Machine Learning

| Polynomial Kernel

Prof. Dr. Mostafa Elhosseini

https://youtube.com/ElhosseiniAcademy

Agenda

- Polynomial kernel
- Dot Product
- Polynomial kernel – derivation
- Coding

Intro

- The two kernels and their corresponding features maps we see in this chapter are:
  - **Polynomial kernel** and
  - Radial basis function (RBF) kernel.
- Both of them consist of adding columns to our dataset in different, yet very effective, ways.

Polynomial kernel

- A polynomial kernel is a type of kernel function used in machine learning algorithms, particularly in Support Vector Machines (SVMs), to compute the similarity between two vectors in a feature space.

- The polynomial kernel is defined as: $\boldsymbol{K(x, y) = (x.y + c)^d}$

- $x.y$ → **dot product** of two vectors x and y, which measures their similarity in the original feature space

Dot Product

- The dot product gives a measure of how similar two vectors are
- A higher dot product value indicates that the vectors are more aligned
- Similarity measures are used to project data points into higher-dimensional spaces where they become more easily separable

Polynomial kernel

- The polynomial kernel is defined as: $\boldsymbol{K(x, y) = (x.y + c)^d}$
- A higher degree d makes the model more flexible, allowing it to capture more complex relationships.
- A lower degree results in a simpler model that may be less prone to overfitting
- Parameter c :
  - Controls the influence of higher-order versus lower-order terms in the polynomial.
  - A larger c value reduces the influence of higher-order polynomial terms, making the decision boundary smoother.

Polynomial kernel

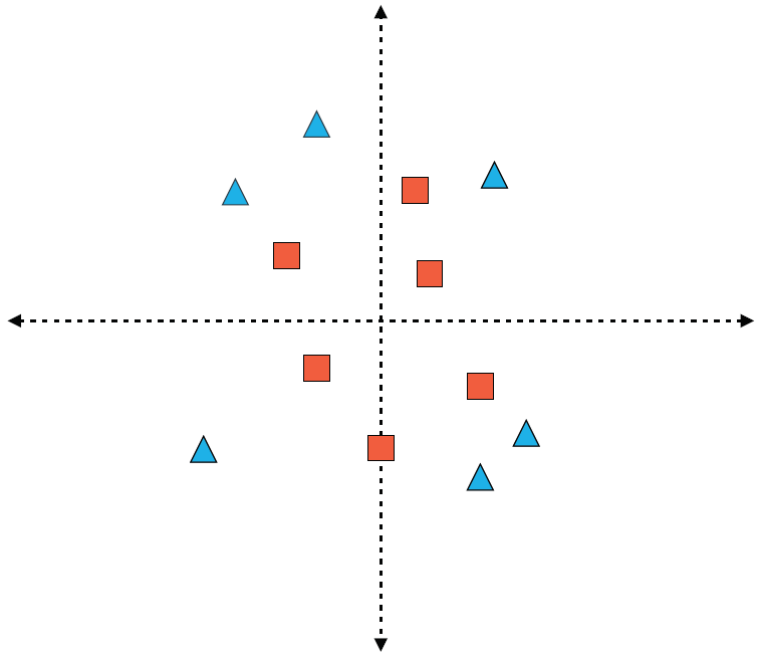
- The kernel method helps us model data using polynomial equations such as circles, parabolas, and hyperbolas.
- Example: A circular dataset

x_1	x_2	y
0.3	0.3	0
0.2	0.8	0
–0.6	0.4	0
0.6	–0.4	0
–0.4	–0.3	0
0	–0.8	0
–0.4	1.2	1
0.9	–0.7	1
–1.1	–0.8	1
0.7	0.9	1
–0.9	0.8	1
0.6	–1	1

