Introduction to Machine Learning

Lecture 7 - Summary of Supervised Learning

Guang Bing Yang, PhD

Supervised Learning

- Supervised learning is one of three types of machine learning.
- It arms to learn a mapping from input X to output y, given a labeled set of input-output pairs D={X, y} called training set.
- In supervised learning, there are two categories of problems: Regression and Classification.
- what is the regression problem and what is the classification problem?

Supervised Learning : Linear Models for Regression

- It is a kind of supervised learning problem.
- It learns a mapping from inputs X to output Y, given a labeled set of pairs—**training sets.** $D = \{(x_i, y_i)\}_{i=1}^{N}$.
- Y is real-valued scalar or continuous, like 0.1234, 123, etc.
- Simply say, learning a continuous function is called **regression**.
- Based on above, this type of supervised learning problem is known as the regression.
- The goal of regression is:
  - To predict the value of one or more continuous output (sometimes also called targets or responsible variable) variables Y given the value of input variables X.

Supervised Learning : Linear Models for Regression

- Given the simplest linear regression model:
  - $y = f(\mathrm{x}, \mathrm{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$,

  where **w** are parameters, and **x** are input variables in D-dimensions, $\mathrm{x} = (x_1, \ldots, x_D)^T$.
- The 'linear' is regard to **parameters w** rather than inputs X.
- The above linear regression model is the simplest one because the both functions of parameters and inputs are linear.
- Use a function, $\Phi(\mathrm{x})$, to replace the **x**, where $\Phi(\,.\,)$ is called **basis function**.
  - For example, in the above the simplest linear regression model, the basis function is $\Phi(\mathrm{x}) = \mathrm{x}$.

Supervised Learning : Linear Models for Regression

- Now, the linear regression model can be represented as:
  - $y = f(\mathrm{x}, \mathrm{w}) = w_0 + w_1\Phi(x_1) + \ldots + w_D\Phi(x_D)$, Or,
  - $y = f(\mathrm{x}, \mathrm{w}) = w_0 + \sum_{j=1}^{M-1} w_j\phi_j(\mathrm{x})$,
  - where $\phi_j(\mathrm{x})$ are known as **basis functions** corresponding to the jth parameter w_j. Note that $\phi_i(\mathrm{x})$ and $\phi_j(\mathrm{x})$ are not necessary to be the same function to inputs X.
  - In realistic machine learning problem, these basis functions are often defined as feature functions.

Supervised Learning : Linear Models for Regression

- The parameter w_0 is a fixed offset in the data and is called "**intercept**" in statistics and "**bias parameter**" in machine learning.
- For convenient, let's define an additional ***dummy basis function*** $\phi_0(\mathrm{x}) = 1$, so that
  - $y = f(\mathrm{x}, \mathrm{w}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathrm{x}) = \mathrm{w}^T\Phi(\mathrm{x})$,
  - where $\mathrm{w} = (w_0, \ldots, w_{M-1})^T$ and $\Phi = (\phi_0, \ldots, \phi_{M-1})^T$, and there are M parameters and M basis functions.
- Some examples of basis functions:
  - *Gaussian* basis function, logistic sigmoid, and others.

Linear Regression Model — Cost Function

- The **cost function** is defined as the difference between expected values and actual values of outputs.
- The most common and the basis cost function in machine learning is the **least squares error** function, also called the **sum-of-squares error** functions:

$$E_D(\mathrm{w}) = ||y - \hat{y}||^2$$

or,

$$E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y_n - \mathrm{w}^T\Phi(\mathrm{x}_n)\}^2$$

which substituted $\hat{y}$ with $\mathrm{w}^T\Phi(\mathrm{x})$, and brought in the summation processing of all training data points

Linear Regression Model — Cost Function & Gradients

- To get the optimized parameters W's values by minimizing the cost function, get expected parameters W.
- By setting the gradient of the $E_D(\mathrm{w})$ over W to zero gives the minimization of the sum-of-squares error function.

$$0 = \frac{\partial E_D(\mathrm{w})}{\partial \mathrm{w}} = \frac{2}{2}\sum_{n=1}^{N}\{y_n - \mathrm{w}^T\Phi(\mathrm{x}_n)\}\Phi(\mathrm{x}_n)^T$$

expanded it, we have: $\mathrm{w}^T(\sum_{n=1}^{N}\Phi(\mathrm{x}_n)\Phi(\mathrm{x}_n)^T) = \sum_{n=1}^{N} y_n\Phi(\mathrm{x}_n)^T$,

