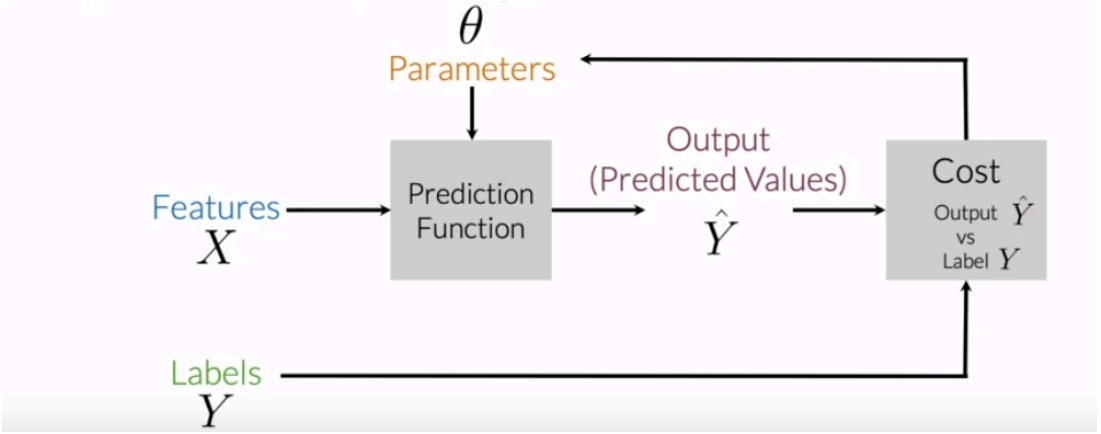
منبع کورس: https://www.coursera.org/specializations/natural-language-processing#courses

**هفته اول:**

- لجستیک رگرسیون(logistic regression algorithm): یک ابزار مهمی است که در nlp کاربردهای زیادی دارد.
- Supervised learning: هدف اینه با داده هایی که داریم خطا (cost) را تا حد ممکن مینیمم کنیم.




- نکات تصویر:
- پارامتر تتا همون w هست که داده ها را نگاشت می ده به یک y
- بهترین تتا اونی هست که y حد با y نزدیک کنه که تابع cost این کار انجام میده.
- اینقدر تتا رو عوض میکنیم تا هزینه مینیمم بشه.

- Sentiment analysis: یک task مرتبط با supervised machine learning classification است.
- برای این تسک یک متن داریم.

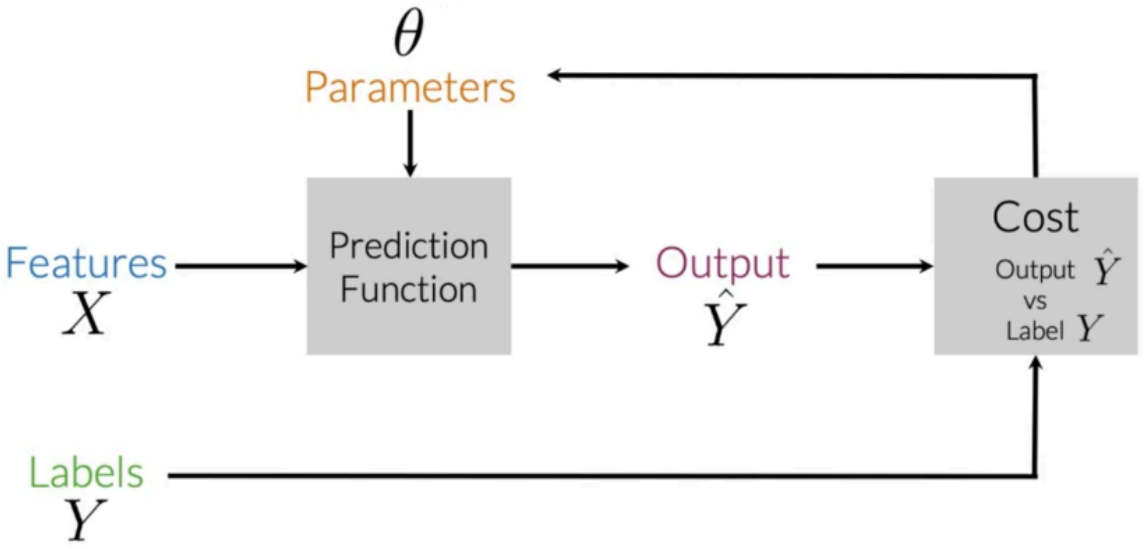
Tweet: I am happy because I am learning NLP

- هدف این تسک اینکه پیشبینی کنیم احساس متن مثبت یا منفی.
- لیبل ها 1 = positive و 0 = negative است.
- برای این تسک از logistic regression classifier استفاده میکنیم.
- مراحل انجام کار Sentiment analysis:
  1. جمع آوری دیتا
  2. استخراج ویژگی
  3. ترین کردن logistic regression classifier با مینیمم کردن تابع هزینه
  4. طبقه بندی کردن نمونه جدید بر اساس مدل train شده