Polynomial kernel

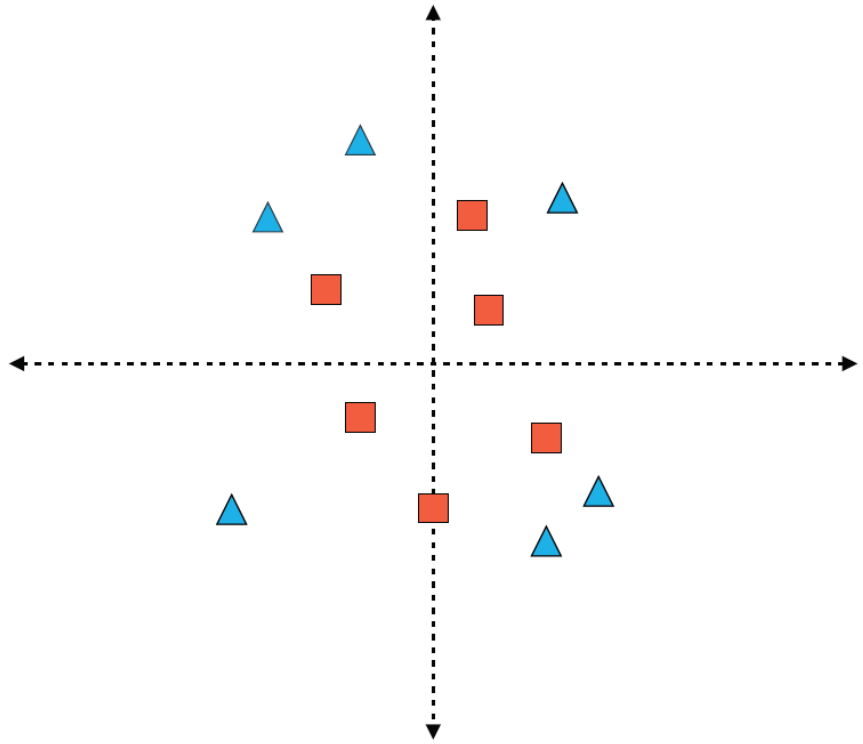
- The points with label 0 are drawn as squares and those with label 1 are drawn as triangles

x_1	x_2	y
0.3	0.3	0
0.2	0.8	0
–0.6	0.4	0
0.6	–0.4	0
–0.4	–0.3	0
0	–0.8	0
–0.4	1.2	1
0.9	–0.7	1
–1.1	–0.8	1
0.7	0.9	1
–0.9	0.8	1
0.6	–1	1



Polynomial kernel

- It is clear that a line won't be able to separate the squares from the triangles. However, a circle would — how can we draw this circle?
- What is a characteristic that separates the squares from the triangles?
- From observing the plot, it seems that the triangles are farther from the origin than the circles.



Polynomial kernel

- The formula that measures the distance to the origin is the square root of the sum of the squares of the two coordinates.

- If these coordinates are x_1, x_2, then this distance is $\sqrt{{x_1}^2 + {x_2}^2}$. Let's forget about the square root, and think only of $\boldsymbol{{x_1}^2 + {x_2}^2}$

