Machine learning

building model:

- Problem formulation
- Data Tidying
- Pre-Processing
  - Filter data
  - Aggregate values
  - Missing value treatment
  - Outlier treatment
  - Variable transformation
  - Variable reduction
- Train-test split
- Model building
- Validation & Model Accuracy
  - in sample error
  - Out of sample error.
- Prediction

Jupyter:

pwd → working directory

df = pd.read_csv('...of_csv', header = 0)

location of header in this case in zeroth row

df.head()

প্রথম কয়টি

df.shape

(row no, col no)

Univariate analysis:

pattern finding যে গুলো use করা হয়।

EDD: (extended data dictionary)

min, max, mode, avg, 25%, 50%, 75% এইসব বলতে পারে।

df.describe()

seaborn :

~~sns~~

import seaborn as sns

sns.distplot(data.age) [histogram of age]

sns.distplot(data.age, kde=False) [কার্নেল ডেনসিটি গ্রাফ]

" " (" , color="red")

sns.Jointplot(x="x", y="y", data="data_name")

↳ [scatter plot দেখাবে]

sns.pairplot(data_name) → [সবগুলো variable এর scatter plot দেখাবে]

outlier:

sns.Jointplot(x="n_hotel_rooms", y='price', data=df)

Treating outliers:

uv=np.percentile(df.n_hot_rooms, [99])[0] → array থেকে মান

(column to see, percentile to see)

df[(df.n_hot_rooms > uv)]

↳ uv value এর চেয়ে বড় ডাটা দেখাবে

df.no_hot_rooms[(df.n_hot_rom > 3*uv)] = 3*uv

lv = np.percentile(df.rainfall, [1])[0]

df[(df.rainfall < lv)]

df.rainfall[(df.rainfall < 0.3*lv)] = 0.3*lv

variable transform:

Treating Missing values:

df.n_hos_beds = df.n_hos_beds.fillna(df.n_hos_beds.mean())

সবগুলা 0 বা যত খুশি চাইলে :-

df = df.fillna(df.mean()).

Bivariate analysis:

মূলত independent variable নিয়ে কাজ।

① df.crime_rate = np.log(1+df.crime_rate)

df['avg_dist'] = (df.dist1 + df.dist2 + df.dist3 + df.dist4)/4

del df['dist1']

মূলত যা করা হয় :-

① scatter plot

② correlation matrix : 0 মানে relation নাই। high মানে independent variable একসাথে independent না। একটাকে রাখা হয়।

Creating dummy variable:

df = pd.get_dummies(df)

del df['airport_NO']

del df['waterbody_None']

Correlation analysis:

df.corr()

Linear Regression:

$Y = \beta_0 + \beta_1 x$ → Model co-efficient

residual, $e_i = y_i - \hat{y}_i$

population regression line, sample regression line

$[\hat{\beta}_1 - 2 \times SE(\hat{\beta}_1), \hat{\beta}_1 + 2 \times SE(\hat{\beta}_1)]$

↳ true value of β_1 এই interval এর মধ্যে থাকে।

Hypothesis test:

H_0 : No relationship between X & Y

H_a : Some " " X & Y

To disapprove H_0,

we calculate T statistics, $t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)}$ → + value larger হবে

+ value যাওয়া মানে probability of observing any value is = |t| or larger.

↳ এই probability কে বলা হয় p value.

p value কম হলে X এবং Y এর মধ্যে Relation থাকে।

Accuracy of Model:

↳ $RSE = \sqrt{\frac{(y_i - \hat{y}_i)^2}{n-2}}$

↳ $R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$

↳ (0 থেকে 1 এর মধ্যে হবে) 1 এর কাছে থাকা ভালো

Simple linear Regression in Python:

```
import statsmodels.api as sm
X = sm.add_constant(df['room_num'])

lm = sm.OLS(df['price'], df['room_num']).fit()
```

lm → model object
OLS → ordinary least square

```
lm.summary
```

another method: (most used)

```
from sklearn.linear_model import LinearRegression

y = df['price']
X = df[['room_num']]   → 2d array

lm2 = LinearRegression()
lm2.fit(X,y)
print(lm2.intercept_, lm2.coef_)   → β0, → β1

lm2.predict(X)

sns.jointplot(x='df['room_num']', y=df['price'], data=df,
              kind='reg')
```

Multiple linear Regression:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \varepsilon$$

β_0 = the intercept
p = no. of predictors
ε = an error term

F statistics বলতে কোন মানকে বুঝি
যদি p value threshold এর নিচে হয় তাহলে বুঝতে হবে model predictor এর সাথে Response এর সম্পর্ক আছে। তখন individual এর t value আর P value দেখতে হবে।

↳ categorical variable:

estimate দ্বারা বুঝা যায় value কত আসবে [illegible],
আর P value দ্বারা বোঝা যায় Dependency or
Relationship between Result & variable.

test-train split:

```
from sklearn.model_selection import train_test_split
X_train, x_test, y_train, y_test = train_test_split(X_multi,
```

```
X_multi = df.drop("price", axis=1)
```
→ column drop
→ 0 for row drop

```
y_multi = df['price']
X_multi_cons = sn.add_constant(X_multi)
lm_multi = sn.OLS(y_multi, X_multi_cons).fit()
lm_multi.summary()
```

Another Method:

```
lm3 = LinearRegression()
lm3.fit(X_multi, y_multi)
print(lm3.intercept_, lm3.coef_)
```

Test_train split:

```
from sklearn.model_selection import train_test_split
X_train, x_test, Y_train, y_test = train_test_split
    (X_multi, y_multi, test_size=
        random_state=0)
```
↓ train set

```
print (X_train.shape, X_test.shape, y_train.shape,
                                    y_test.shape)

lm_a = LinearRegression()

lm_a.fit(X_train, Y_train)

y_test_a = lm_a.predict(X_test)

y_train_a = lm_a.predict(X_train)

from sklearn.metrices import r2_score
```

r2_score? details

```
r2_score(y_test, y_test_a)

r2_score(Y_train, y_train_a)
```

(6) Ridge & Lasso Regression: → ① Step 2 data standardizing

```
from sklearn import preprocessing → stores info about how to
scalar = preprocessing.StandardScalar().fit(X_train)   divide each row
X_train_s = scalar.transform(X_train)

X_test_s = scalar.transform(X_test)

from sklearn.linear_model import Ridge

lm_r = Ridge(alpha = 0.5)

lm_r.fit(X_train_s, y_train)

r2_score(y_test, lm_r.predict(X_test_s))
```

[alpha vary করতে হবে validation curve]

```
from sklearn.model_selection import validation_curve
```

param_range = np.logspace(-2, 8, 100)

↳ 10^{-2} থেকে 10^{8} পর্যন্ত 100 ভাগে create করবে

train_scores, test_scores = validation_curve(Ridge(), X_train_s, y_train, "alpha", param_range, scoring='r2')

print(train_scores) → তিনটা করে দেখা যাবে

print(test_scores)

train_mean = np.mean(train_scores, axis=1)

test_mean = np.mean(test_scores, axis=1)

train_mean

max(test_mean)

sns.Jointplot(x = np.log(param_range), y = test_mean)

np.where(test_mean == max(test_mean))

param_range[33]

lm_r_best = Ridge(alpha = param_range[33])

lm_r_best.fit(X_train_s, y_train)

r2 = score(y_test, lm_r_best.predict(X_test_s))

r2_score(y_train, lm_r_best.predic(X_train_s)

Lasso!

from sklearn.linear_model import Lasso

lm_l = Lass(alpha = 0.4)

⋮ (same as Ridge)

Problem we may face is called Heteroscedasticity

log y বা log√y নিয়ে variance কমানো যায়.

(78) Outlier detect করার জন্য box plot ও use করা হয়।

sns.boxplot(y="n_hot_rooms", data=df)

Categorical এর জন্য countplot

sns.countplot(x="airport", data=df)

(82) missing value

df.info

দিলে দেখা যাবে n_hos_beds এ কম Data আছে। But edd only numerical এর বেলায়, EDD কেন ও ব্যবহার যায়। শুধু দেখায়।

Soln: df.n_hos_beds = df.n_hos_beds.fillna(df.n_hos_beds.mean())

সবগুলো col^n এ ব্যবহার করতে চাইলে:-

df = df.fillna(df.mean())

(84) Var transformation & deletion — সবগুলো জায়গায় distance (আলাদা) তাই একটা variable বানাবো

df['avg_dist'] = (df.dist1 + df.dist2 + df.dist3 + df.dist4)/4

এবার dist গুলোকে delete করলে ও হবে

Soln: del df['dist1']

del df['dist2'], ... ['dist4']

এখন bus_ter col^n কোনো কাজে আসছে না, তাই এটিও delete

del df['bus_ter']

(86) Non-numerical values কে numerical এ নিচ্ছে.

128| হয়| সব dummy variable create করে নিতে হবে।
df = pd.get_dummies(df)
airport হচ্ছে variable এর নাম যে নামে।
airport_yes & airport_no.

88 Classification algo is used for qualitative or catergontical output.
→ logistic Regression
→ linear discrimant analysis
→ K nearest neighbon

যে ধরনের question আসতে পারে:-

(1) Prediction question
→ will the house be sold within 3 months of getting listed

(2) Inferential Question
→ How accurately can we estimate the effect of each of the predictor variables on the response variable.

