人工知能A

Topic 8: PAC learning, SVM, boostingなど

8

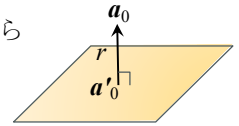
SVMとBoosting

- PAC学習の概要、SVMとアンサンブル学習のBaggingやBoosting (AdaBoost) の基礎について学ぶ。
  - よく見かけるAdaBoostはPAC学習の研究が基本になっています。
- 講義の内容
  - 数学の復習など少し
  - PAC学習（ここは考え方だけ）、
  - Support Vector Machine (SVM)
  - Boosting (Adaboost)
- より詳細は各自で自習するか、大学院で必要に応じて習得する。ここではその準備。

3 数学の復習と準備

数学の復習(1)

- n次元ベクトル空間の（超）平面
  $$w_1x_1 + \cdots + w_nx_n = b$$
  と平面上にない点$\boldsymbol{a}_0 = (a_{01}, \ldots, a_{0n})$を考える。$\boldsymbol{a}_0$から平面に下ろした垂線との交点を$\boldsymbol{a}'_0$, 長さを$r$とする。



- $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{w} = (w_1, \ldots, w_n)$とおけば、この平面は$f(\boldsymbol{x})$=0。ただし$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b$とおいた。
- $\boldsymbol{w}$の単位ベクトルは、$\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$となる。ただし、$\| \boldsymbol{w} \| = \sqrt{w_1^2 + \cdots + w_n^2}$
- したがって、$\boldsymbol{a}_0 = r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} + \boldsymbol{a}'_0$と書ける。$f(\boldsymbol{a}_0) > 0$のときは$r > 0$。逆方向の場合は、$r$が負の数と考えればよい。
- $f(\boldsymbol{a}_0) = \boldsymbol{w}\boldsymbol{a}_0^T - b = \boldsymbol{w}\boldsymbol{a}_0'^T - b + \frac{r}{\|\boldsymbol{w}\|}\boldsymbol{w}\boldsymbol{w}^T$となり、赤の部分は0 (平面上なので) 青は、$\| \boldsymbol{w} \|^2$となるので、$r = f(\boldsymbol{a}_0)/\| \boldsymbol{w} \|$と書ける。

数学の復習(2)

- 関数のベクトル$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^T$ ただし、
$$f_k: \mathbb{R}^n \to \mathbb{R} \text{ for } 1 \le k \le m$$
を考える。$\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$である。

  m×*n*行列$\boldsymbol{A}$と*m*列のベクトル$\boldsymbol{b}$が存在して、
$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}^T + \boldsymbol{b}$$
と表せるとき、$\boldsymbol{f}(\boldsymbol{x})$をアフィン関数(*affine function*)という
  - つまり線形関数です（まれですが、$\boldsymbol{b}=0$のとき、特に線形関数と呼ぶこともあります）。
- 実数の*n*×*n*行列 $\boldsymbol{A}=(a_{ij})$ が半正定値 (*positive semi-definite* or *nonnegative definite*) とは、0でない任意のベクトル$\boldsymbol{c}$に対し、$\boldsymbol{c}\boldsymbol{A}\boldsymbol{c}^T \geqq 0$となること。
  - 例：$\begin{pmatrix} 9 & 6 \\ 6 & 4 \end{pmatrix}$
$$\because (x\ y)\begin{pmatrix} 9 & 6 \\ 6 & 4 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = 9x^2 + 12xy + 4y^2 = (3x + 2y)^2 \geqq 0$$

数学の準備

- このあとの「数学の準備」は紹介だけ。
  - もしかするとオートマトンとかラムダ計算などで知っている人がいるかもしれません。
  - 数学科とか応用数理学科の学生であれば、数学基礎論の入門部分でやると思います。

数学の準備(1)

- 帰納的関数 (*recursive functions*) とは、（直感的には）プログラムとして書ける関数のこと。
  - 再帰的関数、計算可能関数とも言う。
  - 同様な概念が、チューリング機械やラムダ計算 (つまりプログラム言語の SchemeやLISP) でも定義でき、これらは同義であることが知られている。
- 原始帰納的関数 (*primitive recursive functions*)
  - 原始再帰的関数とも言う。
  - ここでは制御構造がloopのみで書かれたプログラムでかける関数のことと考えてよい。

数学の準備(2)

- *n*次元実数空間上の（プログラムとして書ける）実数値関数
$$f: \mathbb{R}^n \to \mathbb{R}$$
に対し$C_+(f) = \{\boldsymbol{x} \in \mathbb{R}^n | f(\boldsymbol{x}) > 0\}, C_-(f) = \{\boldsymbol{x} \in \mathbb{R}^n | f(\boldsymbol{x}) < 0\}$と$\mathbb{R}^n$を分割することを考える。
- 上記の*f*から容易に以下のようなプログラムを作成できる。
$$g(\boldsymbol{x}) = \begin{cases} 1 & \text{for } \boldsymbol{x} \in C_+(f) \\ -1 & \text{for } \boldsymbol{x} \in C_-(f) \end{cases}$$
- このような$g(\boldsymbol{x})$を判別関数（分類関数）と呼ぶ。(=0の部分は適当にどちらかの分割領域に組込でもよい)
- 特にfを超平面$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b$とすると、$C_\pm(f)$はこの面で分割された領域を表す。
- なお $\mathrm{sgn}(h(\boldsymbol{x})) = \begin{cases} 1 & \text{for } h(\boldsymbol{x}) \geq 0 \\ -1 & \text{for } h(\boldsymbol{x}) < 0 \end{cases}$ と定義 ($h(\boldsymbol{x})=0$のときの定義は状況によって)

9 PAC学習

まずだいたいのお話から。SVM（特にソフトマージンSVM）の理論的基礎やAdabootなどが生まれる背景には、計算論（つまりNP-Completeとかいうやつ）にもとづく学習理論がありました。