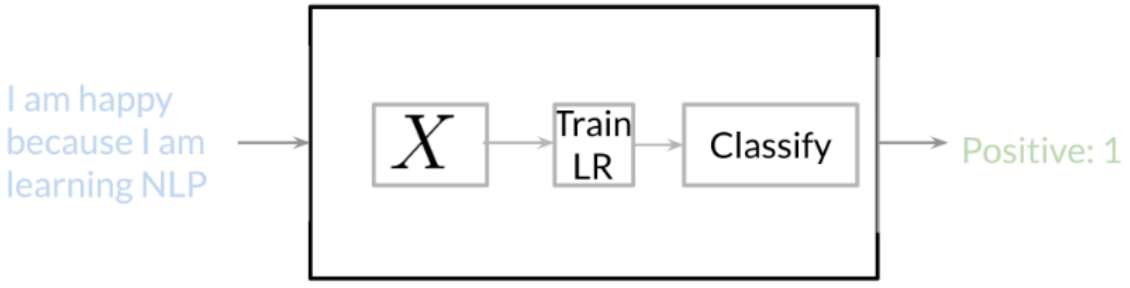
خلاصه مطلب:

# Supervised ML & Sentiment Analysis

In supervised machine learning, you usually have an input $X$, which goes into your prediction function to get your $\hat{Y}$. You can then compare your prediction with the true value $Y$. This gives you your cost which you use to update the parameters $\theta$. The following image, summarizes the process.




To perform sentiment analysis on a tweet, you first have to represent the text (i.e. "I am happy because I am learning NLP ") as features, you then train your logistic regression classifier, and then you can use it to classify the text.

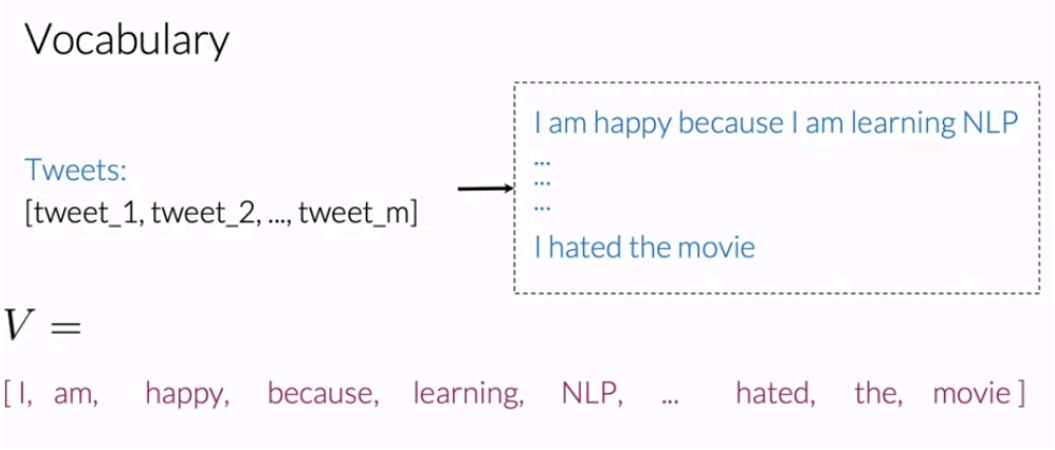



Note that in this case, you either classify 1, for a positive sentiment, or 0, for a negative sentiment.

مرحله دوم: استخراج ویژگی: represent a text as a vector

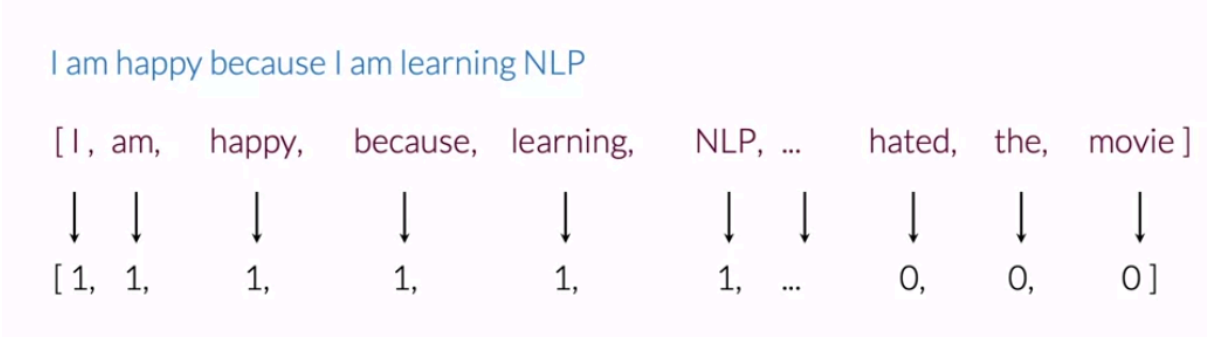
ابتدا جمع اوری کلمه (لیست کلمات یونیک) از مجموعه داده مدنظر است.

(تا بتونی هر متنی را به آرایه عددی encode کنی.)




مرحله دوم، استخراج ویژگی: برای هر متن یک بردار عددی به سایز مجموعه کلماتی که جمع کردی می سازی.

هر کلمه از مجموعه vocabulary رو داشت، 1 میذاری نداشت صفر میذاری.



