Introduction to Machine Learning

Lecture 3 - Linear Model for Classification

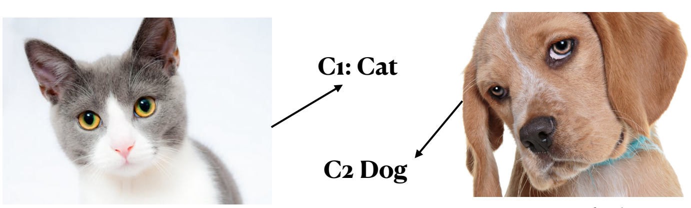
Guang Bing Yang, PhD

Linear Models for Classification

- So far, we have learned the linear models for regression—which have particularly simple analytical and computational properties.
- In this lecture, we will study another kind of linear models for saving classification problems.
- In addition, this lecture will discuss the Bayesian treatment of linear models for classification.

Classification

- The purpose of classification is to assign one of K discrete categories (classes) $C_k, (k = 1,\ldots,K)$ to an input X.
- Each input corresponds to only one class, normally.
- Example: The input vector x as the set of pixels of images, and the output variable t will represent the either cat, class C_1 or dog, class C_2



X —set of pixels

X —set of pixels

Linear Models for Classification

- Due to its simple analytical and computational properties, we will consider linear models first.
- Remember, the linear regression case, the model is linear in parameters:
  - $y(\mathrm{x}, \mathrm{w}) = \mathrm{x}^T\mathrm{w} + w_0$, (both linear for parameters and inputs)
  - $y(\mathrm{x}, \mathrm{w}) = f(\mathrm{x}^T\mathrm{w} + w_0)$, linear in parameters but fixed non-linear in inputs.
- For classification, the model needs to predict discrete class labels (or posterior probabilities in range (0, 1), thus it needs to do one more step—decision of classes.

Linear Models for Classification

- Due to its simple analytical and computational properties, we will consider linear models first.
- Remember, the linear regression case, the model is linear in parameters:
  - $y(\mathrm{x}, \mathrm{w}) = \mathrm{x}^T \mathrm{w} + w_0$, (both linear for parameters and inputs)
  - $y(\mathrm{x}, \mathrm{w}) = f(\mathrm{x}^T \mathrm{w} + w_0)$, linear in parameters but fixed non-linear in inputs.
- For classification, the model needs to predict discrete class labels (or posterior probabilities in range (0, 1), thus it needs to do one more step—making decision.
- Decision boundaries or decision surfaces are defined as boundaries that partition the input space (vector space) into regions, one for each class.

Linear Models for Classification

- The decision surfaces correspond to $y(\mathrm{x}, \mathrm{w}) = const = \mathrm{x}^T \mathrm{w} + w_0$.
- Hence the decision surfaces are linear functions of x, even if the activation function is nonlinear.
- Thus, we say the linear models for classification, the linear is about the decision surfaces over input x because the decision surfaces are const regarding to x.
- Note that these models are no longer linear in parameters, due to the presence of nonlinear activation function.
- Remember the distinguish of the linear between regression and classification.
  - In regression, the linear is over parameters, the basis function can be nonlinear or linear.
  - in classification, the linear is about decision surfaces over input vector x, the activation function normally is non-linear.
- This nonlinearity in parameters leads to more complex analytical and computational properties in classification problems if compared to linear regression.
- Same as the regression models, a fixed nonlinear transformation of the input variables can be applied for by using a vector of basis functions $\Phi(x)$, as we did for regression models.

Notation

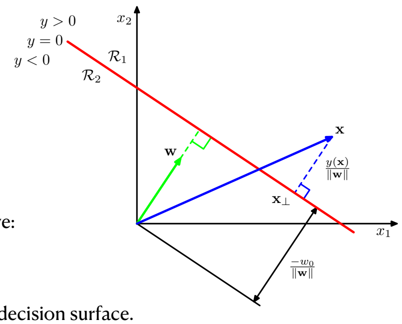
- For the binary classification—the case of two-class problems, use the binary representation for the target value $t \in \{0,1\}$, such that t=1 represents the *positive class* and t=0 represents the *negative class.*
- If the output of the model is represented as the probability that the model assigns to the positive class, we can interpret the t as the probability distribution of the positive class, which is given as $p(C_k | t = 1)$.
- For multiple classification, there are K classes, we use a 1-of-K encoding scheme, in which t is a vector of length K containing a single 1 for the correct class and 0 elsewhere.
- For example, if we have K=5 classes, then an input that belongs to class 2 would be given a target vector as: $t = (0,1,0,0,0)^T$

