Assignment 3A

Botond Lovász

December 19, 2024

1 Principal Component Analysis

1.1 Question 3.1.1

The data-centering step is important for the following reasons:

- It translates the dataset so that its mean is at the origin of the coordinate space. This is important because PCA seeks to find the directions (principal components) that maximize the variance of the data. If the data is not centered, the first principal component might simply point towards the mean of the data rather than capturing the true direction of maximum variance.
- If PCA is performed on non-centered data, the principal components may not accurately represent the directions of maximum variance. This can lead to misleading results, where the principal components are influenced by the mean of the data rather than the true structure of the data. Consequently, the dimensionality reduction may not effectively capture the underlying patterns, leading to poor performance in subsequent analyses or models.
- When the data is centered, $\bar{X} = 0$ and $\bar{Y} = 0$. This simplifies the (estimated) covariance calculation to:

$$\mathrm{Cov}(X, Y) = \frac{1}{n-1}\sum_{i=1}^{n} X_i Y_i$$

1.2 Question 3.1.2

SVD decomposes a matrix A of size $m \times n$ into three matrices: U, Σ, and V^T, such that $A = U\Sigma V^T$. The principal components are derived from the eigenvectors of the covariance matrix. For a data matrix A, the covariance matrix for PCA on the rows is $A^T A$, and for the columns, it is AA^T.

The SVD of the transpose A^T is $V\Sigma^T U^T$. This implies that the roles of U and V are swapped, and the singular values remain the same (though the matrix Σ is transposed).

When performing PCA on the rows of A, we are interested in the eigenvectors of A^TA. These are given by the columns of V from the SVD of A. Conversely, when performing PCA on the columns of A, we are interested in the eigenvectors of AA^T. These are given by the columns of U from the SVD of A.

So, yes, a single SVD operation is sufficient to perform PCA on both the rows and columns of a data matrix.

1.3 Question 3.1.3

Given the dataset $Y = \{y_1, \dots, y_n\}$ with zero-centered data, the variance is defined as:

$$\mathrm{Var}(Y) = \sum_{y \in Y} \|y - \bar{y}\|_2^2 = \sum_{y \in Y} \|y\|_2^2$$

Since the data is zero-centered, $\bar{y} = 0$.

The dataset Y can be represented as a $d \times n$ matrix Y.

The SVD of Y is given by $Y = U\Sigma V^T$, where:

- U is a $d \times d$ orthogonal matrix.
- Σ is a $d \times n$ diagonal matrix with singular values $\sigma_1, \sigma_2, \dots, \sigma_d$.
- V is an $n \times n$ orthogonal matrix.

The variance of Y can be expressed as the sum of the squared norms of its columns:

$$\mathrm{Var}(Y) = \sum_{i=1}^{n} \|y_i\|_2^2 = \mathrm{trace}(Y^TY)$$

Using the SVD, $Y^TY = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^TU^TU\Sigma V^T = V\Sigma^T\Sigma V^T$. Since U is orthogonal, $U^TU = I$. The trace of a matrix is invariant under cyclic permutations, so:

$$\mathrm{trace}(Y^TY) = \mathrm{trace}(\Sigma^T\Sigma) = \sum_{i=1}^{d} \sigma_i^2$$

So, the variance of the dataset Y can be expressed as the sum of the squares of the singular values of Y:

$$\mathrm{Var}(Y) = \sum_{i=1}^{d} \sigma_i^2$$

1.4 Question 3.1.4

The dataset Y is represented as a $d \times n$ matrix. The SVD of Y is $Y = U\Sigma V^T$, where:

- U is a $d \times d$ orthogonal matrix.
- Σ is a $d \times n$ diagonal matrix with singular values $\sigma_1, \sigma_2, \ldots, \sigma_d$.
- V is an $n \times n$ orthogonal matrix.

The first k principal components are the first k columns of U. The matrix W is formed by these k principal components, i.e., $W = U_k$, where U_k is the $d \times k$ matrix consisting of the first k columns of U.

The projected data X is given by $X = W^T Y = U_k^T Y$. Substituting the SVD of Y, we have:

$$X = U_k^T (U\Sigma V^T) = U_k^T U \Sigma V^T$$

Since U is orthogonal, $U_k^T U$ results in a $k \times d$ matrix with the first k rows of the identity matrix, effectively selecting the first k rows of Σ.

The variance of the projected data X is the trace of the covariance matrix of X, which is XX^T. Substituting for X, we have:

$$XX^T = (U_k^T U \Sigma V^T)(V \Sigma^T U^T U_k) = U_k^T U \Sigma \Sigma^T U^T U_k$$

Since U is orthogonal, $U^T U = I$, ergo:

$$XX^T = U_k^T \Sigma \Sigma^T U_k$$

The matrix $\Sigma\Sigma^T$ is a diagonal matrix with the squares of the singular values $\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2$ on the diagonal. The trace of $U_k^T \Sigma \Sigma^T U_k$ is the sum of the first k diagonal elements of $\Sigma\Sigma^T$, which are $\sigma_1^2, \sigma_2^2, \ldots, \sigma_k^2$.

