Get data

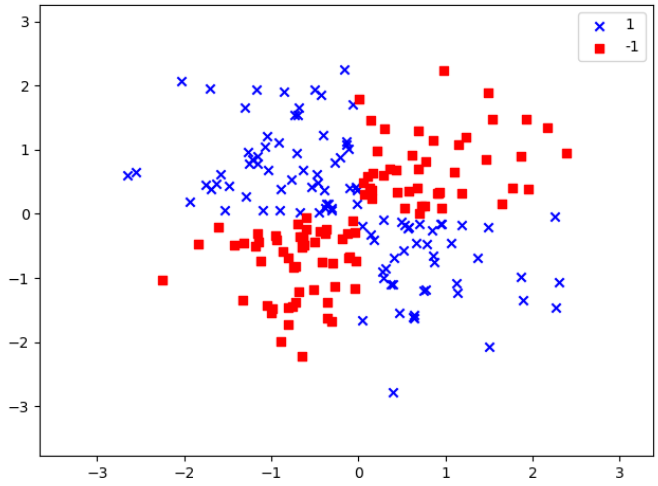
Here we are generating some non-linearly separable data that we will train our classifier on. This data would be akin to your training dataset. There are two classes in our y vector: blue x's and red squares.

```
def generate_data():
    np.random.seed(0)
    X_xor = np.random.randn(200, 2)
    y_xor = np.logical_xor(X_xor[:, 0] > 0,
                           X_xor[:, 1] > 0)
    y_xor = np.where(y_xor, 1, -1)
    return X_xor, y_xor
```

show original data

```
def show_original_data(X_xor, y_xor):
    plt.scatter(X_xor[y_xor == 1, 0],
                X_xor[y_xor == 1, 1],
                c='b', marker='x',
                label='1')
    plt.scatter(X_xor[y_xor == -1, 0],
                X_xor[y_xor == -1, 1],
                c='r',
                marker='s',
                label='-1')

    plt.xlim(X_xor[:, 0].min() - 1, X_xor[:, 0].max() + 1)
    plt.ylim(X_xor[:, 1].min() - 1, X_xor[:, 1].max() + 1)
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()
```



group data with labels

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

get mean on each group:

```
def get_mean(dict_part,part):

    m = np.mean(dict_part, axis=0)
    return {part:m}
get 8 center of each group that grouped before with data set label using kmeans with
8 cluster
def get_means(dict,part):
    kmeans = KMeans(n_clusters=8)
    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
```

Radial basis function kernel

The Radial basis function kernel, also called the RBF kernel, or Gaussian kernel, is a kernel that is in the form of a radial basis function (more specifically, a Gaussian function). The RBF kernel is defined as

$$K_{\mathrm{RBF}}(\mathbf{x}, \mathbf{x}') = \exp\left[-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2\right]$$

where γ is a parameter that sets the “spread” of the kernel.

```
def get_rbf_kernel(X, gamma):
    print("gamma :", gamma)
    new_X = []

    for i, data in enumerate(X):
        new_data = []
        for j, center in enumerate(centers):
            d = np.exp(-sum(pow((np.subtract(data, list(center.values())[0])), 2)) /
gamma)
            new_data.append(d)
        new_X.append(np.array(new_data))
    new_X = np.array(new_X)
    return new_X
```

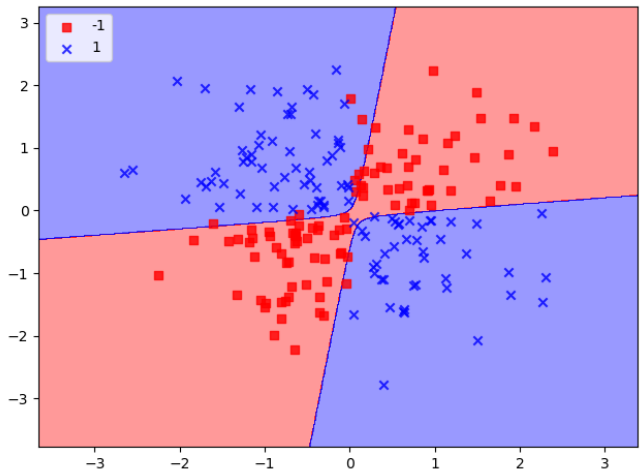
classify original data using mahalanobis

Code:

```
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=0.2)
    if x_test is not None:
        new_X_test = x_test
    dict = get_label_dict(new_X_train, new_y_train)
    c1 = get_means(dict[-1], -1)
    c2 = get_means(dict[1], 1)
    c1.extend(c2)
    means = c1
    inv_cov1 = get_covariance_inverse(dict[-1], -1)
    inv_cov2 = get_covariance_inverse(dict[1], 1)
    inv_cov = {**inv_cov1, **inv_cov2}

    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
```