計算論に基づいた帰納推論と学習理論

- お話として歴史的背景から。
  - 先に参考文献
    - [Gold67] E M. Gold, Language identification in the limit, *Information and Control*, 1967.
    - [Blum75] L. Blum and M. Blum, Toward a mathematical theory of inductive inference, *Information and Control*, 1975.
    - [Valiant84] L. G. Valiant. 1984. A theory of the learnable. *Commun. ACM* Vol.27, 11, 1984.
- 「帰納推論・帰納学習 (indictive reasoning/learning, induction)」とは、いくつかの具体例から一般的な法則や理論を導き出す過程である。
- 学習の一形式と考えられる。

計算論に基づいた帰納推論

- まずは何をしようとしているか。
  - C, Java, pythonなどでプログラムされた関数$F(x)$がある。
    $$F\colon \mathbb{R} \to \mathbb{R}$$
    入力と出力の値$(x_i, F(x_i))$をいくつか観測して、そのプログラムの中身を知ることはできないか？
  - 簡単のため入力は一変数、出力は$+1, -1$のみとします。
- プログラムで書かれた関数$F(x)$のことを帰納的関数と呼びました。言い換えると。
  - ± 1の値をとる帰納関数 $F(x)\colon \mathbb{R} \to \mathbb{R}$の入出力のペアをいくつか観測して、帰納関数の定義を知りたい。

計算論に基づいた帰納推論

- 極限による同定（帰納推論の形式的な定義）
  - $\mathcal{R}$を $F\colon X \to \{-1, 1\}$ $(F \in \mathcal{R})$なるプログラム (帰納的関数) の集合とする。 Xは共通の定義域。
  - プログラム $F \in \mathcal{R}$が生成する学習データの無限の列 $(x_1, y_1), (x_2, y_2), \ldots$.を仮定する。ここで$F(x_i) = y_i$, ただし $y_i = -1$ or 1。 $y_i = 1$ の学習データを正例、$y_i = -1$のデータを負例と呼ぶこともある。つまり教師あり学習。
  - 学習データの無限の列が正例のみのも考えられる（負例はあるのだが、あえて与えない）。

計算論に基づいた帰納推論

- 極限による同定（帰納推論の形式的な定義）
  - 定義 (Identification in the limit, Gold 1964)：
    $\forall F \in \mathcal{R}$に対し、正の整数$n > 0$に対し、アルゴリズム（プログラム）$\mathcal{A}$に、$(x_1, y_1), \ldots, (x_n, y_n), \ldots$ を順に入力すると$\mathcal{A}$は帰納的関数$h_1, h_2, \ldots \in \mathcal{R}$を順に出力し、
    $h_n(x_j) = y_j$ for $1 \leq \forall j \leq n$となるとする。
    このとき十分大きい自然数Nが存在し、$\forall n \geq N$なら$\mathcal{A}$は同じ$h_n = h_N$を出力し続けるとき$\mathcal{A}$ は$\mathcal{R}$を極限で同定可能（学習可能）という。

計算論に基づいた帰納推論

- 当初は計算論というよりは、言語学習（幼児はどうして言語を学習できるのか）の内容が中心であった [Gold67]
  - たとえば、幼児は親から言葉の正例だけが与えられて、負例はあまり与えない（たとえばめちゃくちゃな言葉を言って「これは日本語ではない」というような教育はしていない）。
  - それなのに自然言語を獲得できる。それを理論的に証明できるか？（答えはNOであった．負例が与えられれば ある程度はYES）。
    - 当時の言語学はチョムスキー学派に権威があり、その言語階層に従った。
    - 言語学において自然言語は、一部の例外を除き「文脈自由文法」の範囲であるらしいことが1980年ごろから通説となった。
    - チョムスキー階層（形式言語）は、特に計算理論の関連性が重要視され、現在では計算機科学の分野でのみ多く利用されている。
- [Blum75]が、この研究が「プログラムによる学習可能性と計算量」と繋がることを示し、計算論的学習理論の研究が進む（現在も）。

計算論に基づいた帰納推論

- 学習としては完全に正しくなるという条件が厳しかった。
  - 当然、訓練データものどのように現れる（与えられる）か分からない。
  - データに誤差はないのか？
  - 極限で同定と言われても。。。
  - ＝＞PAC learning

計算論的学習（PAC学習）(1)

- PAC learning = Probably approximately correct learning
- 何をしようとしているか。
  - C, Java, pythonなどで書かれた関数$F(x)$がある。
    $F: \mathbb{R} \to \{-1, 1\}$
    $\mathbb{R}$（整数や有限集合でもよい）上の確率分布があり、その確率に基づいて$x_1, \cdots, x_n, \cdots$が生成され、それに従って入力と出力の組（ただし$F(x) = \pm 1$）
    $(x_1, F(x_1)), \cdots, (x_n, F(x_n)), \cdots$
    をいくつか観測して、$F(x)$ とほとんど同じとなるプログラムを高い確率で生成できるか？
    - 「ほとんど」とは高確率で正しいこと。頻繁に発生するxについて$F(x)$ は正しく、あまり発生しないxについては正しくないかも。

計算論的学習（PAC学習）(2)

- PAC学習では標本空間 (起こり得る事象) X（前では $\mathbb{R}$）上の確率分布 $\mathcal{D}$を仮定し、標本Sはその分布に従って選択される。
- $\mathcal{D}$に従って$x \in X$が選択される確率をP_xと置く。$\sum_{x \in X} P_x = 1$である。
- X上の判別関数の集合H (前の$\mathcal{R}$に相当) を仮説集合という。ここから正解を探す。
- PAC学習では学習アルゴリズム$\mathcal{L}$の成功を示すパラメータとして信頼度δと近似度εを導入する (ただし、$0 \leq \delta, \varepsilon \leq 1$)。
- 学習データの表現は前述と同じ。その真の判別関数を$h^* \in H$と表す。$\mathcal{L}$はHから候補hを出力する。なお、 $h^* \in H$と仮定する。
- h^*とhの誤差 (差分、距離) を

$$d_X(h^*, h) = \sum_{x \in X, h^*(x) \neq h(x)} P_x$$

と定義する。全体での誤差を測る。

計算論的学習（PAC学習）(3)