Approaches to Classification

- Basically, there are three approaches to classification problems.
- First approach: create a **discriminant function**—directly maps each input vector to a specific class.
- Second approach: model the ***conditional probability distribution*** $p(C_k | \mathrm{x})$ with a **discriminative** approach.
  - model $p(C_k | \mathrm{x})$,
  - e.g., logistic regression
- Third approach: model the ***class conditional densities*** $p(\mathrm{x} | C_k)$ together with the class prior probabilities $p(C_k)$. Then, infer **posterior** probability using **Bayes' rule**:
  - $p(C_k | \mathrm{x}) = \dfrac{p(\mathrm{x} | C_k) p(C_k)}{p(\mathrm{x})}$,
  - e.g., fit multivariate Gaussians to the input vectors.

Discriminant Function

- Given a **discriminant function**—$y(\mathrm{x}, \mathrm{w}) = \mathrm{x}^T\mathrm{w} + w_0$.
- Define a **decision boundary (surface)**:
  - $y(\mathrm{x}) = 0$
- Assign x to C_1 if $y(\mathrm{x}) \geq 0$, and class C_2 otherwise.
- Two points x_A, x_B lie on the decision surface,
- we have: $y(\mathrm{x}_A) = y(\mathrm{x}_B) = 0$ and $\mathrm{w}^T(\mathrm{x}_A - \mathrm{x}_B) = 0$
- The W is orthogonal to the decision surface.
- if x is a point on the decision surface, then we have:
  - $\frac{\mathrm{w}^T\mathrm{x}}{||\mathrm{w}||} = -\frac{w_0}{||\mathrm{w}||}$
  - Hence, w_0 determines the position of the decision surface.

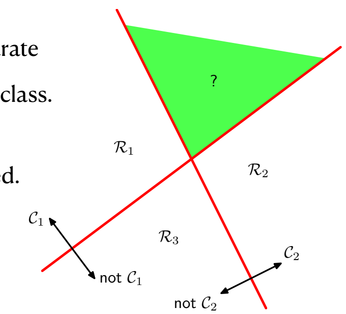


yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Discriminant Function for Multiple Classes

- Consider a **discriminant function** to K>2 classes.
- One solution is use K-1 classifiers,
  - each of them is a two-class problem—separate the points in class C_k from points not in that class.
- Issues: some points are ambiguously classified.

One vs. the rest

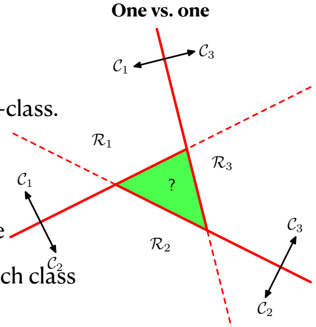


yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Discriminant Function for Multiple Classes

- Consider a **discriminant function** to K>2 classes.
- An alternative solution is use K(K-1)/2 binary discriminant classifiers.
  - each of functions discriminates between two-class.
- Issues: some points are ambiguously classified.
- A simple solution:
  - use K linear discriminant functions of the form: $y_k(\mathrm{x}) = \mathrm{x}^T\mathrm{w}_k + w_{k0}, k = 1,\ldots,K$, for each class
  Assign x to class C_k if $y_k(\mathrm{x}) > y_j(\mathrm{x}) \forall j \neq k$

One vs. one



yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Discriminant Function for Multiple Classes

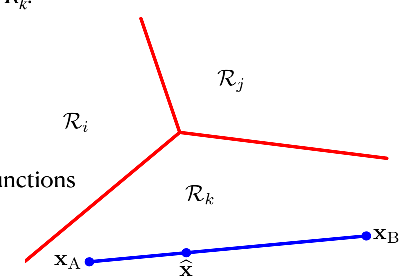
- This solution keeps the decision surfaces convex and singly connected.
- For any two points X_A, X_B inside the region R_k:

we have: $y_k(\mathrm{x}_A) > y_j(\mathrm{x}_A)$ and $y_k(\mathrm{x}_B) > y_j(\mathrm{x}_B)$

implies that for a positive α,

$y_k(\alpha\mathrm{x}_A + (1-\alpha)\mathrm{x}_B) > y_j(\alpha\mathrm{x}_A + (1-\alpha)\mathrm{x}_B)$

because of the linearity of the discriminant functions and the convex of the decision surfaces.



yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Least Squares Loss for Classification

- Consider a general case—K classes using 1-of-K encoding scheme for the target vector t.
- Simplify the Least Square approximates the conditional expectation $E[t\,|\,\mathrm{x}]$.
- Remember each class is described by its own linear model: $y_k(\mathrm{x}) = \mathrm{x}^T\mathrm{w}_k + w_{k0}, k = 1,\ldots,K$
- merge interpreter or bias part into the parameter vector: $\tilde{\mathrm{w}}_k = (w_{0k}, \mathrm{w}_k^T)^T$ and add one to input vector x: $\tilde{\mathrm{x}} = (1,\mathrm{x}^T)^T$.
- The updated linear model denoted using vectors: $\mathrm{y}(\mathrm{x}) = \tilde{W}^T\tilde{\mathrm{x}}$
- A Python Numpy solution for this merge processing is given as follows:

Least Squares Loss for Classification

- A Python Numpy solution for this merge processing is given as follows:

Vector Notation for Linear Model

$$y_k(\mathbf{x}) = \mathbf{x}^T\mathbf{w}_k + w_{k0}, k = 1,\ldots,K$$

to

$$\mathbf{y}(\mathbf{x}) = \tilde{W}^T\tilde{\mathbf{x}}$$

while

$$\tilde{\mathbf{w}}_k = (w_{0k}, \mathbf{w}_k^T)^T$$

and

$$\tilde{\mathbf{x}} = (1, \mathbf{x}^T)^T$$

```
[1]: import numpy as np

[2]: def vector_notation(x, ones):
         assert x.shape[0]==ones.shape[0], print("input vector and add-on ones must have the same row dimension.")
         return np.hstack((ones, x))

[4]: x = np.arange(10).reshape(5, 2)
     one = np.ones_like(np.arange(5)).T[:, np.newaxis]

[5]: # add one to the input vector as the first column
     x1=vector_notation(x, one)

[8]: w = np.random.random(2).reshape(1,2)
     w0=np.array([0.3]).reshape(-1, 1)

[10]: tilde_w = vector_notation(w, w0.T)

[11]: tilde_w

[11]: array([[0.3       , 0.67619597, 0.1589746 ]])
```

Evaluation

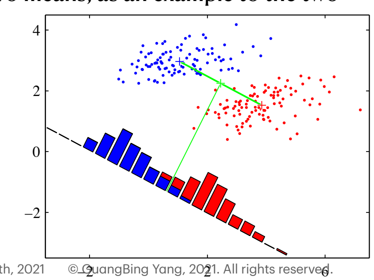
```
[12]: y=np.dot(x1, tilde_w.T)

[13]: y

[13]: array([[0.4589746 ],
             [2.12931573],
             [3.79965687],
             [5.469998  ],
             [7.14033914]])

[14]: y1=np.dot(x, w.T)+w0

[15]: y1

[15]: array([[0.4589746 ],
             [2.12931573],
             [3.79965687],
             [5.469998  ],
             [7.14033914]])

[16]: np.allclose(y, y1)

[16]: True

[6]: x, one, x.shape, one.shape

[6]: (array([[0, 1],
             [2, 3],
             [4, 5],
             [6, 7],
             [8, 9]]),
      array([[1],
             [1],
             [1],
             [1],
             [1]]),
      (5, 2),
      (5, 1))
```


Least Squares Loss for Classification

- Given a dataset $\{\mathrm{x}_n, \mathrm{t}_n\}, n = 1,\ldots,N$.
- Based on the **normal equation** and using some matrix algebra, we have the optimal weights (trained parameters):
  - $\tilde{W} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T$, while $\tilde{X} \in R^{N,D+1}$, whose nth row is $\tilde{x}_n^T$, $T \in R^{N,K}$, whose nth row is t_n^T.
- For a new input x is assigned to a class for which: $y_k(\mathrm{x}) = \tilde{X}^T\tilde{\mathrm{w}}_k, k = 1,\ldots,K$ is largest.
- The least Squares is sensitive to outliers.
- Usually, using logistic regression or Fisher's Linear Discriminant to solve this issue.

