Project Overview

Machine Learning: Supervised and Unsupervised learning

Data Understanding

Features

1. fLength: continuous # major axis of ellipse [mm]
2. fWidth: continuous # minor axis of ellipse [mm]
3. fSize: continuous # 10-log of sum of content of all pixels [in #phot]
4. fConc: continuous # ratio of sum of two highest pixels over fSize [ratio]
5. fConc1: continuous # ratio of highest pixel over fSize [ratio]
6. fAsym: continuous # distance from highest pixel to center, projected onto major axis [mm]
7. fM3Long: continuous # 3rd root of third moment along major axis [mm]
8. fM3Trans: continuous # 3rd root of third moment along minor axis [mm]
9. fAlpha: continuous # angle of major axis with vector to origin [deg]
10. fDist: continuous # distance from origin to center of ellipse [mm]
11. class: g,h # gamma (signal), hadron (background)

g = gamma (signal): 12332

h = hadron (background): 6688

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Dataset: The data set was generated by a Monte Carlo program, Corsika, described in: D. Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998).

```
pd.read_csv("/content/magic04.data")
```

	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.011	-8.2027	40
0	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6

```
cols = ["fLength", "fWidth", "fSize", "fConc", "fConc1", "fAsym", "fM3Long", "fM3Trans", "fAlpha", "fDist", "class"]
df = pd.read_csv("/content/magic04.data", names = cols)
df.head()
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480

```
df.shape
```

```
(19020, 11)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19020 entries, 0 to 19019
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   fLength   19020 non-null  float64
 1   fWidth    19020 non-null  float64
 2   fSize     19020 non-null  float64
 3   fConc     19020 non-null  float64
 4   fConc1    19020 non-null  float64
 5   fAsym     19020 non-null  float64
 6   fM3Long   19020 non-null  float64
 7   fM3Trans  19020 non-null  float64
 8   fAlpha    19020 non-null  float64
 9   fDist     19020 non-null  float64
 10  class     19020 non-null  object
dtypes: float64(10), object(1)
memory usage: 1.6+ MB
```

```
df.isna().sum()
```

```
fLength     0
fWidth      0
fSize       0
fConc       0
fConc1      0
fAsym       0
fM3Long     0
fM3Trans    0
fAlpha      0
fDist       0
class       0
dtype: int64
```

The dataset is clean, it has no missing values

```
df["class"].unique()
```

```
array(['g', 'h'], dtype=object)
```

```
#transforming the class feature
df["class"] = (df["class"] == 'g').astype(int)
```

