values for the strings constructed from the vocabulary of L, where the stochastic automata that generates L assigns a probability greater than 0 to each string L.
- If Σ^* is the set of strings on the vocabulary Σ of L, then the set of distributions for L is
  $D_L = \{D \in \mathcal{D} : \forall s \in \Sigma^*(s \in L \Leftrightarrow P_D(s) > 0)\}$

Constraining Distributions in PAC Learning

- C&T define a set of probabilistic deterministic FSAs (PDFAs), each of which generates a stochastic regular language (a set of strings in a regular language to which the PDFA assigns probability values).
- A stochastic language L specifies a probability distribution for the strings in L.
- C&T show that if we restrict the set of possible distributions for a PAC model to those generated by PDFAs, the class of regular languages that these automata define is PAC learnable, on the basis of positive evidence only .

Constraining Distributions in PAC Learning

- C&T define a set of probabilistic deterministic FSAs (PDFAs), each of which generates a stochastic regular language (a set of strings in a regular language to which the PDFA assigns probability values).
- A stochastic language L specifies a probability distribution for the strings in L.
- C&T show that if we restrict the set of possible distributions for a PAC model to those generated by PDFAs, the class of regular languages that these automata define is PAC learnable, on the basis of positive evidence only .

Constraining Distributions in PAC Learning

- C&T define a set of probabilistic deterministic FSAs (PDFAs), each of which generates a stochastic regular language (a set of strings in a regular language to which the PDFA assigns probability values).
- A stochastic language L specifies a probability distribution for the strings in L.
- C&T show that if we restrict the set of possible distributions for a PAC model to those generated by PDFAs, the class of regular languages that these automata define is PAC learnable, on the basis of positive evidence only .

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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,
- $\tau : Q \times \Sigma \cup \{\zeta\} \rightarrow Q \cup \{q_f\}$ is the transition function, and
- $\gamma : Q \times \Sigma \cup \{\zeta\} \rightarrow [0, 1]$ is the next symbol probability function ($\gamma(q, \sigma) = 0$ when $\tau(q, \sigma)$ is not defined).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Characterizing PDFAs

A PDFA A is a tuple $\langle Q, \Sigma, q_0, q_f, \zeta, \tau, \gamma \rangle$, 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).

The probability of a string $Pr(s) = \gamma(q_0, s\zeta)$.

Residual and Suffix Distributions

- Residual distribution of a string s with $\gamma(q_0, s) > 0$:

  $Pr_s(t) = \frac{\gamma(q_0, st\zeta)}{\gamma(q_0, s)}$

- Suffix distribution of the state q:

  $Pr_q(s) = \gamma(q, s\zeta)$

- μ-distinguishability:

  for every pair of states u, v there is a string s such that $|Pr_u(s) - Pr_v(s)| > \mu$

Residual and Suffix Distributions

- Residual distribution of a string s with $\gamma(q_0, s) > 0$:

  $Pr_s(t) = \frac{\gamma(q_0, st\zeta)}{\gamma(q_0, s)}$
- Suffix distribution of the state q:

  $Pr_q(s) = \gamma(q, s\zeta)$
- μ-distinguishability:

  for every pair of states u, v there is a string s such that $|Pr_u(s) - Pr_v(s)| > \mu$

Residual and Suffix Distributions

- Residual distribution of a string s with $\gamma(q_0, s) > 0$:

  $Pr_s(t) = \frac{\gamma(q_0, st\zeta)}{\gamma(q_0, s)}$
- Suffix distribution of the state q:

  $Pr_q(s) = \gamma(q, s\zeta)$
- μ-distinguishability:

  for every pair of states u, v there is a string s such that $|Pr_u(s) - Pr_v(s)| > \mu$

Residual and Suffix Distributions

- The fact that PDFAs are deterministic and the independence assumptions that they encode entail that for any string s, there is a state q_s such that $\tau(q_0, s) = q_s$.
- It follows that $Pr_{q_s} = Pr_s$.
- The key insight here is that the residual distribution of any string is equal to the suffix distribution of some state.

Residual and Suffix Distributions

- The fact that PDFAs are deterministic and the independence assumptions that they encode entail that for any string s, there is a state q_s such that $\tau(q_0, s) = q_s$.
- It follows that $Pr_{q_s} = Pr_s$.
- The key insight here is that the residual distribution of any string is equal to the suffix distribution of some state.

Residual and Suffix Distributions

- The fact that PDFAs are deterministic and the independence assumptions that they encode entail that for any string s, there is a state q_s such that $\tau(q_0, s) = q_s$.
- It follows that $Pr_{q_s} = Pr_s$.
- The key insight here is that the residual distribution of any string is equal to the suffix distribution of some state.

Distinguishing States in the Construction of a PDFA

- The $L_\infty(D)$ norm for a distribution D is the largest probability value that D assigns to its arguments.
- The L_∞ norm between two distributions D_1 and D_2, $L_\infty(D_1 - D_2)$, is the maximal difference between the probability values that each assigns to the same argument ($\max_s |D_1(s) - D_2(s)|$).
- $P_q(s) = \gamma(q, s\zeta)$.
- Two states q, q' are $\mu - \textit{distinguishable}$ if $L_\infty(P_q - P_{q'}) > \mu$, where $\mu > 0$.

Distinguishing States in the Construction of a PDFA

- The $L_\infty(D)$ norm for a distribution D is the largest probability value that D assigns to its arguments.
- The L_∞ norm between two distributions D_1 and D_2, $L_\infty(D_1 - D_2)$, is the maximal difference between the probability values that each assigns to the same argument ($\max_s |D_1(s) - D_2(s)|$).
- $P_q(s) = \gamma(q, s\zeta)$.
- Two states q, q' are $\mu -$ *distinguishable* if $L_\infty(P_q - P_{q'}) > \mu$, where $\mu > 0$.

