PAC = PAExact and other Equivalent Models in Learning

Nader H. Bshouty [*]
Department of Computer Science
Technion, 32000
Haifa, Israel
bshouty@cs.technion.ac.il

Dmitry Gavinsky
Department of Computer Science
Technion, 32000
Haifa, Israel
demitry@cs.technion.ac.il

Abstract

The Probably Almost Exact model (PAExact) [BJT02] can be viewed as the Exact model relaxed so that

1. *The counterexamples to equivalence queries are distributionally drawn rather than adversarially chosen.*

2. *The output hypothesis is equal to the target with negligible error ($1/\omega(poly)$ for any poly).*

This model allows studying (Almost) Exact learnability of infinite classes and is in some sense analogous to the Exact-learning model for finite classes.

It is known that PAExact-learnable $\Rightarrow$ PAC-learnable [BJT02]. In this paper we show that if a class is PAC-learnable (in polynomial time) then it is PAExact-learnable (in polynomial time). Therefore,
PAExact-learnable = PAC-learnable.

It follows from this result that if a class is PAC-learnable then it is learnable in the Probabilistic Prediction model from examples with an algorithm that runs in polynomial time for each prediction (polynomial in log*(the number of trials)) and that after polynomial number of mistakes achieves a hypothesis that predicts the target with probability $1 - 1/2^{poly}$.*

We also show that if a class is PAC-learnable in parallel then it is PAExact-learnable in parallel.

Those and other results mentioned in the introduction answer the open problems posed in [B97, BJT02].

*This research was supported by the fund for promotion of research at the Technion. Research no. 120-025.

1. Introduction

In this paper we study two learning models that lie between the PAC and Exact models. The Probably Exact model (PExact) model, introduced by Bshouty [B97], is the Exact model (learning *exactly* the target function from equivalence queries) in which each counterexample to an equivalence query EQ_D is drawn according to a distribution D rather than maliciously chosen. When the concept class is infinite it is impossible to achieve an exact learning of the target. The Probably Almost Exact model (PAExact), introduced by Bshouty, Jackson and Tamon [BJT02], is the same as the PExact model but requires that the hypothesis produced by the learning algorithm have negligible ($1/\omega(poly)$) error.

The goal of learning is to make computers (learners) build (learn) a simulator (hypothesis) that simulates (this depends on the model) a phenomena (target function) from observed examples of inputs (the domain of the function) and outputs (the value of the function) assuming the input is generated according to some fixed distribution D. Since we (the learner) cannot efficiently (in polynomial time) learn *any* target function (by simple counting argument) we also assume that it is given a class C (concept class) of hypothesis and there is a hypothesis $h \in C$ that simulates the target. In the PAC learning model "simulate" means that the simulator can predict the output of a random input (according to D) with probability at least $1-\epsilon$. So PAC learnability of C implies that we can efficiently build a simulator h for such phenomena such that for a random input, h simulates it correctly with probability at least $1-\epsilon$. On the other hand Exact learnability of C implies that the learner can learn such simulator from any observed example and can update the simulator after each error and after small (polynomial) number of errors it gets a perfect simulator. That is, a simulator that does not error. The disadvantage of the PAC model is in that the simulator is not perfect. Especially when an error in the simulator cost us a fortune. On the other hand, the Exact model disadvantage is in that the concept class we learn (by counting arguments) must be finite and each concept in the class must have small representation. Also an Exact learner must be able to learn the function in all the domain points even points that will not be relevant for the simulation. The PAExact learning model is a model that addresses those disadvantages. It is a model in which the learner can update the simulator after each mistake and at the same time does not require a perfect simulator but a very close to a perfect one. That is, after small number (polynomial) of mistakes the simulator will err with very small ($1/\omega(poly)$) probability. In this model the class can be infinite and the simulator will be (with probability exponentially close to 1) perfect against any polynomial time algorithm. The PExact-model addresses the case when a PAExact-learning algorithm can achieve a probability of error that is less than $D(x)$ for any x in the domain. That is, the simulator is perfect after polynomial number of errors. Therefore, PExact-model studies exact learnability when the examples of the domain inputs are generated according to some distribution (e.g., uniform distribution).

In [BJT02], Bshouty et al. showed that Exact-learnable $\Rightarrow$ PExact-learnable $\Rightarrow$ PAExact-learnable $\Rightarrow$ PAC-learnable. They also showed that under the standard cryptographic assumption that one-way functions exist, PExact-learnable $\neq$ PAC-learnable (Based on the construction in [B94]).

In this paper we show that PAC-learnable = PAExact-learnable. We have two constructions. The first construction is similar to Schapire boosting algorithm [S90]. In Schapire boosting algorithm a PAC-learning algorithm that learns with error ϵ can be turned into a PAC-learning algorithm that PAC-learns with error $\epsilon' = 3\epsilon^2 + 2\epsilon^3$ that runs in polynomial time in $1/\epsilon$. Using Schapire construction the algorithm will not run in polynomial time when $\epsilon = 1/\omega(poly)$. In our construction we start from a PAC-learning algorithm that learns with error ϵ and runs in time T_1 and an PAExact-learning algorithm that learns with error η and runs in time T_2 and construct an PAExact-learning algorithm that learns with error $\epsilon\eta$ and runs in time $T_1 + cT_2$ for some constant $c > 2$. We use this construction $\log poly$ times and gets a polynomial time algorithm that PAExact-learns the class with error $1/poly^{O(\log poly)} = 1/\omega(poly)$.

The second construction is based on Mansour and McAllester [MA00] boosting algorithm and uses equivalence queries with randomized hypotheses. We use their booster with a hypothesis H^{η_0} that η_0-approximates the target and learn a new hypothesis $H^{\eta_0/2}$ that $\eta_0/2$-approximates the target in time $\log^5(1/\eta_0)$. Therefore in time $\log^6(1/\eta)$ we can learn a hypothesis with error η. That is, after polynomial time we learn a hypothesis with error less than $1/2^{poly}$. This shows that under bounded poly-bit distributions (distributions such that $D(x) = 0$ or $D(x) \geq 1/2^{poly}$) PExact-learnable = PAExact-learnable = PAC-learnable. It follows from this result that if a class is PAC-learnable then it is learnable in the Probabilistic Prediction model from examples with an algorithm that runs in polynomial time for each prediction (polynomial in log(the number of trials)) and that after polynomial number of mistakes achieves a hypothesis that predicts the target with probability $1 - 1/2^{poly}$. Specifically, after d mistakes we achieve a hypothesis with error

$$\eta = \frac{1}{2^{\min(d^{1/6},(d/w)^{1/5})}}$$

where w is the complexity of the PAC learning algorithm with constant ϵ. Previous algorithms run in polynomial time in the number of trials (and exponential in the number of mistakes d) to achieve such error.

