CS 559 Project 1

Fall 2022

Name: Akshay Atam
CWID: 20016304
aatam@stevens.edu

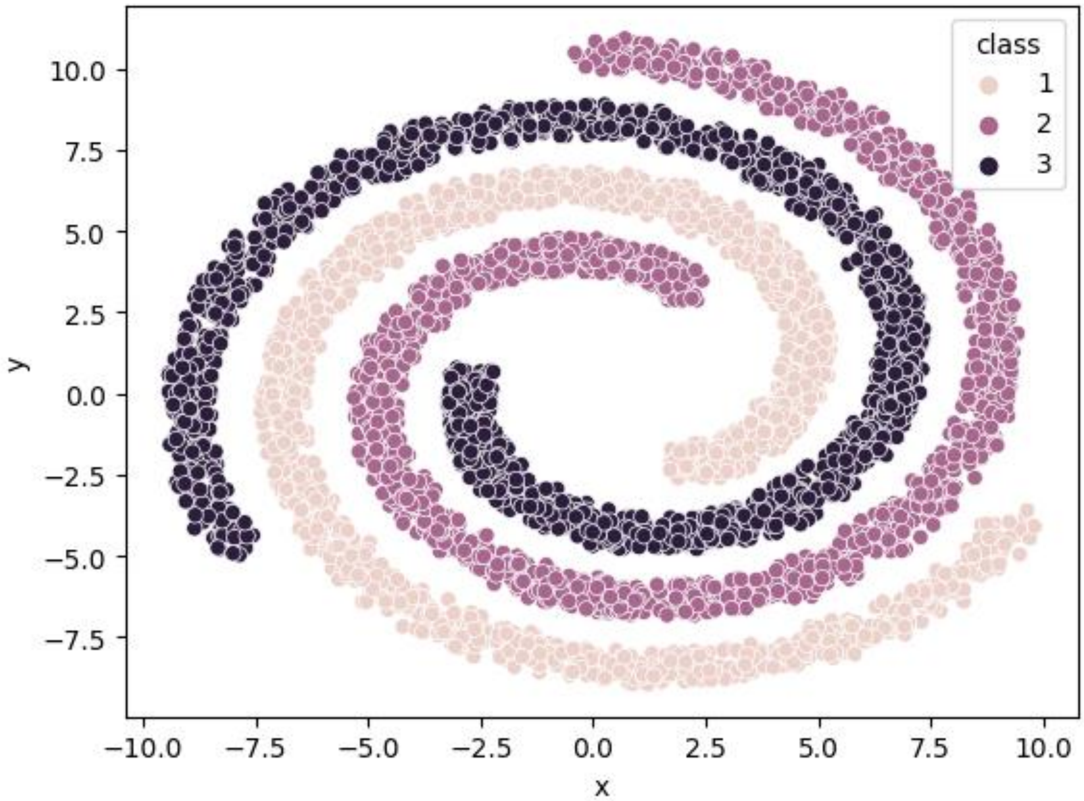
Introduction

Machine Learning enables us to process data in order to achieve desired results. Different tasks require learning of different architectures and each technique is different from the other. This project revolves around classification of non-linear data using various machine learning methods.

This report is divided into seven sections. The first section provides a brief introduction of the dataset, while the subsequent sections shows the working of the different tasks given with the output with the last section for conclusions and future scope.

Dataset Introduction

There are two CSV files given for this project – Data_train.csv containing the training examples and Data_test.csv containing the test examples. We can view the training dataset through a scatter plot from the seaborn library.



The above figure shows a two-dimensional view of the dataset. The data provided have two features and one class label. There are in total of three classes. Upon plotting the data into a scatter plot, we see that the data provided is spiral in nature.

A spiral data would require some transformations in order to work with a linear classifier. A linear classifier such as Logistic Regression will fail to classify the dataset and will lead to poor accuracy.

Spiral datasets are usually not common in the Machine Learning space but they are useful in fields such as astronomy.

Before applying different machine learning models, we need to initialize the training and testing datasets. This can be simply achieved by the following piece of code:

```
df_train = pd.read_csv('Data_train.csv', usecols=['x','y','class'])
df_test = pd.read_csv('Data_test.csv', usecols=['x','y','class'])

# split data
X_train = df_train.drop('class', axis=1)
X_test = df_test.drop('class', axis=1)

y_train = df_train['class']
y_test = df_test['class']
```

Task 0: Naïve Logistic Regression

The first task requires implementation of simple Logistic Regression on the base dataset. We need to call the LogisticRegression module from the scikit-learn package

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr.predict(X_test)
print(f"Accuracy on Naive Logistic Regression: {lr.score(X_test, y_test)}%")
```

The code outputs the following:

```
Accuracy on Naive Logistic Regression: 34.773333333333334%
```

As we can see, the accuracy using logistic regression is very poor. This is due to the fact that logistic regression is trying to fit a linear model (line) onto our dataset. As we can see from the spiral dataset, we cannot separate all the points without having to misclassify any. The poor accuracy shows us that majority of the points have been misclassified.

Task 1: Train Data Transformation

This section involves the transformation of data to a new space where the data points are linearly separable. To do this, we need to map the features with a basis function and transform it in the new space.

A log transformation is used to transform the points to the new space. Additionally, an exponential variant was also chosen but during the final traning and testing, the logarithmic function showed more accuracy. Hence, it was chosen.

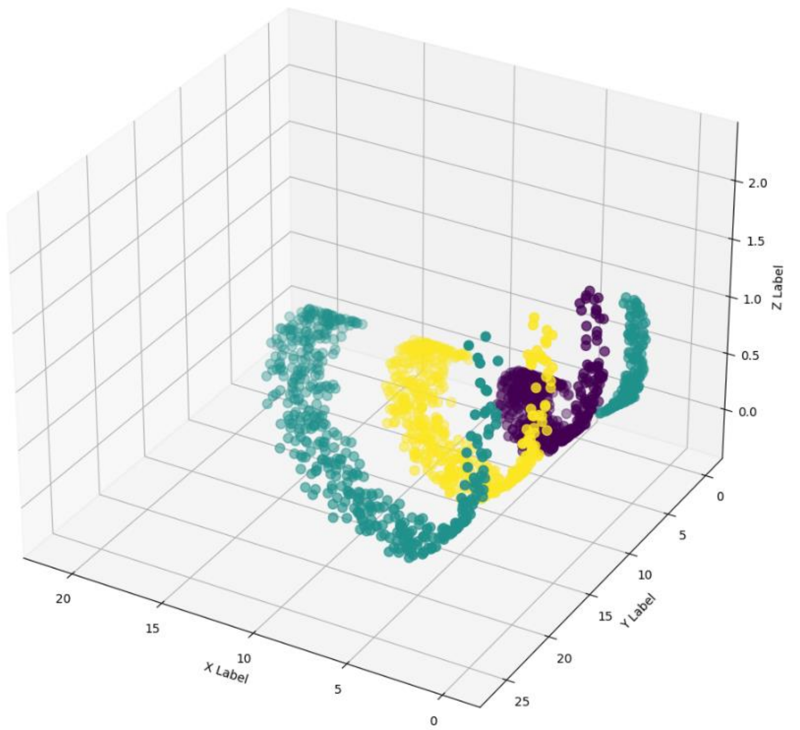
First, the input features were converted to numpy arrays.