Distinguishing States in the Construction of a PDFA

- The $L_\infty(D)$ norm for a distribution D is the largest probability value that D assigns to its arguments.
- The L_∞ norm between two distributions D_1 and D_2, $L_\infty(D_1 - D_2)$, is the maximal difference between the probability values that each assigns to the same argument ($\max_s |D_1(s) - D_2(s)|$).
- $P_q(s) = \gamma(q, s\zeta)$.
- Two states q, q' are $\mu -$ *distinguishable* if $L_\infty(P_q - P_{q'}) > \mu$, where $\mu > 0$.

Distinguishing States in the Construction of a PDFA

- The $L_\infty(D)$ norm for a distribution D is the largest probability value that D assigns to its arguments.
- The L_∞ norm between two distributions D_1 and D_2, $L_\infty(D_1 - D_2)$, is the maximal difference between the probability values that each assigns to the same argument $(\max_s |D_1(s) - D_2(s)|)$.
- $P_q(s) = \gamma(q, s\zeta)$.
- Two states q, q' are $\mu -$ *distinguishable* if $L_\infty(P_q - P_{q'}) > \mu$, where $\mu > 0$.

Distinguishing States in the Construction of a PDFA

- The suffix distributions of two states are at least μ apart in the L_∞ norm.
- We can make the empirical estimates of the suffix distributions within $\mu' = \mu/4$ of the true distribution.
- We can test whether two strings s and t are in the same state by checking the L_∞ norm between the empirical estimates of their residual distributions.
- If the L_∞-norm between two empirical distributions is less than $\mu/2$, then they correspond to the same state.

Distinguishing States in the Construction of a PDFA

- The suffix distributions of two states are at least μ apart in the L_∞ norm.
- We can make the empirical estimates of the suffix distributions within $\mu' = \mu/4$ of the true distribution.
- We can test whether two strings s and t are in the same state by checking the L_∞ norm between the empirical estimates of their residual distributions.
- If the L_∞-norm between two empirical distributions is less than $\mu/2$, then they correspond to the same state.

Distinguishing States in the Construction of a PDFA

- The suffix distributions of two states are at least μ apart in the L_∞ norm.
- We can make the empirical estimates of the suffix distributions within $\mu' = \mu/4$ of the true distribution.
- We can test whether two strings s and t are in the same state by checking the L_∞ norm between the empirical estimates of their residual distributions.
- If the L_∞-norm between two empirical distributions is less than $\mu/2$, then they correspond to the same state.

Distinguishing States in the Construction of a PDFA

- The suffix distributions of two states are at least μ apart in the L_∞ norm.
- We can make the empirical estimates of the suffix distributions within $\mu' = \mu/4$ of the true distribution.
- We can test whether two strings *s* and *t* are in the same state by checking the L_∞ norm between the empirical estimates of their residual distributions.
- If the L_∞-norm between two empirical distributions is less than $\mu/2$, then they correspond to the same state.

LearnDFA: A State-Merging Algorithm for Learning PDFAs

- The algorithm LearnDFA incrementally constructs a DFA in which each state has a multiset of strings, where this multiset represents the suffix distribution.
- Starting with the full multiset of strings in the language at the initial state, LearnDFA progressively moves through multisets of suffixes for the strings in this multiset by comparing candidate nodes with nodes in the DFA, and
  (1) adding a new state if the candidate has a different distribution than the existing states of the DFA, or
  (2) adding a new arc if it is identical to one of these states.

LearnDFA: A State-Merging Algorithm for Learning PDFAs

- The algorithm LearnDFA incrementally constructs a DFA in which each state has a multiset of strings, where this multiset represents the suffix distribution.
- Starting with the full multiset of strings in the language at the initial state, LearnDFA progressively moves through multisets of suffixes for the strings in this multiset by comparing candidate nodes with nodes in the DFA, and
  1. adding a new state if the candidate has a different distribution than the existing states of the DFA, or
  2. adding a new arc if it is identical to one of these states.

LearnDFA: A State-Merging Algorithm for Learning PDFAs

- The algorithm LearnDFA incrementally constructs a DFA in which each state has a multiset of strings, where this multiset represents the suffix distribution.
- Starting with the full multiset of strings in the language at the initial state, LearnDFA progressively moves through multisets of suffixes for the strings in this multiset by comparing candidate nodes with nodes in the DFA, and
  1. adding a new state if the candidate has a different distribution than the existing states of the DFA, or
  2. adding a new arc if it is identical to one of these states.

LearnDFA: A State-Merging Algorithm for Learning PDFAs

- The algorithm LearnDFA incrementally constructs a DFA in which each state has a multiset of strings, where this multiset represents the suffix distribution.
- Starting with the full multiset of strings in the language at the initial state, LearnDFA progressively moves through multisets of suffixes for the strings in this multiset by comparing candidate nodes with nodes in the DFA, and
  1. adding a new state if the candidate has a different distribution than the existing states of the DFA, or
  2. adding a new arc if it is identical to one of these states.

A PAC Learnability Result for PDFA Generated Regular Languages

C&T prove the following theorem.

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

A PAC Learnability Result for PDFA Generated Regular Languages

C&T prove the following theorem.

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

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG) $G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG) $G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG)
$G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG) $G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG) $G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

Probabilistic Context Free Grammars

A probabilistic context free grammar (PCFG) $G = \langle N, T, S, P, S \rangle$, where

- N is the set of non-terminal symbols,
- T is the set of terminal symbols,
- S is the start symbol of G (corresponding to the root node of a sentence),
- P is the set of production (CFG) rules, and
- D is a function assigning probabilites to the elements of P.