The variance of the projected data X is indeed given by the sum of the squares of the first k singular values of Y:

$$\mathrm{Var}(X) = \sum_{i=1}^{k} \sigma_i^2$$

1.5 Question 3.1.5

The residual data points $Z = \{z_1, \ldots, z_n\}$ are defined as $z_i = y_i - WW^T y_i$. In matrix form, the residual data is $Z = Y - WW^T Y$.

The projection matrix WW^T projects Y onto the subspace spanned by the first k principal components. The residuals Z are the components of Y orthogonal to this subspace.

The variance of the residual data Z is the trace of the covariance matrix of Z, which is ZZ^T. Substituting for Z, we have:

$$Z = Y - WW^T Y = (I - WW^T)Y$$

Therefore, the covariance matrix of Z is:

$$ZZ^T = (I - WW^T)YY^T(I - WW^T)^T$$

Using the SVD of $Y = U\Sigma V^T$, we have:

$$YY^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$$

The matrix $\Sigma\Sigma^T$ is diagonal with $\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2$ on the diagonal.

The variance of the residual data Z is the sum of the squares of the singular values corresponding to the dimensions not captured by the first k principal components:

$$\text{Var}(Z) = \sum_{i=k+1}^{d} \sigma_i^2$$

So, the variance of the residual data Z is indeed given by the sum of the squares of the singular values from $k+1$ to d. Therefore, the variance of the original data is the sum of the variance explained by PCA and the variance of the residual data:

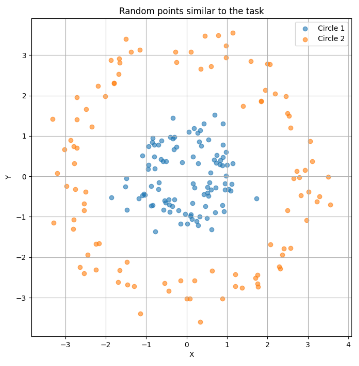
$$\text{Var}(Y) = \text{Var}(X) + \text{Var}(Z)$$

2 Classifying non-linearly-separable data with a linear classifier

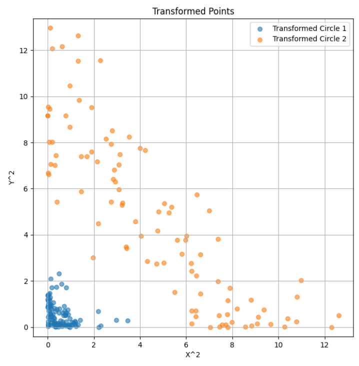
2.1 Question 3.2.6

To map the data into a two-dimensional space where they become linearly separable, we used a simple transformation by squaring the attributes. We chose this because we see from the given figure, that the data is centered around the same point and it is distributed in 2 different circles. We can easily separate 2 circles by their radious, but the task asked to map it into a 2D space, so we take the square of the attributes to get a 2D space and still be able to separate them based on the original radious. (We could have also used the absolute value of the attributes to get a 2D space, or map it into polar coordinates.)

- For each data point (x, y), compute new features by squaring the original attributes:
$$\phi_1(x, y) = x^2, \quad \phi_2(x, y) = y^2$$
- Each data point (x, y) in the original space is transformed to (x^2, y^2) in the new space.



(a) Points similar to the task



(b) Transformed points

Figure 1: Figures showing the points and their transformed counterparts

2.2 Question 3.2.7

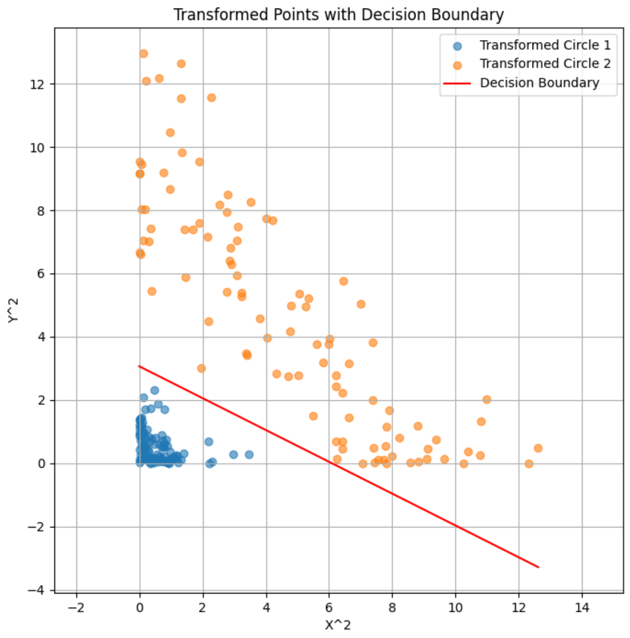


Figure 2: Transformed points with decision boundary

- Given a new data point (x, y) in the original space, apply the same transformation used during training:

$$\phi_1(x, y) = x^2, \quad \phi_2(x, y) = y^2$$

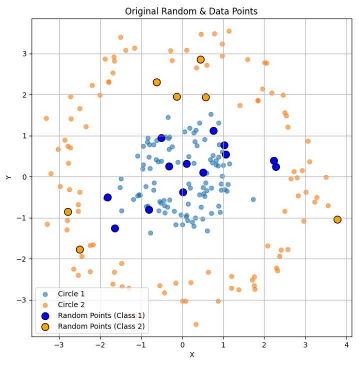
  This maps the point to the transformed space where the data is linearly separable.

