# Modelling

Madiba Hudson-Quansah

# Contents

# Chapter 1

# Linear Regression

## 1.1 Introduction

**Definition 1.1.1: Linear Regression**

A model that assumes a linear relationship between the input variables ( **X**) and the single output variable (**Y**). This model makes predictions by computing a weighted sum of the input variables, plus a constant bias / intercept term (**b**). Represented as:

$$\mathbf{y} = b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Or

$$\mathbf{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Where:

- $\mathbf{y}$ is the predicted value.
- $n$ is the number of features.
- $x_i$ is the $i$-th feature value.
- $\theta_j$ is the $j$-th model parameter including the bias term $\theta_0$ and feature weights $\theta_1, \theta_2, \ldots, \theta_n$

In vectorized form, the linear regression model can be represented as:

$$\mathbf{y} = h_\theta(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

Where:

- $h_{\boldsymbol{\theta}}(\mathbf{x})$ i the hypothesis function for some parameters $\boldsymbol{\theta}$.
- $\boldsymbol{\theta}$ is the model's parameter vector, containing the bias term $\theta_0$ and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$.
- $\mathbf{x}$ is the input feature vector, containing $x_0$ to $x_n$, with $x_0$ always equal to 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$.

**Note:-**

In machine learning vectors are often represented as column vectors, which is why $\mathbf{x}$ is a column vector, and the dot product $\boldsymbol{\theta} \cdot \mathbf{x}$, simplifies to $\boldsymbol{\theta}^T \mathbf{x}$, where $\boldsymbol{\theta}^T$ is the transpose of $\boldsymbol{\theta}$.

## 1.2 Loss / Cost Function

In training a linear regression model, we need to find the value of $\boldsymbol{\theta}$ such that the model makes the best predictions on the training data. To do this, we need a way to measure how well (or poorly) the model is performing on the training data. We can do this by defining a **loss function** that measures the difference between the predicted value and the actual value. The goal is to minimize this difference.

For linear regression, the most common loss function is the **Mean Squared Error (MSE)**:

$$\text{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left( \boldsymbol{\theta}^T \mathbf{x}^i - \mathbf{y}^i \right)^2$$

Where $m$ is the number of instances in the dataset.

### 1.2.1 The Normal Equation

One way to find the value of $\boldsymbol{\theta}$ that minimizes the cost function is to use the **Normal Equation**. This is a closed-form solution that gives the result directly as is derived from the derivation of the cost function. Given by:

$$\hat{\boldsymbol{\theta}} = \left( \mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

Where:

- $\hat{\boldsymbol{\theta}}$ is the value of $\boldsymbol{\theta}$ that minimizes the cost function.
- $\mathbf{y}$ is the vector of target values containing $\mathbf{y}^{(1)}$ to $\mathbf{y}^{(m)}$.

This method is computationally expensive when the number of features is large, as the complexity of inverting the matrix is $O(n^3)$. However, it is linear with regard to the number of instances in the training set, so it handles large training sets efficiently. Another less computationally expensive method is the **Gradient Descent** algorithm.

### 1.2.2 Derivation of the Normal Equation

The loss function:

$$\text{MSE}(X, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left( \mathbf{x}^i \boldsymbol{\theta} - y^i \right)^2$$

Where

- $\mathbf{x}^i$ is a vector of the $i$-th instance's feature values from the input matrix $X$, where each instance is represented as row of the matrix. I.e. $\mathbf{x}^i$ is of the form:

  $$\mathbf{x}^i = \begin{bmatrix} 1 \\ x_1^i \\ x_2^i \\ \vdots \\ x_n^i \end{bmatrix}$$

  Where $n$ is the number of features.

- $\boldsymbol{\theta}$ is a vector of the model's parameters / weights for each input feature including the bias / intercept $\theta_0$. I.e. $\boldsymbol{\theta}$ is of the form:

  $$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

  Where $n$ is the number of features. This results in the multiplication $\mathbf{x}^i \theta$ being equivalent to the dot product $\mathbf{x}^i \cdot \theta$ and therefore a scalar value.

- $y^i$ is the actual target value of the $i$-th instance and is a scalar value.
- $m$ is the number of instances in the dataset.

