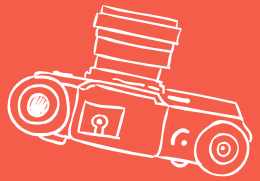
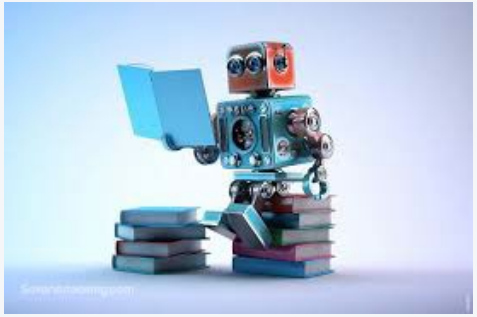
به نام خدا



یادگیری ماشین



یادگیری ماشین

آرش عبدی هجراندوست

arash.abdi.hejrandoost@gmail.com

دانشگاه علم و صنعت

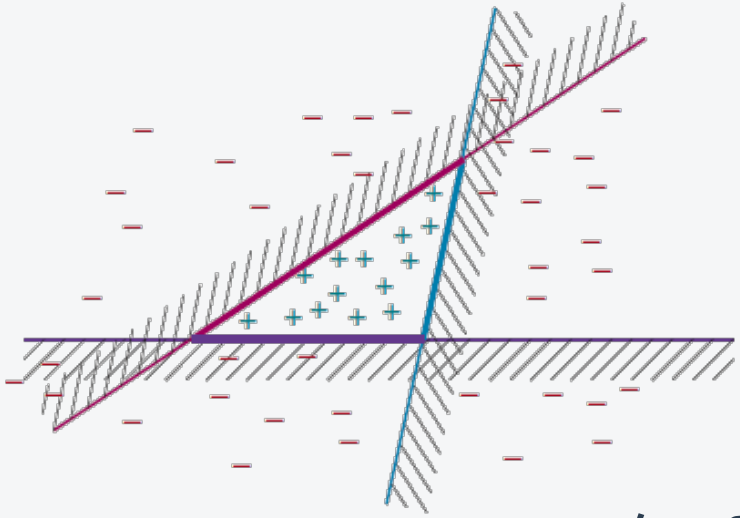
دانشکده مهندسی کامپیوتر

~~نیم سال اول ۱۴۰۱-۱۴۰۲~~

یادگیری گروهی
Ensemble Learning

- چند فرضیه به جای یک فرضیه
- ترکیب نتایج با متوسط گیری، رای گیری و ...
- مدل های پایه می‌توانند ساده باشند
- مزایا:
  - کاهش بایاس
  - کاهش واریانس

کاهش بایاس یا بادگیری گروهی



- ✗ مشابهت با شبکه های عصبی
- ✗ زمان: n برابر
- ✗ زمان مدل پیچیده تر ممکن است نمایی باشد

کاهش واریانس با یادگیری گروهی

- فرضا به تعداد k=5 دسته‌بندی کننده دودوئی داشته باشیم.
  - خروجی نهایی با رای گیری : اکثریت
- برای دسته بندی غلط ۳ خطا لازم است.
- دقت هر دسته بند مستقل :۸۰٪ (فرض استقلال در هر دسته‌بند)
- با فرض آنکه هر کدام روی زیرمجموعه‌ای متفاوت از دیتاست آموزش ببینند:
  دقت هر کدام = ۷۵٪ (زیرا روی انتخاب دیتاست آموزشی، محدودیت گذاشته‌ایم)
- دقت یادگیری گروهی (با فرض مستقل بودن یادگیرنده های پایه که روی دیتاستهای مختلف آموزش می‌بینند) :

$$\binom{5}{3}.0.75^3.0.25^2 + \binom{5}{4}.0.75^4.0.25^1 + \binom{5}{5}.0.75^5 = 0.8965$$

- و با ۱۷ دسته بند دارای دقت های فوق، دقت ۹۹٪ خواهد بود!

