CS 229 Machine learning

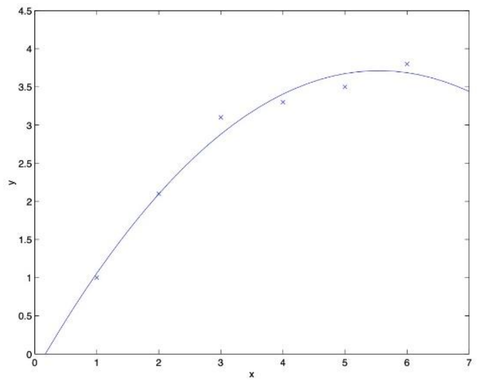
https://aman.ai/cs229/newton-method/

Locally Weighted Linear Regression

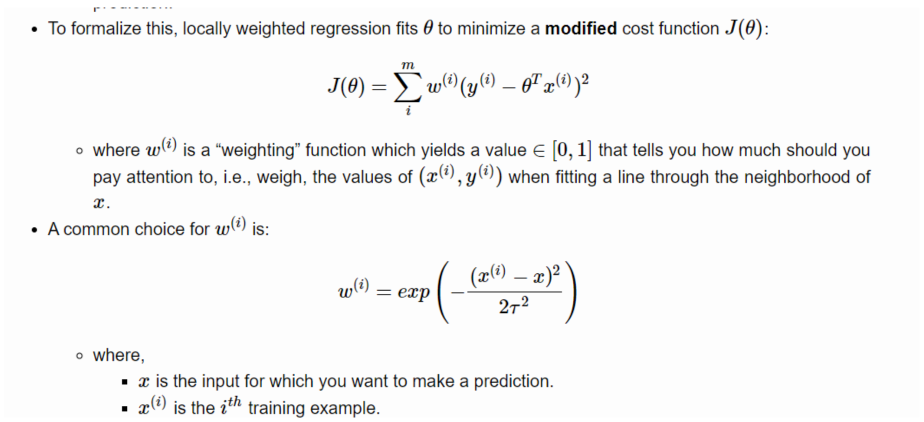
https://aman.ai/cs229/locally-weighted-linear-regression/

1 Goal, key idea, and challenges

- **Goal**: an algorithm that modifies linear regression to make it fit non-linear functions.
- **Key idea**: To fit a dataset that is inherently non-linear, a straight line wouldn't be the best choice and we'd have to explore polynomials.
- **Example**: if we had added an **extra feature** x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data (cf. figure below).



- **Challenges**: adding high-order features results in overfitting.
- Modified cost function:



- To formalize this, locally weighted regression fits θ to minimize a **modified** cost function $J(\theta)$:

$$J(\theta) = \sum_{i}^{m} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$$

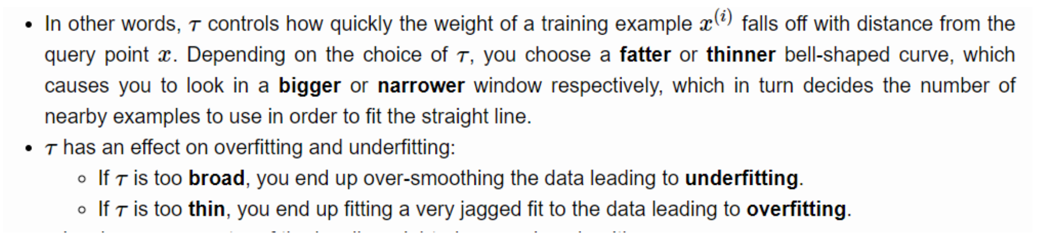
  - where $w^{(i)}$ is a "weighting" function which yields a value $\in [0, 1]$ that tells you how much should you pay attention to, i.e., weigh, the values of $(x^{(i)}, y^{(i)})$ when fitting a line through the neighborhood of x.
- A common choice for $w^{(i)}$ is:

$$w^{(i)} = exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

  - where,
    - x is the input for which you want to make a prediction.
    - $x^{(i)}$ is the i^{th} training example.

Where τ is the bandwidth parameters

2 Bandwidth Parameter



- In other words, τ controls how quickly the weight of a training example $x^{(i)}$ falls off with distance from the query point x. Depending on the choice of τ, you choose a **fatter** or **thinner** bell-shaped curve, which causes you to look in a **bigger** or **narrower** window respectively, which in turn decides the number of nearby examples to use in order to fit the straight line.
- τ has an effect on overfitting and underfitting:
  - If τ is too **broad**, you end up over-smoothing the data leading to **underfitting**.
  - If τ is too **thin**, you end up fitting a very jagged fit to the data leading to **overfitting**.