PCFGs and Probability Distributions for Languages

- For every non-terminal $A \in N$, $\sum_{A \to \alpha \in P} D(A \to \alpha) = 1$.
- For a derivation $A \Rightarrow^* \alpha$, the probability of the derivation is the product of the probabilities for the rules applied in the derivation.
- The probability that a PCFG G determines for a string s is the sum of the probabilities that G assigns to the derivations of s.
- The distribution P_D that a PCFG specifies for a language L is the probability values that P_D assigns to the strings in L.
- If G is consistent, then $\sum_{s \in T^*} P_D(s) = 1$.

PCFGs and Probability Distributions for Languages

- For every non-terminal $A \in N$, $\sum_{A \to \alpha \in P} D(A \to \alpha) = 1$.
- For a derivation $A \Rightarrow^* \alpha$, the probability of the derivation is the product of the probabilities for the rules applied in the derivation.
- The probability that a PCFG G determines for a string s is the sum of the probabilities that G assigns to the derivations of s.
- The distribution P_D that a PCFG specifies for a language L is the probability values that P_D assigns to the strings in L.
- If G is consistent, then $\sum_{s \in T^*} P_D(s) = 1$.

PCFGs and Probability Distributions for Languages

- For every non-terminal $A \in N$, $\sum_{A \to \alpha \in P} D(A \to \alpha) = 1$.
- For a derivation $A \Rightarrow^* \alpha$, the probability of the derivation is the product of the probabilities for the rules applied in the derivation.
- The probability that a PCFG G determines for a string s is the sum of the probabilities that G assigns to the derivations of s.
- The distribution P_D that a PCFG specifies for a language L is the probability values that P_D assigns to the strings in L.
- If G is consistent, then $\sum_{s \in T^*} P_D(s) = 1$.

PCFGs and Probability Distributions for Languages

- For every non-terminal $A \in N$, $\sum_{A \to \alpha \in P} D(A \to \alpha) = 1$.
- For a derivation $A \Rightarrow^* \alpha$, the probability of the derivation is the product of the probabilities for the rules applied in the derivation.
- The probability that a PCFG G determines for a string s is the sum of the probabilities that G assigns to the derivations of s.
- The distribution P_D that a PCFG specifies for a language L is the probability values that P_D assigns to the strings in L.
- If G is consistent, then $\sum_{s \in T^*} P_D(s) = 1$.

PCFGs and Probability Distributions for Languages

- For every non-terminal $A \in N$, $\sum_{A \to \alpha \in P} D(A \to \alpha) = 1$.
- For a derivation $A \Rightarrow^* \alpha$, the probability of the derivation is the product of the probabilities for the rules applied in the derivation.
- The probability that a PCFG G determines for a string s is the sum of the probabilities that G assigns to the derivations of s.
- The distribution P_D that a PCFG specifies for a language L is the probability values that P_D assigns to the strings in L.
- If G is consistent, then $\sum_{s \in T^*} P_D(s) = 1$.

NTS Languages

- A CFG is non-terminally distinct (NTS) iff for every $A \in N$, if $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$, then $A \Rightarrow^* \alpha B\gamma$.
- For any two non-terminals A, C in an NTS grammar, the string sets derivable from A and C are disjoint.
- This propery corresponds to the requirement that the phrases of distinct syntactic categories in a natural language do not overlap.
- Clark (2006) shows that a subclass of CF languages, generated by a restricted set of NTS PCFGs, is PAC learnable from positive evidence only, given certain conditions on the probability distributions for these grammars.

NTS Languages

- A CFG is non-terminally distinct (NTS) iff for every $A \in N$, if $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$, then $A \Rightarrow^* \alpha B\gamma$.
- For any two non-terminals A, C in an NTS grammar, the string sets derivable from A and C are disjoint.
- This propery corresponds to the requirement that the phrases of distinct syntactic categories in a natural language do not overlap.
- Clark (2006) shows that a subclass of CF languages, generated by a restricted set of NTS PCFGs, is PAC learnable from positive evidence only, given certain conditions on the probability distributions for these grammars.

NTS Languages

- A CFG is non-terminally distinct (NTS) iff for every $A \in N$, if $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$, then $A \Rightarrow^* \alpha B\gamma$.
- For any two non-terminals A, C in an NTS grammar, the string sets derivable from A and C are disjoint.
- This propery corresponds to the requirement that the phrases of distinct syntactic categories in a natural language do not overlap.
- Clark (2006) shows that a subclass of CF languages, generated by a restricted set of NTS PCFGs, is PAC learnable from positive evidence only, given certain conditions on the probability distributions for these grammars.

NTS Languages

- A CFG is non-terminally distinct (NTS) iff for every $A \in N$, if $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$, then $A \Rightarrow^* \alpha B\gamma$.
- For any two non-terminals A, C in an NTS grammar, the string sets derivable from A and C are disjoint.
- This propery corresponds to the requirement that the phrases of distinct syntactic categories in a natural language do not overlap.
- Clark (2006) shows that a subclass of CF languages, generated by a restricted set of NTS PCFGs, is PAC learnable from positive evidence only, given certain conditions on the probability distributions for these grammars.

Additional Constraints on NTS Grammars: Non-Ambigiuity

- Clark (2006) specifies a subclass $\mathcal{C}_{NTS}$ of NTS grammars that satisfy three conditions.
- The members of $\mathcal{C}_{NTS}$ are unambiguous, where a grammar G is unambiguous iff every string in the language that it generates has only one (rightmost) derivation in G.
- This constraint significantly reduces the set of NTS grammars, and so of NTS languages.

