机器学习基石 I: Perceptron

Key word: Definition of Machine Learning, Perceptron Learning Algorithm, PLA Python Script, Convergence of PLA, Pocket Algorithm

1 导论

1.1 机器学习的定义

在定义机器学习之前, 首先要定义什么是学习. 学习是通过观察（Observation）从而获得技能（Skill）的过程. 而机器学习便是计算机通过观察数据从而获得技能的过程.

而技能（Skill）是提高某种可评价能力的表现（如提高预测的精准度）. 因此机器学习就是计算机通过观察或计算数据, 从而提高某种能力的过程. 这是对机器学习的粗糙定义.

考虑这样一个例子, 如何定义一颗树, 使得计算机能够通过该定义自动辨别一棵树.

图 1: Tree

你会发现这不是一个容易的事情, 这便是机器学习的 Motivation. 一般而言机器学习的应用场景有三个特点:

1. 存在一个潜在的可以被学习的规则模式;
2. 该规则模式难以简单的定义;
3. 存在由该规则模式产生的数据;

例如预测股票市场的表现, 如果我们假设金融市场是无效的, 则股票收益率可以由历史值预测, 但是具体的预测公式难以被定义, 因此我们可以通过机器学习以及股票历史收益预测股票的未来收益.

1.2 机器学习的应用

机器学习在衣食住行育乐中都有广泛应用, 在这里我们举一个机器学习在娱乐（推荐系统）中的应用：考虑如何为一个用户推荐电影. 一个电影有多个特征, 如：是否为喜剧、是否为动作片、是否有某位明星出演等等. 通过其他用户为该电影的评价可以获得一部电影的特征向量, 再与用户自身对这些特征喜爱程度的向量做内积, 结果越高则说明该用户喜欢这部电影的可能性越高.

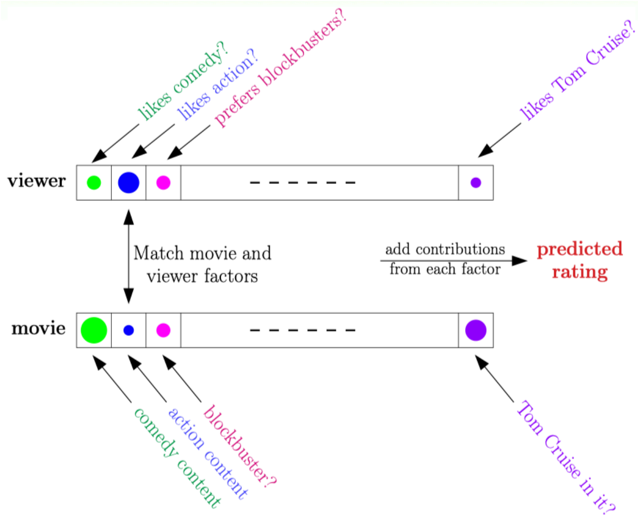


图 2: recommender system

注 1.1. *参考网易云音乐的推荐算法.*

1.3 机器学习的组成

考虑如下一个场景, 当客户申请信用时, 如何评价其违约的可能, 这是该客户的特征:

x_1	age	23
x_2	gender	female
x_3	annual salary	NTD 1000000
x_4	year in residence	1
x_5	year in job	0.5
x_6	current debt	200000

对于该例我们有以下基本概念：输入（input）$\mathbf{x} \in \mathcal{X}$, 即一份客户的申请；输出（output）$y \in \mathcal{Y}$, 即批准该申请后是否违约；可被学习的未知规则模式, 即目标函数（target function）$f : \mathcal{X} \mapsto \mathcal{Y}$ 即真实的信贷评价公式；数据, 也称训练样本

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\},$$

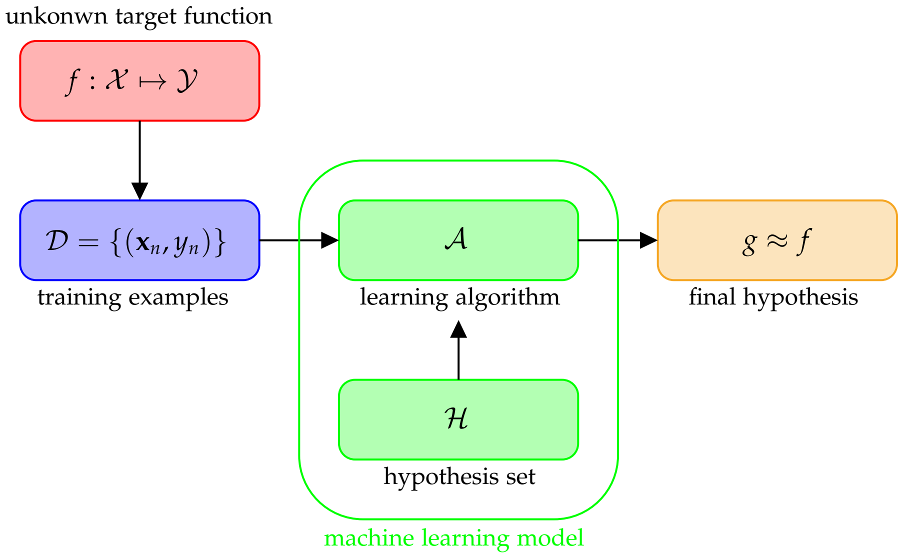
即银行的历史信贷数据；假设（hypothesis）$g : \mathcal{X} \mapsto \mathcal{Y}$, 即从训练数据中估计得到的评价公式, 理想的 hypothesis 应该逼近于 f. 机器学习的过程可以被简化为

$$\mathcal{D} = \{(\mathbf{x}_n, y_n)\} \text{ from } f \to \mathcal{A} \to g.$$

其中 f 是未知的真实函数, 数据集 $\mathcal{D}$ 在 f 规则下产生, 通过通过机器学习算法 $\mathcal{A}$ 获得逼近于 f 的 hypothesis g.

