WEEK-1(reg algo)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/cars24-price.csv')
Gap
df.head()//type for ex
Gap
from sklearn.model_selection import train_test_split
X = df.drop('selling_price',axis=1)
Y = df['selling_price']

X_train , X_test , Y_train ,Y_test = train_test_split( X ,Y , train_size =0.3 , random_state=1)

from sklearn.linear_model
import LinearRegression
model = LinearRegression()
model.fit(X_train , Y_train)
y_pred =model.predict(X_test)
y_pred[:10]//type for ex
Gap
Y_test[:10]//type for ex
GAP
model.score(X_test,Y_test)//type for ex
Gap
model.intercept_ //type for ex
Gap
```

WEEK-2(LR,DT,RF)

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("/content/train.csv")

df = df.drop(['Name','Ticket','Cabin'], axis=1)

df.head(5) //type for exec
Gap
sns.heatmap(df.isnull()) //type for exec
Gap
df['Age'] = df['Age'].interpolate()

sns.heatmap(df.isnull()) //type for exec
Gap
df = df.dropna()

df.head() //type for exec
Gap
df.info() //type for exec
```

```
Gap
Embarked = pd.get_dummies(df['Embarked'])
Sex = pd.get_dummies(df['Sex'])
df = pd.concat((df,Embarked,Sex), axis=1)
df.head() //type for exec
Gap
df = df.drop(['Sex','Embarked'],axis=1)

x = df.values
y = df['Survived'].values
x = np.delete(x,1,axis=1)
Gap
### Linear Regression
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train,y_train)
lr.score(x_test,y_test)
Gap
### Decision Tree Classifier

from sklearn import tree
dt_clf = tree.DecisionTreeClassifier(max_depth = 5)
dt_clf.fit(x_train,y_train)

dt_clf.score(x_test,y_test)
Gap
y_pred = dt_clf.predict(x_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
Gap
### Random Forest Classifier

from sklearn import ensemble
rf_clf = ensemble.RandomForestClassifier(n_estimators=100)
rf_clf.fit(x_train,y_train)
rf_clf.score(x_test,y_test)
Gap



WEEK-3(SVM)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv(“/content/creditcard.csv")
Gap
df.head(10) //Type for exec
Gap
count_class=pd.value_counts(df["Class"], sort= True)
count_class.plot(kind= ‘bar')
Gap
frauds= len(df[df["Class"]==1])
normal= len(df[df["Class"]==0])
print("The number of fraud transactions( Class = 1): ",frauds)
print("The number of normal transactions( Class = 0): “,normal)
Gap
```

```
fraud_index= np.array(df[df["Class"]==1].index)
normal_index= df[df["Class"]==0].index
random_normal_indices= np.random.choice(normal_index,
500, replace= False)
random_normal_indices= np.array(random_normal_indices)

new_indices= np.concatenate([fraud_index, random_normal_indices])


new_data= df.iloc[new_indices, :]

print("No of Fraud Tranasactions:",len(new_data[new_data["Class"]== 1]))
print("No of Normal Tranasactions:",len(new_data[new_data["Class"]== 0]))
print(new_data.head(10))
Gap
new_data = new_data.drop(["Time","Amount"], axis= 1)
new_data.head(10)
Gap
X = new_data.iloc[:,new_data.columns != 'Class'].values
Y = new_data.iloc[:,new_data.columns == 'Class'].values

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
Gap
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size= 0.25, random_state= 0)

print("x_train: ", len(x_train), "y_train: ",len(y_train))
print("x_test: ", len(x_test), "y_test: “,len(y_test)) //type for exec
Gap
classifier= SVC(C= 1, kernel= 'rbf', random_state= 0)
classifier.fit(x_train, y_train.ravel()) //type for exec
Gap
y_pred = classifier.predict(x_test)
cm = confusion_matrix(y_test, y_pred)
print(cm) //type for exec
Gap
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))//type for exec




WEEK-4(Logistic Regression)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/isl/titanic/titanic.csv")
df.head() //type for exec
Gap
df.describe() //type for exec
Gap
df.describe(include = ['O']) //type for exec
Gap
df_corr = df.corr() //type for exec
Gap
```

```
sns.heatmap(df_corr, annot=True) //type for exec
Gap
df.isna().sum() //type for exec
Gap
df.Age.fillna(df['Age'].mean(), inplace=True)
Gap
df.Embarked.replace('NaN', np.nan, inplace=True)
df['Embarked'].fillna(list(df['Embarked'].mode())[0], inplace=True)
Gap
df.isna().sum()//type for exec
Gap
dummies = pd.get_dummies(df['Embarked'])
df = pd.concat([df, dummies], axis=1)
df.head()
Gap
dummies = pd.get_dummies(df['Sex'])
df = pd.concat([df, dummies], axis=1)
df.head()
Gap
df.drop(['Sex', 'Embarked'], axis=1, inplace=True)
df.head()
Gap
from sklearn.preprocessing
import StandardScaler scaler = StandardScaler()
scaler_series = scaler.fit_transform(df[['Age', 'Fare']])
scaler_df = pd.DataFrame(scaler_series, columns=['Age', 'Fare'])
df.drop(['Age', 'Fare'], axis=1, inplace=True)
df = pd.concat([df, scaler_df], axis=1)
df.head()
Gap
X = df.drop(['Survived'], axis=1)
y = df['Survived']
Gap
from sklearn.linear_model
import LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(X, y)
Gap
clf.score(X, y)
```