Can be expressed in matrix-vector form as:

$$\text{MSE}(X, h_{\boldsymbol{\theta}}) = \frac{1}{m} \left( X\boldsymbol{\theta} - \mathbf{y} \right)^T \left( X\boldsymbol{\theta} - \mathbf{y} \right)$$

Where:

- $X$ is a input matrix containing all the input features of all instances in the dataset. $X$ is of the form:

$$X = \begin{bmatrix} \left(\mathbf{x}^1\right)^T \\ \left(\mathbf{x}^2\right)^T \\ \vdots \\ \left(\mathbf{x}^m\right)^T \end{bmatrix}$$

  With each vector $\mathbf{x}^i$ being an instance's input features previously defined as column vectors, hence the transposition.

- $\mathbf{y}$ is a vector of the target values of all instances in the dataset. $\mathbf{y}$ is of the form:

$$\mathbf{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

- $\boldsymbol{\theta}$ is still the model's parameter vector as previously defined.
- $m$ is still the number of instances in the dataset.

This leads us to finding the partial derivative of this expression with respect to $\boldsymbol{\theta}$ and minimizing it, equating it to zero, given below where $L(\boldsymbol{\theta})$ is the loss function:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{m}\left(X\boldsymbol{\theta} - \mathbf{y}\right)^T\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ L(\boldsymbol{\theta}) &= \frac{1}{m}\left(\left(X\boldsymbol{\theta}\right)^T - \mathbf{y}^T\right)\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T X^T - \mathbf{y}^T\right)\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T X^T X\boldsymbol{\theta} - \boldsymbol{\theta}^T X^T\mathbf{y} - \mathbf{y}^T X\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \\ &\text{Let } \mathbf{a} = X^T\mathbf{y} \text{ and } S = X^T X \text{ then} \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T S\boldsymbol{\theta} - \boldsymbol{\theta}^T\mathbf{a} - \mathbf{a}\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \end{aligned}$$

We then take the partial derivative of the resulting expression with respect to $\boldsymbol{\theta}$:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\boldsymbol{\theta}^T S\boldsymbol{\theta} - \boldsymbol{\theta}^T\mathbf{a} - \mathbf{a}\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \\ &= \frac{1}{m}\left(2S\boldsymbol{\theta} - \mathbf{a} - \mathbf{a} + 0\right) \\ &= \frac{1}{m}\left(2S\boldsymbol{\theta} - 2\mathbf{a}\right) \end{aligned}$$

Then equate the partial derivative to zero:

$$\begin{aligned} \nabla_{\theta} L(\boldsymbol{\theta}) &= \frac{1}{m}\left(2S\boldsymbol{\theta} - 2\mathbf{a}\right) \\ 0 &= 2S\boldsymbol{\theta} - 2\mathbf{a} \\ 0 &= S\boldsymbol{\theta} - \mathbf{a} \\ S\boldsymbol{\theta} &= \mathbf{a} \\ \hat{\boldsymbol{\theta}} &= S^{-1}\mathbf{a} \\ \hat{\boldsymbol{\theta}} &= \left(X^T X\right)^{-1} X^T\mathbf{y} \end{aligned}$$

## 1.3 Gradient Descent

**Definition 1.3.1: Gradient Descent**

A generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

### 1.3.1 Batch Gradient Descent

To implement Gradient Descent, we need to compute the gradient of the cost function with regard to each model parameter $\theta_j$ or parameter vector $\boldsymbol{\theta}$. In other words, we need to calculate how much the cost function will change if we change $\theta_j$ just a little bit. This is called a **partial derivative**. This is given by:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Where:

- $\alpha$ is the learning rate.
- $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ is the gradient of the cost function with respect to the current $\boldsymbol{\theta}$ and is equivalent to the partial derivative of the cost function with respect to $\theta_j$.

The partial derivative of the cost function with respect to $\theta_j$ is found the same way as the normal equation but we do not equate it to zero. Given by:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{2}{m} \left( X^T X \boldsymbol{\theta} - X^T \mathbf{y} \right)$$

### 1.3.2 Stochastic Gradient Descent

**Definition 1.3.2: Stochastic**

Randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

Batch gradient descent requires the whole training set to compute the gradients at each step, which makes it slow when the dataset is large. The Stochastic Gradient Descent algorithm is a variation of the gradient descent algorithm that updates the weights based on a randomly chosen training instance. So instead of the equation above, we have:

$$\boldsymbol{\theta} \leftarrow \theta_i - \alpha \nabla_{\theta_i} L(\theta_i)$$

This makes the algorithm much faster because it has little data to manipulate at each step. However, due to its stochastic nature, it is less stable than batch gradient descent often converging to a value close to the minimum but not the minimum itself.