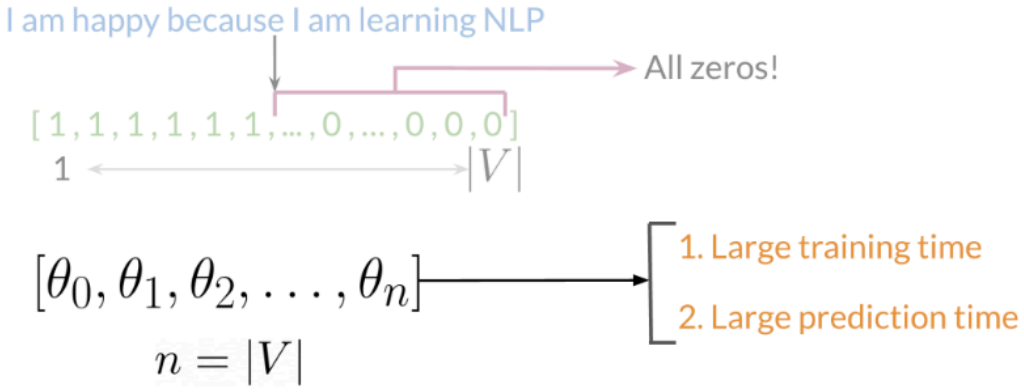

باعث ایجاد ماتریس sparse میشه. که مشکلاتی داره مثل:

- تعداد زیادی ویژگی 0 دارد.
- اگه تعداد کلمات زیاد باشه لجستیک رگرسیون باید تعداد زیادی ویژگی یاد بگیره. که مدت زمان ترین و مدت زمان پیش بینی زیاد میشه.

خلاصه مطالب استخراج ویژگی:

# Vocabulary & Feature Extraction

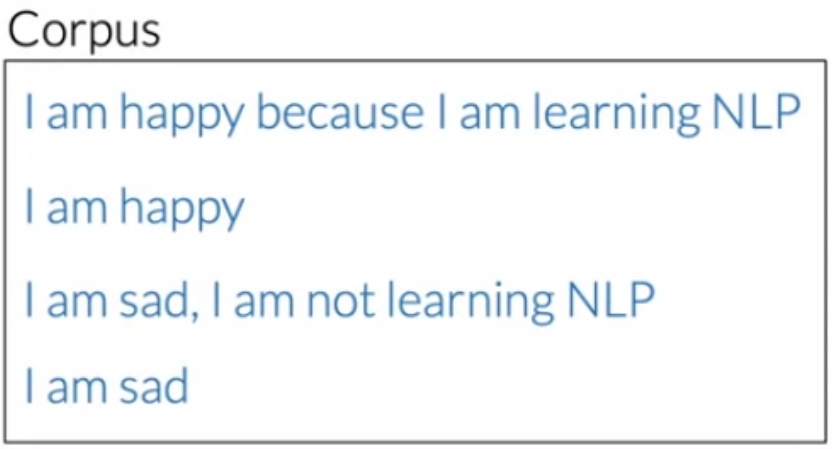
Given a tweet, or some text, you can represent it as a vector of dimension $V$, where $V$ corresponds to your vocabulary size. If you had the tweet "I am happy because I am learning NLP", then you would put a 1 in the corresponding index for any word in the tweet, and a 0 otherwise.




As you can see, as $V$ gets larger, the vector becomes more sparse. Furthermore, we end up having many more features and end up training $\theta$ $V$ parameters. This could result in larger training time, and large prediction time.

استفاده از **generates counts** به عنوان ویژگی برای طبقه‌بند لجستیک رگرسیون:

- مجموعه corpus که شامل 4 تا توییت است.



- یونیک ورد ها رو جدا میکنیم. که 8 کلمه است.

| Vocabulary |
| --- |
| I |
| am |
| happy |
| because |
| learning |
| NLP |
| sad |
| not |

- از این مجموعه corpus تعداد 2 تاش کلاس مثبت و دوتاش کلاس منفی است.

| Positive tweets | Negative tweets |
| --- | --- |
| I am happy because I am learning NLP | I am sad, I am not learning NLP |
| I am happy | I am sad |

- در هر کلاس منفی و مثبت به صورت جداگانه تعداد دفعاتی که کلمات تکرار شده است را بدست میاریم.

## Positive and negative counts

Positive tweets

| |
|---|
| I am happy because I am learning NLP |
| I am happy |

| Vocabulary | PosFreq (1) |
|---|---|
| I | 3 |
| am | 3 |
| happy | 2 |
| because | 1 |
| learning | 1 |
| NLP | 1 |
| sad | 0 |
| not | 0 |

## Positive and negative counts

| Vocabulary | NegFreq (0) |
|---|---|
| I | 3 |
| am | 3 |
| happy | 0 |
| because | 0 |
| learning | 1 |
| NLP | 1 |
| sad | 2 |
| not | 1 |

Negative tweets

| |
|---|
| I am sad, I am not learning NLP |
| I am sad |

- در نهایت ما تعداد دفعات تکرار کلمات در هر کلاس به صورت جدول زیر داریم.

## Word frequency in classes

| Vocabulary | PosFreq (1) | NegFreq (0) |
|---|---|---|
| I | 3 | 3 |
| am | 3 | 3 |
| happy | 2 | 0 |
| because | 1 | 0 |
| learning | 1 | 1 |
| NLP | 1 | 1 |
| sad | 0 | 2 |
| not | 0 | 1 |

- در مرحله استخراج ویژگی با این روش دیگه لازم نیست برای هر توییت به تعداد کلمات موجود در corpus ویژگی استخراج بشه، 3 ویژگی برای هر توییت (xm) کافی است.

ویژگی اول بایاس است.

ویژگی دوم، جمع frequency کلمات کلاس مثبت که در آن توییت هست

ویژگی سوم، جمع frequency کلمات کلاس منفی که در توییت هست

توجه، اگر در توییت ای کلمه ای چندبار تکرار بشه در محاسبه فقط یکبار در نظر گرفته میشه.

مثلا برای توییت زیر استخراج ویژگی رو محاسبه میکنیم.