For bounded poly-bit distributions it learns the target exactly after polynomial number of mistakes. For example, for the uniform distribution over $\{0,1\}^n$ our algorithm learns the target exactly in polynomial time after $n^5 \max(n, w)$ mistakes.

For each of the above models $\mathcal{M}$ we define the deterministic models D-$\mathcal{M}$ (e.g. D–PAC, D-Exact) where the learner is a deterministic algorithm. That is, the confidence parameter δ is provided only to account for uncertainty inherent in accessing examples through the probability distribution D and not to cover any randomness in the algorithm itself. In [BJT02] Bshouty et al. showed that D-Exact-learning = D-PExact-learning. It is easy to show that any randomized PAExact and PAC learning algorithm can be changed into a deterministic one. This can be done simply by replacing the random bits in the algorithm by bits that can be generated from the oracle.

The following diagrams summarize the current state of our knowledge:

$$\begin{tikzcd}
\text{D-Exact} \arrow[r, phantom, "="] \arrow[d, Rightarrow] & \text{D-PExact} \arrow[d, Rightarrow] & \begin{array}{c}\Rightarrow\\ \not\Leftarrow\end{array} & \text{D-PAExact} \arrow[r, phantom, "="] \arrow[d, equal] & \text{D-PAC} \arrow[d, equal] \\
\text{Exact} \arrow[r, Rightarrow] & \text{PExact} & & \text{PAExact} \arrow[r, phantom, "="] & \text{PAC}
\end{tikzcd}$$

and under bounded poly-bit distribution we have

$$\mathsf{Exact} \underset{\not\Leftarrow}{\Rightarrow} \mathsf{PExact}_{bpb} = \mathsf{PAExact}_{bpb} = \mathsf{PAC}_{bpb}.$$

Unsolved open problems still remain: (1) $\mathsf{Exact} = \mathsf{PExact}$?, (2) $\mathsf{Exact} = \mathsf{D\text{-}Exact}$?. Notice that if $\mathsf{PExact} = \mathsf{D\text{-}PExact}$ then $\mathsf{Exact} = \mathsf{PExact}$ and $\mathsf{Exact} = \mathsf{D\text{-}Exact}$.

In section 2 we define the learning models. In section 3 we give the first construction and prove that PAC=PAExact. In section 3 we give the second construction that achieve error $1/2^{poly}$.

2. Learning Models and Definitions

In learning, a *teacher* has a *target function* $f \in C$ where $f : X \to \{0,1\}$ and a *target distribution* D over X. The *learner* knows X and C but does not know the distribution D nor the function f.

The term "polynomial" and the *problem size* notation I_f that we will use in this paper depends on X, C and f and it can be different in different settings. For example, for Boolean functions $\{0,1\}^n \to \{0,1\}$, C is a set of formulas (e.g. DNF, Decision tree, etc.) and "polynomial" means $poly(n, size_C(f))$ where $size_C(f)$ is the minimal size formula in C that is equivalent to f. We always define I_f such that "polynomial" will mean $poly(I_f)$. Then I_f can be defined as the sum of the parameters in the poly. So in the above Boolean case $I_f = n + size_C(f)$. For infinite domains X the parameter n is usually replaced by the VC-dimension of the class $VC(C)$ and $I_f = VC(C) + size_C(f)$.

The learner can ask the teacher *queries* about the target. The queries we consider in this paper are:

Example Query according to D ($\mathbf{Ex}_D$) [V84] For the example query the teacher chooses $x \in X$ according to a distribution D and returns $(x, f(x))$ to the learner.

Equivalence Query (EQ) [A88] For the equivalence query the learner asks $\mathrm{EQ}(h)$ for some polynomial size circuit h. The teacher (which can be an adversary with unlimited computational power) chooses $y \in X_{f\Delta h} = \{x \mid f(x) \neq h(x)\}$ and returns y. If $X_{f\Delta h}$ is empty, the teacher answers "YES", indicating that h is equivalent to f.

Equivalence Query according to D ($\mathbf{EQ}_D$)[B97] For the equivalence query according to D the learner asks $\mathrm{EQ}_D(h)$ for some polynomial size circuit h. The teacher chooses $y \in X_{f\Delta h}$ according to the induced distribution of D on $X_{f\Delta h}$ and returns y. If $\Pr_D[X_{f\Delta h}] = 0$, the teacher answer "YES", indicating that h is equivalent to f over $Z_D(X) = \{x | D(x) \neq 0\}$.

The EQ_D can be extended to also handle random hypothesis. A random hypothesis $h_r : X \times R \to \{0,1\}$ is a polynomial size circuit where for an input $x_0 \in X$ it randomly uniformly chooses $r_0 \in R$ and returns $h_{r_0}(x_0)$. In that case, $\mathrm{EQ}_D(h_r(x))$ returns an output of the following

1. Choose $x_0 \in_D X$.
2. Choose $r_0 \in R$.
3. if $f(x_0) \neq h_{r_0}(x_0)$ then output$(x_0, f(x_0))$ else goto (1).

That is, each x_0 is received with probability

$$\frac{D(x_0)\Pr_r[h_r(x_0) \neq f(x_0)]}{\sum_x D(x)\Pr_r[h_r(x) \neq f(x)]}.$$

We say that the hypothesis h_r *η-approximates* f with respect to D if

$$E_r[\Pr_D[f(x) \neq h(x)]] \leq \eta$$

where here and elsewhere $\Pr_D$ denotes $\Pr_{x \in_D X}$.

