e 5

* Comparison between Regression & Classification (slide - 2)

- In regression, the labels aren't binary or ternary or anything like that. We map the features to a continuous entity like for example the Real line.
  Generaly, it's $\mathbb{R}$ maps to $\mathbb{R}$ (features & labels $\in \mathbb{R}$)
- Many types of losses are used in regression analysis. But the 'squared error loss' is of focus in this study.

★ Linear Regression (slide - 4 -)

- Here instead of classifying data based on value of $\theta^T x + \theta_0$, we actually use it's value as our label. And we measure our loss by taking difference between the guess & actual value, & averaging their squares.
- In the loss function, for ease of functionality, we expand the feature vector by 1 & add value '1' to

all data points' last feature. (the extra feature) & add θ_0 to the last feature (extra feature).

- We formulate those equations for all data points into matrices. So now, we can say that we are just taking the magnitude squared of the resultant vector for our loss.

★ Linear Regression (A direct solution) (slide - 5 -)

- In this case, where we have to minimize the loss function, we can directly minimize it without using numerical methods or gradient descents or stuff because if you ~~look~~ look carefully & expand the loss function, it's a simple paraboloid. So, it has only 1 global minimum (which is unique)
- Now, if you invert it (multiply loss function by -1), you would see that there is no global minimum.
- Another example would the 3rd one given in the slide where there exists a global minimum, it's just not unique. A line defines the global minimum instead of a single point in this case.
- In linear algebra, if the matrix of second derivatives is positive definite, then the function has unique minima & it "curves up"

* Linear Regression (A direct Solution) (slide - 6 - 94)

- See matrix derivatives to understand how we get $\nabla_\theta J(\theta)$
- Function "curves up" → Matrix of 2nd derivatives is invertible

- Here, we have only considered a feature vector of only 1 feature. But, loss function $J(\theta)$ is plotted in 3-D because we have assumed θ_0 to be the 2nd dimension. (though θ_0 is already included in the θ vector)

★ Linear Regression (A direct solution) (Slide - 7 - 99)

- Now, if we consider a 2 dimensional feature vector (excluding θ_0), we would get a 4-D graph of $J(\theta)$ which we can't draw.
- Note that the hypothesis is a hyperplane that is always linear for linear regression & classification.
- Note:- θ is always perpendicular to the hyperplane. It's the normal vector of the hyperplane.

✱ What can go wrong in practice? (Slide - 8 - 119)

- Sometimes there isn't a unique hyperplane:- Imagine a case where you have 2-D feature vectors. So, if we

plot $\theta^T x + \theta_0$ against θ_1 & θ_2, we will get a 3-D plot. Now, ideally for 2-D feature vectors, there might be a best fitting hyperplane for which all the data points are as co-planar as possible. That will be our required hyperplane.

But, imagine that the data points fit a line in 3-D instead of a plane. Then you can have infinitely many hyperplanes that contain that fitting line.

- Sometimes, there's technically a unique best hyperplane but just because of noise:-

Consider the above example, where the data fits a line in 3-D instead of a plane. Thus, there might be infinite many planes that satisfy our requirements / model.

But consider this: our data is rather noisy & this noise is spread out relatively, very slightly more in one particular plane. Thus, we will get a hyperplane which is unique, even though the data seems to fit a line in 3-D.

- Feature encodings (one-hot); real life features often co-related; many features dimensions :-

It can happen that one or multiple of your features are functions of other features. i.e. they are correlated, hence there is redundancy & there probably won't be a unique hyperplane.

Another thing that can happen is you have equal no. of features and data points. For example, a 1-D feature with only one data point will have infinite lines (hyperplanes) passing through it. A 2-D feature with only two data points will have infinite planes (hyper-

planes) passing through it. (Because you need atleast 3 points to define a plane).

Now, this doesn't happen for lower dimensional feature vectors. But imagine this: you have higher no. of data points than raw features (schema. But like you saw with the example of one-hot encoding, you could get many features (extra dimensions) from just one type of feature. Hence, the features (extra dimensions) can blow up. So, if you get no. of features = no. of data points, you won't get unique hyperplanes.

- <u>How to choose among planes? - Preferance for θ components being near zero.</u>

Imagine a situation like mentioned in above points where you have multiple hyperplanes as solutions & you have to choose ~~from~~ one of them. Every one has the same loss. Or in case of rather noisy data some ~~feat~~ hyperplanes are just slightly better than the others (But you shouldn't discard others because the slightly better planes are result of noise & not actual good reasons).

Now, we have earlier studied that higher θ values imply higher confidence. But in this case, higher θ values also imply that some features may have higher influence than others. But even though some θs have high value & some have low value, their loss is same or almost the same.

So, ideally you should choose θ with smaller values, so that you don't show ~~bias~~ illogical bias between features and their values.

High θ ⇒ high confidence (But we don't have any reason for this high confidence / bias).

Hence, we opt for lower θ values (hyperplanes with lower θ values)

★ Regularizing Linear Regression (slide- 9- 138)

- One way we can regularize linear regression is adding a penalty for higher θ_s
- When we use $\|\theta\|^2$ penalty or L-2 regularizer, it's called ridge regression.

★ Some notes on features (Slide-10-Last)

- Note that when we use a ~~linear~~ regularizer in the loss function, we assume that all the features in the feature vector are all on the same scale.
- If they are not, the features won't be equally penalized. We need equal penalization.

★ Optimizing Linear regression (Slide- 11- Last)

- Note that the direct solution was an analytical solution. For complex problems, we can't use or rather can't get an analytical solution. Hence, we go for numerical solutions.
- Note that irrespective of which solution we choose, we also have to consider the running time & computational power required. For example, we can always reduce the

step size parameter in gradient descent to as small as we want, but that would require more computational power & more no. of steps. This will result in very high running times which may be impractical. So, you have to compromise between accuracy & running time.

- Now, consider the above example where we have an analytical solution. We just have to calculate the inverse & multiply it by the remaining matrices in the equation. But (and it's a Big but), the time complexity for computing matrix inverse of $d \times d$ dimensions is $O(d^3)$ which is really bad. So, we may prefer gradient descent when there are very high no. of dimensions.

Lecture 5* continued:-

★ Stochastic Gradient Descent (& G.D. vs S.G.D.) (Slide - 13 - last)

- In gradient descent, what you do is first select the initial parameters θ & θ_0. Then, you calculate the ~~gradient~~ loss on the Entire dataset with given values of θ & θ_0. Then you do gradient & multiply by step size & use it to update the θ & θ_0.
- Basically, before you update θ & θ_0, you calculate the loss on the entire dataset. This becomes time costly for large no. of iterations.

- In stochastic gradient descent, you calculate the loss on a random data point from the dataset & then you update the θ & θ_0 values.

- Thus, you only use 1 single data point instead of the entire dataset before updation of θ & θ_0.
- This becomes very noisy & relatively inaccurate than regular gradient descent & you would need much more no. of iterations to obtain good θ & θ_0.
- But, obviously, S.G.D. becomes extremely cheap in terms of time & computational power required.
- Hence, again it comes to the question of the right compromise & you should choose between G.D. & S.G.D. based on time, power, necessity, etc.