################################

### general features

################################

# read csv file as dataframe

import pandas as pd

cols = ["fLength", "fWidth", "fSize", "fConc", "fConc1", "fAsym", "fM3Long", "fM3Trans", "fAlpha", "fDist", "class"]

df = pd.read\_csv("magic04.data", names=cols)

# show start of dataframe

df.head()

# drop some columns from dataframe

df = df.drop(["wind", "visibility", "functional"], axis=1)

# change target feature to int

df["class"] = (df["class"] == "g").astype(int)

# draw histogram to all features

for label in cols[:-1]:

  plt.hist(df[df["class"]==1][label], color='blue', label='gamma', alpha=0.7, density=True)

  plt.hist(df[df["class"]==0][label], color='red', label='hadron', alpha=0.7, density=True)

  plt.title(label)

  plt.ylabel("Probability")

  plt.xlabel(label)

  plt.legend()

  plt.show()

# show classification report

from sklearn.metrics import classification\_report

classification\_report(y\_test, y\_pred)

# split data with np split

train, valid, test = np.split(df.sample(frac=1), [int(0.6\*len(df)), int(0.8\*len(df))])

# normalize and oversample

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import RandomOverSampler

def scale\_dataset(dataframe, oversample=False):

  X = dataframe[dataframe.columns[:-1]].values

  y = dataframe[dataframe.columns[-1]].values

  scaler = StandardScaler()

  X = scaler.fit\_transform(X)

  if oversample:

    ros = RandomOverSampler()

    X, y = ros.fit\_resample(X, y)

  data = np.hstack((X, np.reshape(y, (-1, 1))))

  return data, X,

# draw graphs for relations between all features

for i in range(len(cols)-1):

  for j in range(i+1, len(cols)-1):

    x\_label = cols[i]

    y\_label = cols[j]

    sns.scatterplot(x=x\_label, y=y\_label, data=df, hue='class')

    plt.show()

# scatter used for datapoints and plot for lines

plt.scatter(transformed\_x[:,0], transformed\_x[:,1])

plt.show()

################################

### KNN --> KNeighborsClassifier

################################

from sklearn.neighbors import KNeighborsClassifier

knn\_model = KNeighborsClassifier(n\_neighbors=5)

knn\_model.fit(X\_train, y\_train)

y\_pred = knn\_model.predict(X\_test)

################################

### Naive Bayes --> GaussianNB

################################

from sklearn.naive\_bayes import GaussianNB

nb\_model = GaussianNB()

nb\_model = nb\_model.fit(X\_train, y\_train)

y\_pred = nb\_model.predict(X\_test)

################################

### Logistic Regression --> LogisticRegression

################################

from sklearn.linear\_model import LogisticRegression

lg\_model = LogisticRegression()

lg\_model = lg\_model.fit(X\_train, y\_train)

y\_pred = lg\_model.predict(X\_test)

################################

### SVM --> SVC

################################

from sklearn.svm import SVC

svm\_model = SVC()

svm\_model = svm\_model.fit(X\_train, y\_train)

y\_pred = svm\_model.predict(X\_test)

################################

### Neural Netowrk

################################

import tensorflow as tf

def plot\_history(history):

  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

  ax1.plot(history.history['loss'], label='loss')

  ax1.plot(history.history['val\_loss'], label='val\_loss')

  ax1.set\_xlabel('Epoch')

  ax1.set\_ylabel('Binary crossentropy')

  ax1.grid(True)

  ax2.plot(history.history['accuracy'], label='accuracy')

  ax2.plot(history.history['val\_accuracy'], label='val\_accuracy')

  ax2.set\_xlabel('Epoch')

  ax2.set\_ylabel('Accuracy')

  ax2.grid(True)

  plt.show()

def train\_model(X\_train, y\_train, num\_nodes, dropout\_prob, lr, batch\_size, epochs):

  nn\_model = tf.keras.Sequential([

      tf.keras.layers.Dense(num\_nodes, activation='relu', input\_shape=(10,)),

      tf.keras.layers.Dropout(dropout\_prob),

      tf.keras.layers.Dense(num\_nodes, activation='relu'),

      tf.keras.layers.Dropout(dropout\_prob),

      tf.keras.layers.Dense(1, activation='sigmoid')

  ])

  nn\_model.compile(optimizer=tf.keras.optimizers.Adam(lr), loss='binary\_crossentropy',

                  metrics=['accuracy'])

  history = nn\_model.fit(

    X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_split=0.2, verbose=0

  )

  return nn\_model, history

y\_pred = least\_loss\_model.predict(X\_test)

y\_pred = (y\_pred > 0.5).astype(int).reshape(-1,)

################################

### KMeans --> KMeans

################################

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters = 3).fit(X)

################################

### PCA --> PCA

################################

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

transformed\_x = pca.fit\_transform(X)

################################

### Linear Regression --> LinearRegression

################################

from sklearn.linear\_model import LinearRegression

temp\_reg = LinearRegression()

temp\_reg.fit(X\_train\_temp, y\_train\_temp)

temp\_reg.score(X\_test\_temp, y\_test\_temp)

# plot points and regression line

plt.scatter(X\_train\_temp, y\_train\_temp, label="Data", color="blue")

x = tf.linspace(-20, 40, 100)

plt.plot(x, temp\_reg.predict(np.array(x).reshape(-1, 1)), label="Fit", color="red", linewidth=3)

plt.legend()

plt.title("Bikes vs Temp")

plt.ylabel("Number of bikes")

plt.xlabel("Temp")

plt.show()

################################

###  Regression with Neural Net

################################

temp\_normalizer = tf.keras.layers.Normalization(input\_shape=(1,), axis=None)

temp\_normalizer.adapt(X\_train\_temp.reshape(-1))

temp\_nn\_model = tf.keras.Sequential([

    temp\_normalizer,

    tf.keras.layers.Dense(1)

])

temp\_nn\_model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.1), loss='mean\_squared\_error')

history = temp\_nn\_model.fit(

    X\_train\_temp.reshape(-1), y\_train\_temp,

    verbose=0, epochs=1000, validation\_data=(X\_val\_temp, y\_val\_temp))

plot\_loss(history)