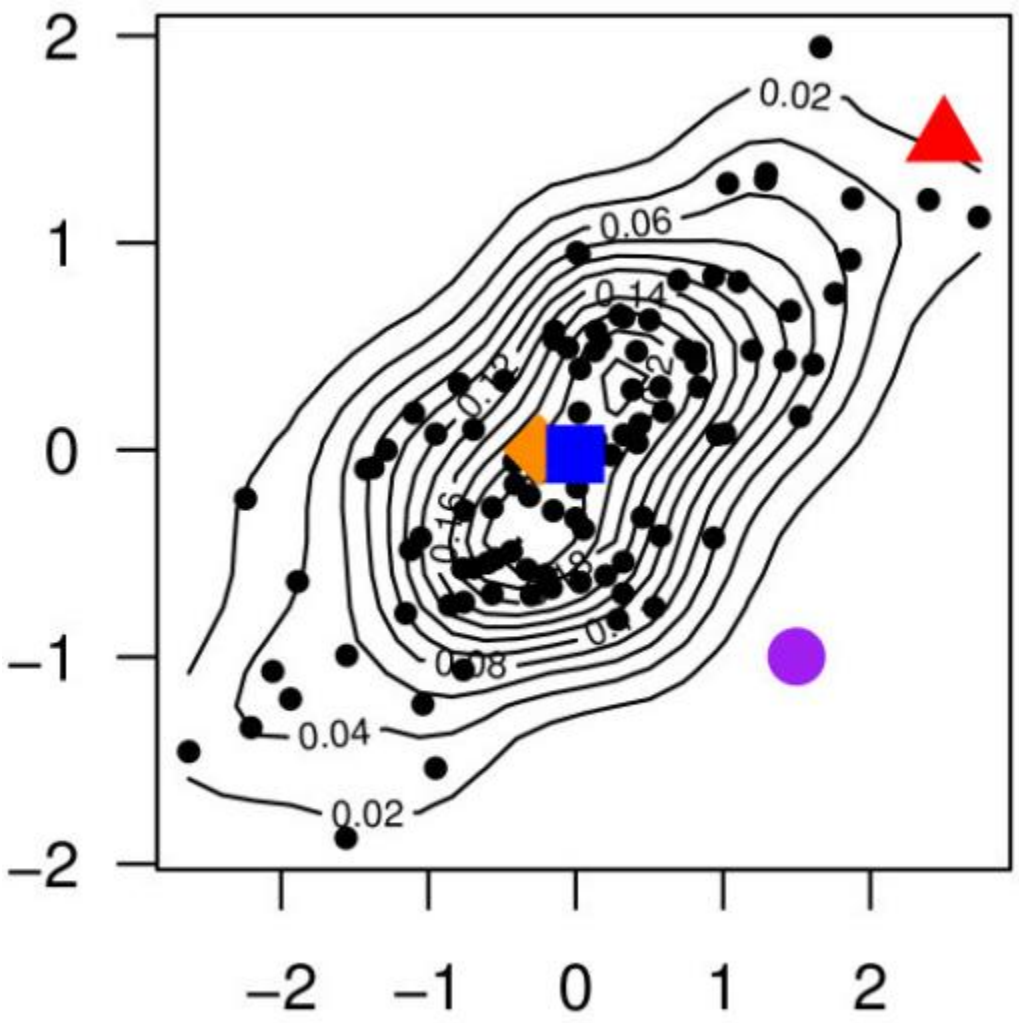
What is the Mahalanobis distance?

The Mahalanobis distance (MD) is **the distance between two points in multivariate space**. In a regular Euclidean space, variables (e.g. x, y, z) are represented by axes drawn at right angles to each other; The distance between any two points can be measured with a ruler. For uncorrelated variables, the Euclidean distance equals the MD. However, if two or more variables are correlated, the axes are no longer at right angles, and the measurements become impossible with a ruler. In addition, if you have more than three variables, you can't plot them in regular 3D space at all. The MD solves this measurement problem, as it measures distances between points, even correlated points for multiple variables.



The Mahalanobis distance measures distance relative to the centroid — a base or central point which can be thought of as an overall mean for multivariate data. The centroid is a point in multivariate space where all means from all variables intersect. The larger the MD, the further away from the centroid the data point is.

Formal Definition

The Mahalanobis distance between two objects is defined (Varmuza & Filzmoser, 2016, p.46) as:
d (Mahalanobis) = $[(x_B - x_A)^T * C^{-1} * (x_B - x_A)]^{0.5}$
Where**:**
x_A and x_B is a pair of objects, and
C **is the sample covariance matrix.**

Another version of the formula, which uses distances from each observation to the central mean:
$d<x_{i=[xi} - \bar{x})^t C^{-1}(x_i - \bar{x})]^{0.5}$
Where:
x_i = an object vector
$\bar{x}$ = arithmetic mean vector</x

Wine data set

The wine dataset is a classic and very easy multi-class classification dataset.