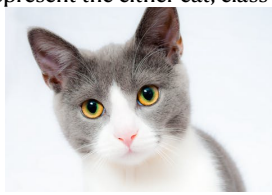
one more step,

$$\mathrm{w} = \frac{\sum_{n=1}^{N} y_n\Phi(\mathrm{x}_n)^T}{(\sum_{n=1}^{N}\Phi(\mathrm{x}_n)\Phi(\mathrm{x}_n)^T)} = (\sum_{n=1}^{N}\Phi(\mathrm{x}_n)\Phi(\mathrm{x}_n)^T)^{-1}\sum_{n=1}^{N} y_n\Phi(\mathrm{x}_n)^T,$$

A compact expression: $\mathrm{w}_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\mathrm{y}$, also called ***Normal Equations*** when $\Phi^T\Phi$, where Φ is called *design matrix* (also the basis function of x)

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

C1: Cat

C2 Dog

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

- Decision boundaries or decision surfaces are defined as boundaries that partition the input space (vector space) into regions, one for each class.
- The decision surfaces correspond to $y(\mathrm{x}, \mathrm{w}) = const = \mathrm{x}^T\mathrm{w} + w_0$.
- 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 $p(C_k \mid t = 1)$.
- 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:
  - **discriminant function**—directly maps each input vector to a specific class.
  - ***conditional probability distribution*** $p(C_k \mid \mathrm{x})$ with a **discriminative** approach.
    - model $p(C_k \mid \mathrm{x})$,
    - e.g., logistic regression
  - ***class conditional densities*** $p(\mathrm{x} \mid C_k)$ together with the class prior probabilities $p(C_k)$. Then, infer **posterior** probability using **Bayes' rule**:
    - $p(C_k \mid \mathrm{x}) = \dfrac{p(\mathrm{x} \mid C_k)p(C_k)}{p(\mathrm{x})}$,
    - e.g., fit multivariate Gaussians to the input vectors.

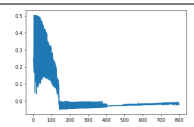
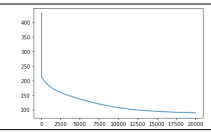
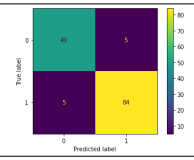
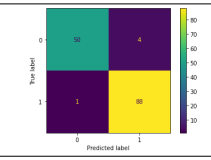
Least Squares Loss for Classification

- 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 $E[t \mid \mathrm{x}]$.
- 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

- Given a dataset $\{\mathrm{x}_n, \mathrm{t}_n\}, n = 1,\ldots,N$.
- Based on the **normal equation** and using some matrix algebra or gradient of least mean square algorithm (like our A2), 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.
  - $W_{t+1} = W_t - \eta(\hat{y} - y)X, \ \ \forall t \in N$, where $dW = (\hat{y} - y)X$, and η is the learning-rate .
- 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 and local optimum issues
- Usually, using logistic regression or Fisher's Linear Discriminant to solve this issue.
- Here, let's focus on logistic regression.

Performance comparison of Least Squares Loss vs. Logistic Regression

Least Mean Square Loss/Gradient	Cross-entropy/Logistic regression	
		loss curve
precision, recall, f2, accuracy (0.9438202247191011, 0.9438202247191011, 0.9438202247191011, 0.9300699300699301)	precision, recall, f1, accuracy (0.9565217391304348, 0.9887640449438202, 0.9723756906077348, 0.9650349650349965)	P, R, F1
		Confusion Matrix

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: $\nabla E_{\mathbf{w}_j}(\mathrm{w}_1, \ldots, \mathrm{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\mathrm{x}_n,$

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.

Probabilistic Generative Models

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

$$p(C_k \mid \mathrm{x}) = \frac{p(\mathrm{x} \mid C_k)p(C_k)}{p(\mathrm{x})}.$$

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

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

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

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.

Probabilistic Generative Models

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

$$p(C_1 \mid \mathrm{x}) = \frac{p(\mathrm{x} \mid C_1)p(C_1)}{p(\mathrm{x} \mid C_1)p(C_1) + p(\mathrm{x} \mid C_2)p(C_2)} = \frac{1}{1 + exp(-a)} = \sigma(a),$$ 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.

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.