Based on the analysis above, when τ is small, the window is flat; when τ is big, the window is narrower.

3 Bandwidth Parameter

- The major benefit of locally weighted regression is that fitting a non-linear dataset doesn't require fiddling manually with features.
- Locally weighted linear regression is useful when you have a relatively low dimensional dataset, i.e., when the number of features n isn't too large, say 2 or 3.

Newton’s Method

1 Motivation

- Gradient ascent is a very effective algorithm, but it takes baby steps at every single iteration and thus needs a lot of iterations to converge.
- **Newton’s method**: is another algorithm for maximizing $\ell(\theta)$ which allows you to take much bigger steps and thus converge much earlier than gradient ascent, with usually an order of magnitude lesser number of iterations. However, each iteration tends to be more expensive in this case.

2 Newton’s method-one dimension

- **Problem**: suppose we have a function $f: R \mapsto R$, and we wish to find a value of θ such that $f(\theta) = 0$. Here, $\theta \in R$ is a real number.
- **Update rule**: the update rule for Newton’s method is given by

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

- **Quadratic Convergence (reduce the number of iterations)**: Assume that on one iteration Newton’s method has 0.01 error (i.e., on the X-axis, you’re 0.01 away from the true minimum or where f'(·) is equal to 0). After the iteration, the error could quadratically jump to 0.0001, and after two iterations it can further jump to 0.00000001.

3 Newton’s method-vector

- **Update rule**: The generalization of Newton’s method to a multi-dimensional setting (also called the **Newton-Raphson method**) is given by

$$\theta := \theta - H^{-1}\nabla_\theta \ell(\theta)$$

  - where,
    - $\nabla_\theta \ell(\theta)$ is the vector of partial derivatives of $\ell(\theta)$ with respect to the θ_i’s.
    - H is an $n \times n$ matrix (actually, $(n+1) \times (n+1)$, assuming that we include the intercept term) called the **Hessian**. The Hessian thus becomes a squared matrix $\mathbb{R}^{(n+1)\times(n+1)}$ where the $(n+1)$ dimension is equal to the size of the parameter vector θ.
- The Hessian matrix is defined as the matrix of second-order partial derivatives whose entries are given by,

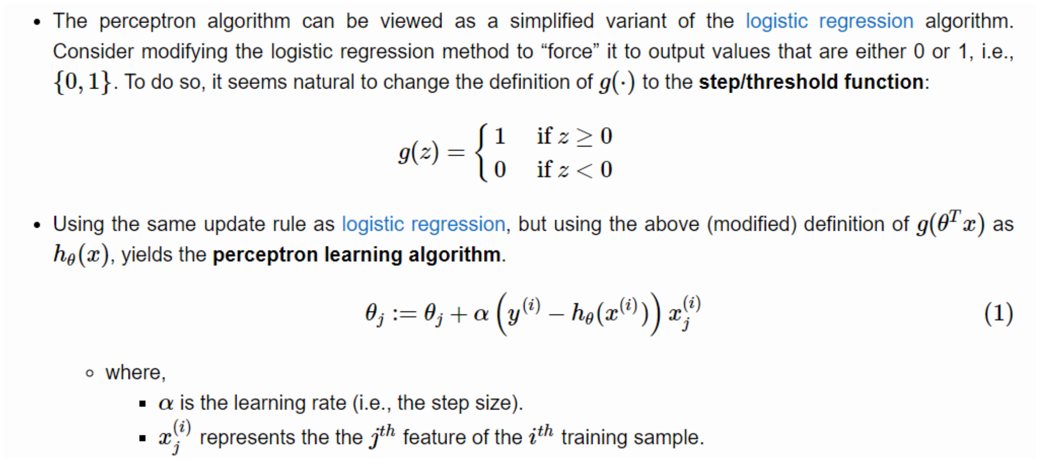
$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial\theta_i \partial\theta_j}$$

- **Challenges**: when the dimension n is large, the computational cost is very high due to the inverse operation.
- **Method selection**:
  a) When n is small (e.g., 10-50), apply Newton’s method
  b) When n is large (e.g., 1000), apply the gradient descendant method.

The Perceptron Algorithm

1 Motivation

- The perceptron algorithm can be viewed as a simplified variant of the logistic regression algorithm.