- Use the trained linear classifier to predict the label of the transformed data point (ϕ_1, ϕ_2). The classifier uses a decision boundary, typically defined by a weight vector $\mathbf{w}$ and a bias b. The decision function is:

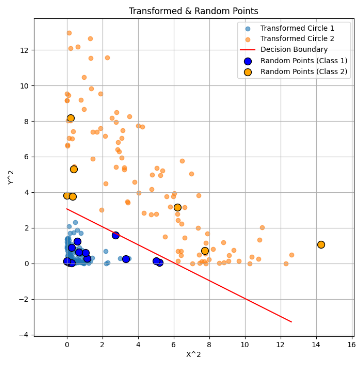
$$f(\phi_1, \phi_2) = \mathbf{w} \cdot \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} + b$$

  If $f(\phi_1, \phi_2) > 0$, classify the point as one class; otherwise, classify it as the other class.

- In our case, the decision boundary is a line, and the points are classified based on which side of the line they lie on. We can easily compute this by using the equation of the line and calculating the y' value for the input x^2 and seeing if it is greater than the transformed y^2 value.



(a) Random points



(b) Transformed Random points

Figure 3: Figures showing the random points and their transformed counterparts with classification

3 Spectral graph analysis

3.1 Question 3.3.8

The normalized Laplacian L is defined as:

$$L = I - \frac{1}{d}A$$

where I is the identity matrix.

$$x^T L x = x^T \left(I - \frac{1}{d}A \right) x = x^T I x - \frac{1}{d} x^T A x$$

Since A is the adjacency matrix is symmetric (of an undirected graph) and $a_{ij} = 1$ if $(i, j) \in E$ or $(j, i) \in E$ and 0 otherwise.

$$x^T A x = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} a_{ij} x_i x_j = 2 \sum_{(u,v)\in E} x_u x_v$$

$$x^T I x = \sum_{v=1}^{|V|} x_v^2$$

$$x^T L x = \sum_{v=1}^{|V|} x_v^2 - \frac{2}{d} \sum_{(u,v)\in E} x_u x_v$$

Summing over all edges $(u, v) \in E$:

$$\sum_{(u,v)\in E} (x_u - x_v)^2 = \sum_{(u,v)\in E} (x_u^2 + x_v^2 - 2x_u x_v)$$

Since G is d-regular, each vertex i appears in exactly d edges:

$$\sum_{(u,v)\in E} x_u^2 + \sum_{(u,v)\in E} x_v^2 = d \sum_{v=1}^{|V|} x_v^2$$

Therefore:

$$\sum_{(u,v)\in E} (x_u - x_v)^2 = d \sum_{v=1}^{|V|} x_v^2 - 2 \sum_{(u,v)\in E} x_u x_v$$

Divide by d to match the form:

$$\frac{1}{d} \sum_{(u,v)\in E} (x_u - x_v)^2 = \sum_{v=1}^{|V|} x_v^2 - \frac{2}{d} \sum_{(u,v)\in E} x_u x_v$$

This matches the expression for $x^T L x$, proving the statement.

3.2 Question 3.3.9

A symmetric matrix L is positive semidefinite if for all vectors $x \in \mathbb{R}^{|V|}$, the quadratic form $x^T Lx \geq 0$.

The normalized Laplacian $L = I - \frac{1}{d}A$ is symmetric because both I (the identity matrix) and A (the adjacency matrix of an undirected graph) are symmetric matrices.

From the previous derivation, we have shown that for any vector $x \in \mathbb{R}^{|V|}$:

$$x^T Lx = \frac{1}{d} \sum_{(u,v)\in E} (x_u - x_v)^2$$

The expression $(x_u - x_v)^2$ is always non-negative for any real numbers x_u and x_v. Therefore, the sum $\sum_{(u,v)\in E}(x_u - x_v)^2$ is non-negative. d is a positive constant as it is the degree of the graph.

Since $x^T Lx = \frac{1}{d}\sum_{(u,v)\in E}(x_u - x_v)^2 \geq 0$ for all vectors x, it follows that the normalized Laplacian L is positive semidefinite.

3.3 Question 3.3.10

A **non-trivial vector** x_* in the context of minimizing $x^T Lx$ refers to a vector that is not the trivial solution, which is typically the zero vector or any constant vector. A vector x_* is non-trivial if it is not a constant vector.

The vector x_* can be used to embed the vertices of the graph into the real line. This embedding assigns a real number to each vertex, effectively placing the vertices along a line.

The value x_{*i} represents the position of vertex i on the real line. This embedding can be meaningful if it reflects the graph's structure, with closely connected vertices having similar values.

Equation (2) states:

$$x^T Lx = \frac{1}{d} \sum_{(u,v)\in E} (x_u - x_v)^2$$

The expression $\sum_{(u,v)\in E}(x_u - x_v)^2$ measures the total squared difference between connected vertices. Minimizing $x^T Lx$ minimizes these differences, leading to an embedding where connected vertices are placed close together on the real line.