I am sad, I am not learning NLP

جمع frequency در کلاس مثبت برای توییت بالا، به صورت زیر است:

- کلمه Happy و because در توییت وجود نداره.
- مابقی کلمات(بدون در نظر گرفتن تکرار) frequency شون جمع میشه.

## Feature extraction

| Vocabulary | PosFreq (1) |
|---|---|
| I | 3 |
| am | 3 |
| happy | 2 |
| because | 1 |
| learning | 1 |
| NLP | 1 |
| sad | 0 |
| not | 0 |

جمع frequency برای کلاس منفی 11 است.

## Positive and negative counts

| Vocabulary | NegFreq (0) |
|---|---|
| I | 3 |
| am | 3 |
| happy | 0 |
| because | 0 |
| learning | 1 |
| NLP | 1 |
| sad | 2 |
| not | 1 |

فرمول محاسبه ویژگی به صورت زیر است:

## Feature extraction

I am sad, I am not learning NLP

$$X_m = [1, \sum_w freqs(w, 1), \sum_w freqs(w, 0)]$$

$$\downarrow$$

$$X_m = [1, 8, 11]$$

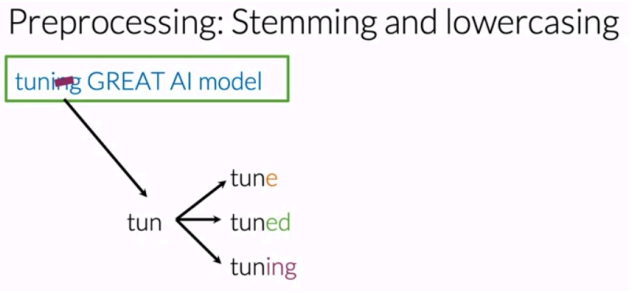
Preprocess: شامل stemming - stop word میشه.

- Stop word مثل حذف punctuation حذف کلمات بی ربط میشه.

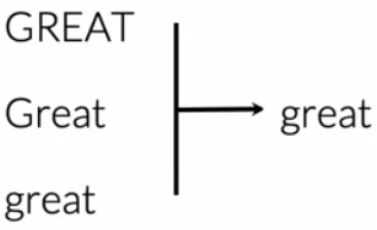
| Stop words |
| --- |
| and |
| is |
| are |
| at |
| has |
| for |
| a |

| Punctuation |
| --- |
| , |
| . |
| : |
| ! |
| “ |
| ‘ |

- stemming : بدست آوردن ریشه کلمات.




- Low case کردن کلمات:



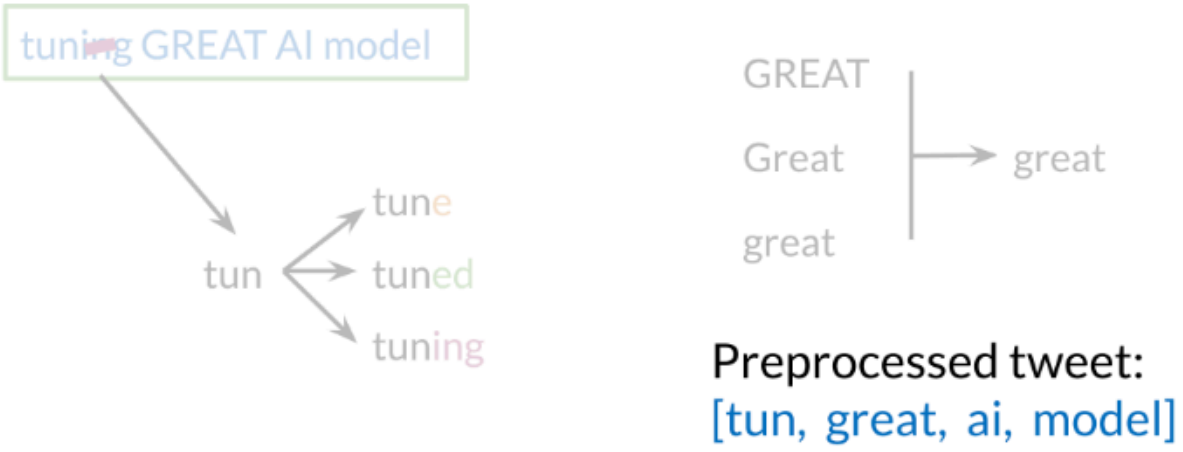

خلاصه مطالب:

# Preprocessing

When preprocessing, you have to perform the following:

1. Eliminate handles and URLs
2. Tokenize the string into words.
3. Remove stop words like "and, is, a, on, etc."
4. Stemming- or convert every word to its stem. Like dancer, dancing, danced, becomes 'danc'. You can use porter stemmer to take care of this.
5. Convert all your words to lower case.

For example the following tweet "@YMourri and @AndrewYNg are tuning a GREAT AI model at https://deeplearning.ai!!!" after preprocessing becomes




$[tun, great, ai, model]$. Hence you can see how we eliminated handles, tokenized it into words, removed stop words, performed stemming, and converted everything to lower case.

کد پیش پردازش:

https://github.com/ElhamHosseini73/LearnNLP/blob/main/preprocessing.ipynb

برای زبان فارسی tools مناسب جهت پیش پردازش hazm است. https://github.com/roshan-research/hazm

ماتریس ویژگی:

$$\begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix}$$

سطرها داده ها و ستون ها ویژگی ها هستند.