- The perceptron algorithm can be viewed as a simplified variant of the logistic regression algorithm. Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1, i.e., $\{0,1\}$. To do so, it seems natural to change the definition of $g(\cdot)$ to the **step/threshold function**:

$$g(z)=\begin{cases}1 & \text{if } z\geq 0\\ 0 & \text{if } z<0\end{cases}$$

- Using the same update rule as logistic regression, but using the above (modified) definition of $g(\theta^T x)$ as $h_\theta(x)$, yields the **perceptron learning algorithm**.

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)} \tag{1}$$

  - where,
    - α is the learning rate (i.e., the step size).
    - $x_j^{(i)}$ represents the the j^{th} feature of the i^{th} training sample.

- **Update rule**: the update rules for the perceptron, logistic regression and linear regression are all similar, with the only difference being the definition of $h_\theta(x)$ or $g(\theta^T x)$. The reason behind this common theme is that these algorithms are types of generalized linear models.

Generalized Linear Models

Gaussian Discriminant Analysis

https://aman.ai/cs229/gda/

1 Generative learning algorithm

- **Discriminative learning algorithms:** Algorithms that try to learn **P(y|x)** directly (such as logistic regression), or algorithms that try to learn the mapping from the input space X to the labels (such as the perceptron algorithm) are called **discriminative learning algorithms**.
- **Generative learning algorithms**: algorithms that instead try to model P(x ∣ y) (and P(y))
- **Bayes rules:** given $p(y)$ and $p(x|y)$, we want to learn

$$\begin{aligned} P(y \mid x) &= \frac{P(x \mid y)P(y)}{P(x)} \\ &\approx P(x \mid y)P(y) \end{aligned}$$

  Where we can normalize the joint distribution to find the posterior distribution.

- The above equation represents the underlying framework we'll use to build generative learning algorithms such as Gaussian discriminant analysis and Naive Bayes.
- **Takeaways**:
  - Discriminative learning algorithms learn $P(y \mid x)$, i.e., the distribution of the output given an input. In other words, learn $h_\theta(x) = \{0, 1\}$ directly.
  - Generative learning algorithms learn $P(x \mid y)$, i.e., the distribution of features given a class, and $P(y)$ which is called the class prior. In the tumor-identification setting, where a tumor may be malignant or benign, these models learn the features for each case and use them to make a prediction.

2 Gaussian discriminant analysis

- **Application**: GDA can be used for continuous-valued features, say, tumor classification.
- **Multivariate Gaussian Distribution**:
  - Assume X is distributed as a multivariate Gaussian, i.e, $X \in \mathbb{R}^n$, parameterized by a mean vector $\mu \in \mathbb{R}^n$ and a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$, where $\Sigma \geq 0$ is symmetric and positive semi-definite. Formally, this can be written as,

$$X \sim \mathcal{N}(\mu, \Sigma)$$

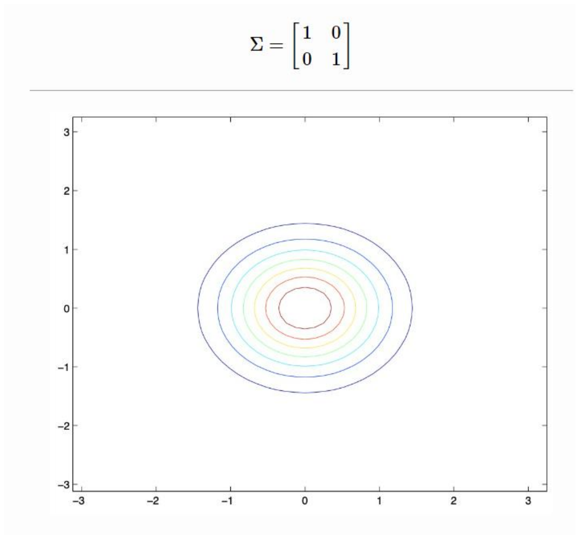
  - The probability density function (PDF) of X is given by:

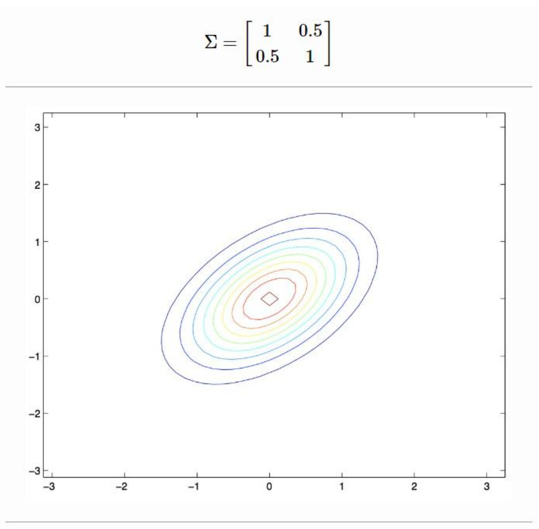
$$P(X; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)\right)$$