By minimizing $x^T Lx$, the embedding x_* captures the graph's connectivity structure. Vertices that are directly connected or part of the same cluster will have similar values, reflecting their proximity in the graph.

This embedding is useful for visualizing the graph's structure and identifying clusters or communities within the graph.

A Jupyter Notebook Output

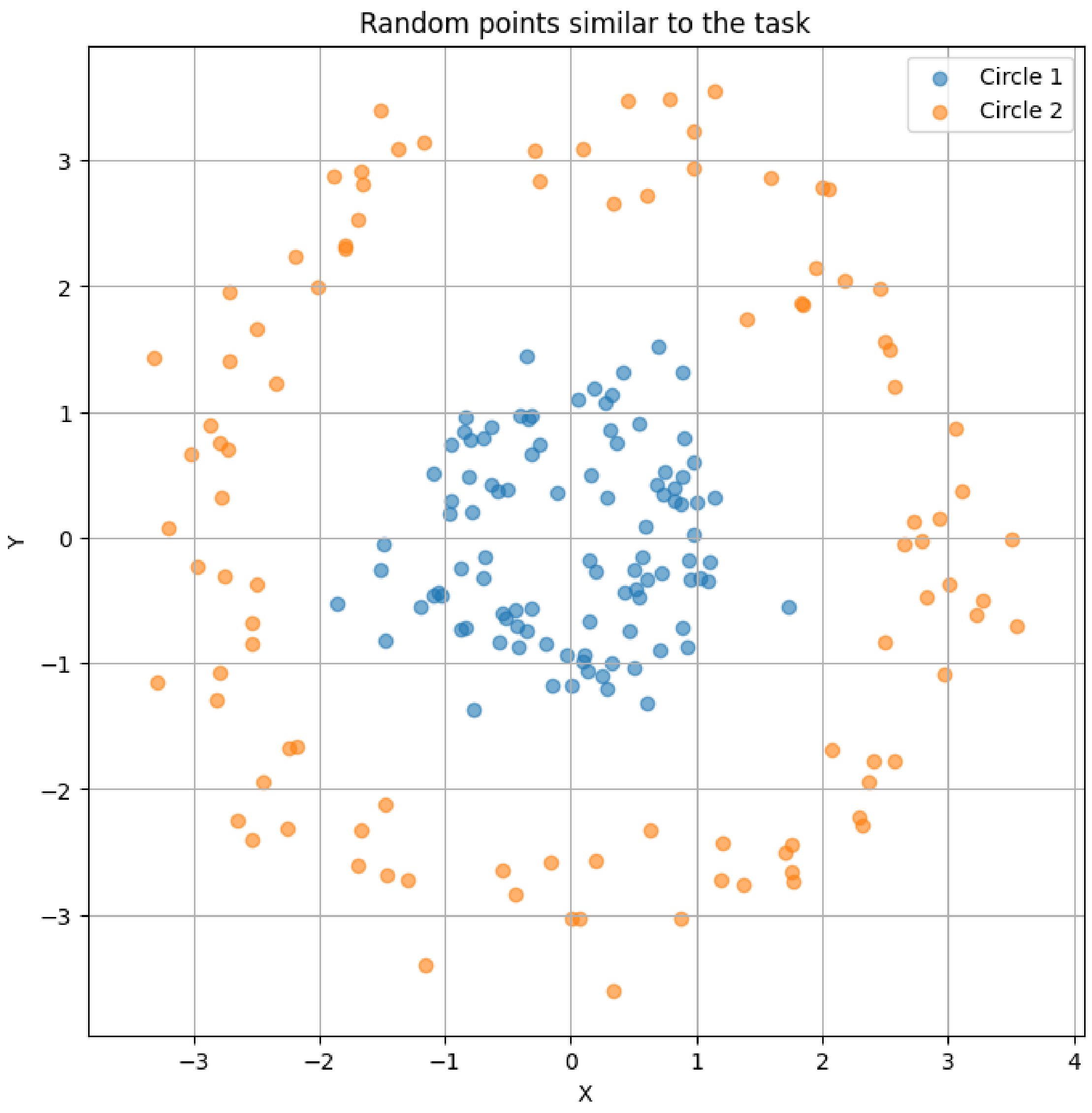
test

December 19, 2024

1 Generate random points for 2 circles

We're generating based on radius and a noise level, which gives a similar distribution to the task.

[48]:
```
import numpy as np
import matplotlib.pyplot as plt

num_points = 100
radius1 = 1
radius2 = 3
noise_level = 0.3

angles = np.linspace(0, 2 * np.pi, num_points)

circle1_x = radius1 * np.cos(angles) + np.random.normal(0, noise_level,␣
 ↪num_points)
circle1_y = radius1 * np.sin(angles) + np.random.normal(0, noise_level,␣
 ↪num_points)

angles = np.linspace(0, 2 * np.pi, num_points)

circle2_x = radius2 * np.cos(angles) + np.random.normal(0, noise_level,␣
 ↪num_points)
circle2_y = radius2 * np.sin(angles) + np.random.normal(0, noise_level,␣
 ↪num_points)

plt.figure(figsize=(8, 8))
plt.scatter(circle1_x, circle1_y, label='Circle 1', alpha=0.6)
plt.scatter(circle2_x, circle2_y, label='Circle 2', alpha=0.6)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Random points similar to the task')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.savefig('random_points.png')
plt.show()
```



