Page No.:
Date:

Classification.

Categorizing some unknown items into a discrete set of categories or classes.

- Supervised Learning approach.

Algorithms.

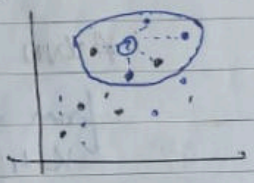
- Decision Trees. , • Naive Bayes ; Linear Discriminant Analysis
- K-Nearest Neighbor , • Logistic Regression,
- Neural Network , • Support Vector Machines (SVM)

K-Nearest Neighbor (KNN)

It takes bunch of labeled points and uses them to learn how to label other points.

1. Pick a value for k
2. Cal. the distance of unknown case from all cases.
3. Select the k-observations in the training data that are "nearest" to the unknown data point.
4. Predict the response of the unknown data point using the most popular response value from the k-nearest neighbors

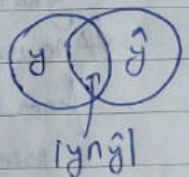
$$Dis(x_1, x_2) = \sqrt{\sum_{i=0}^{n} (x_{1i} - x_{2i})^2}$$



Evaluation Metrics. [Accuracy]

Jaccard index

$$J(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} = \frac{|y \cap \hat{y}|}{|y| + |\hat{y}| - |y \cap \hat{y}|}$$



$|y \cap \hat{y}|$

High Accuracy at 1.0

F1-Score

precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1-Score = 2 × (prc × rec) / (prc + rec)

Confusion Matrix

y \ ŷ	Churn=1	Churn=0
Churn=1	6 TP	9 FN
Churn=0	1 FP	24 TN

High Accuracy at 1.0

ML with Python

Page No.
Date

ML is the subfield of computer science that gives "computers the ability to learn without being explicitly programmed".

- Regression / Estimation
  - Predicting continuous value
- Classification
  - Predicting the item class/category of a case
- Clustering
  - Finding the structure of data; summarization
- Associations
  - Associating frequent co-occuring items/events
- Anomaly detection
  - Discovering abnormal and unusual cases
- Sequence mining
  - Predicting next event
- Dimension Reduction
  - Reducing the size of data (PCA)
- Recommendation systems
  - Recommending items

Python libraries for ML

NumPy, SciPy, matplotlib, pandas, scikit learn

Supervised
- ↳ Regression
- ↳ Classification

Unsupervised
- ↳ Dimension reduction
- ↳ Density estimation
- ↳ Market basket analysis
- ↳ Clustering

Regression

X: Independent 1 or more data. can be categorical also.

Y: Dependent only 1 and must be continuous value

- Ordinal Regression "
- Poisson "
- Fast forest quantile "
- Linear, Polynomial, Lasso, stepwise, Ridge "
- Bayesian linear "
- Neural Network "
- Decision forest "
- Boosted decision tree "
- KNN (K-nearest neighbors)

Teacher's Signature .....

Page No.:
Date:

Logloss

$$\text{LogLoss} = -\frac{1}{n}\sum(y \times \log(\hat{y}) + (1-y) \times \log(1-\hat{y}))$$

High Accuracy at 0.0

y	ŷ
Churn	Predicted churn
1	0.91
1	0.13
0	0.04
0	0.23
0	0.43

```
X = df[['A', 'B', 'C' ... ]].values  #.astype(float)
y = df['Z'].values

#Normalize Data
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))

from sklearn.model_selection import train_test_split
X_train ... y_test = train_test_split(X, y, test_size=0.2, random_
                                      state=4)

#KNN
from sklearn.neighbors import KNeighborsClassifier
K=4
neigh = KNeighborsClassifier(n_neighbors=K).fit(X_train, y_train)
yhat = neigh.predict(X_test)
#Accuracy.
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score
                              (y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))

# For other Ks
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
for n in range(1, Ks):
    neigh = KNeighborsClassifier(n_neighbors=n).fit(X_train, y_tr
    yhat = neigh.predict(X_test)
```

mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)

std_acc[n-1] = np.std(yhat == y_test)/np.sqrt(yhat.shape[0])

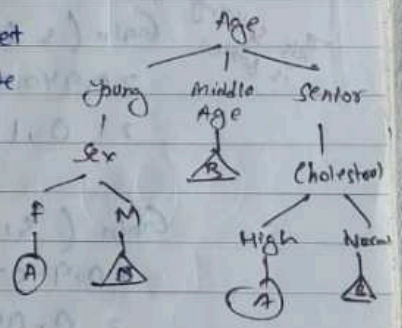
mean_acc.