Classes	3
Samples per class	[59,71,48]
Samples total	178
Dimensionality	13
Features	real, positive

Params:

return_X_y – If True, returns ``(data, target)`` instead of a Bunch object. See below for more information about the `data` and `target` object.

Returns:

Dictionary-like object, the interesting attributes are: 'data', the data to learn, 'target', the classification labels, 'target_names', the meaning of the labels, 'feature_names', the meaning of the features, and 'DESCR', the full description of the dataset.

Load and split data set

```
def get_datas():
    return load_wine(return_X_y=True)
```

Cluster

```
def get_label_dict(X_train, y_train) -> dict:
    return {label: X_train[y_train == label] for label in np.unique(y_train)}
```

Mean on each group

```
def get_mean(dict_part, part):
    m = np.mean(dict_part, axis=0)
    return {part: m}


def get_means(dict, part):
    kmeans = KMeans(n_clusters=2)
    kmeans.fit(dict)
    labels = kmeans.labels_
    new_dict = get_label_dict(dict, labels)
    means = []
    for key, value in new_dict.items():
        means.append(get_mean(value, part))
    return means
```

Covariance inverse on each group

```
def get_covariance_inverse(dict_part, part):
    inv_cov = sp.linalg.inv(np.cov(dict_part.T))
    return {part: inv_cov}
```

mahalanobis distance

```
def mahalanobis(x, m, inv_cov):
    x_minus_m = x - m
    left_term = np.dot(x_minus_m, inv_cov)
    mahal = np.dot(left_term, x_minus_m.T)
    return mahal

def get_distance(x=None, y=None, x_test=None):
    dist = []
    new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(x, y,
test_size=0.7)
    if x_test is not None:
        new_X_test = x_test
    dict = get_label_dict(new_X_train, new_y_train)
    means = []
    inv_cov = {}
    for key in dict.keys():
        means.extend(get_means(dict[key], key))
        inv_cov={**inv_cov,**get_covariance_inverse(dict[key], key)}

    for i in range(0, len(new_X_test)):
        a = cluster_on_mahalanobis_distance(new_X_test[i], means=means,
inv_covs=inv_cov)
        dist.append(a)
    get_confusion_matrix(y_test=new_y_test, dist=dist)
    return dist
```

Cluster using mahanobis distance

```
def cluster_on_mahalanobis_distance(x, means: List[dict] = None, inv_covs=None):
    mahal = []
    for mean in means:
        mahal.append(mahalanobis(x, list(mean.values())[0],
inv_covs[list(mean.keys())[0]]))
        # mahal.append(mahalanobis(x, list(mean.values())[0]))
    mi_dist = mahal.index(min(mahal))
    return list(means[mi_dist].keys())[0]
```

euclidean distance

```
def euclidean(x, m):
    x_minus_m = x - m
    euclid = np.dot(x_minus_m, x_minus_m.T)
    return euclid
```

```
ef get_distance_euclidean(x=None, y=None, x_test=None):
    dist = []
    new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(x, y,
test_size=0.7)
    if x_test is not None:
        new_X_test = x_test
    dict = get_label_dict(new_X_train, new_y_train)
    means = []
    for key in dict.keys():
        means.extend(get_means(dict[key], key))
    for i in range(0, len(new_X_test)):
        a = cluster_on_euclidean_distance(new_X_test[i], means=means)
        dist.append(a)
    get_confusion_matrix(y_test=new_y_test,dist=dist)
    return dist
```

Cluster using euclidean distance

```
def cluster_on_euclidean_distance(x, means: List[dict] = None):
    euclid = []
    for mean in means:
        euclid.append(euclidean(x, list(mean.values())[0]))
        # mahal.append(mahalanobis(x, list(mean.values())[0]))
    min_dist = euclid.index(min(euclid))
    return list(means[min_dist].keys())[0]
```

Check correctness present

```
def get_confusion_matrix(dist):
    print('Confusion Matrix :')
    print(confusion_matrix(y_test,dist))
    print ('Accuracy Score :',accuracy_score(y_test,dist))
Confusion Matrix
```

Confusion matrix

Compute confusion matrix to evaluate the accuracy of a classification.

By definition a confusion matrix \(C\) is such that \(C_{i, j}\) is equal to the number of observations known to be in group \(i\) and predicted to be in group \(j\).

Thus in binary classification, the count of true negatives is \(C_{0,0}\), false negatives is \(C_{1,0}\), true positives is \(C_{1,1}\) and false positives is \(C_{0,1}\).

Result

data_shape	test_size	mahalanobis_accuracy	euclidean Accuracy
[53,13]	0.7	0.848	0.704

Mahanobis confusion matrix:

Confusion Matrix :

[[39 1 0]

[2 49 0]

[0 16 18]]

Euclidean confusion matrix

[[34 1 5]

[4 29 19]

[3 13 17]]