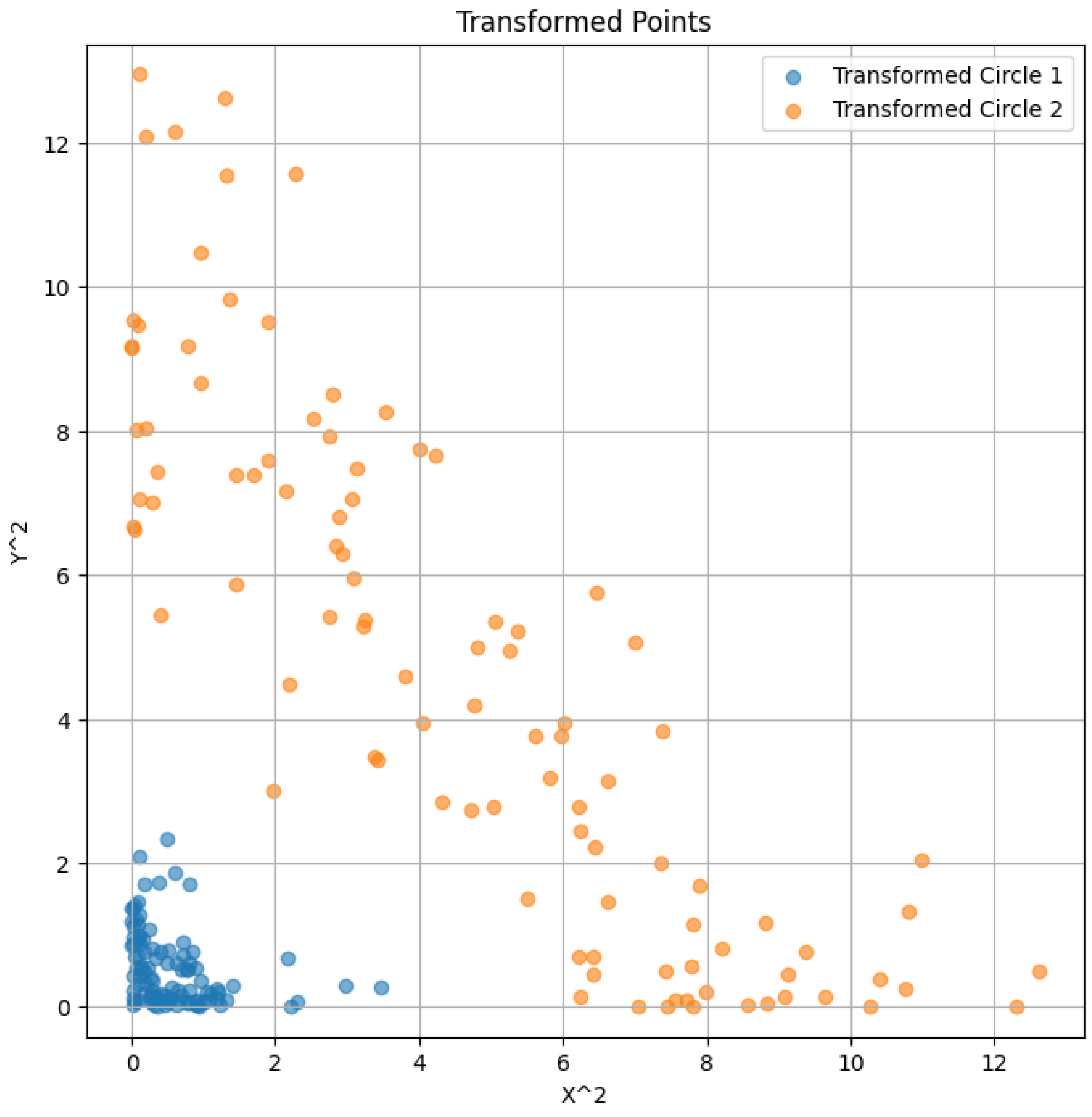
2 Transform the points by squaring their coordinates

We chose squaring as it very simple (and as the task asked to map it to another 2D space), but there are many ways to transform the data to make it linearly separable.

We can see, just by looking at the plot, that the transformed points are linearly separable, as the decision boundary will be a simple line.