- 帰納的関数のクラスHをPAC学習可能とは、判別関数$h^* \in H$とX上の任意の確率分布$\mathcal{D}$に対して、多項式時間の学習アルゴリズム$\mathcal{L}$が存在し、任意の$0 \leq \delta, \varepsilon \leq 1$について

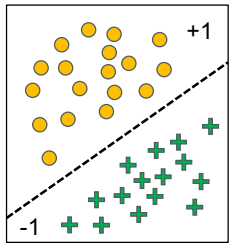
$$\Pr(d_X(h^*, h) \leq \varepsilon) \geq 1 - \delta$$

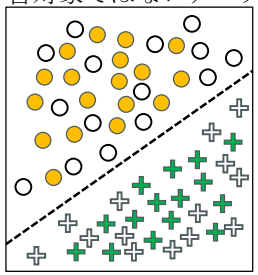
となる$h \in H$ を出力し停止すること。[Valiant84]

  - したがってあまり与えられないデータについては正解できなくても仕方ない(ε)。[逆に言えば正解して欲しい部分は、ちゃんと学習
    ➡ Boosting (Adaboost)]
  - データも確率的に与えられるので、望ましいデータ系列ばかりではないので、このようなhをいつも出力できるわけではない$(1 - \delta)$。[なら、数回試したり並列化して多数決を取ればいいかも (アンサンブル学習)]

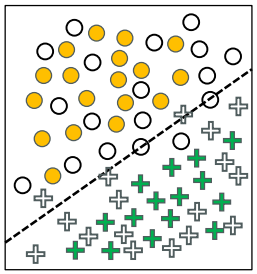
計算論的学習（PAC学習）(4)

- 学習対象ではないデータも分類 (極限による同定)
- 学習サンプル (S) と区分





白抜きの点を含めてX



- 多少は誤差があっても、ほとんどが合っていればよい (PAC)

計算論的学習（PAC学習）(5)

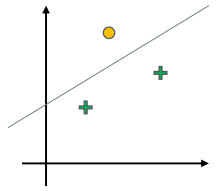
- ここでいくつかの疑問：
  1. 正解に必要な訓練データって最低いくつぐらい？また訓練データに正解してもX全体に対する正解の割合は分からないのでは？（今後これは省略）
  2. データが複雑で大量でも区別できるの？その目安は？
  3. 精度を上げる（一致する確率を高くする）学習アルゴリズムは難しい。それなら、誤差が大きめの弱い学習器をいつか合わせて、精度の高い強力な学習器はできるか？

22 VC次元 (Vapnik-Chervonenkis dimension)

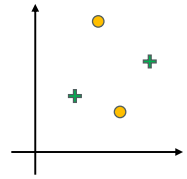
VC次元、次元の呪い

VC次元

- PAC学習の疑問2
  - 訓練データが与えられたとして、いつも区別（つまり判定する関数を (PAC) 学習）できるのか？
  - 前の式では$d_X(h^*, h) - d_S(h^*, h)$は、$|H|$に依存（標本での一致度と全体での一致度の差）。でもHが無限なら？ (Sは標本)
  - 直線（超平面）で区別することを考えよう。
    - ２次元：３点なら常にできる（一つの直線に乗るような場合は除く）
    - ４点以上は無理なことがある。
  - では３次元では？一般にn次元では？
    - （超）平面で分ける場合には、$n+1$の点まで分類できることが分かっている。



できた



できない

23

VC次元

- X上の判別関数からなる仮説集合をHと表し、Xの部分集合（サンプル）をSとおく。 Sを$h \in H$の値が+1か−1で判別することを考える。$|S|=N$とおく。
  - $C_+(h) = \{x \in X \mid h(x) = 1\}$とおく。
  - Sを2分割する方法は2^N通り。Sの全ての部分集合の集まりを2^Sと書くことがある。したがって$|2^S| = 2^N$。
  - もしHに含まれる判別関数でSの全ての分割方法が表せるときHはSを細分化できる (shatter) という。これは、$\Pi_H(S) = \{C_+(h) \cap S \mid h \in H\}$とおいたとき、$\Pi_H(S) = 2^S$となることである。
  - HのVC次元、VCdim(H)を以下のように定義する。
    $VCdim(H) = \max\{|S| \mid S \subset X, S$は$H$によって細分化できる$\}$
    最大値であることに注意。
  - つまり、全ての分割が表せる標本の最大数。

24

VC次元の例と訓練データ数の目安

- Hを2次元実数空間$\mathbb{R}^2$上の平面で区分する関数とする。このときVCdim(H)=3である。
- 一般にHをn次元実数空間$\mathbb{R}^n$上の超平面 (n変数の1次式)の集合とすると、 VCdim(H) = n + 1となる。
- 参考：
  VCdim(H) = dとする。$h^* \in H$のサンプルSに矛盾しない$h \in H$を出力するアルゴリズムが存在するなら、 h^*はPAC学習可能で、必要訓練データ数mは

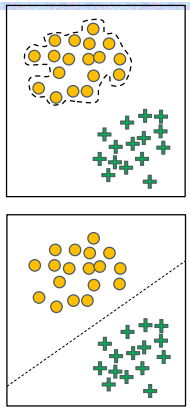
$$m \geq max\left(\frac{4}{\delta}\log\frac{2}{\delta}, \frac{8d}{\varepsilon}\log\frac{13}{\varepsilon}\right)$$

  であることが知られている。Logの低は2。

25

VC次元の大小

- VC次元は大きい方（よってデータの次元が高い方）が判別能力が上がるのでよい？ ＝＞ そうともいえない。
- 学習の汎化（一般化）の観点
  - 訓練データ（教師信号）だけではなく、そこにないデータもある程度適切に区分して欲しい。
  - 次元が上がるとパラメータの組み合わせは指数的に増えるので、複雑なことも高次元の複雑な関数で表せる。あまりにも厳密となり、訓練データ以外のデータはうまく判定できなくなる（右図：どちらがよさそう？）＝＞次元の呪い（curse of dimensionality. いろいろな意味がある）
  - データ含まれる誤差（雑音）も学習する。
  - 過学習 (overfitting) の問題