- ✗ وقتی احتمال خطا کاهش یابد، واریانس نیز کاهش می‌یابد
- ✗ هرچند فرض استقلال فرض درستی (دقیقی) نیست.
  - بخشی از داده ها یا پارامترهای تنظیم شده بر اساس داده ها و ... مشترک است
  - توزیع داده‌های آموزشی برای همه یکسان است
- ✗ اگر یادگیرنده ها کاملا corelate باشند، عملا یکی هستند و دقت یادگیری گروهی فرقی با یادگیرنده پایه نخواهد داشت.
- ✗ اگر قدری استقلال وجود داشته باشد، دقت بیشتر می‌شود.
- ✗ استقلال بیشتری ← انتظار دقت بیشتر

روش های یادگیری گروهی

- Bagging ✗
- Random forest ✗
- Stacking ✗
- boosting ✗

Bagging

- Bagging = Bootstrap Aggregating
- Bootstrap: In statistics, a sample with replacement
- Bagging : تجمیع با جایگذاری
- آموزش k مدل هر یک با نمونه برداری با جایگذاری از دیتاست
  - هر نمونه بَرداری N گانه می‌تواند مشتمل بر نمونه های تکراری باشد
- ترکیب نتایج با:
  - رای گیری برای دسته بندی
  - متوسط گیری برای تقریب تابع

$$h(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} h_i(\mathbf{x})$$

- ✗ وقتی تعداد داده کم باشد یا بیش برازش محتمل باشد، مناسب است.
- ✗ تلاش دارد پایداری مدل را افزایش دهد ← کاهش واریانس
- ✗ امکان اعمال در هر مدلی (بیشتر درخت تصمیم)
  - درخت تصمیم پایداری کمی دارد: با تغییر جزئی در دیتا ممکن است همه چیز دگرگون شود!
- ✗ امکان اجرای موازی هر مدل به صورت مستقل

جنگل تصادفی : Random Forest

- تجمیع با جایگذاری (Bagging) در درخت تصمیم، درخت‌های مشابه تولید می‌کند
  - ویژگی‌های مهم (با دستاورد اطلاعاتبالا) در ریشه و بالای درخت قرار می‌گیرند.
- جنگل تصادفی، نوعی از تجمیع با جایگذاری درخت تصمیم (Decision Tree Bagging) است
- با هدف تامین بیشتر تنوع در درختان برای کاهش واریانس
- ایده محوری: ایجاد تصادف در انتخاب ویژگی در گره‌های درخت
- در هر گره: زیرمجموعه‌ای تصادفی از ویژگی قابل انتخاب باشند (بر اساس gain)
  - مثلا $\sqrt{n}$ ویژگی (از n تا) برای دسته بندی یا n/3 ویژگی برای تقریب تابع

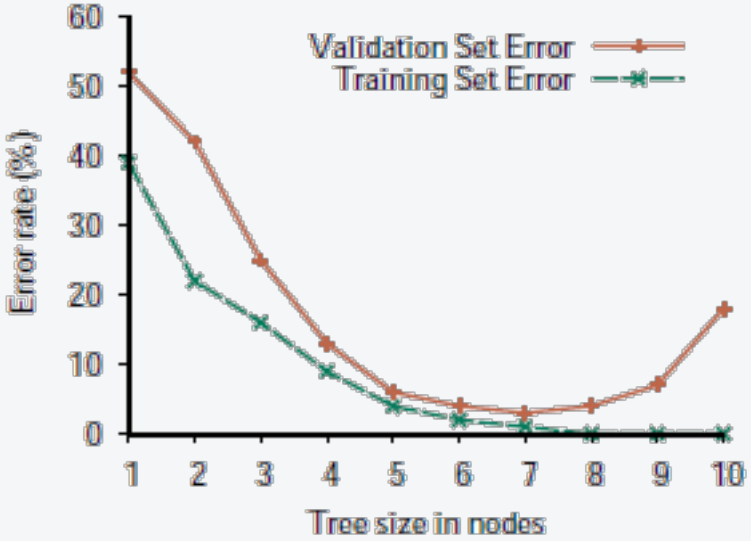
ExtraTrees (Extremely randomized Trees)