Additional Constraints on NTS Grammars: Non-Ambigiuity

- Clark (2006) specifies a subclass $\mathcal{C}_{NTS}$ of NTS grammars that satisfy three conditions.
- The members of $\mathcal{C}_{NTS}$ are unambiguous, where a grammar G is unambiguous iff every string in the language that it generates has only one (rightmost) derivation in G.
- This constraint significantly reduces the set of NTS grammars, and so of NTS languages.

Additional Constraints on NTS Grammars: Non-Ambigiuity

- Clark (2006) specifies a subclass $\mathcal{C}_{NTS}$ of NTS grammars that satisfy three conditions.
- The members of $\mathcal{C}_{NTS}$ are unambiguous, where a grammar G is unambiguous iff every string in the language that it generates has only one (rightmost) derivation in G.
- This constraint significantly reduces the set of NTS grammars, and so of NTS languages.

Additional Constraints on NTS Grammars: Non-Ambigiuity

- $L = \{a^n | n > 0\}$ is an ambiguous NTS language (ie. when it is taken as a language generated by an NTS grammar).
- As this language contains the string *aaa*, its NTS grammar must contain the rules $S \rightarrow a$ and $S \rightarrow SS$.
- These rules produce two distinct rightmost derivations for *aaa*.
  1. $S \Rightarrow SS \Rightarrow Sa \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$
  2. $S \Rightarrow SS \Rightarrow SSS \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$

Additional Constraints on NTS Grammars: Non-Ambigiuity

- $L = \{a^n | n > 0\}$ is an ambiguous NTS language (ie. when it is taken as a language generated by an NTS grammar).
- As this language contains the string *aaa*, its NTS grammar must contain the rules $S \rightarrow a$ and $S \rightarrow SS$.
- These rules produce two distinct rightmost derivations for *aaa*.
    1. $S \Rightarrow SS \Rightarrow Sa \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$
    2. $S \Rightarrow SS \Rightarrow SSS \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$

Additional Constraints on NTS Grammars: Non-Ambigiuity

- $L = \{a^n | n > 0\}$ is an ambiguous NTS language (ie. when it is taken as a language generated by an NTS grammar).
- As this language contains the string *aaa*, its NTS grammar must contain the rules $S \rightarrow a$ and $S \rightarrow SS$.
- These rules produce two distinct rightmost derivations for *aaa*.
   1. $S \Rightarrow SS \Rightarrow Sa \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$
   2. $S \Rightarrow SS \Rightarrow SSS \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$

Additional Constraints on NTS Grammars: Non-Ambigiuity

- $L = \{a^n | n > 0\}$ is an ambiguous NTS language (ie. when it is taken as a language generated by an NTS grammar).
- As this language contains the string *aaa*, its NTS grammar must contain the rules $S \rightarrow a$ and $S \rightarrow SS$.
- These rules produce two distinct rightmost derivations for *aaa*.
  1. $S \Rightarrow SS \Rightarrow Sa \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$
  2. $S \Rightarrow SS \Rightarrow SSS \Rightarrow SSa \Rightarrow Saa \Rightarrow aaa$

Additional Constraints on NTS Grammars: Non-Redundancy and Non-Duplication

- The members of $\mathcal{C}_{NTS}$ contain no redundant non-terminals $(\forall A \in N(\exists u \in T^*(A \Rightarrow^* u) \wedge \exists l, r \in T^*(S \Rightarrow^* lAr)))$.
- The grammars in $\mathcal{C}_{NTS}$ contain no duplicate non-terminals (no non-terminals that generate the same strings).
- Unlike the non-ambiguity condition, these two constraints concern only the form of the grammar, but they do not alter the class of NTS languages.

Additional Constraints on NTS Grammars: Non-Redundancy and Non-Duplication

- The members of $\mathcal{C}_{NTS}$ contain no redundant non-terminals $(\forall A \in N(\exists u \in T^*(A \Rightarrow^* u) \wedge \exists l, r \in T^*(S \Rightarrow^* lAr)))$.
- The grammars in $\mathcal{C}_{NTS}$ contain no duplicate non-terminals (no non-terminals that generate the same strings).
- Unlike the non-ambiguity condition, these two constraints concern only the form of the grammar, but they do not alter the class of NTS languages.

Additional Constraints on NTS Grammars: Non-Redundancy and Non-Duplication

- The members of $\mathcal{C}_{NTS}$ contain no redundant non-terminals $(\forall A \in N(\exists u \in T^*(A \Rightarrow^* u) \wedge \exists l, r \in T^*(S \Rightarrow^* lAr)))$.
- The grammars in $\mathcal{C}_{NTS}$ contain no duplicate non-terminals (no non-terminals that generate the same strings).
- Unlike the non-ambiguity condition, these two constraints concern only the form of the grammar, but they do not alter the class of NTS languages.

Syntactic Congruence

- Two strings u, v are syntactically congruent in L iff they share all and only the same syntactic contexts in L ($u \equiv_L v$ *iff* $\forall l, r \in L$ $lur \Leftrightarrow lvr$).
- Let $Ct \subseteq T^* X\ T^*$ be the set of pairs of left and right contexts for strings in T^*, and Ct_u the set of contexts in which the substring u occurs.
- u, v are syntactically congruent in L iff their contexts are identical for L ($u \equiv_L v$ *iff* $Ct_u =_L Ct_v$)

Syntactic Congruence

- Two strings u, v are syntactically congruent in L iff they share all and only the same syntactic contexts in L ($u \equiv_L v$ *iff* $\forall l, r \in L$ $lur \Leftrightarrow lvr$).
- Let $Ct \subseteq T^* X\ T^*$ be the set of pairs of left and right contexts for strings in T^*, and Ct_u the set of contexts in which the substring u occurs.
- u, v are syntactically congruent in L iff their contexts are identical for L ($u \equiv_L v$ *iff* $Ct_u =_L Ct_v$)