#plotting

```
plt.plot(range(1,Ks), mean_acc, 'g')
plt.fill_between(range(1,Ks), mean_acc-1*std_acc, mean_acc+1*std_acc,
                 alpha=0.10)
plt.fill_between(range(1,Ks), mean_acc-3*std_acc, mean_acc+3*std_acc,
                 alpha=0.10, color='y')
plt.legend(('Accuracy', '+/- 1xstd', '+/- 3xstd'))
plt.ylabel('Accuracy')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
print("The best accuracy was with", mean_acc.max(), "with K=",
      mean_acc.argmax()+1)
```

Decision Tree.

1. Choose an attribute from your dataset
2. Calculate the significance of attribute in splitting of data.
3. Split data based on the value of the best attribute.
4. Go to step 1.



Entropy : measure of randomness or uncertainty.

1 Drug A	3 Drug A
7 Drug B.	5 Drug B
Entropy is low ✓	Entropy is high X
0 Drug A	4 Drug A
8 Drug B.	4 Drug B
Entropy is 0 ✓✓ Pure node	Entropy = 1 XX

Teacher's Signature

Page No.
Date

Evaluation

```
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat, pos_label=0)
# Confusion Matrix in previous page
print(confusion_matrix(y_test, yhat, labels=[1,0]))

cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],
                      normalize=False, title='Confusion matrix')

print(classification_report(y_test, yhat))
```

```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

General Cost function

$\sigma(\theta^T X) \rightarrow P(y=1|x)$

Cost function. $Cost(\hat{y}, y) = \frac{1}{2}(\sigma(\theta^T X) - y)^2$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(\hat{y}, y)$$

$$Cost(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y=1 \\ -\log(1-\hat{y}) & \text{if } y=0 \end{cases}$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^i \log(\hat{y}^i) + (1-y^i)\log(1-\hat{y}^i)$$

Best parameter of our model?

by minimize the cost function.

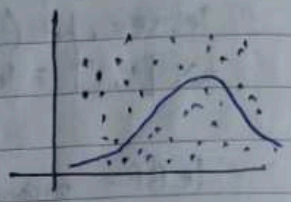
How to min the cost function.

Using Gradient Descent

Page No.:
Date:

SVM (Support Vector Machines)

It works by mapping data to high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable.



Mapping data into higher dimensional space is called Kernelling.

• Linear, • Polynomial, • RBF • Sigmoid

Dataset:

ID	Clump	UnifSize	UnifShap....	BareNuc	class
..	5			1	2
..	5			10	2
..	3			2	2
..	6			4	2
..	4			1	2

In class 2 = Benign 4 = Malignant

```
ax = cell_df[cell_df['Class']==4][0:50].plot(kind='scatter', x='Clump',
                    y='UnifSize', color='  ', label='Malignant');
cell_df[cell_df['Class']==2][0:50].plot(kind='scatter', x='Clump',
                    y='UnifSize', color='  ', label='Benign', ax=ax)
plt.show()

cell_df.dtypes   # Let BareNuc be object
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')

feature_df = cell_df[['Clump', ..... 'Mit']]
X = np.asarray(feature_df)

cell_df['Class'] = cell_df['Class'].astype('int')
y = np.array(cell_df['Class'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=4)
```

#1 Convert the dataset into only numeric dataset then normalization

```
from sklearn.preprocessing import StandardScaler
X = df.values[:, 1:]
X = np.nan_to_num(X)
clus_dataset = StandardScaler().fit_transform(X)

clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum,
                 n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

→ New column.

```
df["clus_km"] = labels
df.groupby('clus_km').mean()

area = np.pi * (X[:, 1]) ** 2
plt.scatter(X[:,0], X[:,3], s = area, c = labels.astype(np.float),
            alpha = 0.5)
```

#2D

```
plt.xlabel('Age')
plt.ylabel('Income')
plt.show()
```

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize = (8,6))
plt.clf()
```

#3D