- برای ایجاد تنوع بیشتر تر(!):
  - تصادفی کردن انتخاب نقاط برش (split points) در ویژگی ها
  - انتخاب تعدادی نقطه تصادفی در بازه مقادیر ممکن ویژگی جاری (مربوط به گره جاری) و انتخاب مقداری (برای برش) که دستاورد اطلاعات بیشتر می‌دهد.
- با این کار احتمال یکسان بودن درختان جنگل کاهش می‌یابد.
  - هرچند احتمالا نقاط نزدیک به هم برای برش روی ویژگی های یکسان انتخاب شود، اما نقاط برش روی ویژگی های یکسان در درختهای متفاوت، به دلیل تصادف، دقیقا یکسان نخواهند بود و باعث ایجاد تغییرات جزئی در درختان خواهد شد.
- درختان تولید شده با این روش: ExtraTrees

- زمان تولید جنگل تصادفی k برابر زمان تولید یک درخت
  - نیست!
  - تعداد ویژگی قابل انتخاب در هر گره کمتر است
  - نیازی به هرس درخت نیست
    - خود جنگل خودش بیش برازش را می‌کاهد
  - امکان ساخت موازی درختان وجود دارد.
- مثلا برای مساله با ۱۰۰ ویژگی:
  - با ۳ پردازنده موازی
  - k=100 درخت
  - زمان تقریبا برابر با ساخت یک درخت تصمیم با یک پردازنده

- پارامترهای آزاد در جنگل:
  - تعداد درختان
  - تعداد/درصد داده های آموزشی برای هر درخت
  - تعداد ویژگی قابل انتخاب در هر گره
  - تعداد مقادیر تصادفی برای انتخاب نقطه برش
- تنظیم با cross-validation
- استفاده از خطای out-of-bag (به عنوان cross-validation)
  - خطای متوسط روی هر نمونه با در نظر گرفتن درختانی که برای آموزش از آن نمونه استفاده نکرده‌اند.

- جنگل تصادفی هم با افزایش بیش از حد درختان در معرض بیش برازش است:



- این روش کاربردهای بسیاری در صنایع دارد.
- در مسابقات مختلف توسط تیم ها انتخاب میشود
  - اخیرا شبکه های عمیق و gradient boosting جای آن را گرفته اند
- مانند سایر روشهای گروهی امکان کار موازی دارند

پشته سازی – Stacking

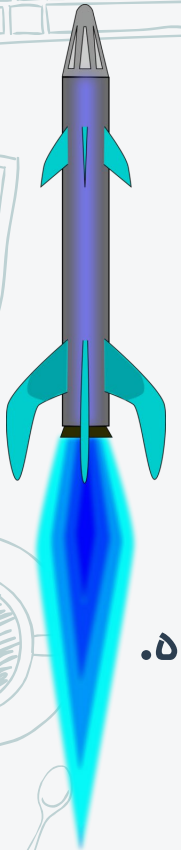
- Stacked Generalization= Stacking
- تجمیع با جایگذاری (Bagging) : استفاده از یک مدل با داده‌های مختلف
- پشته سازی: استفاده از مدل‌های یادگیری مختلف با داده‌های یکسان
- مثلا ترکیب SVM، رگرسیون لاجستیک و درخت تصمیم

در داده های رستوران

- داده ها به سه بخش آموزشی، اعتبارسنجی (validation) و آزمایشی تقسیم شده و در داده‌های آموزشی هر سه مدل آموزش داده می‌شوند.
- در داده های اعتبارسنجی، هر سه مدل خروجی تولید می‌کنند و داده ها با خروجی مدل ها تقویت می‌شود:
- داده اولیه:
  - x1=Yes, No, No, Yes, Some, $$$, No, Yes, French, 0–10;y1=Yes
- داده تقویت شده:
  - x2=Yes, No, No, Yes, Full, $, No, No, Thai, 30–60, **Yes, No, No**; y2=No
- با داده های تقویت شده اعتبارسنجی، مجددا آموزش مدل (مثلا SVM یا حتی مدلی جدید) انجام می‌شود

- در آموزش جدید، هم از اصل داده ها و هم از نتایج و نظرات مدلهای لایه قبل استفاده می‌شود.
- ممکن است صرفا ترکیبی وزن دار از خروجی های لایه قبل تولید شود
  - با وزن‌هایی ثابت
  - یا وزن‌های متناسب و مرتبط با برخی از ویژگیهای اصلی داده
    - وقتی فلان ویژگی، بهمان است، مدل بیسار(!) وزن بیشتری داشته باشد!
- امکان پشته‌سازی با لایه‌های بیشتر نیز وجود دارد.
- پشته‌سازی بایاس را کاهش می‌دهد.
- راست کار تیم‌های مسابقات
  - هر کسی روش کاملا متفاوت خودش را با هر مدلی که بلد است توسعه دهد و نهایتا ترکیب کنند.

