به نام خداوند
تئوری یادگیری ماشین
دکتر سیدصالحی
جلسه دوم

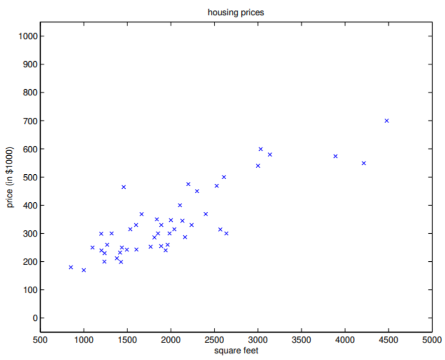
دانشکده ریاضی و علوم کامپیوتر
بهمن ماه ۱۴۰۲

مسئله یادگیری $supervised$

اکنون یک نمونه از مشکلات یادگیری تحت نظارت را بررسی میکنیم. فرض کنید مجموعه داده ای داریم که مناطق زندگی و قیمت ۴۷ خانه را از تهران ارائه می دهد:

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

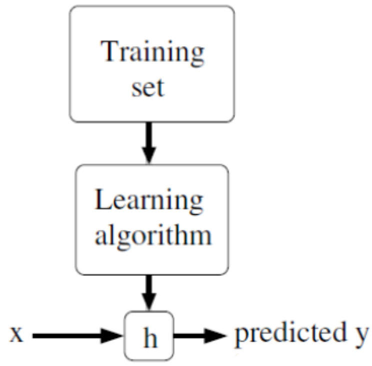
شکل ۱: داده های ۴۷ خانه در تهران



شکل ۲: داده های پلات شده

$h(x)$* مجموعه همه ی خط ها در فضای دو بعدی است و فضای فرضیه ساخته شده $h : X \longrightarrow Y$ برابر با تمام خطوط ممکن در فضای دو بعدی می باشد .

هنگامی که متغیر هدفی که می‌خواهیم پیش‌بینی کنیم پیوسته باشد، مانند مثال قیمت خانه های شهر، مسئله یادگیری را مسئله **رگرسیون** می‌نامیم. وقتی y بتواند فقط تعداد کمی از مقادیر گسسته را به خود



بگیرد و تعداد $output$ ها محدود باشد، ما آن را یک مسئله $classification$ می‌نامیم.

$Linear\ regression$

برای انجام یادگیری تحت نظارت، باید تصمیم بگیریم که چگونه فرضیه های h را مدل کنیم در کامپیوتر. به عنوان یک انتخاب اولیه، فرض کنید تصمیم داریم y را به عنوان تابع خطی x تقریب کنیم:

$$h_w(x) = w_0 + w_1 x_1 + w_2 x_2$$

جایی که w_i ها پارامترها یا وزن های مدل نامیده می شوند که فضای توابع خطی نگاشت $h : X \longrightarrow Y$ را پارامتر می کنند. در اینجا w_0 عرض از مبدا و w_1 شیب می باشد .

برای ساده کردن $notation$، ما x_0 را ۱ قرار میدهیم معرفی می کنیم، به طوری که حاصل میشود:

$$h(x) = \sum_{i=0}^{d} w_i x_i = w^T x,$$

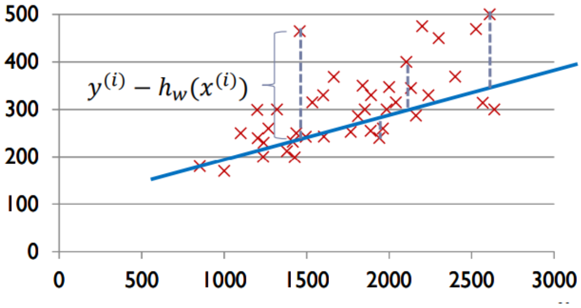
ولی خب بدیهتا تابع ما با دیتای واقعا اختلافاتی دارد و خطاهایی موجود است که به صورت زیر میتوان اندازه گیری کردشان:

$$loss = (y - h(x))^2$$

حال، با توجه به یک داده های آموزشی، چگونه پارامترهای w را انتخاب کنیم یا یاد بگیریم؟ به نظر می رسد یک روش معقول این است که $h(x)$ را به y نزدیک کنیم، حداقل برای مثال های آموزشی که داریم. اکنون تابعی را تعریف می‌کنیم که برای هر مقدار w، میزان نزدیکی $h(x(i))$ به $y(i)$ مربوطه را اندازه می‌گیرد.

$costfunction$ را تعریف می کنیم:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2$$



شکل ۳: یک نمایش از اینکه چگونه یک مدل خطی با مجموعه ای از داده ها مطابقت دارد.

$The\ learning\ algorithm$

هدف اصلی ما این است که w را طوری انتخاب کنیم که $J(w)$ را به حداقل برسانیم. برای هر فرضیه در فضای فرضیه یک تابع هزینه در نظر میگیرم و با مینیمم کردن تابع هزینه پارامتر های بهترین فرضیه رو یاد میگیره که حل این مساله $optimization$ است.