Syntactic Congruence

- Two strings u, v are syntactically congruent in L iff they share all and only the same syntactic contexts in L ($u \equiv_L v$ *iff* $\forall l, r \in L$ $lur \Leftrightarrow lvr$).
- Let $Ct \subseteq T^* X\ T^*$ be the set of pairs of left and right contexts for strings in T^*, and Ct_u the set of contexts in which the substring u occurs.
- u, v are syntactically congruent in L iff their contexts are identical for L ($u \equiv_L v$ *iff* $Ct_u =_L Ct_v$)

Characterizing Syntactic Congruence Probabilistically

- Let $C : T^* \ X \ T^* \rightarrow [0, 1]$ be a function from context pairs to probability values, C_u the context function for a substring u in L, and P_D a distribution for L.
- $C_u^{P_D}(l, r) = \dfrac{P_D(lur)}{\sum_{l,r} P_D(lur)}$ (the probability that P_D assigns to lur divided by the expected number of occurrences of u).
- u, v are probabilistically congruent for a distribution D iff their context functions are identical for P_D ($u \simeq_{P_D} v \ iff \ C_u^{P_D} = C_v^{P_D}$).

Characterizing Syntactic Congruence Probabilistically

- Let $C : T^* \ X \ T^* \rightarrow [0, 1]$ be a function from context pairs to probability values, C_u the context function for a substring u in L, and P_D a distribution for L.
- $C_u^{P_D}(l, r) = \dfrac{P_D(lur)}{\sum_{l,r} P_D(lur)}$ (the probability that P_D assigns to lur divided by the expected number of occurrences of u).
- u, v are probabilistically congruent for a distribution D iff their context functions are identical for P_D ($u \simeq_{P_D} v$ *iff* $C_u^{P_D} = C_v^{P_D}$).

Characterizing Syntactic Congruence Probabilistically

- Let $C : T^* \; X \; T^* \rightarrow [0, 1]$ be a function from context pairs to probability values, C_u the context function for a substring u in L, and P_D a distribution for L.
- $C_u^{P_D}(l, r) = \frac{P_D(lur)}{\sum_{l,r} P_D(lur)}$ (the probability that P_D assigns to lur divided by the expected number of occurrences of u).
- u, v are probabilistically congruent for a distribution D iff their context functions are identical for P_D ($u \simeq_{P_D} v$ *iff* $C_u^{P_D} = C_v^{P_D}$).

Three Parameters for Learning PCFGs: μ_1-Distinguishability

- Clark (2006) posits three parameters whose values determine lower bounds on properties of non-terminals to insure their identification from data samples.
- A PCFG is $\mu_1 -$ *distinguishable* iff for every $A \in N$ there is a string u such that $D(A \Rightarrow^* u) > \mu_1$.
- This property sets a lower bound on the probability of strings for each non-terminal in the grammar, and so provides a confidence threshold for the distinguishability of the members of N.

Three Parameters for Learning PCFGs: μ_1-Distinguishability

- Clark (2006) posits three parameters whose values determine lower bounds on properties of non-terminals to insure their identification from data samples.
- A PCFG is $\mu_1 -$ *distinguishable* iff for every $A \in N$ there is a string u such that $D(A \Rightarrow^* u) > \mu_1$.
- This property sets a lower bound on the probability of strings for each non-terminal in the grammar, and so provides a confidence threshold for the distinguishability of the members of N.

Three Parameters for Learning PCFGs: μ_1-Distinguishability

- Clark (2006) posits three parameters whose values determine lower bounds on properties of non-terminals to insure their identification from data samples.
- A PCFG is $\mu_1 -$ *distinguishable* iff for every $A \in N$ there is a string u such that $D(A \Rightarrow^* u) > \mu_1$.
- This property sets a lower bound on the probability of strings for each non-terminal in the grammar, and so provides a confidence threshold for the distinguishability of the members of N.

Three Parameters for Learning PCFGs: ν-Separability

- A PCFG is $\nu -$ *separable* for some $\nu > 0$ if for every pair of strings u, v in the set of substrings of $L(G)$ such that $\neg(u \equiv v)$, $L_\infty(C_u - C_v) \geq \nu min(L_\infty(C_u), L_\infty(C_v))$.
- This property specifies a minimal distance between the context distributions of non-congruent strings.

Three Parameters for Learning PCFGs: ν-Separability

- A PCFG is $\nu -$ *separable* for some $\nu > 0$ if for every pair of strings u, v in the set of substrings of $L(G)$ such that $\neg(u \equiv v)$, $L_\infty(C_u - C_v) \geq \nu min(L_\infty(C_u), L_\infty(C_v))$.
- This property specifies a minimal distance between the context distributions of non-congruent strings.

Three Parameters for Learning PCFGs: μ_2-Reachability

- A PCFG is μ_2-reachable if, for every non-terminal $A \in N$ there is a string u such that $A \Rightarrow^* u$, and $L_\infty(C_u) > \mu_2$.
- This property is equivalent to the requirement that for every $A \in N$ $L_\infty(C_A) > \mu_2$.
- It specifies a lower bound on the frequency of contexts for non-terminals.

Three Parameters for Learning PCFGs: μ_2-Reachability

- A PCFG is μ_2-reachable if, for every non-terminal $A \in N$ there is a string u such that $A \Rightarrow^* u$, and $L_\infty(C_u) > \mu_2$.
- This property is equivalent to the requirement that for every $A \in N$ $L_\infty(C_A) > \mu_2$.
- It specifies a lower bound on the frequency of contexts for non-terminals.