```
x = np.array(df_train['x'])
y = np.array(df_train['y'])
label = np.array(df_train['class'])
```

Second, a mapping function is used to view our transformation in the new space. A total of three features were chosen.

```
def mapping(x, y):
    z = np.c_[(x, y)]

    a = z[:,0]
    b = z[:,1]

    z_1 = a * np.log(a)
    z_2 = b * np.log(b)
    z_3 = 0.5**a * np.log(b)

    # using np.exp (little lower accuracy)
    #z_1 = np.exp(np.sqrt(a))
    #z_2 = np.exp(np.sqrt(b))
    #z_3 = np.exp(np.sqrt(a+b))

    trans_x = np.array([z_1, z_2, z_3])
    return trans_x
```

The following is the output for using the log transformation



It can be seen that the features are separated except for some values of a certain class not having the points together. This will result in a lower accuracy, however, the overall accuracy is increased when using this transformation.

Task 2: Linear Parametric Classification

This task applies the transformed data into the Logistic Regression model with using GridSearchCV. A total of 100 alpha values were chosen between 10^{-5} and 10^{5}. The use of different values of alpha is to find the best model and report its accuracy. Following were the steps to achieve the goal.

1. Import Libraries

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
```

2. Helper function for making the model and evaluating the model

```
def evalute(model, X_train, X_test, y_train, y_test):
    print(f"Training Score: {model.score(X_train, y_train)}")
    print(f"Testing  Score: {model.score(X_test, y_test)}")

    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print("Weights:")
    print(np.hstack((model.intercept_[:,None], model.coef_)))

    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True)
    plt.show()

def make_models(X_train, X_test, y_train, y_test):
    print(f"Result:")
    model = LogisticRegression()
    model.fit(X_train, y_train)
    evalute(model, X_train, X_test, y_train, y_test)


    print(f"Optimizing Model....")
    alpha = np.linspace(10**(-5), 10**(5), 100)

    params = {
        "tol": alpha
    }

    grid = GridSearchCV(LogisticRegression(), params, n_jobs=-1,
verbose=1, return_train_score=True, cv=2)
    res = grid.fit(X_train, y_train)

    scores = res.cv_results_["mean_test_score"]

    plt.figure(figsize=(15, 5))
    sns.lineplot(x=alpha, y=scores)
    plt.show()

    print(f'Best Model aplha value:
```

```
{res.best_params_["tol"]}\nAccuracy on test:
{(res.best_score_)*100}%.')
```

3. Add the transformation columns to the dataframe

```
def add_cols(df):
    df['f1'] = df.x * np.log(df.x)
    df['f2'] = df.y * np.log(df.y)
    df['f3'] = 0.5**df.x * np.log(df.y)

    # using exponents
    #df['f1'] = np.exp(np.sqrt(df.x))
    #df['f2'] = np.exp(np.sqrt(df.y))
    #df['f3'] = np.exp(np.sqrt(df.x + df.y))
    # remove NaN values (decreases data but increases accuracy)
    df = df.dropna()

    return df
```

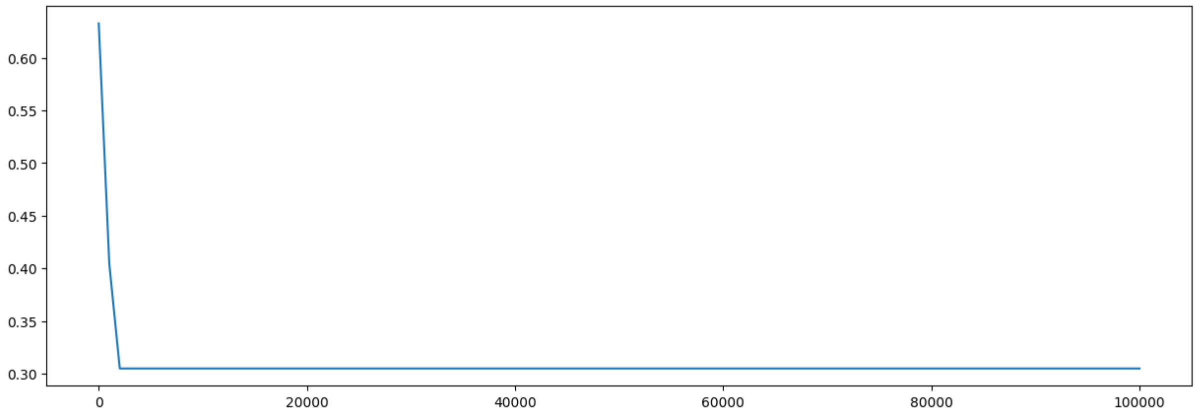
```
# create duplicate dataframe to store the pre processed values
df_tr = pd.read_csv('Data_train.csv', usecols=['x','y','class'])
df_te = pd.read_csv('Data_test.csv', usecols=['x','y','class'])

df_tr = add_cols(df_tr)
df_te = add_cols(df_te)

x_tr = df_tr.drop(['x', 'y', 'class'], axis=1)
x_te = df_te.drop(['x', 'y', 'class'], axis=1)

y_tr = df_tr['class']
y_te = df_te['class']
```

4. Result

```
# show output
make_models(x_tr, x_te, y_tr, y_te)
```

The output is as follows:

```
Result:
Training Score: 0.615530303030303
Testing  Score: 0.6186915887850467
Weights:
[[ 2.97712126 -0.2526569  -0.13651127 -1.61946735]
 [-2.16212877  0.17675564  0.09302591  1.42418913]
 [-0.8149925   0.07590125  0.04348536  0.19527822]]
```




```
Best Model aplha value: 1e-05
Accuracy on test: 63.25757575757576%.
```

Task 3: Transformation using Kernel Method

This task requires the use of kernel functions to transform the original dataset. There are five different kernel functions used in this project, all imported from sklearn.preprocessing.