The learning models we will consider in this paper are

PAC (Probably Approximately Correct)[V84] In the PAC learning model we say that an algorithm $\mathcal{A}$ of the learner *PAC-learns* the class C if for any $f \in C$, any distribution D and for any $\epsilon, \delta > 0$ the algorithm $\mathcal{A}(\epsilon, \delta)$ asks example queries according to D, Ex_D and with probability at least $1-\delta$, outputs a hypothesis $h \in C$ that ϵ-approximates f according to D. We say that C is *PAC-learnable* if there is an algorithm that PAC-learns C in time $poly(1/\epsilon, \log(1/\delta), I_f)$.

Exact (Exactly Correct) [A88] In the Exact model we say that algorithm $\mathcal{A}$ of the learner *Exact-learns* the class C if for any $f \in C$ and for any δ the algorithm $\mathcal{A}(\delta)$ asks equivalence queries and with probability at least $1-\delta$ outputs a polynomial size hypothesis h that is equivalent to f. We say that C is *Exact-learnable* if there is an algorithm that Exact-learns C in time $poly(\log(1/\delta), I_f)$.

PExact (Probably Exactly Correct) [B97] In the Probably Exact model we say that algorithm $\mathcal{A}$ of the learner *Probably Exact-learns* the class C if for any $f \in C$ and for any δ the algorithm $\mathcal{A}(\delta)$ asks equivalence queries with respect to D, EQ_D and with probability at least $1-\delta$ outputs a polynomial size circuit h that satisfies $\Pr_D[X_{f\Delta h}] = 0$. We say that C is *Probably Exact-learnable* if there is an algorithm that Probably Exact-learns C in time $poly(\log(1/\delta), I_f)$.

PAExact (Probably Almost Exactly Correct) [BJT02] In the Almost Exact model we say that algorithm $\mathcal{A}$ of the learner *Almost Exact-learns* the class C

if for any $f \in C$ and for any δ the algorithm $\mathcal{A}(\delta)$ asks equivalence queries with respect to D, EQ_D and with probability at least $1-\delta$ outputs a polynomial size circuit h that satisfies $\Pr_D[X_{f\Delta h}] = 1/\omega(poly(I_f))$. We say that C is *Almost Exact-learnable* if there is an algorithm that Almost Exact-learns C in time $poly(\log(1/\delta), I_f)$.

In the online learning model [L88] the teacher sends a point $x \in X$ to the learner and the learner has to predict $f(x)$. The learner returns to the teacher the prediction y. If $f(x) \neq y$ then the teacher returns "mistake" to the learner. The goal of the learner is to minimize the number prediction mistakes.

Online [L88] In the online model we say that algorithm $\mathcal{A}$ of the learner *Online-learns* the class C if for any $f \in C$ and for any δ, algorithm $\mathcal{A}(\delta)$ with probability at least $1-\delta$ makes bounded number of mistakes. We say that C is *Online-learnable* if the number of mistakes and the running time of the learner for each prediction is $poly(\log(1/\delta), I_f)$.

Probabilistic Prediction (PP) [HLW94] In the Probabilistic Prediction the points sent to the learner are chosen from X according to some distribution D. We say an algorithm $\mathcal{A}$ of the learner *η-PP-learns* the class C if for any $f \in C$ and for any δ the algorithm $\mathcal{A}(\delta)$ with probability at least $1-\delta$ after bounded number of mistakes can predict the answer with probability greater than $1-\eta$. We say that C is *η-PP-learnable* if the number of mistakes and the running time of the learner at each trial is $poly(\log(1/\delta), I_f)$.

It is known that Exact-learnable=Online-learnable [L88] and PAC-learnable=$1/poly(I_f)$-PP-learnable [HLW94]. In the same way one can prove that PAExact-learnable=$1/poly(I_f)$-PP-learnable. Our result in this paper implies that PAC-learnable=PAExact-learnable=$1/2^{poly(I_f)}$-PP-learnable.

3. Boosting in the PAExact-model

In this section we prove the following two Theorems

Theorem 1 *If C is PAC-learnable then C is PAExact-learnable.*

Theorem 2 *If C is PAC-learnable then C is $1/I_f^{\log I_f}$-PP-learnable.*

Our proof is similar to Shapire boosting algorithm [S90]. We first prove

Theorem 3 *Let C be a class of functions. Let $A(\epsilon,\delta)$ be a PAC-learning algorithm that learns the class C with error $\epsilon < 1/8$ and confidence δ in time $m_A(\epsilon,\delta,I_f)$. Let $B(\eta,\delta)$ be a PAExact-learning algorithm that learns the class C with error $\eta < \epsilon/4$ and confidence δ in time $m_B(\eta,\delta,I_f)$. Then, there is an PAExact-learning algorithm $R(B,A)$ that learns the class C with error $\epsilon\eta$ and confidence $1/4$ in time*

$$m_{R(B,A)}(\eta\epsilon, 1/8, I_f) \leq (c+1)m_B(\eta, 1/32, I_f) + m_A(\epsilon^2/2, 1/32, I_f),$$

where $c > 1$ and $m_B(\eta, 1/32, I_f)(2\epsilon)^c \leq 1/32$.

```
R(B,A)

1) Run $B(\eta, 1/32) \rightarrow h_1$
2) Run $A(\epsilon^2/2, 1/32) \rightarrow h_2$
        $\mathrm{Ex}_D \longleftrightarrow$ (is replaced with)
        with prob. 1/2 ask $\mathrm{EQ}_D(h_1)$
        with prob. 1/2 ask $\mathrm{EQ}_D(\bar{h}_1)$
3) Run $B(\eta, 1/32) \rightarrow g$
        $\mathrm{EQ}_D(g) \longleftrightarrow \mathrm{EQSimulate}(g)$
4) Output $(Maj(h_1, h_2, g))$
```

The algorithm $R(B,A)$.

Proof. We start by describing the algorithm $R = R(B,A)$. In the first step of R, it runs the algorithm $B(\eta, 1/32)$ to get a hypothesis h_1 in time $m_B(\eta, 1/32, I_f)$ where with probability at least $31/32$ we have

$$\Pr_D[h_1 = f] \geq 1 - \eta. \tag{1}$$

In the second step of R, the algorithm runs the PAC learning algorithm $A(\epsilon^2/2, 1/32)$ in time $m_A(\epsilon^2/2, 1/32, I_f)$ where each example is with probability $1/2$ an answer of $EQ_D(h_1)$ and with probability $1/2$ an answer of $EQ_D(\bar{h}_1)$. Algorithm A will learn a hypothesis h_2. In the third step in R the algorithm runs $B(\eta, 1/32)$ and replaces each query $EQ_D(g)$ with EQSimulate(g). See next page.