更具体而言, g 有多种形式, 如 $h_1: x_3 > 800000$, $h_2: x_6 > 100000$, $h_3: x_4 + x_5 > 5$ 等等, 这些好的假设或者坏的假设一起构成假设集（hypothesis set）$\mathcal{H} = \{h_n\}$. 而机器学习算法就是从假设集 $\mathcal{H}$ 中找到最符合 $\mathcal{D}$ 的一个假设 h_i 构成 g, 因此一个机器学习模型包含两个元素：机器学习算法 $\mathcal{A}$ 和假设集 $\mathcal{H}$. 可以看到机器学习的过程有两个输入：一是输入的数据 $\mathcal{D}$；二是允许的假设集 $\mathcal{H}$.

下面对机器学习给出更为具体的定义：机器学习即从数据 $\mathcal{D}$ 出发, 从假设集 $\mathcal{H}$ 中找出最符合 f 的 g:



Example 1.1. How to use the four sets below to form a learning problem for song recommendation?

$\mathcal{S}_1 = [0, 100]$; $\mathcal{S}_2$ = all possible (userid, songid) pairs; $\mathcal{S}_3$ = all formula that 'multiplies'user factors and song factors, indexed by all possible combinations of such factors; $\mathcal{S}_4$ = 1,000,000 pairs of ((userid, songid), rating).

then, $\mathcal{S}_1 = \mathcal{Y}, \mathcal{S}_2 = \mathcal{X}, \mathcal{S}_3 = \mathcal{H}, \mathcal{S}_4 = \mathcal{D}$:

$$\mathcal{S}_4 \to \mathcal{A} \text{ on } \mathcal{S}_3 \to (g: \mathcal{S}_2 \mapsto \mathcal{S}_1).$$

2 感知器 Perceptron

2.1 感知器假设集 Perceptron Hypothesis Set $\mathcal{H}$

这一节我们考察假设集 $\mathcal{H} = \{h\}$ 的结构. 回顾上一节银行信贷例子中客户的特征：x_1 至 x_6. 对于每个客户, 我们称 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^T$ 为该客户的特征向量, 对特征向量计算一个加权“分数”, 如果得分大于某一限值 T 则批准客户申请, 小于则拒绝（先不考虑相等的情况）.

$$\sum_{i=1}^{d} \omega_i x_i > T \quad \text{approve;}$$
$$\sum_{i=1}^{d} \omega_i x_i < T \quad \text{deny.}$$

对于这两种情况, 我们用 $+1$ 表示 approve, 用 -1 表示 deny, 我们便定义了一个公式 $h \in \mathcal{H}$:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{d} \omega_i x_i - T\right).$$

我们称这样的 h 为**感知器**, 不同的 ω_i 和 T 便可以定义出不同的 h. 对于感知器, 我们可以进行一些化简:

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sign}\left(\sum_{i=1}^{d}\omega_i x_i - T\right) \\ &= \operatorname{sign}\left(\sum_{i=1}^{d}\omega_i x_i + \underbrace{-T}_{\omega_0}\cdot\underbrace{+1}_{x_0}\right) \\ &= \operatorname{sign}\left(\sum_{i=0}^{d}\omega_i x_i\right) \\ &= \operatorname{sign}\left(\mathbf{w}^T\mathbf{x}\right). \end{aligned}$$

我们称 $\mathbf{w}$ 为权重（weight）, 每一个 $\mathbf{w}$ 决定一个假设 h.

考虑一个 $\mathbb{R}^2$ 感知器: $h(\mathbf{x}) = \operatorname{sign}(\omega_0 + \omega_1 x_1 + \omega_2 x_2)$. 每一个客户的特征构成 $\mathbb{R}^2$ 平面上的一个点; 标签（或称为目标）y 则为 $+1$（approve）或 -1（deny）; 假设 hypothesis h 为 $\mathbb{R}^2$ 平面上的一条直线, 其中正例在直线的一边（$\omega_0 + \omega_1 x_1 + \omega_2 x_2 > 0$）, 负例在直线的另一边, 因此感知器也被称为线性二分分类器. 注意, 不同的 $\mathbf{w}$ 决定不同的直线, 从而对客户产生不同分类结果.