    - where "$|\Sigma|$" denotes the determinant of the matrix Σ.
  - The expected value of X is given by μ:

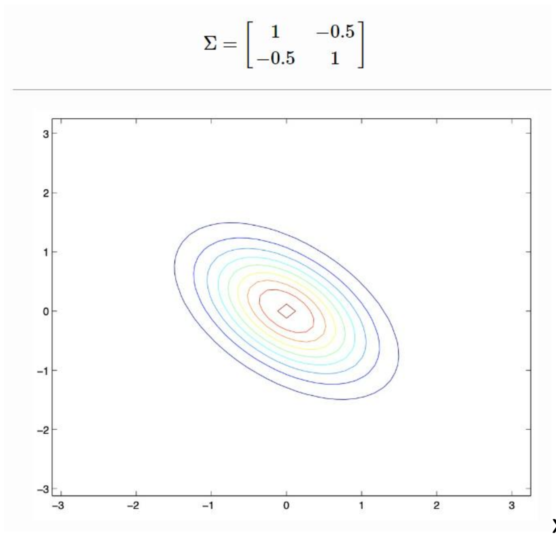
$$\mathrm{E}[X] = \int_x x P(x; \mu, \Sigma) dx = \mu$$

- The covariance matrix Σ:





x1=x2



x1=-x2

- The **Gaussian Discriminant Analysis (GDA) Model**: focus on $p(y)$ and $p(x|y)$
    a) $p(y)$: Bernoulli random variable and Bernoulli distribution
    b) $p(x|y)$: Conditional Gaussian distribution

$$
\begin{aligned}
x \mid y = 0 &\sim \mathcal{N}(\mu_0, \Sigma) \\
x \mid y = 1 &\sim \mathcal{N}(\mu_1, \Sigma) \\
y &\sim \mathrm{Bernoulli}(\phi)
\end{aligned}
$$

Where the parameters of our GDA model are μ_0, μ_1, Σ and ϕ. Note that we used the **same covariance matrix** Σ for both classes, but different mean vectors μ0 and μ1.

    c) By maximizing the log-likelihood function, we have

$$\begin{aligned}\ell(\phi,\mu_0,\mu_1,\Sigma) &= \log\prod_{i=1}^{m} P(x^{(i)},y^{(i)};\phi,\mu_0,\mu_1,\Sigma)\\ &= \log\prod_{i=1}^{m} P(x^{(i)}\mid y^{(i)};\mu_0,\mu_1,\Sigma)P(y^{(i)};\phi)\end{aligned}$$

Then the estimation of 4 parameters μ_0, μ_1, Σ and ϕ are

$$\phi = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)}=1\right\}}{m}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)}=0\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)}=0\right\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)}=1\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)}=1\right\}}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)}-\mu_{y^{(i)}}\right)\left(x^{(i)}-\mu_{y^{(i)}}\right)^T$$

d) Why is the same covariance?

By using two separate means and a single covariance matrix, the decision boundary ends up being linear which is the case for a lot of problems.

- Make predictions using GDA:

$$\underset{y}{\operatorname{argmax}}\, P(y\mid x) = \underset{y}{\operatorname{argmax}} \frac{P(x\mid y)P(y)}{P(x)}$$

$$\doteq \underset{y}{\operatorname{argmax}}\, p(x|y)\, P(y)$$

3 Discussion: GDA and Logistic Regression

- Suppose the optimal parameters μ_0, μ_1, Σ and ϕ are given, the prediction of y is

$$P(x \mid y=1; \phi, \mu_0, \mu_1, \Sigma) = \frac{P(x \mid y=1; \phi, \mu_0, \mu_1, \Sigma) P(y=1; \phi)}{P(x; \phi, \mu_0, \mu_1, \Sigma)} \quad (1)$$

- If we view the quantity $P(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma)$ as a function of x, it can be expressed in the form of sigmoid function.

$$P(y=1 \mid x; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1+\exp(-\theta^T x)}$$

  - where θ is some appropriate function of $\phi, \Sigma, \mu_0, \mu_1$. Note that this uses the convention of redefining the $x^{(i)}$'s on the right-hand-side to be $n+1$ dimensional vectors by adding the extra coordinate $x_0^{(i)} = 1$. This is exactly the form that logistic regression – a discriminative algorithm – used to model $P(y=1 \mid x)$.