In the forth and last step of R, it takes the output g of the third step and outputs $Maj(h_1, h_2, g)$

Let $h_\vee = h_1 \vee h_2$ and $h_\wedge = h_1 \wedge h_2$. We now prove the following Claims

Claim 4 *With probability at least $31/32$ we have*

$$\Pr_D[h_\vee = 1 \wedge f = 0] \leq \eta + \epsilon^2, \quad \Pr_D[h_\vee = 0 \wedge f = 1] \leq \eta\epsilon^2,$$

```
EQSimulate(g)

1) i ← 0
2) Repeat EQ_D(Maj(h_1, h_2, g)) → x; i ← i + 1
3) Until h_1(x) ≠ h_2(x) or i = c.
4) If i = c Then output g and HALT.
        Else return(x).
```

$\Pr_D[h_\wedge = 1 \wedge f = 0] \le \eta\epsilon^2$, $\Pr_D[h_\wedge = 0 \wedge f = 1] \le \eta + \epsilon^2$, *and* $\Pr_D[h_1 \ne h_2] \le 2(\eta + \epsilon^2)$.

Claim 5 *If at any stage in step 3, EQSimulate halts and output g then with probability at least $29/32$*

$$\Pr_D[Maj(h_1, h_2, g) \ne f] \le \eta\epsilon.$$

Claim 6 *If algorithm R does not halt in some call to EQSimulate then Algorithm A in step 3 will run in time $(c+1)m_A(\eta, 1/32, I_f)$ and with probability at least $29/32$ outputs a hypothesis h such that*

$$\Pr_D[Maj(h_1, h_2, g) \ne f] \le \eta\epsilon.$$

Proof of Claim 1. In the second step of algorithm R, it takes $m_A(\epsilon^2/2, 1/32, I_f)$ examples S where each example in S is with probability $1/2$ an answer of $EQ_D(h_1)$ and with probability $1/2$ an answer of $EQ_D(\bar{h}_1)$. Then it runs $A(\epsilon^2/2, 1/16)$ on S.

Let D' be the distribution of the examples in the second step of R. Let $H_1 = [h_1 \ne f]$ and $H_2 = [h_2 \ne f]$. Then by (1) and the properties of A, with probability at least $30/32$,

$$\Pr_D[H_1] \le \eta \text{ and } \Pr_{D'}[H_2] \le \frac{\epsilon^2}{2}.$$

Now

$$\begin{aligned}\Pr_{D'}[H_2] &= \Pr_{D'}[H_2|H_1]\Pr_{D'}[H_1] + \Pr_{D'}[H_2|\bar{H}_1]\Pr_{D'}[\bar{H}_1] \\ &= \frac{1}{2}\Pr_D[H_2|H_1] + \frac{1}{2}\Pr_D[H_2|\bar{H}_1],\end{aligned}$$

and since $\Pr_{D'}[H_2] \le \epsilon^2/2$ we have

$$\Pr_D[H_2|H_1] \le \epsilon^2 \quad\text{and}\quad \Pr_D[H_2|\bar{H}_1] \le \epsilon^2.$$

Therefore, $\Pr_D[H_2] \le \epsilon^2$. Now we are ready to prove the claim. We have

$$\begin{aligned}&\Pr_D[h_\vee = 1 \wedge f = 0] \\ \le\ &\Pr_D[H_1] + \Pr_D[H_2] \le \eta + \epsilon^2\end{aligned}$$

and

$$\begin{aligned}\Pr_D[h_\vee = 0 \wedge f = 1] &= \Pr_D[h_1 = 0 \wedge h_2 = 0 \wedge f = 1] \\ &\le \Pr_D[H_1 \wedge H_2] \\ &= \Pr_D[H_1]\Pr_D[H_2|H_1] \le \eta\epsilon^2.\end{aligned}$$

In the same way it is easy to show that

$$\Pr_D[h_\wedge = 1 \wedge f = 0] \le \eta\epsilon^2,\ \Pr_D[h_\wedge = 0 \wedge f = 1] \le \eta + \epsilon^2.$$

Now we have

$$\begin{aligned}\Pr_D[h_1 \ne h_2] &= \Pr_D[h_\wedge \ne h_\vee] \\ &\le \Pr_D[h_\wedge = 0 \wedge f = 1] + \\ &\quad \Pr_D[h_\vee = 1 \wedge f = 0] \\ &\le 2(\eta + \epsilon^2).\diamond\end{aligned}$$

Proof of Claim 2. Suppose the algorithm halts in some call to EQSimulate(g) and let S be the set of examples obtained from $EQ_D(Maj(h_1, h_2, g))$ in step 2 of EQSimulate(g). Then $|S| = c$ and for every $x \in S$ we have $h_1(x) = h_2(x)$. Since x is a counterexample we also have $f(x) \ne h_1(x) = h_2(x)$ for every $x \in S$. Suppose g is not a "good" hypothesis, i.e., $\Pr_D[Maj(h_1, h_2, g)] \ge \eta\epsilon$. The probability that EQSimulate returns g is

$$\Pr_S[f \ne h_1 = h_2\ |Maj(h_1, h_2, g) \ne f].$$

Denote $M = [Maj(h_1, h_2, g) \ne f]$. Then

$$\begin{aligned}&\Pr_S[f \ne h_1 = h_2\ |M] \\ =\ &(\Pr_D[f \ne h_1 = h_2 \mid M])^c \\ =\ &\left(\frac{\Pr_D[f \ne h_1 = h_2]}{\Pr_D[M]}\right)^c \\ \le\ &\left(\frac{\Pr_D[h_\vee = 0 \wedge f = 1] + \Pr_D[h_\wedge = 1 \wedge f = 0]}{\eta\epsilon}\right)^c \\ =\ &(2\epsilon)^c.\end{aligned}$$

Therefore, the probability that some EQSimulate returns a bad output is at most

$$m_B(\eta, 1/32, I_f)(2\epsilon)^c \le \frac{1}{32}.$$

Since failing in the first two steps of the algorithm can happen with probability at most $2/32$, the probability that g is good is at least $29/32$.$\diamond$