# Chapter 2

# Locally Weighted Regression (LOWESS)

## 2.1 Introduction

**Definition 2.1.1: Locally Weighted Regression**

A non-parametric regression algorithm that makes predictions by fitting several local linear models to a dataset. It generally follows the same representation as linear regression, but the weight of each instance is determined by a kernel / weight function that gives more weight to instances closer to the instance being predicted, usually using the Gaussian kernel, given by:

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

Instead of fitting a single model to the entire dataset, LOWESS fits a model to a subset of the data, where the size of the subset is determined by the bandwidth parameter $\tau$. The weight function $w$ gives more weight to instances closer to a specific point of interest usually the point being predicted, with the weight decreasing as the distance from the point of interest increases.

The bandwidth parameters $\tau$, determines how many points are considered in the local model, with smaller values making the model more sensitive to local fluctuations in the data, while larger values make the model produce a smoother fit.

## 2.2 Kernel Functions

**Definition 2.2.1: Kernel Function**

Determines how weights are assigned to the neighbouring points.

### 2.2.1 Gaussian Kernel

Provides weights based on the normal distribution, giving more weight to points closer to the point of interest. Given by:

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

Where:

- $w$ is the weight assigned to the $i$-th instance.
- $x^i$ is the $i$-th instance's feature value.
- $x$ is the feature value of the point of interest.
- $\tau$ is the bandwidth parameter which determines the size of the subset of the data that the model fits to.

### 2.2.2 Tri-cube Kernel

Provides weights based on the tri-cube function, giving more weight to points closer to the point of interest. Given by:

$$w = \left(1 - \left(\frac{\left|x^{(i)} - x\right|}{\tau}\right)^3\right)^3$$

Where:

- $w$ is the weight assigned to the $i$-th instance.
- $x^i$ is the $i$-th instance's feature value.
- $x$ is the feature value of the point of interest.
- $\tau$ is the bandwidth parameter which determines the size of the subset of the data that the model fits to.

## 2.3 Parametric vs Non-parametric Models

**Definition 2.3.1: Parametric Model**

This model assumes a specific form for the underlying data distribution and have a fixed number of parameters, with these parameters being learned from the training data.

**Definition 2.3.2: Non-parametric Model**

This model does not assume a specific form for the underlying data distribution and can adapt their shape based on the data, with the number of parameters increasing with the size of the training data.

# Chapter 3

# Logistic Regression

## 3.1 Introduction

**Definition 3.1.1: Logistic Regression**

A classification algorithm based on regression that estimates the probability that an instance belongs to a particular class. Given by:

$$\hat{p} = h_{\boldsymbol{\theta}}(\boldsymbol{\theta} \cdot \mathbf{x}) = \sigma\left(\boldsymbol{\theta}^T \times \mathbf{x}\right)$$

Where $\sigma$ is the **sigmoid function** defined by:

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$

This function outputs a value between 0 and 1, which can be interpreted as a probability.

Once a Logistic regression model has estimated the probability $\hat{p}$ that an instance belongs to the positive class, it can make its prediction $\hat{y}$, based on a specified threshold value, usually 0.5. Given by:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geqslant 0.5 \end{cases}$$

## 3.2 Training and Loss Function

In training a Logistic regression model, we need to find a parameter vector $\boldsymbol{\theta}$ such that the model estimates high probabilities for positive instances $y = 1$ and low probabilities for negative instances $y = 0$. The loss function for a single instance can then be given by:

$$\ell(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0 \end{cases}$$

This loss function works as the value of $-\log t$ grows large when $t$ approaches 0, i.e. the model will be penalized if it estimates a probability close to 0 for a positive instance, and similarly for a probability close to 1 for a negative instance. On the other hand $-\log t$ is close to 0 when $t$ is close to 1, so the loss will be small if the model estimates a probability close to 0 for a negative instance or close to 1 for a positive instance.

