① Derive the gradient descent training rule assuming that the target function representation is:-

$$O_d = \omega_0 + \omega_1 x_1 + \dots \omega_n x_n$$

Define explicitly the cost function E, assuming that a set of training examples D is provided, where each training examples $d \in D$ is associated with the target output t_d.

Soln:-

The cost/error function $E = \sum_{d \in D} (t_d - O_d)^2$.

We know,

From gradient decent algorithm.

$$\Delta\omega_i = -\alpha \frac{dE}{d\omega_i}$$ (where α is learning rate)

Now,

Represent $dE/d\omega_i$ in terms of the input units x_{id}, outputs O_d and target values t_d

So,

$$\frac{dE}{d\omega_i} = \frac{d\sum_{d \in D}(t_d - O_d)^2}{d\omega_i} = \sum_{d \in D} 2(t_d - O_d) \times \frac{d(t_d - O_d)}{d\omega_i}$$

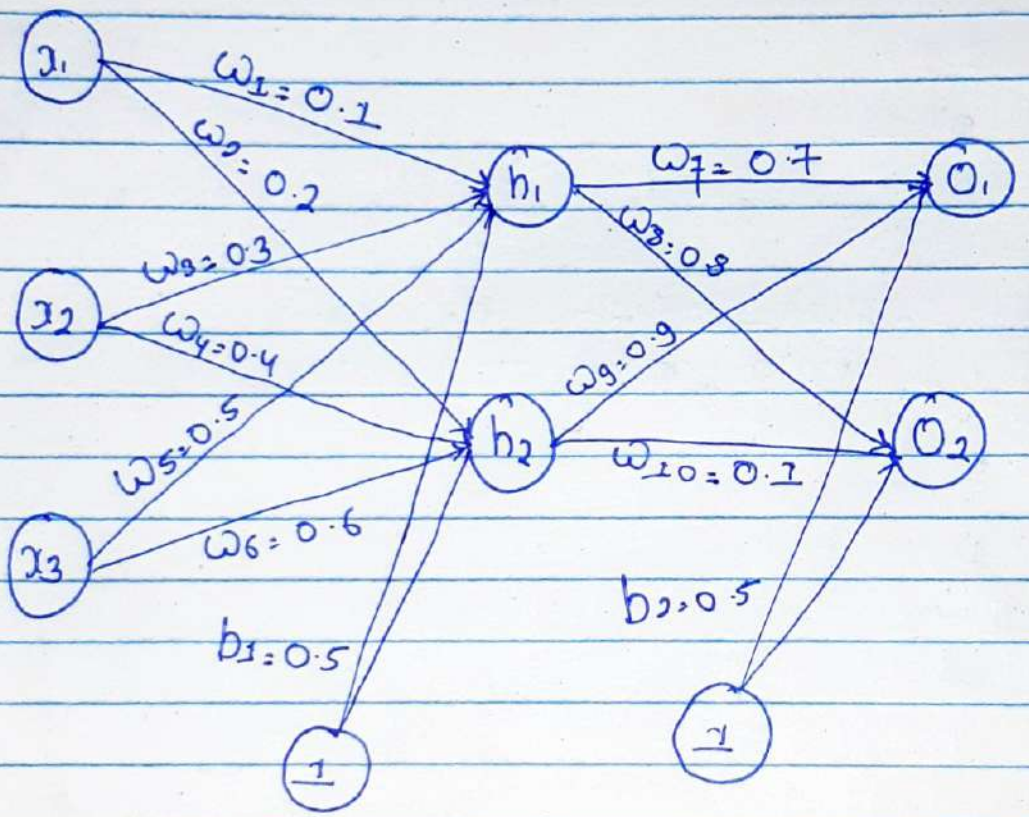
$$= \sum_{d \in D} 2(t_d - O_d) \times \left(-\frac{dO_d}{d\omega_i}\right)$$

$$= -\sum_{d \in D} 2(t_d - O_d) \times \frac{d(\omega_0 + \dots + \omega_i x_{id} + \dots + \omega_n x_{nd})}{d\omega_i}$$

$$= -\sum_{d \in D} 2(t_d - O_d)\, x_{id}$$

$$\therefore \Delta\omega_i = \alpha \sum_{d \in D} 2(t_d - O_d) x_{id}.$$

2. The following network is designed for binary classification with 3 input parameters. $x_1 = 1$, $x_2 = 4$ and $x_3 = 5$ and target output as $O_1 = 0.1$ and $O_2 = 0.05$. The network is trained using standard error back propagation, i.e. the square error should minimized using gradient search. Now, forward propagate through the network to determine the error derivate and update w_i.



Solution:-

Given,

Input parameter.

$x_1 = 1$

$x_2 = 4$

$x_3 = 5$

Target output.

$O_1 = 0.1$

$O_2 = 0.05$

Weights.

$w_1 = 0.1$
$w_2 = 0.2$
$w_3 = 0.3$
$w_4 = 0.4$
$w_5 = 0.5$
$w_6 = 0.6$
$w_7 = 0.7$
$w_8 = 0.8$
$w_9 = 0.9$
$w_{10} = 0.1$

Bios.

