Binary Joint Use Case (Single DataFrameCase)

In this vignette a use case of the Binary Channel Entropy Triangle is presented. We are going to evaluate different multiclass-classification scenarios in order to analyze the data. The main functionalities for the classification of the database will be extracted from: [https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/ (https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/)](https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/)

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Download the databases

In this case, the csv files for the use case, are stored locally

In [2]:

```
#df = pd.read_csv('Arthitris.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop([
'Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing val
ues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
#df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 699 entries, 1 to 699
Data columns (total 10 columns):
Clump Thickness                699 non-null int64
Uniformity of Cell Size        699 non-null int64
Uniformity of Cell Shape       699 non-null int64
Marginal Adhesion              699 non-null int64
Single Epithelial Cell Size    699 non-null int64
Bare Nuclei                    683 non-null float64
Bland Chromatin                699 non-null int64
Normal Nucleoli                699 non-null int64
Mitoses                        699 non-null int64
Class                          699 non-null object
dtypes: float64(1), int64(8), object(1)
memory usage: 60.1+ KB
```

In [4]:

```
df = df.fillna(0)
df.head(5)
```

Out[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitose
1	5	1	1	1	2	1.0	3	1	
2	5	4	4	5	7	10.0	3	2	
3	3	1	1	1	2	2.0	3	1	
4	6	8	8	1	3	4.0	3	7	
5	4	1	1	3	2	1.0	3	1	

Prepare the data for the classification (Features - Classes)

We are going to load the train_test_split that will allow us to separe automatically the data in a train/test sets. Additionally, we are going to import the contingency matrix that will allow us to calculate the joint entropy matrix of the classifier

In [5]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Separating the dataframe for features and classes

In [6]:

```
X = df[df.columns[df.columns != 'Class']]
y = df['Class']
```

We are now to define some classificators for evaluating their performance with the BreastCancer database

In [7]:

```
# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

KNN

KNN - Classifier (Don´t run the code if you want to implement other classifier)

Downloading the sklearn Knn classifier and fitting it into our data

In [8]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
```

Out[8]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

Once we have design our classifier, we are going to evaluate the accuracy

In [9]:

```
print(knn.score(X_test, y_test))
```

```
0.9771428571428571
```

Finally, we will compute the confusion matrix of the classified data

In [10]:

```
knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
cm
```

Out[10]:

```
array([[110,   2],
       [  2,  61]])
```

KNN - Channel Bivariate Entropy Triangle Plotting

The last step will be calculating the entropic measures for the contingency matrix and plot the entropy triangle. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle, and will appear behind the triangle plot for comparission

In [11]:

```
edf = jentropies_binary(cm)
#edf1 = jentropies(pd.DataFrame(y_test),pd.DataFrame(knn_predictions))
```

In [12]:

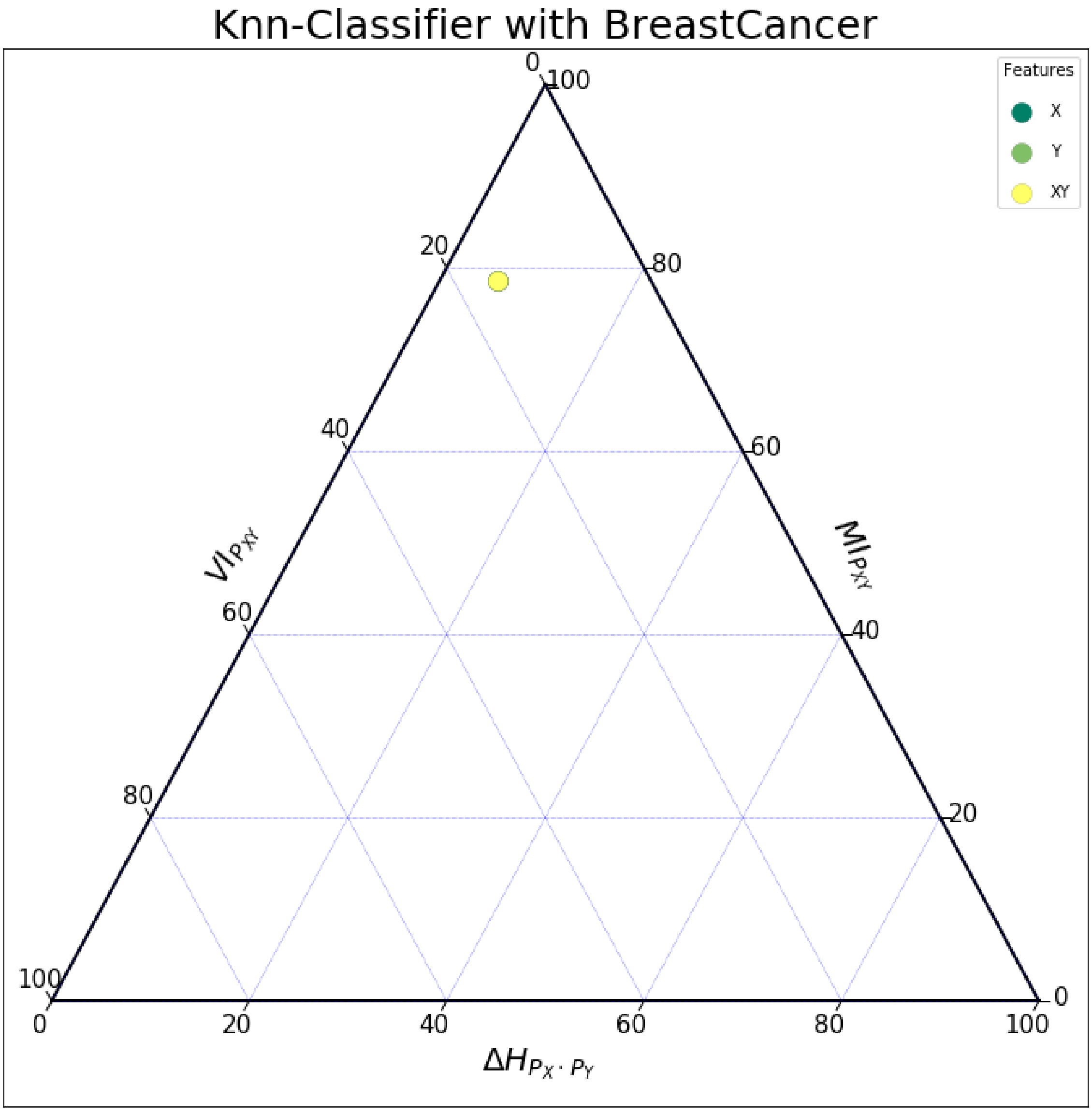
```
edf
#edf1
```

Out[12]:

	H_U2	H_P2	DeltaH_P2	M_P2	VI_P2
Type					
X	1.0	0.942683	0.057317	0.786867	0.155816
Y	1.0	0.942683	0.057317	0.786867	0.155816
XY	2.0	1.885366	0.114634	1.573734	0.311632

In [13]:

```
entriangle(edf,s_mk=150, gridl = 20, pltscale=12 ,fonts = 20, ticks_size= 15,cha
rt_title="Knn-Classifier with BreastCancer")
```



Naive Bayes

Naive Bayes - Classificator

Now we are going to provide another example of the channel bivariate entropy triangle using the Naive Bayes classificatior. We will need first to download the GaussianNB class from scikit-learn and apply the following command to train the classifier

In [14]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train, y_train)
```

Once we have model our classifier, we are going to evaluate the accuracy using the test set

In [15]:

```
print(gnb.score(X_test, y_test))
```

```
0.9542857142857143
```

Finally, we will compute the confusion matrix of the classified data

In [16]:

```
gnb_predictions = gnb.predict(X_test)
cm = confusion_matrix(y_test, gnb_predictions)
cm
```

Out[16]:

```
array([[106,   6],
       [  2,  61]])
```

Naive Bayes - Channel Bivariate Entropy Triangle Plotting

The last step will be calculating the entropic measures for the contingency matrix and plot the entropy triangle. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle. First we will calculate the entropy data frame for the contingency matrix

In [17]:

```
edf = jentropies_binary(cm)
```

In [18]:

```
edf
```

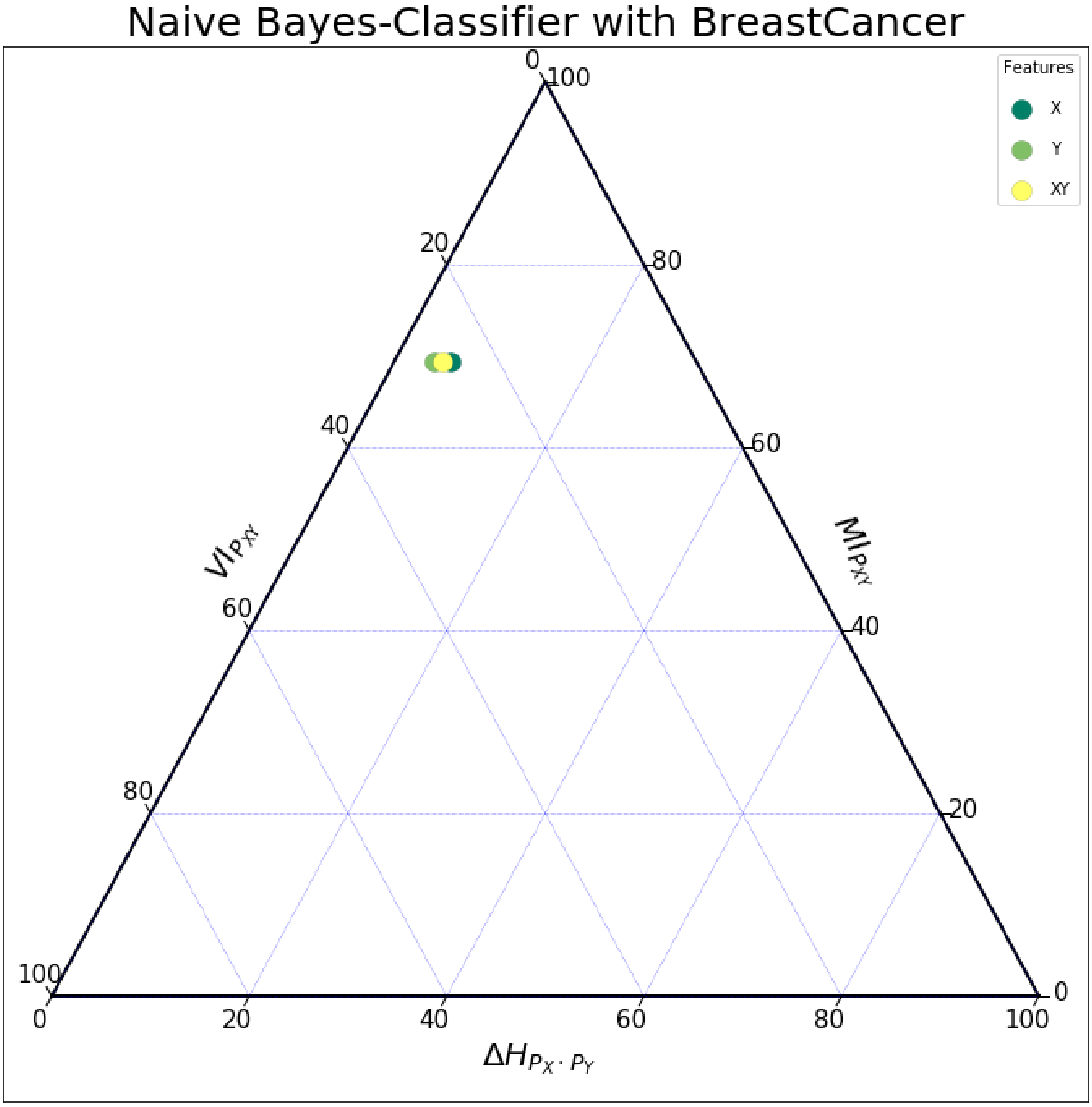
Out[18]:

	H_U2	H_P2	DeltaH_P2	M_P2	VI_P2
Type					
X	1.0	0.942683	0.057317	0.694046	0.248637
Y	1.0	0.960035	0.039965	0.694046	0.265989
XY	2.0	1.902718	0.097282	1.388092	0.514626

Once we obtained the appropiate entropy data frame, we just need to execute the entriangle function for the plotting

In [19]:

```
entriangle(edf,s_mk=150, gridl = 20, pltscale=12 ,fonts = 20, ticks_size= 15,cha
rt_title="Naive Bayes-Classifier with BreastCancer")
```



Naive Bayes-Classifier with BreastCancer
Features
X
Y
XY
0
100
20
80
40
60
$VI_{P_{XY}}$
60
40
$MI_{P_{XY}}$
80
20
100
0
0
20
40
60
80
100
$\Delta H_{P_X \cdot P_Y}$

Multivariate Joint Use Case (Single DataFrameCase)

In this vignette a use case of the Multivariate Channel Entropy Triangle is presented. We are going to evaluate the effectiveness of feature transformation using PCA in entropic terms.

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation.

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Download the databases

In this case, the csv files for the use case, are stored locally. Now it´s time to load the database in which we are going to apply the feature transformation.

In [2]:

```
#df = pd.read_csv('Arthitris.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
#df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop
(['Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing v
alues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

In [4]:

```
df.head(5)
```

Out[4]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
Id					
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:

```
df = discretization(df).fillna(0)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:35: UserWarning: Discretizing data!
  warning("Discretizing data!")
```

Prepare the data for the PCA feature transformation (Features - Classes)

Importing the Sklearn modules for the feature transformation.