Naive Bayes - a Probabilistic Generative Model

- A naive Bayes classifier is a simple probabilistic generative model.
- The goal is to use a generative approach to classify vectors of discrete values features, $x \in \{1,\ldots,K\}^D$, where,
  - K — the number of values for each feature, and
  - D — the number of features.
- The class conditional distribution is given as p(x|y=c).
- The simplest approach is to assume the features are conditional independent given the class label, which is $p(x \mid y = c, W) = \prod_{j=1}^{D} p(x_j \mid y = c, W_{jc})$

Naive Bayes - a Probabilistic Generative Model

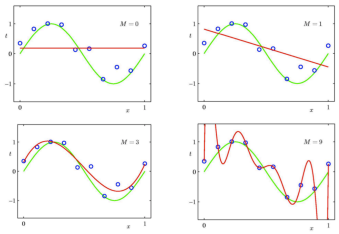
- A naive Bayes classifier for a binary classification problem:
- $x_j \in \{0,1\}$, use Bernoulli distribution: $p(x \mid y = c, W) = \prod_{j=1}^{D} Ber(x_j \mid \mu_{jc})$, where μ_{jc} is the mean of feature j in objects of class.
- It is also called the multivariate Bernoulli naive Bayes model.
- To fit the model, one can calculate the MLE or the MAP estimate the parameters with the posterior P(w|D).
  - The prior is given as: $\pi_c = \frac{N_c}{N}$, where $N_c = \sum_i I(y_i = c)$, the number of examples in class c.
  - The parameter is given as: $\hat{W}_{jc} = \frac{N_{jc}}{N_c}$

Over-fitting and under-fitting problems

- Machine learning uses some approximation approaches to estimate the parameters.
- This approximation simplifies the learning processes, but
- It brings a significant problem—over-fitting and under-fitting, particular the over-fitting problem.
- The over-fitting is about the trained model *perfectly* matches the training data.
- In other words, the trained model has *memorized* the details of training data, even any noise signals in the training data.
- The consequence is that the trained model performs very poorly in predictions of new data, which the model never sees before.
- A serious over-fitting error can make the model lose the predicative capability totally.

Over-fitting and under-fitting problems

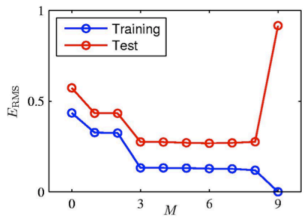
- Use the polynomial curve fitting from the text book, *Pattern Recognition and Machine Learning:*
- Some Fits to the Data

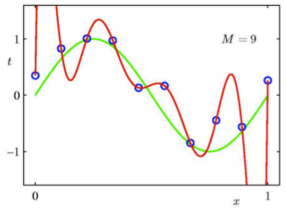


For M=9, we have fitted the training data perfectly.

Over-fitting and under-fitting problems

- Testing the model on 100 data points, which were sampled using the same procedure used for generating the training data.
-



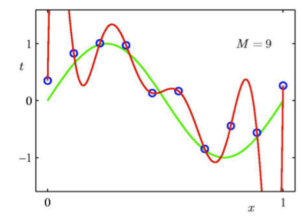


For M=9, the training error is zero! The parameters **w** can be fitted exactly to the data points. But the test error is huge, why?

Over-fitting and under-fitting problems

- As M increases, the magnitude of coefficient becomes larger.
- For M=9, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.
- The consequence is that more flexible polynomials with larger M tune to the random noise on the target values.

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



Over-fitting and under-fitting problems

- For a complex model (having many parameters), more training data can make over-fitting problem less serious.
- For a few of training data, a complex model is highly likely over-fitted.
- So, solutions to overcome the over-fitting problem are:
  1. using more training data.
  2. simplify the model, but this is not work for most of cases since a complex problem needs a complex model.

Thus, a more general solution is needed: **Generalization/regularization**

Generalization/regularization

- The purpose of a good generalization is to let the model to make accurate predictions for new test data that is not known during learning.
- The cost function with regularization takes the form:

$$\hat{E}_D = E_D(\mathrm{w}) + \lambda E_W(\mathrm{w})$$

where λ is the *regularization coefficient* that controls the relative importance of the **data-dependent error** $E_D(\mathbf{w})$ and the regularization term $E_W(\mathrm{w})$.