$b_1 = 0.5$
$b_2 = 0.5$

Now,

$h_1 = x_1 w_1 + x_2 w_3 + x_3 w_5 + b_1$
$= 1 \times 0.1 + 4 \times 0.3 + 5 \times 0.5 + 0.5$
$= 4.3$

So,

$Out h_1 = \frac{1}{1 + e^{-h_1}}$
$= \frac{1}{1 + e^{-4.3}}$
$= 0.986$

Also,

$h_2 = x_1 w_2 + x_2 w_4 + x_3 w_6 + b_1$
$= 1 \times 0.2 + 4 \times 0.4 + 5 \times 0.6 + 0.5$
$= 5.3$

So,

$Out h_2 = \frac{1}{1 + e^{-h_2}} = \frac{1}{1 + e^{-5.3}} = 0.995$

Again

$O_1 = out_{h_1} \times w_7 + out_{h_2} \times w_9 + b_2$

$= 0.986 \times 0.7 + 0.995 \times 0.9 + 0.5$

$= 2.085$

So,

$out_{O_1} = \frac{1}{1+e^{-O_1}} = \frac{1}{1+e^{-2.085}} = 0.889$

Again

$O_2 = out_{h_1} \times w_8 + out_{h_2} \times w_{10} + b_2$

$= 0.986 \times 0.8 + 0.995 \times 0.1 + 0.5$

$= 1.388$

So,

$out_{O_2} = \frac{1}{1+e^{-O_2}} = \frac{1}{1+e^{-1.388}} = 0.800$

Now, we are using MSE to get total error.

$Error = \frac{1}{2}\{(T_1 - out_{O_1})^2 + (T_2 - out_{O_2})^2\}$

$= \frac{1}{2}\{(0.1 - 0.889)^2 + (0.05 - 0.800)^2\}$

$= 0.5925$

Now, our goal with back propagation is to update each weight because the target ouput (t_1) is 0.1 but neural network predict output as 0.889 and similary target (t_2) and out_{y_2}.

Weight update for get new weight of w_1.

$$New(w_1) = Old(w_1) - \alpha \frac{dE}{dw_1}$$

where, α is the learning rate.

So,

$$\frac{dE}{dw_1} = \frac{dE}{dout_{y_1}} \times \frac{dout_{y_1}}{dy_1} \times \frac{dy_1}{dout_{h_1}} \times \frac{dout_{h_1}}{dh_1} \times \frac{dh_1}{dw_1}$$

Now, we need to figure out each piece in equation.

$$\frac{dE}{dout_{y_1}} = \frac{d\left\{\frac{1}{2}(T_1 - out_{y_1})^2 + (T_2 - out_{y_2})^2\right\}}{dout_{y_1}}$$

$$= \frac{1}{2}\left[\frac{d\{(T_1 - out_{y_1})^2}{dout_{y_1}} + \frac{d(T_2 - out_{y_2})^2}{dout_{y_1}}\right]$$

$$= \frac{1}{2}\left[\frac{d(T_1 - out_{y_1})^2}{d(T_1 - out_{y_1})} \times \frac{d(T_1 - out_{y_1})}{dout_{y_1}}\right] + 0$$

$$= \frac{1}{2} \times 2(T_1 - out_{y_1}) \times (-1)$$

$$= out_{y_1} - T_1$$

$$\frac{d(out_{y_1})}{dy_1} = \frac{d\left\{\frac{1}{1+e^{-y_1}}\right\}}{dy_1}$$

$$= out_{y_1}(1 - out_{y_1}) \quad (\because \text{derivative of Sigmoid function})$$

$$\frac{dy_1}{douth_2} = \frac{d(outh_2 \times w_7 + outh_2 \times w_9 + b_2)}{douth_2}$$

$$= \frac{d(outh_1 \times w_7)}{douth_2} + \frac{douth_2 \times w_9}{douth_1} + \frac{db_2}{douth_1}$$

$$= outh_1 \times \frac{dw_7}{douth_1} + w_7 \frac{douth_1}{douth_1} + 0 + 0$$

$$= 0 + w_7$$

$$= w_7$$

$$\frac{douth_1}{dh_1} = \frac{d\left\{\frac{1}{1+e^{-h_1}}\right\}}{dh_1}$$

$$= outh_1 (1 - outh_1) \quad (\because \text{derivative of Sigmoid function})$$

$$\frac{dh_1}{dw_1} = \frac{d(x_1 w_1 + x_2 w_3 + x_3 w_5 + b_1)}{dw_1}$$

$$= \frac{d(x_1 w_1)}{dw_1} + \frac{d(x_2 w_3)}{dw_1} + \frac{d(x_3 w_5)}{dw_1} + \frac{db_1}{dw_1}$$

$$= x_1 \frac{dw_1}{dw_1} + w_1 \frac{dx_1}{dw_1} + 0 + 0 + 0$$

$$= x_1 \times 1 + w_1 \times 0$$

$$= x_1$$

So,

$$\frac{dE}{dw_1} = (outy_1 - T_1) + outy_1 (1 - outy_1) \times w_7 \times outh_1 (1 - outh_1) \times x_1$$

$$= (0.889 - 0.1) + \{0.889 (1 - 0.889) \times 0.7 \times 0.986 (1 - 0.986) \times 1$$

$$= 0.789 \times 0.098 \times 0.7 \times 0.013 \times 1$$

$$= 0.00073$$

$$\therefore \frac{d\varepsilon}{dw_1} = 0.00073$$

So,

$$New(w_1) = Old(w_1) - 0.5 \times 0.00073$$

$$= 0.099$$

3. Why weighted word are important in Natural Language processing. Describe TF-IDF with a suitable example.

→ Let's try to understand a basic facts first:

a. Machine doesn't understand characters, words, or sentences.

b. Machine can only process numbers.

c. Text data must be encoded as numbers for input or output for any machine.