To make the code neat, kernels are defined in a dictionary and are subsequently evaluated. The train and test features are transformed using the fit_transform method. The entire process looks like the following:

- Import libraries

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import SplineTransformer
from sklearn.preprocessing import FunctionTransformer
```

- Create a kernel dictionary

```
kernels = {
    "PolynomialFeatures": PolynomialFeatures(),
    "PowerTransformer": PowerTransformer(),
    "QuantileTransformer": QuantileTransformer(),
    "SplineTransformer": SplineTransformer(),
    "FunctionTransformer": FunctionTransformer(),
}
```

- Helper functions for making the model and its evaluation

```
def evalute(model, X_train, X_test, y_train, y_test):
    print(f"Training Score : {model.score(X_train, y_train)}")
    print(f"Testing  Score : {model.score(X_test, y_test)}")

    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True)
    plt.show()
```

```
def make_models(X_train, X_test, y_train, y_test):
    print(f"Result:")
    model = LogisticRegression()
    model.fit(X_train, y_train)
    evalute(model, X_train, X_test, y_train, y_test)


    print(f"Optimizing Model....")
    alpha = np.linspace(10**(-5), 10**(5), 100)

    params = {
```

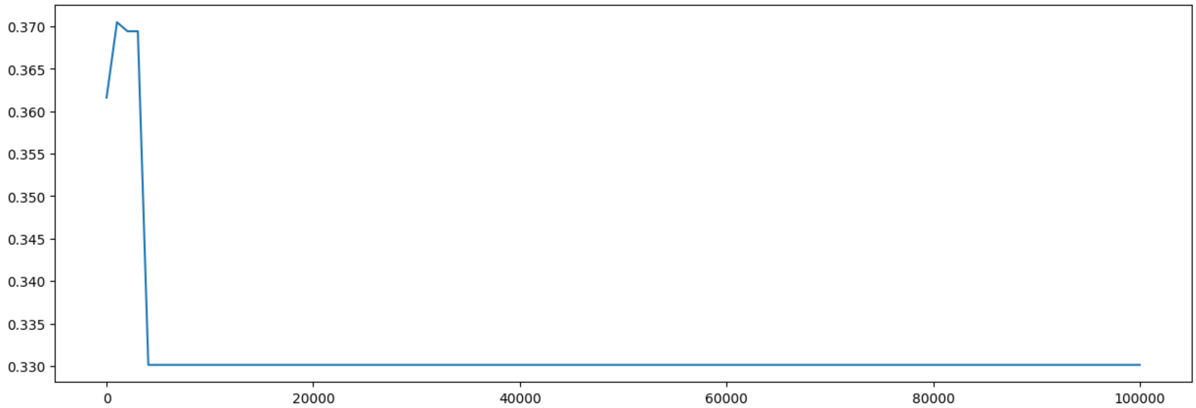
```
        "tol": alpha
    }

    grid = GridSearchCV(LogisticRegression(), params, n_jobs=-1,
verbose=1, return_train_score=True, cv=2)
    res = grid.fit(X_train, y_train)

    scores = res.cv_results_["mean_test_score"]

    plt.figure(figsize=(15, 5))
    sns.lineplot(x=alpha, y=scores)
    plt.show()

    print(f'Best model alpha value:
{res.best_params_["tol"]}\nAccuracy on test:
{(res.best_score_)*100}%.')
```

- Output (original data)

```
for name, kernel in kernels.items():
    print(f"Transformation using Kernel {name}")
    X_tr = kernel.fit_transform(X_train)
    X_ts = kernel.fit_transform(X_test)
    make_models(X_tr, X_ts, y_train, y_test)
    print()
    print("="*120)
    print()
```

Polynomial features

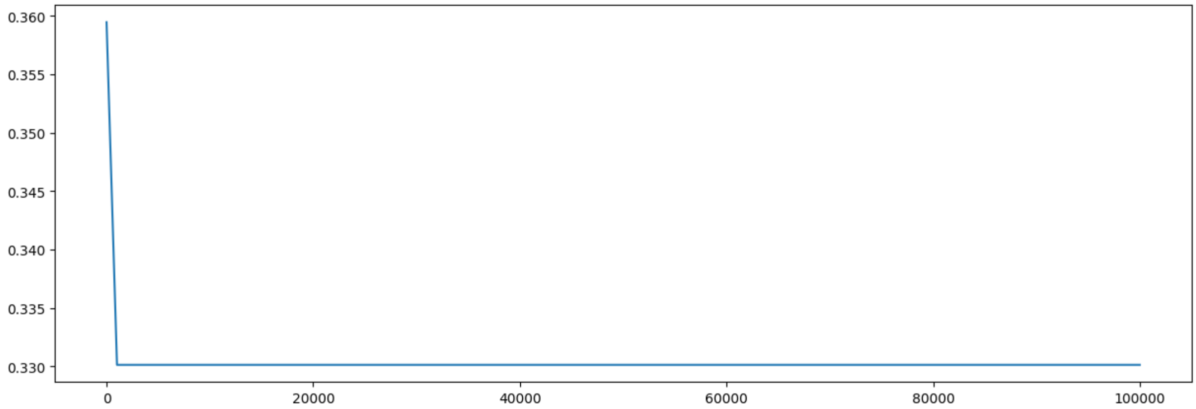
```
Transformation using Kernel PolynomialFeatures
Result:
Training Score : 0.36675555555555556
Testing  Score : 0.3488
```




```
Best model alpha value: 1010.10102
Accuracy on test: 37.048826849938 23%.
```

Power transformer

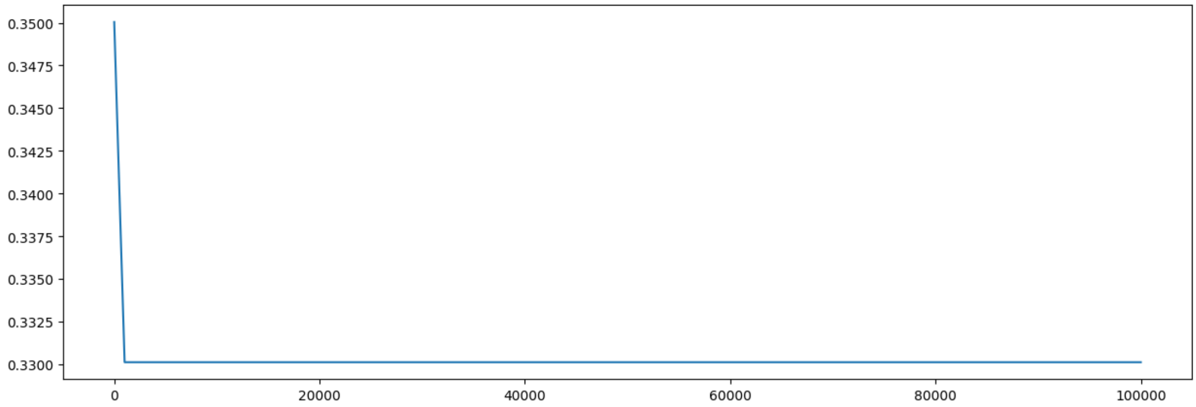
```
Transformation using Kernel PowerTransformer
Result:
Training Score : 0.3624888888888889
Testing  Score : 0.3456
```