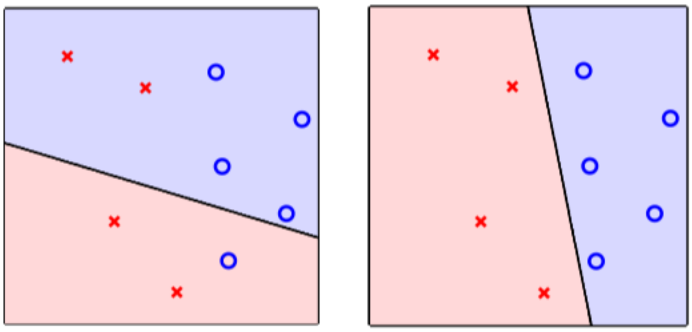


图 3: 两个感知器

2.2 感知器学习算法 Perceptron Learning Algorithm

我们现在已经知道了一个可能的假设集 $\mathcal{H}$, 即 $\mathcal{H} = \{h : h = \operatorname{sign}(\mathbf{w}^T\mathbf{x}), \mathbf{w} \in \mathbb{R}^d\}$. 考虑二维情况, 我们需要设计一个算法 $\mathcal{A}$ 从 $\mathcal{H}$ 中选出好的 g, 因为 g 完全由 $\mathbf{w}$ 决定, 因此我们只需找出最好 (最符合输入数据集 $\mathcal{D}$) 的 $\mathbf{w}$ 即可.

一种设计想法是从 $\mathbb{R}^2$ 中任意选择一个 $\mathbf{w_0}$, 然后再其判断错例的基础上逐步修正 $\mathbf{w}_t$. 考虑以下两种情况, 对于 $t = 0, 1, 2, \cdots$, 如果 $y_n = -1$ 而 $\operatorname{sign}(\mathbf{w}_t^T\mathbf{x}_n) = +1$, 即 $\mathbf{w}_t^T\mathbf{x}_n > 0$, 则说明向量 $\mathbf{w}$ 与 $\mathbf{x}$ 的夹角小于 90 度, 因此通过

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}_n = \mathbf{w}_t + y_n\mathbf{x}_n$$

将其修正. 如果 $y_n = +1$ 而 $\operatorname{sign}(\mathbf{w}_t^T\mathbf{x}_n) = -1$, 即 $\mathbf{w}_t^T\mathbf{x}_n < 0$, 则说明向量 $\mathbf{w}$ 与 $\mathbf{x}$ 的夹角大于 90 度, 因此通过

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}_n = \mathbf{w}_t + y_n\mathbf{x}_n$$

将其修正. 综上, 我们得到一个对 $\mathbf{w}_t$ 的修正算法:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_n\mathbf{x_n}.$$

若 $y_n \neq \operatorname{sign}(\mathbf{w}_t^T\mathbf{x}_n)$, 则

$$
\begin{aligned}
y_n \mathbf{w}_{t+1}^T \mathbf{x}_n &= y_n(\mathbf{w}_t + y_n \mathbf{x}_n)^T \mathbf{x}_n \\
&= y_n \mathbf{w}_t^T \mathbf{x}_n + y_n^2 \mathbf{x}_n^T \mathbf{x}_n \\
&\geq y_n \mathbf{w}_t^T \mathbf{x}_n.
\end{aligned}
$$

即若 $y_n = +1$, 则 $\mathbf{w}_{t+1}^T \mathbf{x}_n > \mathbf{w}_t^T \mathbf{x}_n$, 说明 $\mathbf{w}$ 与 $\mathbf{x}_n$ 的夹角从过大在减小; 若 $y_n = -1$, 则 $\mathbf{w}_{t+1}^T \mathbf{x}_n < \mathbf{w}_t^T \mathbf{x}_n$ 说明 $\mathbf{w}$ 与 $\mathbf{x}_n$ 的夹角从过小在增大, 即确实在根据错误进行学习.

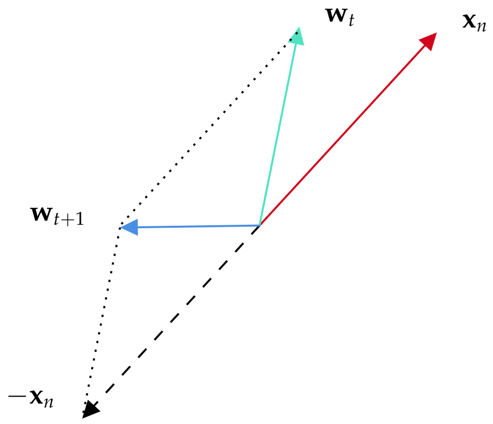


图 4: $y_n = -1$ 而 $h_t(\mathbf{x}_n) = +1$ 的情况

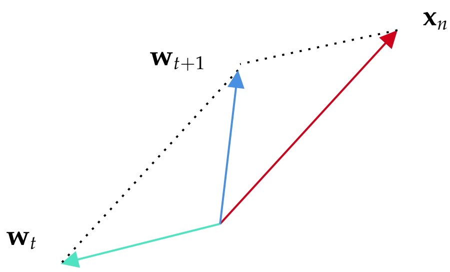


图 5: $y_n = +1$ 而 $h_t(\mathbf{x}_n) = -1$ 的情况

注意 $\mathbf{w}\mathbf{x} = \omega_0 + \omega_1 x_1 + \omega_2 x_2 = 0$, 即 $x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\omega_0}{\omega_2}$. 即感知器的分隔线是权重向量 $\mathbf{w}$ 的法线.