Three Parameters for Learning PCFGs: μ_2-Reachability

- A PCFG is μ_2-reachable if, for every non-terminal $A \in N$ there is a string u such that $A \Rightarrow^* u$, and $L_\infty(C_u) > \mu_2$.
- This property is equivalent to the requirement that for every $A \in N$ $L_\infty(C_A) > \mu_2$.
- It specifies a lower bound on the frequency of contexts for non-terminals.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of *uv* substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings *a*, and
  - identifying the initial *S* symbol with the congruence class of strings in the language.

The PACCFG Algorithm

Clark (2006) defines the PACCFG algorithm as follows.

- Gather a finite sample.
- Identify frequent substrings in the sample.
- Test the substrings for probabilistic congruence, and identify the probabilistic congruence classes.
- Create a grammar by
  - adding non-terminals for each congruence class,
  - adding production rules $[uv] \rightarrow [u][v]$ for congruence classes of uv substrings,
  - adding production rules $[a] \rightarrow a$ for congruence classes of single symbol substrings a, and
  - identifying the initial S symbol with the congruence class of strings in the language.

PAC Learnability of NTS Languages

- The PACCFG algorithm invokes values for the parameters of μ_1-distinguishability, ν-separability, and μ_2-reachability to identify the non-terminals of a PCFG from samples.
- Clark (2006) shows that, with appropriate values for these parameters, the class of unambiguous, non-redundant NTS CFGs which do not contain duplicate non-terminals is PAC learnable from positive data only.

PAC Learnability of NTS Languages

- The PACCFG algorithm invokes values for the parameters of μ_1-distinguishability, ν-separability, and μ_2-reachability to identify the non-terminals of a PCFG from samples.
- Clark (2006) shows that, with appropriate values for these parameters, the class of unambiguous, non-redundant NTS CFGs which do not contain duplicate non-terminals is PAC learnable from positive data only.

Negative Evidence in IIL and PAC Learning Paradigms

- Both IIL and PAC learning paradigms assume that either negative evidence in the form of membership labeling is available for every data sample, or for none of them.
- This assumption is unrealistic, as human learners receive negative evidence only for a proper subset of the primary linguistic data (PLD).
- Moreover, this negative evidence does not generally take the form of explicit labelling.

Negative Evidence in IIL and PAC Learning Paradigms

- Both IIL and PAC learning paradigms assume that either negative evidence in the form of membership labeling is available for every data sample, or for none of them.
- This assumption is unrealistic, as human learners receive negative evidence only for a proper subset of the primary linguistic data (PLD).
- Moreover, this negative evidence does not generally take the form of explicit labelling.

Negative Evidence in IIL and PAC Learning Paradigms

- Both IIL and PAC learning paradigms assume that either negative evidence in the form of membership labeling is available for every data sample, or for none of them.
- This assumption is unrealistic, as human learners receive negative evidence only for a proper subset of the primary linguistic data (PLD).
- Moreover, this negative evidence does not generally take the form of explicit labelling.

Inferring Ungrammaticality from Low Frequency

- Indirect negative evidence has been informally posited in the linguistics and acquisition literature, but no attempt has been made to formalize this concept of evidence in a learning model.
- Clark and Lappin (2009) (C&L) propose a way of doing this that represents indirect negative evidence stochastically as a two-part inference procedure.
- The learner first infers the low probability of a string from its low frequency in the data.
- He/She then derives the ungrammaticality of a string from its comparatively low probability.

Inferring Ungrammaticality from Low Frequency

- Indirect negative evidence has been informally posited in the linguistics and acquisition literature, but no attempt has been made to formalize this concept of evidence in a learning model.
- Clark and Lappin (2009) (C&L) propose a way of doing this that represents indirect negative evidence stochastically as a two-part inference procedure.
- The learner first infers the low probability of a string from its low frequency in the data.
- He/She then derives the ungrammaticality of a string from its comparatively low probability.

Inferring Ungrammaticality from Low Frequency

- Indirect negative evidence has been informally posited in the linguistics and acquisition literature, but no attempt has been made to formalize this concept of evidence in a learning model.
- Clark and Lappin (2009) (C&L) propose a way of doing this that represents indirect negative evidence stochastically as a two-part inference procedure.
- The learner first infers the low probability of a string from its low frequency in the data.
- He/She then derives the ungrammaticality of a string from its comparatively low probability.

Inferring Ungrammaticality from Low Frequency

- Indirect negative evidence has been informally posited in the linguistics and acquisition literature, but no attempt has been made to formalize this concept of evidence in a learning model.
- Clark and Lappin (2009) (C&L) propose a way of doing this that represents indirect negative evidence stochastically as a two-part inference procedure.
- The learner first infers the low probability of a string from its low frequency in the data.
- He/She then derives the ungrammaticality of a string from its comparatively low probability.

From Low Frequency to Low Probability

- C&L assume each sentence in a presentation is generated independently from the same probability distribution, where this is the Independently and Identically Distributed assumption (IID) common in statistical analysis.
- The IID is an idealizing assumption that abstracts away from the obvious probability dependencies among sentences that are conditioned by semantic, dialogue, discourse, and other factors.
- The hope is that over very large amounts of data the IID converges on an approximation of the facts.
- The inference from the low frequency of a string in a data set to its low probability in the distribution for the language follows from the IID.

From Low Frequency to Low Probability

- C&L assume each sentence in a presentation is generated independently from the same probability distribution, where this is the Independently and Identically Distributed assumption (IID) common in statistical analysis.
- The IID is an idealizing assumption that abstracts away from the obvious probability dependencies among sentences that are conditioned by semantic, dialogue, discourse, and other factors.
- The hope is that over very large amounts of data the IID converges on an approximation of the facts.
- The inference from the low frequency of a string in a data set to its low probability in the distribution for the language follows from the IID.