As mentioned in the above points we cannot pass raw text into machines as input until we convert them into numbers, hence we need to perform text encoding. Text encoding is a process to convert meaningful text into number/vector representation so as to preserve the context and relationship between words and sentences. Such that a machine can understand the pattern associated with any text and can make out the context of sentences.

So, weighted word techniques such as Bag of word (BOW) and TF-IDF come in. This weighted word doesn't hold semantic relation between the word.

Bag of Word (BOW)

The Bag of Word is the simplest form of text representation in numbers. Like the term itself, we can represent a sentence as bag of words vector (a string of numbers).

Let's take an example:-

1. This movie is very scary and long
2. This movie is not scary and is slow
3. This movie is spooky and good.

First we will build a vocabulary from all the unique words in the above three movie reviews. The vocabulary consists of these 11 words.

'This', 'movie', 'is', 'scary', 'and', 'long', 'not', 'slow', 'spooky', 'good'.

	1	2	3	4	5	6	7	8	9	10	11	Length of the review (in words)
	This	movie	is	very	scary	and	long	not	slow	spooky	good	
1	1	1	1	1	1	1	1	0	0	0	0	7
2	1	1	2	0	1	1	0	1	1	0	0	8
3	1	1	1	0	0	1	0	0	0	1	1	6

Vector of Review 1: [1 1 1 1 1 1 1 0 0 0 0]
Vector of Review 2: [1 1 2 0 1 1 0 1 1 0 0]
Vector of Review 3: [1 1 1 0 0 1 0 0 0 1 1]

That's the core idea behind bag of word.

Drawbacks of using a Bag-of-words.

1. If the new sentences contain new words, then our vocabulary size would increase and thereby, the

the length of vector coould increase too.

② Additionally, the vectors would also contain many Os, thereby resulting in a sparse matrix (which is what we would like to avoid)

3. We are retaining no information on the grammar of the sentences nor on the ordering of the words in the text.

To overcome the drawback of BOW, TF-IDF come in.

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

Term Frequency (TF) :- It is a measure of how frequently a term, t, appears in a document, d.

$$tf_{t,d} = \frac{n_{t,d}}{\text{Number of terms in the document.}}$$

Here, in the numerator, n is the number of times the term 't' appears in the document 'd'. Thus, each document and term would have its own TF value.

Here

Vocabulary: 'This', 'movie', 'is', 'very', 'scary', 'and', 'long', 'not', 'slow', 'spooky', 'good'.

Number of word in review 1 = 7

Number of word in review 2 = 8

Number of word in review 3 = 6

Example:

TF of the word 'this' in review 2 = $\frac{\text{number of times 'this' appears in review 2}}{\text{number of terms in review 2}}$

$= 1/8$

We can calculate the term frequencies for all the terms and all the reviews in this number.

Term	Review 1	Review 2	Review 3	TF Review 1	TF Review 2	TF Review 3
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0/7=0	1/8	0
slow	0	1	0	0/7=0	1/8	0
spooky	0	0	1	0/7=0	0	1/6
good	0	0	1	0/7=0	0	1/6

Inverse Document Frequency (IDF)

→ IDF is a measure of how important a term is. We need the IDF value because just the TF alone is not sufficient to understand the importance of words.

$idf_i = \log \frac{\text{Number of documents}}{\text{Number of documents with term 'i'}}$

We can calculate the IDF values of 'word' this in Review 2.

IDF 'this' in review 2 = log (Number of documents / Number of documents containing the word 'this')

$$= \log\left(\frac{3}{3}\right) = \log(1) = 0$$

The IDF values of the entire vocabulary would be:-

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	log(3/3) = 0.00
movie	1	1	1	log(3/3) = 0.00
is	1	2	1	log(3/3), log(3,3) = 0.00
very	1	0	0	log(3/1) = 0.48
Scary	1	1	0	log(3/2) = 0.18
and	1	1	1	log(3/3) = 0.00
long	1	0	0	log(3/1) = 0.48
not	0	1	0	log(3/1) = 0.48
slow	0	1	0	log(3/1) = 0.48
spooky	0	0	1	log(3/1) = 0.48
good	0	0	1	log(3/1) = 0.48

Hence, we can see that words like 'is', 'the', 'and' etc, are reduced to 0 and have little importance; while words like 'scary', 'long', 'good', etc., are the words with more importance and thus have higher value.