Result:



Confusion Matrix :

[[191 10]

[2 197]]

Accuracy Score : 0.97

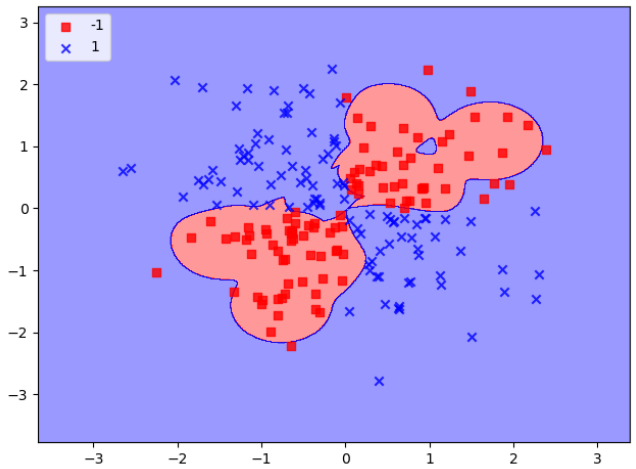
```
Classify data using Euclidean distance:
def cluster_on_euclidean_distance(x, means: List[dict] = None):
    euclid = []
```

```
    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]


def 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=0.2)
    if x_test is not None:
        new_X_test = x_test
    dict = get_label_dict(new_X_train, new_y_train)
    c1 = get_means(dict[-1], -1)
    c2 = get_means(dict[1], 1)
    c1.extend(c2)
    means = c1
    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
```

Classify data in rbf space:

Gama =0.1



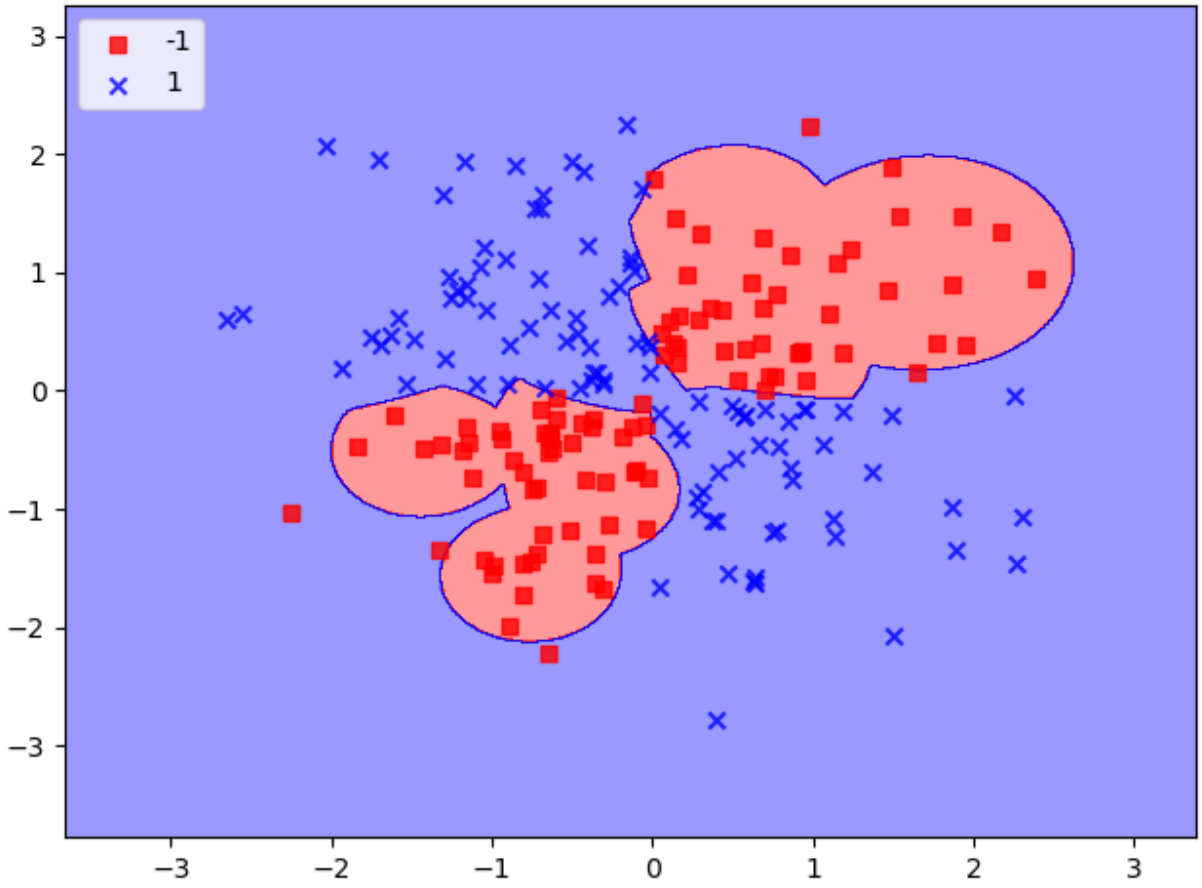
Confusion Matrix :