- Now let's add a column to table

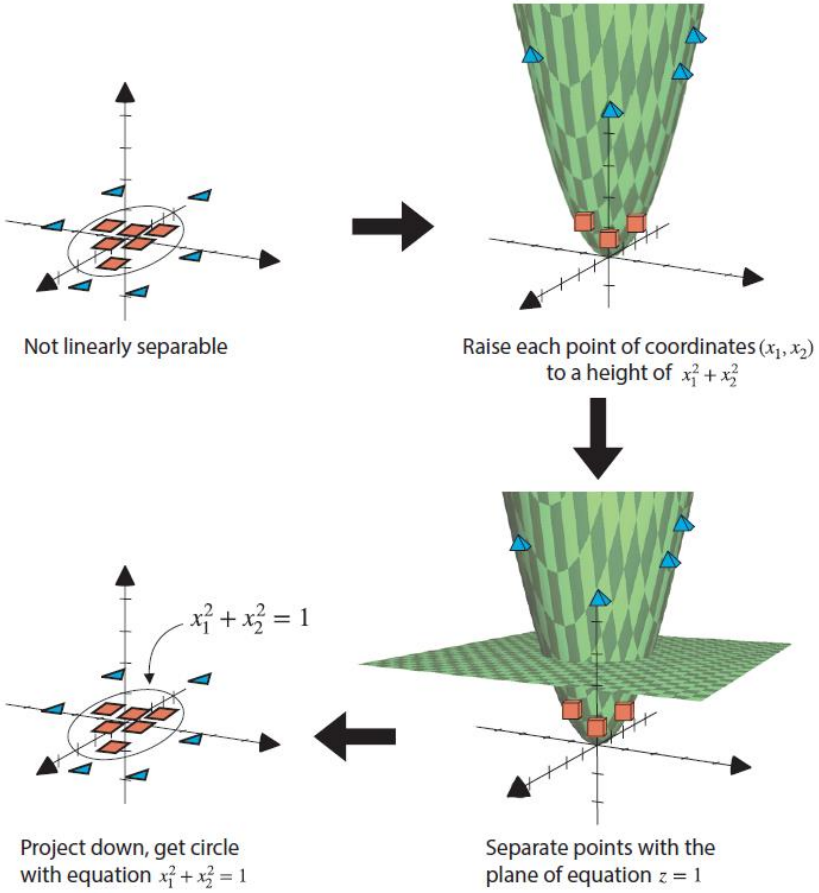
Polynomial kernel

- We can see the trend. All the points labeled 0 satisfy that the sum of the squares of the coordinates is less than 1, and the points labeled 1 satisfy that this sum is greater than 1. Therefore, the equation on the coordinates that separates the points is precisely $\boldsymbol{x_1^2 + x_2^2 = 1}$

x_1	x_2	$x_1^2 + x_2^2$	y
0.3	0.3	0.18	0
0.2	0.8	0.68	0
–0.6	0.4	0.52	0
0.6	–0.4	0.52	0
–0.4	–0.3	0.25	0
0	–0.8	0.64	0
–0.4	1.2	1.6	1
0.9	–0.7	1.3	1
–1.1	–0.8	1.85	1
0.7	0.9	1.3	1
–0.9	0.8	1.45	1
0.6	–1	1.36	1

Polynomial kernel

- Note that this is not a linear equation
- The geometric way to imagine this is depicted in figure below. Our original set lives in the plane, and it is impossible to separate the two classes with a line. But if we raise each point $(\boldsymbol{x_1}, \boldsymbol{x_2})$to the height $\boldsymbol{x_1}^{\boldsymbol{2}} + \boldsymbol{x_2}^{\boldsymbol{2}}$, this is the same as putting the points in the paraboloid with equation $z = x_1^{\,2} + x_2^{\,2}$.



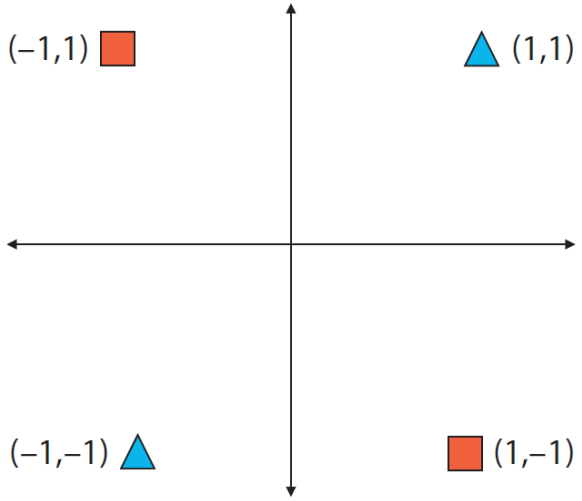
Polynomial kernel

- The distance we raised each point is precisely the square of the distance from that point to the origin. Therefore, the squares are raised a small amount, because they are close to the origin, and the triangles are raised a large amount, because they are far away from the origin.
- Now the squares and triangles are far away from each other, and therefore, we can separate them with the horizontal plane at height 1—in other words, the plane with equation $z = 1$. As a final step, we project everything down to the plane

Polynomial kernel

Example: The modified XOR dataset

x_1	x_2	y
–1	–1	1
–1	1	0
1	–1	0
1	1	1



- It is impossible to draw a line that separates the triangles from the squares. However, we can use a polynomial equation to help us, and this time we'll use the product of the two features.