```
Best model alpha value: 1e-05
Accuracy on test: 35.9465413830018%.
```

Quantile transformer

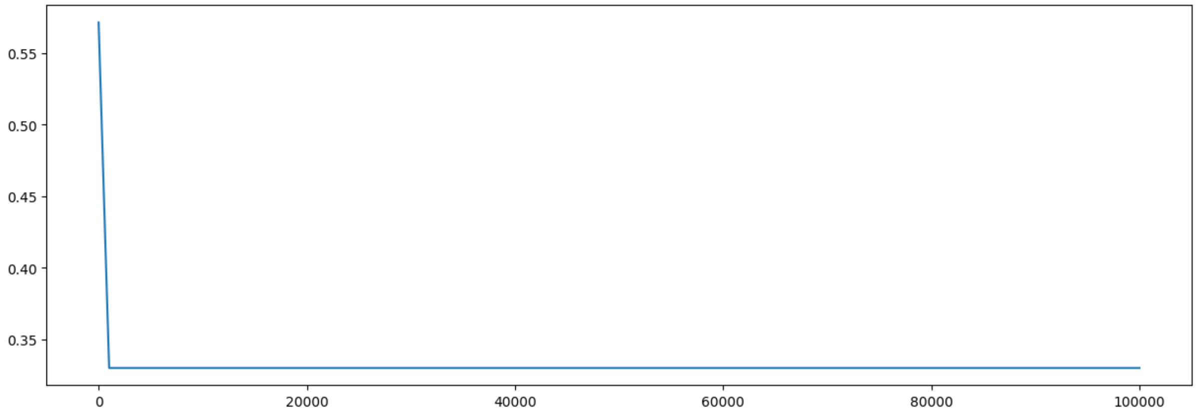
```
Transformation using Kernel QuantileTransformer
Result:
Training Score : 0.3511111111111111
Testing  Score : 0.33226666666666665
```




```
Best model alpha value: 1e-05
Accuracy on test: 35.00432861248248%.
```

Spline transformer

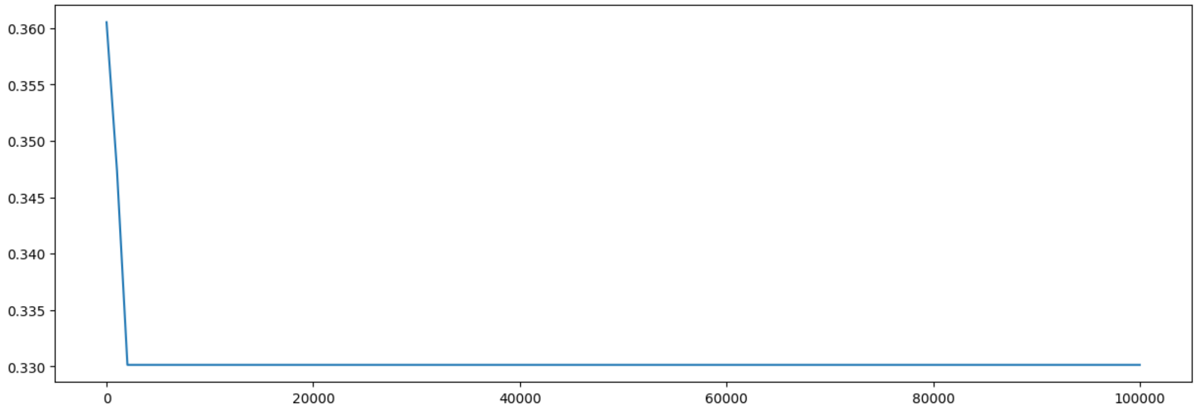
```
Transformation using Kernel SplineTransformer
Result:
Training Score : 0.5831111111111111
Testing  Score : 0.5850666666666666
```




```
Best model alpha value: 1e-05
Accuracy on test: 57.13757731200244%.
```

Function transformer

```
Transformation using Kernel FunctionTransformer
Result:
Training Score : 0.36266666666666667
Testing  Score : 0.34773333333333334
```




```
Best model alpha value: 1e-05
Accuracy on test: 36.05323965797893%.
```

- Output (Transformed data)

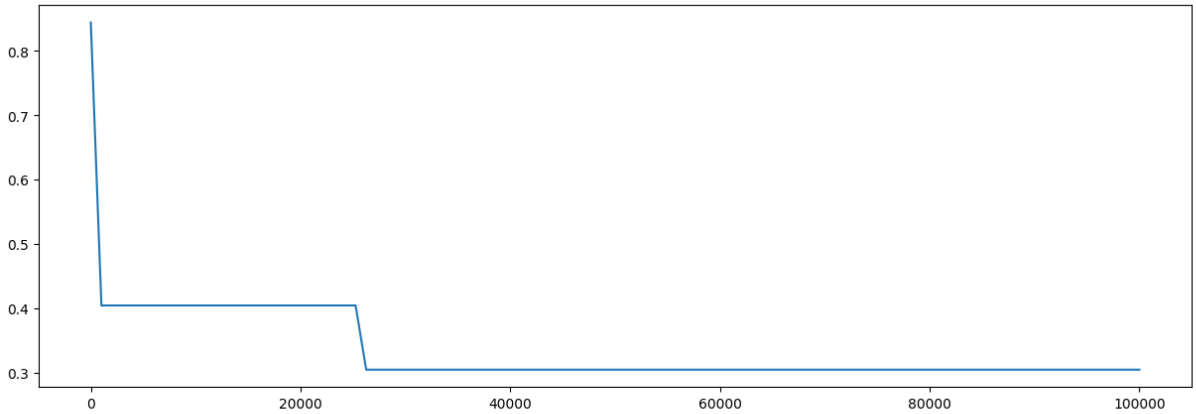
```
# for transformed data
for name, kernel in kernels.items():
    print(f"Transformation using Kernel {name}")
    X_tr = kernel.fit_transform(x_tr)
    X_ts = kernel.fit_transform(x_te)
    make_models(X_tr, X_ts, y_tr, y_te)
    print()
```