90| logistic Regression:

$$p(x) = \frac{e^{B_0 + B_1(x)}}{1+e^{B_0+B_1 x}}$$

Linear Regression → ordinary least square Method use করে
logistic " → Maximum likelihood " "

91) creating logistic Regression Model

- → Sklearn library use করে → preferred
- ↳ statsmodel " " "

Sklearn:

X = df[['price']] → two dimension হতে হবে

[single variable]

y = df['sold']

```
from sklearn.linear_model import LogisticRegression

clf_lrs = LogisticRegression()

clf_lrs.fit(X, y)
```

[To see β_0 & β_1 :]

clf_lrs.coef_ (β_1 পাবো)

clf_lrs.intercept_

94) multiple predictors

দুইটির ব্যবহার করে option হইতে 3 (যেমন Sold হবে কিনা, (যেমন 3 হইতে হবে কিনা, বা না হবে কিনা) But

logistic use করা যায়, But

LDA (Linear discriminant analysis use করা হয় main

(95) X = df.loc[: , df.columns != 'sold'] সবগুলো নিব

↳ sold বাদে

y = df['sold']

clf_lr = LogisticRegression()

clf_lr.fit(X, y)

clf_lr.coef_

↳ (β_1)

clf_lr.intercept_

(β_0)

(97) confusion matrix

		True default status	
		NO	Yes
Predicted default status	NO	9432	(138)
	Yes	(235)	195

138 → Type2 error (মহিলাকে pregnant না বলা) {কারো মোবাইল unlock না হওয়া} → এইক্ষেত্রে boundary value বাড়িয়ে দিতে হবে

235 ↓ type1 error {(ছেলেকে pregnant বলা)} {face unlock for anyone}

(98) creating confusion matrix:

clf_lr.predict_proba(X)

col 0 → probab of not sold
col 1 → probab of sold

by default boundary condition = 0.5

y_pred = clf_lr.predict(X)

y_pred_03 = (clf_lr.predic_proba(X)[:,1] >= 0.3)

~~from~~

accuracy (micro gen)

from sklearn.metrics import confusion_matrix

confusion_matrix(y, y_pred)

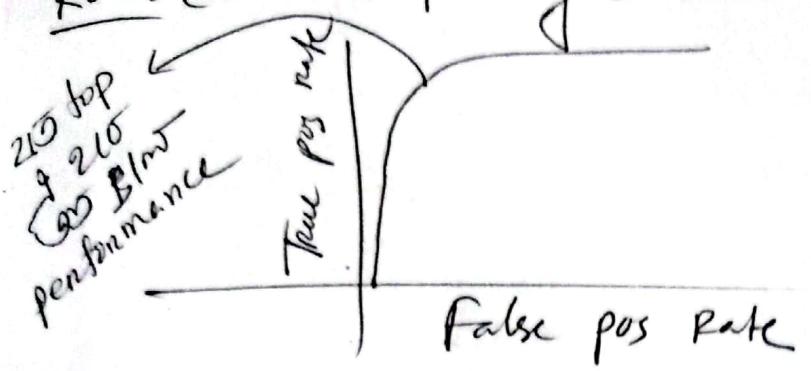
confusion_matrix(y, y_pred_03)

(99) evaluating performance

True \ Predicted	−	+
−	True Neg (TN)	False Pos (FP) → N
+	False Neg (FN)	True Pos (TP) → P

Name	def	Meanings
False pos rate	FP/N	Type I error, 1 − Specificity = TN/N
True " "	TP/P	Type II error, power, sensitivity, recall
Pos pred value	TP/P*	precision, 1 − false discovery proportion
Neg " "	TN/N*	

ROC: (Recieven operating characteristics)



(100) evaluating model performance:

$$\text{precision} = \frac{\text{True Pos}}{\text{True pos + False pos}}$$

$$\text{Recall} = \frac{\text{True Pos}}{\text{True pos + False neg}}$$

```
from sklearn.metrics import precision_score, recall_score

precision_score(y, y_pred)
recall_score(y, y_pred)

from sklearn.metrics import roc_auc_score

roc_auc_score(y, y_pred)
```

Linear discriminant analysis

Height	fit	not fit	
low	13	22	= 35
medium	15	25	= 40
high	20	5	= 25
	48	52	100

→ হিসাবে Not fit assign হবে to all classes

Bayes classifier: Assigns conditional probability & assign the class with highest probability.

কোনো একটা লোক medium Fit হওয়া সম্ভাবনা কি?

$$= 15/100 = \frac{15}{48} \times \frac{48}{100} = \frac{40}{100} \times \frac{15}{40}$$

($\frac{15}{48}$ ↑ Medium হই সম্ভাবনা; $\frac{48}{100}$ ↑ Fit হও সম্ভাবনা; $\frac{40}{100}$ ↑ medium হওয়ার; $\frac{15}{40}$ ↑ Fit হওয়ার সম্ভাবনা)

$$\Rightarrow 15/40 = \left(\frac{15}{48} \times \frac{48}{100}\right) \div \frac{40}{100}$$

Normal distribution হলে prediction result ভাল আসে।

(103) from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

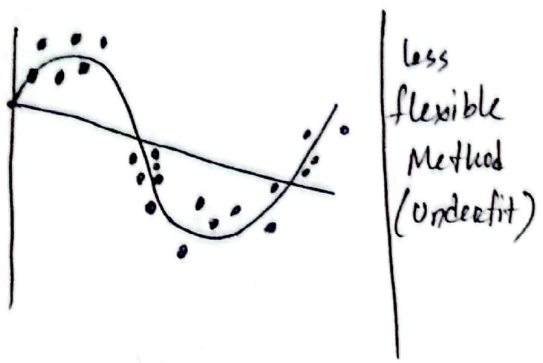
```
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(X, y)
y_pred_lda = clf_lda.predict(X)
confusion_matrix(y, y_pred_lda)
```

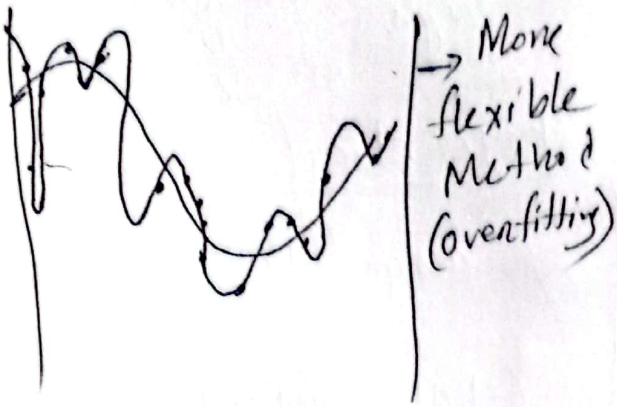
(105) K-nearest neighbors classifier

test-train split

training error → Performance of model on the previously seen data.

Test error → performance of model on the unseen data.





Split techniques

- Validation set approach
  - (80:20)
    - incase of large no of observations
- Leave one out cross validation
  - first observation keep ගලා (n-1) use කරල training
- K-fold validation
  - divide into k set
    - 1 test ට ගලා k-1 training ට ගනවා

06) from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 0)
clf_LR = LogisticRegression()
clf_LR.fit(X_train, y_train)
y_test_pred = clf_LR.predict(X_test)
from sklearn.metrices import accuracy_score, confusion_matrix
confusion_matrix(y_test, y_test_pred)
accuracy_score(y_test, y_test_pred)
```

→ $\begin{bmatrix} 36 & 22 \\ 13 & 31 \end{bmatrix}$ → actual class (rows)

predicted (coln)

8) KNN

optimum value of K choose করতে হবে।

scale matter করে। যেমন Age এবং salary. এক্ষেত্রে dif
যেমন salary 1000 diff মূল্য অনেক বেশি। But KNN
will consider this 1000 as very high. এজন্য
standardize করতে হয় independent ভাবে।

(109) from sklearn import preprocessing

scaler = preprocessing. StandardScaler().fit(X_train)

X_train_s = scaler.transform(X_train)

scaler = preprocessing.StandardScaler().fit(X_test)

X_test_s = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier

clf_knn_1 = KNeighborsClassifier(n_neighbors = 1)

clf_knn_1.fit(X_train_s, y_train)

confusion_matrix(y_test, clf_knn_1.predict(X_test_s))

accuracy_score(y_test, clf_knn_1.predict(X_test_s))

K = 3 দিয়ে আগের মত similarly বের করতে হবে।

But এক্ষেত্রে K এর মূল্য value বার বার পরিবর্তন হবে। model
create করতে হবে। এবং 1 টি model create করতে কত
সময় লাগে

~~110~~ f

(110) from sklearn.model selection import GridSearchCV

params = {'n_neighbors': [1,2,3,4,5,6,7,8,9,10]}

grid_search_cv = GridSearchCV(KNeighborsClassifier(), params)

grid_search_cv.fit(X_train_s, y_train)

{grid_search_cv.best_params_} (1-10 এর মধ্যে best parameter বের করে)

optimised_KNN = grid_search_cv.best_estimator_

y_test_pred = optimised_KNN.predict(X_test_s)

confusion_matrix(y_test, y_test_pred)

accuracy_matrix(y_test, y_test_pred)

(112) $Pr(>|z|)$ এর value যার কম সেইটা Result এ impact এ বেশি

Estimate variable যত বড় impact এ বেশি, (-) sign মানে inverse relation.

↳ এইটা Logistic Regression এর ক্ষেত্রে বেশি সত্য, KNN এর ক্ষেত্রে সত্য না।

Logistic, LDA, KNN এর comparison

Method	Accuracy
Logistic	65%
LDA	66.6%
KNN(K=3)	55%

All the steps till now

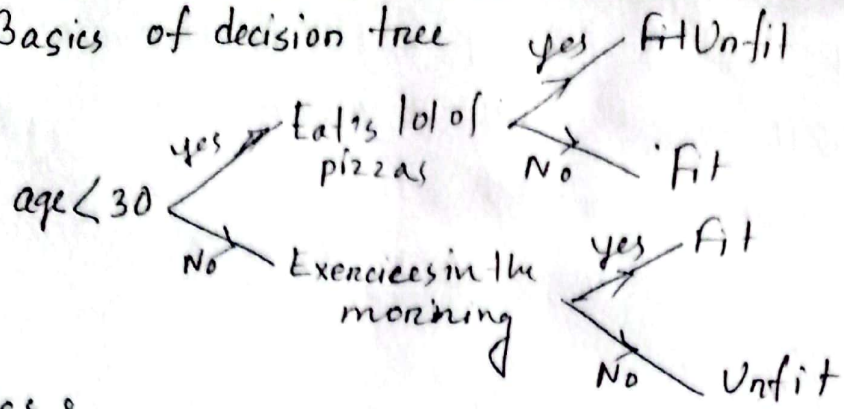
(1) Data collection (Relevant data)

(2) Pre processing
- Outlier Treatment (removal of unwanted observations)
- Missing value imputation
- Variable transformation

(3) Model training
- test, train split
- Use derived templates to train
- Different combination use करके create perform करना
- test set को apply करके Result compare करना

(4) Select best model
- For prediction purposes the model with best accuracy
- For interpretation purposes (like finding out relation between different variables) look at the coefficient values of parametric models.

(114) Basics of decision tree



Types:

- Regression tree: For continuous quantitative target variable.
  ex: Predicting rainfall, revenue, marks. etc.
- Classification tree: For discrete categorical target variables
  ex: High or low, Win or loss, healthy or unhealthy etc

(115) Regression tree

কিভাবে বা কিসের Basis এ Node এ ভাগ করব?

$$\sum_{J=1}^{J} \sum_{i \in R_j} (y_i - y_{R_j})^2$$

[এই Result এর থেকে কম হয়ে যেভাবে সেভাবে node ভাগ করব]

y_i ↑ actual, y_{R_j} ↓ Response/predicted value

RSS (Residual sum of squares)

no.	score	hours	mid
1	35	6	42
2	38	5	65
3	40	7	35
4	45	6	75
5	35	8	60
6	65	11	50
7	70	12	45
8	75	18	40
9	80	14	80
10	65	12	82

Mean 39 ↓ score

$RSS = (35-39)^2 + (38-39)^2 + \dots (35-39)^2$

Mean 75 ↓ scor

$RSS = (65-75)^2 + (70-75)^2 + \dots (85-75)^2$

এইরকম 5,5 করে (2,8), (3,7), (4,6), (6,4), (7,3) (8,2) সবগুলা combination নিয়ে RSS বের করব যেটা সবথেকে কম RSS হবে সেইটা নিতে হবে.

এখানে এই mean টা score কে না দিয়ে Midterm ও
ও দেওয়া যাইতো। আরো variable থাকলে তাদের দিয়ে ও দেওয়া
তাই এর জন্য একটা approach গ্রহণ করা হয়।

- top down, greedy approach that is known as recursive binary splitting.
  - top down কারণ root থেকে
  - binary কারণ প্রতি node-এ দুইটা করে ভাগ হয়
  - greedy কারণ future না দেখে শুধু বর্তমানে যে node এ আছে সেই node এর জন্য best split choose করে।

কতক্ষণ পর্যন্ত ~~Node~~ leap এ ভাগ করবে তা depend করে stopping Criteria এর উপর যেটা আমাদের দিয়ে দিতে হবে।

(116) Stopping Criteria

- There should be a minimum number of observation (Data আকারে) required for further split.
- There should be a minimum number of observation needed at each node after spliting.
- There should be maximum layers of tree (or depth), possible.

(117) ~~used~~ required Data set (movie budget/collection in box office)

(118) Regression tree code.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df=pd.read_csv("Movie-regression.csv", header=0)
df.head()
df.info()
```