- In the case of the sum-of-square error function, the above formula can take this form:

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - \mathrm{w}^T\phi(\mathrm{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$

Generalization/regularization

- In machine learning, when $q = 1$, we call the regularization as L1 or **lasso** regularization, if $q = 2$, it is called L2 or **weight decay** regularization. They are most common regularization methods.
- Thus, the error function with L1 regularization is:

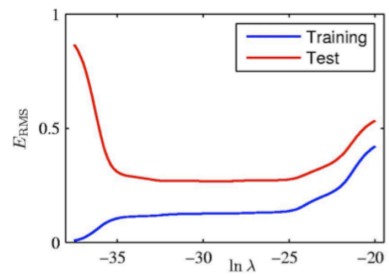
$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - \mathrm{w}^T\phi(\mathrm{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|$$

- and the error function with L2 regularization is:

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - \mathrm{w}^T\phi(\mathrm{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^2$$

Generalization/regularization

- After applying the regularization, the training and testing cost functions of the polynomial curve fitting model basically match to each other.



	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Support Vector Machine (SVM)

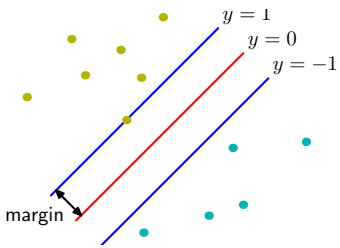
- In kernel methods, one of the significant limitations is that the kernel function must be evaluated for all possible pairs x_n and x_m of training points. This process is very computational expensive.
- SVM comes out a solution as a kernel-based algorithm that has sparse solutions—which means the kernel function evaluated at a subset of the training data points.
- An important property of SVM is that the determination of the model parameters corresponds to a convex optimization problem—which means there is a global optimum.
- SVM uses Lagrange multipliers as optimization constraints to optimize the parameters to find the global optimums.
- One of limitations of SVM is that it is a decision machine and so does not provide posterior probabilities.

Maximum Margin Classifiers (SVM)

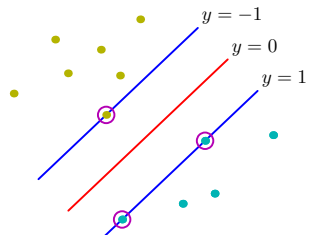
- For a binary classification, the linear model is given as: $y(x) = w^T\phi(x) + w_0$
- The classifier separates the data points based on $y(x_n) > 0, \forall x$, having $t_n = +1$ and $y(x_n) < 0, \forall x$, having $t_n = -1$, so $t_n y(x_n) > 0$ for all data points.
- The problem of this approach is that there are multiple ways to separate the data. We need to find the one that gives the smallest generalization error.
- SVM approaches this problem via a concept of the margin—defines the smallest distance between the decision boundary and any of the samples.
- To keep the generalization, SVM choses the decision boundary to be the one for which the margin is maximized.

Maximum Margin Classifiers (SVM)

- The diagram in the left side shows the margin is defined as the distance between decision boundary and the closest of the data points.
- The right side shows that maximizing the margin leads to a particular choice of decision boundary. The location of this boundary is determined by a subset of the data points, which known as ***support vectors***, indicated by the circles.



Margin



Support Vectors

Maximum Margin Classifiers (SVM)

- So, the intuition of the SVM approach is to find this subset of the data points, which known as ***support vectors***.
- Remember the Gaussian kernels having a covariance Σ or variance σ^2. In the limit σ^2 ->o, the optimal hyperplane is shown to be one having maximum margin.
- The intuition behind this result is that as σ^2 is reduced, the hyperplane is increased dominated by nearby data points relative to more distance ones. In the limit, the hyperplane becomes independent of data points that are not support vectors.
- Based on this, we find such small set of data points that can independently determine the decision boundary, such a subset of the data points are called ***support vectors***
- Other data points that do not belong to ***support vectors*** will not participate in the prediction process.

Maximum Margin Classifiers (SVM)

- The Lagrange multipliers $a_n \geq 0$ is used to find the optimum of parameters.
- $L(\mathrm{w}, b, \mathrm{a}) = \frac{1}{2}||\mathrm{w}||^2 - \sum_{n=1}^{N} a_n\{t_n(\mathrm{w}^T\Phi(\mathrm{x}_n) + b) - 1\}$,
- minimize L w.r.t w and b equal to zero, obtain following two conditions:
  - $\mathrm{w} = \sum_{n=1}^{N} a_n t_n \Phi(\mathrm{x}_n)$
  - $0 = \sum_{n=1}^{N} a_n t_n$
- Substitute w and b from L(w, b, a) using these conditions then gives the dual representation of the maximum margin problem in which we maximize,
- $\hat{L}(\mathrm{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathrm{x}_n, \mathrm{x}_m)$, w.r.t **a** subject to the constraints