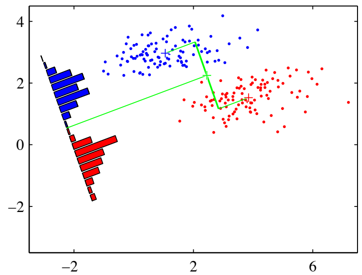
Fisher's Linear Discriminant

- Intuition: Project input vector to a low dimension, then maximizes the class separation in the projection.
- The separation of the projected class means is the simplest measure. The input vector is projected onto the line that joins the two means, as an example to the two-class classification.
- Fisher's idea is about *"maximize a function that:"*
  - produces the largest separation between the projected class means,
  - also minimizes class overlap by giving a small variance within each class.
- There is overlap between classes.



Fisher's Linear Discriminant

- Apply Fisher's linear discriminant, we hope get the projection shown in the following chart. Two classes are separated well with very less overlay.
- The mean of two classes is given by:
  - $m_1 = \frac{1}{N_1}\sum_{n\in C_1} x_n, m_2 = \frac{1}{N_2}\sum_{n\in C_2} x_n$
  - They are arithmetic means of input points—how many data points belong to class C1 and C2.
- Projecting onto the vector separating maximumly the two classes (between-class variance):
  - $w \propto m_1 - m_2$



Fisher's Linear Discriminant

- Minimize the within-class variance:
- $s_1^2 = \sum_{n\in C_1}(y_n - m_1)^2$, $s_2^2 = \sum_{n\in C_2}(y_n - m_2)^2$, where $m_k = w^T m_k$, $y_n = w^T x_n$, *where* $k = \{1,2,...K\}$
- define the total within-class variance be: $s_1^2 + s_2^2$.
- Fisher's criterion: maximize ration of the between-class variance to within-class variance: $J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{w^T S_b w}{w^T S_w w}$
- where the between-class and within-class covariance matrices are given by:
- $S_b = (m_2 - m_1)(m_2 - m_1)^T$ and $S_w = \sum_{n\in C_1}(x_n - m_1)(x_n - m_1)^T + \sum_{n\in C_2}(x_n - m_2)(x_n - m_2)^T$
- To maximize the $J(w)$, we have to maximize $w^T S_b w$ and minimize $w^T S_w w$
- Differentiating $J(w)$ w.r.t. w we have: $(w^T S_b w)S_w w = (w^T S_w w)S_b w$,
- multiplying by S_w^{-1} to both side of the above expression, and consider the $w^T S_b w$ and $w^T S_w w$ are scalar factors and $S_b w$ is always in the direction of $m_2 - m_1$, we have $(w^T S_b w)S_w^{-1}S_w w = (w^T S_w w)S_w^{-1}S_b w$, the optimal solution is:
- $w \propto S_w^{-1}(m_2 - m_1)$, hence we found w is the proportional to the difference of the class means, which is the same as our hypothesis shown in the previous slide.

yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Fisher's Linear Discriminant

- Maximizing J(w) is equivalent to the following constraint optimization problem, known as the generalized eigenvalue problem: $min_w - w^T S_b w, \quad \rightarrow w^T S_w w^T = 1,$
- The Lagrangian: $\mathbf{L}(w) = - w^T S_b w + \lambda(w^T S_w w - 1)$.
- Differentiating $J(w)$ w.r.t. w, the following equation needs to be hold:
- $2S_b w = 2\lambda S_w w$
- This is given by the eigenvector of $S_w^{-1}S_b$, where w is its largest eigenvalue—which is also the first level of derivatives of J(w) r.s.t. w.

yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Second approach—Probabilistic Generative Models

- Model class conditional densities $p(x\,|\,C_k)$ together with the prior probabilities $p(C_k)$ for the classes. Remember the Bayes' rule:

$$p(C_k\,|\,x) = \frac{p(x\,|\,C_k)p(C_k)}{p(x)}.$$

- Each class has its own class conditional densities $p(x\,|\,C_k)$ and prior $p(C_k)$.
- For two-class case (binary classification), the posterior probability of class C_1 is given as:

$$p(C_1\,|\,x) = \frac{p(x\,|\,C_1)p(C_1)}{p(x\,|\,C_1)p(C_1) + p(x\,|\,C_2)p(C_2)} = \frac{1}{1 + exp(-a)} = \sigma(a),$$ this is the logistic sigmoid function, where define:

$$a = \ln\frac{p(x\,|\,C_1)p(C_1)}{p(x\,|\,C_2)p(C_2)} = \ln\frac{p(C_1\,|\,x)}{1 - p(C_1\,|\,x)},$$

which is known as the logic function. It is the log of the ratio of probabilities of two classes, also known as the log-odds.