- **Takeaway**:
  a) Both logistic regression and Gaussian discriminant analysis (GDA) turn out to use sigmoid function to predict y.
  b) However, logistic regression and GDA may have different parameters θ.

4 How to choose models between Logistic regression and GDA?

- In general, GDA and logistic regression yield different decision boundaries when trained on the same dataset
- **Modeling assumption**: GDA makes stronger modeling assumptions about the data than logistic regression
- **When to choose GDA**: if the data features satisfy Gaussian distribution (correct model, even with small dataset).
- **Logistic regression**: more robust and less sensitive to incorrect modeling assumptions since it makes significantly weaker assumptions.
- **Takeaway**:
  a) Compared to say logistic regression for classification, GDA is simpler and more computationally efficient to implement in some cases. In case of small data sets, GDA might be the better choice than logistic regression.
  b) GDA makes stronger modeling assumptions, and is more data efficient (i.e., requires less training data to learn "well") when the modeling assumptions are correct or at least approximately correct.
  c) On the flip side, Logistic regression makes weaker assumptions, and is significantly more robust to deviations in modeling assumptions.
  d) In today's era of big data, there is a strong trend to use logistic regression which makes fewer assumptions about the data and just lets the algorithm figure out whatever it wants to figure out from the data. Thus, for problems that have a lot of data available, logistic regression should be the algorithm of choice. Because with more data, you could overcome telling the algorithm less about the nature of the data.

Naive Bayes

1 Motivation

- GDA: the feature vectors x were continuous, real-valued vectors.
- **Naïve Bayes**: the feature vectors x is discrete-valued, e.g., for say, a spam vs. no-spam classifier or sentiment classification using tweets

2 Example

- **Training set**: a set of emails labeled as spam or non-spam
- **Parameter:** note that in this subsection, the number of samples is m, the dimension of the features is k.
- **Features**: represent an email via a feature vector whose length is equal to the number of words in the dictionary. Specifically, if an email contains the ith word of the dictionary, then we will set $x_i = 1$; otherwise, we let $x_i = 0$. For instance, the vector is used to represent an email that contains the words "a" and "buy," but not "aardvark," "aardwolf" or "zygmurgy."

$$x = \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{array}\right] \begin{array}{c} \text{a} \\ \text{aardvark} \\ \text{aardwolf} \\ \vdots \\ \text{buy} \\ \vdots \\ \text{zygmurgy} \end{array}$$

- The set of words encoded into the feature vector is called the vocabulary, so the dimension of x is equal to the size of the vocabulary.
- **Label**: $y \in \{0,1\}$ represents either spam or non-spam.

3 Naïve Bayes Model

- Bayes rule: $p(y|x) \propto p(x|y)p(y)$
- $P(x \mid y)$: We will assume that the x_i's are conditionally independent given y.

$$
\begin{aligned}
&P(x_1, \ldots, x_{50000} \mid y) \\
&\quad = P(x_1 \mid y) P(x_2 \mid y, x_1) P(x_3 \mid y, x_1, x_2) \cdots P(x_{50000} \mid y, x_1, \ldots, x_{49999}) \\
&= P(x_1 \mid y) P(x_2 \mid y) P(x_3 \mid y) \cdots P(x_{50000} \mid y) \\
&= \prod_{i=1}^{n} P(x_i \mid y)
\end{aligned}
$$

- The parameters are shown below: $y \sim Bernolli(\phi_y), x_i|y = 0 \sim Bernolli(\phi_{i|y=0})$, and $x_i|y = 1 \sim Bernolli(\phi_{i|y=1})$.
- Then the max-likelihood function is

$$\mathcal{L}\left(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}\right) = \prod_{i=1}^{m} P\left(x^{(i)}, y^{(i)}\right)$$

- The estimation of parameters is

$$
\begin{aligned}
\phi_{j|y=1} &= \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}} \\
\phi_{j|y=0} &= \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}} \\
\phi_y &= \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}{m}
\end{aligned}
$$

- where,
  - the "$\wedge$" symbol means the "and" operation.
  - $1\{\cdot\}$ represents the indicator function which returns 1 if its argument is true, otherwise 0. In other words, the indicator of a true statement is equal to 1 while that of a false statement is equal to 0.
- The parameters have a very natural interpretation. For instance, $\phi_{j|y=1}$ is just the fraction of the spam $(y = 1)$ emails in which word j does appear.

