Magic Dataset

Introduction

We are supposed to use this 'magic' dataset to generate a prediction of wether the energy particles are of type gamma (g) or hadron (h).

In [22]:
```python
import numpy as np
```

Loading data

We are loading the data from file using numpy genfromtext function. Which will load all the features in a hashmap like data structure to help us easily manipulate the data

In [23]:
```python
data_set = np.genfromtxt('magic04.data', delimiter=',', dtype=None, encoding='utf-8')
data_set
```

Out[23]:
```
array([( 28.7967,  16.0021, 2.6449, 0.3918, 0.1982,   27.7004,   22.011 ,  -8.2027, 40.092 ,  81.882
8, 'g'),
       ( 31.6036,  11.7235, 2.5185, 0.5303, 0.3773,   26.2722,   23.8238,  -9.9574,  6.3609, 205.261
, 'g'),
       (162.052 , 136.031 , 4.0612, 0.0374, 0.0187,  116.741 ,  -64.858 , -45.216 , 76.96  , 256.788
, 'g'),
       ...,
       ( 75.4455,  47.5305, 3.4483, 0.1417, 0.0549,   -9.3561,   41.0562,  -9.4662, 30.2987, 256.516
6, 'h'),
       (120.5135,  76.9018, 3.9939, 0.0944, 0.0683,    5.8043,  -93.5224, -63.8389, 84.6874, 408.316
6, 'h'),
       (187.1814,  53.0014, 3.2093, 0.2876, 0.1539, -167.3125, -168.4558,  31.4755, 52.731 , 272.317
4, 'h')],
      dtype=[('f0', '<f8'), ('f1', '<f8'), ('f2', '<f8'), ('f3', '<f8'), ('f4', '<f8'), ('f5', '<f
8'), ('f6', '<f8'), ('f7', '<f8'), ('f8', '<f8'), ('f9', '<f8'), ('f10', '<U1')])
```

Separating Gammas from Hadrons

In order to make both datasets equal in size we will separate Gs from Hs. Knowing that gamma data is in the first 12332 rows we separate the gamma in a variable called g_class

Gamma data

In [49]:
```python
g_class=data_set[:12332]
g_class
```

Out[49]:
```
array([( 28.7967,  16.0021, 2.6449, 0.3918, 0.1982,  27.7004,  22.011 ,  -8.2027, 40.092 ,  81.8828,
'g'),
       ( 31.6036,  11.7235, 2.5185, 0.5303, 0.3773,  26.2722,  23.8238,  -9.9574,  6.3609, 205.261 ,
'g'),
       (162.052 , 136.031 , 4.0612, 0.0374, 0.0187, 116.741 , -64.858 , -45.216 , 76.96  , 256.788 ,
'g'),
       ...,
       ( 22.0913,  10.8949, 2.2945, 0.5381, 0.2919,  15.2776,  18.2296,   7.3975, 21.068 , 123.281 ,
'g'),
       ( 56.2216,  18.7019, 2.9297, 0.2516, 0.1393,  96.5758, -41.2969,  11.3764,  5.911 , 197.209 ,
'g'),
       ( 31.5125,  19.2867, 2.9578, 0.2975, 0.1515,  38.1833,  21.6729, -12.0726, 17.5809, 171.227 ,
'g')],
      dtype=[('f0', '<f8'), ('f1', '<f8'), ('f2', '<f8'), ('f3', '<f8'), ('f4', '<f8'), ('f5', '<f
8'), ('f6', '<f8'), ('f7', '<f8'), ('f8', '<f8'), ('f9', '<f8'), ('f10', '<U1')])
```

Hadron data

In [25]:
```python
h_class=data_set[12332:]
h_class
```

Out[25]:
```
array([( 93.7035, 37.9432, 3.1454, 0.168 , 0.1011,   53.2566,   89.0566,  11.8175, 14.1224, 231.9028,
'h'),
       (102.0005, 22.0017, 3.3161, 0.1064, 0.0724,  -54.0862,   43.0553, -15.0647, 88.4636, 274.9392,
'h'),
       (100.2775, 21.8784, 3.11  , 0.312 , 0.1446,  -48.1834,   57.6547,  -9.6341, 20.7848, 346.433 ,
'h'),
       ...,
       ( 75.4455, 47.5305, 3.4483, 0.1417, 0.0549,   -9.3561,   41.0562,  -9.4662, 30.2987, 256.5166,
'h'),
       (120.5135, 76.9018, 3.9939, 0.0944, 0.0683,    5.8043,  -93.5224, -63.8389, 84.6874, 408.3166,
'h'),
       (187.1814, 53.0014, 3.2093, 0.2876, 0.1539, -167.3125, -168.4558,  31.4755, 52.731 , 272.3174,
'h')],
      dtype=[('f0', '<f8'), ('f1', '<f8'), ('f2', '<f8'), ('f3', '<f8'), ('f4', '<f8'), ('f5', '<f
8'), ('f6', '<f8'), ('f7', '<f8'), ('f8', '<f8'), ('f9', '<f8'), ('f10', '<U1')])
```