اولین راهی که به ذهن می رسد مشتق گیری و برابر صفر قرار دادن تابع ما است یعنی :

$$\frac{\partial J(w)}{\partial w_0} = 0$$

$$\frac{\partial J(w)}{\partial w_1} = 0$$

در واقع مقدار تابع گرادیان برابر صفر است و دو دستگاه معادلات خطی داریم و میتوانیم این دو را به دست آوریم . برای حالت $multivariate$ ماتریس در نظر می گیریم به نوعی که برای هر سمپل به اندازه ی d فیچر داریم . هر سطر ۱ سمپل را نشان می دهد پس :

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

شکل ۴: ماتریس های سمپل ها، وزن ها و y

که میتوان میزان هزینه را به شکل زیر بدست آورد:

$$J(w) = \sum_{i=1}^{n}(y^{(i)} - h_w(x^{(i)}))^2 = \sum_{i=1}^{n}(y^{(i)} - w^T x^{(i)})^2$$

برای پیاده سازی این الگوریتم، باید مشخص کنیم که عبارت مشتق جزئی در سمت راست چیست. بیایید ابتدا آن را برای حالت اگر ما فقط یک مثال آموزشی داریم (x, y) کار کنیم تا بتوانیم از مجموع در تعریف J صرف نظر کنیم. داریم:

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(w) &= \frac{\partial}{\partial w_j}(h_w(x) - y)^2 \\
&= 2(h_w(x) - y).\frac{\partial}{\partial w_j}(h_w(x) - y)^2 \\
&= 2(h_w(x) - y)^2.\frac{\partial}{\partial w_j}\left(\sum_{i=1}^{d} w_i x_i - y\right) \\
&= 2(h_w(x) - y)x_i
\end{aligned}$$

مشتق تابع هزینه را با توجه به w ها را صفر قرار دهید.

$$J(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \boldsymbol{0} \Rightarrow \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}$$

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\,\boldsymbol{X}^T\boldsymbol{y}$$

چرا $X^T X$ وارون پذیر است ؟

ماتریس سمپل X ها فول رنک می باشد و وارون پذیر است . البته با چالش هایی نیز مواجه هستیم .

بهینه سازی تابع هزینه

تا به اینجا به سادگی جواب به صورت $closedform$ به دست می آمد و مشکلی نداشتیم . اما در اکثر مواقع این اتفاق به راحتی نمی افتد و نیاز به راهکرد دومی داریم که میتوانیم به صورت $iterative$ عمل کنیم . مدل دلخواهی در فضای فرضیه انتخاب کرده و با پارامتر های دلخواه در هر گام طوری حرکت کنیم که تابع هزینه ما کاهش یابد . بدین صورت بعد از مدتی امیدوار هستیم به کمینه تابع هزینه برسیم : هرگاه وابسته

به اینکه در کجا هستیم جهت حرکت مشخص می شود .

احتمالا اولین الگوریتمی که به ذهنمون میرسد الگوریتم $gradient\ descent$ است.

- Steps:
  - Start from $\boldsymbol{w}^0$
  - Repeat
    - Update $\boldsymbol{w}^t$ to $\boldsymbol{w}^{t+1}$ in order to reduce J
    - $t \leftarrow t+1$
  - until we hopefully end up at a minimum

مرور الگوریتم $gradient\ descent$

از w_0 شروع کرده و هدف ما کمینه کردن تابع $J(w)$ است . در خلاف جهت گرادیان حرکت می کنیم پس داریم :

$$w^{t+1} = w^t - \eta \nabla J(w^T)$$

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{w_1} \\ \vdots \\ \frac{\partial J(w)}{w_d} \end{bmatrix}$$

$$\eta : learningrate$$

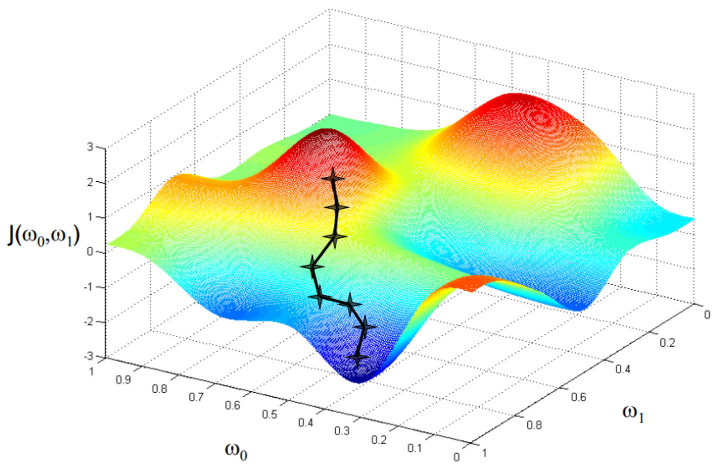
* در $Gradientacsent$ بر خلاف مورد بالا تابعی داریم که میخواهیم آنرا ماکسیمم کنیم . مثلا تابع $score$ و در جهت گرادیان حرکت می کنیم . ...