[49]:
```
transformed_circle1_x = np.square(circle1_x)
transformed_circle1_y = np.square(circle1_y)
transformed_circle2_x = np.square(circle2_x)
transformed_circle2_y = np.square(circle2_y)
```

```
plt.figure(figsize=(8, 8))
plt.scatter(transformed_circle1_x, transformed_circle1_y, label='Transformed␣
 ↪Circle 1', alpha=0.6)
plt.scatter(transformed_circle2_x, transformed_circle2_y, label='Transformed␣
 ↪Circle 2', alpha=0.6)
plt.xlabel('X^2')
plt.ylabel('Y^2')
plt.title('Transformed Points')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.savefig('transformed_points.png')
plt.show()
```

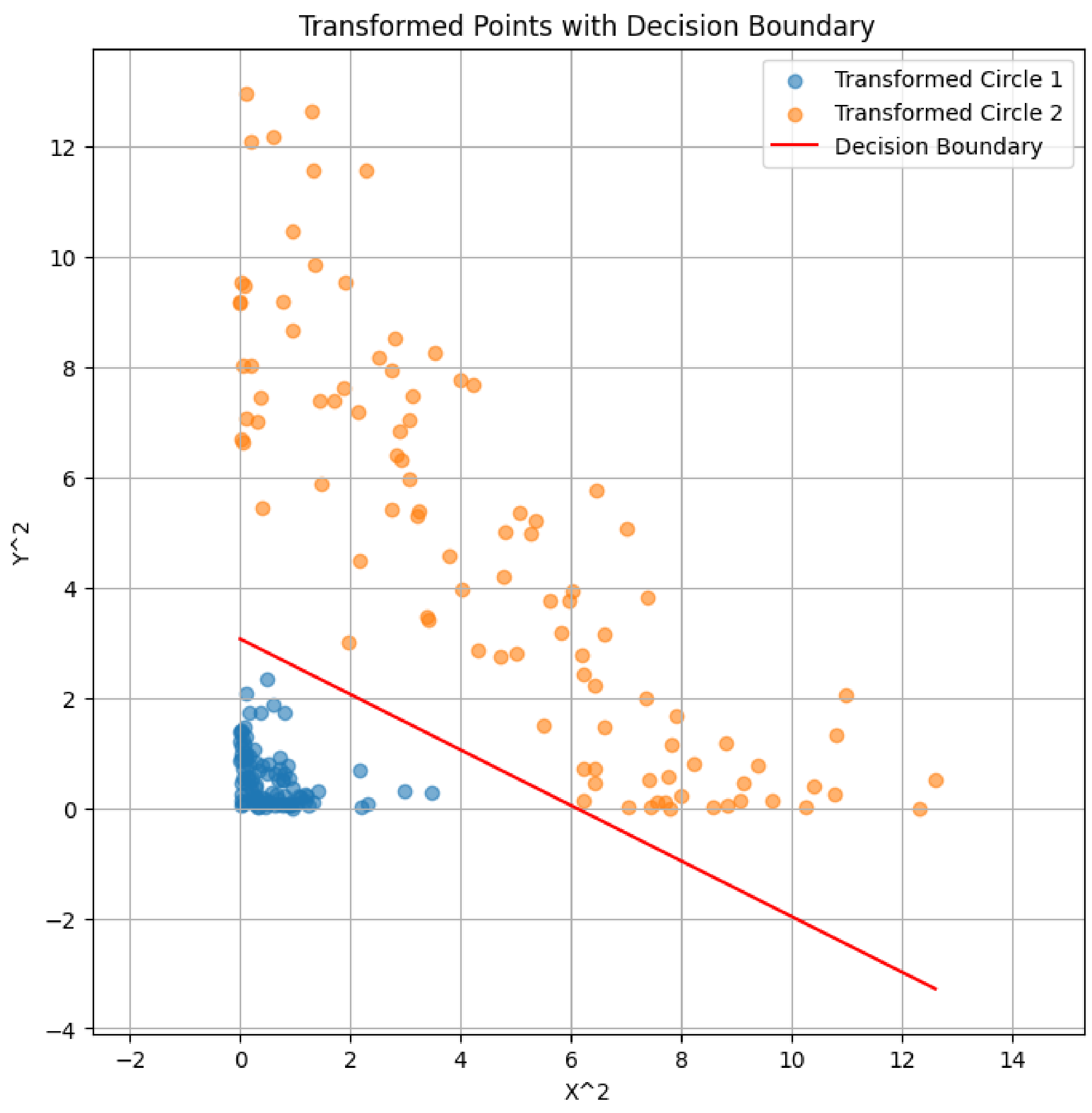


3 Train a single layer perceptron

We're using the Perceptron class from sklearn, as it is a simple and effective.