下面用 python 实现感知器学习算法:

```
#  生成数据
randlist1 = np.array([np.random.uniform(-5,5) for i in range(500)])
randlist2 = np.array([np.random.uniform(0,3) for i in range(500)])
data_p1_0 = pd.DataFrame(
    {
        "x0": 1,
        "x1": randlist1,
        "x2": randlist1 + randlist2,
        "y" : -1
    }
)

data_m1_0 = pd.DataFrame(
    {
        "x0": 1,
        "x1": randlist1,
        "x2": randlist1 - randlist2,
        "y" : +1
    }
)
```

```python
df = pd.concat([data_m1_0, data_p1_0], axis=0)

# 建模 画图
plt.figure(figsize=(8,6))
plt.scatter(data_m1_0["x1"], data_m1_0["x2"], c="r", alpha=0.5, linewidths=0)
plt.scatter(data_p1_0["x1"], data_p1_0["x2"], c="b", alpha=0.5, linewidths=0)
w = pd.Series([0, 0, 0], index=["x0", "x1", "x2"]); times = 1;  wl = []; x=np.linspace(-5,5,
                                                              100)
wl.append(w)
mis_p = df[df["y"]==+1][np.dot(df[df["y"]==+1].iloc[:,0:df.shape[1]-1], w) <= 0]
mis_n = df[df["y"]==-1][np.dot(df[df["y"]==-1].iloc[:,0:df.shape[1]-1], w) > 0]
mis = pd.concat([mis_p, mis_n], axis=0)
length = mis.shape[0]
while length > 0:
    mis_point = mis.iloc[np.random.randint(length),:]
    w = w + mis_point["y"] * mis_point.iloc[0: mis_point.shape[0]-1]
    mis_p = df[df["y"]==+1][np.dot(df[df["y"]==+1].iloc[:,0:df.shape[1]-1], w) <= 0]
    mis_n = df[df["y"]==-1][np.dot(df[df["y"]==-1].iloc[:,0:df.shape[1]-1], w) > 0]
    mis = pd.concat([mis_p, mis_n], axis=0)
    length = mis.shape[0]
    times += 1;  wl.append(w)
    if times in [1, 2, 3, 4, 5, 10, 15, 50, 100]:
        plt.plot(x, -(w[1]/w[2])*x - (w[0]/w[2]), "--", label=f"{times} hypothesis")
    if times > 50000 :
        print("over 50000 times loops")
        break
plt.plot(x, -(w[1]/w[2])*x - (w[0]/w[2]), "c", label=f"end hypothesis (times={times})")
plt.legend()
plt.savefig("/Users/wanghaoming/Documents/LaTeX_doc/Machine_Learning/pla.png", bbox_inches='
                                                              tight', dpi=500)
```

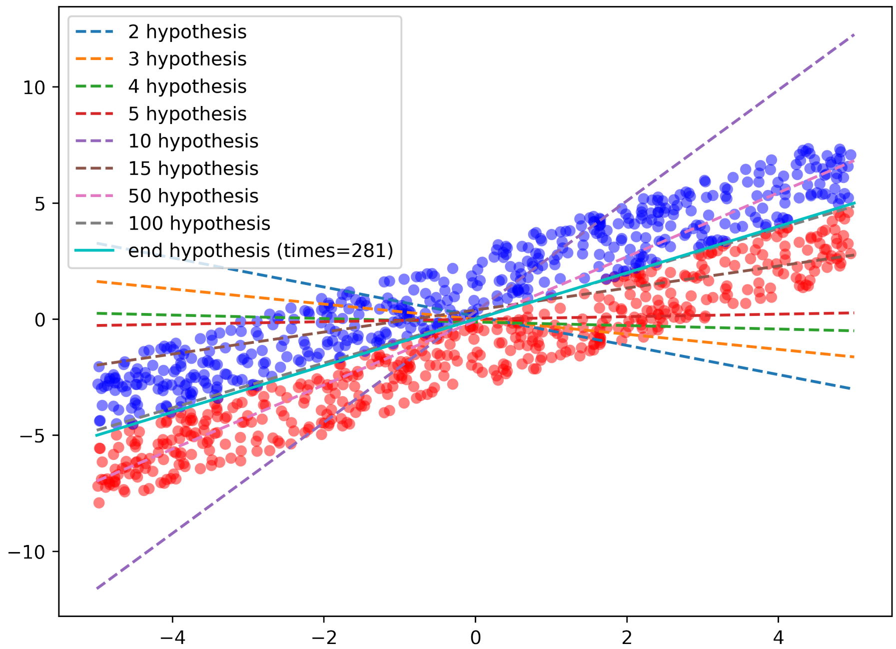
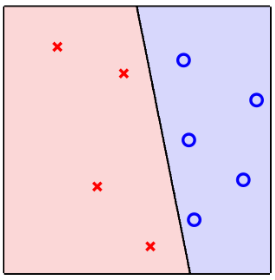


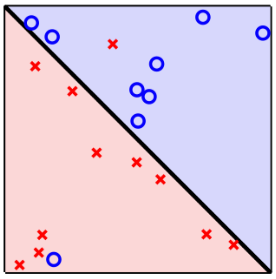
图 6: 感知器学习算法

2.3 可线性分隔数据 Linear Separable Data

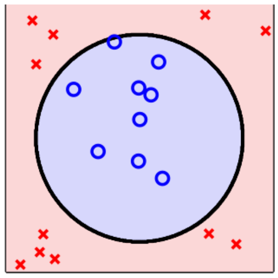
需要注意的是感知器模型不能处理任意类型的数据, 感知器模型有解的必要条件是输入数据 $\mathcal{D}$ 是线性可分的, 即可以用一条直线 ($\mathbb{R}^2$) 或一个 (超) 平面 ($\mathbb{R}^d$) 将两类标签的数据完美分割. 对于无法线性可分的数据, PLA 不会收敛.