yguangbing@gmail.com, Guang.B@chula.ac.th February 5th, 2021

Second approach—Probabilistic Generative Models

- The posterior probability of the class C_1 is given as:

  $p(C_1 \mid \mathrm{x}) = \dfrac{p(\mathrm{x} \mid C_1)p(C_1)}{p(\mathrm{x} \mid C_1)p(C_1) + p(\mathrm{x} \mid C_2)p(C_2)} = \dfrac{1}{1 + exp(-a)} = \sigma(a)$, this is the logistic sigmoid function,

The term sigmoid means S-shaped: it maps the whole real number into (0,1). See the review lecture note and lecture 1 for more details. Repeat here its properties:

$$\sigma(-a) = 1 - \sigma(a), \quad \frac{d}{da}\sigma(a) = \sigma(a)(1 - \sigma(a)) .$$

They are easy to be verified. You can do it.

Second approach—Probabilistic Generative Models

- For multiple classes case, K>2 the class C_k is given as:

$$p(C_k \mid \mathrm{x}) = \frac{p(\mathrm{x} \mid C_k)p(C_k)}{\sum_j p(\mathrm{x} \mid C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}, \quad a_k = \ln[p(\mathrm{x} \mid C_k)p(C_k)] .$$

This is the **Softmax** function,

It is a smoothed version of the max function:

if $a_k \gg a_j, \forall j \neq k$, then $p(C_k \mid \mathrm{x}) \approx 1, \ p(C_j \mid \mathrm{x}) \approx 0.$

See the review lecture note and lecture 1 for more details. For implementation of the softmax and its derivatives, see assignment 1.

Third approach—Discriminative Modelling

- In the second approach, we model the class conditional densities with prior distribution, then applying the Bayes' rule to get the posterior distribution of the class, which is a fully generative modeling.
- In the discriminative approach, we model the $p(C_k \mid \mathrm{x})$ directly by representing them as parametric models, and optimize parameters using the training data. (e.g., logistic regression).
- Let's focus on Logistic regression. Use the two-class classification as an example.

Logistic Regression — Discriminative Modelling

- Let's focus on Logistic regression. Use the two-class classification as an example.
- Given $a = \mathrm{w}^T\mathrm{x}$, the logistic sigmoid function (given in previous slides):
- $p(C_1 \mid \mathrm{x}) = \dfrac{1}{1 + exp(-\mathrm{w}^T\mathrm{x})} = \sigma(\mathrm{w}^T\mathrm{x})$, where $p(C_2 \mid \mathrm{x}) = 1 - p(C_1 \mid \mathrm{x})$.
- This model is known as logistic regression (Note that this is a model for classification).
- Let's see how to obtain the optimal parameters using Maximum Likelihood Estimation approach.
- For a two-class case, the likelihood function takes form:
- $p(\mathrm{t} \mid \mathrm{X},\mathrm{w}) = \prod_{n=1}^{N} (y_n^{t_n}(1 - y_n)^{1-t_n}), \ y_n = \sigma(\mathrm{w}^T\mathrm{x}),$
- Define an error function by taking the negative log of the likelihood:

  $E(\mathrm{w}) = -\ln p(\mathrm{t} \mid \mathrm{w}) = -\sum_{n=1}^{N} [t_n \ln y_n + (1 - t_n)\ln(1 - y_n)] = \sum_{n=1}^{N} E_n$, where $y_n = \sigma(a_n)$, and $a_n = \mathrm{w}^T\mathrm{x}_n$, here we can use any basis function $\Phi(\mathrm{x}_n)$ to replace x_n .
- Differentiating and using the chain rule: $\dfrac{d}{dy_n}E_n = \dfrac{y_n - t_n}{y_n(1 - y_n)}, \ \dfrac{d}{d\mathrm{w}}y_n = y_n(1 - y_n)\mathrm{x}_n$, since $\dfrac{d}{da}\sigma(a) = \sigma(a)(1 - \sigma(a))$, hence,
- $$\frac{d}{d\mathrm{w}}E_n = \frac{E_n}{dy_n}\frac{dy_n}{d\mathrm{w}} = (y_n - t_n)\mathrm{x}_n,$$

Logistic Regression — Discriminative Modelling

- Based on the result of the differentiating shown in the previous slide, we obtain the error function:
- $\nabla E(\mathrm{w}) = \sum_{n=1}^{N} (y_n - t_n)\mathrm{x}_n$, where y_n is the prediction, and t_n is the target.
- This is exactly the same form as the gradient of the sum-of-squares error function for the linear regression model.