Applying this loss function over the entire training set, we want the average loss over all the training instances so we must find the sum of all the losses and divide by the number of instances. Given by:

$$\ell(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \log\left(\hat{p}^i\right) + \left(1 - y^i\right)\log\left(1 - \hat{p}^i\right)\right]$$

This in turn was derived from the log likelihood function, which is the product of the probabilities of the instances in the training set being classified correctly, the use of product due to the AND operation of the probabilities. Given by:

$$\log \ell(\boldsymbol{\theta}) = \log \prod_{n=1}^{m} \left( \frac{1}{\sqrt{2\pi\boldsymbol{\theta}}} \right) \exp\left( -\frac{\left(y^i - x^i\boldsymbol{\theta}\right)^2}{2\theta^2} \right)$$

This was simplified to a summation due to $\log(a \times b) = \log a + \log b$ giving us:

$$\log \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left( \log \frac{1}{\sqrt{2\pi\boldsymbol{\theta}}} + \log \exp\left( \frac{\left(y^i + x^i\boldsymbol{\theta}\right)}{2\boldsymbol{\theta}^2} \right) \right)$$

This was further simplified using the properties of the logarithm function to give us:

$$\log \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left( -\frac{1}{2} \log 2\pi\boldsymbol{\theta} - \frac{\left(y^i + x^i\boldsymbol{\theta}\right)^2}{2\boldsymbol{\theta}^2} \right)$$

Using Bernoulli's distribution and applying the sigmoid function to the linear regression model, we can posit that the hypothesis function's output is the probability that the instance belongs to the positive class and now exists in the range $[0, 1]$. Given this we can simplify the likelihood of a positive instance and the inverse probability of a negative instance using the output of the hypothesis function, giving us:

$$\log \ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left( y^i \log\left(\hat{p}^i\right) + \left(1 - y^i\right) \log\left(1 - \hat{p}^i\right) \right)$$

By using the log loss we are implicitly making the assumption that the instances follow a Gaussian distribution around the mean of their respective classes. This is a common assumption in logistic regression and is the reason why the log loss is used as the loss function.

There exists no closed-form equation to compute the value of $\boldsymbol{\theta}$ that minimizes the cost function, but due to the convex nature of the cost function, we are guaranteed to find the global minimum using the Gradient Descent or any other optimization algorithm. The partial derivative of the cost function with respect to $\theta_j$ is given by:

$$\nabla_{\theta_j} \ell(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left( y^i - \sigma\left(\boldsymbol{\theta}^T \mathbf{x}^i\right) \right) x_j^i$$

Giving the update rule for the Gradient Descent algorithm as:

$$\theta_j := \theta_j + \alpha \left( y^i - \sigma\left(\boldsymbol{\theta}^T x^i\right) \right) x_j^i$$

## 3.3 Newton's Method for maximizing $\ell(\theta)$

Newton's method for optimization is a second-order optimization algorithm that provides an alternative to the Gradient Descent Algorithm in optimizing the cost function.

Given a function $f : \mathbb{R} \to \mathbb{R}$ to find the value of $x$ such that $f(x) = 0$, where $x \in \mathbb{R}$ is a real number. Using newton's method we can approximate the value of $x$ by iteratively updating it using the following update rule:

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

Where $f'$ is the derivative of $f$. This can be thought of intuitively as approximating the function $f$ via a linear function that is tangent to the $f$ at the current guess $x$ and then finding the point at which the tangent function is equal to zero, i.e. intersects the $x$-axis.

In optimizing $\ell(\theta)$, we use the same approach with $f(x) = L'(\theta)$ to obtain the update rule:

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

In the case of logistic regression $\theta$ is a vector $\boldsymbol{\theta}$, do Newton's method must be generalized to a multi-dimensional space, also called the Newton-Raphson method, given by:

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

Where:

- $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$ is the vector of partial derivatives of the loss function with respect to the $\theta_i$
- $H$ is a $n \times n$ matrix called the Hessian matrix, whose elements are given by:

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

Newton's method typically converges faster than the Gradient Descent Algorithm, but is more expensive due to computing and inverting the Hessian matrix $H$, but if $n$ is not too large it is computationally feasible. When Newton's method is used to optimize the log likelihood function, $\ell(\boldsymbol{\theta})$, of logistic regression, it is called **Fisher Scoring**.