```
    print("="*120)
    print()
```

Following is the output:

Polynomial features

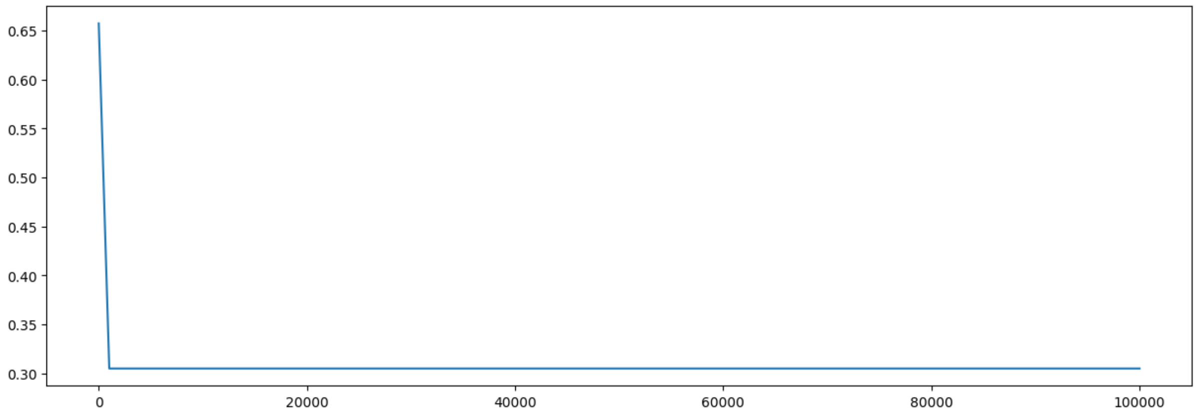
```
Transformation using Kernel PolynomialFeatures
Result:
Training Score : 0.8390151515151515
Testing  Score : 0.8448598130841122
```




```
Best model alpha value: 1e-05
Accuracy on test: 84.40656565656566%.
```

Power transformer

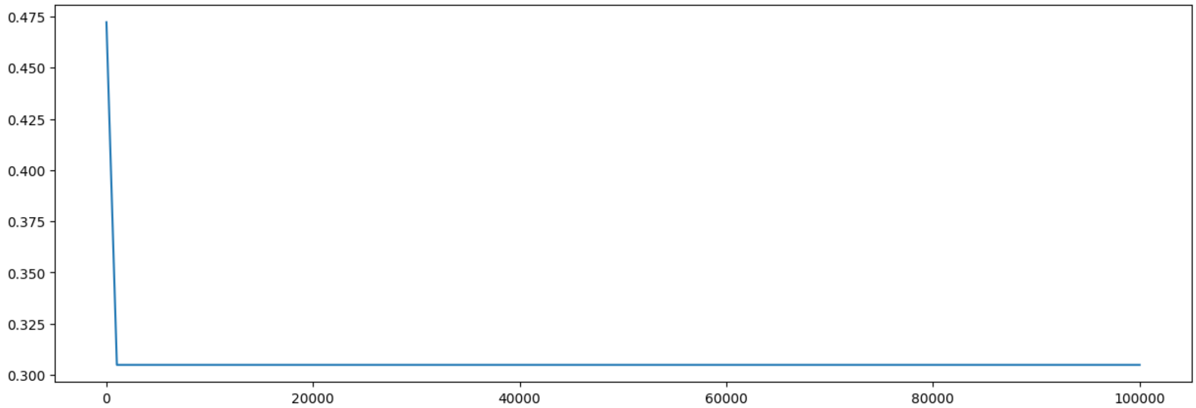
```
Transformation using Kernel PowerTransformer
Result:
Training Score : 0.6647727272727273
Testing  Score : 0.6224299065420561
```




```
Best model alpha value: 1e-05
Accuracy on test: 65.71969696969697%.
```

Quantile transformer

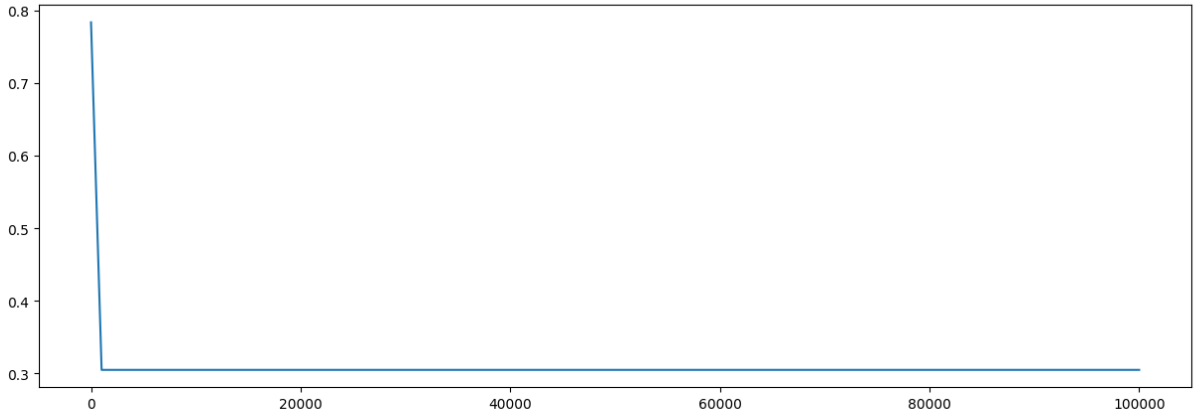
```
Transformation using Kernel QuantileTransformer
Result:
Training Score : 0.4671717171717172
Testing  Score : 0.47102803738317756
```




```
Best model alpha value: 1e-05
Accuracy on test: 47.22222222222222%.
```

Spline transformer

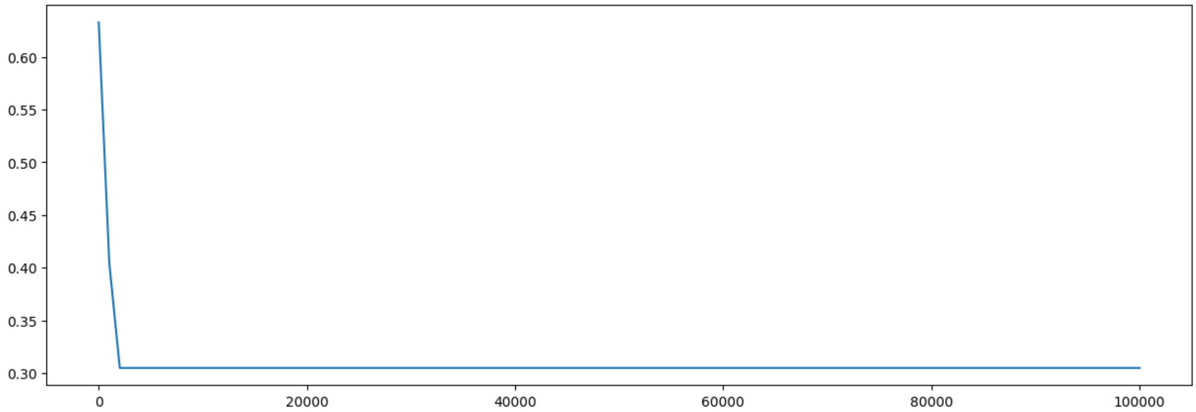
```
Transformation using Kernel SplineTransformer
Result:
Training Score : 0.8074494949494949
Testing  Score : 0.8242990654205608
```




```
Best model alpha value: 1e-05
Accuracy on test: 78.40909090909092%.
```

Function transformer

```
Transformation using Kernel FunctionTransformer
Result:
Training Score : 0.615530303030303
Testing  Score : 0.6186915887850467
```