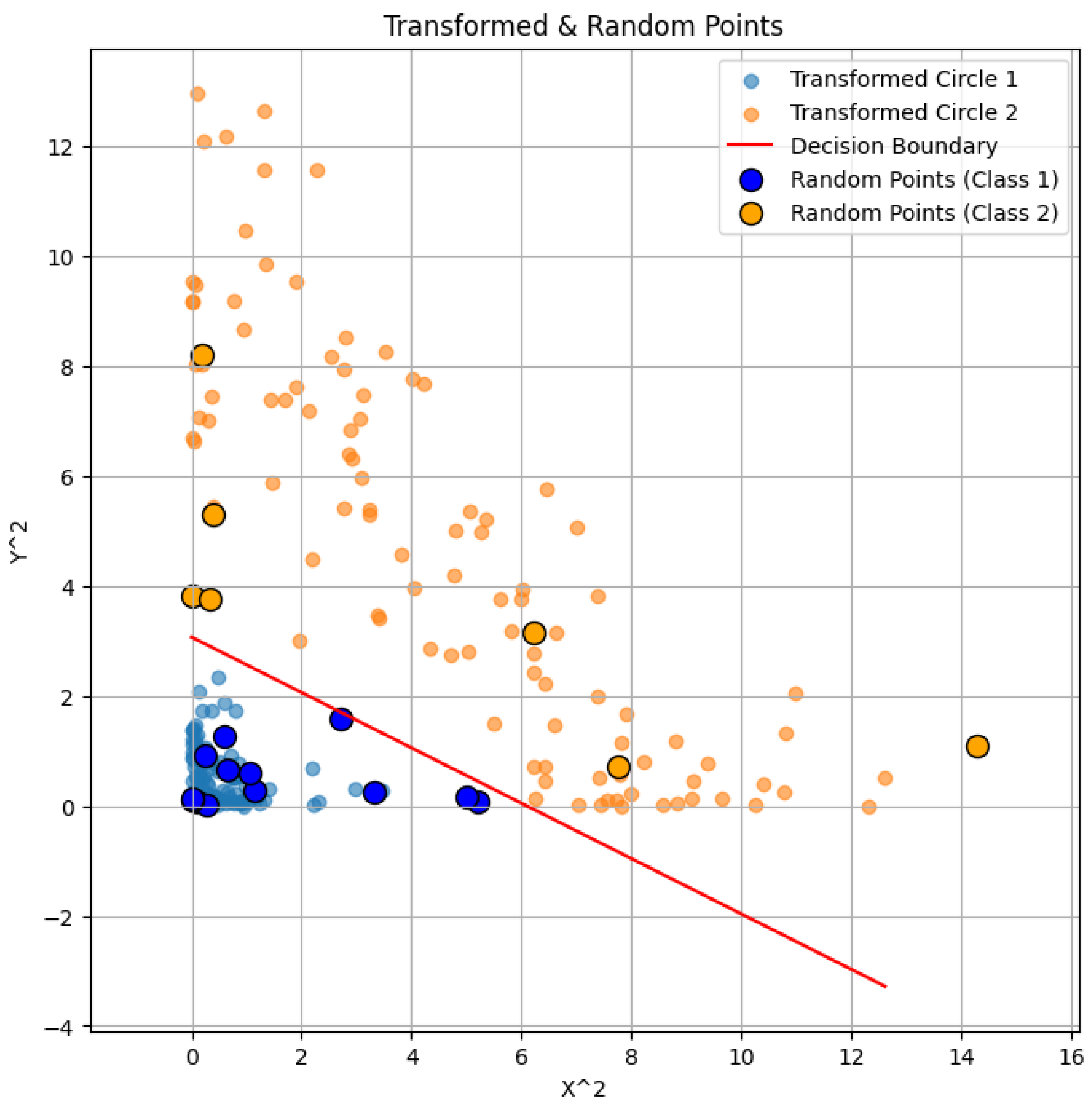
It's not always perfect (it have an angle of -45 degrees), but it's close to the task.

[50]:
```python
from sklearn.linear_model import Perceptron

X = np.vstack((np.column_stack((transformed_circle1_x, transformed_circle1_y)),
               np.column_stack((transformed_circle2_x, transformed_circle2_y))))
y = np.hstack((np.zeros(num_points), np.ones(num_points)))

single_layer_perceptron = Perceptron(max_iter=1000, tol=1e-3)
single_layer_perceptron.fit(X, y)

coef = single_layer_perceptron.coef_[0]
intercept = single_layer_perceptron.intercept_

plt.figure(figsize=(8, 8))
plt.scatter(transformed_circle1_x, transformed_circle1_y, label='Transformed␣
 ↪Circle 1', alpha=0.6)
plt.scatter(transformed_circle2_x, transformed_circle2_y, label='Transformed␣
 ↪Circle 2', alpha=0.6)

x_values = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
plt.plot(x_values, decision_boundary, label='Decision Boundary', color='red')

plt.xlabel('X^2')
plt.ylabel('Y^2')
plt.title('Transformed Points with Decision Boundary')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.savefig('transformed_points_with_decision_boundary.png')
plt.show()
```