Maximum Margin Classifiers (SVM)

- $\hat{L}(\mathrm{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathrm{x}_n, \mathrm{x}_m)$, w.r.t **a** subject to the constraints
- $a_n \geq 0,\ \ n = 1,\ldots,N$, and $\sum_{n=1}^{N} a_n t_n = 0$. $. Here the kernel function is defined by $k(\mathrm{x}, \mathrm{x}') = \Phi(\mathrm{x})^T\Phi(\mathrm{x}')$.
- For binary classification $y(\mathrm{x}) = \mathrm{w}^T\Phi(\mathrm{x}) + b$, the $y(\mathrm{x}) = \sum_{n=1}^{N} a_n t_n k(\mathrm{x}, \mathrm{x}_n) + b$, and the Karush-Kuhn-Tucker (KKT) condition gives the constrained optimization of this form:
  - $a_n \geq 0$
  - $t_n y(\mathrm{x}_n) - 1 \geq 0$
  - $a_n\{t_n y(\mathrm{x}_n) - 1\} = 0$

Maximum Margin Classifiers (SVM)

- Thus, for each data point, either $a_n = 0$ or $t_n y(\mathrm{x}_n) = 1$.
- Any data point for which $a_n = 0$ will not appear in the sum $y(\mathrm{x}) = \sum_{n=1}^{N} a_n t_n k(\mathrm{x}, \mathrm{x}_n) + b$ and hence plays no role in making predictions for new data points.
- The remaining data points are ***support vectors***, and they satisfy $t_n y(\mathrm{x}_n) = 1$, they correspond to points that lie on the maximum margin hyperplanes in feature space.
- This is the central property of the SVMs in practice. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors retained.

Introduction to Neural Networks

- Neural Networks can work with both regression and classification problems.
- Its main components are:
  - the forward propagation algorithm, and
  - cost function,
  - the network training, also
  - the backpropagation algorithm

Basic Concepts

- The purpose is to find useful representation of the target variables:
  - $\mathrm{t} = \mathrm{y}(\mathrm{x}, \mathrm{w}) + \epsilon(\mathrm{x})$, where
    - $\mathrm{t} = (t_1, \ldots, t_N)$ and $\mathrm{x} = (x_1, \ldots, x_N)^T$ are the observations.
    - $\epsilon(\mathrm{x})$ is the residual error.
- For example, to a linear model:
  - $y(\mathrm{x}, \mathrm{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathrm{x})\right)$
    - $\phi = (\phi_0, \ldots, \phi_M)^T$ is the fixed model basis functions.
    - $\mathrm{w} = (w_0, \ldots, w_M)^T$ are the model parameters—also called coefficients.
    - For regression: f(.) is the identity function.
    - For classification: f(.) is an non-linear activate function.

Basic Concepts — Feed-forward Neural Networks

- Feed-forward Neural Networks generalize the linear model:
  - $y(\mathrm{x}, \mathrm{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathrm{x})\right)$, where
    - the goal of feed-forward is to let the basis itself, as well as the coefficients w_j, will be adapted.
    - In other words, make the basis functions depend on the parameters.
    - The network uses the same form of the basis function
    - The basis function is a non-linear function of a linear combination of the inputs.

Basic Concepts — Feed-forward Neural Networks

- First, construct M linear combinations of the input variables $x_1, \ldots, x_D$ in the form:
  - $a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$, where
    - a_j are the activations, $j = 1,\ldots,M$.
    - $w_{ji}^{(1)}$ are weights for layer 1, where $i = 1,\ldots,D$.
    - $w_{j0}^{(1)}$ are the biases for the layer 1.
    - Each linear combination a_j is transformed by a (nonlinear differentiable) activation function:
      - $z_j = h(a_j)$

Basic Concepts — Feed-forward Neural Networks

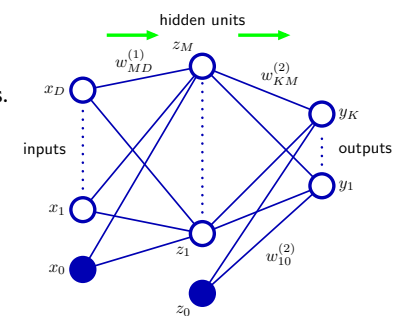
- The output activations of the hidden layer $z_j = h(a_j)$ are linearly combined in layer two:
  - $a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$, where
    - a_k are the output activations, $k = 1,\ldots,K$.
    - $w_{kj}^{(2)}$ are weights for layer 2, where $j = 1,\ldots,D$.
    - $w_{k0}^{(2)}$ are the biases for the layer 2.
    - The output activations a_k are transformed by output activation function:
      - $y_k = \sigma(a_k)$
        - y_k are the final outputs.
        - $\sigma(a)$ is a sigmoidal function (for binary classification)

