**Machine Learning**

# **Understanding Data**

7 questions you must ask yourself.

1. How big data is?
2. How does data look like?
3. What are data types of columns?
4. Are there any missing values?
5. How does data look mathematically?
6. Are there any duplicate values?
7. How is the correlation between columns?

# **EDA**

Exploratory Data Analysis

Asking question again and again and finding their answers.

## **Univariant Analysis**

### **Categorical Variable**

1. Count Plot, how much value were there in each
2. For percentage use Pie chart

### **Numerical Variable**

1. **Histogram**, to understand the distribution of the data
2. **Displot**, tell the probability of the number, it shows curve + histogram. On y-axis probability and on x-axis values. Also called PDF (Probability Density Functions). Also shows data is skewed or not.
3. **Boxplot**, shows the outliers and quartile of the data

## **Bivariant Analysis**

1. **Scatterplot** (Numerical - Numerical), relationship between columns to understand a relationship or trend, which would be impossible to see in almost any other form.
2. **Barplot** (Numerical – Categorical)
3. **Boxplot** (Numerical – Categorical)
4. **Displot** (Numerical – Categorical) you can check the probability of for example, survived not survived with age column.
5. **HeadMap** (Categorical – Categorical)
6. **ClutsterMap** (Categorical – Categorical), shows which value is closer to each other, it creates a dendrogram for relationship.
7. **Pairplot,** shows the scatter plot of each numerical column with others.
8. **Lineplot**, if on x-axis time-based data.
9. **PivotTable** draw table of two columns, based on index.

## **Profiling**

Is the python library which automatically do some of above works. And return HTML interactive page.

# **Feature Engineering**

Preparing dataset for machine learning model

## **Feature Transformer**

If we think a column will not give us well performance, we transform it in some other form by changing it.

### **Missing Values**

Either you fill missing values or remove.

### **Handling Categorical Values:**

By converting the categorical values into numerical

### **Outlier Detection**

### **Feature Scaling**

Converting the feature (columns) numerical values into given range.

Is last step before giving dataset to the model.

#### Standardization

Also called z-score normalization.

Create new column with given column example, age has some value, subtract each value of age from its mean and divide it on the standard deviation. This will give us new column; new columns mean will be = to zero and standard deviation will be = to 1.

In terms of mean your centering, and in terms of scaling you are scaling down the large values, or small values to scale up but to the 1.

Datapoints remains same to actual values, but data become centered and scaled.

Draw displot of actual data and scaled data you will understand.

Outliers behave same with or without standardization.

**We must do** standardization for these algorithm

* K-means (Euclidian distance)
* KNN (Euclidian distance)
* PCA (we need to control variance and do mean centering)
* ANN
* Gradient Decent

**We do not need scaling** for decision tree, random forest, XG boost, GB boost.

#### Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

Normalization is the process of translating data into the range [0, 1] (or any other range) or simply transforming data onto the unit sphere.

* **Minmax scaling**, actual value – minimum value of column / maximum value of column – minimum value of column. [0 to 1]
* **Mean normalization**, actual value – mean value of column / maximum value of column – minimum value of column. [-1 to 1]. We use where centered data is needed.
* **Max absolute scaling**, where too many zeros we use max absolute scaling, actual value / |max value of column|.
* **Robust scaling**, actual value – median value of column / IQR (75% - 25%). Robust to outliers if data has a lot of outliers.

#### Normalization vs Standardization

Is feature scaling is required?

What to do normalization / standardization?

If we know min and max value of feature such as for CNN, we know for picture resolution 0 to 255, so we can use their max min scaler.

Else use standardization

## **Feature Construction**

Creating new column with given columns.

## **Feature Selection**

Select the main feature from the dataset and gives these features to model so model can perform better.

## **Feature Extraction**

Converting given input features into completely new features. We do feature reduction as well as feature creation.

# **Algorithms**

## **Linear Regression**

Line of equation Y = m \* x + b, here m is slope and b is point on y-axis.

We must draw a line which pass closely to all points for predicting output points, here m is weightage how much x depends on Y.

We can calculate m and b value with two methods.

* Closed form expression also called ordinary least square (OLS) and can find through direct formula. For lower dimension we use direct formula to find m and b.
* Un-closed form expression also called Gradient descent.

### **Simple Linear Regression Using Direct Formula**

If 1 input column and 1 output.

Y=mx+b, here m is slope and b is point on y-axis.

We must draw a line which pass closely to all points.

M is weightage how much x depends on Y.

Need to minimize the distance between points and the line we will draw.