Week-6(Delta Learning)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("/content/drive/MyDrive/isl/Iris/iris.csv",
names=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'label'])
df.head() //type for exec
Gap
df.describe() //type for exec
Gap
count = df['label'].value_counts()
count
Gap
count.plot(kind = 'bar', x=count.index) //type for exec
Gap
```

```
X = df.drop(['label'], axis = 1)
X.head()
Gap
y = df['label']
y = y.map({'Iris-setosa' : 1,'Iris-versicolor' : 1, 'Iris-virginica' : 3})
Gap
w1, w2, w3, w4, b = tuple(np.random.normal(size=5))
print("Initial Weights + bias : ", (w1, w2, w3, w4, b))
Gap
epochs = 5
alpha = 0.01
Gap
MSE = []
for i in range(epochs):
error = []
for i in range(len(X)):

y_pred = w1*X.iloc[i, 0] + w2*X.iloc[i, 1] + w3*X.iloc[i, 2] + w4*X.iloc[i, 3] + b

Gap
t = y[i]
err = diff**2
error.append(err)
delta_w1 = alpha*diff*X.iloc[i, 0]
delta_w2 = alpha*diff*X.iloc[i, 1]
delta_w3 = alpha*diff*X.iloc[i, 2]
delta_w4 = alpha*diff*X.iloc[i, 3]
delta_b = alpha*diff
Gap
w1 = w1 + delta_w1
w2 = w2 + delta_w2
w3 = w3 + delta_w3
w4 = w4 + delta_w4
b = b + delta_b


Gap

MSE.append(np.array(error).mean())

for i in range(epochs):

print("Mean Square error at epoch -", str(i+1)+" :", MSE[i])

Gap

plt.plot(np.array(MSE))
plt.xticks(np.arrange(epochs),[str(i+1) for i in range(epochs)])
plt.xlabel(‘Epochs’)
plt.ylabel(“Mean Square Error”)
plt.title(“Epochs vs MSE”)
plt.show()


WEEK-5(K-Means)
```

Cell:

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


dataset=load_iris()

X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
```

Cell:

```
# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')
```

Cell:

```
# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Cell:

```
plt.figure(figsize=(14,11))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
# K-PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')


# GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

WEEK-7(increment learning algorithm)

```
# importing libraies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = [[2.7810836,2.550537003,0],
[1.465489372,2.362125076,0], [3.396561688,4.400293529,0], [1.38807019,1.850220317,0],
[3.06407232,3.005305973,0], [7.627531214,2.759262235,1], [5.332441248,2.088626775,1],
[6.922596716,1.77106367,1], [8.675418651,-0.242068655,1], [7.673756466,3.508563011,1]]
df = pd.DataFrame(dataset, columns=['X1', 'X2', 'label'])
df
Gap
def train_data(data):
X = data.drop(['label'], axis=1) y = data['label']
return X, y
Gap
def train(X, y):
```

```
w1, w2, b = tuple(np.random.normal(size=3))
print("Initial Weights + bias : ", (w1, w2, b))


alpha = 0.01
flag = True j=0
while True:
error = []
j += 1
for i in range(len(X)):
y_pred = w1*X.iloc[i, 0] + w2*X.iloc[i, 1] + b

t = y[i]
diff = t - y_pred

err = diff**2
error.append(err)
delta_w1 = diff*X.iloc[i, 0]
delta_w2 = diff*X.iloc[i, 1]
delta_b = diff
w1 = w1 + alpha*delta_w1 w2 = w2 + alpha*delta_w2 b = b + alpha*delta_b
MSE = np.array(error).mean()
print("Epoch -", str(j))
 print("Mean Square error :", MSE) print("Weights :-")
print("w1 :", w1)
print("w2 :", w2)
print("b :", b)
if MSE < 0.05:
break return w1, w2, b

Gap
X, y = train_data(df)

w1, w2, b = train(X, y)

Gap

def predict(x):
y_pred = w1*x[0] + w2*x[1] + b

return 1.0 if y_pred > 0.3 else 0.0

Gap

for (i, j) in zip(X.values, y):
y_p = predict(i)
print("I/P :", i[0], i[1])
print("Expected :", j, "Predicted :", y_p)

print("----------------------------------")
```