```
Best model alpha value: 1e-05
Accuracy on test: 63.25757575757576%.
```

Task 4: Non-parametric KNN Classification

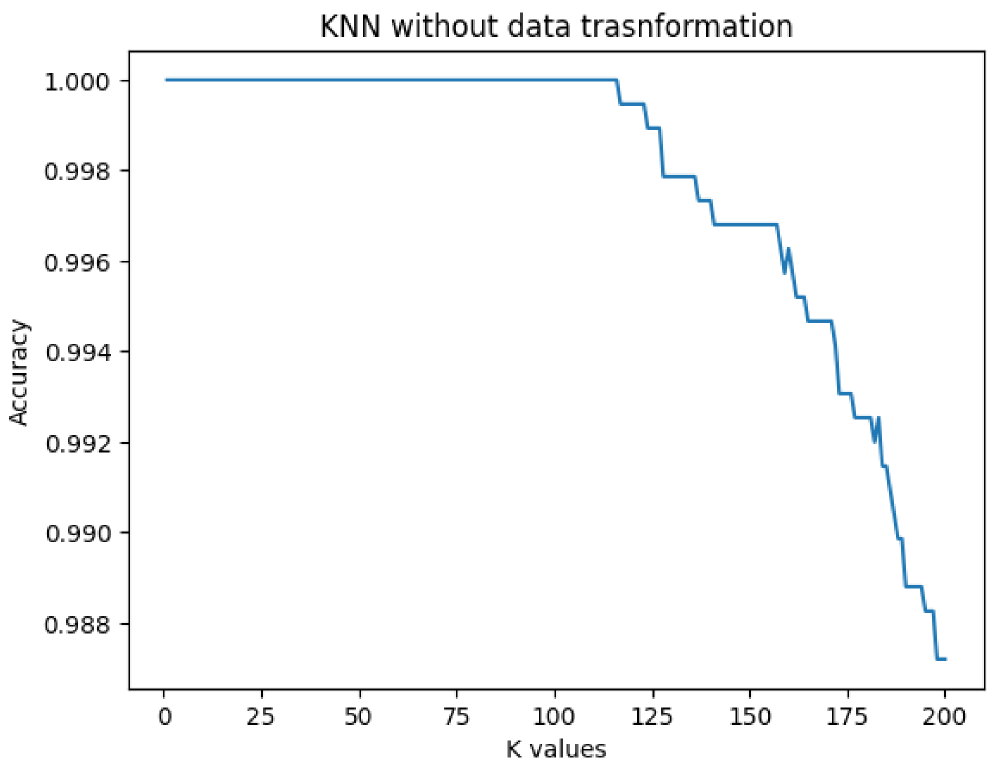
The final task is to apply K Nearest Neighbors on the dataset. There are two parts in this task – the first is to apply KNN on the original dataset and the other to apply the transformed dataset (in Task 1) and use kernel functions on the original dataset (in Task 3).

Part 1 – Without data transformations

```
from sklearn.neighbors import KNeighborsClassifier


# classifying the original data from K=1 to K=200
score=[]
K_value = {}
score_vis=[]
for i in range(1,201):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    K_value={i:knn.score(X_test, y_test)}
    score_vis.append(knn.score(X_test, y_test))
    score.append(K_value)
```

```
plt.xlabel("K values")
plt.ylabel("Accuracy")
plt.title("KNN without data trasnformation")
plt.plot(range(1,201), score_vis)
plt.show()
```

Following is the output when KNN is used on non-transformed dataset

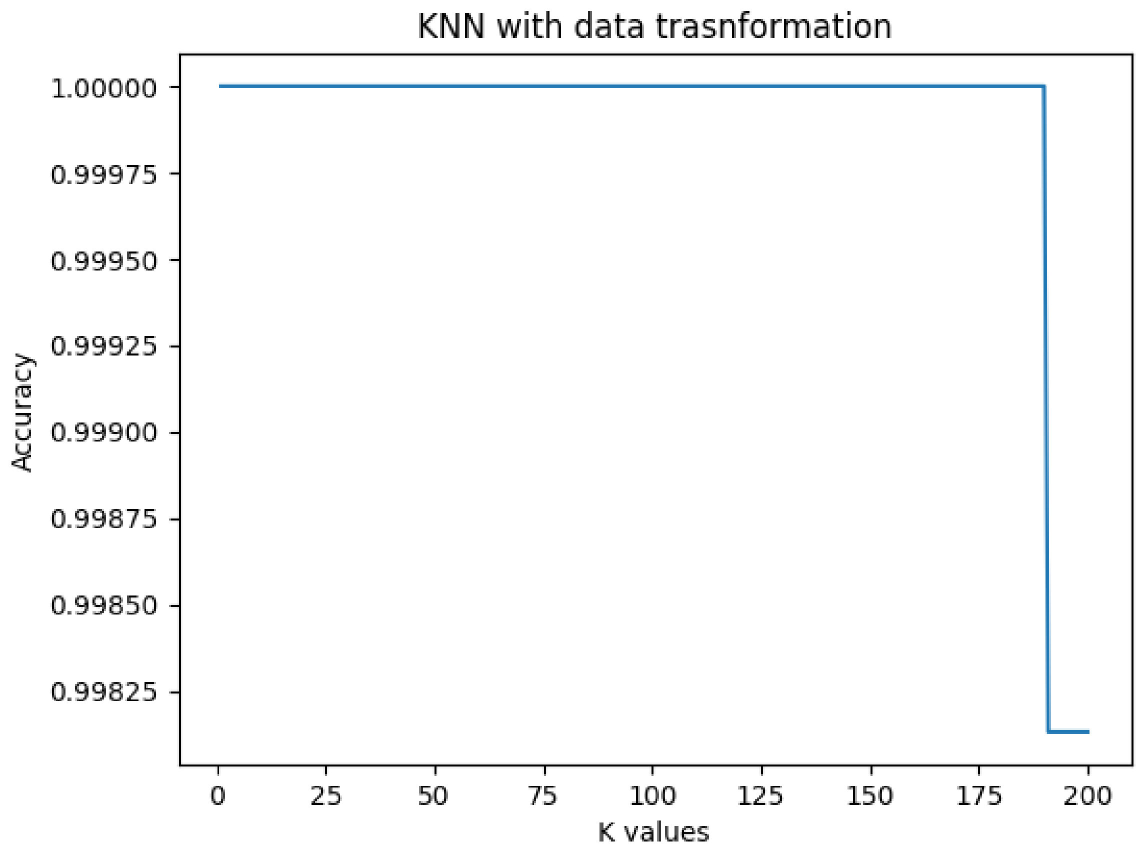


Part 2 – With data transformations

```
# task 1
score=[]
K_value = {}
score_vis=[]
for i in range(1,201):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_tr, y_tr)
    K_value={i:knn.score(x_te, y_te)}
    score_vis.append(knn.score(x_te, y_te))
    score.append(K_value)
```