(linear separable)



(not linear separable)



(not linear separable)

我们首先考虑线性可分的情况. 假设数据集 $\mathcal{D}$ 是线性可分的, 那么 PLA 算法是否总是收敛的? 因为 $\mathcal{D}$ 是线性可分的, 因此对于 $\forall(\mathbf{x_n}, y_n) \in \mathcal{D}$ 存在 f 与 $\mathbf{w}_f$ 使得

$$f(\mathbf{x_n}) = \operatorname{sign}\left(\mathbf{w}_f^T \mathbf{x}_n\right) = y_n,$$

即

$$\min_n y_n \mathbf{w}_f^T \mathbf{x}_n > 0.$$

假设模型在第 $t, t = 0, 1, 2, \cdots$ 次迭代过程中对点 $(\mathbf{x}_{n(t)}, y_{n(t)})$ 判断错误, 即

$$\operatorname{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)} \Leftrightarrow y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} \leq 0$$

则有

$$\begin{aligned}
\mathbf{w}_f^T \mathbf{w}_{t+1} &= \mathbf{w}_f^T (\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}) \\
&= \mathbf{w}_f^T \mathbf{w}_t + y_{n(t)} \mathbf{w}_f^T \mathbf{x}_{n(t)} \\
&\geq \mathbf{w}_f^T \mathbf{w}_t + \min_n y_n \mathbf{w}_f^T \mathbf{x}_n \\
&> \mathbf{w}_f^T \mathbf{w}_t.
\end{aligned}$$

又因为

$$\begin{aligned}
||\mathbf{w}_{t+1}||^2 &= ||\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}||^2 \\
&= ||\mathbf{w}_t||^2 + 2 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} + y_{n(t)}^2 ||\mathbf{x}_n||^2 \\
&\leq ||\mathbf{w}_t||^2 + ||\mathbf{x}_n||^2 \\
&\leq ||\mathbf{w}_t||^2 + \max_n ||\mathbf{x}_n||^2.
\end{aligned}$$

定义 $R^2 = \max_n ||\mathbf{x}_n||^2, \rho = \min_n y_n \frac{\mathbf{w}_f^T \mathbf{x}_n}{||\mathbf{w}_f||}$, 令 $\mathbf{w}_0 = \mathbf{0}$, 则有:

$$\begin{aligned}
||\mathbf{w}_t||^2 &\leq ||\mathbf{w}_{t-1}||^2 + R^2 \\
&\leq ||\mathbf{w}_{t-2}||^2 + 2R^2 \\
&\cdots \\
&\leq ||\mathbf{w}_0||^2 + tR^2 \\
&= tR^2.
\end{aligned}$$

即 $||\mathbf{w}_t|| \le R\sqrt{t}$. 又因为

$$
\begin{aligned}
\frac{\mathbf{w}_f^T\mathbf{w}_t}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} &= \frac{\mathbf{w}_f^T(\mathbf{w}_{t-1}+y_{n(t-1)}\mathbf{x}_{n(t-1)})}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} \\
&= \frac{\mathbf{w}_f^T\mathbf{w}_{t-1}}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} + \frac{y_{n(t-1)}\mathbf{w}_f^T\mathbf{x}_{n(t-1)}}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} \\
&= \frac{\mathbf{w}_f^T\mathbf{w}_{t-2}}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} + \frac{y_{n(t-2)}\mathbf{w}_f^T\mathbf{x}_{n(t-2)}}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} + \frac{y_{n(t-1)}\mathbf{w}_f^T\mathbf{x}_{n(t-1)}}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} \\
&\dots \\
&= \frac{1}{||\mathbf{w}_t||}\cdot\sum_{i=1}^{t}\frac{y_{n(i)}\mathbf{w}_f^T\mathbf{x}_{n(i)}}{||\mathbf{w}_f^T||} \\
&\ge \frac{1}{||\mathbf{w}_t||}\cdot t\cdot\rho \ge \frac{t\rho}{R\sqrt{t}} \\
&= \frac{\rho}{R}\sqrt{t}.
\end{aligned}
$$

因为 $\frac{\mathbf{w}_f^T\mathbf{w}_t}{||\mathbf{w}_f^T||\cdot||\mathbf{w}_t||} \le 1$, 所以 $\sqrt{t} \le \frac{R}{\rho}$, 即最多迭代 $t = \frac{R^2}{\rho^2}$ 次后, 感知器学习模型的权重向量 $\mathbf{w}_t$ 会收敛到最优权重向量 $\mathbf{w}_f$.