Making Gs the same size as Hs

We are taking a random 6688 rows from the gamma array to make both datasets equal

In [26]:
```python
g_class=np.random.choice(g_class,size=6688,replace=False)
print(g_class)
print(g_class.shape)
```

```
[(28.7043, 22.2965, 3.0396, 0.2766, 0.1401,  -28.2185, -14.4249, 12.2056, 61.727 , 148.541, 'g')
 (71.3468, 25.1598, 3.0253, 0.2047, 0.1203, -120.403 ,  51.7147, 14.6913,  8.6575, 211.858, 'g')
 (21.561 ,  6.7887, 2.07  , 0.7149, 0.4213,   22.5915,  13.3083, -6.4567, 10.302 , 180.155, 'g')
 ...
 (12.9774, 11.0775, 2.1945, 0.7476, 0.4441,  -14.3096,   6.7507,  9.7284, 68.4457, 212.57 , 'g')
 (21.2858, 11.4472, 2.2122, 0.5828, 0.3098,  -11.4379, -19.7686, -8.6863, 50.0471, 238.41 , 'g')
 (28.971 ,  7.5578, 2.2923, 0.5816, 0.3291,   12.3711,  -7.6712, -5.4745, 41.669 , 110.649, 'g')]
(6688,)
```

Constructing a new data array

We will concatinate the new shortened g_class with the h_class to have our new dataset with equal parameters for both predections.

In [27]:
```python
data=np.concatenate((g_class,h_class),axis=0)
print(data)
print(data.shape)
```

```
[( 28.7043, 22.2965, 3.0396, 0.2766, 0.1401,  -28.2185,  -14.4249,  12.2056, 61.727 , 148.541 , 'g')
 ( 71.3468, 25.1598, 3.0253, 0.2047, 0.1203, -120.403 ,   51.7147,  14.6913,  8.6575, 211.858 , 'g')
 ( 21.561 ,  6.7887, 2.07  , 0.7149, 0.4213,   22.5915,   13.3083,  -6.4567, 10.302 , 180.155 , 'g')
 ...
 ( 75.4455, 47.5305, 3.4483, 0.1417, 0.0549,   -9.3561,   41.0562,  -9.4662, 30.2987, 256.5166, 'h')
 (120.5135, 76.9018, 3.9939, 0.0944, 0.0683,    5.8043,  -93.5224, -63.8389, 84.6874, 408.3166, 'h')
 (187.1814, 53.0014, 3.2093, 0.2876, 0.1539, -167.3125, -168.4558,  31.4755, 52.731 , 272.3174, 'h')]
(13376,)
```

Randomizing data by shuffle and splitting

We are shuffling all our data using numpy

Spliting data to Training, Testing, Validation data sets

In [28]:
```python
rng = np.random.default_rng()
rng.shuffle(data)
train,test_validate=np.array_split(data,[int(0.70 * len(data))])
test,validation=np.array_split(test_validate,[int(0.50 * len(test_validate))])
```

In [29]:
```python
print(f"train: {train.shape[0]}\n"
      f"test: {test.shape[0]}\n"
      f"validation: {validation.shape[0]}")
```

```
train: 9363
test: 2006
validation: 2007
```

Separating features from the perdiction

We are taking the first ten features and putting them in array x_data and the last feature (the predection) in array y_data

In [33]:
```python
def return_x_y_arrays(data_set_to_be_sliced):
      buf=data_set_to_be_sliced.tolist()
      x_data= []
      y_data=[]
      for d in buf:
            x=d[:10]
            x_data.append(x)
            y=np.array([d[10]])
            y_data.append(y)
      x_data=np.array(x_data)
      y_data=np.array(y_data).ravel()
      return x_data,y_data
```

In [34]:
```python
x_train,y_train=return_x_y_arrays(train)
x_validation,y_validation=return_x_y_arrays(validation)
x_test,y_test=return_x_y_arrays(test)
```

Fitting and training the model

In [35]:
```python
from sklearn.neighbors import KNeighborsClassifier as knn
model=knn(n_neighbors=1)
model.fit(x_train, y_train)
```

Out[35]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Scoring the model using test and validation datasets

We will use various K values to score the model

k=1 k=3 k=10 k=23 -> Best k=600

In [36]:
```python
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
```

```
0.7532402791625125
0.7478824115595416
```

In [50]:
```python
model.n_neighbors=3
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
```

```
0.7701894317048853
0.7593423019431988
```

In [37]:
```python
model.n_neighbors=10
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
```

```
0.7666999002991027
0.7623318385650224
```

In [56]:
```python
model.n_neighbors=23
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
```

```
0.7791625124626121
0.7678126557050324
```

In [38]:
```python
model.n_neighbors=600
print(model.score(x_test,y_test))
print(model.score(x_validation,y_validation))
```

```
0.7298105682951147
0.726457399103139
```