We can now compute the tf.IDF score of each word in corpus.

$$(tf\_idf)_{t,d} = tf_{td} \times idf_t$$

Term	Review 1	Review 2	Review 3	IDF	TF-IDF Review 1	TF-IDF Review 2	TF-IDF Review 3
This	1	1	1	0.00	0.000	0.000	0.000
movie	1	1	1	0.00	0.000	0.000	0.000
is	1	2	1	0.00	0.000	0.000	0.000
very	1	0	0	0.48	0.068	0.000	0.000
scary	1	1	0	0.18	0.025	0.022	0.000
and	1	1	1	0.00	0.000	0.000	0.000
long	1	0	0	0.48	0.068	0.000	0.000
not	0	1	0	0.48	0.000	0.060	0.000
slow	0	1	0	0.48	0.000	0.060	0.000
spooky	0	0	1	0.48	0.000	0.000	0.080
good	0	0	1	0.48	0.000	0.000	0.080

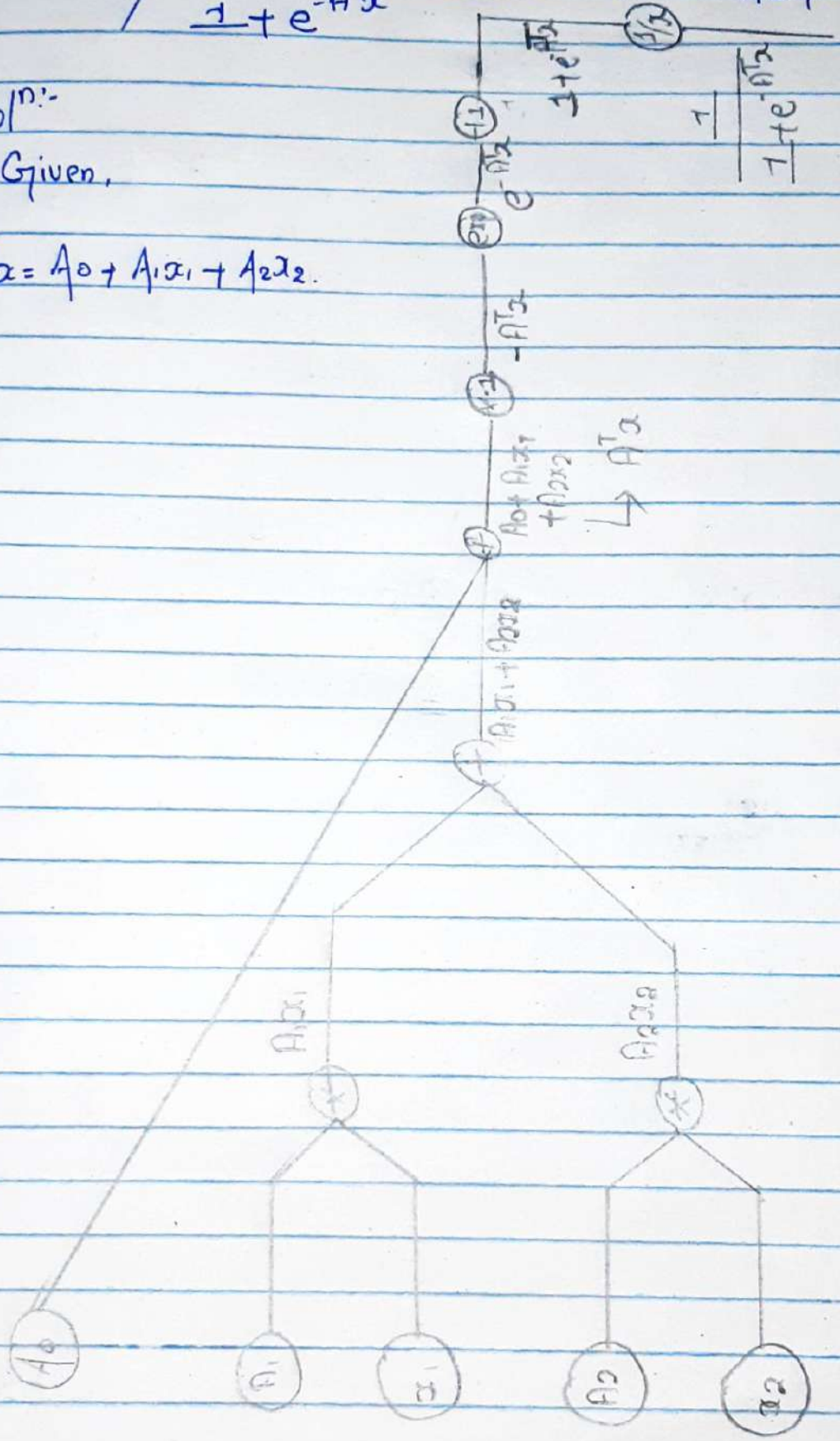
We have now obtained the TF_IDF scores for our vocabularity. TF_IDF gives higher (larger) values for less ~~important~~ frequent words and is high when both IDF and TF values are high i.e. the word is rare in all the documents combined but the frequent in a single document.

4. Draw a detailed computational graph of the sigmoid function. c/: $\frac{1}{1+e^{-A^T x}}$ where $A^T x = A_0 + A_1 x_1 + A_2 x_2$.

Sol^n:-

Given,

$A^T x = A_0 + A_1 x_1 + A_2 x_2$.