继续上面的 python 程序, 我们来计算 ρ 和 R. 首先计算权重向量 $\mathbf{w}_t$ 向最优权重向量 $\mathbf{w}_f$ 的收敛过程. 在上面的 python 程序中, 两簇数据是在 $y = x$ 直线分别加减一个非负随机数生成的, 因此理论上的最优规则 f 就是 $0 + x - y = 0$, 因此 $\mathbf{w}_f = (0, 1, -1)^T$.

```
wl1 = pd.DataFrame(wl)
wl2 = wl1[["x0", "x1", "x2"]].reset_index(drop=True)
wf = np.array([0,1,-1])
wt = wl2.dot(wf) / (np.array([wl2.loc[v].dot(wl2.loc[v]) for v in wl2.index]) * wf.dot(wf)) **
    0.5
plt.plot(
    range(wt.shape[0]),
    wt,
    label= r"$\frac{\mathbf{w}_{f}^{T}\mathbf{w}_t}{||\mathbf{w}_f||\cdot||\mathbf{w}_t||}$"
)
plt.legend()
```

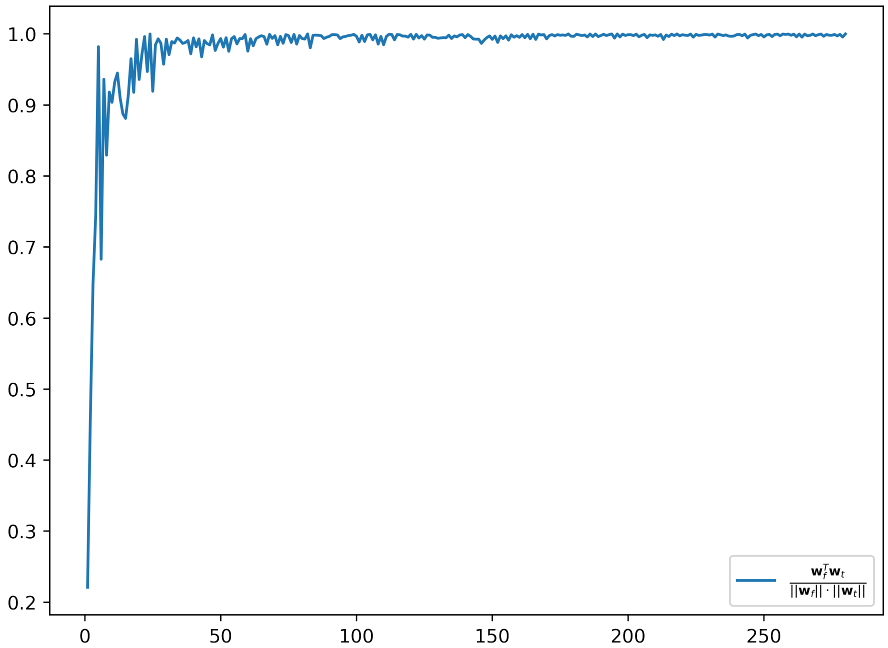


图 7: 权重向量的收敛过程

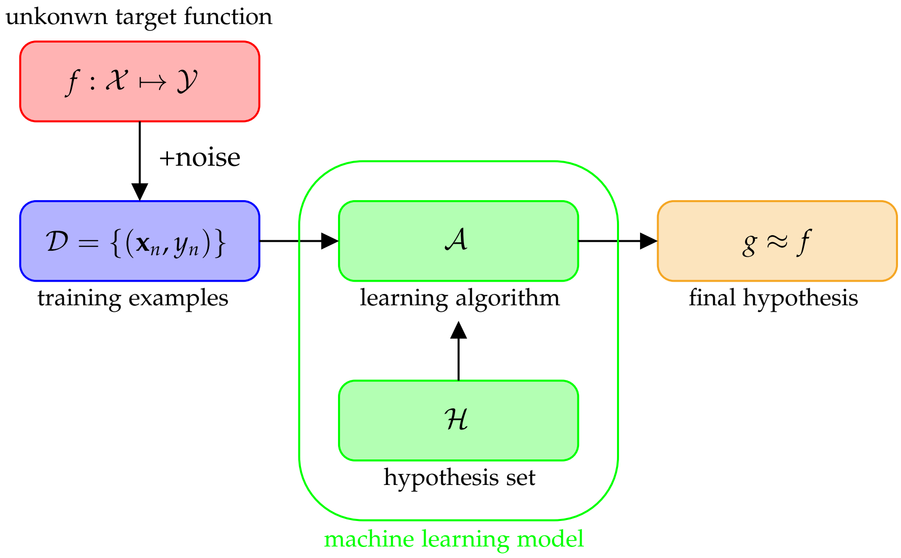
计算 ρ, R 和最大迭代次数:

```
#  计算收敛过程
x_vec = data[["x0", "x1", "x2"]].reset_index(drop=True)
y = data["y"]
R = (np.array([x_vec.loc[r].dot(x_vec.loc[r]) for r in x_vec.index]).max()) ** 0.5
rhot = ((np.array([wf.dot(x_vec.loc[v]) for v in x_vec.index]) * y) / (wf.dot(wf))**0.5)
rho = rhot.min()
maxt = (R/rho)**2
```

运算结果为 39432926 次. 需要注意, 因为该例中数据集 $\mathcal{D}$ 是我们人为定义的, 因此可以知道确切的 $\mathbf{w}_f$ 从而计算 ρ 和最大运行次数. 但是现实中的数据我们是不知道 $\mathbf{w}_f$, 通过假设 $\mathbf{w}_f$ 的存在性, 我们只能证明最大运行次数的存在性, 但无法具体解出.

2.4 非线性可分数据 Non-Linear Separable Data

上面 PLA 算法的前提是数据 $\mathcal{D}$ 是线性可分的, 这样 $\mathbf{w}_f$ 才存在, 但是在获取数据的过程可能存在噪音, 即即使最优的规则 f 是线性函数, 我们获得的数据 $\mathcal{D}$ 可能也不是线性可分的.