short cut
- Shift + enter: current cell run + next cell ready
- Alt + enter: current cell run + insert empty cell
- shift tab দিলে parameter দেখায়
- Read_csv এর জন্য forward slash

. . . .fitUntil

(120) df.info() দিয়ে দেখা গেছে time take এ 506 এর জায়গায় ...
Data আছে। বাকি 12 টা "missing value impute" করতে হবে। mean ...
or mode or zero যেকোনো কিছু দিয়ে করা যায়।

```
t_mean=df['Time_taken'].mean()
df['Time_taken'].fillna(value = t.mean), inplace = True)
df.info()
```

এখন data store হবে permanently, false হলে শুধু show হবে save হবে না।

(121) Categorical (3D, movie_genre) কে numerical এ transform করতে হবে।

```
df = pd.get_dumies(df, columns = ["3D_available", "Genre"],
                   drop_first = True)
df.head()
```

[n-1 কলামে first টা বাদ দিবে]

(122) X-y split করতে হবে
collection বাদে সবই X. → all rows → all except collection

```
X = df.loc[:, df.columns != "collection"]
X.shape()
y = df["collection"]
y.head()/shape()
```

(123) //Test_train split করা

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, testsize = 0.2,
                                                    random state = 0)
```

constant 0/1/2 রাখলে প্রতিবার same data আসবে

(125) //creating decision tree

```
from sklearn import tree
regtree = tree.DecisionTreeRegressor(max_depth = 3)
regtree.fit(X_train, y_train)
```

```
y_train_pred = regtree.predict(x_train)
y_test_pred =    "   .   "   (x_test)
```

127) // evaluation

```
from sklearn.metrices import mean_squared_error, r2_score

mean_squared_error(y_test, y_test_pred)
r2_score(y_train, y_train_pred)
r2_score(y_test, y_test_pred)
```

128) How to plot decision tree //

- → create dot file
- → convert dot file to image
- → use the image to graph

```
dot_data = tree.export_graphviz(regtree, out_file = none)
from Ipython.display import image
import pydotplus
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

এইটা install থাকে না, install করতে anaconda command prompt → conda install pydotplus

cmd থেকে করতে চাইলে → pip install pydotplus

129) Max depth এর পরেও better solution থাকতে পারে। সেইজন্য tree pruning করা হয়।

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \quad \} RSS$$

Here
- α = Tuning parameter
- T = number of leaf nodes

এখান থেকে α এর মান বের করে যে ভ্যালুগুলো যে RSS পাওয়া যাবে সেইটা Best Regression tree হবে।

130 ✱ Controlling tree growth

Shortcut ✱ Shift + tab 3 times
✱ + tab 1 time

(i) Maximum number of levels in tree

```
regtree1 = tree.DecisionTreeRegressor(max_depth=3)
regtree1.fit(X_train, y_train)
dot_data = tree.export_graphviz(regtree1, outfile=None,
                                feature_names=X_train.columns, filled=True)
graph1 = pydotplus.graph_from_dot_data(dot_data)
Image(graph1.create_png())
```

(ii) minimum observation at internal node

minimum 10 বা sample নাম্বার ↑ 20

```
regtree2 = tree.DecisionTreeRegressor(min_samples_split=10)
```

..... বাকি Same.

(iii) minimum observations at leaf node

```
regtree3 = tree.DecisionTreeRegressor(min_samples_leaf=25)
```

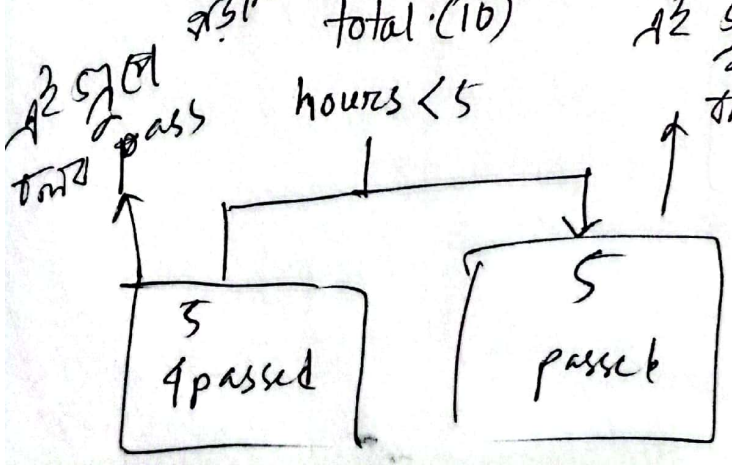
.... বাকি Same.

যে কয়টি condition এ প্রয়োগ apply করতে চাও তা দেবে

```
regtreeM = tree.DecisionTreeRegressor(min_samples_leaf=25,
                                      max_depth=4)
```

132 Classification Tree:

Regression এ mean use করা হয় কোন কারণে mode.



-ইখানেও Recursive. binary splitting use হয়। But
হয় না। ৩টি জিনিস use হয়:-
→ Classification error rate → আগের example এ বাকিগুলো ১ জন এবং বাকিগুলো ২ জন total $\frac{3}{10} \times 100 = 30\%$ misclassified. error = 30%.
→ Gini index
→ Cross Entropy

Gini index । বা ০ দ্বারা বুঝায় pure class. কোনো নেই।

(133) star_tech_oscar পাবে কিনা সেইটা predict করব।

(134) //code:

```
    import numpy as np
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt
df = pd.read_csv("Movie_classification.csv", header = 0)
df.head() / info()
//missing value imputation.
df['Time_taken'].fillna(value = df['Time_taken'].mean(), inplace
                                                        = True)

//dummy
df = pd.get_dummies(df, columns = ["3D_available", "Genre"],
                                      drop first = True)
//xy split
X = df.loc[:, df.columns != "start_Tech_Oscar"]
y = df["start_Tech_Oscar"]

// Test train split
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
                                                    random_state = 0)
//training
from sklearn import tree
clftree = tree.DecisionTreeClassifier(max_depth = 3)
clftree.fit(X_train, y_train)
// predicting values
y_train_pred = clf_tree.predict(X_train)
y_test_pred = clf_tree.predict(X_test)
//Model performance
from sklearn.metrics import accuracy_score, confusion_matrix
confusion_matrix(y_train, y_train_pred)
confusion_matrix(y_test, y_test_pred)
accuracy_score(y_test, y_test_pred)
//Ploting decision tree
dot_data = tree.export_graphviz(clftree, out_file=None, filled=True,
                                feature_name = X_train.columns)
from IPython.display import Image
import pydotplus
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
// controlling tree growth (similarly)
clftree2 = tree.DecisionTreeClassifier(min_samples_leaf = 20,
                                       max_depth = 9)
---- 같은 same (Repeat)
```

(137) ad & disad of decision tree

advantage
- easy to explain
- more human like
- graphically show
- dummy create করা ছাড়াই qualitative data handle prediction handle করতে পারে।

disad
- same level of predictive accuracy প্রদান না।

(138) **Ensemble Methods**
- Bagging
- Random forest
- Boosting

decision tree have high variance কারণ একটা Data set কে দুইভাগে ভাগ করলে দুই রকম result পাবো,

একটা training set এ একটা model use না করে যদি সেইটাকে আলাদা করে তারপর ভিন্ন ভিন্ন model apply করে সেইগুলোর average করা হয় তাহলে accuracy বাড়ে। এইটাকে ~~ensemble~~ Bagging method বলে।

~~train~~ data set কে বিভিন্ন পার্টে ভাগ করা হল Bootstrapping এর Randomly Data pick করে add করতে পারে।
same training set থেকে বার বার same নিতে পারে।

Bagging
- pruning is not done, full length trees are grown
- individual trees have high variance & low bias, averaging reduces the variance.
- In regression, we take the average of predicted values.
- In case of classification, we take majority vote i.e. most predicted class will be taken as the final prediction

(139) // Bagging code (classification)

```
from sklearn import tree
clf_tree = tree.DecisionTreeClassifier()
from sklearn.ensemble import BaggingClassifier
bag_clf = BaggingClassifier(base_estimator = clf_tree, n_estimator = 1000,
                            bootstrap = True, n_jobs = -1, random_state = 42)
bag_clf.fit(X_train, y_train)
confusion_matrix(y_test, bag_clf.predict(X_test))
accuracy_score(y_test, u)
```

subset with replacement

বিভিন্ন ধরণের Base মডেল

বড় ডাটা সেট

multicore process ও best performance

(141) // Random forest

Bagging create correlated trees

→ improvement over bagging

→ আমাদের যত গুলো predictor variable (p) আছে তার থেকে কিছু সংখ্যক Random ভাবে select করা হবে,

→ কতগুলো Predictor value ব্যবহার করা হবে?