VC次元の大小

- 判別は可能でVC次元がなるべく小さい空間で判別する。
  - 過学習を減らし、確率的に正解に近づける。
- 先ずは超平面で判別することを考える

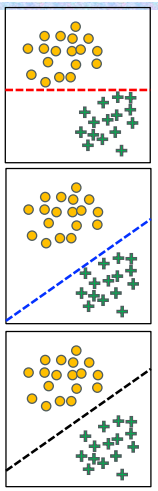
29

SVM(1) 線形分類
ハードマージン

SVMは、1960年代に考案されました。ここではその当時の線形分類に関して説明します。特に、マージン最大化とVC次元次元の最小化の概念について。

ハードマージンと区分する境界線

- 直感的な説明
  - 例外なく分離可能とする (ハードマージン)
  - 右図の赤青黒の区分線は、どれがよい？
  - 赤線はその近くまでサンプルデータがあり、境界がアンバランス。マージンが狭い
  - 青線は＋によりすぎている。マージンが偏っている
  - 黒は両者の真ん中当たりの広い空間で分類している（マージンが共に大きく、バランスがとれている）
- 黒点線の境界線ようなマージンを決定する超平面の決定は？



マージン（余白）最大化 (1)

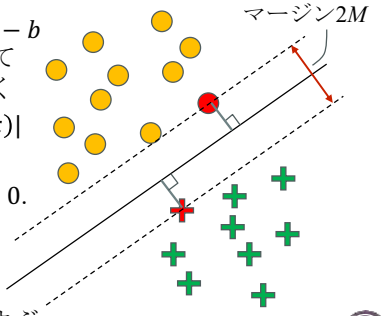
- 分類できる平面のうち、その平面までの距離の最小値が、両者とも最大化（バランス）するように平面を決める。
  - 両者の点からの距離の最小値を最大化すればよい。（一方から離れすぎると、他方のサンプルからの距離が近くなり、最大化とならない）。
  - サンプルを$S \subset \mathbb{R}^n$、平面を$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b$とおく。定数倍して$\| \boldsymbol{w} \| = 1$と仮定して良い。$f(\boldsymbol{x})$による判別関数を$h(\boldsymbol{x})$とおく

$$\max_{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}} \min_{\boldsymbol{x}\in S} \frac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|} = \max_{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}} \min_{\boldsymbol{x}\in S} |f(\boldsymbol{x})|$$

であり、さらに
$f(\boldsymbol{x})$は正確に分類できるので$f(\boldsymbol{x})h(\boldsymbol{x}) > 0$.
特に

$$f(\boldsymbol{x})h(\boldsymbol{x}) - M \geq 0$$

  - ここでMはマージン。マージンを決定する点線上の点をサポートベクトルとよぶ。



マージン（余白）最大化 (2)

- この値Mが1となるように線形変換（拡大・縮小）する。
  $\boldsymbol{w}$を$\hat{\boldsymbol{w}} = \boldsymbol{w}/M$, bを$\hat{b} = b/M$, $\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x})/M$と変換。$\hat{f}(\boldsymbol{x})$の最小値は1なので、

$$\max_{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}} \min_{\boldsymbol{x}\in S} \frac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|} = \max_{\hat{\boldsymbol{w}}\in\mathbb{R}^n, \hat{b}\in\mathbb{R}} \frac{1}{\|\hat{\boldsymbol{w}}\|} \min_{\boldsymbol{x}\in S} |\hat{f}(\boldsymbol{x})| = \max_{\hat{\boldsymbol{w}}\in\mathbb{R}^n, \hat{b}\in\mathbb{R}} \frac{1}{\|\hat{\boldsymbol{w}}\|}$$

  ただし (s.t.)

$$\hat{f}(\boldsymbol{x})h(\boldsymbol{x}) - 1 \geq 0 \quad \text{for } \forall \boldsymbol{x} \in S$$

- $\hat{f}(\boldsymbol{x})$と$f(\boldsymbol{x})$は同じ平面。$\hat{\boldsymbol{w}}, \hat{b}$を単純に$\boldsymbol{w}, b$と表すと、以下の$\boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}$を求めればよい（$\| \boldsymbol{w} \|$より$\| \boldsymbol{w} \|^2 = \boldsymbol{w}\boldsymbol{w}^T$の方が容易）

$$\min_{\boldsymbol{w}\in\mathbb{R}^n, b\in\mathbb{R}} \frac{1}{2} \| \boldsymbol{w} \|^2 \qquad (7.1)$$

$$\text{s.t.} \quad f(\boldsymbol{x})h(\boldsymbol{x}) = |f(\boldsymbol{x})| \geq 1 \quad \text{for } \forall \boldsymbol{x} \in S$$

  ただし、$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b$。上記は2次計画問題となる。
- 式(7.1)はminだが、マージン最大化の制約であることに注意。

34 SVM (2) 線形分類
ソフトマージン

プログラムを書くときはライブラリなどを使うだけになると思いますが、それには簡単ではありませんが自然な流れと、厳密な証明があります。

数学の準備(1)

- ラグランジュの未定乗数法のやや一般化（再掲）
- $f(x_1, \ldots, x_n)$, $g_k(x_1, \ldots, x_n)$は関数 ($1 \leqq k \leqq m$)。$\boldsymbol{x}=(x_1, \ldots, x_n)$ とおく。条件$g_k(x_1, \ldots, x_n)=0$のもと、$f(x_1, \ldots, x_n)$を最小化（最大化）する解 $\boldsymbol{c}=(c_1, \ldots, c_n)$ は、

$$L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = f(x_1, \ldots, x_n) - \sum_{k=1}^{m} \lambda_k g_k(x_1, \ldots, x_n)$$