کد word frequency :

https://colab.research.google.com/github/ElhamHosseini73/LearnNLP/blob/main/word_frequencies.ipynb

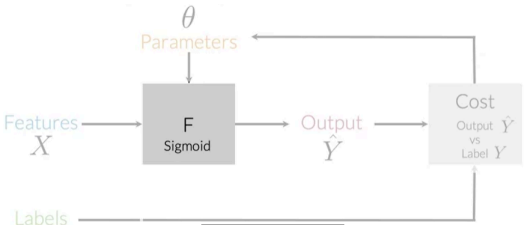
تابع هدف:

Overview لجستیک رگرسیون:

لجستیک رگرسیون از تابع سیگموئیدی استفاده میکنه که خروجی آن احتمال بین 0 و 1 است.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

لجستیک رگرسیون یک supervise است.

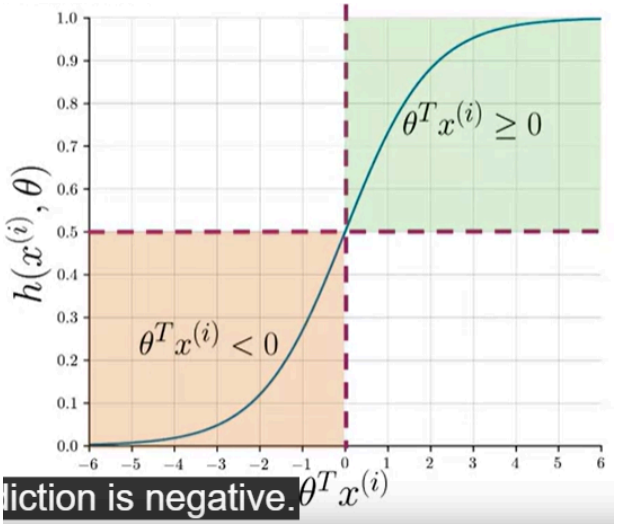



تابع سیگموئید در لجستیک رگرسیون به صورت زیر است.

$$h(x^{(i)}, \theta) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$

تتا ترانسپوز میشه و در داده ضرب میشه.

خروجی عدد بین صفر و یک است که یک رگرسیون هست و برای تصمیم گیری (classification) یک ترشولد در نظر میگیریم که بطور معمول 0.5 است.




اگه تتا ترانسپوز در داده یک عدد مثبت بشه، کلاس مثبت

اگر تتا ترانسپوز در داده یک عدد منفی بشه کلاس منفی

تتا و داده ها به صورت زیر اند که dot product تتا ترانسپوز و داده یک عدد میشه.

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$

Iterate بر روی cost انجام میشه تا زمانی که مجموعه پارامتر تتا را پیدا کنی.

ارزیابی cost function و iteration

1. یک تتا اولیه (initial) داری

$\theta$ — Initialize parameters

2. سپس لجستیک رگرسیون محاسبه میکنیم. (تابع هدف)

$h = h(X, \theta)$ — Classify/predict

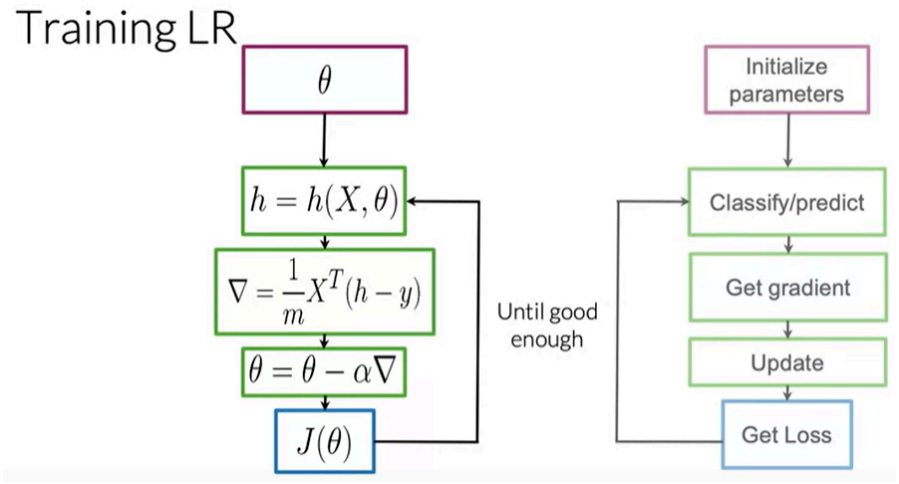
3. سپس گرادیان تابع cost محاسبه میکنیم.

$\nabla = \frac{1}{m} X^T (h - y)$ — Get gradient

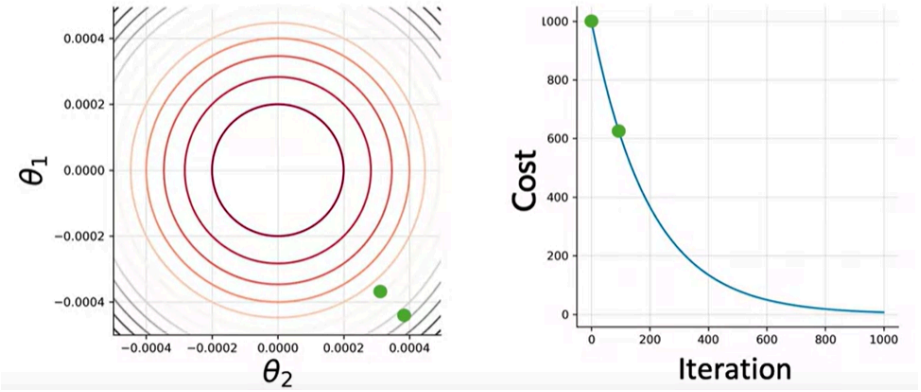
4. آپدیت کردن تتا

$\theta = \theta - \alpha \nabla$ — Update

5. این مراحل ادامه میدیم تا یا به تتا مد نظر برسیم یا به iteration مشخص شده




بعد از 100 بار iterat به نقطه زیر میرسیم.