```
plt.xlabel("K values")
plt.ylabel("Accuracy")
plt.title("KNN with data trasnformation")
plt.plot(range(1,201), score_vis)
plt.show()
```

After applying the transformed data, following is the output



Part 3 – Using kernel functions

```
def make_models(X_train, X_test, y_train, y_test):
    print(f"Result of Model without Optimization are....")
    model = KNeighborsClassifier(n_neighbors=3)
```

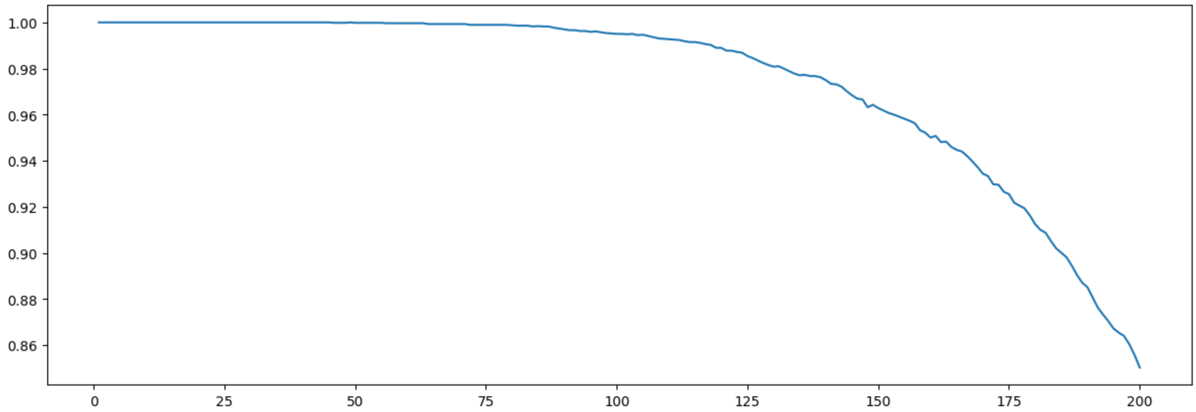
```
    model.fit(X_train, y_train)
    evalute(model, X_train, X_test, y_train, y_test)


    print(f"Optimizing Model....")
    k = range(1, 201, 1)

    params = {
        "n_neighbors": k
    }

    grid = GridSearchCV(KNeighborsClassifier(n_neighbors=3), params, 
n_jobs=-1, verbose=1, return_train_score=True, cv=2)
    res = grid.fit(X_train, y_train)

    scores = res.cv_results_["mean_test_score"]


    plt.figure(figsize=(15, 5))
    sns.lineplot(x=k, y=scores)
    plt.show()

    print(f'Accuracy: {(res.best_score_)*100}%.')
```

```
for name, kernel in kernels.items():
    print(f"Transformation using Kernel {name}")
    X_tr = kernel.fit_transform(X_train)
    X_ts = kernel.fit_transform(X_test)
    make_models(X_tr, X_ts, y_train, y_test)
    print()
    print("="*120)
    print()
```

Following are the result of using kernel methods

Polynomial features

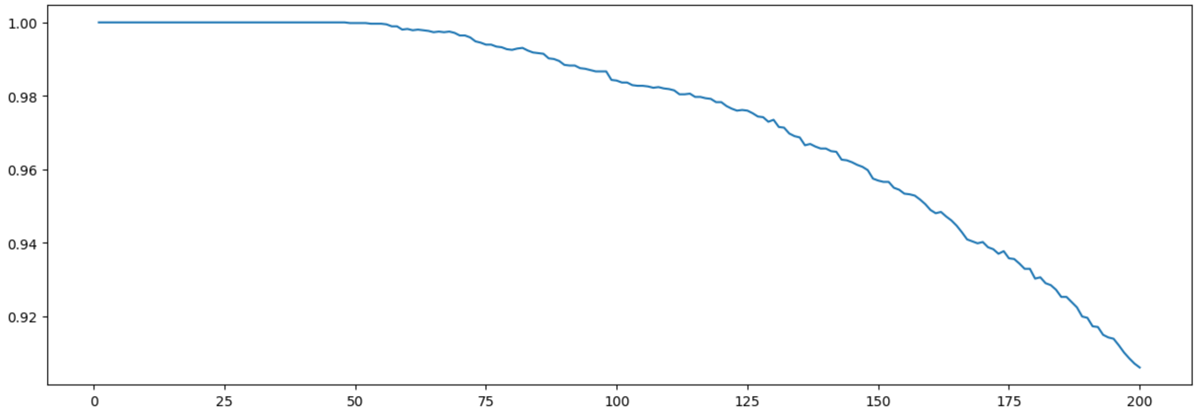
```
Transformation using Kernel PolynomialFeatures
Result of Model without Optimization are....
Training Score : 1.0
Testing  Score : 1.0
```




```
Accuracy: 100.0%.
```

Power transformer

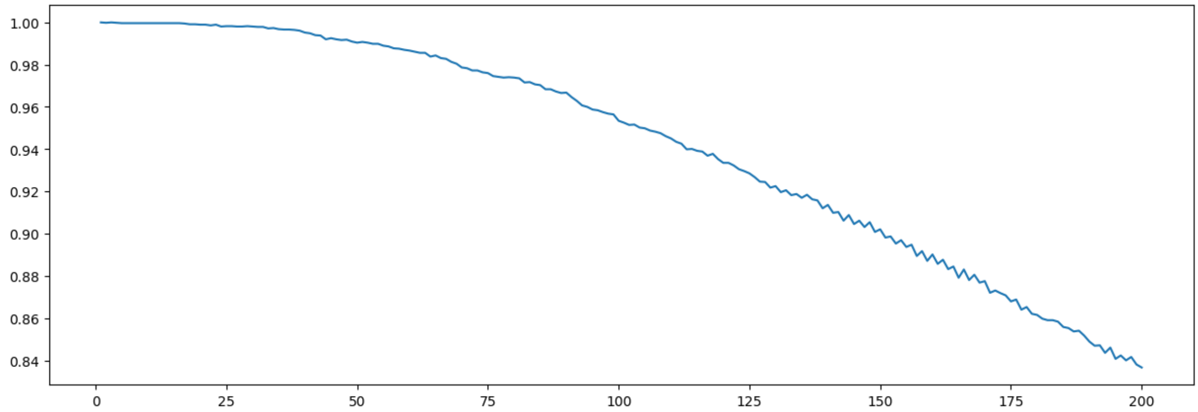
```
Transformation using Kernel PowerTransformer
Result of Model without Optimization are....
Training Score : 1.0
Testing  Score : 1.0
```