WEEK-8(logic gates bp)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt import seaborn as sns
or_arr = np.array([[-1, -1, -1], [-1, 1, 1],
[1, -1, 1],
[1, 1, 1]])
and_arr = np.array([[-1, -1, -1],
[-1, 1, -1], [1, -1, -1], [1, 1, 1]])
xor_arr = np.array([[-1, -1, -1], [-1, 1, 1],
[1, -1, 1],
[1, 1, -1]])
df_or = pd.DataFrame(or_arr, columns = ['X1', 'X2', 'label'])
df_or.head()

Gap

df_and = pd.DataFrame(and_arr, columns = ['X1', 'X2', 'label'])
df_and.head()
Gap
df_xor = pd.DataFrame(xor_arr, columns = ['X1', 'X2', 'label'])
df_xor.head()
Gap
def train_data(data):
X = data.drop(['label'], axis=1)
y = data['label']
return X, y
Gap
def train(X, y):
w1, w2, b = tuple(np.random.normal(size=3))
print("Initial Weights + bias : ", (w1, w2, b))
alpha = 0.1

MSE = []
for i in range(epochs):
        error = []
        for i in range(len(X)):
                y_pred = w1*X.iloc[i, 0] + w2*X.iloc[i, 1] + b
                t = y[i]
                err = diff**2
                error.append(err)
                delta_w1 = diff*X.iloc[i, 0]
                delta_w2 = diff*X.iloc[i, 1]
                delta_b = diff
                w1 = w1 + alpha*delta_w1
                 w2 = w2 + alpha*delta_w2
                b = b + alpha*delta_b
```

```
            MSE.append(np.array(error).mean())
        for i in range(epochs):
            print("Mean Square error at epoch -", str(i+1)+" :", MSE[i])
        plt.plot(np.array(MSE))
        plt.xticks(np.arange(epochs), [str(i+1)
        for i in range(epochs)]) plt.xlabel('Epochs')
        plt.ylabel("Mean Square Error")
        plt.title("Epoch vs MSE")
        plt.show()
X, y = train_data(df_or)
 train(X, y)
```

WEEK-9(BUILDING BLOCKS OF CNN)

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
mnist=tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()
print(y_train[5])
```

Gap

```
plt.imshow(x_train[5])
plt.show()
```

Gap

```
model=tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x_train,y_train,epochs=3)
```

Gap

```
predictions=model.predict(x_test)
print(predictions[3])
```

Gap

```
print(np.argmax(predictions[3]))
```

Gap

```
plt.imshow(x_test[3])
plt.show()
```

WEEK-10(RNN)

```
import numpy as np
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import os
import tensorflow as tf
from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_log_error

from datetime import datetime
from datetime import timedelta

from tensorflow.keras import layers
from tensorflow.keras import Input
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint,
ReduceLROnPlateau, EarlyStopping

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Gap

```
train_df = gpd.read_file("/kaggle/input/covid19-global-
forecasting-week-4/train.csv")
train_df["ConfirmedCases"] =
train_df["ConfirmedCases"].astype("float")
train_df["Fatalities"] = train_df["Fatalities"].astype("float")
#The country_region
train_df["Country_Region"] =
[ row.Country_Region.replace("'","").strip(" ") if
row.Province_State=="" else
str(row.Country_Region+"_"+row.Province_State).replace("'","").str
ip(" ") for idx,row in train_df.iterrows()]
```

Gap

```
extra_data_df.head()
```

Gap

```
train_df = train_df.merge(extra_data_df, how="left",
on=['Country_Region','Date']).drop_duplicates()
train_df.head()
```

Gap

```
trend_df["temporal_inputs"] =
[np.asarray([trends["infection_trend"],trends["fatality_trend"],trends["rest
riction_trend"],trends["quarantine_trend"],trends["school_trend"]]) for
idx,trends in trend_df.iterrows()]
```

```
trend_df = shuffle(trend_df)
```

Gap

```
trend_df.head()
```

Gap

```
#temporal input branch
temporal_input_layer = Input(shape=(sequence_length,5))
main_rnn_layer = layers.LSTM(64, return_sequences=True,
recurrent_dropout=0.2)(temporal_input_layer)

#demographic input branch
demographic_input_layer = Input(shape=(5))
demographic_dense = layers.Dense(16)(demographic_input_layer)
demographic_dropout = layers.Dropout(0.2)(demographic_dense)