Boosting

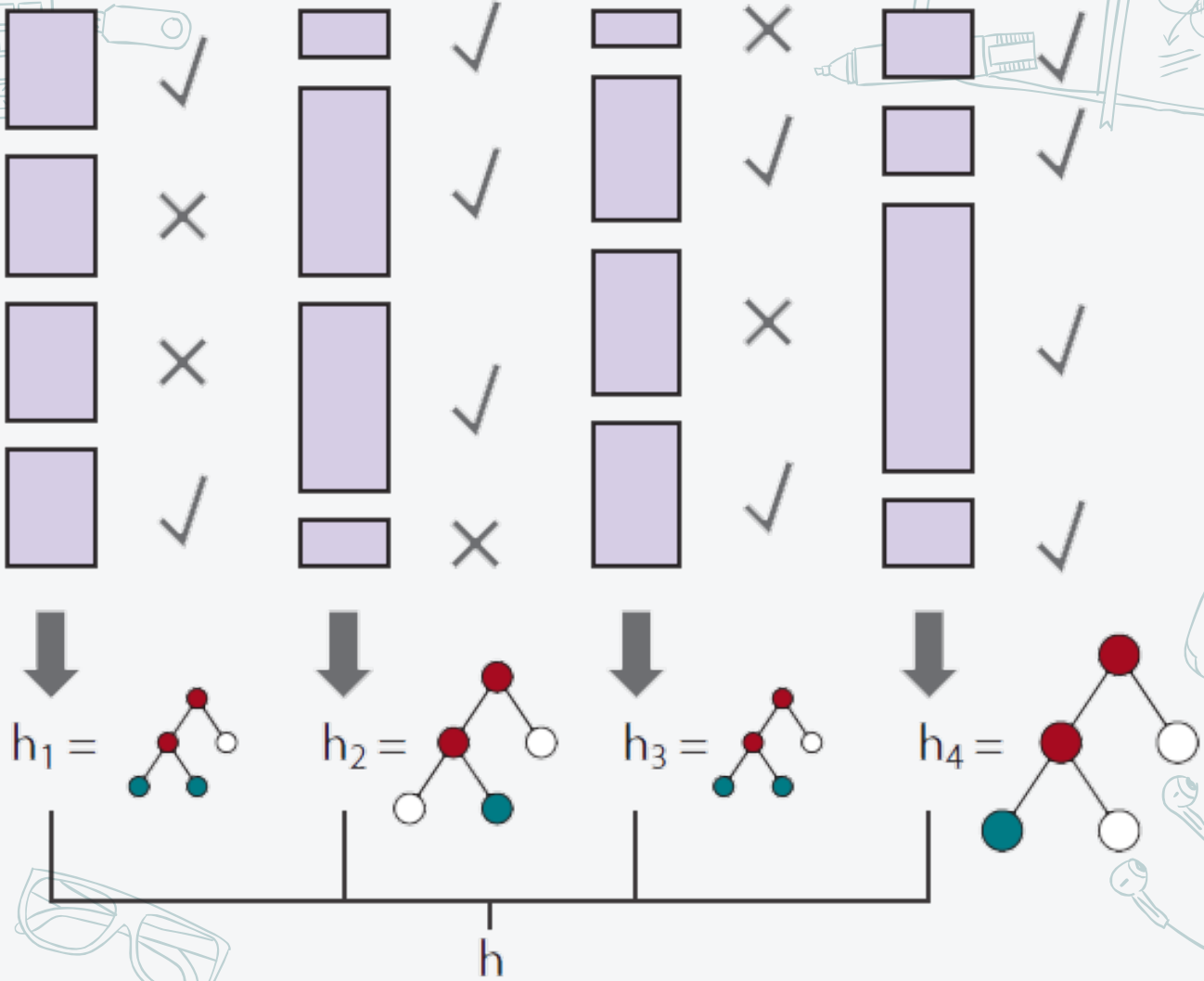


- الگوریتم تقویت (Boosting)
- معروف‌ترین روش یادگیری گروهی
- بر اساس وزن دادن به نمونه‌های آموزشی
  - نمونه وزن‌دار ≡ داشتن کپی‌های بیشتر/کمتر از نمونه متناسب با وزنش
- ایده محوری:
  - به نمونه‌های خطا (در دسته‌بندی یا تقریب)، اهمیت/وزن بیشتر بده.
  - چراکه لابد نمونه‌های سخت‌تری برای دسته‌بندی/تقریب بوده‌اند.

- مجموعه آموزشی ابتدائی با وزن برابر ۱ برای همه نمونه‌ها
- یادگیری (آموزش) فرضیه اول h1
- انتساب وزن بیشتر برای نمونه‌های خطا در مجموعه آموزشی
- یادگیری فرضیه دوم h2 (امید است خطاهای قبلی بهتر دسته‌بندی شده باشند)
- تکرار چرخه فوق به تعداد k بار (پارامتر الگوریتم)
- در نهایت رای گیری وزن دار بین فرضیه‌های k گانه
  - وزن هر فرضیه (z_i) بر اساس میزان صحت خروجی فرضیه‌ها تعیین می‌شود.

$$h(\mathbf{x}) = \sum_{i=1}^{K} z_i h_i(\mathbf{x})$$

- الگوریتم حریصانه (و بدون بازگشت به عقب است).