```
ax = Axes3D(fig, rect = [0, 0, .95, 1], elev = 48, azim = 134)

plt.cla()
ax.set_xlabel('Education')
ax.set_ylabel('Age')
ax.set_zlabel('Income')
ax.scatter(X[:,1], X[:,0], X[:,3], c = labels.astype(np.float))
```

Teacher's Signature

```
import sklearn.cluster import KMeans
import sklearn.datasets as make_blobs
```

Page No.:
Date:

Clustering

K-Means Clustering

On randomly generated dataset

```
np.random.seed(0)
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2,-1], [2,-3], [1,1]],
                  cluster_std=0.9)
plt.scatter(X[:,0], X[:,1], marker='.')
# Setting up K-Means.
k-means = KMeans(init="k-means++", n_cluster=4, n_init=12)
k_means.fit(X)
k_means_labels = k-mean.labels_
k_means_cluster_centers = k_means.cluster_centers_
# Plotting.
fig = plt.figure(figsize=(6,4))

colors = plt.cm.Spectral(np.linspace(0,1, len(set(k_means_labels))))
ax = fig.add_subplot(1,1,1)

for k, col in zip(range(len([[4,4],[-2,-1],[2,-3],[1,1]])), colors):

        my_members = (k_means_labels == k)
        cluster_center = k_means_cluster_centers[k]
        ax.plot(X[my_members,0], X[my_members,1], 'w',
                markerfacecolor=col, markers='.')
        ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor
                markeredgecolor='k', markersize=6)

ax.set_title('KMeans')
ax.set_xticks(())
ax.set_yticks(())
plt.show()
```

#2D

#3D

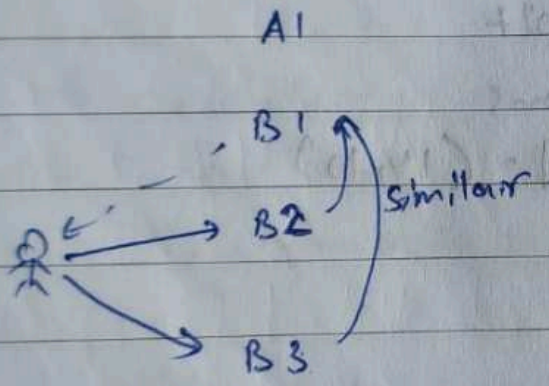
([1,0,0]), marker='o', markersize=5,
alpha=0.75)

plt.show()

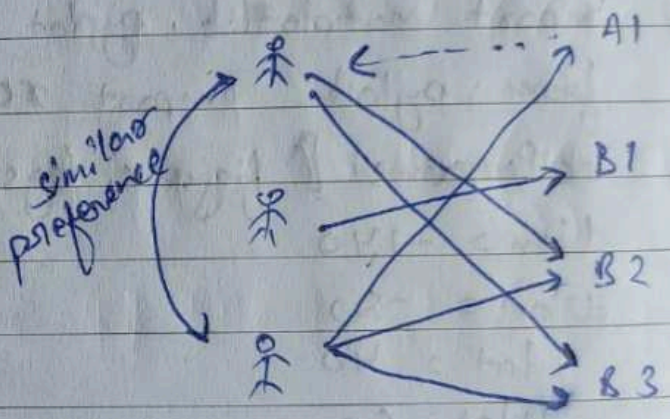
Recommender Systems.

It capture the pattern of peoples behavior and use to predict what else they might count or like.

Content Based

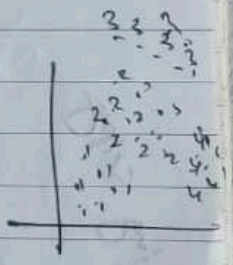


Collaborative Filtering

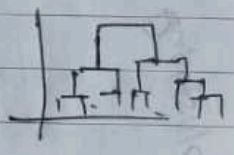


Teacher's Signature .....