# Chapter 4

# Perceptron

**Definition 4.0.1: Perceptron**

A single layer neural network that classifies instances by computing a weighted sum of the input features and then applying a step function to the result. Given by:

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

Where:

- $g(z)$ is the step function
- $z$ is the hypothesis function given by:

$$z = \boldsymbol{\theta}^T \mathbf{x}$$

This model can be though of a predecessor to the logistic regression model, with the step function being a more discrete version of the sigmoid function. Thus the update rule for the Perceptron model is given by:

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

This is also called the perceptron learning model

# Chapter 5

# Exponential Family of Distributions

## 5.1 Introduction

**Definition 5.1.1: Exponential Family of Distributions**

A family of probability distributions that can be represented in the form:

$$\mathbb{P}\left(\mathbf{y};\eta\right)=b\left(\mathbf{y}\right)\exp\left(\eta^{T}T\left(\mathbf{y}\right)-a\left(\eta\right)\right)$$

Or

$$\mathbb{P}\left(\mathbf{y};\eta\right)=\frac{b\left(\mathbf{y}\right)\exp\left(\eta^{T}T\left(\mathbf{y}\right)\right)}{e^{a(\eta)}}$$

Where:

- $\mathbf{y}$ is the random variable / Data
- $\eta$ is the natural parameter.
- $T\left(\mathbf{y}\right)$ is the sufficient statistic.
- $a\left(\eta\right)$ is the log partition function.
- $b\left(\mathbf{y}\right)$ is the base measure.

### 5.1.1 The Sufficient Statistic

**Definition 5.1.2: The Sufficient Statistic**

A statistic that retains all the information in the data about the parameter of interest.

## 5.2 Bernoulli Distribution

**Definition 5.2.1: Bernoulli Distribution**

Defined as:

$$\text{Bern}\left(X|\mu\right)=\mu^{x}\left(1-\mu\right)^{1-x}$$

Where:

- $X$ is the random variable, taking values in $\{0,1\}$.
- $\mu$ is the probability of success.

$$\begin{aligned}\exp\log\left(\mu^X\left(1-\mu\right)^{1-X}\right) &= \exp\left(X\log\mu+(1-X)\log\left(1-\mu\right)\right)\\ &= \exp\left(y\log\mu-y\log\left(1-\mu\right)+\log\left(1-\mu\right)\right)\\ &= \exp\left(y\log\left(\frac{\mu}{1-\mu}\right)+\log\left(1-\mu\right)\right)\end{aligned}$$

$\therefore$:

$b\left(y\right) = 1$

$\eta = \log\left(\frac{\mu}{1-\mu}\right)$

$T\left(\mathbf{y}\right) = \mathrm{y}$

$a\left(\eta\right) = \log\left(1-\mu\right)$

## 5.3 Normal / Gaussian Distribution

**Definition 5.3.1: Gaussian Distribution**

Defined as:

$$\mathrm{Norm}\left(X|\mu,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{\left(X-\mu\right)^2}{\sigma^2}\right)$$

Where:

- $X$ is the random variable.
- $\mu$ is the mean of the distribution.
- $\sigma$ is the variance of the distribution.

$$\begin{aligned}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{\left(X-\mu\right)^2}{\sigma^2}\right) &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\right)\right)\\ &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\\ &= \log\left(1\left(2\pi\sigma^2\right)^{-\frac{1}{2}}\right)-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\\ &= -\frac{1}{2}\log\left(2\pi\sigma^2\right)-\frac{1}{2}\frac{\left(X-\mu\right)^2}{\sigma^2}\\ &= -\frac{1}{2}\log\left(2\pi\sigma^2\right)-\frac{1}{2}\frac{X^2-2X\mu+\mu^2}{\sigma^2}\\ &= \exp-\frac{1}{2}\log\left(2\pi\sigma^2\right)-\frac{X^2-2X\mu+\mu^2}{2\sigma^2}\\ &= \exp-\frac{1}{2}\log\left(2\pi\sigma^2\right)-\left(\frac{X^2}{2\sigma^2}-\frac{2X\mu}{2\sigma^2}+\frac{\mu^2}{2\sigma^2}\right)\\ &= \frac{1}{\sqrt{2\pi\sigma^2}}-\exp\left(\frac{X^2}{2\sigma^2}-\frac{2X\mu}{2\sigma^2}+\frac{\mu^2}{2\sigma^2}\right)\end{aligned}$$

Where:

- $\eta = \mu$

- $T\left(y\right) = y$
- $a\left(\eta\right) = \frac{\mu^2}{2}$
- $b\left(y\right) = \left(\frac{1}{\sqrt{2\pi}}\right) \exp\left(\frac{-y^2}{2}\right)$

# Chapter 6

# Generalized Linear Models (GLMs)

## 6.1 Introduction

**Definition 6.1.1: Generalized Linear Model**

A generalization of linear regression that allows the dependent variable to have an error distribution other than the normal distribution.