- **Prediction**: the posterior probability is shown as

$$
\begin{aligned}
P(y = 1 \mid x) &= \frac{P(x \mid y = 1) P(y = 1)}{P(x)} \\
&= \frac{\left(\prod_{i=1}^{n} P(x_i \mid y = 1)\right) P(y = 1)}{\left(\prod_{i=1}^{n} P(x_i \mid y = 1)\right) P(y = 1) + \left(\prod_{i=1}^{n} P(x_i \mid y = 0)\right) P(y = 0)}
\end{aligned}
$$

- **Question**: if the ith component is not binary, but $x_i \in \{0, 1, \cdots, n\}$, the continual distribution $p(x_i|y)$ will be replaced by binomial distribution.

4 Laplace Smoothing

- **Motivation**: when some word appears at the same time in a dictionary, the ML estimation is

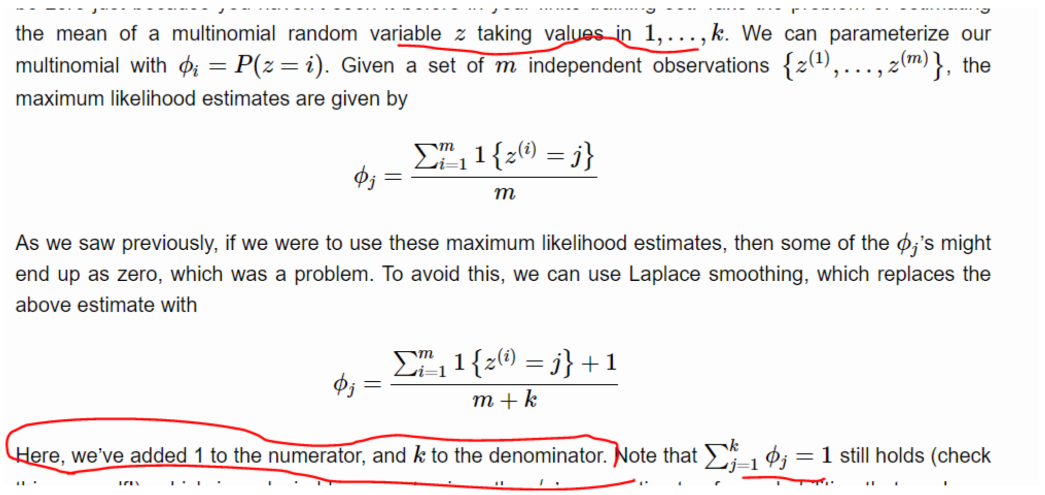
$$\phi_{35000|y=1} = \frac{\sum_{i=1}^{m} 1\left\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}} = 0$$

$$\phi_{35000|y=0} = \frac{\sum_{i=1}^{m} 1\left\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\right\}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}} = 0$$

Which makes the conditional prediction 0.

$$\begin{aligned} P(y=1 \mid x) &= \frac{\prod_{i=1}^{n} P\left(x_i \mid y=1\right) P(y=1)}{\prod_{i=1}^{n} P\left(x_i \mid y=1\right) P(y=1) + \prod_{i=1}^{n} P\left(x_i \mid y=0\right) P(y=0)} \\ &= \frac{0}{0} \end{aligned}$$

- **Laplace smoothing**: suppose k is the alphabet size of the ith component. It is equivalent to add a uniform distribution.



the mean of a multinomial random variable z taking values in $1, \ldots, k$. We can parameterize our multinomial with $\phi_i = P(z=i)$. Given a set of m independent observations $\{z^{(1)}, \ldots, z^{(m)}\}$, the maximum likelihood estimates are given by

$$\phi_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\}}{m}$$

As we saw previously, if we were to use these maximum likelihood estimates, then some of the ϕ_j's might end up as zero, which was a problem. To avoid this, we can use Laplace smoothing, which replaces the above estimate with

$$\phi_j = \frac{\sum_{i=1}^{m} 1\{z^{(i)} = j\} + 1}{m+k}$$

Here, we've added 1 to the numerator, and k to the denominator. Note that $\sum_{j=1}^{k} \phi_j = 1$ still holds (check

- **Example**: suppose $x_i \in \{0,1\}$. We have the following estimation

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\right\} + 1}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\} + 2}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} 1\left\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\right\} + 1}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\} + 2}$$