→ for Regression, $M = p/3$ → predictor variable total

→ for Classification, $M = \sqrt{p}$

→ If the variables are highly correlated, smaller value of M should be used.

(142) # ~~Rado~~ Random forest Code (Classification)

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators = 1000, n_jobs = -1,
                                random_state = 42)
rf_clf.fit(X_train, y_train)
confusion_matrix(y_test, rf_clf.predict(X_test))
accuracy_score(y_test, rf_clf.predict(X_test))
```

(143) Optimising Random Forest

```
from sklearn.model_selection import GridSearchCV
rf_clf = RandomForestClassifier(n_estimators=250, random_state=42)
params_grid = {"max_features": [4,5,6,7,8,9,10],
               "min_samples_split":[2,3,10]}
grid_search = GridSearchCV(rf_clf, params_grid, n_jobs=-1, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
grid_search.best_params_
cvrf_clf = grid_search.best_estimator_
accuracy_score(y_test, cvrf_clf.predict(X_test))
```

cross validation

max: 7, min: 3

(145) Boosting

↳ each tree is grown from using information of previously grown tree.

Three boosting

- Gradient Boost → add করে দ্বিতীয় tree, এভাবে n সংখ্যক tree.
- Ada Boost
- XG Boost → similar as gradient but better perform because regularization is used.

Regularization:

The cost function we are trying to optimize (MSE in regression etc) also contains a penalty term for number of variables. In a way, we want to minimize the number of variables in final model along with the MSE or accuracy. This helps in avoiding overfitting.

(146) // Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gbc_clf = GradientBoostingClassifier()
gbc_clf.fit(X_train, y_train)
accuracy_score(y_test, gbc_clf.predict(X_test))
gbc_clf2 = GradientBoostingClassifier(learning_rate = 0.02, max_depth=1,
                                      n_estimators = 1000).
accuracy_score(y_train, gbc_clf2.predict(X_train))
v          v (y_test, . . . . . (X_test))
```

(148) // Ada Boosting

```
from sklearn.ensemble import AdaBoostClassifier
ada_clf = AdaBoostClassifier(learning_rate=0.02, n_estimators=5000)
ada_clf.fit(X_train, y_train)
accuracy_score(y_train, ada_clf.fit(X_train))
accuracy_score(y_test, . . . . (X_test))
ada_clf2 = AdaBoostClassifier(rf_clf, learning_rate=0.05, n_estimators
                              =500)
ada_clf2.fit(X_train, y_train)
accuracy_score(y_test, ada_clf2.predict(X_test))
```

(150) // XGBoost

First XGBoost install করতে হবে। তারপর সেটা করতে cmd তে।

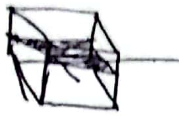
`pip install xgboost` anconda তে করতে চাইলে conda install xgboost

```
import xgboost as xgb
xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=10000
                            learning_rate=0.3, n_jobs=-1)
xgb_clf.fit(X_train, y_train)
```

```
accuracy_score(y_test, xgb_clf.predict(X_test))
xgb.plot_importance(xgb_clf)
// Grid search use xgb optimise
xgb_clf = xgb.XGBClassifier(n_estimator=250, learning_rate=0.1,
                            random_state=42)
param_test1 = {
    'max_depth': range(3,10,2),
    'gamma' : [0.1, 0.2, 0.3],
    'subsample': [0.8, 0.9],
    'colsample_bytree': [0.8, 0.9],
    'reg_alpha': [1e^-2, 0.1, 1]
    }
grid_search = GridSearchCV(xgb_clf, param_test1, CV=5, n_jobs=-1,
                           scoring='accuracy')
grid_search.fit(X_train, y_train)
cvxg_clf = grid_search.best_estimator_
accuracy_score(y_test, cvxg_clf.predict(X_test))
grid_search.best_params_
```

52) // Out of Box Machine learning tehniques (Support Vector Machi

- → Maximal Margin Classifier [Support vector machine is a generalization of this classifier]
  - ↓ Classes are linearly separable
- → Support vector classifiers
- → " " Machine

Hyperplane :- plane that divides P dimensional space into two parts
- For one 1 D space →  point দিয়ে Divide
- For 2D space →  Line দিয়ে divide
- " 3D space →  plane দিয়ে divide

So there is A (P-1) plane that divide P space into two parts.

predictor variable = 2D/3D/20D space

54) we have to find the hyperplane that divides this spaces.

Data যদি perfectly separable হয় তাহলে infinite no of hyperplane থাকবে।
- Calculate the perpendicular distance of observations from hyperplane
- Minimum value of distance is called Margin.
- Choose the hyperplane with maximum value of margin

Support Vectors
- The observation which fall on margin are known as support vectors
- These classifiers depends on support vectors only.
- That is why this technique is different from conventional ML

55) Limitation
- If the two classes are not separable by a hyperplane Then Maximal Margin Classifier can't be used.
- very sensitive to points near to the line (support vectors)

56) Support Vector Classifier
- soft margin classifier (কিছু point হয়তো ভুল দিকে যাবে, কিছু সঠিক দিকে থাকবে)
- কিছু কিছু error allow করে। misclassified [illegible] correct classified but on the wrong side.

Steps:
- We create a misclassification budget (B)
- wrong side এ থাকা point এর distance এর sum limit করে such that $(x_1 + x_2 + x_3 + \dots) < B$

→ try to maximize margin while trying to stay within budget
→ Software package এ C use করা হয়। $C \propto \frac{1}{B}$.

Impact of C
→ when C is small, margins will be wide & there will be many support vectors & many misclassified observations.
→ When C is large, margins will be narrow & there will be fewer support vectors & fewer misclassified values.
→ low cost value prevents overfitting & may give better test set performance
→ Optimal value of C find করে use করতে হয়।

↓
https://cs.stanford.edu/~karpathy/svmjs/demo/

(157) Limitation
→ ok it works when support vector classifier is linearly separable
→ 1st overcome করার জন্য এটি SVM. যেখানে non linear separator ব্যবহার করা।

(158) Kernel Based Support Vector Machines
→ it is an extension of SVC which uses kernel to create non linear boundaries

Kernels:
→ It is some functional relationship between two observations.
→ Linear → $K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$
→ polynomial → $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$
→ Radial → $K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$

↓
Gamma defines how much influence a single training example has. The larger the gamma is, the closer other examples must be to be affected.

// Code (164) Regression Tree ಗೆ ಯಾವುದೇ ಬದಲಾವಣೆ ಇಲ್ಲ

(165) // SVM based Regression Model

```
from sklearn.svm import SVR
svr = SVR(kernel='linear', C=1000)
svr.fit(X_train_std, y_train)
y_test_pred = svr.predict(X_test_std)
y_train_pred = svr.predict(X_train_std)

from sklearn.metrics import mean_squared_error, r2_score
mean_squared_error(y_test, y_test_pred)
r2_score(y_train, y_train_pred)
r2_score(y_test, y_test_pred)
```

[r2 value ranges between -1 & 1. -1=1 - perfect fit
0 - not fit, should be 0.5 & 0.7]

166-167 → ಯಾವುದೇ ಬದಲಾವಣೆ ಇಲ್ಲ

(168) Standardizing data

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler().fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_test_std
```

(169) // SVM based Classification Model

```
from sklearn. import svm
clf_svm_1 = svm.SVC(kernel='linear', C=0.01)
clf_svm_1.fit(X_train_std, y_train)
y_train_pred = clf_svm_1.predict(X_train_std)
y_test_pred = " . " (X_test_std)
from sklearn.metrics import accuracy_score, confusion_Matrix
```

```
confusion_matrix (y_test, y_test_pred)
accuracy_score (  "  ,  "  )
clf_svm_l . n_support_
```
{ দুইটা element show করে। প্রথমটা ১ম class এর support vector, 2nd টা 2nd class এর }

(170) // Hyper Parameter Tuning

```
from sklearn.model_selection import GridSearchCV
params = {'C':(0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000)}
clf_svm_l = svm.SVC( kernel = 'linear')     (poly struck out)
svm_grid_lin = GridSearchCV (clf_svm_l, params, n_jobs = -1, cv = 10,
                  scoring = 'accuracy', verbose = 1)
                                               ↓
                                        no msg we get
svm_grid_lin.fit (X_train_std, y_train)
svm_grid_lin.best_param_
linsvm_clf = svm_grid_lin.best_estimator_
accuracy_score( y_test, linsvm_clf.predict (X_test_std))
```

(171) // SVM class with polynomial kernel.

```
clf_svm_p3 = svm.SVC( kernel = 'poly', degree = 2, C = 0.1)
clf_svm_p3 . fit (X_train_std, y_train)
y_train_pred = clf_svm_p3.predict (X_train_std)
y_test_pred = clf_svm_p3. predict (X_test_std)
accuracy_score (y_test, y_test_pred)
clf_svm_p3.n_support_
```

(172) // SVM class with Radial

```
clf_svm_r = svm. SVC (kernel = 'rbf', gamma = 0.5, C = 10)
. . . . .
. . . . .    } Same as before
. . .
clf_svm_r. n_support_
```

Radial Grid