  としたとき、

$$\frac{\partial L(\boldsymbol{x}, \boldsymbol{\lambda})}{\partial x_1} = \cdots = \frac{\partial L(\boldsymbol{x}, \boldsymbol{\lambda})}{\partial x_n} = \frac{\partial L(\boldsymbol{x}, \boldsymbol{\lambda})}{\partial \lambda_1} = \frac{\partial L(\boldsymbol{x}, \boldsymbol{\lambda})}{\partial \lambda_m} = 0 \qquad (2)$$

  の解（つまり停留点、極値）でもある。

➡ これをさらに一般化

数学の準備(2)

- $f(x_1, \ldots, x_n)$, $g_k(x_1, \ldots, x_n)$, $h_j(x_1, \ldots, x_n)$は関数 ($1 \leqq k \leqq m$, $1 \leqq j \leqq l$)。 $\boldsymbol{x}=(x_1, \ldots, x_n)$ とおく。条件 $g_k(x_1, \ldots, x_n)=0$, $h_k(x_1, \ldots, x_n) \geqq 0$のもと、 $f(x_1, \ldots, x_n)$を最小化する解 $\boldsymbol{c}=(c_1, \ldots, c_n)$ を求めるとする。このときラグランジュ関数とは、

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\xi}) = f(\boldsymbol{x}) - \sum_{k=1}^{m} \lambda_k g_k(\boldsymbol{x}) - \sum_{j=1}^{l} \xi_j h_j(\boldsymbol{x})$$

  と定義する。
- なお、 $h_k(x_1, \ldots, x_n)$にマイナスをつけて、不等号の向きを変えると、上記の式を以下のように表すこともある。

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\xi}) = f(\boldsymbol{x}) + \sum_{k=1}^{m} \lambda_k g_k(\boldsymbol{x}) + \sum_{j=1}^{l} \xi_j h_j(\boldsymbol{x})$$

数学の準備(3)

- Kuhn-Tucker theorem（クーン・タッカーの定理）

  $f(x_1, \ldots, x_n)$, $g_k(x_1, \ldots, x_n)$, $h_j(x_1, \ldots, x_n)$は関数 ($1 \leqq k \leqq m$, $1 \leqq j \leqq l$)。 $\boldsymbol{x}=(x_1, \ldots, x_n)$ とおく。条件 $g_k(x_1, \ldots, x_n)=0$, $h_k(x_1, \ldots, x_n) \geqq 0$のもと、 $f(x_1, \ldots, x_n)$を最小化する解 $\boldsymbol{c}=(c_1, \ldots, c_n)$ を求めるとする。このとき$\boldsymbol{x} = \boldsymbol{w}^*$が最適解となる必要十分条件は、ラグランジュ関数$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\xi})$に対し、

$$\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\lambda}, \boldsymbol{\xi})}{\partial \boldsymbol{x}} = 0,$$
$$\frac{\partial L(\boldsymbol{w}^*, \boldsymbol{\lambda}, \boldsymbol{\xi})}{\partial \boldsymbol{\lambda}} = 0,$$
$$\xi_j h_j(\boldsymbol{w}^*)=0,\ h_j(\boldsymbol{w}^*) \geq 0,\ \xi_j \geq 0$$

  となる$\boldsymbol{\lambda}, \boldsymbol{\xi}$の解が存在することである。

ソフトマージン

- $\forall x \in S$について$f(\boldsymbol{x})h(\boldsymbol{x}) \geq 1$とする仮定は強すぎ
  - データの雑音。実世界ではそれほど綺麗なデータはない。
- $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$とし$(\boldsymbol{x}_k, y_k) = (\boldsymbol{x}_k, h(\boldsymbol{x}_k)) \in \mathbb{R}^n \times \{1, -1\}$と表す。上記の仮定を以下のように緩和する

$$y_k f(\boldsymbol{x}_k) \geq 1 - \zeta_k, \qquad \zeta_k > 0 \quad \text{for } 1 \leq \forall k \leq m$$

  - この場合、「マージンの間」や「マージンを超えて異なるクラスの点」も存在することになる。
  - そうすると最大マージンって何？という疑問。
  - 上記の式は、$f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b = 1, f(\boldsymbol{x}) = -1$の間をマージンとしている。
  - 分類にエラー（誤分類）があるのは$\zeta_k > 1$のときなので、$\sum \zeta_k \leq K$ (K は正の整数) ならエラーとなる点は高々 K個となる。これも小さい方がよい。

ソフトマージンによる分類 (1)

- (7.1)を含め、これらを合わせて以下の最適化問題（2次計画問題）として表す。

$$\min_{\boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}, \boldsymbol{\zeta} \in \mathbb{R}^m} \left(\frac{1}{2} \| \boldsymbol{w} \|^2 + C \sum_{k=1}^{m} \zeta_k \right)$$

  s.t.

$$y_k\left(\boldsymbol{w}\boldsymbol{x}_k^T + b\right) \geq 1 - \zeta_k,$$
$$\zeta_k \geq 0 \qquad \text{for } 1 \leq \forall k \leq m$$

  - ここでCは正則化係数 (regularization parameter)という。マージンはなるべく大きい方がよいが、エラー（誤分類）は少ない方がよい。マージンの大きさと誤分類のバランスをとる。
    - Cが大きいとエラーが大きく作用し、誤分類を許しにくくなる。（特にCが極めて大きいときは事実上のハードマージンの分類となる）。
    - 逆にCが小さいと、エラーを許容し、マージンを大きく取る。
    - Cの値は実験的に求める（→　交差検定 (cross validation) ）

ソフトマージンによる分類 (2)

- 前の式からラグランジュ関数Lを定義。以下の最適化問題（2次計画問題）として表す (Kuhn-Tucker theorem)

$$L(\boldsymbol{w},b,\boldsymbol{\zeta},\boldsymbol{\xi},\boldsymbol{\xi}') = \frac{1}{2}\boldsymbol{w}\boldsymbol{w}^T + C\sum_{k=1}^{m}\zeta_k - \sum_{k=1}^{m}\xi_k\left(\zeta_k - 1 + y_k\left(\boldsymbol{w}\boldsymbol{x}_k^T + b\right)\right) - \sum_{k=1}^{m}\xi_k'\zeta_k$$