5. From a given 3 by 3 confusion matrix, calculate the following

confusion Matrix		Actual			
		A	O	M	Support
predicted	A	7	8	9	24
	O	1	2	3	6
	M	3	2	1	6
		11	12	13	Total = 36

a. Accuracy $= \frac{7+2+1}{36} = 0.2778$

(i) True positive rate (Recall of class A)
$= \frac{7}{11} = 0.63$

(ii) True positive rate (Recall of class O)
$= \frac{2}{12} = 0.1667$

(iii) True positive rate (Recall of class M)
$= \frac{1}{13} = 0.076$

(iv) Positive predictive value (Precision of class A)
$= \frac{7}{24} = 0.291$

(v) Positive predictive value (Precision of class O)
$= \frac{2}{6} = 0.333$

(vi) Positive predictive value (Precision of class M)
$= \frac{1}{6} = 0.1667$

b. F_1 Score.

(i) F_1 Score of class A $= \frac{2 \times R_A \times P_A}{R_A + P_A}$

$= \frac{2 \times 0.63 \times 0.291}{0.63 * 0.291}$

$= 0.398$

(ii) F_1-Score of class O $= \frac{2 \times R_O * P_O}{R_O + P_O}$

$= \frac{2 \times 0.1667 * 0.333}{0.1667 + 0.333}$

$= 0.222$

(iii) F_1-Score of class M $= \frac{2 \times R_M * P_M}{R_M + P_M}$

$= \frac{2 \times 0.076 * 0.1667}{0.076 + 0.1667}$

$= 0.104$.

(iv) Support of Recall A = 11
(v) Support of Recall O = 12
vi Support of Recall M = 13
vii Support of Precision A = 24
viii Support of precision O = 6
IX Support of precision M = 6

C. Micro_average and Micro_average.

So,

Total TP = 7+2+0 = 10

Total FP = (8+9) + (1+3) + (3+2) = 26

Total FN = (1+3) + (8+2) + (9+1) = 26

$$\text{Micro Recall} = \frac{TP}{TP+FN} = \frac{10}{10+26} = 0.2778$$

$$\text{Micro Precision} = \frac{TP}{TP+FP} = \frac{10}{10+26} = 0.2778$$

$$\text{Micro F1 score} = \frac{2 \times \text{Micro Recall} \times \text{Micro Precision}}{\text{Micro Recall} + \text{Micro Precision}}$$

$$= 2 \times \frac{0.2778 \times 0.2778}{0.2778 + 0.2778}$$

$$= 0.2778$$

Hence

Micro Recall = Micro Precision = Micro F1 Score = Accuracy

$$\text{Macro Average for Precision} = \frac{0.291 + 0.333 + 0.1667}{3}$$

$$= 0.263$$

$$\text{Macro-Average for Recall} = \frac{0.63 + 0.1667 + 0.076}{3}$$

$$= 0.2909$$

$$\text{Macro-Average for F1-Score} = \frac{0.398 + 0.222 + 0.104}{3}$$

$$= 0.241.$$

d. Weighted Average

Weighted Average for Recall

$$= \frac{0.63 \times 11 + 0.1667 \times 12 + 0.076 \times 13}{36}$$

$$= 0.275$$

Weighted Average for Precision

$$= \frac{0.291 \times 24 + 0.333 \times 6 + 0.1667 \times 6}{36}$$

$$= 0.277.$$

Weighted Average for F1-Score

$$= 2 \times \frac{\text{Weighted Recall} \times \text{Weighted Precision}}{\text{Weighted Recall} + \text{Weighted Precision}}$$

$$= 2 \times \frac{0.275 \times 0.277}{0.275 + 0.277}$$

$$= 0.275$$

6. What is overfitting and why it is a problem? Give an example of a method to reduce the risk of overfitting. When is Ridge regression favorable over Lasso regression? Discuss

→ Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that if it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply on the new data and negatively impact the model ability to generalize.

Overfitting is more likely with nonparametric and nonlinear model that have more flexibility when learning a target function. As such, many non_para
- metric machine learning algorithms also include peremeters or techniques to limit and constrain how much detail the model learns.

For exampre, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detailed it has picked up.

Some of an example of a method to reduce the risk of overfitting are:-

a. Cross validation
b. Train with more data
c. Remove features
d. Early Stopping
e. Regularization
f. Ensembling
g. Dropout

According to the writer of ISLR Hastie, TibShirani, Lasso regression is used in the presence of few variables with medium/large size effect. And Ridge regression is used in the presence of many variables with small/medium-sized effects.

Conceptually, we can say both perameter shrinkage and variable collection are done by lasso regression whereas perameter shrinkage is done by ridge regression and ends up with all model coefficients. Ridge regression might be better choice in the presence of correlated variables. Even, in cases where the least square estimates have more varionce, ridge regression fits well. Therefore, it depends on the objective of our model.

7. How is best attribute selected in a decision tree? Select the root node from the sample data below:

Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
overcast	Hot	High	False	yes
Sunny	Mild	High	False	yes
Sunny	Cool	Normal	True	No
overcast	Cool	Normal	True	yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	yes
Sunny	Mild	Normal	False	yes
Rainy	Mild	Normal	True	yes
overcast	Mild	High	True	yes
overcast	Hot	Normal	False	yes
Sunny	Mild	High	True	No

→ A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continues value).