4 Generate 20 random points and transform then classify them

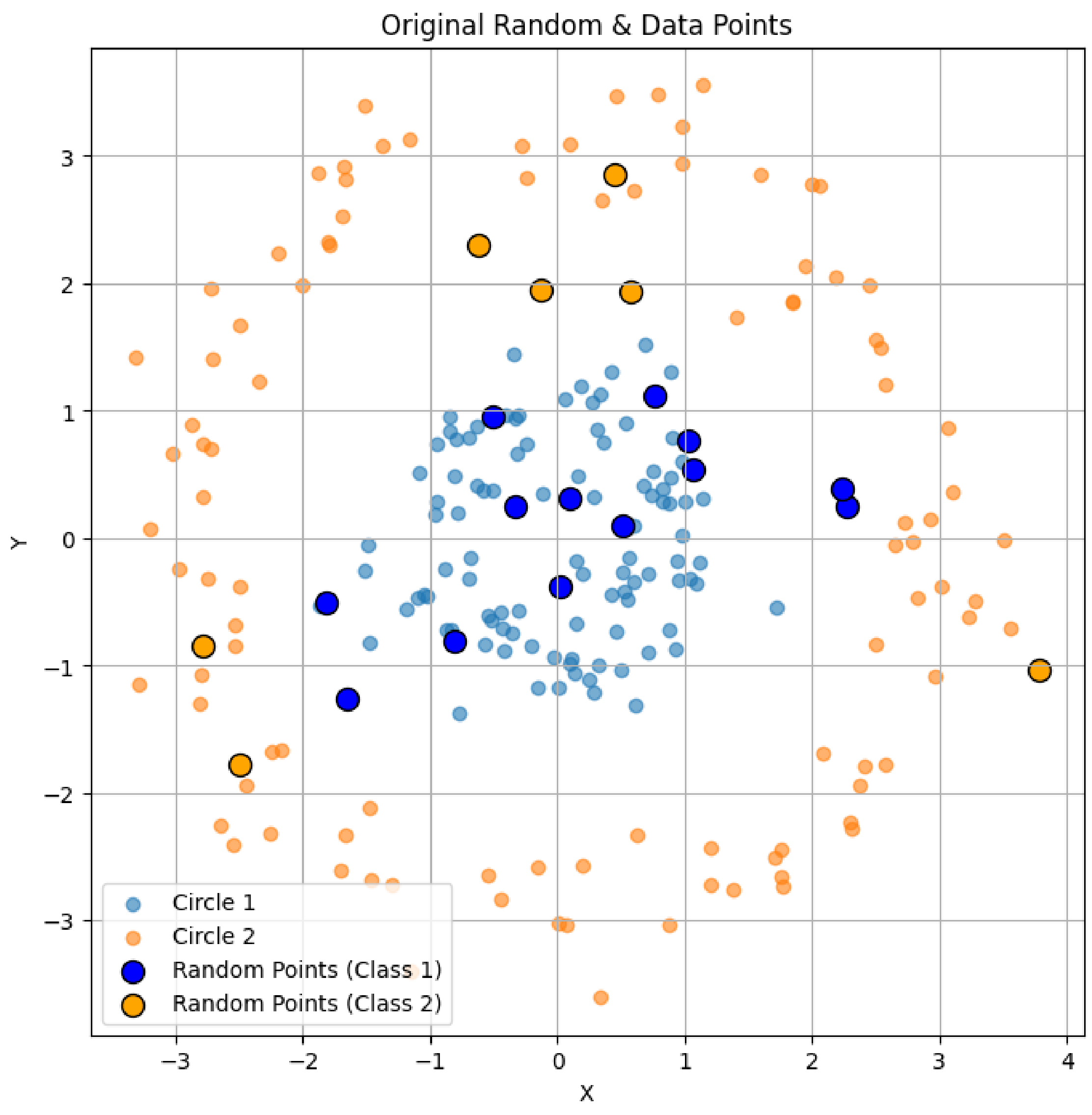
[51]:
```
random_points = np.random.normal(0, 1.4, (20, 2))

transformed_random_points = np.column_stack((random_points[:, 0]**2,␣
↪random_points[:, 1]**2))

predicted_classes = single_layer_perceptron.predict(transformed_random_points)

plt.figure(figsize=(8, 8))
plt.scatter(transformed_circle1_x, transformed_circle1_y, label='Transformed␣
 ↪Circle 1', alpha=0.6)
plt.scatter(transformed_circle2_x, transformed_circle2_y, label='Transformed␣
 ↪Circle 2', alpha=0.6)

x_values = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)
decision_boundary = -(coef[0] * x_values + intercept) / coef[1]
plt.plot(x_values, decision_boundary, label='Decision Boundary', color='red')

colors = ['blue' if cls == 0 else 'orange' for cls in predicted_classes]
for point, color in zip(transformed_random_points, colors):
    plt.scatter(point[0], point[1], color=color, edgecolor='black', s=100)

plt.scatter([], [], color='blue', label='Random Points (Class 1)',␣
 ↪edgecolor='black', s=100)
plt.scatter([], [], color='orange', label='Random Points (Class 2)',␣
 ↪edgecolor='black', s=100)

plt.xlabel('X^2')
plt.ylabel('Y^2')
plt.title('Transformed & Random Points')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.savefig('transformed_random_data_points.png')
plt.show()
```



5 Plot the original points with colors based on classification

[52]:
```
plt.figure(figsize=(8, 8))
plt.scatter(circle1_x, circle1_y, label='Circle 1', alpha=0.6)
plt.scatter(circle2_x, circle2_y, label='Circle 2', alpha=0.6)

colors = ['blue' if cls == 0 else 'orange' for cls in predicted_classes]
for point, color in zip(random_points, colors):
    plt.scatter(point[0], point[1], color=color, edgecolor='black', s=100)

plt.scatter([], [], color='blue', label='Random Points (Class 1)',␣
 ↪edgecolor='black', s=100)
```

```
plt.scatter([], [], color='orange', label='Random Points (Class 2)',␣
  ↪edgecolor='black', s=100)

plt.xlabel('X')
plt.ylabel('Y')
plt.title('Original Random & Data Points')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.savefig('original_random_data_points.png')
plt.show()
```


Original Random & Data Points
Y
X
Circle 1
Circle 2
Random Points (Class 1)
Random Points (Class 2)