かつ　$\boldsymbol{\xi},\boldsymbol{\xi}' \geq \boldsymbol{0}$

- このうち$\boldsymbol{\xi},\boldsymbol{\xi}'$で微分して0となるものは後回し。まずは
- $\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{k=1}^{m}\xi_k\, y_k\boldsymbol{x}_k = \boldsymbol{0} \quad \Rightarrow \boldsymbol{w} = \sum_{k=1}^{m}\xi_k\, y_k\boldsymbol{x}_k z$
  - たとえば、$\frac{\partial L}{\partial w_i} = w_i - \sum_{k=1}^{m}\xi_k\, y_k x_{ki}$
- $\frac{\partial L}{\partial b} = \sum_{k=1}^{m}\xi_k\, y_k = 0$
- $\frac{\partial L}{\partial \zeta} = C - \boldsymbol{\xi} - \boldsymbol{\xi}' = \boldsymbol{0}$。これらを代入。➡この式から上の斜線

ソフトマージンによる分類

- $L(\boldsymbol{w},b,\boldsymbol{\zeta},\boldsymbol{\xi},\boldsymbol{\xi}')$を最大化する$\boldsymbol{\xi}$を求める。

$$\max_{\boldsymbol{\xi}} L(\boldsymbol{w},b,\boldsymbol{\zeta},\boldsymbol{\xi},\boldsymbol{\xi}')$$

$$= \max_{\boldsymbol{\xi}}\left(\frac{1}{2}\sum_{k=1}^{m}\xi_k\, y_k\boldsymbol{x}_k\sum_{j=1}^{m}\xi_j\, y_j\boldsymbol{x}_j^T + \sum_{k=1}^{m}\xi_k - \sum_{k,j=1}^{m}\xi_k y_k\xi_j y_j\boldsymbol{x}_j\boldsymbol{x}_k^T\right)$$

$$= \max_{\boldsymbol{\xi}}\left(\sum_{k=1}^{m}\xi_k - \frac{1}{2}\sum_{k,j=1}^{m}\xi_k y_k\xi_j y_j\boldsymbol{x}_j\boldsymbol{x}_k^T\right)$$

s.t. $0 \leq \xi_k \leq C, \sum_{k=1}^{m}\xi_k\, y_k = 0$

- 注：$\sum_{k=1}^{m}\xi_k\, y_k = 0$から$b$の項は消える。$C - \boldsymbol{\xi} = \boldsymbol{\xi}' \geq \boldsymbol{0}$ から $C \geq \boldsymbol{\xi}(\geq 0)$.

- この問題を[サポートベクターマシンの双対問題]といい、これを解く。

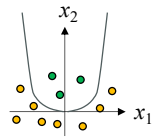
ソフトマージンによる分類

- $\boldsymbol{\xi}$の値によりサンプルを分類できる（正式にはもう少し議論は必要だが詳細は省く。多分、最適化などの大学院の講義で）。
  - $0 \leq \xi_k < C$ のときは、$y_k\left(\boldsymbol{w}\boldsymbol{x}_k^T + b\right) \geq 1$であり、そのサンプルを正しく分類できている。特に、$\boldsymbol{\xi} = \boldsymbol{0}$の点は境界線上にあり（スライド「マージン（余白）最大化 (1)」の点線。$y_k\left(\boldsymbol{w}\boldsymbol{x}_k^T + b\right) = 1$）、この境界線を定める点となる。
  - $\xi_k = C$のときは$y_k\left(\boldsymbol{w}\boldsymbol{x}_k^T + b\right) < 1$となりマージンを超え（つまりマージン内だったり、相手側のマージンの向こう側にある。この点については、正しく分類できない。
  - RにSVMのパッケージがあるので、興味があれば、各自調べてみてください。

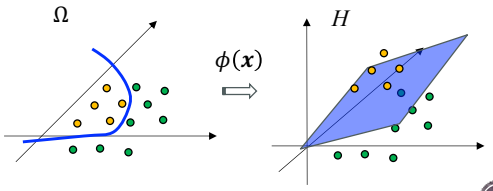
43 SVM (3) 非線形への拡張

高次元への写像

- 線形の超平面での分類は限度がある。
  - 高次元に写像（下図）。高次元であればVC次元も大きくなり埋め込むことができる。
    - 右図： $(x_1, x_2) \Rightarrow (x_1, x_1^2, x_2) = (y_1, y_2, y_3)$
    - $x_1 - x_1^2 + x_2 = 0$ （符号が逆かも．．）
    - 次元が高くなれば、計算コストも高くなる（これも次元の呪い）
    - 次元の呪い（過学習）も考慮すべき。何らかの制限が必要だが、最大マージンという制限が次元の呪いへの対応（の一部）となっている。



  - 非線形関数で分類。これはニューラルネットのような手法に相当するが、計算コストは高い。また非線形の関数はたくさんあるので、どれが良いかの判断も難しい



高次元への写像

- アイデア：非線形の空間を高次元の空間へ$\mathbb{R}^n$の要素を埋め込み、その空間で線形分離を試みる。
  - $f(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}^T - b$に$\boldsymbol{w} = \sum_{k=1}^{m} \xi_k\, y_k \boldsymbol{x}_k$（スライド39）を代入すると、

$$f(\boldsymbol{x}) = \sum_{k=1}^{m} \xi_k\, y_k \boldsymbol{x}_k \boldsymbol{x}^T - b$$

    となり、超平面もまたその判別関数も内積の形式になる。
- 事象の空間$X \subset \mathbb{R}^n$ とサンプル$S \subset \mathbb{R}^n$とする。高次元への写像

$$\boldsymbol{\phi}(\boldsymbol{x}): X \to \mathbb{R}^{n'}$$

  - 高次元に写像しても、内積で計算ができるような変換なら、同様に計算できる。これをカーネルトリックと呼ぶ。
  - つまり、$\boldsymbol{\phi}(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)^T = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$と高次元での内積だけから計算できるとき簡単になる可能性がある。

