```
# Use different loss functions to classify the nature of features in the Magic Gamma
# Telescope database

import pandas as pd
from sklearn.model_selection import train_test_split # Changed train_selection to train_test_split
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.utils import shuffle



df=pd.read_csv( 'C:/Users/HP/OneDrive/Desktop/ml 7th sem codes/datasets/magic.csv' )
df



classG=df[df['class']== 'g' ]
classH=df[df['class']== 'h' ]
countG, countH = df['class'].value_counts()
classGUnder = classG.sample(countH)
newDataset = pd.concat([classGUnder, classH], axis=0)
newDataset.to_csv('balanced_dataset.csv',index=False)
# df1=pd.read_csv( 'balanced_dataset.csv' )
# df1
newDataset = pd.read_csv('balanced_dataset.csv')
newDataset['class'].hist()



x = newDataset.drop('class', axis=1) # 1 for column, 0 for index
```

```
y = newDataset['class'] # Remove the trailing comma to avoid creating a tuple

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)



!pip install tensorflow



import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(x_train.shape[1],)))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

loss_functions = {
    'binary_crossentropy': 'binary_crossentropy',
    'hinge': 'hinge',
    'focal_loss' : 'binary_crossentropy' #Placeholder for focal loss
    }
def focal_loss(gamma=2.0, alpha=0.25):
    def focal_loss_fixed(y_true, y_pred):
        epsilon = tf.keras.backend.epsilon()
```

```
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        cross_entropy = -y_true * tf.math.log(y_pred)
        loss = alpha * tf.pow(1 - y_pred, gamma) * cross_entropy
        return tf.reduce_mean(tf.reduce_sum(loss, axis=1))
    return focal_loss_fixed



import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers, models

# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"
column_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',
                'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
data = pd.read_csv(url, header=None, names=column_names)

# Preprocess the data
data['class'] = data['class'].map({'g': 1, 'h': 0})  # Convert class labels to binary
X = data.drop('class', axis=1)
y = data['class']

# Split the dataset into majority and minority classes
classG = data[data['class'] == 1]
classH = data[data['class'] == 0]

# Balance the dataset by undersampling the majority class
```

```
countH = len(classH)
classGUnder = classG.sample(countH)
newDataset = pd.concat([classGUnder, classH], axis=0)

# Split the balanced dataset
X = newDataset.drop('class', axis=1)
y = newDataset['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the model
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model with different loss functions
loss_functions = {
    'binary_crossentropy': 'binary_crossentropy',
    'hinge': 'hinge',
    'focal_loss': focal_loss(gamma=2.0, alpha=0.25)  # Use the custom focal loss function
}

# Custom focal loss function
def focal_loss(gamma=2.0, alpha=0.25):
    def focal_loss_fixed(y_true, y_pred):
```

```python
        epsilon = tf.keras.backend.epsilon()
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        cross_entropy = -y_true * tf.math.log(y_pred)
        loss = alpha * tf.pow(1 - y_pred, gamma) * cross_entropy
        return tf.reduce_mean(tf.reduce_sum(loss, axis=1))
    return focal_loss_fixed

# Train and evaluate the model with different loss functions
for loss_name, loss_function in loss_functions.items():
    print(f"Training with loss function: {loss_name}")

    model.compile(optimizer='adam', loss=loss_function, metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Test accuracy with {loss_name}: {test_accuracy:.4f}\n")
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras import layers, models
# Load the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.data"
column_names = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym',
                'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
data = pd.read_csv(url, header=None, names=column_names)
```

```
# Preprocess the data
data['class'] = data['class'].map({'g': 1, 'h': 0})  # Convert class labels to binary
X = data.drop('class', axis=1)
y = data['class']

# Split the dataset into majority and minority classes
classG = data[data['class'] == 1]
classH = data[data['class'] == 0]

# Balance the dataset by undersampling the majority class
countH = len(classH)
classGUnder = classG.sample(countH)
newDataset = pd.concat([classGUnder, classH], axis=0)

# Split the balanced dataset
X = newDataset.drop('class', axis=1)
y = newDataset['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Custom focal loss function
def focal_loss(gamma=2.0, alpha=0.25):
    def focal_loss_fixed(y_true, y_pred):
        epsilon = tf.keras.backend.epsilon()
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        cross_entropy = -y_true * tf.math.log(y_pred)
```

```
        loss = alpha * tf.pow(1 - y_pred, gamma) * cross_entropy
        return tf.reduce_mean(tf.reduce_sum(loss, axis=1))
    return focal_loss_fixed

# Build the model
model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification

# Compile the model with different loss functions
loss_functions = {
    'binary_crossentropy': 'binary_crossentropy',
    'hinge': 'hinge',
    'focal_loss': focal_loss(gamma=2.0, alpha=0.25)  # Use the custom focal loss function
}

# Train and evaluate the model with different loss functions
for loss_name, loss_function in loss_functions.items():
    print(f"Training with loss function: {loss_name}")

    model.compile(optimizer='adam', loss=loss_function, metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Test accuracy with {loss_name}: {test_accuracy:.4f}")

    # Get predictions
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype(int)  # Convert probabilities to binary predictions
```

```
    # Calculate precision, recall, and F1 score
    report = classification_report(y_test, y_pred, target_names=['Class H (0)', 'Class G (1)'], output_dict=True)

    precision = report['Class G (1)']['precision']
    recall = report['Class G (1)']['recall']
    f1_score = report['Class G (1)']['f1-score']

    print(f"Precision for Class G (1): {precision:.4f}")
    print(f"Recall for Class G (1): {recall:.4f}")
    print(f"F1 Score for Class G (1): {f1_score:.4f}\n")



import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are already defined from your previous code
# y_pred contains the binary predictions from the model

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Class H (0)', 'Class G (1)'], yticklabels=['Class H (0)', 'Class G (1)'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.show()
```