在这种情况下, PLA 算法失效, 因为我们无法求出不犯错误的 $\mathbf{w}$. 因此退而求其次, 我们寻找犯错误最少的 $\mathbf{w}$, 即

$$\mathbf{w}_g \leftarrow \arg\min_{\mathbf{w}} \sum_{n=1}^{N} [\![y_n \neq \text{sign}\left(\mathbf{w}^T \mathbf{x}_n\right)]\!].$$

这里符号 $[\![\text{condition}]\!]$ 表示布尔运算. 由此得出修正 PLA 算法: pocket 算法: 由 $\mathbf{w}_0$ 开始, 对于每个 $\mathbf{w}_t$, 随机找出一个错误的估计值 $(\mathbf{x}_n, y_n)$, 令

$$\mathbf{w}_t' = \mathbf{w}_t + y_n \mathbf{x}_n.$$

若 $\mathbf{w}_t'$ 犯的错误比 $\mathbf{w}_t$ 少, 则令

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t',$$

否则继续找下一个随机点. 迭代足够多次, 返回 $\mathbf{w}_p$ 以及 $g = \text{sign}\left(\mathbf{w}_p^T \mathbf{x}\right)$.

因为 pocket 算法需要比较 $\mathbf{w}_t'$ 与 $\mathbf{w}_t$ 犯错的多少, 所以 pocket 算法比 PLA 算法要慢.

继续上文的 python 程序:

```python
#  生成数据
randlist1 = np.array([np.random.uniform(-5,5) for i in range(500)])
randlist2 = np.array([np.random.uniform(-0.1,3) for i in range(500)])
data_p1_0 = pd.DataFrame(
    {
        "x0": 1,
        "x1": randlist1,
        "x2": randlist1 + randlist2,
        "y" : -1
    }
)

data_m1_0 = pd.DataFrame(
    {
        "x0": 1,
        "x1": randlist1,
        "x2": randlist1 - randlist2,
        "y" : +1
    }
)
df = pd.concat([data_m1_0, data_p1_0], axis=0)

plt.figure(figsize=(8,6))
plt.scatter(data_m1_0["x1"], data_m1_0["x2"], c="r", alpha=0.5, linewidths=0)
plt.scatter(data_p1_0["x1"], data_p1_0["x2"], c="b", alpha=0.5, linewidths=0)
plt.savefig("/Users/wanghaoming/Documents/LaTeX_doc/Machine_Learning/non_sep.png", bbox_inches
                                               ='tight', dpi=500)
```