From Low Frequency to Low Probability

- C&L assume each sentence in a presentation is generated independently from the same probability distribution, where this is the Independently and Identically Distributed assumption (IID) common in statistical analysis.
- The IID is an idealizing assumption that abstracts away from the obvious probability dependencies among sentences that are conditioned by semantic, dialogue, discourse, and other factors.
- The hope is that over very large amounts of data the IID converges on an approximation of the facts.
- The inference from the low frequency of a string in a data set to its low probability in the distribution for the language follows from the IID.

From Low Frequency to Low Probability

- C&L assume each sentence in a presentation is generated independently from the same probability distribution, where this is the Independently and Identically Distributed assumption (IID) common in statistical analysis.
- The IID is an idealizing assumption that abstracts away from the obvious probability dependencies among sentences that are conditioned by semantic, dialogue, discourse, and other factors.
- The hope is that over very large amounts of data the IID converges on an approximation of the facts.
- The inference from the low frequency of a string in a data set to its low probability in the distribution for the language follows from the IID.

From Low Probability to Ungrammaticality

- Grammaticality does not reduce to a high probability value for a string.
- Some grammatical strings in a language have vanishingly rare frequency, and so they have low probability
- We also cannot identify ungrammaticality with 0 probability, as some ungrammatical strings do occur in the PLD.
- We need to specify a suitable lower bound on probability to distinguish grammatical from ungrammatical strings.

From Low Probability to Ungrammaticality

- Grammaticality does not reduce to a high probability value for a string.
- Some grammatical strings in a language have vanishingly rare frequency, and so they have low probability
- We also cannot identify ungrammaticality with 0 probability, as some ungrammatical strings do occur in the PLD.
- We need to specify a suitable lower bound on probability to distinguish grammatical from ungrammatical strings.

From Low Probability to Ungrammaticality

- Grammaticality does not reduce to a high probability value for a string.
- Some grammatical strings in a language have vanishingly rare frequency, and so they have low probability
- We also cannot identify ungrammaticality with 0 probability, as some ungrammatical strings do occur in the PLD.
- We need to specify a suitable lower bound on probability to distinguish grammatical from ungrammatical strings.

From Low Probability to Ungrammaticality

- Grammaticality does not reduce to a high probability value for a string.
- Some grammatical strings in a language have vanishingly rare frequency, and so they have low probability
- We also cannot identify ungrammaticality with 0 probability, as some ungrammatical strings do occur in the PLD.
- We need to specify a suitable lower bound on probability to distinguish grammatical from ungrammatical strings.

A Lower Probability Bound for Grammatical Strings

- Given that the learner learns from unlabelled data, there must be a function from the set of distributions for a language $\mathcal{D}(L)$ to that language.
- This condition entails the Disjoint Distribution Assumption (DDA):
  If $L \neq L'$ then $\mathcal{D}(L) \cap \mathcal{D}(L') = \emptyset$.
- If g is a function that maps a string into a lower bound probability value for grammaticality, relative to a distribution, then we can specify the restricted set of possible distributions for a language as $\mathcal{D}(L, g) = \{D : p_D(s) > g_D(s) \Leftrightarrow s \in L\}$.

A Lower Probability Bound for Grammatical Strings

- Given that the learner learns from unlabelled data, there must be a function from the set of distributions for a language $\mathcal{D}(L)$ to that language.
- This condition entails the Disjoint Distribution Assumption (DDA):
  If $L \neq L'$ then $\mathcal{D}(L) \cap \mathcal{D}(L') = \emptyset$.
- If g is a function that maps a string into a lower bound probability value for grammaticality, relative to a distribution, then we can specify the restricted set of possible distributions for a language as $\mathcal{D}(L, g) = \{D : p_D(s) > g_D(s) \Leftrightarrow s \in L\}$.

A Lower Probability Bound for Grammatical Strings

- Given that the learner learns from unlabelled data, there must be a function from the set of distributions for a language $\mathcal{D}(L)$ to that language.
- This condition entails the Disjoint Distribution Assumption (DDA):
  If $L \neq L'$ then $\mathcal{D}(L) \cap \mathcal{D}(L') = \emptyset$.
- If g is a function that maps a string into a lower bound probability value for grammaticality, relative to a distribution, then we can specify the restricted set of possible distributions for a language as
  $\mathcal{D}(L, g) = \{D : p_D(s) > g_D(s) \Leftrightarrow s \in L\}$.

Specifying the Threshold Function

- Defining the restricted set of possible distributions in terms of the lower bound function g satisfies DDA.
- To have content this definition must be supplemented with a characterization of g.
- It is useful to specify g in a way that renders it dependent on properties of its distribution.
- One way of doing this is to make it sensitive to the conditional probabilities of a class-based n-gram language model of the kind described, for example, in Pereira (2000).
- When g depends on properties of D, the learner will need to estimate these properties in order to determine g.

Specifying the Threshold Function

- Defining the restricted set of possible distributions in terms of the lower bound function g satisfies DDA.
- To have content this definition must be supplemented with a characterization of g.
- It is useful to specify g in a way that renders it dependent on properties of its distribution.
- One way of doing this is to make it sensitive to the conditional probabilities of a class-based n-gram language model of the kind described, for example, in Pereira (2000).
- When g depends on properties of D, the learner will need to estimate these properties in order to determine g.

Specifying the Threshold Function

- Defining the restricted set of possible distributions in terms of the lower bound function g satisfies DDA.
- To have content this definition must be supplemented with a characterization of g.
- It is useful to specify g in a way that renders it dependent on properties of its distribution.
- One way of doing this is to make it sensitive to the conditional probabilities of a class-based n-gram language model of the kind described, for example, in Pereira (2000).
- When g depends on properties of D, the learner will need to estimate these properties in order to determine g.