```
Accuracy: 100.0%.
```

Quantile transformer

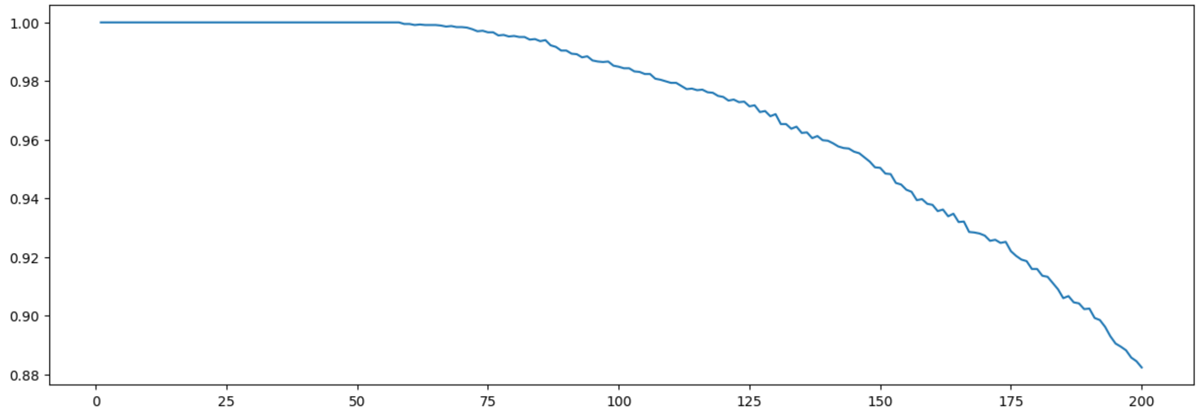
```
Transformation using Kernel QuantileTransformer
Result of Model without Optimization are....
Training Score : 1.0
Testing  Score : 1.0
```




```
Accuracy: 100.0%.
```

Spline transformer

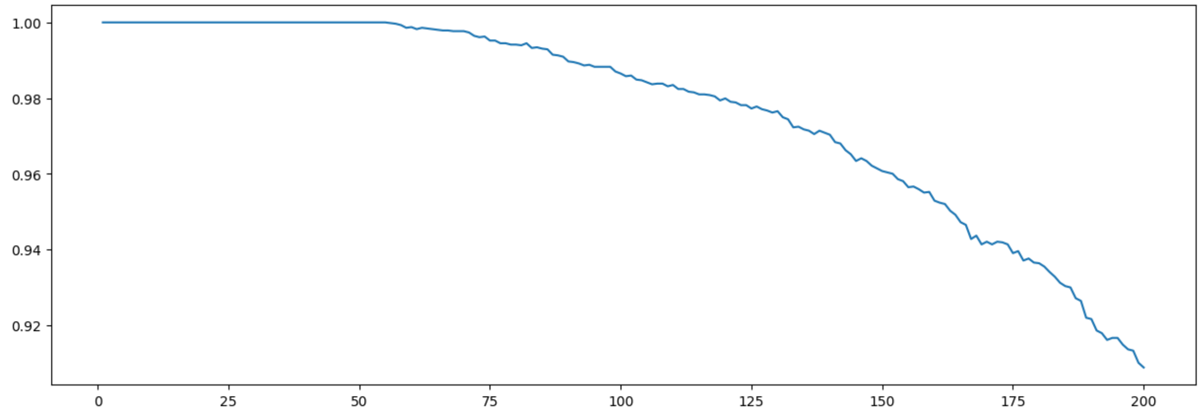
```
Transformation using Kernel SplineTransformer
Result of Model without Optimization are....
Training Score : 1.0
Testing  Score : 1.0
```




```
Accuracy: 100.0%.
```

Function transformer

```
Transformation using Kernel FunctionTransformer
Result of Model without Optimization are....
Training Score : 1.0
Testing  Score : 1.0
```




```
Accuracy: 100.0%.
```

Result

With the following project, we can summarize the results as follows

- Logistic Regression (original data)

Accuracy	34.773333333333334%

- Logistic Regression (transformed data)

Training Score	0.615530303030303 (61.5%)
Testing Score	0.6186915887850467 (61.8%)
Best alpha value	1e-05
Accuracy on test at alpha = 1e-05	63.25757575757576%

- Kernel methods
    - Logistic Regression (original data)

Kernel	Training Score	Testing Score	Best alpha value	Accuracy at best alpha
PolynomialFeatures	0.3667 (36.6%)	0.3488 (34.8%)	1010.10102	37.04882684993823%
PowerTransformer	0.3624 (36.2%)	0.3456 (34.5%)	1e-05	35.9465413830018%
QuantileTransformer	0.3511 (35.1%)	0.3322 (33.2%)	1e-05	35.00432861248248%
SplineTransformer	0.5831 (58.31%)	0.585 (58.5%)	1e-05	57.13757731200244%
FunctionTransformer	0.3626 (36.26%)	0.3477 (34.77%)	1e-05	36.05323965797893%

    - Logistic Regression (transformed data)

Kernel	Training Score	Testing Score	Best alpha value	Accuracy at best alpha
PolynomialFeatures	0.839 (83.9%)	0.8448 (84.4%)	1e-05	84.40656565656566%
PowerTransformer	0.6647 (66.4%)	0.6224 (62.2%)	1e-05	65.71969696969697%
QuantileTransformer	0.4671 (46.7%)	0.471 (47.1%)	1e-05	47.22222222222222%
SplineTransformer	0.8074 (80.74%)	0.8242 (82.42%)	1e-05	78.40909090909092%
FunctionTransformer	0.6155 (61.55%)	0.6186 (61.8%)	1e-05	63.25757575757576%

- K Nearest Neighbors (using kernels)

Kernel	Accuracy
PolynomialFeatures	100%
PowerTransformer	100%
QuantileTransformer	100%
SplineTransformer	100%
FunctionTransformer	100%

Conclusions and Future Scope

This project has shown the use of Logistic Regression, Kernel methods and K Nearest Neighbors to classify non-lineaer, spiral data. The use of kernel methods on these algorithms show acceptable accuracies with the approach that was applied. However, that does not mean that there is no room for improvement. The dataset has not been linearly separated and there also have been imputation done which resulted in the decrease of data size. Nevertheless, the imputation proved to be useful as the accuracies of Logistic Regression showed an improvement. Furthermore, it was proved with visualizations that in K Nearest Neighbors, as the value of k increases, the accuracies start to drop. Overall, the transformations improved the metrics of the machine learning models.

In future, a more robust transformation will be necessary for Logistic Regression to achieve the highest accuracy. Moreover, to handle such non-linear data, the use of Gaussian Processes, Support Vector Machines and use of similarities with Spectral Clustering would prove to be even more powerful algorithms than the ones implemented in this project.

This project provided an example classifying non-linear data through logarithmic transformation and provide necessary metrics for evaluation of the discussed algorithms

For more information and the full code, kindly refer to the jupyter notebook attached with this report.