Basic Concepts — Feed-forward Neural Networks

- The complete two layer model:
  - $y_k(\mathrm{x}, \mathrm{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$, where
    - $h(\,.\,)$ is the basis or activation function and $\sigma(a)$ are sigmoidal functions, e.g., the logistic function.
    - Again, for regression, the $\sigma(a)$ becomes to the identity.
- Absorb the biases $w_{k0}^{(2)}$ and $w_{j0}^{(1)}$ into the weight sets, we get the compact form:
  - $y_k(\mathrm{x}, \mathrm{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right) \right)$
- Evaluation of the above model (network) is called forward propagation.

Basic Concepts — Feed-forward Neural Networks

- This two-layer network diagram is given as right Figure.
- The approximation process can be represented by a network:
  - Nodes are input, hidden and output units. Links are corresponding weights.
  - Information propagates 'forwards' from the explanatory variable x to the estimated response $y_k(\mathrm{x}, \mathrm{w})$.



Basic Concepts — Feed-forward Neural Networks

- The Properties and generalizations:
  - Normally $K \leq D \leq M$, which means that the network is redundant if all $h(\cdot)$ are linear.
  - There may be more than one layer of hidden units.
  - Individual units need not be fully connected to the next layer.
  - Individual links may skip over one or more subsequent layers.
  - Networks with two or more layers are universal approximations.
  - Any continuous function can be uniformly approximated to arbitrary accuracy, given enough hidden units.
  - This is true for many definitions of $h(\cdot)$, but excluding polynomials.
  - There may be symmetries in the weight space, meaning that different choices of w may define the same mapping from input to output.

Basic Concepts — Feed-forward Neural Networks

- Maximum Likelihood Parameters:
  - Maximum likelihood is the same as minimizing the residual error between $y_k(\mathrm{x}, \mathrm{w})$ and t_n.
  - Let the target be a scalar-valued function, which is Normally distributed around the estimate:
    - $p(t \mid \mathrm{x}, \mathrm{w}) = \mathbb{N}(t \mid y(\mathrm{x}, \mathrm{w}), \beta^{-1})$
  - Consider the sum of squared-errors: $E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(\mathrm{x}_n, \mathrm{w}) - t_n)^2$
  - The maximum-likelihood estimate of w can be obtained by (numerical) minimization:
    - $\mathrm{w}_{ML} = \min_w E(\mathrm{w})$
    - After get the w_{ML}, the precision, β can also be estimated. E.g. if the N observations are i.i.d. (Independent and identically distributed random variables), then their joint probability is:
      $p(\mathrm{t} \mid \mathrm{x}, \mathrm{w}, \beta) = \prod_{n=1}^{N} p(t_n \mid \mathrm{x}_n, \mathrm{w}, \beta)$

Basic Concepts — Feed-forward Neural Networks

- Maximum Likelihood Parameters:
  - The negative log-likelihood, in this case, is:
    - $-\log p(\mathrm{t} \mid \mathrm{x}, \mathrm{w}, \beta) = \beta E(\mathrm{w}_{ML}) - \frac{N}{2}\log\beta + \frac{N}{2}\log 2\pi$
  - By obtain the derivative $d/d\beta = E(\mathrm{w}_{ML}) - \frac{N}{2\beta}$, we have:
    - $\frac{1}{\beta_{ML}} = \frac{2}{N}E(\mathrm{w}_{ML})$
  - For $K > 2$ target variables, $\frac{1}{\beta_{ML}} = \frac{2}{NK}E(\mathrm{w}_{ML})$

Basic Concepts — Feed-forward Neural Networks

- Parameter Optimization:
  - Iterative search for a local minimum of the error:
    - $\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} + \Delta\mathrm{w}^{(\tau)}$
  - The local minimum is based on $\nabla E = 0$ at a minimum of the error.
  - τ is the time-step or iteration step.
  - $\Delta\mathrm{w}^{(\tau)}$ is the weight update.