```
df['class'].unique()
```

```
array([1, 0])
```

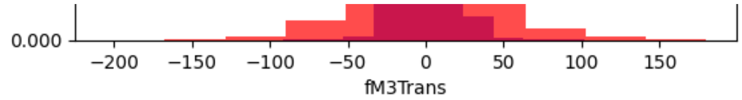
1 = gamma

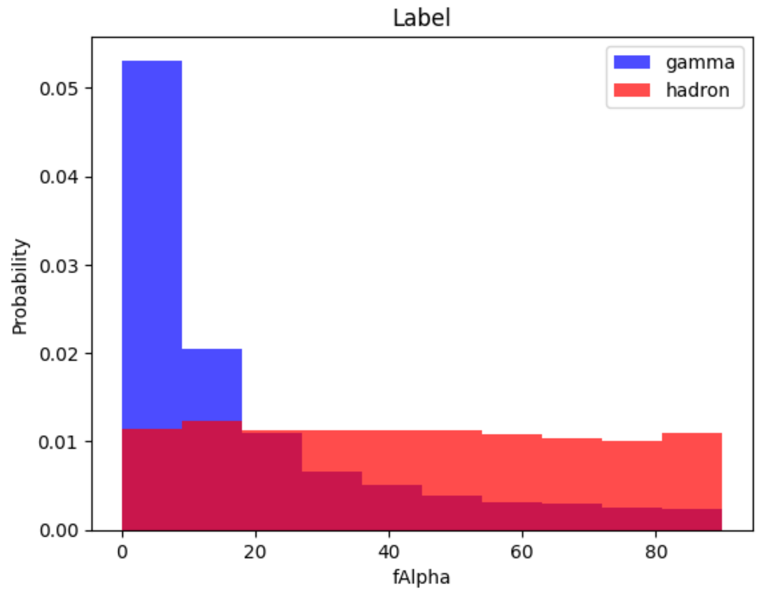
0 = hadron

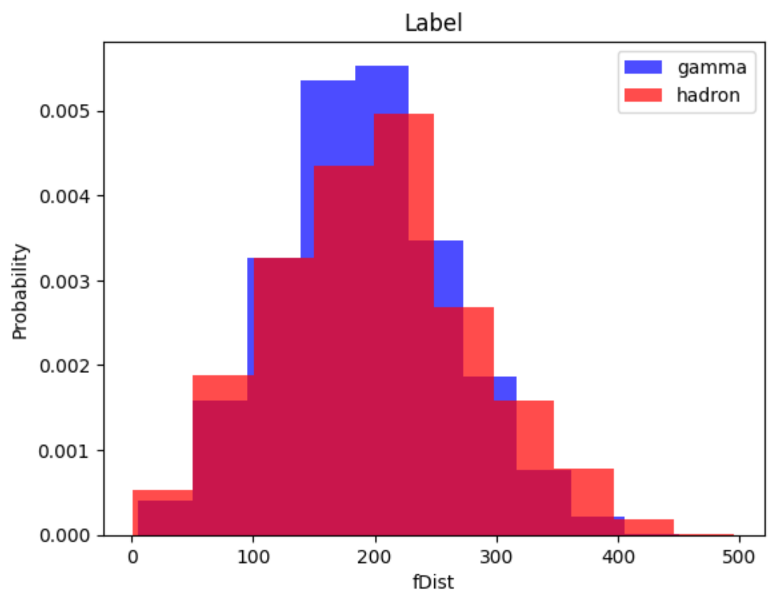
```
df.tail()
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAl
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7

```
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()
```


0.000
−200 −150 −100 −50 0 50 100 150
fM3Trans


Label
gamma
hadron
Probability
0.05 0.04 0.03 0.02 0.01 0.00
0 20 40 60 80
fAlpha


Label
gamma
hadron
Probability
0.005 0.004 0.003 0.002 0.001 0.000
0 100 200 300 400 500
fDist

Splitting the data into Train, validation and Test Set

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

```
#scale of the features is way off

from sklearn.preprocessing import StandardScaler
#oversample our training dataset by increasing the number of hadron
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, (len(y),1))))
  return data, X, y
```

```
print(len(train[train["class"]==1]))
print(len(train[train["class"]==0]))
```

```
7373
4039
```

```
train, X_train, y_train = scale_dataset(train, oversample=True)
```

```
valid, X_valid, y_valid = scale_dataset(valid, oversample =False)
test, X_test, y_test = scale_dataset(test, oversample=False)
```

Model

K-Nearest-Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred = knn_model.predict(X_test)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      1327
           1       0.86      0.87      0.87      2477

    accuracy                           0.82      3804
   macro avg       0.81      0.80      0.81      3804
weighted avg       0.82      0.82      0.82      3804
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
y_pred = nb_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.68      0.41      0.51      1327
           1       0.74      0.90      0.81      2477

    accuracy                           0.73      3804
   macro avg       0.71      0.65      0.66      3804
weighted avg       0.72      0.73      0.70      3804
```

Logistic Regression

```
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)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.72      0.70      1327
           1       0.84      0.81      0.83      2477

    accuracy                           0.78      3804
   macro avg       0.76      0.77      0.76      3804
weighted avg       0.78      0.78      0.78      3804
```

Support Vector Machines

```
from sklearn.svm import SVC
```

```
svm_model = SVC()
svm_model = svm_model.fit(X_train, y_train)
```

```
y_pred = svm_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80      1327
           1       0.89      0.89      0.89      2477

    accuracy                           0.86      3804
   macro avg       0.84      0.84      0.84      3804
weighted avg       0.86      0.86      0.86      3804
```

```
#svm has the highest accuracy
```