[[17 5]

[1 17]]

Accuracy Score : 0.85

Gamma=0.3



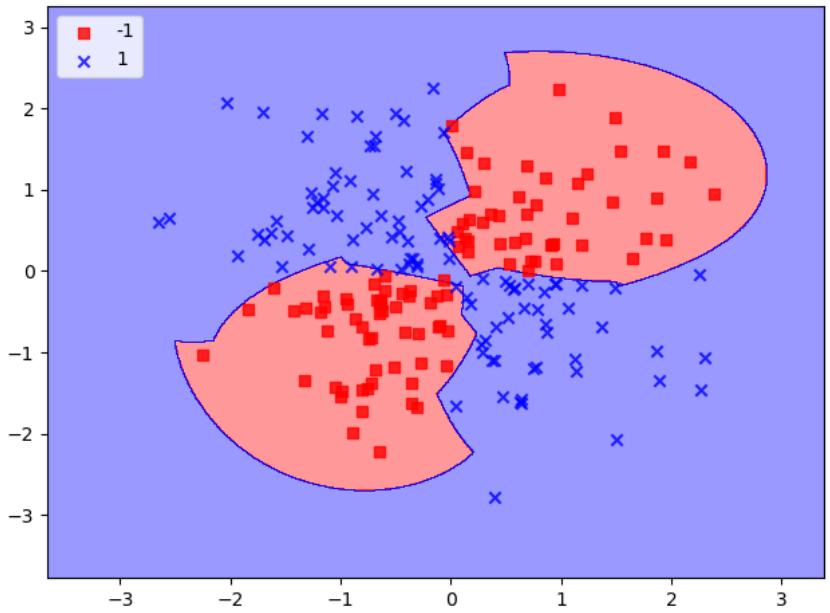
Confusion Matrix :

[[17 5]

[2 16]]

Accuracy Score : 0.825

Gamma =1



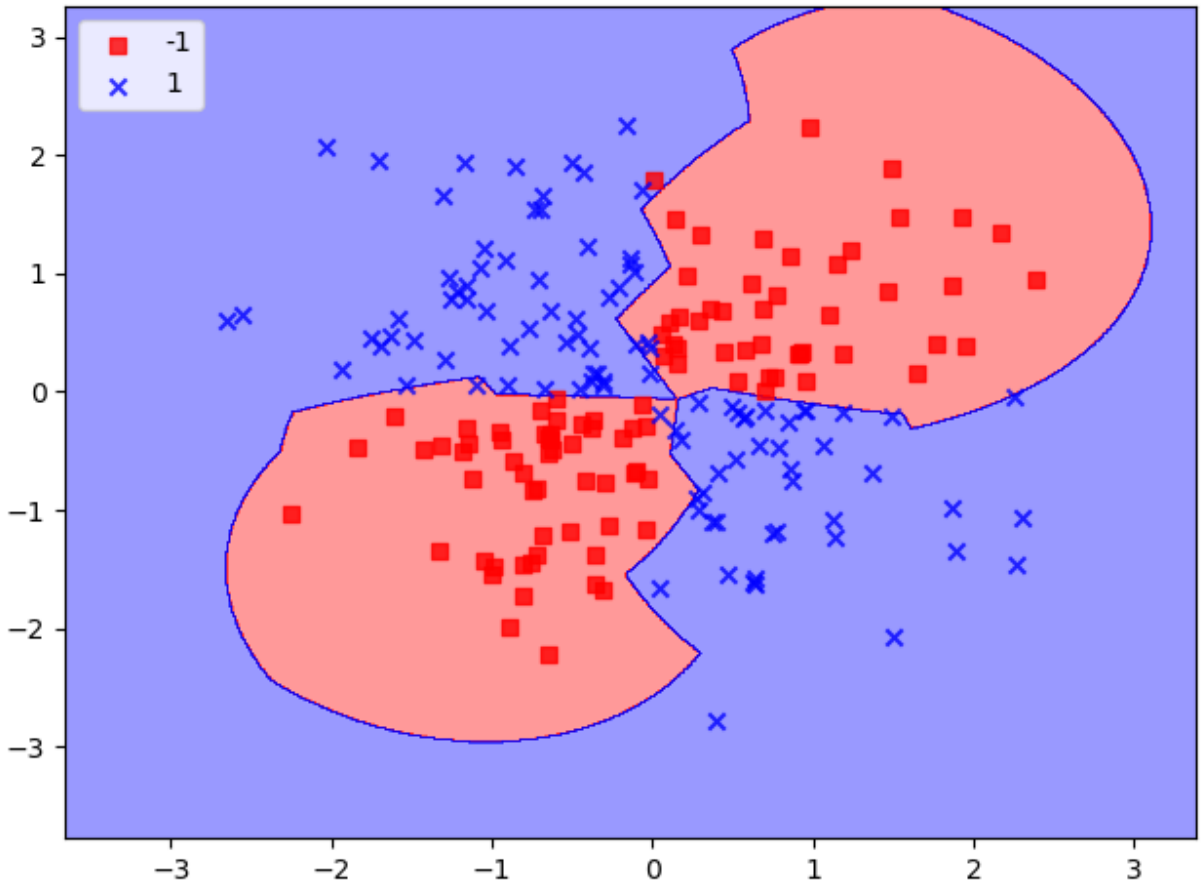
Confusion Matrix :