Basic Concepts — Feed-forward Neural Networks

- To optimize the parameters, an approximation approach, namely local quadratic approximation is applied here:
  - $E(\mathrm{w}) \approx E(\hat{\mathrm{w}}) + (\mathrm{w} - \hat{\mathrm{w}})^T \mathrm{b} + \frac{1}{2}(\mathrm{w} - \hat{\mathrm{w}})^T \mathrm{H}(\mathrm{w} - \hat{\mathrm{w}})$
- $\mathrm{b} = \nabla E|_{w=\hat{w}}$ is the gradient at $\hat{w}$.
- $(\mathrm{H})_{ij} = \frac{\partial E}{\partial w_i \partial w_j}|_{w=\hat{w}}$ is the Hessian $\nabla\nabla E$ at $\hat{w}$
- if $\mathrm{w} \approx \hat{\mathrm{w}}$ then $\nabla E \approx \mathrm{b} + \mathrm{H}(\mathrm{w} - \hat{\mathrm{w}})$.
- Let w^* is at the minimum of E. so $\mathrm{b} = \nabla E|_{w=w^*} = 0$. then
  - $E(\mathrm{w}) = E(\mathrm{w}^*) + \frac{1}{2}(\mathrm{w} - \mathrm{w}^*)^T \mathrm{H}(\mathrm{w} - \mathrm{w}^*)$

Basic Concepts — Feed-forward Neural Networks

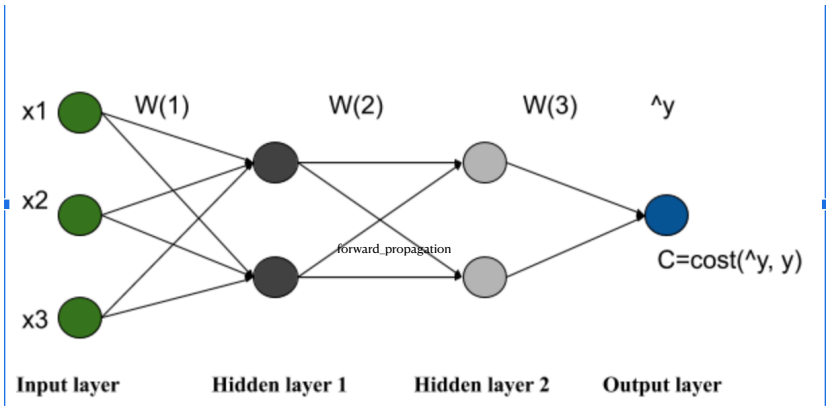
- Let w^* is at the minimum of E. so $\mathrm{b} = \nabla E|_{w=w^*} = 0$. then
  - $E(\mathrm{w}) = E(\mathrm{w}^*) + \frac{1}{2}(\mathrm{w} - \mathrm{w}^*)^T \mathrm{H}(\mathrm{w} - \mathrm{w}^*)$
  - where $\mathrm{H} = \nabla\nabla E|_{w=w^*}$ is the Hessian.
  - the eigenvectors $\mathrm{H}\mathrm{u}_i = \lambda \mathrm{u}_i$ are orthonormal.
  - $(\mathrm{w} - \mathrm{w}^*) = \sum_i \alpha_i \mathrm{u}_i$
  - Here we have: $\frac{1}{2}(\mathrm{w} - \mathrm{w}^*)^T \mathrm{H}(\mathrm{w} - \mathrm{w}^*) = \frac{1}{2}(\sum_i \lambda_i \alpha_i \mathrm{u}_i)^T (\sum_j \alpha_j \mathrm{u}_j)$
  - $E(\mathrm{w}) = E(\mathrm{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$ because $\mathrm{u}_i^T \mathrm{u}_j = \mathrm{I}$, where I is identity matrix.

Basic Concepts — Feed-forward Neural Networks

- Gradient Descent (GD)
  - The simplest approach is to update w by a displacement in the negative gradient direction.
    - $\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} - \eta \nabla E(\mathrm{w}^{(\tau)})$
  - This is a steepest descent algorithm.
  - η is the learning rate.
  - This is a batch method, as evaluation of ∇E involves the entire data set.
  - Conjugate gradient or quasi-Newton methods may, in practice, be preferred.
  - A range of starting points $\{w^{(0)}\}$ may be needed, in order to find a satisfactory minimum.