Specifying the Threshold Function

- Defining the restricted set of possible distributions in terms of the lower bound function *g* satisfies DDA.
- To have content this definition must be supplemented with a characterization of *g*.
- It is useful to specify *g* in a way that renders it dependent on properties of its distribution.
- One way of doing this is to make it sensitive to the conditional probabilities of a class-based *n*-gram language model of the kind described, for example, in Pereira (2000).
- When *g* depends on properties of *D*, the learner will need to estimate these properties in order to determine *g*.

Specifying the Threshold Function

- Defining the restricted set of possible distributions in terms of the lower bound function g satisfies DDA.
- To have content this definition must be supplemented with a characterization of g.
- It is useful to specify g in a way that renders it dependent on properties of its distribution.
- One way of doing this is to make it sensitive to the conditional probabilities of a class-based n-gram language model of the kind described, for example, in Pereira (2000).
- When g depends on properties of D, the learner will need to estimate these properties in order to determine g.

Revising PAC Learning with Indirect Evidence

- Given g it is possible to model indirect negative evidence through membership queries on large samples of data.
- The learner can test a number of strings polynomial in the sample for grammaticality by computing the probability of each string s from its frequency, and then comparing its probability to the threshold value $g(s)$.
- C&L revise the definition of PAC learning so that an algorithm effectively learns $\mathcal{L}$ not for every distribution $D \in \mathcal{D}$, but for every distribution $D \in \mathcal{D}(L, g)$.
- In this revised PAC learning paradigm the data set is not labelled, and the set of possible distributions on the data is restricted by a function giving a lower probability bound for membership in the language.

Revising PAC Learning with Indirect Evidence

- Given g it is possible to model indirect negative evidence through membership queries on large samples of data.
- The learner can test a number of strings polynomial in the sample for grammaticality by computing the probability of each string s from its frequency, and then comparing its probability to the threshold value $g(s)$.
- C&L revise the definition of PAC learning so that an algorithm effectively learns $\mathcal{L}$ not for every distribution $D \in \mathcal{D}$, but for every distribution $D \in \mathcal{D}(L, g)$.
- In this revised PAC learning paradigm the data set is not labelled, and the set of possible distributions on the data is restricted by a function giving a lower probability bound for membership in the language.

Revising PAC Learning with Indirect Evidence

- Given g it is possible to model indirect negative evidence through membership queries on large samples of data.
- The learner can test a number of strings polynomial in the sample for grammaticality by computing the probability of each string s from its frequency, and then comparing its probability to the threshold value $g(s)$.
- C&L revise the definition of PAC learning so that an algorithm effectively learns $\mathcal{L}$ not for every distribution $D \in \mathcal{D}$, but for every distribution $D \in \mathcal{D}(L, g)$.
- In this revised PAC learning paradigm the data set is not labelled, and the set of possible distributions on the data is restricted by a function giving a lower probability bound for membership in the language.

Revising PAC Learning with Indirect Evidence

- Given g it is possible to model indirect negative evidence through membership queries on large samples of data.
- The learner can test a number of strings polynomial in the sample for grammaticality by computing the probability of each string s from its frequency, and then comparing its probability to the threshold value $g(s)$.
- C&L revise the definition of PAC learning so that an algorithm effectively learns $\mathcal{L}$ not for every distribution $D \in \mathcal{D}$, but for every distribution $D \in \mathcal{D}(L, g)$.
- In this revised PAC learning paradigm the data set is not labelled, and the set of possible distributions on the data is restricted by a function giving a lower probability bound for membership in the language.

Conclusions

- C&T show that if distributions are restricted to those determined by PDFAs, then the class of regular languages that these automata generate are PAC learnable from positive data only.
- Clark (2006) extends this approach to NTS languages generated by PCFGS, to demonstrate PAC learnability from positive data only for an interesting subclass of CF languages.
- C&L propose a stochastic model for indirect negative evidence, and they integrate it into PAC learning.
- This work indicates that by modifying the distribution free assumption of PAC learning, the class of effectively learnable languages can be significantly expanded.

Conclusions

- C&T show that if distributions are restricted to those determined by PDFAs, then the class of regular languages that these automata generate are PAC learnable from positive data only.
- Clark (2006) extends this approach to NTS languages generated by PCFGS, to demonstrate PAC learnability from positive data only for an interesting subclass of CF languages.
- C&L propose a stochastic model for indirect negative evidence, and they integrate it into PAC learning.
- This work indicates that by modifying the distribution free assumption of PAC learning, the class of effectively learnable languages can be significantly expanded.

Conclusions

- C&T show that if distributions are restricted to those determined by PDFAs, then the class of regular languages that these automata generate are PAC learnable from positive data only.
- Clark (2006) extends this approach to NTS languages generated by PCFGS, to demonstrate PAC learnability from positive data only for an interesting subclass of CF languages.
- C&L propose a stochastic model for indirect negative evidence, and they integrate it into PAC learning.
- This work indicates that by modifying the distribution free assumption of PAC learning, the class of effectively learnable languages can be significantly expanded.

Conclusions

- C&T show that if distributions are restricted to those determined by PDFAs, then the class of regular languages that these automata generate are PAC learnable from positive data only.
- Clark (2006) extends this approach to NTS languages generated by PCFGS, to demonstrate PAC learnability from positive data only for an interesting subclass of CF languages.
- C&L propose a stochastic model for indirect negative evidence, and they integrate it into PAC learning.
- This work indicates that by modifying the distribution free assumption of PAC learning, the class of effectively learnable languages can be significantly expanded.