[[18 4]

 [4 14]]

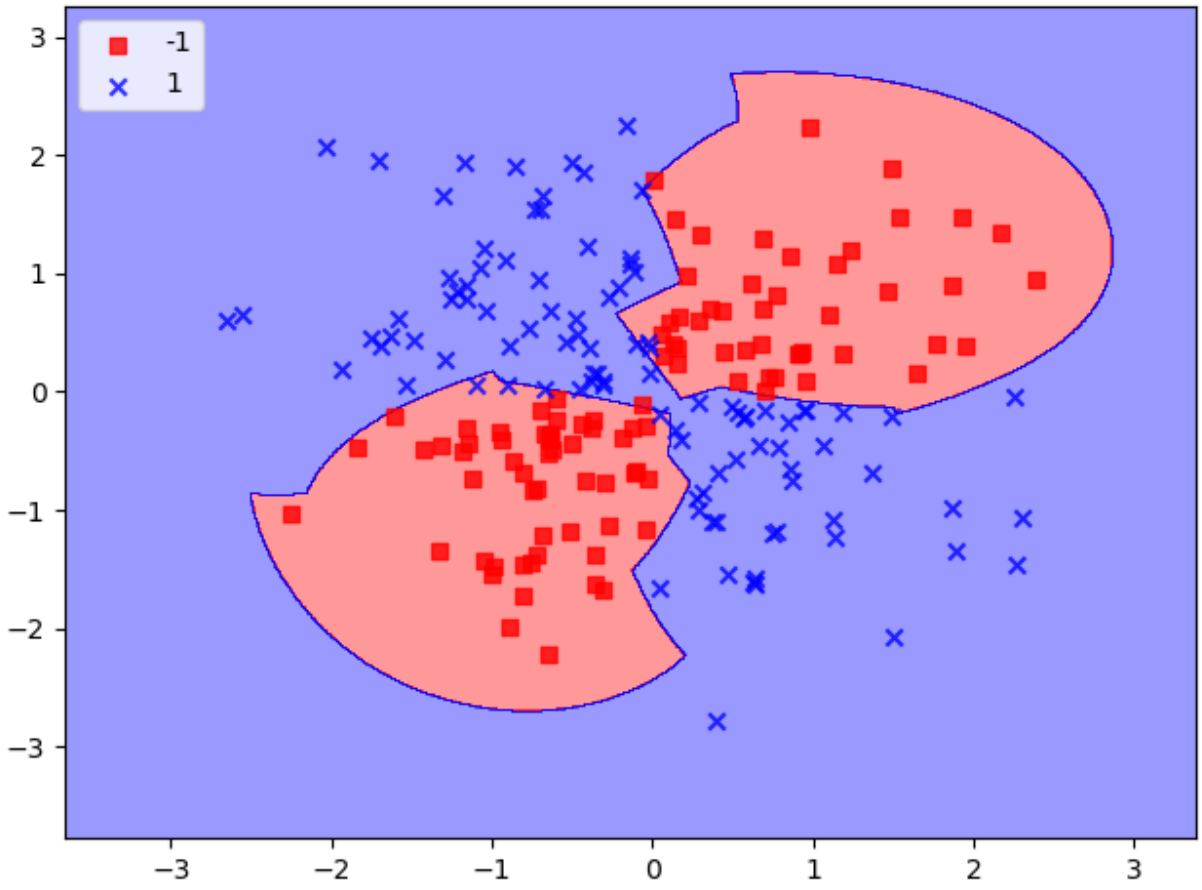
Accuracy Score : 0.8

Gama=5

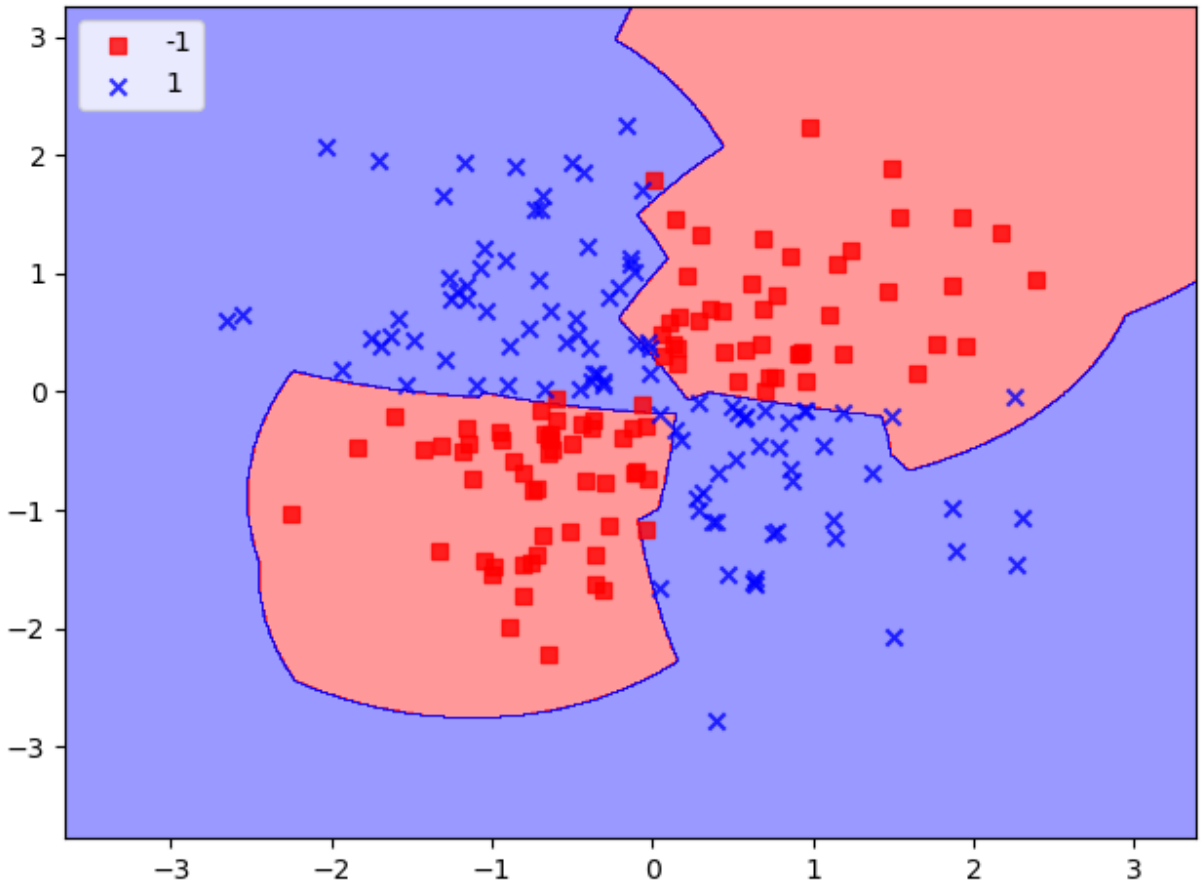



```
Confusion Matrix :
[[22  0]
 [ 3 15]]
Accuracy Score : 0.925
Gamma=2
Confusion Matrix :
[[20  2]
 [ 4 14]]
```

```
Accuracy Score : 0.85
```




```
Gamma=10
```




```
Confusion Matrix :
[[22  0]
 [ 3 15]]
Accuracy Score : 0.925
```

Gamma

Data=2000 , n_cluster=2

gamma	accuracy	
_	0.9825	original
0.01	0.785	small
0.1	0.905	small
5	0.985	=
10	0.985	=
100	0.9575	big
1000	0.9575	big

The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The gamma

parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting.

When gamma is very small, the model is too constrained and cannot capture the complexity or “shape” of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes

References:

https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_using_rbf_kernel/

https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html