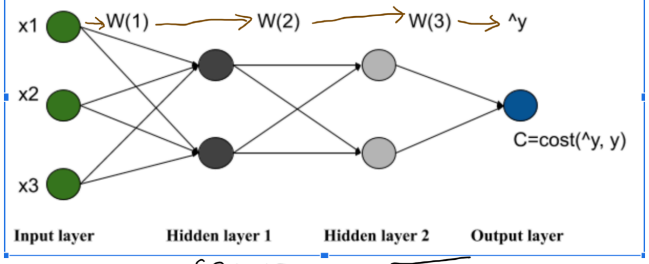
Basic Concepts — Feed-forward Neural Networks

- Optimization scheme:
  - Each iteration of the descent algorithm has two stages:
    1. Evaluate derivatives of error with respect to weights (involving backpropagation of error though the network).
    2. Use derivatives to compute adjustments of the weights (e.g. steepest descent).This is a batch method, as evaluation of ∇E involves the entire data set.
  - Backpropagation is a general principle, which can be applied to many types of network and error function.



Forward propagation

$\hat{y} = W^{(3)} f(W^{(2)} f(W^{(1)} x + b^{(1)}) + b^{(2)})$



$C = \text{cost}(\hat{y}, y)$ ① MSE: $C = \sqrt{(\hat{y} - y)^2}$

② cross-entropy: $C = H(\hat{y}, y) = -\sum p(y) \log q(\hat{y})$

Basic Concepts —Backpropagation Neural Networks

- Backpropagation "repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the neural network and the desired output vector."[3]
- Chain rule: $\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l}$
- m - number of neurons in l-1 layer: $z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^1$
- by differentiation (calculating derivative): $\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$
- the final value, $\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1}$
- Reference: Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

Recap

- Linear regression is a type of supervised learning
- The training data consists of x and y, the labeled data, and output y is real-valued scalar or continuous
- The **cost function** is defined as the difference between expected values and actual values of outputs.
- The most common cost function is the **least squares error** function, also called the **sum-of-squares error**.
- 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** $p(\mathrm{x} \mid C_k)$ 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.

Recap

- Logistic regression is a classification approach. For two-class case, it is a sigmoid function, for multi class case, it is a softmax function.
- The over-fitting is about the trained model *perfectly* matches the training data.
- Regularization is the approach to solve the over-fitting problem.
- SVM comes out a solution as a kernel-based algorithm that has sparse solutions—which means the kernel function evaluated at a subset of the training data points.
- SVM maximizes the margin leads to a particular choice of decision boundary. The location of this boundary is determined by a subset of the data points, which known as ***support vectors***. That is SVM name comes from.

Recap

- There is a need for an alternative approach to overcome the disadvantages of SVMs and other parameter based models. Here is the Neural Network.
- The intuition comes out the feed-forward neural network, also known as the multilayer perceptron.
- The term 'neural network' came from biological systems. Machine learning focus on neural networks as efficient models for statistical pattern recognition.
- Feed-forward Neural Networks generalize the linear model.

Recap

- construct M linear combinations of the input variables $x_1, \ldots, x_D$ in the form: $a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Each linear combination a_j is transformed by a (nonlinear differentiable) activation function: $z_j = h(a_j)$
- The output activations of the hidden layer $z_j = h(a_j)$ are linearly combined in layer two.
- The complete two layer model: $y_k(\mathrm{x}, \mathrm{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \right) \right)$

Recap

- Evaluation of the feed-forward network is called forward propagation.
- Gradient Descent (GD) is the simplest approach is to update w by a displacement in the negative gradient direction.
- To optimize the parameters, evaluate derivatives of error with respect to weights (involving backpropagation of error though the network).
- Backpropagation is the key algorithm in neural network. It uses chain rules to compute gradients of cost function over weights and biases.
- There are three main processes in neural network: Forward propagation, cost function, and Backpropagation.
- The principals of the learning in neural network is about summation of information, non-linearly transformer the summation, re-allocation of weights of all neurones by adjusting the weights from the differentiate errors over weight parameters.

Assignment 3

- Assignment 3 worth 15%, and is about a classification Python programming using Scikit-learn framework. 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 Mar 19th@23:59 (your local time), 2021. It is an individual assignment. Please no late due. Any late due assignment will not be accepted.**
- **Make sure your share your Colab notebook having proper access permission to me to review your work.**
- **If I cannot view your work due to the permission issue, I will send you an email to remind you to re-assign me correct access permissions to your Colab notebook. After 12 hours start from the time that I sent you my reminding email, if I still cannot access your Colab notebook, no evaluation for this assignment will be given.**
- **I will start evaluating your work at Mar 20th, and try my best to give you feedback 1 week after.**

Questions?