If we assign each point a variable and then square each and sum their distance to find error to minimize the distance.

E = d1 \* 2 + d2 \*2 + dn \* 2.

Taking square means, we can find differentiation on any points.

Error function also called **loss function**.

The distance (difference) is what our model predicts and where it actually points is located. So, (Y actual point – Y predict) and their square, sum of all the distance for each point.

We can write this as below.

**Y predict = m \* x + b**



To minimize the error how would we know that the error is now minimum?

We can find through maxima and minima.

When we are at minimum slope is zero. To find slope we take derivative and make it zero. But we have two values m and b, we will take partial derivative of each from this we will get two equations and with these two we will get m and b values.

**Partial Derivative w.r.t b**

E = = 0

∂E / ∂b= ∂/∂b = 0

= 0

= 0

= 0

= 0

= b

As we have found equation of b, now putting equation of b in below equation

E =

E =

**Partial Derivative w.r.t m**

∂E / ∂m=

=

=

**If dividing -2 on both sides**

=

=

=

**Final Equation**  m =

### **Multiple Linear Regression Using Direct Formula**

= Xb

E = ==

E =

= Xb

E =

E= - Xb -

**Xb == if you calculate it their results will be same.**

E= – 2 (Xb)

Now we calculate minimum by taking partial derivative and making it zero

∂E / ∂b = ∂ / ∂b [ – 2 (Xb) ] = 0

= 0 - 2 X + ∂ / ∂b ) = 0

= - 2 X +

(

(

b= (

**Final equation.**  b= (

### **Polynomial Linear Regression**

If X and Y, then for 2 degrees, will be , , and y. Previous you have just 1 feature now

For higher degree of polynomial, it overfit and for lower its underfit.

### **Regularization**

There are three techniques available to overcome overfitting of logistic or linear regression bagging, boosting, and regularization. We will study here regularization. Regularization is technique in which you induce some extra to the ML model to overcome overfitting.

If we talk about overfitting means, model perform well on training but not on testing. here, if overfitting the value of m will be high.

Three types of regularization:

#### **Ridge (L2)**

Adding in the loss function you can reduce the variance. Values of coefficient and overfitting reduce.

In linear regression

Now we will add this in

Now we are reducing both and the result of m (slope) after adding this will be less than the result of simply linear regression loss function.

Now we need to take partial derivate w.r.t b and m,

**for b**

E = = 0

∂E / ∂b= ∂/∂b = 0

= 0

= 0

= 0

= 0

= b

**w.r.t m**

E = +

E = +

**Partial Derivative w.r.t m**

∂E / ∂m= +

= +

= +

**If dividing -2 on both sides**

= +

= +

= +

m =

**For multiple columns**

E =

E = +

Simplifying and then after taking partial derivative of w

W = (

**For Gradient Descent**

But we can write it like this,

L = +

Multiplying both on

= +

= +

= +

After taking derivative **w.r.t** w the result will be

α \*

* How coefficient get effected?

All coefficients decrease towards zero but not = zero.

* What if the value of high?

The result of each coefficient will not be zero, but close towards zero.

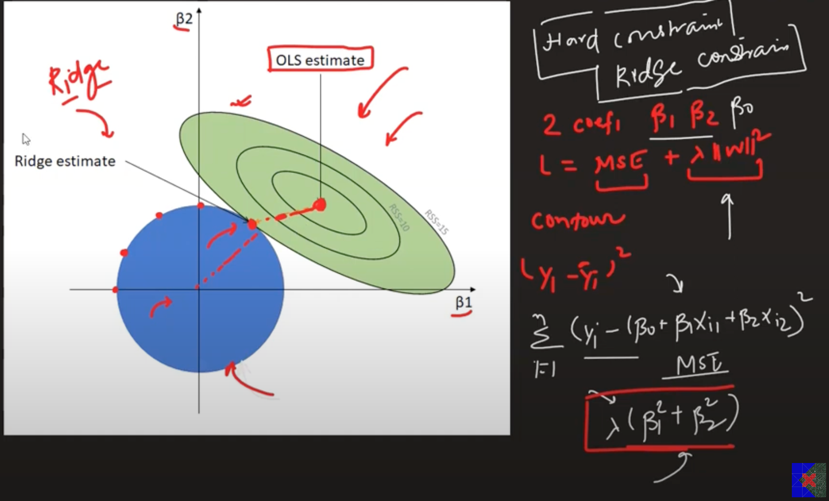
* Bias variance trade of

When value is close to zero bias will decrease, because model overfit and variance will increase, when is high the bias will increase model will underfit and variance will decrease.

