Machine Learning and Photonics

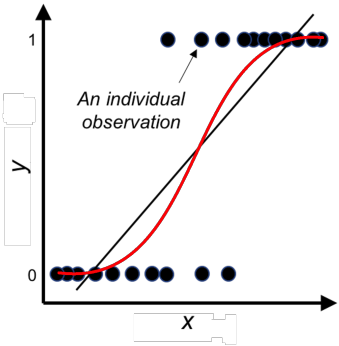
Week 6

Ergun Simsek

University of Maryland Baltimore County
simsek@umbc.edu

Logistic Regression

March 6, 2023



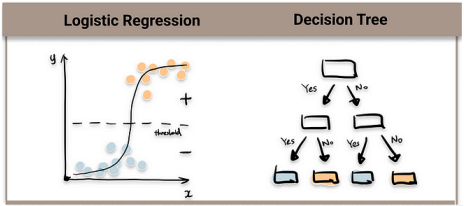
What's Classification?

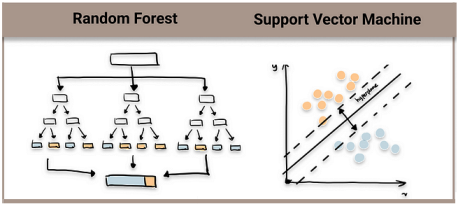
- In ML, classification is a type of supervised learning algorithm where the goal is to assign a label or class to a set of input data based on its features or attributes.
- Our task is to build a model that will learn from labeled examples in order to predict the class labels of new, unseen data.

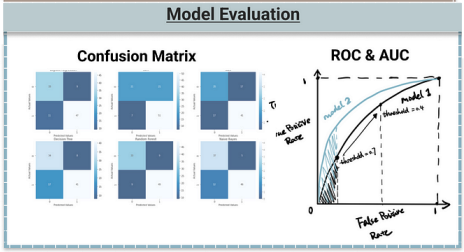
What's Classification?

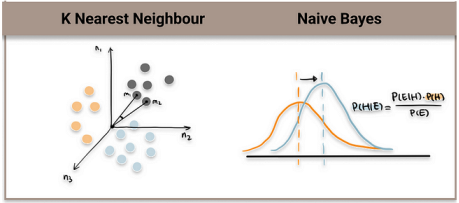
- In ML, classification is a type of supervised learning algorithm where the goal is to assign a label or class to a set of input data based on its features or attributes.
- Our task is to build a model that will learn from labeled examples in order to predict the class labels of new, unseen data.
- Typical examples include
  - Spam e-mail detection,
  - Image classification,
  - Sentiment analysis, and
  - Fraud detection.

Classification Methods and Model Evaluation





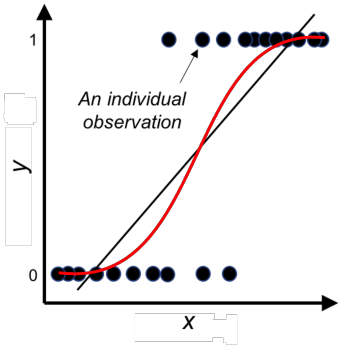




Binary Classification

Binary Classification methods, such as binary logistic regression, are used to predict the probability of a binary outcome.

- A good chip or a malfunctioning one
- An e-mail is spam or not
- A comment is positive or negative,
- A patient has cancer or is cancer-free,
- A transaction is fraud or legit.



Multinomial Classification

When the number of possible outcomes for a categorical target variable is greater than two, we refer to the situation as a *multinomial classification*.

Some examples:

- Target variable: color (possible outcomes: Blue, Red, Green)
- Target variable: car types (possible outcomes: SUV, Sedan, Hatchback, Convertible, Coupe)
- Target variable: age category (possible outcomes: Children, Youth, Adults, Seniors)
- Target variable: house type in US (possible outcomes: Single Family, Condominium, Townhouse, Multi-Family)

Binary Logistic Regression

- Assume that we have *d continuous features* collectively represented by $\vec{x} = (x_1, x_2, \cdots, x_d)$.
- The target variable y is a categorical variable with exactly two possible outcomes.

Binary Logistic Regression

- Assume that we have *d continuous features* collectively represented by $\vec{x} = (x_1, x_2, \cdots, x_d)$.
- The target variable y is a categorical variable with exactly two possible outcomes.
- It is convenient to label the two outcomes as **Failure** and **Success** associated with $y = 0$ and $y = 1$, respectively.

$$\begin{cases} y = 0 : & \text{Failure} \\ \\ y = 1 : & \text{Success} \end{cases} \tag{1}$$

- The above labeling is merely a choice, and you can change the labeling without affecting the predictions of the model.
- Suppose we have recorded the values of features and the target for n observations.

Binary Logistic Regression *(Cont...)*

- Since the target variable is categorical, we can't assume a linear relation.
- Logistic regression takes a **probabilistic approach** to cure this problem.