正定値カーネル

- 定義
  - 集合Ω、　関数$K: \Omega \times \Omega \to \mathbb{R}$が正定値カーネルとは以下の2条件を満たすこと。

1. $K(x, y) = K(y, x)$
2. Ω上のn個の任意の要素$(x_1, \ldots, x_n)$に対し$n \times n$行列

$$\begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{pmatrix}$$

   は正定値行列である。

- 正定値カーネルを利用して高次元に埋め込む。
  - $\boldsymbol{\phi}(\boldsymbol{x}) = K(\cdot, \boldsymbol{x})$ と定義すれば良い。このような写像を使って高次元に埋め込む。

正定値カーネル

- 前出の$L(\boldsymbol{w}, b, \boldsymbol{\zeta}, \boldsymbol{\xi}, \boldsymbol{\xi}')$をは以下のようになる。

$$L(\boldsymbol{w}, b, \boldsymbol{\zeta}, \boldsymbol{\xi}, \boldsymbol{\xi}') = \sum_{k=1}^{m} \xi_k - \frac{1}{2} \sum_{k,j=1}^{m} \xi_k y_k \xi_j y_j \boldsymbol{\phi}(\boldsymbol{x}_j) \boldsymbol{\phi}(\boldsymbol{x}_k)^T$$

$$= \sum_{k=1}^{m} \xi_k - \frac{1}{2} \sum_{k,j=1}^{m} \xi_k y_k \xi_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_k)$$

- と同様な形式となる。
  - ただし、上記を得るにはヒルベルト空間（内積が定義されているベクトル空間）の性質を使う。難しくはないが、長くなるので、ここでは省略する。

SVMの利点と欠点

利点

- 分類の能力は高い。高速。
- 線形の場合は学習結果の解釈が比較的容易。
- 最大マージンという性質で、過学習に対応している。
- 自動的に線形・非線形の特徴抽出ができる
- 複雑な非線関数も近似的に分類可能

欠点

- 線形分離だけでは能力に限界はある。
- 学習結果の解釈は容易ではない。特に非線形の場合
- 非線形ではデータ数が多くなると（たとえば10000以上）急速に遅くなる。

練習問題

- 課題８－１：
  - ソフトマージンとハードマージンについて、違いを述べよ。
- 課題８－２：
  - 次元をあげるとVC次元があがり、判別能力があがるが、学習面からみると、判別能力の向上は必ずしも望ましくはない。その理由について議論せよ。

アンサンブル学習とAdaboost

Ensemble learning and adaptive boosting

アンサンブル学習

- 考え方：精度の低い弱い学習器（弱学習器）を組み合わせて、精度の良い学習器を作りたい（PAC学習の疑問3）。これをアンサンブル学習という
- ここで想定する弱学習器：
  - 多くの場合は、線形分離やあまり深くない決定木などを使う。また、訓練データの一部のみからの学習結果もある。
  - 決定木の場合の極端な例としては深さ1の決定木（木ほど高くないので決定株とも言われる）に制限する方法もある。
  - ただし研究の発端は、弱PAC学習アルゴリズムという概念を定義して、これを通常のPAC学習アルゴリズムに理論的に押し上げることを狙って研究されたのが（あとで述べる）ブースティングの始まり。

参考：弱PAC学習

- 帰納的関数のクラスHを弱PAC学習可能とは、ある定数$\varepsilon_0 < \frac{1}{2}$と$\delta_0$が存在して、任意の判別関数 $h^* \in H$ とX上の任意の確率分布 $\mathcal{D}$ に対して、多項式時間の学習アルゴリズム$\mathcal{L}$が存在し、ついて
$$\Pr(d_X(h^*, h) \leq \varepsilon_0) \geq 1 - \delta_0$$
となる$h \in H$ を出力し停止すること。[Kearns and Valiant 89]

- （PAC学習との比較）帰納的関数のクラスHをPAC学習可能とは、判別関数$h^* \in H$とX上の任意の確率分布$\mathcal{D}$に対して、多項式時間の学習アルゴリズム$\mathcal{L}$が存在し、任意の$0 \leq \delta, \varepsilon \leq 1$について
$$\Pr(d_X(h^*, h) \leq \varepsilon) \geq 1 - \delta$$
となる$h \in H$ を出力し停止すること。[Valiant84]

主なアンサンブル学習の例

- バギング (Bagging, L.Breiman 1996)
  - m組のサンプルを抽出し、それぞれのサンプルからm個の弱学習器を作る。並列化が可能。
  - 組み合わせは、例えば多数決 (voting)とか平均値をとるなど。
  - ランダムフォレスト（前出）は、広い意味で一例
- ブースティング (Boosting, Kearns and Valiant 1989)
  - Adaboost (Freund and Schapire. 1997): 弱学習器で誤差のある訓練データの重みを大きくし、次に(重点的に) 学習する。[つまり弱点を強化]。これを繰り返して学習器の修正を積み重ねる。逐次的。
  - Gradient Boosted Decision Tree (GBDT): 弱学習器として決定木を利用。誤差に基づいてサンプルの重みを変える考え方は同じ。逐次的。
- ブースティングの代表的アルゴリズムであり、よく使われるAdaBoost (adaptive boosting) について概説する。

もう一度確認

- 前のスライドをもう一度確認（＋ちょっと修正）
  - 訓練データ（教師信号）は、X上のある確率分布$\mathcal{D}$に従ってその要素が選択される。
  - $\mathcal{D}$に従って、$x \in X$が選択される確率を$\mathcal{D}(x)$ $(= P_x)$と置く。確率分布なので、$\sum_{x \in X} \mathcal{D}(x) = 1$である。
  - データ正負を決める真の判別関数を$h^* \in H$と表す。仮説集合Hからアルゴリズム$\mathcal{L}$は候補hを出力する。
  - h^*とhの誤差 (差分、距離) を
$$d_X(h^*, h) = \sum_{x \in X, h^*(x) \neq h(x)} \mathcal{D}(x)$$
と定義する。距離をd_Xと書くことにした。
  - 学習アルゴリズムに与えられる入力は$(\boldsymbol{x}_1, y_1) \ldots (\boldsymbol{x}_m, y_m)$ここで$\boldsymbol{x}_i \in X, y_i = h^*(\boldsymbol{x}_i) \in \{-1, 1\}$。なるべく正確に分類したい。