```
params = {'C': (0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50), 'gamma': (0.001, 0.01, 0.1, 0.5, 1)}
clf_svm_r = svm.SVC(kernel='rbf')
svm_grid_rad = GridSearchCV(clf_svm_r, params, n_jobs=-1, cv=3,
                            verbose=1, scoring='accuracy')
svm_grid_rad.fit(X_train_std, y_train)
svm_grid_rad.best_param_
svm_grid_rad svm_clf = svm_grid_rad.best_estimator_
accuracy_score(y_test, rad svm_clf.predict(X_test_std))
```

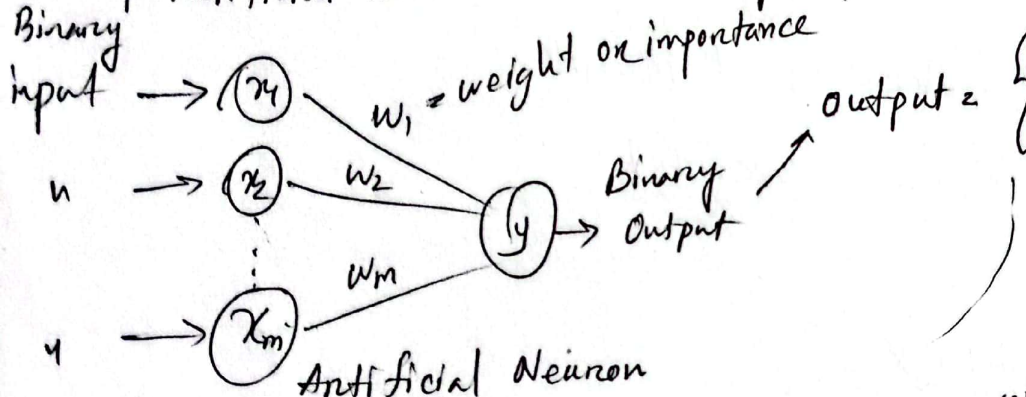
(181) Introduction to deep learning

Dataset

The dataset used here is Fashion MNIST

(182) Perceptron

↳ Artificial neuron are called perceptron.



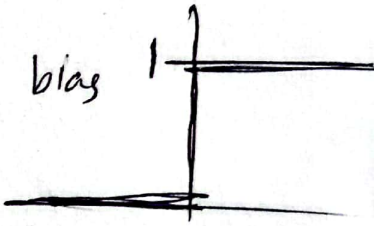
$$\text{output} = \begin{cases} 0 & \text{if } \sum_{j=0}^{M} w_j x_j < \text{thresh} \\ 1 & \text{if } \sum_{j=0}^{M} w_j x_j > \text{thresh} \end{cases}$$

(183) Activation functions

$$\text{output} = \begin{cases} 0 & \sum_{j=0}^{M} w_j x_j + b < 0 \\ 1 & \sum_{j=0}^{M} w_j x_j + b \geq 0 \end{cases}$$

$-\text{threshold} = b$ → more sensitive to individual observ

b is called the bias



↳ step activation function

There are different kind of activation function.

↳ step

↳ Sigmoid

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

$$\text{output} = \frac{1}{1+\exp(-\sum w_j x_j - b)}$$

↳ All neuron with sigmoid function called sigmoid or logistic neuron

184 //Creating Single Perceptron Model

প্রথম sklearn install করা লাগবে

pip install -U scikit-learn

অথবা conda দিয়ে scikit-learn

```
from sklearn.datasets import load_iris
iris = load_iris()
```

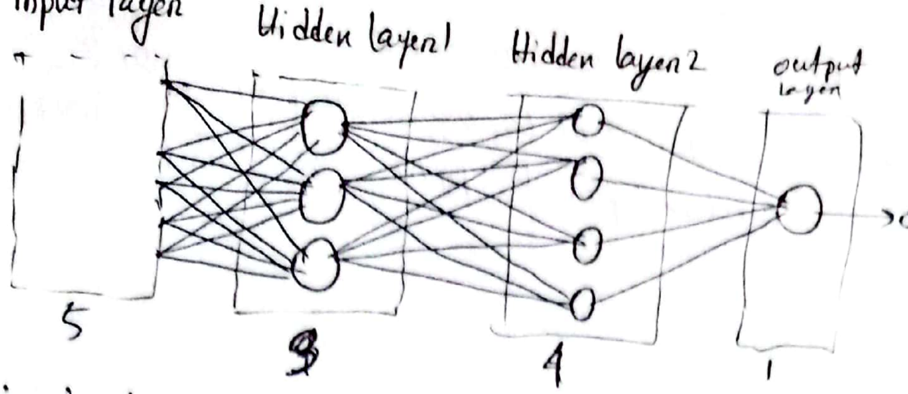
//petal length & Petal width নিয়ে Depend করে iris-type anticipate করব. setosa কিনা বা