Binary Logistic Regression *(Cont...)*

- Since the target variable is categorical, we can't assume a linear relation.
- Logistic regression takes a **probabilistic approach** to cure this problem.
- Although y is a categorical variable, *the probability for y to assume a certain class* is continuous!

$P(y^{(i)}|\vec{x}^{(i)})$: probability that the target variable takes value $y^{(i)}$ for the i-th observation provided that the features are $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_d^{(i)})$. (3)

Binary Logistic Regression *(Cont...)*

- Since the target variable is categorical, we can't assume a linear relation.
- Logistic regression takes a **probabilistic approach** to cure this problem.
- Although y is a categorical variable, *the probability for y to assume a certain class* is continuous!

$P(y^{(i)}|\vec{x}^{(i)})$: probability that the target variable takes value $y^{(i)}$ for the i-th observation provided that the features are $\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots, x_d^{(i)})$. (3)

- Here, probability in (3) is an instance of *conditional probability*. It only concerns the i-th observation.

Binary Logistic Regression *(Cont...)*

Assume that all n observations of the dataset are **independent** of one another.

Recall from statistics that if A and B are independent events, then the probability for the event A and B is given by $P(A \cap B) = P(A) \cdot P(B)$.

Assume that all n observations of the dataset are **independent** of one another.

Recall from statistics that if A and B are independent events, then the probability for the event A and B is given by $P(A \cap B) = P(A) \cdot P(B)$.

To cover the whole dataset, we need to consider all n observations, we define the total probability function $\mathscr{P}$ by

$$\mathscr{P} = P(y^{(1)}|\vec{x}^{(1)})P(y^{(2)}|\vec{x}^{(2)}) \cdots P(y^{(n)}|\vec{x}^{(n)}) = \prod_{i=1}^{n} P(y^{(i)}|\vec{x}^{(i)}) \,. \tag{4}$$

Binary Logistic Regression *(Cont...)*

Now that we have the total probability function (4), we should maximize it in order to find the predictions of the model for the target variable y.

Let's decide how we model the individual probabilities $P(y^{(i)}|\vec{x}^{(i)})$.

Question: Since $P(y^{(i)}|\vec{x}^{(i)})$ is a continuous quantity, can we model it via a linear relation (*i.e.* $P(y^{(i)}|\vec{x}^{(i)}) = \omega_0 + \omega_1 x_1^{(i)} + \cdots + \omega_d x_d^{(i)}$)?

Binary Logistic Regression *(Cont...)*

Now that we have the total probability function (4), we should maximize it in order to find the predictions of the model for the target variable y.

Let's decide how we model the individual probabilities $P(y^{(i)}|\vec{x}^{(i)})$.

Question: Since $P(y^{(i)}|\vec{x}^{(i)})$ is a continuous quantity, can we model it via a linear relation (*i.e.* $P(y^{(i)}|\vec{x}^{(i)}) = \omega_0 + \omega_1 x_1^{(i)} + \cdots + \omega_d x_d^{(i)}$)?
Answer: No!

- The linear (and polynomial) functions are unbounded
- $P(y^{(i)}|\vec{x}^{(i)})$ is a probability, and hence it belongs to the interval $(0, 1)$!

Binary Logistic Regression *(Cont...)*

Now that we have the total probability function (4), we should maximize it in order to find the predictions of the model for the target variable y.

Let's decide how we model the individual probabilities $P(y^{(i)}|\vec{x}^{(i)})$.

Question: Since $P(y^{(i)}|\vec{x}^{(i)})$ is a continuous quantity, can we model it via a linear relation (*i.e.* $P(y^{(i)}|\vec{x}^{(i)}) = \omega_0 + \omega_1 x_1^{(i)} + \cdots + \omega_d x_d^{(i)}$)?
Answer: No!

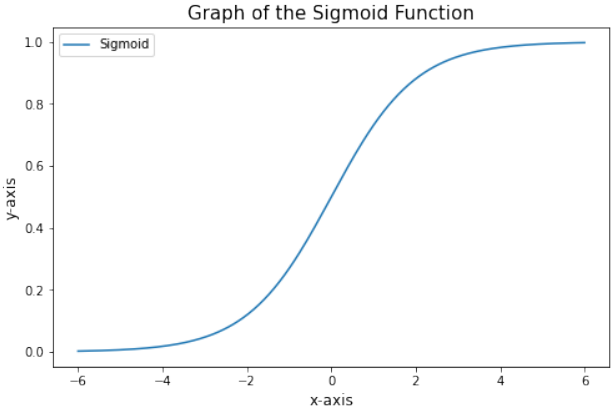
- The linear (and polynomial) functions are unbounded
- $P(y^{(i)}|\vec{x}^{(i)})$ is a probability, and hence it belongs to the interval $(0, 1)$!