Boosting

- データX上の初期分布$\mathcal{D}_1$を等確率とする ($\mathcal{D}_1(\boldsymbol{x}_j) = 1/m$)とする。これに従ってサンプルを選択。 $t = 1$。
  1. $\mathcal{D}_t$に基づくサンプルからアルゴリズム$\mathcal{A}$が弱学習器がh_tを出力する。
  2. $\varepsilon_t = d_X(h^*, h_t)$を求める。
  3. ε_tに基づいて「h_tの重要度」と「新しいX上の分布$\mathcal{D}_{t+1}$」を定義して1へ戻る。
  4. 上記を一定回数 (T回) 繰り返し、得られた出力$h_1 h_2, \ldots, h_T$を組み合わせて最終出力を$H(\boldsymbol{x})$とする。
- アンダーラインの部分がboostingの方法ごとに違う。
- アルゴリズム$\mathcal{A}$には学習アルゴリズムがくる。

Adaboost

- 訓練データX上の初期分布$\mathcal{D}_1$を等確率とする。つまり、$\mathcal{D}_1(\boldsymbol{x}_j) = 1/m$とする。 $t = 1$。
  1. $\mathcal{D}_t$に基づくサンプルからアルゴリズム$\mathcal{A}$が弱学習器がh_tを出力
  2. $\varepsilon_t = d_X(h^*, h_t)$ // 誤差（エラー）を計算
  3. $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$をエラーに基づく$h_t$の重要度とする。
  4. $\mathcal{D}_{t+1}(\boldsymbol{x}_j) = \frac{\mathcal{D}_t(\boldsymbol{x}_j)\exp(-\alpha_t y_j h_t(\boldsymbol{x}_j))}{Z_t}$と定義。ここで$Z_t$は$\sum_{x\in X}\mathcal{D}_{t+1}(\boldsymbol{x}) = 1$となるように正規化するもの。
  5. $H(\boldsymbol{x}) = \mathrm{sgn}\left(\sum_{t=1}^{T}\alpha_t\, h_t(\boldsymbol{x})\right)$ // これは重み付きの多数決。

Adaboost

- Adaboostの計算は何をしているのか？(1)
  - まず$\mathcal{D}_{t+1}$から。$\sum_{\boldsymbol{x}_j\in X, y_j h_t(\boldsymbol{x}_j)=-1}\mathcal{D}_t(\boldsymbol{x}_j)\exp\left(-\alpha_t y_j h_t(\boldsymbol{x}_j)\right)$
    $= \sum\mathcal{D}_t(\boldsymbol{x}_j)\exp(\alpha_t) = \sum\mathcal{D}_t(\boldsymbol{x}_j)\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} = \sqrt{\varepsilon_t(1-\varepsilon_t)}$
  - $\sum_{\boldsymbol{x}_j\in X, y_j h_t(\boldsymbol{x}_j)=1}\mathcal{D}_t(\boldsymbol{x}_j)\exp\left(-\alpha_t y_j h_t(\boldsymbol{x}_j)\right)$
    $= \sum\mathcal{D}_t(\boldsymbol{x}_j)\exp(-\alpha_t) = \sum\mathcal{D}_t(\boldsymbol{x}_j)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} = \sqrt{\varepsilon_t(1-\varepsilon_t)}$
    - 正しかった部分とエラーの部分が等確率（1/2)となるように分布を決めている。
    - これはh_tがランダムになるようにわざと再定義して、次の学習している。（一方、例えば$\varepsilon_t = 0.9$とすると、これは精度良くエラーを出しているので−1倍すれば、精度良い学習器になる。上記のようにするのが、前の学習器にとっては不利なものとなる）

Adaboost

- Adaboostの計算は何をしているのか？(2)
  - $\alpha_t h_t(\boldsymbol{x}) = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}\times h_t(\boldsymbol{x})$
    - 直感的にはε_t（逆に$1-\varepsilon_t$）が小さいものほど重みを大きくして強く反映している。
    - $\varepsilon_t = 0.5$で$\alpha_t = 0$となる。ランダムに相当し、情報がないということ。
    - 仮に$\varepsilon_t > 0.5$とすると、
      $\alpha_t h_t(\boldsymbol{x}) = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}\times h_t(\boldsymbol{x}) = \frac{1}{2}\ln\frac{1-(1-\varepsilon_t)}{1-\varepsilon_t}\times(-h_t(\boldsymbol{x}))$
      となり、$-h_t(\boldsymbol{x})$を使う。これならエラーは$1-\varepsilon_t < 0.5$
    - それまでの弱点を補強する形で、アルゴリズム$\mathcal{A}$が新しい学習器を見つける。

Adaboostの利点と欠点

利点

- 実装が容易。処理も高速
- パラメータが少ない (Tぐらい)
- 弱学習器との独立性が高く、うまく活用できる。
- 汎化能力は高い
- 過学習 (overfitting)が他の方法に比べて起こりにくいと言われる

欠点

- 解の最適性に難あり
- ノイズデータの影響を受けやすい。過学習になることも。
- 弱学習器に対してデータが複雑な場合、判別がうまくいかなかったり、逆に過学習になることもある。

Ensemble Learning文献

- Breiman, L. “Bagging Predictors,” Machine Learning (1996) 24: 123. https://doi.org/10.1023/A:1018054314350
- M.J. Kearns and L. Valiant. “Cryptographic limitations on learning Boolean formulae and finite automata,” Proc. of the 21th ACM STOC, pages 433–444, 1989.
- Yoav Freund and Robert E Schapire. 1997. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,” *J. Comput. Syst. Sci.* 55.