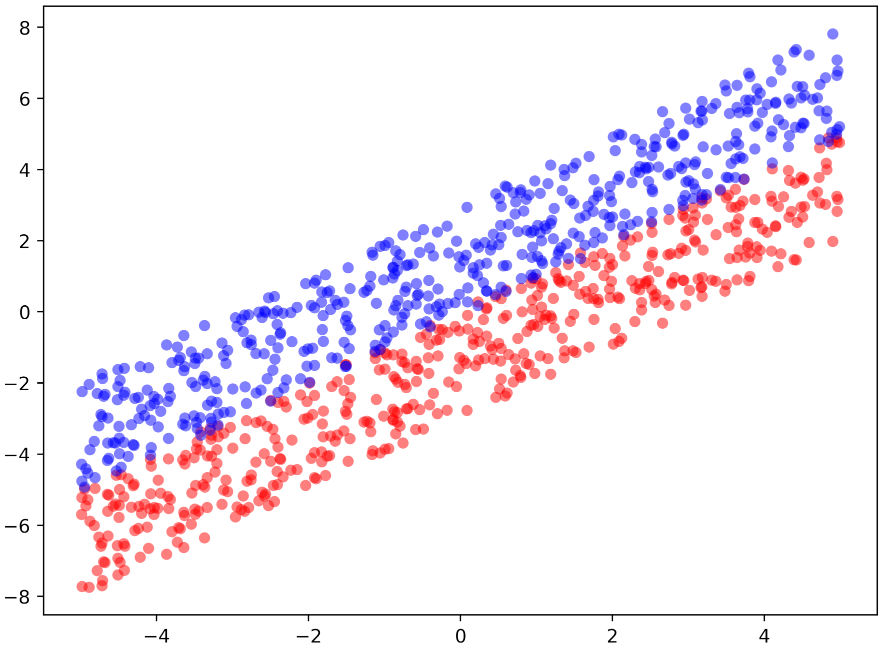


图 8: 非线性分割数据

```
# pocket 算法
plt.figure(figsize=(8,6))
plt.scatter(data_m1_0["x1"], data_m1_0["x2"], c="r", alpha=0.5, linewidths=0)
plt.scatter(data_p1_0["x1"], data_p1_0["x2"], c="b", alpha=0.5, linewidths=0)
w = pd.Series([0, 0, 0], index=["x0", "x1", "x2"]); times = 1;  x=np.linspace(-5,5,100)

mis_p = df[df["y"]==+1][np.dot(df[df["y"]==+1].iloc[:,0:df.shape[1]-1], w) <= 0]
mis_n = df[df["y"]==-1][np.dot(df[df["y"]==-1].iloc[:,0:df.shape[1]-1], w) > 0]
mis = pd.concat([mis_p, mis_n], axis=0)
length = mis.shape[0]
lengthl = []
T = 50000
for i in range(T):
    lengthl.append(length)
    mis_point = mis.iloc[np.random.randint(mis.shape[0]),:]
    w1 = w + mis_point["y"] * mis_point.iloc[0: mis_point.shape[0]-1]
    mis_p = df[df["y"]==+1][np.dot(df[df["y"]==+1].iloc[:,0:df.shape[1]-1], w1) <= 0]
    mis_n = df[df["y"]==-1][np.dot(df[df["y"]==-1].iloc[:,0:df.shape[1]-1], w1) > 0]
    mis1 = pd.concat([mis_p, mis_n], axis=0)
    length1 = mis1.shape[0]
    if length1 < length:
        w = w1
        mis = mis1
        length=length1
lengthl.append(length)

plt.plot(x, -(w[1]/w[2])*x -  (w[0]/w[2]), "c", label=f"end hypothesis (times={T})")
plt.legend()
plt.savefig("/Users/wanghaoming/Documents/LaTeX_doc/Machine_Learning/pocket.png", bbox_inches=
                                                'tight', dpi=500)
```

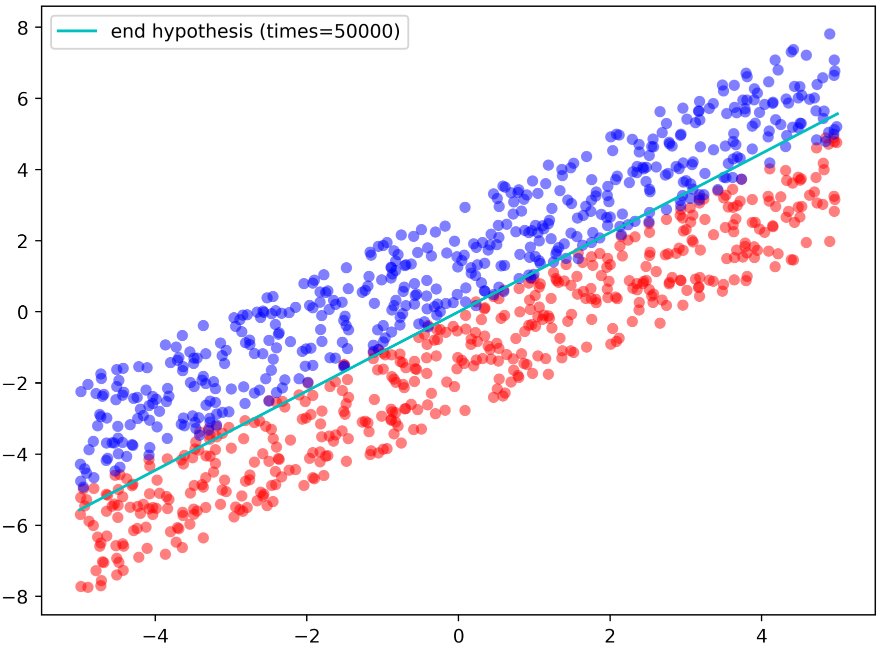


图 9: pocket 算法

```
# 预测错误数
ll = pd.Series(lengthl)
plt.figure(figsize=(8,6))
ll.plot()
plt.xscale("log")
for i in range(1, len(ll)):
    if ll[i] < ll[i-1]:
        plt.text(i ,ll[i], ll[i])
plt.savefig("/Users/wanghaoming/Documents/LaTeX_doc/Machine_Learning/minlen.png", bbox_inches=
                                                'tight', dpi=500)
```

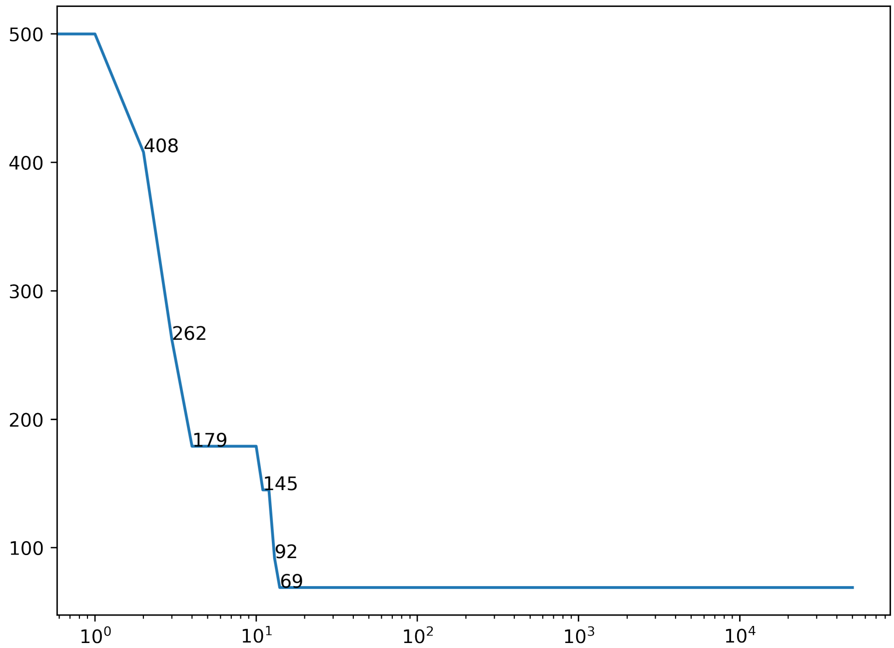


图 10: 预测错误数