In logistic regression, the *sigmoid function* is chosen to take care of this job.

Sigmoid Function σ

$$\sigma : \mathbb{R} \to (0, 1)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}} . \quad (5)$$

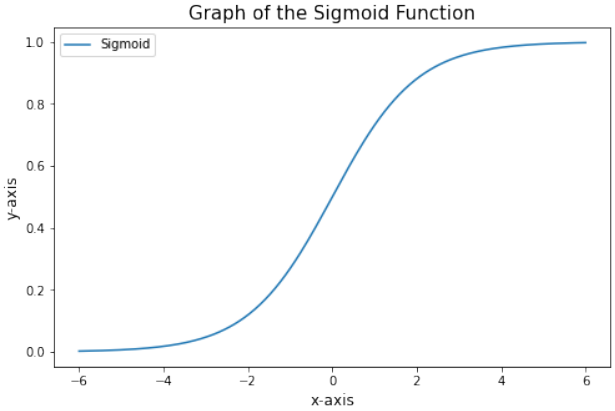
σ is a continuous and smooth bounded function that maps the entire real line $\mathbb{R} = (-\infty, +\infty)$ to the interval $(0, 1)$.



Sigmoid Function σ

$$\sigma : \mathbb{R} \to (0, 1)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}} . \qquad (5)$$

σ is a continuous and smooth bounded function that maps the entire real line $\mathbb{R} = (-\infty, +\infty)$ to the interval $(0, 1)$.



Note one can choose a different bounded function for this purpose.

Binary Logistic Regression *(Cont...)*

We can now define a linear function z

$$z^{(i)} = \omega_0 + \omega_1 x_1^{(i)} + \cdots + \omega_d x_d^{(i)} = \omega_0 + \vec{\omega} \cdot \vec{x}^{(i)} \,, \tag{6}$$

where

- ω_0 as usual represents the bias term and
- the weight vector $\vec{\omega} = (\omega_1, \omega_2, \cdots, \omega_d)$ contains the weights that correlate with the features $\vec{x} = (x_1, x_2, \cdots, x_d)$.

Binary Logistic Regression *(Cont...)*

In order to fulfill the boundedness property of the probability function $P(y^{(i)}|\vec{x}^{(i)})$, we define

- $\sigma(z^{(i)})$ is the probability to observe success (*i.e.* $y^{(i)} = 1$)
- $1 - \sigma(z^{(i)})$: is the probability to observe failure

for the *i*-th observation.

Binary Logistic Regression *(Cont...)*

In order to fulfill the boundedness property of the probability function $P(y^{(i)}|\vec{x}^{(i)})$, we define

- $\sigma(z^{(i)})$ is the probability to observe success (*i.e.* $y^{(i)} = 1$)
- $1 - \sigma(z^{(i)})$: is the probability to observe failure

for the *i*-th observation.

$$P(y^{(i)}|\vec{x}^{(i)}) = \begin{cases} \sigma(z^{(i)}) \,, & y^{(i)} = 1 \\ \\ 1 - \sigma(z^{(i)}) \,, & y^{(i)} = 0 \end{cases} \,. \tag{7}$$

Binary Logistic Regression: Odds

Let us denote the probability of finding success in the i-th observation by $p(z^{(i)}) = P(y^{(i)} = 1|\vec{x}^{(i)})$.

Then, the first case of (7) implies that $p(z^{(i)}) = \dfrac{1}{1+e^{-z^{(i)}}}$. One can then show that

$$e^{z^{(i)}} = e^{\omega_0 + \vec{\omega}\cdot\vec{x}^{(i)}} = \frac{p(z^{(i)})}{1-p(z^{(i)})} \,. \tag{8}$$

The quantity on the RHS of (8) is called the **odds** for the i-th observation.

Binary Logistic Regression: Odds

Let us denote the probability of finding success in the i-th observation by $p(z^{(i)}) = P(y^{(i)} = 1|\vec{x}^{(i)})$.

Then, the first case of (7) implies that $p(z^{(i)}) = \dfrac{1}{1+e^{-z^{(i)}}}$. One can then show that

$$e^{z^{(i)}} = e^{\omega_0 + \vec{\omega}\cdot\vec{x}^{(i)}} = \frac{p(z^{(i)})}{1-p(z^{(i)})} \,. \tag{8}$$

The quantity on the RHS of (8) is called the **odds** for the i-th observation.

- Odds belong to the interval $(0, \infty)$.
- Values of odds close to 0 indicate a low probability of success for the i^{th} observation.
- Large values of odds indicate a high probability of success for the i^{th} observation.

Binary Logistic Regression: Odds *(Cont...)*

Odds introduce a measure for the probability of success for each individual observation of the dataset.

Note that each observation has its own odds.