Given a random variable $y$ we can create a GLM to predict it's value as a function of $x$. To do this we must make the following assumptions:

- $y \mid x; \theta \sim$ ExponentialFamily $(\eta)$. I.e. given $x$ and $\theta$, the distribution of $y$ follows some exponential family distribution, with parameter $\eta$.
- Given $x$, our goal is to predict the expected value of $T(y)$ given $x$. Usually $T(y) = y$, meaning we want the prediction output $h(x)$ output by our learned hypothesis $h$ to satisfy:

  $$h(x) = E[y \mid x]$$

  For example for logistic regression we had $h_{\boldsymbol{\theta}}(x)$ as

  $$h_{\boldsymbol{\theta}}(x) = \mathbb{P}(y = 1 \mid x; \theta) = 0 \times \mathbb{P}(y = 0 \mid x; \theta) + 1 \times \mathbb{P}(y = 1 \mid x; \theta) = E[y \mid x; \theta]$$

- The natural parameter $\eta$ and the input features $x$ are linearly related:

  $$\eta = x\boldsymbol{\theta}$$

  Or if $\boldsymbol{\eta}$:

  $$\eta_i = \theta_i^T x$$

# Chapter 7

# Softmax Regression / Multinomial Logistic Regression

**Definition 7.0.1: Softmax Regression / Multinomial Logistic Regression**

A generalization of Logistic Regression to support multiple classes. Where the score for a class $k$ is given by:

$$s_k(\mathbf{x}) = \left(\boldsymbol{\theta}^k\right)^T \mathbf{x}$$

Where:

- $\mathbf{x}$ is an instance's feature vector.
- $\boldsymbol{\theta}^k$ is the parameter vector for the class $k$, where each class has its own parameter vector.

In matrix form:

$$s_k(X) = X\boldsymbol{\Theta}$$

Where:

- $X$ is the matrix containing all instances' feature vectors.
- $\boldsymbol{\Theta}$ is the parameter matrix with each row containing the parameter vector for each class. In the form:

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\theta}^1 \\ \boldsymbol{\theta}^2 \\ \vdots \\ \boldsymbol{\theta}^k \end{bmatrix}$$

Where $k$ is the number of classes.

We then use the **Softmax function** to estimate the probability that an instance belongs to a particular class, given by:

$$\hat{p}_k = \sigma(s(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{i=1}^{k} \exp(s_i(\mathbf{x}))}$$

$$h_\theta(x) = \exp\left(\theta^T x^i\right)$$

Where:

- $x^i \in \mathbb{R}^n$
- $h_\theta(x) = [0, 1]$

$$\mathbb{P}\left(y = k \mid X; \theta\right) = \frac{\exp\left(\theta^T x\right)}{\sum_{i=1}^{k} \exp\left(\theta_j^T X\right)}$$

# Chapter 8

# Gaussian Discriminant Analysis

## 8.1 Introduction

**Definition 8.1.1: Discriminative Algorithm**

An algorithm that models the decision boundary between the classes directly from labelled examples. I.e. Determining the label based on the features. This class of algorithm learns mappings from inputs to labels.

**Definition 8.1.2: Generative Algorithm**

An algorithm that models the distribution of each class and then uses Bayes' theorem to estimate the probability that an instance belongs to a particular class.

**Definition 8.1.3: Gaussian Discriminant Analysis**

A generative learning algorithm that assumes that the data is generated by a Gaussian distribution. It models the distribution of each class as a Gaussian distribution and uses Bayes' theorem to estimate the probability that an instance belongs to a particular class. It is used for continuous valued inputs.