Proof of Claim 3. All the answers x of the equivalence queries in EQSimulate satisfies $f(x) \ne$

$Maj(h_1(x),h_2(x),g(x))$ and $h_1(x)\neq h_2(x)$. Therefore, $Maj(h_1(x),h_2(x),g(x))=g(x)$ and x is a counterexample of $EQ_{D'}(g)$ where D' is the induced distribution D on $\{x|h_1(x)\neq h_2(x)\}$. Therefore, if the algorithm does not halt on some EQSimulate then with probability at least $31/32$,

$$\Pr_D[g\neq f|h_1\neq h_2]\leq \eta.$$

Therefore, with probability at least $29/32$ we have

$$\begin{aligned} & \Pr_D[Maj(h_1,h_2,g)\neq f] \\ \leq\ & \Pr_D[h_1=h_2\neq f]+\Pr_D[g\neq f|h_1\neq h_2]\Pr_D[h_1\neq h_2] \\ =\ & \Pr_D[h_\vee=0\wedge f=1]+\Pr_D[h_\wedge=1\wedge f=0] \\ & +\eta\Pr_D[h_1\neq h_2] \\ \leq\ & 2\eta\epsilon^2+\eta(2(\eta+\epsilon^2))=2\eta^2+4\eta\epsilon^2\leq \eta\epsilon.\diamond \end{aligned}$$

We now show how to boost δ. We denote by $R^{\star k}$ the algorithm that runs R k times and takes the majority function $Maj(h_i)$ of the output hypotheses.

Theorem 7 *Let R be an PAExact-learning algorithm for C that runs in time $m_R(\eta,\delta,I_f)$. Then for any odd integer k*

$$m_{R^{\star k}}\left(\frac{k+1}{2}\eta,(4\delta)^{\frac{k+1}{2}},I_f\right)\leq km_R(\eta,\delta,I_f).$$

Proof. We run R k times and take the majority function $Maj(h_i)$ of the outputs. It is clear that if at least $(k+1)/2$ of the outputs h_i are good, i.e. satisfies $\Pr_D[h_i\neq f]\leq\eta$, then $\Pr_D[Maj(h_i)\neq f]\leq (k+1)\eta/2$. Therefore,

$$\begin{aligned} & \Pr\left[\Pr_D[Maj(h_i)\neq f]>\frac{k+1}{2}\eta\right] \\ \leq\ & \Pr\left[\frac{k+1}{2}\text{ of }\ \Pr_D[h_i\neq f]\text{ are greater than }\eta\right] \\ \leq\ & \binom{k}{\frac{k+1}{2}}\delta^{\frac{k+1}{2}}\leq(4\delta)^{\frac{k+1}{2}}.\diamond \end{aligned}$$

Now we are ready to prove our main result. Let $A(\epsilon,\delta)$ be a polynomial time PAC-learning algorithm for C that runs in time $m_A(\epsilon,\delta,I_f)$. Suppose

$$m_A(1/I_f^2,1/32,I_f)\leq I_f^b,$$

for some constant b. Define $\eta=\epsilon^2/2$ and $\epsilon=1/(5I_f)$, and c a constant such that $(c+1)-\log(9(c+2))\geq b$. Define a sequence of PAExact-learning algorithms $B_0,B_1,\ldots,B_{\log I_f+1}$ where

$$B_0=A(\epsilon^2/2,1/32),\ B_{i+1}=R(B_i,A)^{\star 9}.$$

Now it is easy to prove by induction

Lemma 8 *For $\alpha=9(c+2)$ we have*

$$m_{B_{i+1}}((5\epsilon)^{i+1}\epsilon,1/32,I_f)\leq\alpha^i m_A(\epsilon^2/2,1/32,I_f).$$

Proof. For $i=-1$ we have $m_{B_0}(\epsilon,1/32,I_f)=m_A(\epsilon^2/2,1/32,I_f)$. Now we have

$$\begin{aligned} & m_{B_{i+1}}((5\epsilon)^{i+1}\epsilon,1/32,I_f) \\ \leq\ & 9m_{R(B_i,A)}(\epsilon(5\epsilon)^i\epsilon,1/8,I_f) && (2)\\ \leq\ & 9((c+1)m_{B_i}((5\epsilon)^i\epsilon,1/32,I_f)+ \\ & m_A(\epsilon^2/2,1/32,I_f)) && (3)\\ \leq\ & 9((c+1)\alpha^{i-1}m_A(\epsilon^2/2,1/32,I_f)+ \\ & m_A(\epsilon^2/2,1/32,I_f)) && (4)\\ \leq\ & \alpha^i m_A(\epsilon^2/2,1/32,I_f) \end{aligned}$$

In (2) we used Theorem 7. In (3) we used Theorem 3. In (4) we used the induction hypothesis for i. We now show that the condition in Theorem 3 is true. We have

$$\begin{aligned} & m_{B_i}((5\epsilon)^i,1/32,I_f)(2\epsilon)^c \\ \leq\ & \alpha^{i-1}m_A(\epsilon^2/2,1/32,I_f)(2\epsilon)^c \\ =\ & \alpha^{i-1}I_f^b\frac{1}{I_f^c} \\ \leq\ & I_f^{b-c+\log\alpha}<\frac{1}{I_f}<\frac{1}{32}. \end{aligned}$$

Now we have

$$\begin{aligned} & m_{B_{\log I_f+1}}(1/I_f^{\log I_f},1/32,I_f) \\ \leq\ & I_f^{\log\alpha}m_A(1/(30I_f^2),1/32,I_f)=poly(I_f). \end{aligned}$$

Therefore $B_{\log I_f+1}$ is a PAExact-learning algorithm that runs in polynomial time and achieve error $1/I_f^{\log I_f}$. This complete the proof of Theorem 1 and 2.$\diamond$

3.1. Parallel PAExact learning

In this subsection we show

Theorem 9 *If a class C is PAC-learnable in parallel then C is PAExact-learnable in parallel.*

Proof Sketch. Let A be a parallel PAC-learning algorithm for C. For a PAExact-learning algorithm B we denote by $PC(B)$ the parallel complexity of the algorithm. Let B_i as defined before. Then

$$PC(B_{\log\log I_f})=(\log I_f)PC(A).$$

and as before this algorithm achieve error $1/I_f^{\log\log I_f}$.$\diamond$

4. PAExact Learning with Error $1/2^{poly}$

In this section we prove the following

Theorem 10 *If C is PAC-learnable then C is $1/2^{poly}$-PP-learnable.*

4.1 Mansour-McAllester Booster

In this section we give an overview of Mansour McAllester booster. Mansour and McAllester booster in [MA00] is based on other boosting result introduced earlier by Kearns and Mansour in [KM96]. We denote the algorithm by MA.