The logarithm of odds for the *i*-th observation is known as the **log-odds** or **logit**:

$$z^{(i)} = \omega_0 + \vec{\omega} \cdot \vec{x}^{(i)} = \log\left(\frac{p(z^{(i)})}{1 - p(z^{(i)})}\right), \tag{9}$$

and as is clear from equation (9), log-odds can vary from $-\infty$ to $+\infty$.

Binary Logistic Regression *(Cont...)*

We can combine the two branches of (7) into one single expression.

Using a simple trick, this can be done and both branches of (7) can be simultaneously described as follows:

$$P(y^{(i)}|\vec{x}^{(i)}) = \sigma(z^{(i)})^{y^{(i)}} \, (1 - \sigma(z^{(i)}))^{1-y^{(i)}} \, . \tag{10}$$

Binary Logistic Regression *(Cont...)*

We can combine the two branches of (7) into one single expression.

Using a simple trick, this can be done and both branches of (7) can be simultaneously described as follows:

$$P(y^{(i)}|\vec{x}^{(i)}) = \sigma(z^{(i)})^{y^{(i)}} \, (1 - \sigma(z^{(i)}))^{1-y^{(i)}} \, . \tag{10}$$

When $y^{(i)} = 1$ (*i.e.* success takes place), then $1 - y^{(i)} = 0$, and hence $P(y^{(i)}|\vec{x}^{(i)}) = \sigma(z^{(i)})$.

Binary Logistic Regression *(Cont...)*

We can combine the two branches of (7) into one single expression.

Using a simple trick, this can be done and both branches of (7) can be simultaneously described as follows:

$$P(y^{(i)}|\vec{x}^{(i)}) = \sigma(z^{(i)})^{y^{(i)}} \, (1 - \sigma(z^{(i)}))^{1-y^{(i)}} \, . \tag{10}$$

When $y^{(i)} = 1$ (*i.e.* success takes place), then $1 - y^{(i)} = 0$, and hence $P(y^{(i)}|\vec{x}^{(i)}) = \sigma(z^{(i)})$.

On the other hand, when $y^{(i)} = 0$ (*i.e.* failure takes place), then $1 - y^{(i)} = 1$ and $P(y^{(i)}|\vec{x}^{(i)}) = 1 - \sigma(z^{(i)})$.

Binary Logistic Regression *(Cont...)*

To present the total probability function $\mathscr{P}$, we need to multiply the probabilities associated with all *n* observations of the dataset. Hence, we arrive at

$$\boxed{\mathscr{P} = \prod_{i=1}^{n} \sigma(z^{(i)})^{y^{(i)}} \left(1 - \sigma(z^{(i)})\right)^{1-y^{(i)}}} \,. \tag{11}$$

Goal: We aim to find the **optimal values** for the weights ω_0 and $\vec{\omega}$ that maximize the total probability function (11).

Binary Logistic Regression *(Cont...)*

To present the total probability function $\mathscr{P}$, we need to multiply the probabilities associated with all n observations of the dataset. Hence, we arrive at

$$\mathscr{P} = \prod_{i=1}^{n} \sigma(z^{(i)})^{y^{(i)}} \left(1 - \sigma(z^{(i)})\right)^{1-y^{(i)}} . \tag{11}$$

Goal: We aim to find the **optimal values** for the weights ω_0 and $\vec{\omega}$ that maximize the total probability function (11).

Difficulty: We need to take the derivatives and set them to zero but taking the derivative of a product is a tedious task!

Desired: Conversion of the product in (11) into a sum:

$$\prod_{i=1}^{n} \longrightarrow \sum_{i=1}^{n} . \tag{12}$$

Desired: Conversion of the product in (11) into a sum:

$$\prod_{i=1}^{n} \longrightarrow \sum_{i=1}^{n} . \tag{12}$$

Solution: Define the **likelihood function** to be the logarithm of the total probability function

$$\ell(\omega_0, \vec{\omega}) = \log(\mathscr{P}) . \tag{13}$$

The logarithm on the RHS automatically converts the product into a sum because $\log(ab) = \log(a) + \log(b)$.

Binary Logistic Regression *(Cont...)*

Since logarithm is a monotonically increasing function, maximization of the total probability function results in the maximization of the likelihood function.

ML people prefer to minimize a cost function (rather than maximizing a function).

Binary Logistic Regression *(Cont...)*

Since logarithm is a monotonically increasing function, maximization of the total probability function results in the maximization of the likelihood function.

ML people prefer to minimize a cost function (rather than maximizing a function).

Then we can easily fulfill the desire by introducing the cost function to be

$$J(\omega_0, \vec{\omega}) = -\ell(\omega_0, \vec{\omega}) = -\sum_{i=1}^{n} \left[y^{(i)} \log(\sigma(z^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})) \right]. \quad (14)$$

The minus sign makes the maxima of the likelihood function correspond to the minima of the cost function of the logistic regression.

Binary Logistic Regression *(Cont...)*

Since logarithm is a monotonically increasing function, maximization of the total probability function results in the maximization of the likelihood function.