In using Gaussian Distribution Analysis we make the following assumptions:

- The data is modelled in a Multivariate Gaussian distribution.

$$\mathbb{P}(x \mid t = k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right]$$

  Where:

  - $k$ is the class.
  - $\Sigma$ is the covariance matrix of the distribution and thus $|\Sigma|$ is the determinant of the covariance matrix. And each class $k$ has its own covariance matrix $\Sigma_k$.
  - $D$ is the number of features.
  - $\mu_k$ is mean vector of the distribution for each class, where $\mu_k$ has $D$ parameters for $DK$ total

The Bayes theorem is then used to estimate the probability that an instance belongs to a particular class, given by:

$$\mathbb{P}(y = 1 \mid x) = \frac{\mathbb{P}(x \mid y = 1)\,\mathbb{P}(y = 1)}{\mathbb{P}(x)}$$

Where:

- $\mathbb{P}(x)$ is the probability that an instance is generated by the Gaussian distribution:

$$\mathbb{P}(x) = \mathbb{P}(x \mid y = 1)\,\mathbb{P}(y = 1) + \mathbb{P}(x \mid y = 0)\,\mathbb{P}(y = 0)$$

- $\mathbb{P}(x \mid y = 1)$ is the probability that an instance belongs to class 1 given the features, i.e.:

$$\mathbb{P}(x \mid y = k) = \frac{1}{(2\pi)^{D/2}\,|\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)\right]$$

- $\mathbb{P}(y = 1)$ is the probability of the instance belonging to class 1, which as it can only be 1 or 0, is Bernoulli distributed:

$$\mathbb{P}(y) = \phi^y (1 - \phi)^{1-y}$$

Given this form we can optimize the parameters $\phi$, $\boldsymbol{\mu}_k$, and $\boldsymbol{\Sigma}_k$ to maximize the likelihood of an instance $\phi$ belonging to a particular class $k$. The close forms for these parameters are given by:

$$\phi = \frac{1}{N}\sum_{i=1}^{N} r_1^i$$

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{N} r_k^i \cdot \mathbf{x}^i}{\sum_{i=1}^{N} r_k^i}$$

$$\boldsymbol{\Sigma_k} = \frac{1}{\sum_{i=1}^{N} r_k^i}\sum_{i=1}^{N} r_k^i (\mathbf{x}^i - \boldsymbol{\mu}_k)(\mathbf{x}^i - \boldsymbol{\mu}_k)^T$$

$$r_k^i = \mathbb{I}\left[y^i = k\right]$$

The decision boundary between classes is given by:

$$\log \mathbb{P}(y \mid \mathbf{x}) = \mathbf{x}^T \sum_{\ell}^{-1} \mathbf{x} - 2\boldsymbol{\mu}_\ell^T \sum_{\ell}^{-1} \mathbf{x} + c$$

Where $c$ is a constant that depends on the class priors and the parameters of the Gaussian distributions. With the shape of the decision boundary being quadratic, i.e. a quadratic discriminant function, making it a conic section.

## 8.2 Gaussian Discriminant Analysis vs. Logistic Regression

- GDA makes stronger assumptions about the data than Logistic Regression, i.e. that the data is generated by a Gaussian Distribution. This can be an advantage if the data is actually generated by a Gaussian distribution, but if not the quality of predictions can be worse than Logistic Regression.
- Many class-conditional distributions are more efficiently modelled by a Logistic Classifier, as when these distributions are non-Gaussian, which is usually the case, Logistic Regression usually beats GDA.
- GDA can handle missing features.

# Chapter 9

# Naïve Bayes

**Definition 9.0.1: Naïve Bayes**

A generative learning algorithm that assumes that the features are conditionally independent given the class, this is why it is naïve . It models the distribution of each class and uses Bayes' theorem to estimate the probability that an instance belongs to a particular class.

## 9.1 Introduction

Where the $\mathbf{x}$ vectors used in GDA were continuous real-valued vectors, Naïve Bayes assumes that the features are discrete. Starting with the task of spam email classification based on the text of the email, where certain words being present increase the likelihood of the email being spam, we need to represent each email as a discrete vector containing important information about the email. This can be done by specifying a dictionary of words that are important, i.e. indicate an email as spam, and then representing each email as a binary vector where the $i$a-th element is 1 if the $i$-th word in the dictionary is present in the email and 0 otherwise, i.e.:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} \text{Viagra} \\ \text{Nigeria} \\ \vdots \\ \text{Money} \\ \vdots \\ \text{Bank} \end{bmatrix}$$