Multiclass Logistic Regression — Discriminative Modelling

- For multiple class case, the posterior probabilities are represented by a softmax function that transforms the linear functions of input variables to probabilities.
- $p(C_k \mid \mathrm{x}) = y_k(\mathrm{x}) = \dfrac{exp(\mathrm{w}_k^T \mathrm{x})}{\sum_j exp(\mathrm{w}_j^T \mathrm{x})}$
- The likelihood function: $p(T \mid X, \mathrm{w}_1, \ldots, \mathrm{w}_K) = \prod_{n=1}^{N} \left[\prod_{k=1}^{K} p(C_k \mid \mathrm{x}_n)^{t_{nk}} \right] = \prod_{n=1}^{N} \left[\prod_{k=1}^{K} y_{nk}^{t_{nk}} \right]$, where $T \in \mathbf{R}^{N \times K}$
- Define the error function as the negative logarithm of the cross-entropy function for multi-class classification: $W(\mathrm{w}_1, \ldots, \mathrm{w}_K) = -\ln p(T \mid X, \mathrm{w}_1, \ldots, \mathrm{w}_K) = -\sum_{n=1}^{N} \left[\sum_{k=1}^{K} t_{nk} \ln y_{nk} \right]$,
- Its gradient w.r.t. one of the parameter vectors w_j: $\nabla E_{\mathrm{w}_j}(\mathrm{w}_1, \ldots, \mathrm{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\mathrm{x}_n$,

Multiclass Logistic Regression — Discriminative Modelling

- Consider a softmax function for two classes (C1 and C2):
- $p(C_1 \mid \mathrm{x}) = \dfrac{exp(\mathrm{w}_1^T \mathrm{x})}{exp(\mathrm{w}_1^T \mathrm{x}) + exp(\mathrm{w}_2^T \mathrm{x})} = \dfrac{1}{1 + exp(-(\mathrm{w}_1^T \mathrm{x} - \mathrm{w}_2^T \mathrm{x}))} = \sigma(\mathrm{w}_1^T \mathrm{x} - \mathrm{w}_2^T \mathrm{x})$
- Thus, the logistic sigmoid is just a special case of the softmax function.

Recap

- The purpose of classification is to assign one of K discrete categories (classes) $C_k, (k = 1,\ldots,K)$ to an input X.
- There are three approaches to classification problems:
  1. **discriminant function**—directly maps each input vector to a specific class.
  2. **discriminative modelling a conditional probability distribution** $p(C_k \mid \mathrm{x})$
  3. **generative modelling class conditional densities** $p(\mathbf{x} \mid C_k)$ together with **the prior probabilities** $p(C_k)$ for the classes, then using the **Bayes' rule** to get **the posterior** probabilistic distribution of the classes.
- Logistic regression is a classification approach. For two-class case, it is a sigmoid function, for multi class case, it is a softmax function.

Assignment 2

- Assignment 2 worth 15%, and is about linear regression and binary classification Python programming. It was also posted in MS Teams Assignments.
- Copy and download my Colab from Chula G drive to your Google drive (Important note: Don't modify my Colab notebook, otherwise other classmates will see your work.)
- Working on your copy of the Colab notebook. Don't forget to add your name and student id in it.
- After finishing it, share it with me (only me, do not share your work with others.)
- All programming exercises MUST be running correctly in Colab without any errors and exceptions. If your code cannot run at all, and I cannot see any kind of outputs, you receive no grade points for that part.
- Before you submit your Colab notebook, make sure to leave the outputs (results) of the functions in the notebook. I ONLY review the outputs of your functions or the final results.
- The assignment due at Feb 19th, 2021. It is an individual assignment. Please no late due. I will start evaluating your work at Feb 20th, and try my best to give you feedback 1 week after.

Any questions?
Lab session

Feb 12th is a holiday, no online class, but a recorded video lecture will be posted on MS Teams