ML people prefer to minimize a cost function (rather than maximizing a function).

Then we can easily fulfill the desire by introducing the cost function to be

$$J(\omega_0, \vec{\omega}) = -\ell(\omega_0, \vec{\omega}) = -\sum_{i=1}^{n} \left[y^{(i)} \log(\sigma(z^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})) \right]. \quad (14)$$

The minus sign makes the maxima of the likelihood function correspond to the minima of the cost function of the logistic regression.

$J(\omega_0, \vec{\omega})$ is the **cost function of the logistic regression algorithm**.

To minimize $J(\omega_0, \vec{\omega})$, we set $\frac{\partial J(\omega_0, \vec{\omega})}{\partial \omega_j} = 0$ which, after some simple algebraic manipulations, lead to

$$\sum_{i=1}^{n} \left(y^{(i)} - \sigma(z^{(i)})\right) x_j^{(i)} = 0 \,, \tag{15}$$

for $j = 0, 1, \cdots, d$ with $x_0^{(i)} = 1$.

Binary Logistic Regression *(Cont...)*

To minimize $J(\omega_0, \vec{\omega})$, we set $\dfrac{\partial J(\omega_0, \vec{\omega})}{\partial \omega_j} = 0$ which, after some simple algebraic manipulations, lead to

$$\sum_{i=1}^{n} \left(y^{(i)} - \sigma(z^{(i)})\right) x_j^{(i)} = 0 \,, \tag{15}$$

for $j = 0, 1, \cdots, d$ with $x_0^{(i)} = 1$.

- Eq. (15) represents $d + 1$ equations, as j is a free index ranging from 0 to d.
- This system of eq.s need to be solved to determine the optimal ω_0 and $\vec{\omega}$.
- However, unlike the regression case, this cannot be done analytically!
- scikit-learn follows a gradient descent approach through the substitution $\vec{\omega} \longrightarrow \vec{\omega} - \eta \, \vec{\nabla} J(\vec{\omega})$.

Multinomial Logistic Regression

When we have more than two categories, then we need to build a Multinomial Logistic Regression.

MLR is also known as the Softmax Regression.

The softmax function appears very often in machine learning (and in particular in deep learning), and you need to have a solid understanding of this important function.

Softmax Function

$$\text{softmax} : \mathbb{R}^n \to (0,1)^n$$
$$\text{softmax}(z_1, z_2, \cdots, z_n) = \left(\frac{e^{z_1}}{\sum_{i=1}^n e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^n e^{z_i}}, \cdots, \frac{e^{z_n}}{\sum_{i=1}^n e^{z_i}} \right). \tag{16}$$

The input of the softmax function is an n-tuple and its output is another n-tuple.

Softmax Function

$$\text{softmax} : \mathbb{R}^n \to (0,1)^n$$
$$\text{softmax}(z_1, z_2, \cdots, z_n) = \left(\frac{e^{z_1}}{\sum_{i=1}^{n} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{n} e^{z_i}}, \cdots, \frac{e^{z_n}}{\sum_{i=1}^{n} e^{z_i}} \right). \tag{16}$$

The input of the softmax function is an n-tuple and its output is another n-tuple.

The output of the softmax function possesses two crucial properties:

- Every element of the output n-tuple belongs to the interval $(0, 1)$.
  Hence, every element can be interpreted as a **probability**!
- The sum of the n elements of the output n-tuple is 1.
  Hence, the output of the softmax function is a **discrete probability distribution**!

Softmax Function *(Cont...)*

- Suppose we have n entities, and we assign score to these entities.
- By passing the scores to the softmax function, we convert the scores into a discrete probability distribution.
- The individual entries of the output of the softmax function correspond to the probabilities of occurrences.
- The fact that the probabilities add up to 1 indicates that one of the cases will certainly take place.

Let us define the softmax function in python using numpy, and see how the function works in practice.

Softmax Function *(Cont...)*

```
: def softmax(x):                          # Define softmax function
      e = np.exp(x)
      return e/e.sum()

  examples = [[100, 100], [7, 5], [1, 2, 0], [4, 2, -100], [1, 2,␣
   ↪3, 4, 5], [-1, 2, 0, 1, -1, 3]]

  for x in examples:
      print('Softmax(%s)=%s\n' %(x, softmax(x)))
```

Softmax Function *(Cont...)*

```
Softmax([100, 100])=[0.5 0.5]

Softmax([7, 5])=[0.88079708 0.11920292]

Softmax([1, 2, 0])=[0.24472847 0.66524096 0.09003057]

Softmax([4, 2, -100])=[8.80797078e-01 1.19202922e-01 6.00136094e-46]

Softmax([1, 2, 3, 4, 5])=[0.01165623 0.03168492 0.08612854 0.23412166
0.63640865]

Softmax([-1, 2, 0, 1, -1, 3])=[0.01152193 0.23142412 0.03131985
0.08513618 0.01152193 0.62907599]
```