Polynomial kernel

Example: The modified XOR dataset

- Let's add the column corresponding to the product $x_1 x_2$ to the original dataset

x_1	x_2	$x_1 x_2$	y
−1	−1	1	1
−1	1	−1	0
1	−1	−1	0
1	1	1	1

- A good classifier for this data is the one with the following boundary equation: $\boldsymbol{x_1 x_2 = 1}$.

Polynomial kernel

- In both of the previous examples, we used a polynomial expression to help us classify a dataset that was not linearly separable
  - $x_1{}^2 + x_2{}^2$
  - $x_1 x_2 = 1$
- How did we find these expressions?
- we may not have the luxury to look at a plot and eyeball an expression that will help us out. We need a method or, in other words, an algorithm. What we'll do is consider all the possible **monomials of degree 2** (quadratic), containing x_1 and x_2

Polynomial kernel

- These are the following three monomials: $\boldsymbol{x_1}^2, \boldsymbol{x_1 x_2}, \boldsymbol{x_2}^2$
- We call these new variables $\boldsymbol{x_3}, \boldsymbol{x_4}, \textbf{and } \boldsymbol{x_5}$,and we treat them as if they had no relation with x_1 and x_2 whatsoever.
- Let's apply this to the first example (the circle).

x_1	x_2	$x_3 = x_1^2$	$x_4 = x_1 x_2$	$x_5 = x_2^2$	y
0.3	0.3	0.09	0.09	0.09	0
0.2	0.8	0.04	0.16	0.64	0
–0.6	0.4	0.36	–0.24	0.16	0
0.6	–0.4	0.36	–0.24	0.16	0
–0.4	–0.3	0.16	0.12	0.09	0
0	–0.8	0	0	0.64	0
–0.4	1.2	0.16	–0.48	1.44	1
0.9	–0.7	0.81	–0.63	0.49	1
–1.1	–0.8	1.21	0.88	0.64	1
0.7	0.9	0.49	0.63	0.81	1
–0.9	0.8	0.81	–0.72	0.64	1
0.6	–1	0.36	–0.6	1	1

Polynomial kernel

- We can now build an SVM that classifies this enhanced dataset. The way to train an SVM is using the methods learned in the last section
- By inspection, here is one equation of a classifier that works:
- $\mathbf{0}x_1 + \mathbf{0}x_2 + \mathbf{1}x_3 + \mathbf{0}x_4 + \mathbf{1}x_5 - \mathbf{1} = 0$

x_1	x_2	$x_3 = x_1^2$	$x_4 = x_1 x_2$	$x_5 = x_2^2$	y
0.3	0.3	0.09	0.09	0.09	0
0.2	0.8	0.04	0.16	0.64	0
–0.6	0.4	0.36	–0.24	0.16	0
0.6	–0.4	0.36	–0.24	0.16	0
–0.4	–0.3	0.16	0.12	0.09	0
0	–0.8	0	0	0.64	0
–0.4	1.2	0.16	–0.48	1.44	1
0.9	–0.7	0.81	–0.63	0.49	1
–1.1	–0.8	1.21	0.88	0.64	1
0.7	0.9	0.49	0.63	0.81	1
–0.9	0.8	0.81	–0.72	0.64	1
0.6	–1	0.36	–0.6	1	1

Polynomial kernel

- Remembering that $\boldsymbol{x_3 = x_1^2}$ and $\boldsymbol{x_5 = x_2^2}$ we get the desired equation of the circle, as shown next: $x_1^2 + x_2^2 = 1$
- Our nice two-dimensional dataset became a five-dimensional dataset

x_1	x_2	$x_3 = x_1^2$	$x_4 = x_1 x_2$	$x_5 = x_2^2$	y
0.3	0.3	0.09	0.09	0.09	0
0.2	0.8	0.04	0.16	0.64	0
–0.6	0.4	0.36	–0.24	0.16	0
0.6	–0.4	0.36	–0.24	0.16	0
–0.4	–0.3	0.16	0.12	0.09	0
0	–0.8	0	0	0.64	0
–0.4	1.2	0.16	–0.48	1.44	1
0.9	–0.7	0.81	–0.63	0.49	1
–1.1	–0.8	1.21	0.88	0.64	1
0.7	0.9	0.49	0.63	0.81	1
–0.9	0.8	0.81	–0.72	0.64	1
0.6	–1	0.36	–0.6	1	1