آیا مدل generalize هست یا نه؟

1. داده تست جمع اوری میکنیم
2. داده تست به مدل میدیم و نتایج محاسبه میکنیم

$$\begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 = \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

در این شکل برای هر داده تست تابع سیگموئید ( تتا در مرحله train بدست آوردیم ) محاسبه میکنیم. و بعد یک ترشولد داریم اگه بزرگتر مساوی 0.5 بود به کلاس یک و در غیر این صورت به کلاس صفر نسبت میدیم.

3. حالا چون label داده های تست داریم صحت با فرمول زیر محاسبه میکنیم.

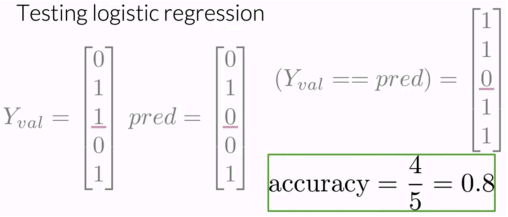
$$\sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$$

4. اگه y پیشبینی شده با label واقعی یکسان بود، یک در غیر این صورت صفر قرار میدیم.

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m == Y_{val_m} \end{bmatrix}$$

5. مثال




در مثال بالا 5 تا داده تست داشتیم که 4 تاشون Y پیشبینی شده با label ها برابر بوده پس طبق فرمول صحت 80 درصد شده.

قبل از اینکه وارد فرمول cost function بشیم، کمی در مورد انواع cost function ها در classification ها بدونیم.

**انواع cost function در طبقه بندها:** به صورت کلی 3 دسته است:

1. استفاده از cost function رگرسیون
2. باینری classification
3. مولتی کلاس classification

روش اول: رگرسیون: این روش error base on the distance است (یعنی مینیمم کردن خطا بین خروجی واقعی و خروجی پیشبینی شده)

روش دوم: یکی از مدل های مشهور روش دوم cross-entropy است.

Cross-entropy تنها یک خروجی (یا 0 یا 1) داره.

**آموزش cross-entropy:**

در classification معمولا از cross-entropy استفاده میشود.

تابع cost کراس آنتروپی به صورت زیر است:

$$loss = \begin{cases} -\log y_p & y_t = 1 \\ -\log 1 - y_p & y_t = 0 \end{cases}$$

ما یک تابع دو ضابطه‌ای داریم که می‌گوید:

- اگر لیبل واقعی یک داده 1 هست (yt=1)، از ضابطه log yp– استفاده کن.
- اگر لیبل واقعی یک داده 0 است (yt=0) از ضابطه پایینی log1-yp– استفاده کن.

به راحتی می‌توانیم این تابع دو ضابطه‌ای را یکپارچه کرده و به شکل زیر بنویسیم:

$$L = -\big(y_t \log y_p + (1 - y_t) \log\big(1 - y_p\big)\big)$$

مگر در X فقط یک نمونه داریم؟ طبیعتا N نمونه داریم. پس قاعدتا باید cost تک‌تکشان را حساب کنیم و بعد میانگین بگیریم.

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_t \log y_p + (1 - y_t) \log\big(1 - y_p\big)$$

واجب است که این فرمول را حفظ کنید.

تابع هزینه:

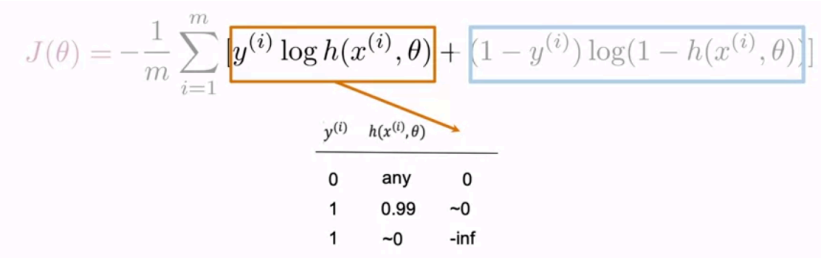
Cost function for logistic regression

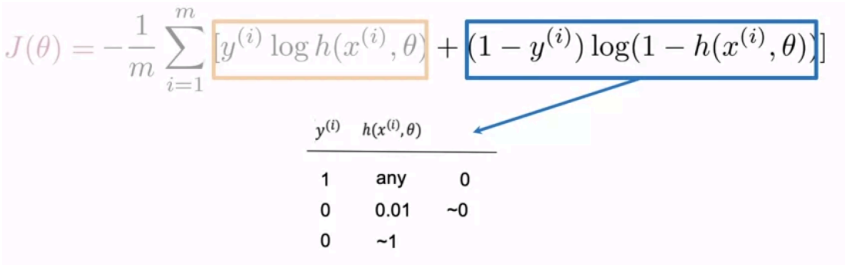
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \log h(x^{(i)}, \theta) + (1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))]$$