```
from scipy import ndimage
from scipy.cluster import hierarchy
from scipy import spatial          # distance_matrix
from sklearn.datasets import make_blobs
# Generated Data
X1, y1 = make_blobs(n_samples = 50, centers = [[4,4], [-2, -1], [1, 1],
                                               [10, 4]], cluster_std = 0.9)
plt.scatter(X1[:, 0], X1[:, 1], marker = 'o')

from sklearn import manifold, datasets
from sklearn.cluster import AgglomerativeClustering

agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
agglom.fit(X1, y1)

plt.figure(figsize = (6,4))
x_min, x_max = np.min(X1, axis = 0), np.max(X1, axis = 0)
X1 = (X1 - x_min) / (x_max - x_min)   # avg. dist from X1

for i in range(X1.shape[0]):
    plt.text(X1[i, 0], X1[i, 1], str(y1[i]),
             color = plt.cm.nipy_spectral(agglom.labels_[i]/10.),
             fontdict = {'weight': 'bold', 'size': 9})

plt.xticks([])
plt.yticks([])
plt.scatter(X1[:, 0], X1[:, 1], marker = '.')
plt.show()
```



```
dist_matrix = distance_matrix(X1, X1)
Z = hierarchy.linkage(dist_matrix, 'complete')
dendro = hierarchy.dendrogram(Z)
```



	Action	Superh	comedy	Sci fi
A_4	1	0	1	1
A_5	1	0	0	0
A_6	1	1	0	1

x user profile.

A_4	0.2	0	0.3	0.3
A_5	0.2	0	0	0
A_6	0.2	0.2	0	0.3

→

0.8
0.2
0.6

A_4 will be above all recomandation.
Suppose Drama come into the list, then model will never ~~~~ recomand Drama.

```
import pandas as pd
from math import sqrt
import matplotlib.pyplot as plt
import numpy as np.

movies_df = pd.read_csv('    .csv')
ratings_df = pd.read_csv('    .csv')
movies_df.head()
```

out

	movieId	title	genres.
	1	A (1995)	Adv | Ani | comedy | fantasy
	2	B (1995)	— — —
	3	C (1995)	— — —
	4	D (1995)	— — —
	5	E (1995)	— — —

```
movies_df['year'] = movies_df.title.str.extract('(\d\d\d\d)', expand=False)
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)', expand=False)
movies_df['year'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

Teacher's Signature

```
userInput = [
    {'title' : 'Breakfast Club', 'rating' : 5},
    {'title' : 'B2', 'rating' : 2.5},
    :
    :
]
inputMovies = pd.DataFrame(userInput)
#Filtering out the movies by title
input_Id = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]

inputMovies = pd.merge(input_Id, inputMovies)
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)


userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin
                (inputMovies['movieId'].tolist())]
userMovies.head(1)
```

out

	movieId	
0	1	A

```
# We only need genre table.
userMovies = userMovies.reset_index(drop = True)
userGenreTable = userMovies.drop('movieId',1).drop('title', 1).
                drop('genres', 1).drop('year', 1)

inputMovies['rating']
```

out

```
# product.
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
userProfile.
```

out

Adv	10.0
Adj	.
(	(
:	(
	(

Teacher's Signature

Page No.:
Date:

call movie

```
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df
                                    ['movieId'])
genreTable = genreTable.drop('movieId', 1).dr -- , - ,

recommendationTable_df = ((genreTable * userProfile).sum(axis=1)
                          /(userProfile.sum()))
recommendationTable_df = r-- . .df. sort_values(ascending=False)
    . . - - . .head()
```

out

```
movieId
5018        0.74 ~
            0.72 ~ .
   .
   .
   .
```

```
# The recommendation table.
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df
                      .head(20).keys())]
```

out.	moviesId	title	genres	year
		A		1999
		B		"
		C		"
		D		"
		E		"

```
movies_df ['genres'] = movies_df.genres.str.split('|')
movies_df.head(1)
```

out	movieID	title	genres	year
0	1	A	[Adv, Ani, comedy, fantasy]	1995

```
movieWithGenres_df = movies_df.copy()
from index, row in movies_df.iterrows():
        for genre in row['genres']:
                moviesWithGenres_df.at[index, genre] = 1
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head(1)
```

out	movieId	title	genres	year	Adv	Adl	- - - - - - - -
0	1	A	[, ,]	1995	1.0	1.0	0.0 0.0 0.0 - - -

```
ratings_df.head(1)
```

out	userId	movieId	rating	timestamp
0	1	1	2.5	1204927694

```
ratings_df = ratings_df.drop('timestamp', 1)
```

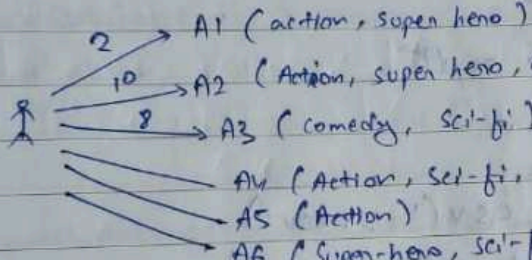
Page No.
Date

Implementing recommender systems.