Polynomial kernel

- The polynomial kernel gives rise to the map that sends the 2-D plane to the 5-D space
- The map is the one that sends the point (x_1, x_2) to the point $(x_1, x_2, {x_1}^2, x_1 x_1, {x_2}^2)$.
- Because the maximum degree of each monomial is 2, we say that this is the polynomial kernel of degree 2. For the polynomial kernel, we always have to specify the degree.

x_1	x_2	$x_3 = x_1^2$	$x_4 = x_1 x_2$	$x_5 = x_2^2$	y
0.3	0.3	0.09	0.09	0.09	0
0.2	0.8	0.04	0.16	0.64	0
–0.6	0.4	0.36	–0.24	0.16	0
0.6	–0.4	0.36	–0.24	0.16	0
–0.4	–0.3	0.16	0.12	0.09	0
0	–0.8	0	0	0.64	0
–0.4	1.2	0.16	–0.48	1.44	1
0.9	–0.7	0.81	–0.63	0.49	1
–1.1	–0.8	1.21	0.88	0.64	1
0.7	0.9	0.49	0.63	0.81	1
–0.9	0.8	0.81	–0.72	0.64	1
0.6	–1	0.36	–0.6	1	1

Polynomial kernel - derivation

- Calculates high-dimensional relationships between pairs of observations

- $Polynomial\ Kernel = (a \times b + r)^d$

  - a, b: two different observations of the dataset
  - r: coefficient of the polynomial
  - d: degree of the polynomial

Polynomial kernel

- $Polynomial\ Kernel = (a \times b + r)^d$
- d: degree of the polynomial
- **When $\boldsymbol{d = 1}$**, the polynomial kernel computes the relationships between each pair of observations in 1-dimension and these relationships are used to find a support vector classifier
- **When $\boldsymbol{d = 2}$**, The polynomial kernel computes the 2-dimensional relationships between each pair of observation, and these relationships are used to find a support vector classifier
- r, d: determined using cross validation

Polynomial kernel

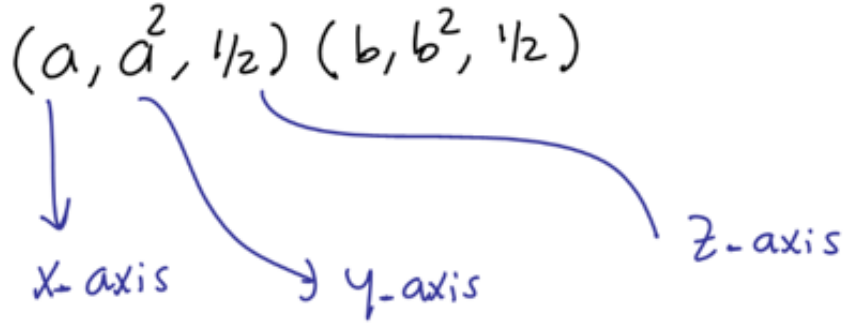
- $Polynomial\ Kernel = (ab + r)^d$

  Let $r = \frac{1}{2}$, $d = 2$

  $(ab + \frac{1}{2})^2 = a^2b^2 + ab + \frac{1}{4}$

  $(ab + \frac{1}{2})^2 = ab + a^2b^2 + \frac{1}{4}$

  $(ab + \frac{1}{2})^2 = \left(a, a^2, \frac{1}{2}\right).\left(b, b^2, \frac{1}{2}\right)$ → dot product