In [6]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Splitting the Data for the Standarization of the features before the transformation.

In [7]:

```
features = df.columns.drop('Species')
x = df[df.columns.drop('Species')].values
# Separating out the target
y = df.loc[:,['Species']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/sklea
rn/utils/validation.py:475: DataConversionWarning: Data with input d
type object was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Transformation of the data. We will store the entropy dataframes in a list, which will store in each position the transformed features with the corresponding number of number of principal components which will be:

Number of cols of original df - index

Example list[0] = Feature transformation with (iris features cols = 4) - (index = 0) = 4 Principal components

In [8]:

```
li = list()
for i in range(len(df.columns)):
    pca = PCA(n_components = (len(df.columns)-1)-i)
    principalComponents = pca.fit_transform(x)
    columns = list(map(lambda x: "principal component " + str(x), range(len(df.c
olumns)-1-i)))
    principalDf = pd.DataFrame(data = principalComponents, columns= columns)
    li.append(principalDf)
```

Channel Multivariate Entropy Triangle

Calculation of the entropy Data Frame for each of the dataframes of the list:

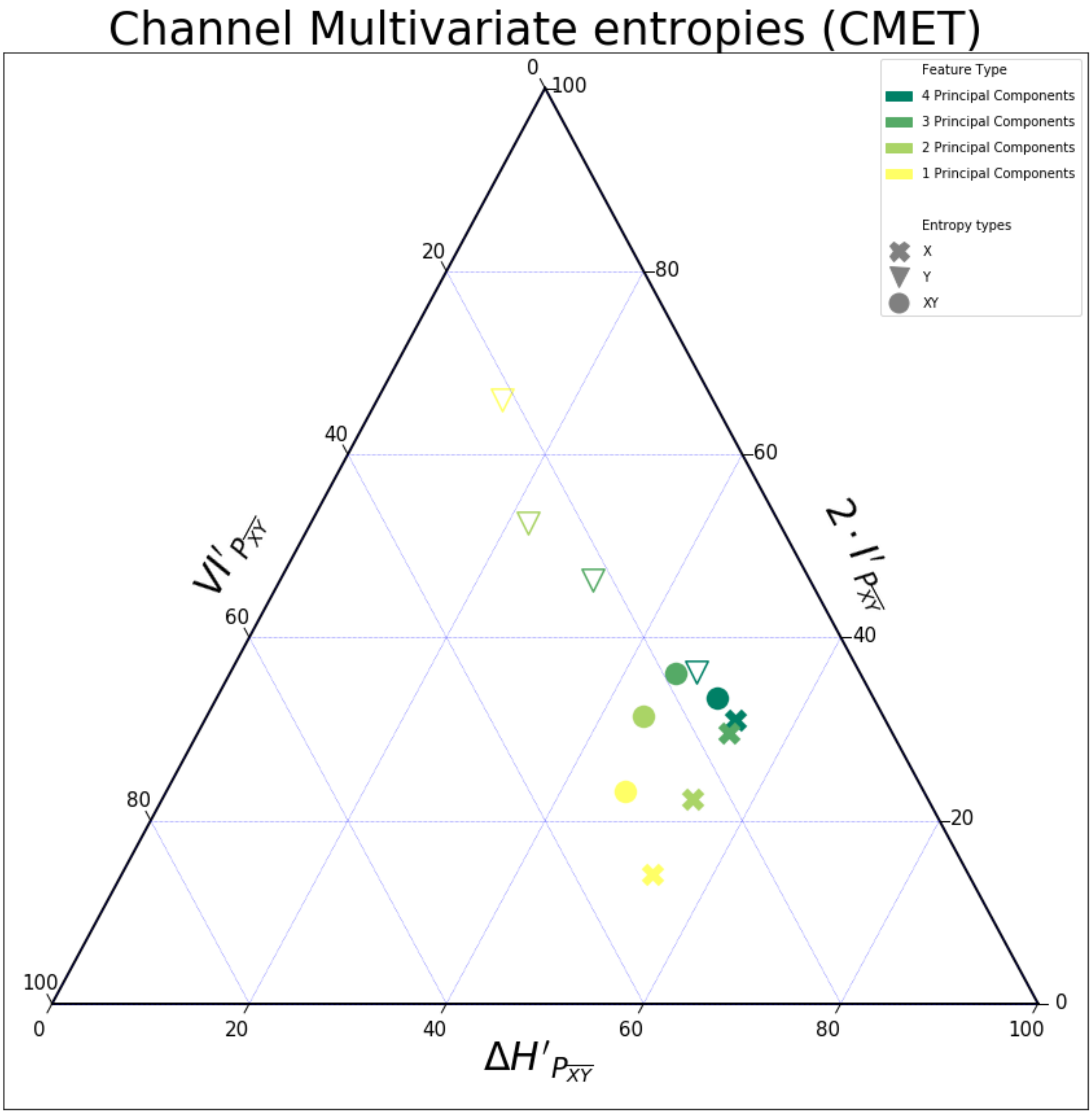
In [9]:

```
edf = list()
for i in range(len(li)-1):
    edf.append(jentropies(df,li[i]))
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/jentropies.py:50: UserWarning: Discretizing data from X D
ataFrame before entropy calculation!
  warning("Discretizing data from X DataFrame before entropy calcula
tion!") #' Throwing a Warning for communicating a discretization of
data
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:35: UserWarning: Discretizing data!
  warning("Discretizing data!")
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/jentropies.py:54: UserWarning: Discretizing data from X D
ataFrame before entropy calculation!
  warning("Discretizing data from X DataFrame before entropy calcula
tion!") #' Throwing a Warning for communicating a discretization of
data
```

In [10]:

```
entriangle_list(edf,s_mk=300,pltscale=15)
```



We can see that using three components maximizes the per-feature transmitted information in the case of Iris.

Multivariate Source Use Case (Single DataFrameCase)

In this vignette I will represent a use case for the Source Multivariate Entropy Triangle with some individual Databases.

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation.

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Dowloading a set of Databases

In this case, the csv files for the use case, are stored locally.

In [2]:

```
#df = pd.read_csv('Arthritis.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
#df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop
(['Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing v
alues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

In [4]:

```
df.head(10)
```

Out[4]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
Id					
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

Discretizing the Data before entropy calculation

We have defined a function for discretizing a whole dataset, the function divides de entries in "NROWS(DF)^(1/3)" equally sized spaces, and turns the original continuous variables into categorical variables.

In [5]:

```
df = discretization(df)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:51: UserWarning: Discretizing data!
  warning("Discretizing data!")
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null category
SepalWidthCm     150 non-null category
PetalLengthCm    150 non-null category
PetalWidthCm     150 non-null category
Species          150 non-null category
dtypes: category(5)
memory usage: 2.8 KB
```

In [7]:

```
df.head(10)
```

Out[7]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	3	0	0	0
2	0	2	0	0	0
3	0	2	0	0	0
4	0	2	0	0	0
5	0	3	0	0	0
6	1	3	0	0	0
7	0	2	0	0	0
8	0	2	0	0	0
9	0	1	0	0	0
10	0	2	0	0	0

Source Entropies Measures calculation

Once we have our data discretized, we will start by calculating the values of the entropies for the posterior plots:

In [8]:

```
'''
As the database is previosly discretized we won´t need the values of the bins
'Type variable select the entropy calculation:'
    Total: Total source entropy decomposition (CPx)
    Dual : Dual source entropy decomposition (DPx instead of CPx)
'''

edf = sentropies(df , type = 'total' , base = 2)
```

In [9]:

```
edf
```

Out[9]:

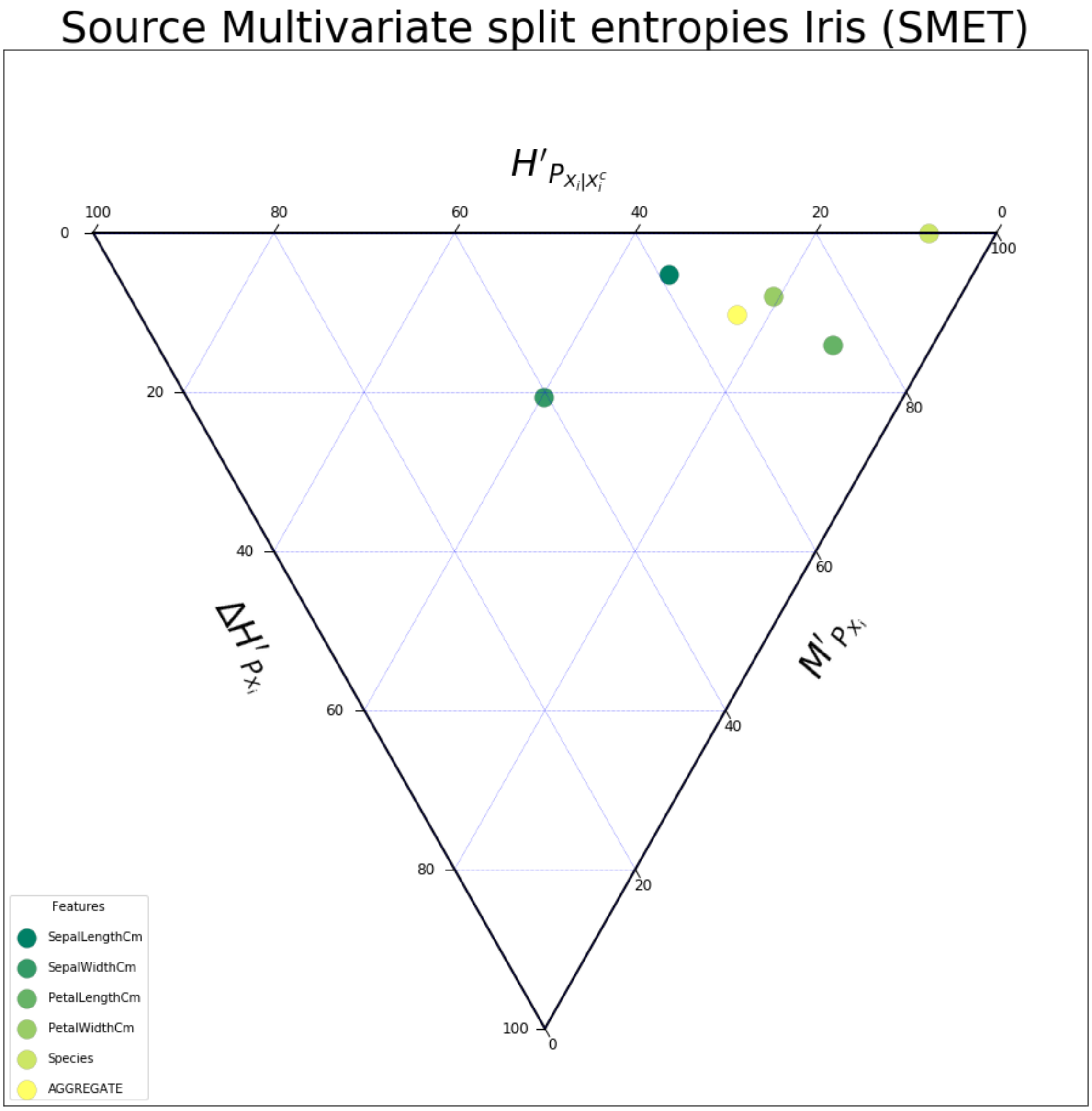
Name	H_Uxi	H_Pxi	DeltaH_Pxi	M_Pxi	VI_Pxi
SepalLengthCm	2.321928	2.200620	0.121308	1.417675	0.782945
SepalWidthCm	2.321928	1.841723	0.480205	0.917768	0.923955
PetalLengthCm	2.321928	1.995571	0.326357	1.738118	0.257453
PetalWidthCm	2.321928	2.137460	0.184468	1.654826	0.482635
Species	1.584963	1.584963	0.000000	1.465241	0.119721
AGGREGATE	10.872675	9.760337	1.112338	7.193628	2.566709

Source Entropies Entropy Triangle Plotting

The last step will be plotting the values calculated previously. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle, and will appear behind the triangle plot for comparison:

In [10]:

```
entriangle(edf,s_mk=250,scale= 100, pltscale=16 , ticks_size=12, gridl = 20, cha
rt_title = "Source Multivariate split entropies Iris (SMET)")
```


Source Multivariate split entropies Iris (SMET)
$H'_{P_{X_i|X_i^c}}$
$\Delta H'_{P_{X_i}}$
$M'_{P_{X_i}}$
100
80
60
40
20
0
Features
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
Species
AGGREGATE

Notice the varying degrees of redundanvy and balancedness of the different features (including the class).