The boosting algorithm MA constructs its final hypothesis h in the form of a branching program (further denoted as BP). The root and the internal nodes of this BP are marked with boolean function over X, so that for any $x \in X$ the corresponding "path" from the root to one of the leaves of the BP may be found. This fact puts each $x \in X$ into correspondence with one of the BP's leaves. The leaves are marked with constant boolean values.

```
MA(d)
    define a BP $T_0$ a single node $n_{1,0}$
    for $j$ from 0 to $d-1$ do:
        set $w_{j+1}$, as specified in [MA00]
        define a BP $T_{j+1}$ to be $T_j$ with nodes
        $n_{1,j+1}, ..., n_{w_{j+1},j+1}$ added
        for $i$ from 1 to $w_{j+1}$ do:
            define $S_{i,j} \triangleq$ {instances reaching $n_{i,j}$}
            define $S^{(0)}_{i,j} \triangleq S_{i,j} \cap \{x|f(x)=0\}$,
            $S^{(1)}_{i,j} \triangleq S_{i,j} \cap \{x|f(x)=1\}$
            define
            $D'_{S_{i,j}}(x) \triangleq \begin{cases} 0 & x \notin S_{i,j} \\ \frac{D(x)}{2 \cdot D(S^{(0)}_{i,j})} & x \in S^{(0)}_{i,j} \\ \frac{D(x)}{2 \cdot D(S^{(1)}_{i,j})} & x \in S^{(1)}_{i,j} \end{cases}$
            call weak learner $W$, with dist. $D'_{S_{i,j}}$
            the returned hypothesis marks $n_{i,j}$
        end-for
        install edges from $n_{i,j}$ to $n_{i,j+1}$,
        as specified in [MA00]
    end-for
    return $T_d$
```

In general, by $n_{i,j}$ we mean the i'th node from layer j . By $S_{i,j}$ we denote the sets of instances from X which arrive at $n_{i,j}$ and by $h_{i,j}$ we denote the predicate which marks $n_{i,j}$.

At stage j in the loop of the booster (see algorithm $MA(d)$) the algorithm builds the nodes at the jth level of the BP. It defines for each node $n_{i,j}$ a distribution $D'_{S_{i,j}}$ on the set of points $S_{i,j}$ that arrives at this node. Originally ([KM96]), this distribution was addressed as a *balanced distribution*. It gives nonzero probabilities only to instances from S, so that 1/2 of the probabilistic weight falls to "zeros" and 1/2 of the weight falls to "ones" of the target function f. Then the algorithm marks the node $n_{i,j}$ with the hypothesis $h_{i,j}$ returned by the weak learner W.

In our case, the target concept class is PAC-learnable, therefore we obviously may assume that W, being faced with some instance distribution $D'_{S_{i,j}}$, produces a hypothesis that c-approximates f w.r.t. $D'_{S_{i,j}}$ with some given *constant accuracy* c. To achieve error η the depth and width of the BP will be

$$d(\eta) = O(\ln(1/\eta)). \tag{5}$$

Note that in [MA00] this values (the depth and the size) are not defined as functions of η, instead, the definition depends on certain measure of the algorithm progress (namely, I, which we will consider later). On the other hand, the booster may stop adding new layers to BP as soon as the required accuracy (above $1-\eta$) may be achieved by simply choosing "right" constant values for all the nodes of the last added layer. Suppose that S is the set of instances arriving at certain leaf of the constructed hypothesis h (i.e., at a node without outgoing edges), then obviously, the choice of the marking value for that leaf must be done in accordance with the classification of the majority of the points which belong to S, if we are interested to lower the classification error of h. In each constructed BP layer j ,it holds that the width

$$w_j = O(\ln(1/\eta)). \tag{6}$$

Together, expressions (5) and (6) provide us with the following fact: If we address by h both the final hypothesis itself and its representation, as constructed by MA, and by $|h|$ we denote the corresponding size of the representation, then it holds that:

$$|h| = O(\ln^2(1/\eta)). \tag{7}$$

To install the edges from the nodes $n_{i,j}$ in level j to the nodes $n_{i,j+1}$ in level $j+1$ of the BP, they had to estimate the value of $I(S)$ where $S = S_{i,j}$ and

$$q(S) \triangleq \Pr_D[f(x)=1|x \in S],$$
$$I(S) \triangleq 2\sqrt{(1-q(S))q(S)},$$

4.2 Running MA with EQ$_D$ Oracle

We now want to run the MA algorithm with the EQ$_D$ oracle to achieve exponentially small error.

We have the following problems

1. How to provide the distribution $D'_{S_{i,j}}$ to the weak learner?
2. How to estimate $I(S)$?

To be able to solve both problems we let our PAExact-learning algorithm runs in stages. At each stage the algorithm has some hypothesis H^η that η-approximates the target function f and the learner goal is to learn a new hypothesis $H^{\eta/2}$ that $\eta/2$-approximates f. We will show that using H^η and the EQ$_D$ oracle it is possible to generate examples according to the balanced distribution $D'_{S_{i,j}}$ and to estimate $I(S_{i,j})$ as long as the weights of $S^{(0)}_{i,j} = \{x \in S_{i,j}, f(x) = 0\}$ and $S^{(1)}_{i,j} = \{x \in S_{i,j}, f(x) = 1\}$ according to D are greater than $\eta/(4t)$ where t is the size of the final hypothesis (which can be estimated before the running of the algorithm). Once $S^{(\xi)}_{i,j}$ weight is less than $\eta/(4t)$, for some $\xi \in \{0,1\}$, we turn node $n_{i,j}$ to a leaf node and label it with ξ. The total error of the hypothesis from those nodes is at most $\eta/4$. Our boosting algorithm will generate a BP of depth $O(\log(1/\eta))$. This guarantee that the error from the nodes at the last level is at most $\eta/4$. Together, the generated hypothesis achieve error at most $\eta/2$.