* Impact of on loss function

When increase the result of coefficient decrease to zero.

* Why called ridge



The result of this will be always on boundary of the blue circle but close to the loss circle that is why it called ridge.

#### **Lasso (L1)**

#### **Elastic Net**

## **Gradient Decent**

Is an optimization algorithm. We give differentiable function to gradient descent it gives us minimum of that function.

**How would I know I am at minimum?**   
You must find slope at that point to find out. If slope is negative than increment in b value and similarly if slope is positive than decrement in value of b.

To control long or short jump you multiply slop with learning rate, I represent it with alpha, its value range between 0 to 1, 0.0001, 0.1 etc.

α \*

**Where to stop?**

* The first step is, if you know that or close to zero and you are not doing any improvement means you reached to minimum. Or if the is minimum mainly 0.0001.
* Second step is you limit the iteration name as epoch. 1000, or 100 etc.

**Start with random value of b.**

for i in epoch:

α \*

If learning rate is = 0.01

To find slope you know

By taking derivative

=

α \*

**How to calculate m?**

Step 1 initialize random values for m and b as well as learning rate and epochs.

α \*

**What is slope in this m?**

**What if we have local minima and global and we reached local minima?**

**Data also effects on step size, if data is in range or standardized than it will reach faster to the minima.**

### **Batch Gradient Descent for multiple features**

Batch gradient descent in which we use all data at the same time and feed it to gradient descent to update m and b. It is same as the above gradient descent; we used above for single feature now in this we are using it for multiple features.

Start with random value of m and b as well as set learning rate and epoch size.

If we have n = 2

For multiple

By taking partial derivative of

∂L / ∂= ∂/∂

∂L/∂

By taking partial derivative of

∂L / ∂= ∂/∂

∂L/∂

By taking partial derivative of

∂L / ∂= ∂/∂

∂L/∂

#### **Problem of batch gradient descent**

* If you have large dataset, you need too much calculation.
* Hardware problem

### **Stochastic Gradient Descent**

To solve batch gradient problems, we use stochastic gradient descent.

Change b and m for each row of data. Which make it faster convergence less epochs, hardware problem solved as you are loading data in memory one by one.

Stochastic select random row from data and change data.

#### **Problem of stochastic gradient descent**

As it selects random rows, so its answer is not consistence.

#### **When to use stochastic gradient descent**

* Big dataset
* Non-Convex function (both global and local minima)

You can do learning schedule mean you can change learning rate with time.

### **Mini-Batch Gradient Descent**

You divide dataset into batch and then change the values of m and b (intercept and coefficient). If randomness is high in stochastic, we use mini batch.

## **Logistic Regression**

Does data is linearly sparable?

### **Perceptron Trick**

Start with random value of A, B, and C. This start with random state draw lines and asking each point you are accurately classify or not and the point tells, if not line move towards it. You can loop it till convergence, or number of epochs.

This gives us positive side

This gives us negative side

If perceptron in negative region we add 1 in its coordinate and subtract from line. And

If perceptron in positive region we add 1 in its coordinate and add from line.

We multiply coordinate with learning rate and then subtract for transformation.

This is how we get new coefficient.

α \*

We can write below line to this as well

=

If is 0 in dataset but according to model its 1 so we say its negative point which is in positive region. In that case we will update w.

α \*

If is 1 in dataset but according to model its 0 so we say its positive point which is in negative region. In that case we will update w.

α \*

Without using if else condition we can simply use this formula

α \*

If 1 and 1 or 0 and 0 then not change else, 1 and 0 move positive, else negative.

Perceptron stops when all classify but sklearn logistic regression improve itself more. We can minimize error on train, but error may occur in test.

#### **Solution of perceptron issue**

Before that we were only focusing on misclassified points but now, we will see all points to check, if misclassified then pull else push. How much we pull, and push depends upon how far line is from the point.

The positive point which is near the line will push more to line, whereas the positive point which is far more from line will push with less force. Conversely the negative point which is misclassified near to line and will pull with less whereas the far point will pull more.

α \*

Before this we were not changing for these cases If 1 in dataset and prediction is 1 or 0 in dataset and prediction is 0 then not change.

But now we will change.

This can be only happened with changing in . The answer should not 0 or 1 but something else. So, for handle this we use sigmoid function.

#### **Sigmoid Function**

The answer will always between 0 and 1 range.

Now if answer of sigmoid is positive then > 0.5 else < 0.5.

Now we use this for prediction.