This is known as the **Bag of Words** / **Vocabulary**, where $n$ is the dimension of $x$ and is equal to the number of words in our vocabulary. Having our input vector we can build a discriminative model, i.e. $\mathbb{P}\left(x \mid y\right)$. But if we have a vocabulary of $n$ words then $x \in \{0,1\}^n$, i.e. $x$ is an $n$ dimensional binary vector, and if we were to model $x$ as a Multinomial distribution, we would end up with a $2^n - 1$ dimensional parameter vector, i.e. too many parameters. Naïve Bayes avoids this in its naïve assumption that all the $x_i$'s are conditionally independent given $y$. This allows us to model the probability of an instance belonging to a particular class using the following form:

$$\begin{aligned} \mathbb{P}\left(x_1, \ldots, x_n \mid y\right) &= \mathbb{P}\left(x_1 \mid y\right) \times \mathbb{P}\left(x_2 \mid y\right) \times \ldots \times \mathbb{P}\left(x_n \mid y\right) \\ &= \prod_{i=1}^{n} \mathbb{P}\left(x_i \mid y\right) \end{aligned}$$

The model is parametrized by:

$$\begin{aligned}\phi_{i|y=1} &= \mathbb{P}\left(x_i = 1 \mid y = 1\right)\\ \phi_{i|y=0} &= \mathbb{P}\left(x_i = 1 \mid y = 0\right)\\ \phi_y &= \mathbb{P}\left(y = 1\right)\end{aligned}$$

Where:

- $\phi_{i|y=1}$ is the probability that the feature $x_i$ is 1, given that the class is 1.
- $\phi_{i|y=0}$ is the probability that the feature $x_i$ is 1, given that the class is 0.
- $\phi_y$ is the probability that any instance belongs to class 1 / the prior probability of class 1.

Therefore given a training set $\{\left(x^i, y^i\right); i = 1, \ldots, m\}$, the joint likelihood of the data is:

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

Where maximizing this likelihood with respect to the parameters $\phi_y, \phi_{i|y=0}, \phi_{i|y=1}$ gives maximum likelihood estimates:

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

Where:

- $\phi_{j|y=1}$ is the fraction of instances in class 1 that contain the word $j$.
- $\phi_{j|y=0}$ is the faction fraction of instances in class 0 that contain the word $j$.
- $\phi_y$ is the fraction of instances that belong to class 1.

Having fit all these parameters we can then make a prediction on a new instance $x$, like so:

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

And pick the class with the highest probability.

In the case where the number of classes is $k$ we would model $\mathbb{P}\left(x \mid y\right)$ as multinomial instead of Bernoulli, and the parameters would be given by:

$$\phi_{j|y=k} = \frac{\sum_{i=1}^{m} 1\{x_j^i = 1 \wedge y^i = k\}}{\sum_{i=1}^{m} 1\{y^i = k\}}$$

$$\phi_y = \frac{\sum_{i=1}^{m} 1\{y^i = k\}}{m}$$

## 9.2 Laplace Smoothing

**Definition 9.2.1: Laplace Smoothing / Additive Smoothing**

A technique used to smooth categorical data. It is used to solve the problem of zero probability, i.e. when a word appears in the test set but not in the training set. It is given by:

$$\phi_{j|y=k} = \frac{\sum_{i=1}^{m} 1\{x_j^i = 1 \wedge y^i = k\} + \alpha}{\sum_{i=1}^{m} 1\{y^i = k\} + \alpha \cdot n}$$

Where:

- $\alpha$ is the smoothing parameter.
- $n$ is the number of features.

Laplace Smoothing solves a common problem in Naïve Bayes where a word appears in the test set but not in the training set, which would result in a zero probability. Laplace Smoothing solves this by adding a small value to the numerator and denominator of the probability calculation. This results in a very small probability for the word instead of a zero probability. This is important as with the naive assumption of independence, simplifying the probability calculation to a series of multiplications, a zero probability would result in the entire probability being zero, i.e.:

$$\mathbb{P}\left(x \mid y\right) = \prod_{i=1}^{n} \mathbb{P}\left(x_i \mid y\right)$$

If for example $x_2$ was a word non-existent in our vocabulary, then $\mathbb{P}\left(x_2 \mid y\right) = 0$, making $\mathbb{P}\left(x \mid y\right) = 0$

# Chapter 10

# Support Vector Machines (SVMs)