*اگر به اندازه ی کافی η کوچک باشد ، داریم :

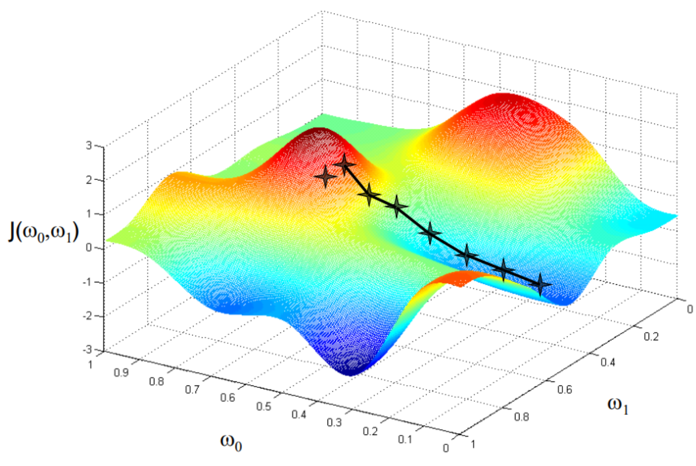
$$J(w^{t+1}) \leq J(w)$$

* η را می توان در هر تکرار به صورت η_t تغییر داد.

معایب کاهش گرادیان: - مشکل کمینه‌های محلی: در الگوریتم کاهش گرادیان، ممکن است به کمینه‌های محلی در تابع هدف برخورد کنیم. این به این معناست که الگوریتم ممکن است در نقاطی که تابع هدف کمینه‌های محلی دارد، متوقف شود و به کمینه‌ی کلی نرسد. - اما وقتی تابع j $convex$ است، تمام کمینه‌های محلی همان کمینه‌های سراسری هستند. که الگوریتم کاهش گرادیان می‌تواند به جواب سراسری همگرا شود.



شکل ۵: $gradient\ descent$ با تابع هزینه غیر محدب - مسیر اول



شکل ۶: $gradient\ descent$ با تابع هزینه غیر محدب - مسیر دوم

انتخاب $learning\ rate$ مناسب در $gradient\ descent$ برای همگرایی کارآمد بسیار مهم است.

- اگر $learning\ rate$ خیلی کم باشد:

 پیشرفت کند است و منجر به همگرایی در طولانی مدت می شود.

 ممکن است در مینیمم های محلی بیفتد و راه حل های کمتر از حد مطلوب را به دست دهد.

- اگر میزان یادگیری خیلی زیاد باشد:

 خطر $overshoot$ راه حل بهینه، ایجاد واگرایی.

 به روز رسانی ها نوسان یا واگرا می شوند که منجر به بی ثباتی می شود.

 $loss\ function$ به شدت نوسان می کند و مانع همگرایی می شود.

بهینه سازی تابع هزینه

$$w^{t+1} = w^t - \eta \nabla J(w^t)$$

$$J(w) = \sum_{i=1}^{n} (y^{(i)} - h_w(x^{(i)}))^2$$

$$w^{t+1} = w^t + \eta \sum_{i=1}^{n} (y^{(i)} - w^T x^{(i)}) x^{(i)}$$

با داشتن کل داده های آموزشی در هر گام بر اساس کل n دیتا ی آموزشی یک گام برداشته می شود و در نهایت به جواب بهینه می رسیم .

کاهش گرادیان تصادفی

کاهش گرادیان تصادفی SGD نوعی از الگوریتم کاهش گرادیان است که برای بهینه‌سازی مدل‌های یادگیری ماشین استفاده می‌شود. این الگوریتم به مشکلات کارایی محاسباتی روش‌های سنتی کاهش گرادیان در مواجهه با مجموعه‌های داده بزرگ در پروژه‌های یادگیری ماشین پاسخ می‌دهد. در SGD، به جای استفاده از کل مجموعه داده در هر تکرار، فقط یک مثال آموزشی تصادفی (یا یک دسته کوچک) برای محاسبه گرادیان و به‌روزرسانی پارامترهای مدل انتخاب می‌شود. این انتخاب تصادفی، تصادفی‌گردی را به فرآیند بهینه‌سازی معرفی می‌کند، به همین دلیل از عبارت ”تصادفی” در کاهش گرادیان تصادفی استفاده می‌شود.

مزیت استفاده از SGD کارایی محاسباتی آن است، به ویژه در مواجهه با مجموعه‌های داده بزرگ. با استفاده از یک مثال یا یک دسته کوچک، هزینه محاسباتی در هر تکرار به طور قابل توجهی کاهش می‌یابد نسبت به روش‌های سنتی کاهش گرادیان که نیاز به پردازش کل مجموعه داده دارند.