**For this we also face some issues, to solve this we need to do this.**

#### **Loss function**

**Maximum Likelihood** is function which tell is which model is performing better by multiplying the probability of the prediction with each point. You take probability based on class and multiplying with each other.

= P(Green1) \* P(Green2) \* P(Red1) \* P(Red2)

We must choose model which result is maximum.

This problem is better for small problem but not best for large dataset.

To solve this we use log, this time we are adding instead of multiplying.

= - log (Green1) - log (Green2) - log (Red1) - log (Red2)

Addition of maximum likelihood of – log is called cross entropy.

For this case we must choose model which result is minimum.

We cannot use it directly; we use it with gradient descent

Also called log loss error and binary cross entry.

#### **Derivative of sigmoid**

### **Gradient Descent**

Taking derivative of w of this part

Now taking derivative of w of this term

**Now**

Gradient Descent

α \*(-

α \*(

### **SoftMax Regression (Multi Classification)**

#### **Softmax Funtion**

For 2 features and 1 target you divide dataset into three parts, all column with 0 class, similarly 1 and 2.

As same you will get 3 model with 3 coefficients each.

This approach is slow for large dataset, as you need to test single row to each model to predict the output.

What you do, is to change the loss function of logistic regression.

### **Polynomial Logistic Regression**

You can use polynomial linear regression for the non-linear problems.

## **Support Vector Machine (SVM)**

Improve the idea of logistic regression. SVM select better line to classify. Gap should be maximized as possible to select the better line for each point.

The core idea in SVM is to find the line which is margin maximizing hyperplane.

How?

For example, from the hyperplane (line) you draw you go upward parallel to the line until line reached to first point, you draw new line called positive hyperplane, same do for negative.

Next you calculate the shortest distance between positive and negative line, which is called margin denote as d.

You do for other hyperplane as well and calculate d for other than you decide which hyperplane is best by selecting the maximum d.

SVM is very robust to outliers

Work on non-linear data (by using kernel)

Both for classification and regression problem.

**Mathematics**

Is general equation.

If hyperplane passed from origin, then

W is always perpendicular to the x (hyperplane).

### **Decision rule**

How do we classify point is negative or positive, we take projection of point with w.

Replace -c with +b

If result is, then positive else negative.

For positive hyperplane

For negative hyperplane

### **Hard Margin SVM**

From positive to hyperplane negative to hyperplane distance should be same.

If multiply with positive, then else

For support vector the point where positive and negative touches is called support vector, for all support vectors equations will be below.

Valid till below constraint follows.

*For +1*

*For -1*

We must maximize this

Argmax (, )

Until this condition is true

The above function only works for linearly separable data, not for non-linear.

### **Soft Margin SVM**

Argmin (, )

For correctly classified point for other points for positive point in negative regions, from positive lines to positive point in negative region distance will be Similarly for negatives. We have to minimize the errors.

We have two terms one is which increases margin and the other is which minimize the error.

SVM has two error margin error and classification error here you solve both.

C is hyper parameter, if c value is high means do not focus on margin error rather reduce classification error. Same reducing C value means do not focus on classification error rather focus on margin error.

### **Kernel Trick**

For non-linear we use kernel trick.

Transform the function which will make it linearly separable for higher dimension features (lower dimension into higher dimension) so non-linear should be separable that is called kernel trick.

## **Naive Bayes Classifier**

## **K Nearest Neighbors**

## **Decision Trees**

## **Random Forest**

## **Bagging**

## **Adaboost**

## **Gradient Boosting**

## **Xgboost**

## **Principle Component Analysis**

# **Bias vs Variance Trade Off**

# **Cross Validation**

# **Machine Learning Matrices**

## **Regression Matrices**

### **MAE**

Mean Absolute Error is the magnitude of difference between prediction and true value.

Same unit for MAE and output column

Robust for outliers, but it is not differentiable for at zero.

Loss function

### **MSE**

It finds the square distance. We can use it as loss function because it is differentiable. It penalizes the outliers simply not robust to outliers.

Loss function

### **RMSE**

It is differentiable, not robust to outliers.

Loss function

### **R2 Score**

Also called coefficient of determination or goodness of fit.

= Sum of squared error regression line

= Sum of squared error mean line

To calculate you are going to 1 rather 0.

Can be in negative mean is making mistake more than .

R2 score means, this number of scores of variances in output column is explained by the input columns

### **Adjusted R2 Score**

In some cases, because of adding irrelevant column r2 score not decrease, and stay same, to handle this we use adjusted r2.

n is number of rows, k is number of independent columns

Good for adjusted r2 score.

## **Classification Matrices**