- Memory-based
  - Uses the entire user item dataset to generate a recommendation
  - Uses statistical tech.
- Model based
  - Develops a model of users in an attempt to learn their preferences.
  - Created using ML tech. like regression, clustering, classification

Disadvantage of content based.

Model only recommends that users already have seen, related to that. But when a new things comes, the model don't recommend it.



Input users matrix

	users
A1	2
A2	10
A3	8

×

movies matrix

	Action	Superhero	comedy	Scifi
A1	1	1	0	0
A2	1	1	1	1
A3	0	0	1	1

↓↓

weighted Genre matrix

	Action	Superhero	comedy	Scifi
A1	2	2	0	0
A2	10	10	10	10
A3	0	0	8	8

↓↓

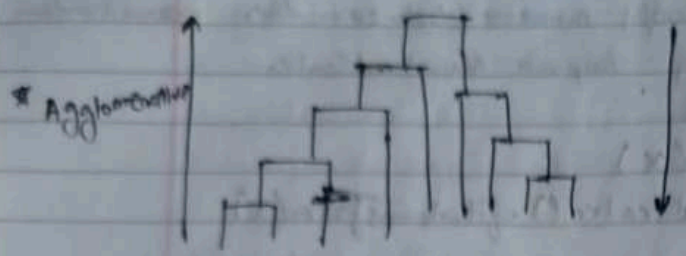
user profit	12	12	18	18

Normalize

0.2	0.2	0.3	0.3

Page No.:
Date:

Hierarchical clustering

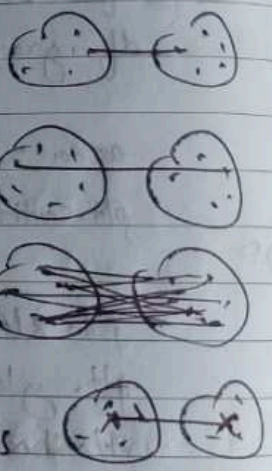


algo:

1. Create n clusters, one for each data point
2. Compute the proximity matrix
3. Repeat
   i. Merge the 2 closest clusters
   ii. Update the proximity matrix
4. Until only a single cluster remains.

Distance blw clusters

- Single - Linkage Clustering
  - Min distance blw clus.
- Complete - Linkage
  - Max dis.
- Average Li - -
  - Avg. -- -
- Centroid Li
  - Distance blw cluster centroids



Ad.

- Doesn't required no. of clusters to be specified
- Easy to implement
- Produces a dendrogram, which helps with understanding the clusters

Disad.

- Can never undo any previous steps throughout the algo
- Generally has long runtimes.
- Sometimes difficult to identify the no. of clusters by the dendrogram.

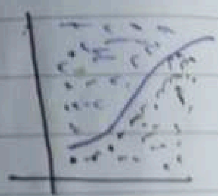
Date .

```
cnf_matrix = confusion_matrix(y_test, yhat, label=[2,4])
np.set_printoption(precision=2)
print(classification_report(y_test, yhat))
plt.figure()
plt.confusion_matrix(cnf_matrix, classes=['Benign(2)','Malignant(4)'],
                     normalize=False, title='Confusion matrix')
```

Error

```
from sklearn.metrics import f1_score
f1_score(y_test, yhat, average="weighted")

from sklearn.matrics import jaccard_score
jaccard_score(y_test, yhat, pos_label=2)
```

Logistic Regression.



```
                                       float
chunn_df = chunn_df[['  _ ' ' _ ' . _ 'chunn']]
Chunn_df['chunn'] = chunn_df['chunn'].astype('int')

X = np.asarray(chunn_df[['tenune      _    .    '']]   # But not...
y = np.asarray(chunn_df['chunn'])
```

#Normalization of Dataset

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
x_train - - - y_test = train_test_split(X, y, test_size=0.2, random_state...