Note

Z-axis is the same for the two points, So we cancel it

Dot Product gives us the high-dimensional coordinates for the data

Polynomial kernel

- $Polynomial\ Kernel = (ab + r)^d$

  Let $r = 1,\ d = 2$

  $(ab + 1)^2 = a^2b^2 + 2ab + 1$

  $(ab + 1)^2 = 2ab + a^2b^2 + 1$

  $(ab + 1)^2 = (\sqrt{2}a, a^2, 1).(\sqrt{2}b, b^2, 1)$ → dot product

$(ab+1)^2 = 2ab + a^2b^2 + 1$

$= (\sqrt{2}a, a^2, 1) \cdot (\sqrt{2}b, b^2, 1)$

new x-axis coordinates are the square root of 2 times the original dosage values

So, we move the points on the x-axis over by a factor of the square root of 2

new y-axis are the same as before — the dosage values squared

Polynomial kernel

- Kernel functions only calculate the relationships between every pair of points as if they are in the higher dimensions → they do not actually do the transformation
- The kernel Trick reduces the amount of computation required for SVM by avoiding the math that transforms the data from low to high dimensions
- So, all we need to do to calculate the high dimensional relationships is to calculate the Dot Product between each pair of point

Polynomial kernel

- Just plug values into the kernel to get the high-dimensional relationships
- Example
- $a = (2,3)$ $b = (4,5)$ $d = 2$ $c = 1$
- $a \,.\, b = (2,3) \,.\, (4,5) = 23$
- $K(a, b) = (a \,.\, b + c)^2 = (23 + 1)^2 = 576$
- This value represents the similarity between the two vectors in the higher-dimensional space implicitly defined by the polynomial kernel.

Polynomial kernel

- if the kernel value between two points is high, it implies that these points will have a stronger influence on the position of the decision boundary. If both points belong to the same class, the SVM will likely position the decision boundary in such a way that these similar points are correctly classified.

Polynomial kernel

- If the kernel values are consistently high, it might indicate that the model is capturing more details of the training data, which could lead to a complex decision boundary. While this can improve accuracy on the training data, it can also lead to overfitting, where the model performs poorly on unseen data.

- Suppose you increase the degree d or the constant c in the polynomial kernel. This will likely increase the kernel values, leading to a decision boundary that might be very curvy and closely fit to the training data points.

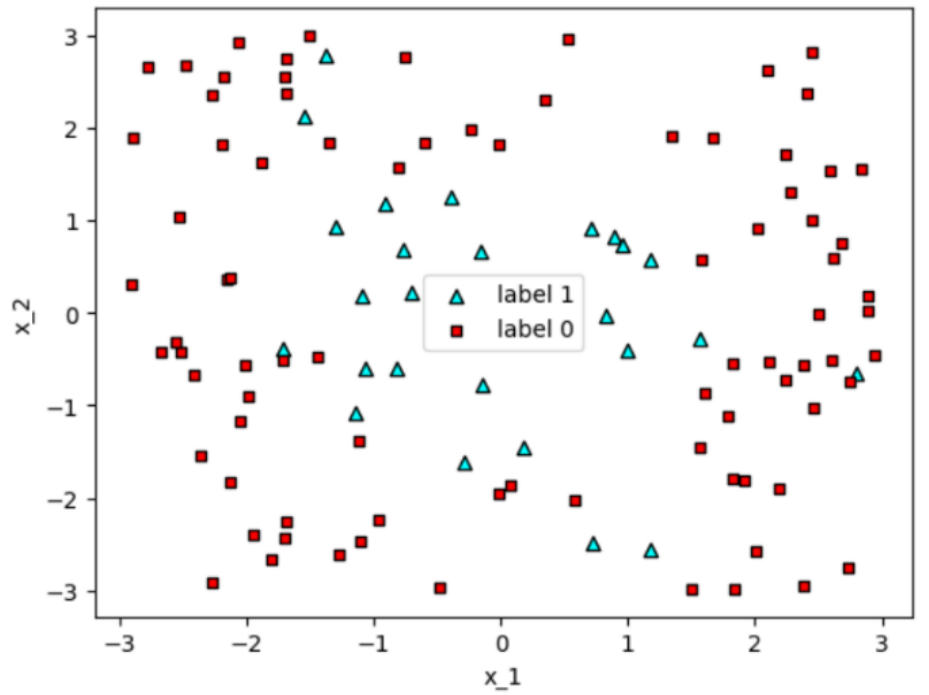
Coding

- Example: Circular dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import utils
from sklearn.svm import SVC
```