```
X = iris.data[:,(2,3)]
iris.target
// lets convert the target variable
y = (iris.target == 0)   # bool এ করে
y = (  "  ).astype(np.int)  # int এ করে
from sklearn.linear_model import Perceptron
perc_clf = Perceptron(random_state=42)
per_clf.fit(X,y)
y_pred = perclf.predict(X)
from sklearn.metrics import accuracy_score
accuracy_score(y, y_pred)
per_clf.coef_
per_clf.interept_
```

185 // Making networks

two ways to stack cells

- parallel
- sequential



} 5-3-4-1 Network

This is also called feed forward Network → one directional processing
যদি কোন layer এর এক perceptron থেকে অন্য perceptron এ input আসে তাহলে তাকে cyclic Network বলে। Recurrent Neural Network (RNN) is an example. (NLP[উদাহরণ])
Fully connected Network- এক layer এর সকল input Next layer এ।

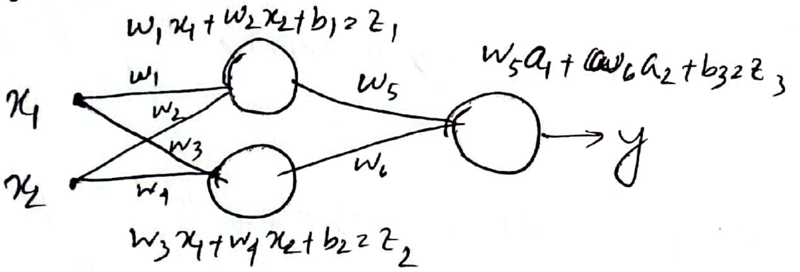
(186) //How neural network work

Sigmoid function, $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-\sum w_j x_j - b)}$

↓ weights ↳ bias

problem statement:

- Establish the values of weights & biases so that predicted output is as close to actual output as possible.

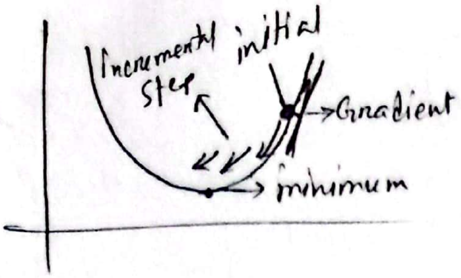


Variables to be established
weights: w_1 ... w_6
biases: b_1, b_2, b_3.

Gradient descent: the values of w's & b's.
The method which is used to find minimum of a function.
↳ it is an optimization technique to find
↳ better than OLS (ordinary least square) as we have large no. of features & complex relationship.

Steps for gradient descent
↳ Assign Random w & B values [initialization]
↳ calculate final output using these values [forward propagation]
↳ Estimate error using error function
↳ find those w & B which can reduce this error [Backward propagation]
↳ Update w & B & repeat → [implementation of GD]

fig: 1



যদি $y = f(x)$ একটা Relation জানা থাকতো তাহলে calculus use করে minimum পাওয়া যাইতো যেহেতু জানা নাই তাই একটা random point থেকে step by step min এ যাচ্ছে

(187) // Back propagation

let say,
assume predicted output $= 0.3$, actual output $= 0$

Distance $= 0 - 0.3 = -0.3$

error function, $= |-0.3| = 0.3$

" " $= (-0.3)^2 = 0.09$

square function works well with regression but not with classification.

So we use, Cross Entropy Error function $= -y \log(y') - (1-y)\log(1-y')$
↓
This looks like fig: 1 which is necessary for gradient descent

Here, y = actual value, y' = predicted value

$CEEF = -y\log(y') - (1-y)\log(1-y')$

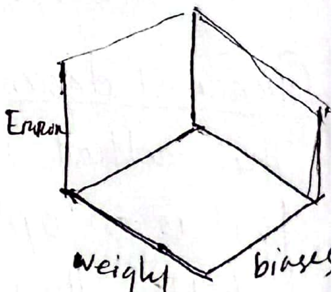
if $y = 0$ then $CEEF = -(1-y)\log(1-y')$ ⇒ y' কে হবে as close to 0 as possible.

if $y = 1$ then, $CEEF = -\log(y')$

to minimize this, we have to minimize $-\log(y')$ ~~or~~ maximize $\log(y')$ ⇒ maximize y'

Since y' lies between 0 & 1. So y' should be as close to 1 as possible

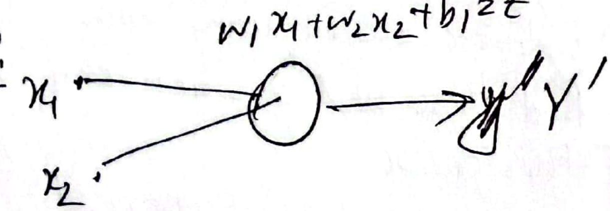
step 4: $w = w - \alpha \Delta w$
$b = b - \alpha \Delta b$



α is learning rate, Δw & Δb are unit steps.
α determines the number of steps we take.

Δw & Δb কিভাবে পাব?
through back propagation

example:



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

Error function
$= -y\log(y') - (1-y)\log(1-y')$

w_1	w_2	b
2	3	-4

→ Random

Step 2: forward propagation

example data (training data)

x_1	x_2	y → Actual output
10	-4	1

$z = w_1x_1 + w_2x_2 + b$

$z = 4, \quad \sigma(z) = 0.982$

Step 3: (Error Calculation)

y'	y
0.982	1

$\{ -y\log(y') - (1-y)\log(1-y') \}$

$E = 0.0079$

Step 4: Back propagation

$\frac{\partial E}{\partial y'}\Big|_{y=1} = -\frac{\partial}{\partial y'}\log(y') = -\frac{1}{y'}$

$\frac{\partial y'}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{(1+e^{-z})^2}$

$\frac{\partial z}{\partial w_1} = x_1 = 10, \quad \frac{\partial z}{\partial w_2} = x_2 = -4, \quad \frac{\partial z}{\partial b} = 1$

Now, $\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y'} \times \frac{\partial y'}{\partial z} \times \frac{\partial z}{\partial w_1} = -0.182$

Similarly, $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y'} \times \frac{\partial y'}{\partial z} \times \frac{\partial z}{\partial w_2} = 0.0746$

& $\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y'} \times \frac{\partial y'}{\partial z} \times \frac{\partial z}{\partial b} = -0.018$

Step 5: updating w & b α (learning rate)

$w_1 = w_1 - \alpha \Delta w_1 = 2 - 5 \times (-0.182) = 2.93$

$w_2 = w_2 - \alpha \Delta w_2 = 3 - 5 \times 0.0746 = 2.627$

$b = b - \alpha \Delta b = -4 - 5 \times (-0.0186) = -3.907$

Step 6: Repeat

x_1	x_2	y
10	-4	1

$z = 2.9 \times 10 + 2.6 \times (-4) + (-3.9) = 14.7$

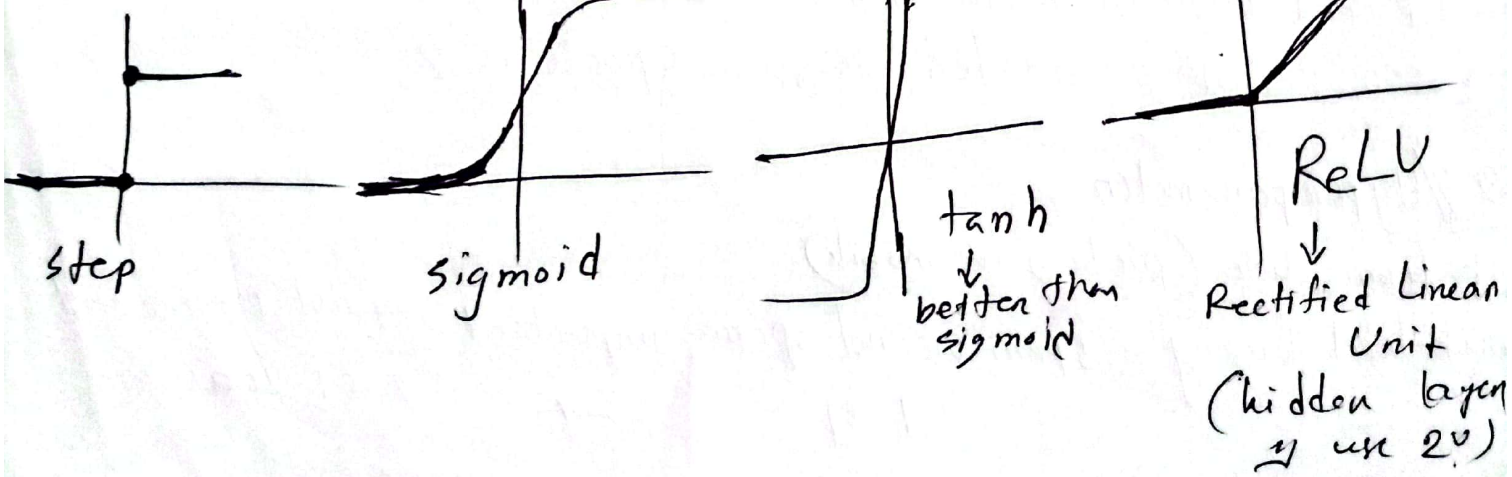
$\sigma(z) = 0.999 = y'$

(188) // some important concepts

Q: Why do we use activation function?

Ans: to put special boundary condition, to get non linear & complex patterns. Otherwise output would be, $z = w_1x_1 + w_2x_2 + b$

Q: What are the different types of activation function?

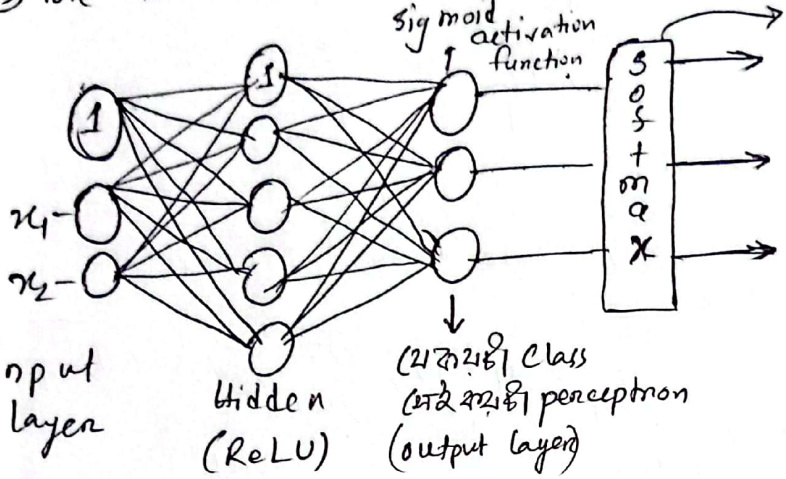


Q: What is multiclass classification? Is there any specific activation function for this?

Ans: ① two classes like 'yes' or 'No', এইরকম Binary classification

② More than 2 classes like 'shirts', 'trousers', 'socks' এইরকম multiclass ।

③ for multiclass we use softmax activation.



→ 3 layer এ output সংখ্যা total sum দিয়ে divide করে probability পাওয়া যায়।

Q: What is the difference between Gradient descent & stochastic gradient descent

→ stochastic :→ each individual training record দিয়ে forward & backward propagation করে w & b এর মান update করা হয়, fast but converge slow & bad.

→ gradient :→ full training set এর record দিয়ে একবারে update করা হয়।
↳ slow but converges well.

→ Mini batch gradient :→ small batch of training set দিয়ে " " ।

Q: What is an Epoch

→ Epoch is one cycle through the full training data. Epoch = 5 means we want the full training data to be fed 5 times. different from iteration.

→ idea behind it is, is to reject its performance on the same data.

→ 1 Record একবার input দিলাম 1000 Record দিয়ে 1000 iterations.

→ 1000 Record 2 বার fed করলাম epoch is 2.

89 // Hyperparameter

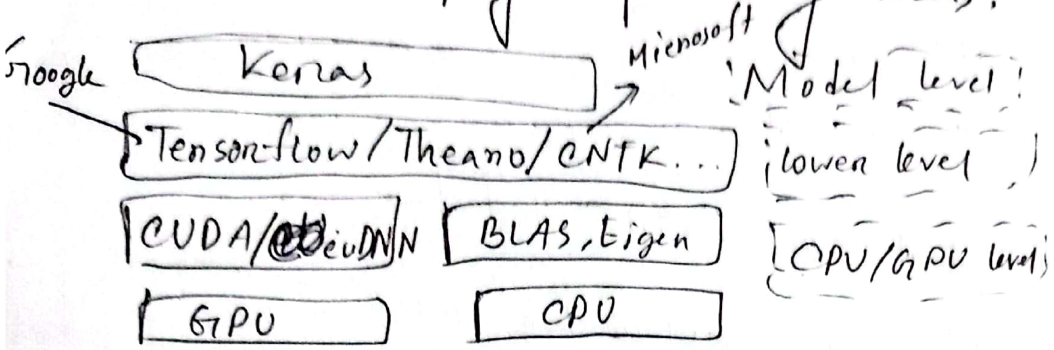
→ from slide (picture in mobile)

multilabel binary = spam or not spam, important or not important

1 টা + 1 টা neuron

= 2 টা Neuron

Keras is a model level library, providing high level building blocks for developing deep learning models.



(191) // Installing tensorflow & Keras

```
import numpy as np
import pandas as pd
% matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
//tensorflow install
conda install tensorflow
conda    u    pip
pip install --upgrade tensorflow==2.0.0-rc1
import tensorflow as tf
from tensorflow import keras
```

```
Keras.__version__
tf.__version__
```

(192) // Dataset for classification

// problem statement & dataset in the mobile. 28 X 28 grayscale image of pant, shirt etc (total 10 things). total 70000 dataset, 60000 train, 10000 test

```
//import dataset
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
plt.imshow(X_train_full[0])  // showing first image
y_train_full[1]  // category লেবেল (০-৯ এর 10)
```

// category 0-9 এর বদলে নাম দেখাতে চাইলে list create করতে হবে।

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

class_names [y_train_full [1]]

X_train_full [10] // 28×28 pixel value show করবে

193 // Normalization (যাতে Gradient descent use করতে) & Test-train Split

X_train_n = X_train_full / 255.0 [// আমরা data গুলো 0 থেকে 1 এর মধ্যে
X_test_n = X_test / 255.0 করব তাহলে 255.0 কে floating করে]

// training set কে আবার training + validation এ ভাগ করতে হবে (X, y দুটোতেই)

X_valid, X_train = X_train_n [:5000], X_train_n [5000:]

y_valid, y_train = ~~y_train~~ y_train_n [:5000], y_train_n [5000:]

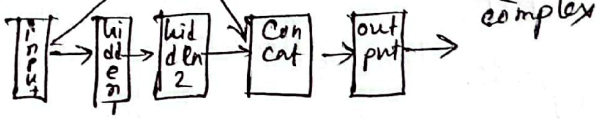
X_test = X_test_n

194 // Different ways to create ANN using Keras

There are two APIs for defining a model in keras

↳ Sequential model API → simple layer by layer

↳ functional API →



195 // Building the neural network using Keras

np.random.seed(42) → Same as random state

tf.random.set_seed(42)

// sequential model API

model = keras.models.Sequential()

model.add(keras.layers.Flatten(input_shape=[28,28])) ← no. of neurons, output 10 টা category এর জন্য

model.add(keras.layers.Dense(300, activation="relu"))

model.add(keras.layers.Dense(100, activation="relu"))

model.add(keras.layers.Dense(10, activation="softmax"))

model.summary()

import pydot → // install না হলে pip install pydot / conda install pydot

keras.utils.plot_model(model)

[28×28 pixel]
↓
input layer
↓
Hidden u 1 } ReLu Activation
↓
Hidden u 2
↓
output layer } softmax activation

[28×28 (2D) থেকে 1D তে আনতাম]

weights, biases = model.layers[1].get_weights()
weights.shape
biases
biases.shape

(196) // Compiling & training the neural network model
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
model_history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))
model_history.params
model_history.history
// plotting
import pandas as pd
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()

(197) // Evaluating performance & ~~pred~~ predicting using Keras
// Evaluate
model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_proba = model.predict(X_new) } probability score assigned to each class.
y_proba.round(2) } round of to 2 digit
y_pred = model.predict_classes(X_new) } predicting the class
y_pred
np.array(class_names)[y_pred]
// checking if the prediction were correct or not
print(plt.imshow(X_test[0]))
print(plt.imshow(X_test[1]))
print(plt.imshow(X_test[2]))

// Dataset that will be used is California housing
// objective here is to predict the price of house using 8 indep

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt.
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
```

// Data set characterstics

Number of instances: 20640
" " attributes: 8 numeric, predictive attributes & the target

Attribute info:

- MedInc → median income in block
- HouseAge → " house age in block
- AveRooms → average no. of rooms.
- AveBedrms → " no. of bedrooms.
- Population → Block population
- AveOccup → average house Occupancy
- ~~Latt~~ Latitude → " " block Latitude
- Longitude → " " longitude

Target: the target variable is the median house value in units of 100,000 for California districts.