Therefore, to each level of the BP we will add two nodes. The 0- and 1-node. Those nodes are for sets that cannot be samples with the balanced distribution.

4.3. Sampling with EQ$_D$-Oracle

In this section we show how to generate examples from $S_{i,j}$ according to the balanced distributions $D'_{S_{i,j}}$ for the weak learner. This will solve the first problem. The balanced distribution is defined as follows: If we denote by $S = S_{i,j}$ then the *balanced distribution* D'_S is

$$D'_S(x) = \begin{cases} 0 & x \notin S \\ \frac{D(x)}{2\cdot D(\{x\in S, f(x)=0\})} & x \in S, f(x) = 0 \\ \frac{D(x)}{2\cdot D(\{x\in S, f(x)=1\})} & x \in S, f(x) = 1 \end{cases}.$$

We first show how an EQ$_D$-Oracle may be used in order to sample efficiently from superpolynomially small subsets of X. Suppose that a certain subset Y of X is defined by means of a polynomial time computable boolean function (or predicate) ϕ_Y as follows:

$$Y = \{y \in X | \phi_Y(y) = 1\}.$$

Suppose that there exists some hypothesis $h(x)$, and the learner's quest is to sample efficiently counterexamples for h w.r.t. the target function f and D, but *only those coming from* Y. We assume that Y is "small", so that repeated calls of $EQ_D(h)$ until a counterexample from Y is received would be inefficient. Suppose further that the learner has a hypothesis $H^\eta(x)$ which η-approximates f w.r.t. D. Instead of repeated asking $EQ_D(h)$ we will ask $EQ_D(h_Y)$ where

$$h_Y(x) \triangleq \begin{cases} H^\eta(x) & \phi_Y(x) = 0 \ (i.e.,\ x \notin Y) \\ h(x) & otherwise \end{cases}.$$

A counterexample produced by the EQ$_D$-oracle as a response to EQ$_D(h_Y)$ comes from Y with probability at least

$$\begin{aligned} & \Pr_D[x \in Y | h_Y \neq f] \\ = & \Pr_D[h \neq f, x \in Y | h_Y \neq f] \\ = & \frac{\Pr_D[h \neq f,\ x \in Y]}{\Pr_D[h_Y \neq f]} \\ = & \frac{\Pr_D[h \neq f,\ x \in Y]}{\Pr_D[h \neq f,\ x \in Y] + \Pr_D[H^\eta \neq f,\ x \notin Y]} \\ \geq & \frac{\Pr_D[h \neq f,\ x \in Y]}{\Pr_D[h \neq f,\ x \in Y] + \eta}. \end{aligned}$$

The above construction makes it possible to receive efficiently a counterexample to h coming from the subset Y when $\Pr_D[h(x) \neq f(x), x \in Y]$ is polynomially smaller than η. However, η itself, may be arbitrarily small.

Still more important, the counterexample is generated according to the target distribution D.

Lemma 11 *Let $\phi_Y(x)$ be a polynomial time computable boolean predicate over X and let $Y = \{y \in X | \phi_Y(y) = 1\}$. Suppose that we have a hypothesis $H^\eta(x)$ which is at least η-approximates f w.r.t. D and that we can make queries to an EQ_D-oracle.*

Then a counterexample w.r.t. f for any hypothesis h may be received so that this counterexample belongs to Y and is generated according to D. In average, the time and the number of queries to the oracle required is

$$O\left(1 + \frac{\eta}{\Pr_D\left[h(x) \neq f(x),\ x \in Y\right]}\right).$$

This can be done using repeated calls of $EQ_D(h_Y(x))$ where

$$h_Y(x) \triangleq \begin{cases} H^\eta(x) & \phi_Y(x) = 0 \ (i.e.,\ x \notin Y) \\ h(x) & otherwise \end{cases} . \diamond$$

Now we consider the problem of generating balanced distributions for the weak learner. We try to do that using the above techniques.

The following Corollary is based on Lemma 11.

Corollary 12 *Let $S \subseteq X$ be a set of instances. Let H^η be a hypothesis which at least η-approximates f w.r.t. D, and let η' be some value smaller than η. Then there exists a procedure which with probability at least $1-\delta$ either produces an example according to the balanced distribution D'_S or returns a constant value $a \in \{0,1\}$ such that*

$$\Pr_D[f(x) \neq a,\ x \in S] \leq \eta'.$$

The average time complexity of the procedure is polynomial in $\ln(1/\delta)$ and η/η'.

Proof Sketch. Using H^η, we may construct the following two hypotheses: For $a \in \{0,1\}$

$$h_S^{(a)} \triangleq \begin{cases} H^\eta(x) & x \notin S \\ 1-a & otherwise \end{cases} .$$

If we pass $h_S^{(0)}$ to the EQ$_D$-oracle many times, we will accept an example from $\{x \in S\,|f(x) = 0\}$ after the average of

$$O\left(1 + \frac{\eta}{D(\{x \in S\,|f(x) = 0\})}\right)$$

calls to the oracle.$\diamond$

We will set

$$\eta' = \frac{\eta}{4t}. \tag{8}$$

If a balanced distribution cannot be produced then by Corollary 12 for some $a \in \{0,1\}$ we have

$$\Pr_D[f(x) \neq a, x \in S] \leq \frac{\eta}{4t}.$$

Therefore labling the node corresponding to S with a constant $1-a$ will add error at most $\eta/4t$. Labling all such nodes will add error at most $\eta/4$.

4.4 Estimating $I(S)$

In this section we solve the second problem. We show that using EQ$_D$ with a randomized hypotheses, we can estimate the value $I(S)$. We prove

Lemma 13 *Let $S \subseteq X$ and H^η be a hypothesis approximating f, s.t. it holds:*

$$D\left(\{x \in X \setminus S | f(x) \neq H^\eta(x)\}\right) \leq \eta.$$

Then using a EQ_D-oracle with randomized hypothesis, the values of $q(S)$ and of $1-q(S)$ may be found with constant multiplicative accuracy in time polynomial in $\log(1/\min(\{q(S), 1-q(S)\}))$, in $\eta/D(\{x \in S|f(x) = 1\})$ and in $\eta/D(\{x \in S|f(x) = 0\})$.