```
# Loading the one_circle dataset

circular_data = pd.read_csv('one_circle.csv')
features = np.array(circular_data[['x_1', 'x_2']])
labels = np.array(circular_data['y'])
utils.plot_points(features, labels)
```

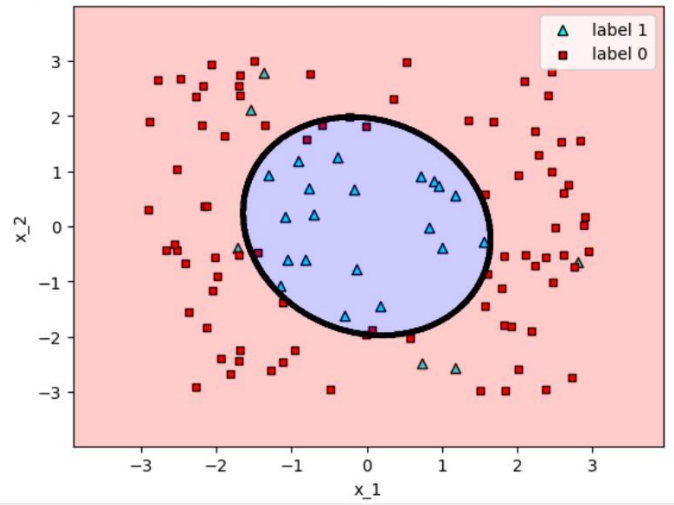


Coding

- Example: Circular dataset

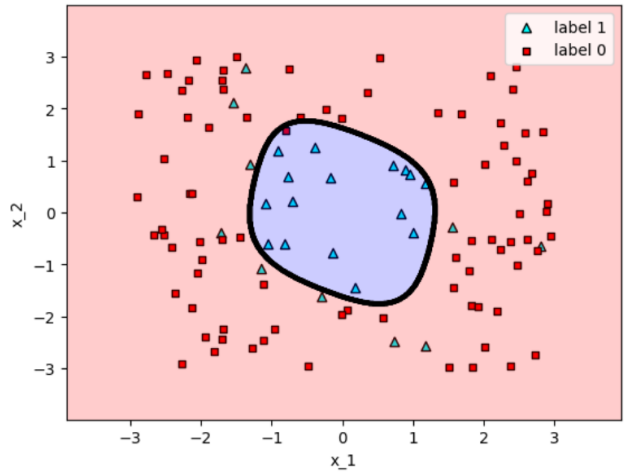
```
# Degree = 2
svm_degree_2 = SVC(kernel='poly', degree=2)
svm_degree_2.fit(features, labels)
print("Polynomial kernel of degree = 2")
print("Accuracy:", svm_degree_2.score(features, labels))
utils.plot_model(features, labels, svm_degree_2)
```

```
Polynomial kernel of degree = 2
Accuracy: 0.8909090909090909
```




```
# Degree = 4
svm_degree_4 = SVC(kernel='poly', degree=4)
svm_degree_4.fit(features, labels)
print("Polynomial kernel of degree = 4")
print("Accuracy:", svm_degree_4.score(features, labels))
utils.plot_model(features, labels, svm_degree_4)
```