Softmax Function *(Cont...)*

```
four_tuples = [[3, 5, 6, 8], [3.5, 5.5, 6.5, 8.5], [-2, 0, 1, 3], [5, 7, 8, 10]]

for x in four_tuples:    # Calculating the softmax for above 4-tuples
    print('Softmax(%s)=%s\n' %(x, softmax(x)))
```

Softmax Function *(Cont...)*

```
Softmax([3, 5, 6, 8])=[0.0056533  0.04177257 0.11354962 0.83902451]

Softmax([3.5, 5.5, 6.5, 8.5])=[0.0056533  0.04177257 0.11354962 0.
↪83902451]

Softmax([-2, 0, 1, 3])=[0.0056533  0.04177257 0.11354962 0.83902451]

Softmax([5, 7, 8, 10])=[0.0056533  0.04177257 0.11354962 0.83902451]
```

Softmax Function *(Cont...)*

```
Softmax([3, 5, 6, 8])=[0.0056533  0.04177257 0.11354962 0.83902451]

Softmax([3.5, 5.5, 6.5, 8.5])=[0.0056533  0.04177257 0.11354962 0.
↪83902451]

Softmax([-2, 0, 1, 3])=[0.0056533  0.04177257 0.11354962 0.83902451]

Softmax([5, 7, 8, 10])=[0.0056533  0.04177257 0.11354962 0.83902451]
```

If we shift all elements of the input n-tuple by the same constant, the softmax function stays constant:

$$\boxed{\text{softmax}(z_1 + c, z_2 + c, \cdots, z_n + c) = \text{softmax}(z_1, z_2, \cdots, z_n)} \quad . \tag{17}$$

Multinomial Logistic Regression *(Cont...)*

Suppose we have a classification problem with K classes.

We need to specify the probability functions $P(y^{(i)} = k|\vec{x}^{(i)})$
(*i.e.* the probability that the target y is observed in class k in the i-th observation, given that the observed features are $\vec{x}^{(i)}$).

Multinomial Logistic Regression *(Cont...)*

Suppose we have a classification problem with K classes.

We need to specify the probability functions $P(y^{(i)} = k|\vec{x}^{(i)})$
(*i.e.* the probability that the target y is observed in class k in the i-th observation, given that the observed features are $\vec{x}^{(i)}$).

We can specify all probabilities at once by

$$\left(P(y^{(i)} = 1|\vec{x}^{(i)}), P(y^{(i)} = 2|\vec{x}^{(i)}), \cdots, P(y^{(i)} = K|\vec{x}^{(i)})\right) = \text{softmax}(z_1^{(i)}, z_2^{(i)}, \cdots, z_K^{(i)}), \tag{18}$$

where $z_j^{(i)} = \omega_{0,j} + \vec{\omega}_j \cdot \vec{x}^{(i)} = \omega_{0,j} + \omega_{1,j}x_1^{(i)} + \omega_{2,j}x_2^{(i)} + \cdots + \omega_{d,j}x_d^{(i)}$ for $j = 1, 2, \cdots, K$.

Multinomial Logistic Regression *(Cont...)*

In the case of multinomial classification, a *new set of weights* is introduced for *each class*.

The total number of parameters of the multinomial logistic regression model is $K(d+1)$.

Multinomial Logistic Regression *(Cont...)*

In the case of multinomial classification, a *new set of weights* is introduced for *each class*.

The total number of parameters of the multinomial logistic regression model is $K(d+1)$.

The parameters of the model can be accommodated in a $(d+1) \times K$ matrix:

$$\omega = \begin{pmatrix} \omega_{0,1} & \omega_{0,2} & \cdots & \omega_{0,K} \\ \omega_{1,1} & \omega_{1,2} & \cdots & \omega_{1,K} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{d,1} & \omega_{d,2} & \cdots & \omega_{d,K} \end{pmatrix} . \tag{19}$$

Each column of (19) casts the weights associated with a class of the target variable.

Multinomial Logistic Regression *(Cont...)*

The multinomial logistic regression cost function is defined by

$$J(\omega) = -\sum_{i=1}^{n}\sum_{k=1}^{K} \log\left(P(y^{(i)} = k|\vec{x}^{(i)})\right) = -\sum_{i=1}^{n}\sum_{k=1}^{K} \log\left[\frac{e^{z_k^{(i)}}}{\sum_{j=1}^{K} e^{z_j^{(i)}}}\right]. \tag{20}$$

Again, this can be solved via gradient descent type numerical method, not analytically.

Metrics to Evaluate Classifiers

- The metrics we used for linear regression can't be used for classification methods.
- To define meaningful metrics, let's define two boring classifiers:
  - **Sharp Classifier:** Label *all observations* with one class regardless of the features
  - **Monkey Classifier:** Label each observation randomly.
- Both classifiers perform poorly.
- We'll use them as a reference point though.