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(x_train,
                                                          y_train)
                proba
yhat = LR.predict_proba(x_test)
```

main code

```
from sklearn import svm
clf = svm.SVC (kernal='rbf')
clf.fit (x_train, y_train)

yhat = clf.predict (x_test)
#Evaluation.
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix (cm, classes, normalize=False, title='Confusion
                           Matrix', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float')/cm.sum(axis=1)[:, np.newaxis]
        print ("Normalized Confusion Matrix")
    else:
        print ('Confusion Matrix without normalization")
    print (cm)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title (title)
    plt.colorbar()
    tick_marks = np.arange (len(classes))
    plt.xticks (tick_marks, classes, rotation=45)
    plt.yticks (tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() /2.
    for i, j in itertools.product (range(cm.shape[0]), range(cm.
                                                    shape[1])):
        plt.text (j, i, format (cm[i, j], fmt),
                  horizontalalignment='center'
                  color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel ("True label")
    plt.xlabel ("Predicted label")
```

Teacher's Signature

Page No.
Date

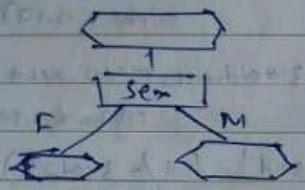
Entropy = $-p(A)\log_2(p(A)) - p(B)\log_2(p(B))$

↑ Total A / Total (A+B)

S = [9 (Drug B), 5 (Drug A)] ← Before splitting.

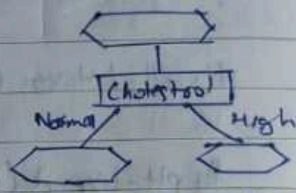
E = $-(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$

= 0.940



S: [3B, 4A] S: [6B, 1A]

E = 0.985 E = 0.592



S: [6B, 2A] S: [3B, 3A]

E = 0.811 E = 1.00

which one is better choise.

The tree with higher Information Gain after splitting

Information Gain = (Entropy before split) − (weighted entropy after split)

Entropy decreases Information gain Increases.

This splitting is better

Gain (S, Sex)

= 0.940 − [(7/14) 0.985 + (7/14) 0.592]

= 0.151

Gain (S, Cholestrol)

= 0.940 − [(8/14) 0.811 + (6/14) 1.0]

= 0.048

Code

X = my-data [['Age', 'Sex', 'BP', 'Cholestrol', 'Na_to_K']].va

Page No.
Date :

Convert categorial variables into dummy (one hot encoding)

```
from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])

le_BP = preprocessing.LabelEncoder()
le_BP.fit(['low','Normal','High'])
X[:,2] = le_BP.transform(X[:,2])

le_Chol = preprocessing.LabelEncoder()
le_Chol.fit(['Normal','High'])
X[:,3] = le_Chol.transform(X[:,3])
print(X[0:5])
```

Out

```
from sklearn.model_selection import train_test_split
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3,
                                                random_state=3)

from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree

drugTree = DecisionTreeClassifier(criterion="entropy", max_depth=4)
drugTree.fit(X_trainset, y_trainset)

predTree = drugTree.predict(X_testset)
```

Accuracy.

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTree's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
# is accuracy near 1.0 ✓
tree.plot_tree(drugTree)
plt.show()
```

Page No.
Date

Multiple LR

$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$

How to estimate θ?

- Ordinary Least Square
  Tries to estimate the value of θ by minimizing MSE.
- An Optimization algo
  - Gradient Descent for large dataset.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asanyarray(train[['ES', 'C', 'FC']])
y = np.asanyarray(train[['CO2']])
regr.fit(x,y)
print(regr.coef_)
```

Evaluation

```
y_hat = regr.predict(test[['ES', 'C', 'FC']])
x = np.asanyarray(test[[ ... ]])
y = np.asanyarray(test[['CO2']])
print("Residual sum of squares: %.2f",
      %.np.mean(y_hat - y) ** 2))
print('Variance score: %.2f' %.regr.score(x,y))
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
train_x = np.asanyarray(train[['ES']])
train_y = np.asanyarray(train[['CO2']])
test_x = np.asanyarray(test[['ES']])
test_y = npasanyarray(test[['CO2']])
poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)

clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
```

```
plt.scatter (train.ES, train.CO2, colour = 'b')
xx = np.arange (0.0, 10.0, 0.1)
yy = clf.intercept_[0] + clf.coef_[0][1] * xx + clf.coef_[0][2]
                                    * np.power (xx, 2)

plt.plot (xx, yy, '-r')
plt.xlabel ("Engine size")
plt.ylabel ("Emission")
```

Evaluation.