// code

```
print(housing.feature_names)
from sklearn.model_selection import train_test_split
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data,
                                                housing.target
                                                random_state=42
```

independent
dependent

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full, random_state=...)
```

// Standardizing the data → { (data - mean) / variance = Standardized data }

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
np.random.seed(42)
tf.random.set_seed(42)
X_train.shape
model = keras.models.Sequential([
        keras.layers.Dense(30, activation="relu", input_shape=[8]),
        keras.layers.Dense(30,     "     = "relu"),
        "     .    "     .    "    (1)
  ])
```

input_shape = X_train.shape[1:] এইভাবেও দিতে হয়।

এখান দিয়ে count করা শুরু হয়।

```
model.summary()
model.compile(loss="mean_squared_error",
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['mae'])
```

learning Rate default 0.01

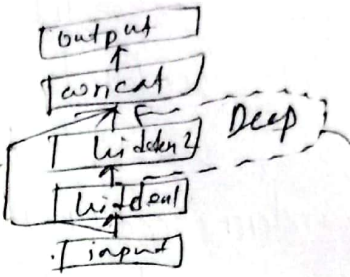
```
model_history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_valid))
mae_test = model.evaluate(X_test, y_test)
model_history.history
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```

[এই graph থেকে দেখে যদি বুঝা যায় যে converge করে নাই তাহলে। এই code আবার Run করলেই 40 epochs হবে]

// predict the value in new dataset

```
X_new = X_test[:3]
y_pred = model.predict(X_new)
print(y_pred)
print(y_test[:3])
```

(199) // Using functional API for complex Architectures




```
# to delete the previous model
del model
keras.backend.clear_session()
input_ = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = ".".".(30, ".".")(hidden1)
concat = ".".".concatenate([input_, hidden2])
output = ".".".Dense(1)(concat)
model = keras.models.Model(inputs=[input_], outputs=[output])
model.summary()
model.compile(loss="mean_squared_error",
              optimizer=keras.optimizers.SGD(lr=1e-3),
              metrics=['mae'])
model_history = model.fit(X_train, y_train, epochs=40, validation_data=
                          (X_valid, y_valid))
mae_test = model.evaluate(X_test, y_test)
mse_test = ".".(".",".")
model_history.history
pd.DataFrame(model_history.history).plot(figsize=(8,5))
plt.grid(true)
plt.gca().set_ylim(0,1)
plt.show()
```

mean absolute error

12 units bigger than hidden1
& add last one

Saving Models & Using Callbacks

model.save("my_func_model.h5") → name i want to give

%pwd → to check the present working directory

%cd C:\Users\pukhr → for changing directory

del model → deleting

keras.backend.clear_session()

model = keras.models.load_model("my_func_model.h5")

model.summary() → same model ২ load হচ্ছে।

y_predic = ~~modelpredi~~ model.predict // কাজ

// Data train করতে ৮-২০ ঘন্টা লাগতে পারে। এইজন্য প্রতি epoch এর পর save করে রাখলে সুবিধা হয়। তাই এ কাজটি করা

model.compile(.....) এর নিচে checkpoint দিতে হবে।

variable file name ex: first like = Model-01.h5

```
checkpoint_cb = keras.callbacks.ModelCheckpoint("Model-{epoch:02d}.h5")
                                                  → two digit
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])
```

```
del model
keras.backend.clear_session()
model = keras.models.load_model("Model-10.h5")
mse_test = model.evaluate(X_test, y_test)
```

// সবগুলো Model Save করা unnecessary. শুধুমাত্র Best model টা রাখলেই হবে এটাও করা যায়।

```
del model
keras.backend.clear_session()
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)])
```

```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(
checkpoint_cb = keras.callbacks.ModelCheckpoint("Best_Model.h5",
                                                 save_best_only=True)

history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])

model = keras.models.load_model("Best_Model.h5")
mse_test = model.evaluate(X_test, y_test)
```

// 10/20 epoch দিলে Best result নাও পাইতে পারি, But 200/300 দিলে আবার বেশি ও হয়ে যাইতে পারে। তাই best result পাওয়ার পর stop করে দিতে হবে। callback লাগবে, দুইটার একটা save best callback এর সাথে আরেকটা এটা।

```
del model
keras.backend.clear_session()
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)])
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))
checkpoint_cb = keras.callbacks.ModelCheckpoint("early_stop_model.h5",
                                                 save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                                                  restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=200,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb])

model = keras.models.load_model("early_stop_model.h5")
mse_test = model.evaluate(X_test, y_test)
```

(patience: no of epochs পর পরেও যদি improve না হয় তাহলে model stop হয়ে যাবে।)

- Number of Layers → most of problems single layer is sufficient. But going deeper increases parameter efficiency. 1st or 2nd layer fine-tune model এবং transfer learning.
- " " neurons per layer
- Type of activation function
- Learning rate
- Weight initialization logic etc

Flexibility of ANN is also a drawback

Transfer learning: কোন একটা model আছে যেটা already face recognition এর জন্য use হচ্ছে। এখন যদি নতুন একটা system design করতে চাই Hairstyle recognize এর জন্য তাহলে আমরা একেবারে ভোড়া থেকে শুরু না করে face recognition model এর first বা second layer থেকে শুরু করতে পারি, একেই বলা হয় transfer learning.

Number of neurons per hidden layer:

- input আর output এর neuron সংখ্যা দিয়ে problem অনুযায়ী।
- আগের pyramid structure (যেমন 300, তারপর 200, তারপর 100 Neuron) prefer করা।
- কিন্তু এখন all hidden layer এ same number of neuron prefer করা।
- neuron যতগুলো থেকে depth বাড়ালে সাধারণত লাভ হয় বেশি।

Learning rate:

In general the ~~maxim~~ optimum learning rate is half of maximum learning rate (The learning rate at which the training algo diverges). ~~Simple~~ approach can be:-

- start with a large value.
- then divide this value with 3 & try again
- repeat this until the training algo stops diverging

Batch size:

should be less than 32 & ~~dom~~ shouldn't be too small. training small batch size ensures that each iteration is very fast. more than 20 then it helps take advantage of hardware & software optimization. (particular for matrix multiplication).

Epoch:

- Number of Layers → most of problems single layer is sufficient. But going deeper increases parameter efficiency. 1st of 2nd layer নিয়ে আরও model use করা যায় transfer learning.
- " " neurons per layer
- Type of activation function
- Learning rate
- Weight initialization logic etc

flexibility of ANN is also a drawback

Transfer learning: ধরি একটা model আছে যেটা already face recognition এর জন্য use হচ্ছে। এখন যদি নতুন একটা system design করতে চাই Hairstyle recognize এর জন্য তাহলে আমরা একেবারে ভোড়া থেকে শুরু না করে face recognition model এর first বা second layer থেকে শুরু করতে পারি, একেই বলা হয় transfer learning.

Number of neurons per hidden layer:

- input আর output এর neuron সংখ্যা হবে problem অনুযায়ী।
- আগের pyramid structure (যেমন 300, তারপর 200, তারপর 100 Neuron) prefer করা।
- কিন্তু এখন all hidden layer এ same number of neuron prefer করা।
- neuron বাড়ানোতে থেকে depth বাড়ানোর সাইড লাভ হয় বেশি।

learning rate:

In general the ~~maxim~~ optimum learning rate is half of maximum learning rate (The learning rate at which the training algo diverges). ~~Simple~~ approach can be:-

- start with a large value.
- then divide this value with 3 & try again
- repeat this until the training algo stops diverging

Batch size:

should be less than 32 & ~~don~~ should not be too small. training small batch size ensures that each iteration is very fast if more than 20 then it helps take advantage of hardware & software optimization. (particular for matrix multiplication).

Epoch:

(210) CNN introduction

↳ better than ANN in image & speech.

↳ ANN এ particular pixel নিয়ে কাজ করা হয়। But human Brain particular pixel নিয়ে কাজ করে না। group of pixel নিয়ে কাজ করে। CNN ও তাই করে।

বাকিটা mobile এ pic তোলা

(211) // stride

↳ first neuron যে group of pixel কে input হিসেবে নিচ্ছে, পরের Neuron হালকা shift করে " " " " " নিচ্ছে। একে বলা হয় stride. যে কয় ঘরে shift করা হবে তা বলা হবে ~~stride~~ stride of ৩. 2 pixel shift করলে stride of 2. stride যত বেশি হবে overlap তত কম এবং no. of neurons needed are also less.

⇒ mobile pic

(212) // padding

stride 3 নিলে

Large stride এর সমস্যা হলো অনেক image/pixel তাও cover হবে না। 16x16 image কে 5x5 এর window নিলে লাস্টের দুইটা pixel column বাকি থেকে যাবে, এর solution হলো first থেকে 1 টা col. আর last থেকে 1 টা col বাদ দেওয়া। 14x14 হয়ে গেলে। 14x14 কে 5x5 এর window আর stride of 3 নিলে সব cover হয়। একে বলা হয় valid padding.

আরেকটা solution হলো আমরা artificially 1 টা column of pixel add করে দিলাম। একে বলা হয় same padding.

obile pic for visualizing.

(213) // filter & feature Maps

↳ window size এর সমান dimension এর একটা matrix (for this case 5x5) ~~widow~~ window এর pixel যেখানে কিছু Data বা value থাকে তার সাথে গুণ করে যোগ করলে 1 টা value পাওয়া যায়। নিচের layer থেকে গুণ করে যোগ করলে 1 টা value layer এ হয়।

Data এক্সট্রাক্ট উপরের layer এ হয়।

example of this can be found in ~~mobile~~ pictures/screenshots.

গুণ করার পর যে output টা পাই তাকে বলা হয় feature map.

vertical filter

0	1	0
0	1	0
0	1	0

থাকলে vertical line clear দেখায়

Horizontal filter

0	0	0
1	1	1
0	0	0

থাকলে Horizontal line clear দেখায়।

Feature Map কিভাবে কাজ করে বিস্তারিত screenshot এ।
Input layer also have multiple layer of information which are called channels.

21) //Channels
↳ the input to our network when we are doing image recognition is image.
↳ grayscale image/Black & white → each pixel contains only one info. That is how wide the pixel is. 0-255 এ 0 means completely black. 255 means white. In between " are shade of grey.
↳ colored/RGB
↓
each pixel have 3 values rather than 1. They are R,G,B. All value 255 হলে white. R(255,255,0) → yellow, (0,255,255) → cyan
এই তিনটা আলাদা layer এ info রাখে। Input layer এই তিনটা (RGB) value নিয়ে তৈরি ২ টা channel. grayscale image এ অনুরূপ ২ টা channel. color pie দেখলে তার example তিনটা color নিয়ে কোন perceive করে।
screenshot এ।
//think-maths.co.uk/spreadsheet } → যেকোনো image এর RGB দেখাই দেখ।

215) //pooling Layer
↳ use করা হয় computational load, memory usage & number of parameter কমানোর জন্য।
↳ এইটা এমন একটা layer যেটার কোনো weight থাকে না। এ আগে নিচের layer এর window থেকে select করে আরো Based on:- {best}
↳ max pooling: max pixel of that window.
↳ Avg " : Avg " of " "
the example (screenshoot) used here have a stride of 2. এই মানে আমাদের feature map 4×4 হলে pooling layer 2×2 হবে তো।
It is a tradeoff. We give away some input info to reduce the computational load on our system.

model (preprocessing)