Proof Sketch. Let us define a random hypothesis

$$h_s^p(x) \triangleq \begin{cases} H^\eta(x) & x \notin S \\ 1 & \text{with probability } p \text{ when } x \in S \\ 0 & \text{otherwise} \end{cases} ,$$

for any $p \in [0,1]$. Let us check what happens when h_s^p is sent to EQ$_D$: First assume that a counterexample x_0 comes from S, then not knowing what $f(x_0)$ is, we may say that:

$$\frac{\Pr[f(x_0) = 0]}{\Pr[f(x_0) = 1]} = \frac{(1-q(S))p}{q(S)(1-p)}, \tag{9}$$

which follows from the above definition of h_s^p.

If for some $p = p_0$ we could say that

$$\frac{\Pr[f(x_0) = 0]}{\Pr[f(x_0) = 1]} = 1 \pm \Delta, \tag{10}$$

then, as follows from Equation (9), that would mean that

$$\tfrac{1-q(S)}{q(S)} = (1 \pm \Delta) \cdot \tfrac{1-p_0}{p_0}.$$

We use binary search (starting from $p = 1/2$) in order to find a value for p, such that condition (10) holds.

It remains to check how long will it take to receive a counterexample from S for h_s^p. Without loss of generality, we assume that $q(S) < 1/2$. In this case the value of parameter p will always remain equal or below $1/2$ during the binary search. Therefore, for any returned counterexample x_0 it holds:

$$\begin{aligned}
&\Pr[x_0 \in S] = \\
&\tfrac{D(\{x \in S|f(x) \neq h_s^p(x)\})}{D(\{x \in S|f(x) \neq h_s^p(x)\}) + D(\{x \in X \setminus S|f(x) \neq h_s^p(x)\})} \\
&\geq \tfrac{D(\{x \in S|f(x)=0, h_s^p(x)=1\}) + D(\{x \in S|f(x)=1, h_s^p(x)=0\})}{\eta + D(\{x \in S|f(x)=0, h_s^p(x)=1\}) + D(\{x \in S|f(x)=1, h_s^p(x)=0\})} \\
&\geq \tfrac{D(\{x \in S|f(x)=0, h_s^p(x)=1\}) + D(\{x \in S|f(x)=1\}) \cdot 1/2}{\eta + D(\{x \in S|f(x)=0, h_s^p(x)=1\}) + D(\{x \in S|f(x)=1\}) \cdot 1/2} \\
&\geq \tfrac{D(\{x \in S|f(x)=1\})}{2\eta + D(\{x \in S|f(x)=1\})}.
\end{aligned}$$

The result follows.◇

4.5 The Complexity of the Algorithm

The algorithm runs in $\log(1/\eta)$ stages and at each stage it builds a BP of size $\log^2(1/\eta)$. To find the weak hypothesis at each node $n_{i,j}$ we run the weak learner with $w = poly(I_f)$ examples where in the worst case each example can be received with $\log^2(1/\eta)$ calls to the EQ$_D$. To estimate $I(S)$ we again use $\log^2(1/\eta)$ calls to EQ$_D$ and the binary search will take in the worst case $\log(1/\eta)$ steps. Therefore the total complexity is

$$\log^5(1/\eta)w + \log^6(1/\eta).$$

Therefore, after d mistakes we achieve a hypothesis with error

$$\eta = \frac{1}{2^{\min(d^{1/6},(d/w)^{1/5})}}$$

5 Open Problems

Here we list some open problems.

1. Is PExact=Exact?
2. Is D-Exact=Exact?
3. Is D-PExact=PExact? Notice that if D-PExact=PExact then D-Exact=Exact and PExact=Exact.
4. Is PAC=$1/2^{poly}$-PP with deterministic hypotheses?
5. Find a more efficient Booster.
6. In [HLW94] it is shown that in the PP-learning model the error $\eta \geq \frac{1}{2^{O(d/VC)}}$ where VC is the VC-dimension of C and d is the number of mistakes. They also gave a double exponential time algorithm that achieve this error. Then they gave an exponential time algorithm (in d) that achieve error $\eta \leq \frac{1}{2^{O((d/VC)^{1/2})}}$ assuming we can solve the consistent hypothesis problem (find a hypothesis $h \in C$ that is a consistent with the sample). In the second construction we showed that after d mistakes we achieve (in polynomial time in d) a hypothesis with error

   $$\eta = \frac{1}{2^{\min(d^{1/6},(d/VC)^{1/5})}}$$

   assuming we can solve the consistent hypothesis problem. Can we achieve a better error?
7. Can we PAExact-learn DNF with membership queries under the uniform distribution?

References

[A88] D. Angluin. Queries and concept learning. *Machine Learning 2(4), pp. 319-342*, 1988.

[B94] A. Blum. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM Journal on Computing 23(5), pp. 990-1000*, 1994.

[B97] N. Bshouty. Exact learning of formulas in parallel. *Machine Learning 26, pp. 25-41*, 1997.

[BJT02] N. Bshouty, J. Jackson and C. Tamon. Exploring learnability between exact and PAC. *Proceedings of the 15th Annual Conference on Computational Learning Theory,* , 2002.

[HLW94] D. Haussler, N. Littlestone and M. Warmuth. Predicting 0,1-functions on randomly drawn points. *Information and Computation 115, pp. 248-292*, 1994.

[KM96] M. Kearns and Y. Mansour. On the Boosting Ability of Top-Down Decision Tree Learning Algorithms. *Proceedings of the 28th Symposium on Theory of Computing, pp. 459-468*, 1996.

[L88] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold learning algorithm. *Machine Learning 2(4), pp. 285-318*, 1988.

[MA00] Y. Mansour and D. McAllester. Boosting using Branching Programs. *Proceedings of the 13th Annual Conference on Computational Learning Theory, pp. 220-224*, 2000.

[S90] R. E. Schapire. The strength of weak learnability. *Machine Learning 5(2), pp. 197-227*, 1990.

[V84] L. Valiant. A theory of learnable. *Communications of the ACM 27(11), pp. 1134-1142*, 1984.