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

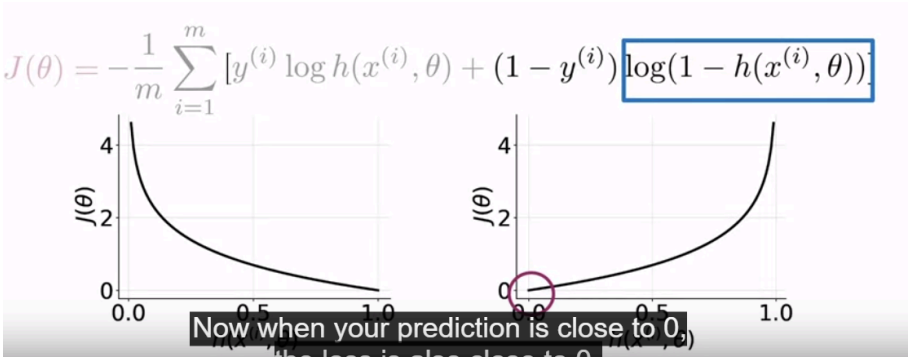
M: تعداد دادهای آموزش است. که تابع خطا باید میانگین روی همه نمونه ها محاسبه بشه.

نکته) در این فرمول اگر لیبل با پیشبینی برابر باشه مقدار صفر میشه اگه برابر نباشه مقدار بینهایت میشه. پس هدف ما این است که تابع cost را مینیمم کنیم.




نمودارهاش :




چرا برای مسائل طبقه بندی cross-entropy انتخاب مناسبی است؟

انتظار ما در مسئله طبقه بندی چیه؟ انتظار ما در مسائل طبقه بندی اینه که اولا حداقل خطای تصمیم گیری داشته باشد، دوما تصمیمات را با سطح اطمینان بالایی بگیرد. یعنی وقتی داده مربوط به کلاس یک هست، شبکه با سطح اطمینان بالایی کلاس یک تشخیص دهد!

برای تصمیمات درستی که با سطح اطمینان بالا گرفته شده، خطای کمتری اختصاص دهیم و برای تصمیمات درست با سطح اطمینان پایین خطای زیادی اختصاص دهیم! از اونجا که Cross-Entropy دقیقا این دو هدف را در خود دارد، برای مسائل طبقه‌بندی انتخاب خیلی خوبی است.

تفاوت cross-entropy با MSE با یک مثال عملی برای مسائل چند کلاسه؟

فرض کنید یک مسئله سه کلاسه داریم و در آن شبکه عصبی برای سه نمونه A,B و C خروجی های زیر را تخمین زده است. حال قرار است که هزینه تصمیم گیری شبکه به ازای هر نمونه محاسبه شده و شبکه براساس آن جریمه وزنهای خود را در جهت کاهش هزینه تنظیم کنید.

| | Ytrue | Ypred |
|---|---|---|
| A: | [1,0,0] | [0.7,0.2,0.1] |
| B: | [0,1,0] | [0.0,0.5,0.5] |
| C: | [0,0,1] | [0.3,0.2,0.5] |

قبل از اینکه میزان جریمه تابع هزینه برای شبکه عصبی را محاسبه کنیم و بیاییم ببینیم که طبق صحبتهایی که داشتیم، هزینه ها باید به چه صورت محاسبه شوند.

خب اولا میشه گفت شبکه لیبل هر سه نمونه را درست تخمین زده ولی با سطح اطمینان پایین! پس باید شبکه به ازای هر سه نمونه جریمه بشه! بلا استثنا! ولی از اونجا که لیبل نمونه‌ی اول را با سطح اطمینان بالاتری در مقایسه با بقیه نمونه ها تخمین زده، پس باید نسبت به بقیه جریمه کمتری داشته باشد. و لیبل دو نمونه بعدی را با سطح اطمینان پایین و البته یکسانی تخمین زده! پس باید جریمه زیادی برای آنها محاسبه شود. از طرفی چون هر دو را با یک سطح اطمینان تصمیم گیری کرده، پس باید به یک میزان جریمه شوند! چرا؟ چون هدف ما اینه که به شبکه بفهمانیم که تخمین درست با سطح اطمینان بالا برای ما اهمیت داره، اینکه بقیه رو چی تخمین زدی مهم نیست. کلاس درست رو با سطح اطمینان بالا باید تخمین بزنی! اینطوری شبکه عصبی متوجه میشه که باید لیبلهای درست را با سطح اطمینان بالاتری تخمین بزنه! حالا ببینیم که تابع هزینه چه جریمه ای را محاسبه می‌کنند؟ همانطور که میبینیم تابع هزینه Cross-Entropy طبق انتظاری که ما داشتیم جریمه ها را محاسبه کرده است. ولی MSE از اونجا که هدفش روی مینیمم کردن اختلاف است، برای دو نمونه دوم و سوم هزینه متفاوتی محاسبه می‌کند. به همین دلیل است که Cross-Entropy برای مسائل طبقه بندی تابع هزینه مناسب تری نسبت به MSE است.

**Cross-Entropy**

$$\text{cost}_1 = -[1 * \log(0.7) + 0 * \log(0.2) + 0 * \log(0.1)] = 0.35$$
$$\text{cost}_2 = -[0 * \log(0) + 1 * \log(0.5) + 0 * \log(0.5)] = 0.69$$
$$\text{cost}_3 = -[0 * \log(0.3) + 0 * \log(0.2) + 1 * \log(0.5)] = 0.69$$

**squared error**

$$\text{cost}_1 = [(1-0.7)^2 + (0-0.2)^2 + (0-0.1)^2] = 0.14$$
$$\text{cost}_2 = [(0-0)^2 + (1-0.5)^2 + (0-0.5)^2] = 0.5$$
$$\text{cost}_3 = [(0-0.3)^2 + (0-0.2)^2 + (1-0.5)^2] = 0.38$$

چرا تابع cost function به این صورت طراحی شد:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h\left(x^{(i)},\theta\right) + \left(1-y^{(i)}\right)\log\left(1-h\left(x^{(i)},\theta\right)\right)\right]$$

ما میخوایم احتمال $P(y|x^{(i)},\theta)$ بیشترین مقدار بشه.