#cases output branch
rnn_c = layers.LSTM(32)(main_rnn_layer)
merge_c = layers.Concatenate(axis=-1)([rnn_c,demographic_dropout])
dense_c = layers.Dense(128)(merge_c)
dropout_c = layers.Dropout(0.3)(dense_c)
cases = layers.Dense(1, activation=layers.LeakyReLU(alpha=0.1),name="cases")
(dropout_c)

#fatality output branch
rnn_f = layers.LSTM(32)(main_rnn_layer)
merge_f = layers.Concatenate(axis=-1)([rnn_f,demographic_dropout])
dense_f = layers.Dense(128)(merge_f)
dropout_f = layers.Dropout(0.3)(dense_f)
fatalities = layers.Dense(1, activation=layers.LeakyReLU(alpha=0.1),
name="fatalities")(dropout_f)


model = Model([temporal_input_layer,demographic_input_layer],
[cases,fatalities])

model.summary()
```

Gap

```
callbacks = [ReduceLROnPlateau(monitor='val_loss', patience=4, verbose=1,
factor=0.6),
             EarlyStopping(monitor='val_loss', patience=20),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss',
save_best_only=True)]
model.compile(loss=[tf.keras.losses.MeanSquaredLogarithmicError(),tf.keras.l
osses.MeanSquaredLogarithmicError()], optimizer="adam")
history = model.fit([X_temporal_train,X_demographic_train], [Y_cases_train,
Y_fatalities_train],
          epochs = 250,
          batch_size = 16,
          validation_data=([X_temporal_test,X_demographic_test],
[Y_cases_test, Y_fatalities_test]),
          callbacks=callbacks)
```

Gap

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss over epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='best')
plt.show()
```

Gap

```
model.load_weights("best_model.h5")
predictions = model.predict([X_temporal_test,X_demographic_test])
display_limit = 30
for inputs, pred_cases, exp_cases, pred_fatalities, exp_fatalities in
zip(X_temporal_test,predictions[0][:display_limit],
Y_cases_test[:display_limit], predictions[1][:display_limit],
Y_fatalities_test[:display_limit]):
    print("================================================")
    print(inputs)
    print("Expected cases:", exp_cases, " Prediction:", pred_cases[0],
"Expected fatalities:", exp_fatalities, " Prediction:", pred_fatalities[0] )
```

Gap

```
def get_RMSLE_for_all_regions(groundtruth_df):
    RMSLE_cases_list = []
    RMSLE_fatalities_list = []
    for region in groundtruth_df.Country_Region.unique():
        RMSLE_cases, RMSLE_fatalities = get_RMSLE_per_region(region,
groundtruth_df, False)
        RMSLE_cases_list.append(RMSLE_cases)
        RMSLE_fatalities_list.append(RMSLE_fatalities)
    print("RMSLE on cases:",np.mean(RMSLE_cases_list))
    print("RMSLE on fatalities:",np.mean(RMSLE_fatalities_list))
get_RMSLE_for_all_regions(groundtruth_df)
```

Gap

```
badly_affected_countries = ["France","Italy","United
Kingdom","Spain","Iran","Germany"]
for country in badly_affected_countries:
    get_RMSLE_per_region(country, groundtruth_df, display_only=True

Gap

def display_comparison(region,groundtruth_df):
    groundtruth = groundtruth_df.query("Country_Region=='"+region+"' and
Date>='2020-04-01' and Date<='2020-04-15'")
    prediction = copy_df.query("Country_Region=='"+region+"' and
Date>='2020-04-01' and Date<='2020-04-15'")

    plt.plot(groundtruth.ConfirmedCases.values)
    plt.plot(prediction.ConfirmedCases.values)
```

```
    plt.title("Comparison between the actual data and our predictions for
the number of cases")
    plt.ylabel('Number of cases')
    plt.xlabel('Date')

plt.xticks(range(len(prediction.Date.values)),prediction.Date.values,rotatio
n='vertical')
    plt.legend(['Groundtruth', 'Prediction'], loc='best')
    plt.show()

    plt.plot(groundtruth.Fatalities.values)
    plt.plot(prediction.Fatalities.values)
    plt.title("Comparison between the actual data and our predictions for
the number of fatalities")
    plt.ylabel('Number of fatalities')
    plt.xlabel('Date')

plt.xticks(range(len(prediction.Date.values)),prediction.Date.values,rotatio
n='vertical')
    plt.legend(['Groundtruth', 'Prediction'], loc='best')
    plt.show()
```

Gap

```
display_comparison("Canada_Newfoundland and Labrador", groundtruth_df)
```