```
Polynomial kernel of degree = 4
Accuracy: 0.9
```

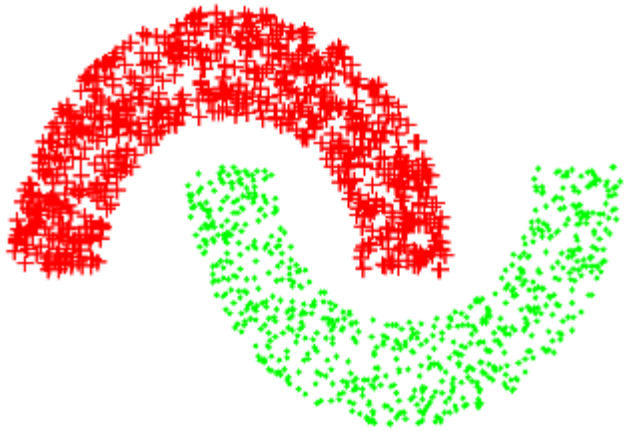


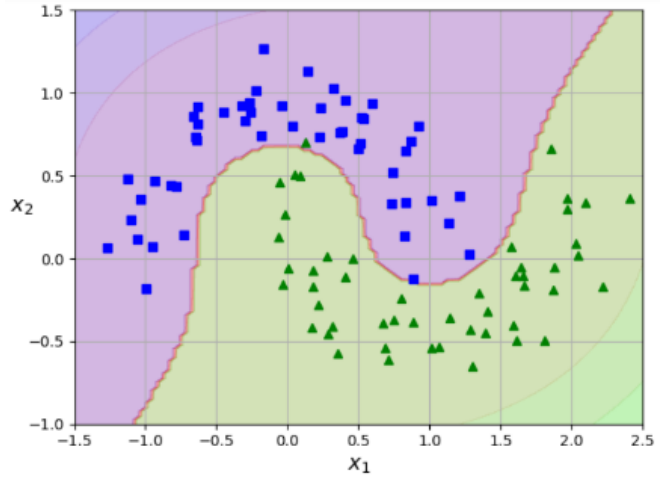
Coding

- Let’s test this on the **moons dataset**, a toy dataset for binary classification in which the data points are shaped as two interleaving crescent moons
- Toy datasets are small, simplified datasets that are often used for educational purposes, quick experimentation, and testing algorithms in machine learning. These datasets are typically easy to understand and visualize, making them ideal for demonstrating the behavior of algorithms without the complexity and noise found in real-world data.

Coding

- Common Examples of Toy Datasets:
  - Iris Dataset
  - **Moons Dataset**
  - Blobs Dataset
  - Circles Dataset
  - Digits Dataset
  - Swiss Roll Dataset



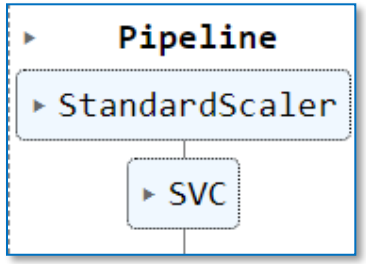


Coding

- Example: Moons Dataset

```
from sklearn.svm import SVC

poly_kernel_svm_clf = make_pipeline(StandardScaler(),
                                    SVC(kernel="poly", degree=3, coef0=1, C=5))
poly_kernel_svm_clf.fit(X, y)
```



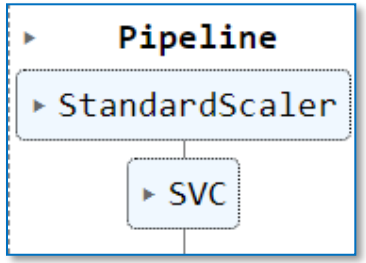
- The hyperparameter $coef0$ controls how much the model is influenced by high-degree terms versus low-degree terms.

Coding

- Example: Moons Dataset

```
from sklearn.svm import SVC

poly_kernel_svm_clf = make_pipeline(StandardScaler(),
                                    SVC(kernel="poly", degree=3, coef0=1, C=5))
poly_kernel_svm_clf.fit(X, y)
```



- Low coef0: When coef0 is small or close to zero, the kernel function emphasizes higher-degree polynomial terms. This can result in a more complex decision boundary, potentially capturing more intricate patterns in the data. However, this might also lead to overfitting, especially if the degree of the polynomial is high.

Radial Basis Function (RBF) kernel

- Using bumps in higher dimensions to our benefit
- build nonlinear boundaries using certain special functions centered at each of the data points.
- To add features computed using a similarity function, which measures how much each instance resembles a particular landmark
- Next Lecture