حالا بخواهیم برای y=0 , y=1 تابع احتمال ترکیبی بنویسیم میشه:

$$P(y|x^{(i)},\theta) = h(x^{(i)},\theta)^{y^{(i)}}(1-h(x^{(i)},\theta)^{(1-y^{(i)})}$$

در فرمول بالا وقتی y=1 باشه میشه قسمت اول. وقتی y=0 باشه میشه قسمت دوم

وقتی y = 0 است $(1 - h(x^{(i)}, \theta)$ باید یک بشه چون ما میخوایم احتمال y مقدار یک باشد، پس $h(x^{(i)}, \theta)$ باید صفر باشه یعنی با lable برابر باشه.

وقتی y =1 است $h(x^{(i)}, \theta)$ باید یک بشه
پس فرمول بالا درسته.

حالا این فرمول بدست اومده برای یک نمونه است. ما چندتا نمونه داریم، چون نمونه های ما مستقل و هم توزیع هستند، احتمال توام شون میشه حاصل ضرب احتمالات آنها،

$$L(\theta) = \prod_{i=1}^{m} h(\theta, x^{(i)})^{y^{(i)}} (1 - h(\theta, x^{(i)}))^{(1-y^{(i)})}$$

که به این میگن ماکزیمم لایکلیهود.

نکته) آموزش لایکلیهود: هدف لایکلیهود برآورد پارامترهای مدل است.
تابع لایکلیهود همان تابع احتمال نیز می‌گویند ولی:
تابع احتمال: احتمال رویدادی (بدون هیچ مشاهده‌ای) با توجه به معلوم بودن مدل و پارامترهای آن است
تابع لایکلیهود: با توجه به مشاهدات انجام شده، در مورد مقدارهای محتمل برای پارامتر مدل تصمیم‌گیری می‌کنیم.

توجه) اگه تعداد داده ها زیاد باشه. خروجی فرمول بالا نزدیک به صفر میشه، چون h بین 0 و 1 است. برای رفع این مشکل از log استفاده میکنیم. (فقط در فضای متفاوت میبره)

نکات log:

$$\log a * b * c = \log a + \log b + \log c$$

$$\log a^b = b \log a$$

چون احتمال میخواستیم ماکزیمم کنیم به صورت زیر می نویسیم.

$$\max_{h(x^{(i)},\theta)} \log L(\theta) = \log \prod_{i=1}^{m} h(x^{(i)}, \theta)^{y^{(i)}} (1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$
$$= \sum_{i=1}^{m} \log h(x^{(i)}, \theta)^{y^{(i)}} (1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$
$$= \sum_{i=1}^{m} \log h(x^{(i)}, \theta)^{y^{(i)}} + \log(1 - h(x^{(i)}, \theta)^{(1-y^{(i)})}$$
$$= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}, \theta) + (1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))$$

چون میانگین میخوایم حساب کنیم تقسیم بر کل داده ها میکنیم.

$$\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h(x^{(i)},\theta) + (1-y^{(i)}) \log(1-h(x^{(i)},\theta))$$

این فرمول میخوایم ماکزیمم کنیم منفی اون رو مینیمم میکنیم.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \left[y^{(i)} \log h\left(x^{(i)},\theta\right) + \left(1-y^{(i)}\right) \log\left(1-h\left(x^{(i)},\theta\right)\right)\right]$$

به این فرمول میگن تابع هزینه.
حالا وقتی میخوایم پارامترها رو به نحوی پیدا کنیم که تابع هزینه مینیمم بشه از مشتق استفاده میکنیم.
مشتق تابع بالا به صورت زیر لینک زیر محاسبه میشود.
https://www.coursera.org/learn/classification-vector-spaces-in-nlp/supplement/afcaR/optional-logistic-regression-gradient

و در نهایت به فرمول زیر میرسه.

$$\nabla_{\theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h^{(i)} - y^{(i)})x_j$$

i ایندکس داده های train است.
j ایندکس وزن تتا است، پس xj ویژگی مرتبط با وزن است.

در نهایت همیشه تتا جدید به صورت زیر اپدیت میشه.

$$\theta_j = \theta_j - \alpha \times \nabla_{\theta_j} J(\theta)$$

آلفا مقداری هست که خودمون انتخاب میکنیم برای کنترل بزرگی تتا
ابعاد تتا، (n*1) است، n برابر با تعداد ویژگی ها است.

از آنجایی که داده های ما به صورت ماتریس میباشد. در محاسبه فرمول های تابع هدف، تابع هزینه، آپدیت تتا باید ابعاد داده ها رو در نظر بگیریم، زیرا به صورت ماتریسی هستند
تابع هدف: در تابع هدف زیر، x و تتا یک ماتریس هستند.
ابعاد x برابر m*3
ابعاد تتا برابر 3*1

$$h(x^{(i)},\theta) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$

تابع هزینه:

$$J = \frac{-1}{m} \times \left(\mathbf{y}^T \cdot log(\mathbf{h}) + (\mathbf{1}-\mathbf{y})^T \cdot log(\mathbf{1}-\mathbf{h})\right)$$