Steps involved for selecting best attribute in a decision tree

a. For each split, individually calculate the entropy (information gain) or Gini impurity, or Chi-Square value of each child node.
b. Calculated the entropy of each split as the weighted average entropy, or Gini impurity or Chi-Square of child nodes.

c. Select the split with lowest entropy, lowest value of Gini impurity, or higher chi-square value
d. until you achieve homogenous nodes, repeat step 1-3.

From above table.
Total Yes = 8 and Total No = 5

Overall entropy of table (T):

$$E(T) = -\frac{Yes}{Yes+No}\log_2\frac{Yes}{Yes+No} - \frac{No}{Yes+No}\log_2\frac{No}{Yes+No}$$

$$= \frac{-8}{8+5}\log_2\left(\frac{8}{8+5}\right) - \frac{5}{8+5}\log_2\left(\frac{5}{8+5}\right)$$

$$= \frac{-8}{13}\log_2\left(\frac{8}{13}\right) - \frac{5}{13}\log_2\left(\frac{5}{13}\right)$$

$$= -0.615\log_2(0.615) - 0.384\log_2(0.384)$$

$$= 0.431 + 0.530$$

$$= 0.961$$

Now, let's evaluate the entrophy and information gain of each attribute.

Entropy and information gain of outlook.

Outlook	Yes	No	E(Yes, No)
Rainy	2	3	$\frac{-2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.970$
Overcast	4	0	$\frac{-4}{4+0}\log_2\frac{4}{4+0} - \frac{0}{4+0}\log_2\frac{0}{4+0} = 0$
Sunny	2	2	$\frac{-2}{2+2}\log_2\frac{2}{2+2} - \frac{2}{2+2}\log_2\frac{2}{2+2} = 1$

Overall entropy of outlook

$$E(T, \text{outlook}) = \frac{5}{13} \times 0.970 + \frac{4}{13} \times 0 + \frac{4}{13} \times 1$$

$$= 0.373 + 0 + 0.307$$

$$= 0.68$$

Information gain.

$$\text{Gain}(T, \text{outlook}) = E(T) - E(T, \text{outlook})$$

$$= 0.961 - 0.68$$

$$= 0.281$$

Entropy and Information of temperature.

Temperature	Yes	No	E(Yes, No)
Hot	2	2	$-\frac{2}{2+2}\log_2\frac{2}{2+2} - \frac{2}{4}\log_2\frac{2}{4} = 1$
Mild	4	2	$-\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} = 0.917$
Cool	2	1	$-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} = 0.918$

Overall entropy of temperature

$$E(T, \text{Temperature}) = \frac{4}{13} \times 1 + \frac{6}{13} \times 0.917 + \frac{3}{13} \times 0.918$$

$$= 0.942.$$

Information Gain of Temperature

$$\text{Gain}(T, \text{Temperature}) = 0.961 - 0.942.$$

$$= 0.019$$

Entropy and Information gain of Humidity.

Humidity	Yes	NO	E(Yes, NO)
High	3	4	$\frac{-3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.9851$
Normal	5	1	$\frac{-5}{6}\log_2\frac{5}{6} - \frac{1}{6}\log_2\frac{1}{6} = 0.65$

Overall entropy of Humidity

$$E(T, \text{Humidity}) = \frac{7}{13} \times 0.9851 + \frac{6}{13} \times 0.65$$

$$= 0.8304.$$

Information gain of Humidity

$$\text{Gain}(T, \text{humidity}) = 0.961 - 0.8304$$

$$= 0.1306$$

Entropy and Information Gain of windy.

Windy	Yes	NO	E(yes, NO)
True	3	3	$\frac{-3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1$
False	5	2	$\frac{-5}{7}\log_2\frac{5}{7} - \frac{2}{7}\log_2\frac{2}{7} = 0.863$

Overall entropy of windy

$$E(T, \text{windy}) = \frac{6}{13} \times 1 + \frac{7}{13} \times 0.863$$

$$= 0.9262.$$

Information gain of windy

$$\text{Gain}(T, \text{Windy}) = 0.961 - 0.9262$$
$$= 0.0348$$

Comparing all the information gain, we get information gain of outlook is high i.e. 0.281.

Therefore, attribute 'outlook' is our root node.

8. Write short notes:-

a. Vanishing gradient and exploding gradients.

→ When training a deep neural network with gradient based learning and backpropagation, we find we find the partial derivaties by traversing the network from the final layer (y-hat) to the initial layer. Using the chain rule, layers that are deeper into the network go through continuous matrix multiplications in order to compute their derivatives.

In a network of n hidden layers, n derivatives will be multiplied together. If the derivatives are large then the gradient will increase exponentially as we propagate down the model until they eventually explode, and this what we call the problem of exploding gradient.

Alternatively, if the derivatives are small then the gradient

will it decrease exponentially as we propogate through the model until it eventually vanishes, and this is the vanishing gradient problem.

b. Dimensionality Reduction

→ Dimensionality Reduction is a machine learning or Statistical technique of reducing the amount of random variables in a problem by obtaining a set of principal variables. This process can be carried out using a number of method that simplify the modeling of complex problems, elimate redundancy and reduce the possibility of the model overfitting and thereby including results that do not belong.