- فرضیه‌ها به صورت سری تولید می‌شوند و قابلیت اجرای موازی وجود ندارد.


h1 =
h2 =
h3 =
h4 =
h

ADABOOST

- یکی از معروف‌ترین الگوریتم‌های از نوع boosting است.
- معمولا بر روی درخت تصمیم (به عنوان یادگیرنده پایه) اعمال میشود.
- ویژگی مهم: حتی اگر یادگیرنده پایه یک یادگیرنده ضعیف (Weak Learning Algorithm) باشد، آدابوست می‌تواند عملکردی عالی در داده‌های آموزشی داشته باشد اگر تعداد یادگیرنده‌ها (K) به اندازه کافی زیاد باشد.
  - در واقع دقت در داده‌های آموزشی، تقویت (boost) می‌شود
  - بایاس کاهش پیدا می‌کند
  - یادگیرنده ضعیف: دقتی در حدود خروجی تصادفی دارد و فقط اندکی بیشتر
  - برای دسته بندی دو کلاسه، دقت = $50\% + \epsilon$
  - تضمینی برای داده‌های دیده نشده (آزمایشی) وجود ندارد.

وزنی کمتر از ۱ به
داده‌های درست
می‌دهد (معادل وزن
بزرگتر به خطاها

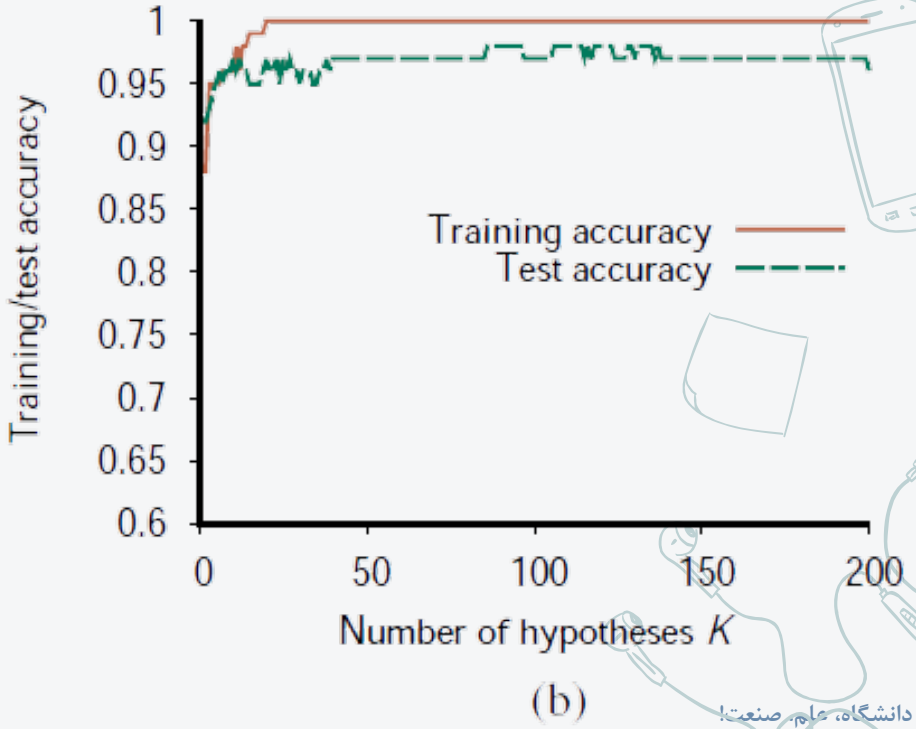
اگر تعداد خطا خیلی
کم باشد، وزن خیلی
زیادی خواهند گرفت
(داده‌های بدقلق!)