In [52]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
KN = KNeighborsClassifier()
k_range = list(range(1, 26, 2)) # 1, 3, 5, ..., 25
param_grid = dict(n_neighbors=k_range)
print(param_grid)
grid = GridSearchCV(KN, param_grid, cv=10, scoring='accuracy', return_train_score=True)

# Training data
g = grid.fit(x_train, y_train)
print(grid.best_params_)
accuracy=grid.best_score_ * 100
print("Accuracy for our training dataset with tuning is : {:.2f}%".format(accuracy) )
```

```
{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25]}
{'n_neighbors': 23}
Accuracy for our training dataset with tuning is : 77.11%
```

In [53]:
```python
best_k = grid.best_params_['n_neighbors']

# Calculation test accuracy score
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(x_train,y_train)
y_test_pred = knn.predict(x_test)
test_accuracy = knn.score(x_test, y_test)
print(test_accuracy)

# Calculation validation accuracy score
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(x_train,y_train)
y_validation_pred = knn.predict(x_validation)
test_accuracy = knn.score(x_validation, y_validation)
print(test_accuracy)
```

```
0.7791625124626121
0.7678126557050324
```

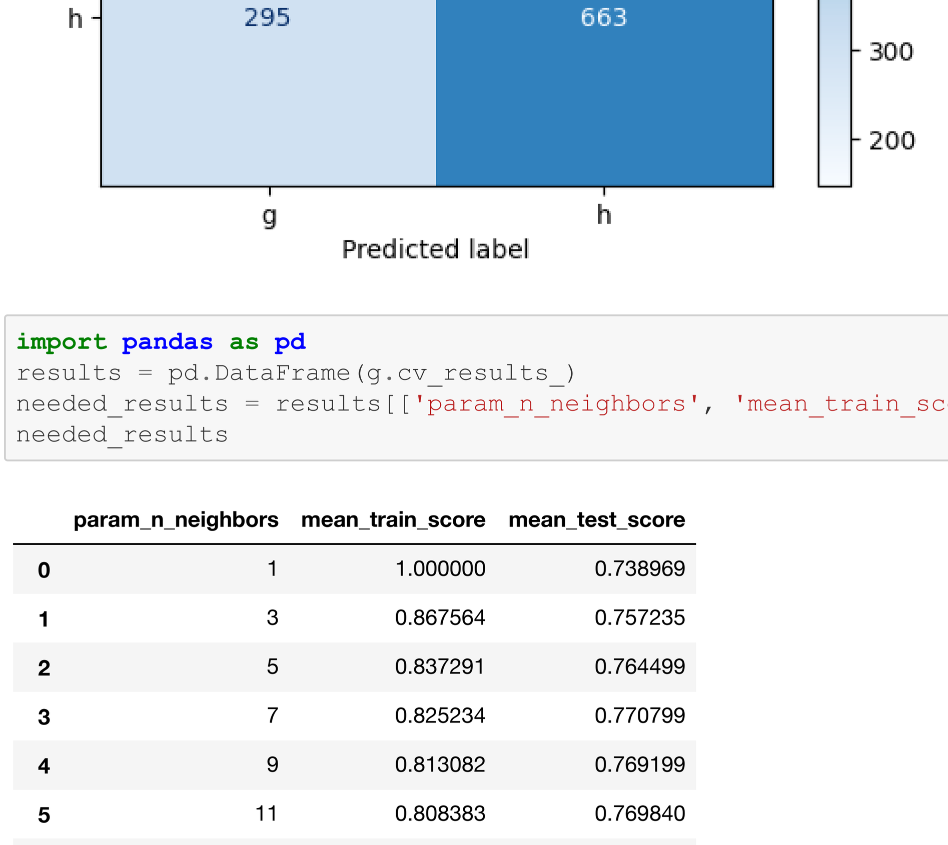
Using confusion matrix to display the results in human-readable graphic

In [57]:
```python
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
matix = confusion_matrix(y_test,y_test_pred)
print(classification_report(y_test,y_test_pred))
disp = ConfusionMatrixDisplay(confusion_matrix=matix,display_labels=['g','h'])
disp = disp.plot(cmap=plt.cm.Blues)
```

```
              precision    recall  f1-score   support

           g       0.75      0.86      0.80      1048
           h       0.82      0.69      0.75       958

    accuracy                           0.78      2006
   macro avg       0.79      0.78      0.78      2006
weighted avg       0.78      0.78      0.78      2006
```



In [55]:
```python
import pandas as pd
results = pd.DataFrame(g.cv_results_)
needed_results = results[['param_n_neighbors', 'mean_train_score', 'mean_test_score']]
needed_results
```

Out[55]:

	param_n_neighbors	mean_train_score	mean_test_score
0	1	1.000000	0.738969
1	3	0.867564	0.757235
2	5	0.837291	0.764499
3	7	0.825234	0.770799
4	9	0.813082	0.769199
5	11	0.808383	0.769840
6	13	0.803387	0.769839
7	15	0.799708	0.770052
8	17	0.797441	0.770481
9	19	0.794902	0.770373
10	21	0.791591	0.768985
11	23	0.789419	0.771123
12	25	0.787616	0.767597