Metrics to Evaluate Binary Classifiers

To be consistent with the ML literature, we'll use

- **negative** label for bigger class
- **positive** label for the smaller class.

E.g.: People with cancer dataset.

- If the sizes of the two classes are equal, you can use either label
- **Balanced Case:** The sizes of the two classes are comparable.
- **Unbalanced Case:** The size of the negative class is much bigger than the size of the positive class.

Binary Classification: 4 Possibiliies

- **True Positive:** A classifier labels a positive instance as positive, resulting in a win for the classifier.
- **True Negative:** A classifier labels a negative item as negative, resulting in a win for the classifier.
- **False Positive:** A classifier mistakenly labels a negative item as a positive, resulting in a **Type I** classification error.
- **False Negative:** A classifier mistakenly labels a positive item as a negative, resulting in a **Type II** classification error.

Binary Classification: 4 Possibiliies

- **True Positive:** A classifier labels a positive instance as positive, resulting in a win for the classifier.
- **True Negative:** A classifier labels a negative item as negative, resulting in a win for the classifier.
- **False Positive:** A classifier mistakenly labels a negative item as a positive, resulting in a **Type I** classification error.
- **False Negative:** A classifier mistakenly labels a positive item as a negative, resulting in a **Type II** classification error.

We can conveniently represent the sizes of the above four distinct cases through the **Confusion Matrix**

$$CM = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix} , \tag{21}$$

Metric 1: Accuracy

The ratio of the number of correct predictions over total predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} . \tag{22}$$

Metric 1: Accuracy

The ratio of the number of correct predictions over total predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}. \tag{22}$$

Issue-1: Even a monkey classifier can achieve 50% accuracy in a balanced dataset.

Metric 1: Accuracy

The ratio of the number of correct predictions over total predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}. \tag{22}$$

Issue-1: Even a monkey classifier can achieve 50% accuracy in a balanced dataset.
Issue-2: For an unbalanced case (e.g. cancer dataset with 5% of the people with cancer), if you use the sharp classifier, you can get 95% accuracy!

Metric 2: Precision

Precisions for the positive and negative classes in a binary classification problem are defined as follows:

$$\begin{aligned} precision[P] &= \frac{TP}{TP + FP}, \\ precision[N] &= \frac{TN}{TN + FN}. \end{aligned} \tag{23}$$

Metric 2: Precision

Precisions for the positive and negative classes in a binary classification problem are defined as follows:

$$\begin{aligned} \textit{precision}[P] &= \frac{TP}{TP + FP}\,, \\ \textit{precision}[N] &= \frac{TN}{TN + FN}\,. \end{aligned} \tag{23}$$

Issue: Consider the cancer dataset with 5% of the people with cancer.
A sharp classifier (all are cancer free), $TP = FP = 0$, it is not even possible to define $\textit{precision}[P]$!

Metric 3: Recall

Recalls for the positive and negative classes in a binary classification problem are defined as follows:

$$\begin{aligned} recall[P] &= \frac{TP}{TP + FN}, \\ recall[N] &= \frac{TN}{TN + FP}. \end{aligned} \tag{24}$$

Note: ($recall[P] = \frac{TP}{TP + FN}$) is also known as **sensitivity** in the ML literature.

Metric 3: Recall

Recalls for the positive and negative classes in a binary classification problem are defined as follows:

$$\begin{aligned} recall[P] &= \frac{TP}{TP + FN}, \\ recall[N] &= \frac{TN}{TN + FP}. \end{aligned} \tag{24}$$

Note: ($recall[P] = \frac{TP}{TP + FN}$) is also known as **sensitivity** in the ML literature.

In this case, the recall of the positive class is 0 for the cancer dataset. This means that the sharp classifier performs very poor on the positive class.

Why do we need both precision and recall?

Consider the cancer classification problem,

- A **false positive** case is a participant who does not actually have cancer, but has mistakenly been labeled as a cancer patient.
- A **false negative** case is a cancer patient who has been labeled as healthy.

Why do we need both precision and recall?

Consider the cancer classification problem,

- A **false positive** case is a participant who does not actually have cancer, but has mistakenly been labeled as a cancer patient.
- A **false negative** case is a cancer patient who has been labeled as healthy.
- Due to the severe consequences, it is evident that we can tolerate false positive cases, but our tolerance for false negative is very low!
- From (24), it is straightforward to see that a high $\mathit{recall}[P]$ (close to 1) guarantees that your classifier has a very few false negatives!