وقتی متوقف می‌شود
که دقت از خروجی
تصادفی بدتر شود

function ADABOOST($examples, L, K$) **returns** a hypothesis
  inputs: $examples$, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: $\mathbf{w}$, a vector of N example weights, initially all $1/N$
    $\mathbf{h}$, a vector of K hypotheses
    $\mathbf{z}$, a vector of K hypothesis weights

$\epsilon \leftarrow$ a small positive number, used to avoid division by zero
for $k = 1$ **to** K **do**
  $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
  $error \leftarrow 0$
  for $j = 1$ **to** N **do** // *Compute the total error for* $\mathbf{h}[k]$
    if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
  if $error > 1/2$ **then break** from loop
  $error \leftarrow \min(error, 1 - \epsilon)$
  for $j = 1$ **to** N **do** // *Give more weight to the examples* $\mathbf{h}[k]$ *got wrong*
    if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
  $\mathbf{w} \leftarrow$ NORMALIZE($\mathbf{w}$)
  $\mathbf{z}[k] \leftarrow \frac{1}{2} \log((1 - error)/error)$ // *Give more weight to accurate* $\mathbf{h}[k]$
return $Function(x) : \sum \mathbf{z}_i \mathbf{h}_i(x)$

❌ اعمال ADABOOST روی Decision Stump
درخت تصمیمی فقط با ریشه (یک ویژگی)



Gradient Boosting

- در ADABOOST، فرضیه‌های جدید اضافه می‌شدند تا به داده‌های خطا توجه بیشتر کنند.
- در اینجا نیز فرضیه اضافه می‌شود، اما توجه آنها صرفا برخی از داده‌ها نیست، بلکه هدفشان یادگیری گرادیان بین خروجی درست و خروجی فرضیه (مدل) قبلی است.
- هم برای تقریب هم دسته‌بندی کاربرد دارد
- برای جلوگیری از بیش برازش، می‌توان از Regularization (کالیبره کردن و تنظیم پارامترها) استفاده کرد.
  - مثلا در درخت تصمیم، اندازه/عمق درخت را محدود کرد و ...

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:

   1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

XGboost

- eXtreme Gradient Boosting
- یک پیاده سازی بهینه، پر استفاده در صنعت و در مسابقات یادگیری ماشین
- مدیریت بهینه حافظه، استفاده بهینه از حافظه کش
- امکان اجرای موازی روی پردازنده‌های مختلف (و GPU)
- استفاده از روش نیوتن رافسون (Newton-Raphson) برای حرکت سریع‌تر به سمت هدف (به جای گرادیان استفاده شده در Gradient Boosting)
  - مبتنی بر تخمین درجه ۲ بسط تیلور تابع loss
  - استفاده از مشتق مرتبه اول و دوم loss

Input: training set $\{(x_i, y_i)\}_{i=1}^N$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Algorithm:

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta).$$

2. For m = 1 to M:

   1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

   2. Fit a base learner (or weak learner, e.g. tree) using the training set $\left\{x_i, -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right\}_{i=1}^N$ by solving the optimization problem below:

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2.$$

$$\hat{f}_m(x) = \alpha \hat{\phi}_m(x).$$

   3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.

بوستیدیه۱