The process of dimensionality reduction is divided into two components, they are feature selection and feature extraction. In feature selection, smaller subsets of features are chosen from a set of many dimensional data to represent the model by filtering, wrapping or embedding. Feature extraction reduces the number of dimensions in a dataset in order to model variables and perform component analysis.

Methods of dimensionality reduction include:

i) Factor Analysis
ii. Low variance Filter
iii. High correlation Filter
iv. Backward Feature selection
v. Principal Component Analysis (PCA)

vi. Linear Discriminant Analysis
vii Method Based on Projections
viii t-Distributed Stochastic Neighbor Embedding (t-SNE)
ix. UMAP
x. Independent Component Analysis
xi Missing value Ratio
xii. Random Forest.

c. LSTM

→ LSTM stands for 'Long short-term memory." which is a special kind of recurrent neural network (RNN). One way to motivate LSTMs is to think about RNNs that suffer from vanishing gradients. Gradients contain information and over time if gradients vanish important localized information is lost. This is an issue for a famous applications of RNNs, understanding languages.

One odd way of thinking about language/sentences they contain long-term dependencies. This happens for many reasons, but one reason is because of concrete grammar rules.

Consider these two sentences
- The man drives away
- The man, after robbing a bank and rushing to his car, drives away.

We know that correct verb form is 'drives' because the subject is singular ("the man"). This is an example of concrete long-term dependency. For RNNs, you want

to make sure that the network "remembers" that the subject is singular, so that it knows to predict the correct verb form.

This is where the LSTM comes in. LSTMs are a more general version of a gated recurrent unit. The idea here is that the LSTM module has multiple "gates" inside of it with trained parameters. Some of these gates control the modules "output" and other gates control "forgetting." In this way the LSTM helps remember cell states.

9. Consider the logistic regression model $y = g(w^T x)$, trained using the binary cross entropy loss function where $g(z) = 1/1+e^{-z}$ is the sigmoid function. Your friend proposes the modified logistic regression, using

$$g(z) = \frac{e^{-z}}{1+e^{-z}_{be}}$$

The model would still, trained using the binary entropy loss. How would the learnt model parameters, as well as the model predictions, differ from conventional logistic regression? Justify your answer mathematically. A purely graphical explanation is not sufficient.

→ The goal of the logistic regression is to predict the target class t from an input z. The probability $P(t=1|z)$ that the input z is classified as $t=1$ is represent by the output y of the logistic function computed as $y = g(z)$. The logistic function g is defined as:-

$$g(z) = \frac{1}{1+e^{-z}}$$

This logistic function, implemented below as logistic(z), maps the input z to an output between 0 and 1 as is illustrated in below.

We can write the probabilities that the class is $t=1$ or $t=0$ given input as:-

$$p(t=1|z) = g(z) = \frac{1}{1+e^{-z}}$$

$$p(t=0|z) = 1 - g(z) = \frac{e^{-z}}{1+e^{-z}}$$

Note that, input z to the logistic function corresponds to the log odds ratio of $P(t=1|z)$ over $P(t=0|z)$

$$\log \frac{P(t=1|z)}{P(t=0|z)} = \log\left(\frac{1/1+e^{-z}}{e^{-z}/1+e^{-z}}\right)$$

$$= \log\left(\frac{1}{e^{-z}}\right)$$

$$= \log(1) - \log(e^{-z})$$

$$= z$$

This means that log odds ratio $\log(P(t=1|z) / P(t=0|z))$ changes linearly with z. And if $z = x\omega$ as in neural networks, this means that log odds ratio changes linearly with the parameters ω and input samples x.

Since neural network typically uses gradient based optimization techniques as gradient descent it is important to define the derivative of the output y of the logistic function with respect to its input z.

$$\therefore \frac{dy}{dz} = y(1-y)$$

$$\therefore y = g(z) = \frac{1}{1+e^{-z}}$$

The output of the model $y = g(z)$ can be interpreted as the probability y that input z belongs to one class ($t = 1$), or probability $1-y$ that z belongs to the other class ($t = 0$) in a two class classification problem.

We note this down as:-

$$P(t=1|z) = g(z) = y.$$

The neural network model will be optimized by maximizing the likelihood that a given set of parameters θ of the model can result in a prediction of correct class of each input sample. The parameter θ transform each input sample i ~~to~~ into an input to the logistic function z_i. The likelihood maximization can be written as:-

$$\underset{\theta}{\operatorname{argmax}} L(\theta|t,z) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} L(\theta|t_i,z_i)$$

The likelihood $\mathcal{L}(\theta|t,z)$ can be rewritten as the joint probability of generating t and z given the parameters θ: $P(t,z|\theta)$. Since, $P(A,B) = P(A|B)\,P(B)$ this can be written as:-

$$P(t,z|\theta) = P(t|z,\theta)\,P(z|\theta)$$

Since, we are not interested in the probability of z we can reduce this to:

$$\mathcal{L}(\theta|t,z) = P(t|z,\theta) = \prod_{i=1}^{n} P(t_i|z_i,\theta)$$

Since t_i is a Bernoulli variable,

$$P(t|z) = \prod_{i=1}^{n} P(t_i=1|z_i)^{t_i} \cdot (1-P(t_i=1|z_i))^{1-t_i}$$

$$= \prod_{i=1}^{n} y_i^{t_i} \cdot (1-y_i)^{1-t_i}$$

Since the logarithmic function is monotone increasing function we can optimize the log-likelihood function $\operatorname{argmax}_\theta \log \mathcal{L}(\theta|t,z)$. This maximum will be same as the maximum from the regular likehood function. The benefit of using the log-likelihood is that it can prevent numerical underflow when the probabilities are low. The likelihood function can be written as:-

$$\log \mathcal{L}(\theta|t,z) = \log \prod_{i=1}^{n} y_i^{t_i} (1-y_i)^{1-t_i}$$

$$= \sum_{i=1}^{n} t_i \log(y_i) + (1-t_i) \log(1-y_i)$$

Minimizing the negative log likehood corresponds to maximizing the likelihood. This error fondion $E(t,y)$ is typically known as the cross_entropy error fonction which is also known as log-loss.

$$E(t,y) = -\log P(O|t,z)$$

$$= -\sum_{i=1}^{n} [t_i \log(y_i) + (1-t_i) \log(1-y_i)]$$

$$= -\sum_{i=1}^{n} [t_i \log(g(z)) + (1-t_i) \log(1-g(z))]$$

This fonction looks complicated but besides the previous derivation there are a couple of intuitions. Why this fonction is used as a loss function for logistic regression. First of all it can be written as:-

$$E(t_i, y_i) = \begin{cases} -\log(y_i) & \text{if } t_i = 1 \\ -\log(1-y_i) & \text{if } t_i = 0 \end{cases}$$

Which in the case of $t_i = 1$ is 0 if the probability to predict the correct class is 1 and goes to infinity as the probability to predict the correct class goes to 0.

Notice that the loss function $E(t,y)$ is equal to the negative log probability that z is classified as it correct class

$$-\log(P(t=1|z)) = -\log(y)$$
$$-\log(P(t=0|z)) = -\log(1-y)$$

By minimizing the negative log probability, we will maximize the log probability. And since t can only be 0 or 1, we can write $\mathcal{E}(t,y)$ as:-

$$\mathcal{E}(t,y) = -t\log(y) - (1-t)\log(1-y)$$

which will give $\mathcal{E}(t,y) = -\sum_{i=1}^{n} [t_i \log(y_i) + (1-t_i)\log(1-y_i)]$

it will sum over all n samples.

Another reason to use the cross-entropy function is that in simple logistic regression this results in a convex loss function, of which the global minimum will be easy to find.

Note: This is not necessarily the case anymore in multilayer neural network.

Now,
The derivative $d\mathcal{E}/dy$ of the loss function with respect to its input can be calculated as:-

$$\frac{d\mathcal{E}}{dy} = \frac{d(-t\log(y) - (1-t)\log(1-y))}{dy}$$
$$= \frac{d(-t\log(y))}{dy} + \frac{d(-(1-t)\log(1-y))}{dy}$$
$$= \frac{-t}{y} + \frac{1-t}{1-y} = \frac{y-t}{y(1-y)}$$

This derivative will give a nice formula it used to calculate the derivative of the loss function with respects to the inputs of the classifier dE/dz. Since the derivative of the logistic function is:-

$$\frac{dy}{dz} = y(1-y)$$

So,

$$\frac{dE}{dz} = \frac{dy}{dz} \times dE/dy$$

$$= y(1-y) \times \frac{y-t}{y(1-y)}$$

$$= y - t.$$

10. What does it mean to pre-train word embeddings and why is it useful? Can you think of an application where it may be useful to train "word embeddings", but the data is not text? Discuss. (This is, embedded objects are not words.)

→ Pretrained word embeddings learned in one task are useful for solving another similar task. These embeddings are trained on large datasets, saved, and then used for solving other tasks. That's why pretrained word embeddings are form of Transfer learning.

Transfer learning, as the name suggests, is about transferring the learnings of one task to another. Learnings could be either weights or embedding. In our case here,

learnings are the embeddings. Hence, this concept is known as pretrained word embeddings.

The performance of Neural Machine Translation (NMT) systems often suffers in low-resource scenarios where sufficiently large-scale parallel corpora cannot be obtained. Pre-trained word embeddings have proven to be invaluable for improving performance in natural language analysis tasks, which often suffer from paucity of data. However, their utility for NMT

Pretrained word embeddings capture the semantic and syntatic meaning of a word as they are trained on large datasets. They are capable of boosting the performance of a Natural Language processing (NLP) model.

Pre-trained word_embedding is used fore-
a Sparsity of training data

One of the primary reasons for not doing is the sparsity of training data. Most real-world problems contain a dataset that has a large volume of rare words. The embeddings learned from these datasets cannot arrive at the right representation of the word.

In order to achieve this, the dataset must contain a rich vocabulary. Frequently occuring words build such a rich vocabulary.

b. Large number of trainable parameters

Secondly, the number of trainable parameters increases while learning embeddings from scratch. This results in a slower training process. Learning embedding from scratch might also leave you in an unclear state about the representations of the words.

So, the solution to all above pre-trained word embedding

The application of word embedding are :-

a. Associating Neural Word embeddings with deep image representations using Fisher vectors

b Incorporating visual features into word embeddings: a bimodal autoencoder-based approach.