Metric 4: $F1$-Score

$F1$-score is the *harmonic mean* of the precision and recall for each class.

$$F1[P] = \frac{2\,precision[P]\,recall[P]}{precision[P] + recall[P]}\,,$$
$$F1[N] = \frac{2\,precision[N]\,recall[N]}{precision[N] + recall[N]}\,. \tag{26}$$

Metric 4: $F1$-Score

$F1$-score is the *harmonic mean* of the precision and recall for each class.

$$F1[P] = \frac{2\,precision[P]\,recall[P]}{precision[P] + recall[P]}\,,$$
$$F1[N] = \frac{2\,precision[N]\,recall[N]}{precision[N] + recall[N]}\,. \tag{26}$$

- If one is looking for a single metric to make a good judgement about the performance of a classifier, $F1$-score would be that metric.
- $F1$-score is a very tough measure to beat (because the harmonic mean is always less than or equal to the arithmetic mean)

Summary of the Metrics

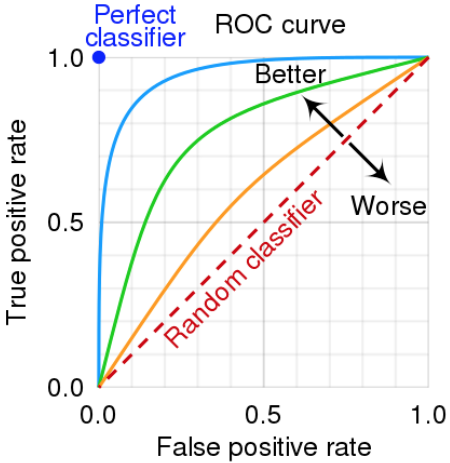
1. Accuracy is a misleading metric in unbalanced classification problems!
2. High precision is very hard to achieve in unbalanced classification problems!
3. $F1$-score does the best job of any single metric, but all four of them work together to evaluate the performance of a classifier!

Receiver-Operator Characteristic (ROC)

ROC is a graphic method of evaluating binary classifiers in which one plots the true positive rate versus the false positive rate at various thresholds.

$$TPR = \frac{TP}{TP + FN} = sensitivity,$$
$$FPR = \frac{FP}{FP + TN} = 1 - specificity\ . \quad (27)$$

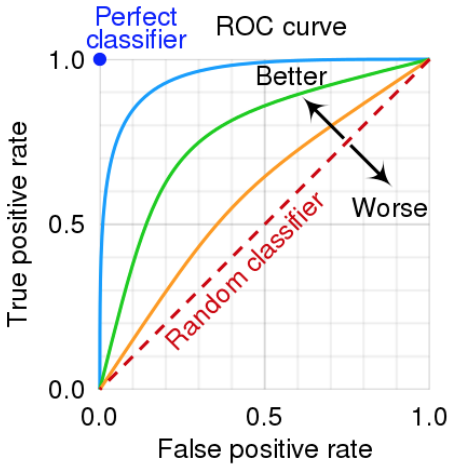
where $specificity = \frac{TN}{TN+FP}$



Receiver-Operator Characteristic (ROC) *(Cont...)*

- AUC = Area Under the Curve
- AUC for the monkey classifier is 1/2.
- AUC for a perfect classifier is 1.
- AUC for a given classifier is equal to the arithmetic mean of sensitivity and specificity, *i.e.*

$$AUC = \frac{sensitivity + specificity}{2}.$$



Metrics to Evaluate Multinomial Classifiers

Suppose

- We are dealing with a classification problem with K classes.
- Let $C[i, j]$ denote the number of instances of class i that received label j by the classifier.
  - i (the first index) in $C[i, j]$ denotes the true (actual) label and
  - j (the second index) denotes the predicted label.

Metrics to Evaluate Multinomial Classifiers

Suppose

- We are dealing with a classification problem with K classes.
- Let $C[i, j]$ denote the number of instances of class i that received label j by the classifier.
  - i (the first index) in $C[i, j]$ denotes the true (actual) label and
  - j (the second index) denotes the predicted label.
- We can then define a $K \times K$ **confusion matrix** as follows:

$$CM = \begin{pmatrix} C[1,1] & C[1,2] & \cdots & C[1,K] \\ C[2,1] & C[2,2] & \cdots & C[2,K] \\ \vdots & \vdots & \ddots & \vdots \\ C[K,1] & C[K,2] & \cdots & C[K,K] \end{pmatrix}. \tag{28}$$

Metrics to Evaluate Multinomial Classifiers

$$Accuracy = \sum_{i=1}^{K} C[i,i] / \sum_{j=1}^{K} \sum_{k=1}^{K} C[j,k] \ . \tag{29}$$

Precision for each class $[i]$ of the classification:

$$Precision[i] = C[i,i] / \sum_{j=1}^{K} C[j,i] \ . \tag{30}$$

Recall for each class $[i]$ of the classification:

$$recall[i] = C[i,i] / \sum_{j=1}^{K} C[i,j] \ . \tag{31}$$

Note: scikit-learn library calculates the confusion matrix and all the above mentioned metrics for a classifier and represents the summary in a nice manner.