```
from sklearn.metrics import r2_score
test_x_poly = poly.transform (test_x)
test_y_ = clf.predict (test_x_poly)
print ("MAE : %.2f" % np.mean (np.absolute (test_y_ - test_y)))
print ("MSE : %.2f" % np.mean ((test_y_ - test_y) ** 2))
print ("R2-score : %.2f" % r2_score (test_y, test_y_))
```

[Logistic graph. $Y = a + \frac{b}{1 + c^{(x-d)}}$

```
x = np.arange (-5.0, 5.0, 0.1)
Y = 1- 4/(1+np.power (3, x-2))
plt.plot (x, Y)
```

Page No.
Date :

* RSE = Relative Square Error = $\frac{\sum_{j=1}^{n} (y_j - \hat{y}_j)^2}{\sum_{j=1}^{n} (y_j - \bar{y})^2}$

$R^2 = 1 - RSE$

Lab for SLR

```
# Suppose we need specific column from a df
cdf = df[['ES', 'C', 'FC', 'CO2']]
# Hist
viz = cdf[['ES', 'C', 'FC', 'CO2']]
viz.hist()
# Scatter plot.
plt.scatter(cdf.FC, cdf.CO2, color='b')
# Spliting data
msk = np.random.rand(len(cdf)) < 0.8
train = cdf[msk]
test = cdf[~msk]
# Training data distribution in scatter plot
plt.scatter(train.ES, train.CO2)
# Model
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanyarray(train[['ES']]) # convert a list into array.
train_y = np.asanyarray(train[['CO2']])
regr.fit(train_x, train_y)
# plotting LR.
plt.scatter(train.ES, train.CO2)
plt.plot(train_x, regr.coef_[0][0]*train_x + regr.intercept_[0], '-r')
# Error
from sklearn.metrics import r2_score
test_x = np.asanyarray(test[['ES']])
test_y = np.asanyarray(test[['CO2']])
test_y_ = regr.predict(test_x)
print("MAE: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("MSE: %.2f" % np.mean((test_y_ - test_y)**2))
print("R2-score: %.2f" % r2_score(test_y, test_y_))
```

Teacher's Signature

Page No.
Date:

Simple LR

Error : $\frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$

$\hat{y} = \theta_0 + \theta_1 x_1$

MSE : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ it should be minimum.

$\theta_1 = \frac{\sum_{i=1}^{s}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s}(x_i - \bar{x})^2}$

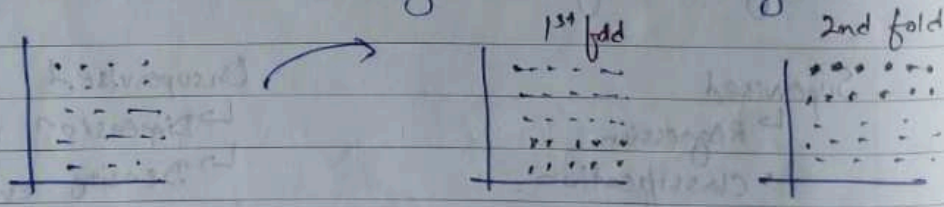
$\theta_0 = \bar{y} - \theta_1 \bar{x}$ [$\bar{x}$ = Σ of every x / Total
$\bar{y}$ = " " y / "]

It is fast, no parameter tuning, easy to understand

Model evaluation approaches.

- Train and Test on the same dataset → may result into overfit
  High Training accuracy
  Low out of sample accuracy
- Train/Test split → Mutually exclusive, more accurate
  But Highly dependent on which dataset is the data is trained and tested
- K-fold cross-Validation [It solves all issues]

  It split same data set into various folds means splitting a table into training and testing set and where each testing set varrious from one another and train with the rest of the data set and calculate the accuracy, and at last we avarage all the accuracy.



MAE : Mean Average Error $\frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$

MSE : Mean Square Error $\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2$

RMSE : Root Mean Square Error $\sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$

Relative Absolute Error RAE : $\frac{\sum_{j=1}^{n}|y_j - \hat{y}_j|}{\sum_{j=1}^{n}|y_j - \bar{y}|}$

Teacher's Signature