```
import numpy as np
import pandas as pd
% matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
// fashion MNIST data use করা।
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal",
               "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

training data → training এর জন্য
test " → test এর জন্য
Validation data → used for tuning the hyperparameters & evaluate the models.

```
//Data Reshape
# ANN এ 1D input নিতাম। কিন্তু CNN এ 3D input নিতে (Dataset, channel)
X_train_full = X_train_full.reshape((60,000, 28, 28, 1))
X_test = X_test.reshape((10000, 28, 28, 1))
//Data Normalization
X_train_n = X_train_full/255
X_test_n = X_test/255
// Spliting the data
X_valid, X_train = X_train_n[:5000], X_train_n[5000:]
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test_n
//Creating model architecture
np.random.seed(42)
tf.random.set_seed(42)
```

217 // structure & Compile

ANN এর ~~আগে~~ flatten করে input হত এখন convolution layer এর (26x26x32)

↓ padding করলে ; 32 ↓ 32 filter

Pooling Layer add করা যায়।

(13x13x32) ↓ → এইটা হল আমরা 2x2 max pooling use করব।

filter ↑ ; color হলে 3 হত

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=1,
    padding='valid', activation='relu', input_shape=(28,28,1)))
model.add(keras.layers.MaxPooling2D((2,2)))
```

→ আমরা pool করছি 2x2 matrix দিয়ে।

```
//3D কে flatten করে 1D করা হল।
model.add(keras.layers.Flatten())
model.add( " . " . Dense(300, activation= "relu"))
model.add( " . " . " (100, " = "relu"))
model.add( " . " . " (10, activation= "softmax"))
model.summary()
```

multilevel classification problem এর জন্য নেওয়া

```
//compiling
model.compile(loss="sparse-categorical-crossentropy",
    optimizer="sgd", metrics=["accuracy"])
```

218 // Training & results

```
model_history = model.fit(X-train, y-train, epochs=30, batch-size=64,
    validation_data=(X-valid, y-valid))

import pandas as pd
pd.DataFrame(model-history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
ev= model.evaluate(X-test-n, y-test)
ev
```

X_new = X_test [:3]

y_pred = model.predict_classes(X_new)

y_pred

y_test [:3]

print(plt.imshow(X_test[0].reshape((28,28))))

keras.backend.clear_session

del model.

(219) // Comparison - Pooling vs without Pooling layer দিয়ে comparison দিছে না।
সব same কোনো coding শুধু একটায় pooling layer দিয়ে ... But accuracy 3 হয় 220.
pooling use করলে টাইম কম লাগছে দিচ্ছে

(226) // Project (creating CNN from Scratch)

→ Classify colored images of cats & dogs (Data from kaggle)
- → It is a Binary classification problem
- → colored images (So 3 channel)
- → No standard dimension
- → Here they are using subset of 50000 kaggle data

in a particular manner (screenshot).

Data inside the folder is order in a particular manner

Process:
- → CNN with small dataset & 4 convo network & pooling
- → " " data Augmentation → is the process of creating artificial datasets for small datasets. (transforming this images into, like zooming in/out, cropping, rotating etc)
- → Transfer learning

228 // Project - preprocessing

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from  "  import keras
```

//saving the directories first এ ডিরেক্টরি টা ফাইল থেকে link copy করলে forward slash ... Backward এ দিতে হবে

train_dir = r'C:\Users\pukhn\x\y\z\cats_&dogs small\train'

validation_dir = r'C:\.......\validation'

test_dir = r'C:\.......\test'

//Data preprocessing steps → https://keras.io/preprocessing/image/

- → Read the picture files
- → Decode the Jpeg content to RBG grids of pixels
- → convert these into floating point tensors.
- → Rescale the pixel values (between 0 & 255) to the [0,1] interval.

→ এইসবগুলো step করার জন্যে একটা module আছে। ~~Kera~~ ImageDataGenerator

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

//Generating batches of tensor image data

```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150,150),      → standardizing using a target size.
    batch_size=20,
    class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
    validation_dir, target_size=(150,150), batch_size=20, class_mode='binary')
```

(note on train_generator:) output of this will be in the form of x & y. y labels will depend on the name of the folder. In this case cats & dogs.

(229) // Project training

```
from tensorflow.keras import layers
  "     "       "      "   models
model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
{same, just use 128} এইবার করো। layer 2টো convolution এ 128 দিয়ে, same করো
same                                              total 4 টা layer.
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

(notes:) 2 বার keras.layer use করে না, আলাদা করে layer import করলে — 32 filters, 3x3 window

3d input 150x150x3 → 3 channel (RGB)

```
from tensorflow.keras import optimizers
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

→ RMSprop: SGD এর চেয়ে Better incase of image processing.

→ এখানে just .fit use করছি। But আগে Data থেকে generator এ যেত।

```
history = model.fit_generator(
    train_generator, steps_per_epoch=100, epochs=20,
    validation_data=validation_generator, validation_steps=50)
```

steps_per_epoch: কিভাবে বের করা হয় — total image no. / batch size

validation_steps: validation image no. / batch size

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```

} এখানে দেখা যাচ্ছে overfit হয়ে গেছে। কারণ epoch বাড়ালে training accu বাড়ছে কিন্তু validation accu বাড়ছে না। → এটাকে fix করতে Augmentation.

```
model.save("model.h5")
from tensorflow.keras import backend as K
K.clear_session()
del model
```

(237) // Project - Data augmentation preprocessing

- - - - - - - আগের মত

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(150,150),
    batch_size=32, class_mode='binary')
```

validation_generator = test_data_gen.flow_from_directory(
validation_dir, target_size=(150,150),
batch_size=32, class_mode='binary')

(238) // Data Augmentation - training & results

model = models.Sequential()

4 conv layer

4 pooling layer

flatten layer

model.add(layers.Dropout(0.5)) → each epoch ↗ 50% neuron de activate ↘ very effective for avoiding overfitting

1 Dense layer relu

output layer sigmoid

model.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-4),
metrics=['acc'])

// training

history = model.fit_generator(train_generator, steps_per_epoch=100,
epochs=100, validation_data=validation_generator,
validation_steps=50)

model.save("project_cnn.h5")

pd.DataFrame(history.history).plot(figsize=(8,5))

plt.grid(True)

plt.gca().set_ylim(0,1)

plt.show()

অনেক অনেকের তৈরি বিভিন্ন CNN model structure আমরা আমাদের নিজেদের Data তে use করতে পারি। একেই বলা হয় transfer learning.
এই Architecture গুলা একটা competition থেকে বাইরে এসেছে নাম হল ImageNet Large Scale Visual Recognition Challenge (ILSVRC).
এরকম কিছু Architecture হলো LeNet (1998), AlexNet (2012), ZF Net (2013), Google Net (2014), VGG Net (2014), ResNet (2015).
More data in the screeshot.

(240) // LeNet

↳ simple archi with 3 convo layer & 2 Avrg Pooling layer.

More about this in the screenshot.

(241) // VGG

→ 138 million param
→ convo layer use না করে convo block use করছে 2টা, 3টা করে
↳ এইখানে relu use করা হয়েছে এবং softmax.

More about this in the screenshot.

(242) // Google Net

→ এইটাতে new concept use করা হয়েছিল। যাকে বলা হচ্ছে inception mo
↳ very complex network.

More info about this in the screenshot.

(243) // Transfer learning

→ Mostly we use the convolution base of other network.
↳ এর উপরে আমরা একটা classifier add করে আমাদের model তৈরি করি। But convolution base কেন ই use করা হচ্ছে কারণ যত ও যখন যে Data উপরে use করা হচ্ছে অবশ্যই তো যে feature গুলো উপরে দেখছে same feature গুলো দেখছে।

(244) // Project - Transfer learning

↳ VGG 16 এর weights ও bias download করে apply করবো।

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255, rotation_range=40, width_shift_range=0.2,
    height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(150,150), batch_size=20, class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
    validation_dir, target_size=(150,150), batch_size=20, class_mode='binary')

from tensorflow.keras.applications import VGG16
conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(150,150,3))
conv_base.summary()
from tensorflow.keras import models
      "      "    "    layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(  "  .  "  (1,    "   ='sigmoid'))
model.summary()

conv_base.trainable = False
// compiling

from tensorflow.keras import optimizers
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])
```

conv base যে কয়টি layer আছে তা।

এটা optional. কেননা VGG already optimized param এর weight optimize again করতে না চাইলে এটা use করা, time save করতে চাইলে।

এটা না use করলে already optimized.

```
history = model.fit_generator(
        train_generator,
          steps_per_epoch = 100,
          epochs = 30, validation_data = validation_generator,
          validation_steps = 50,
          callbacks = [checkpoint_cb])

pd.DataFrame(history.history).plot(figsize=(8,5))
 plt.grid(True)
 plt.gca().set_ylim(0,1)
 plt.show()
hist_df = pd.DataFrame(history.history)         }  saving the
hist_csv_file = 'history.csv'                   }  history (if wanted)
with open(hist_csv_file, mode='w') as f:        }
          hist_df.to_csv(f)

test_generator = test_datagen.flow_from_directory(
              test_dir,
              target_size = (150,150),
              batch_size = 20,          → total image / batch size
              class_mode = 'binary')                   ↑
model.evaluate_generator(test_generator, steps = 50)
```

↳ যেহেতু Directory থেকে নিচ্ছি তাই evaluate_ generator ব্যবহার শুধু evaluate use করলে হবে না।