prediction-and-analysis-using-svm

July 27, 2024

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Load the dataset
df = pd.read_csv('diabetes.csv')

# View the dimensions of the data
print("Dimensions of the data:", df.shape)

# Gather summary statistics
summary_stats = df.describe()
print(summary_stats)
```

```
Dimensions of the data: (768, 9)
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Observations:

```
1. There are missing values in columns , 'Glucose', 'BloodPressure', SkinThickness', 'Insulin'
```

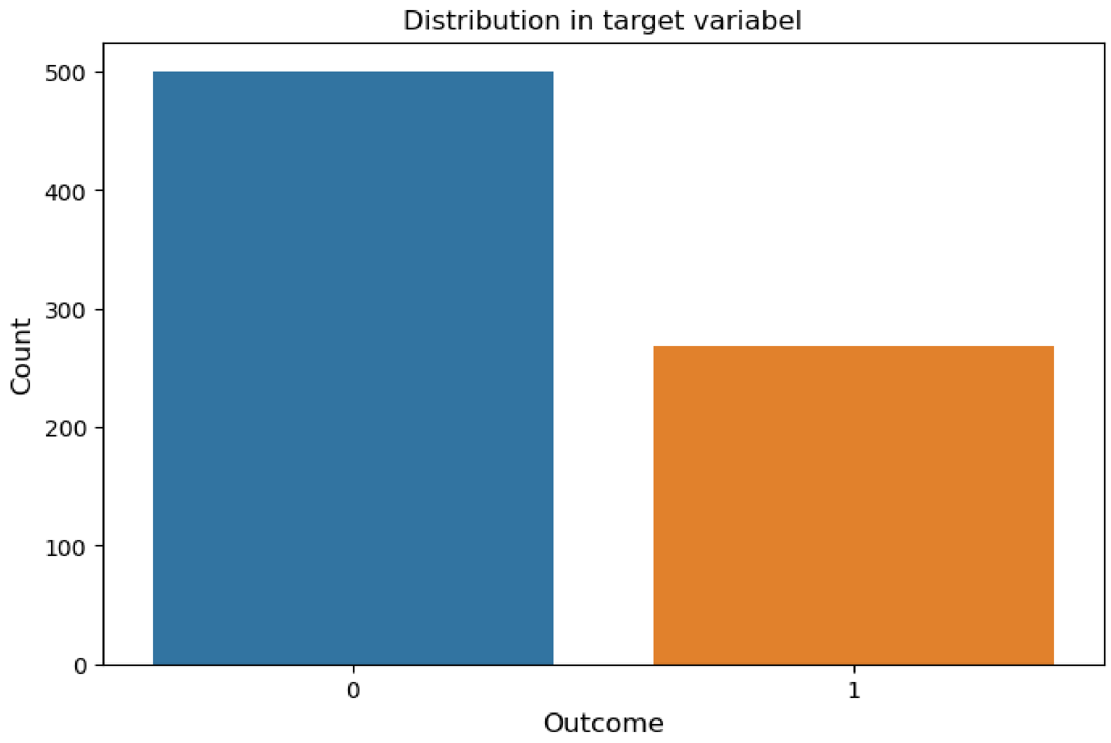
2. The range of values vaires over a large values. "Pregnancies" ranges from 0 to 17, "Glucose'

3. There is a imbalance in 'Outcome' column as the mean value shows only 34%. This shows that

4. Huge difference in mean and median of 'Insulin' and 'SkinThickness' shows there is a skewne

[2]:
```
plt.figure(figsize = (8,5))
sns.countplot(x = df['Outcome'])
plt.xlabel('Outcome', size = 12)
plt.ylabel('Count', size = 12)
plt.title('Distribution in target variabel', size = 12)
```

[2]:
```
Text(0.5, 1.0, 'Distribution in target variabel')
```



[3]:
```
print(df.dtypes)
```

```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
```

```
Age                         int64
Outcome                     int64
dtype: object
```

Observations:

1. All the data are in their proper type, no need for any changes.

[4]:
```python
print(df.head())
```

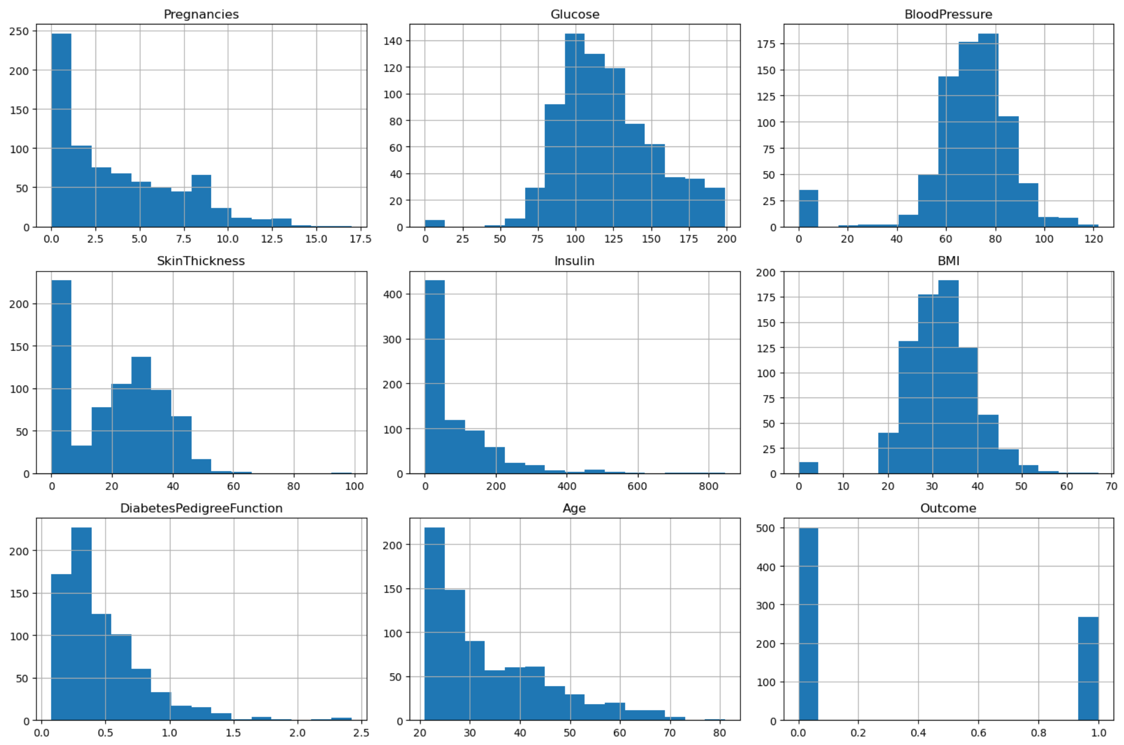
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
1. Univariate analysis focuses on examining the distribution and characteristics of a single va
    It helps in understanding the central tendency, spread, and shape of the distribution of ea

2. Pairwise analysis, on the other hand, explores the relationships between pairs of variables
    Pairwise analysis is particularly useful for identifying potential predictor variables or u
    It helps in identifying correlations, associations, or patterns between two variables.
```

[5]:
```python
# Univariate visualization - Histogram for each feature
df.hist(bins=15, figsize=(15, 10), layout=(3, 3))
plt.tight_layout()
plt.show()
```



Pregnancies: The distribution is heavily right-skewed, with most values concentrated towards 0, indicating that a large portion of the data has few or no pregnancies.

Glucose: The distribution appears to be roughly normal, with the majority of values clustered around the mean (potentially around 120).

BloodPressure: The distribution is slightly right-skewed, with a peak around 70-80 mm Hg, which is within the normal range for diastolic blood pressure.

SkinThickness: The distribution is heavily right-skewed, with a large number of zero or very low values, indicating that many individuals have minimal skin thickness measurements.

Insulin: The distribution is heavily right-skewed, with a large number of zero or very low values, suggesting that many individuals have low or no insulin levels measured.

BMI: The distribution appears to be roughly normal, with the majority of values falling within the overweight or obese range (25-35), potentially indicating a correlation between higher BMI and the presence of diabetes in the dataset.

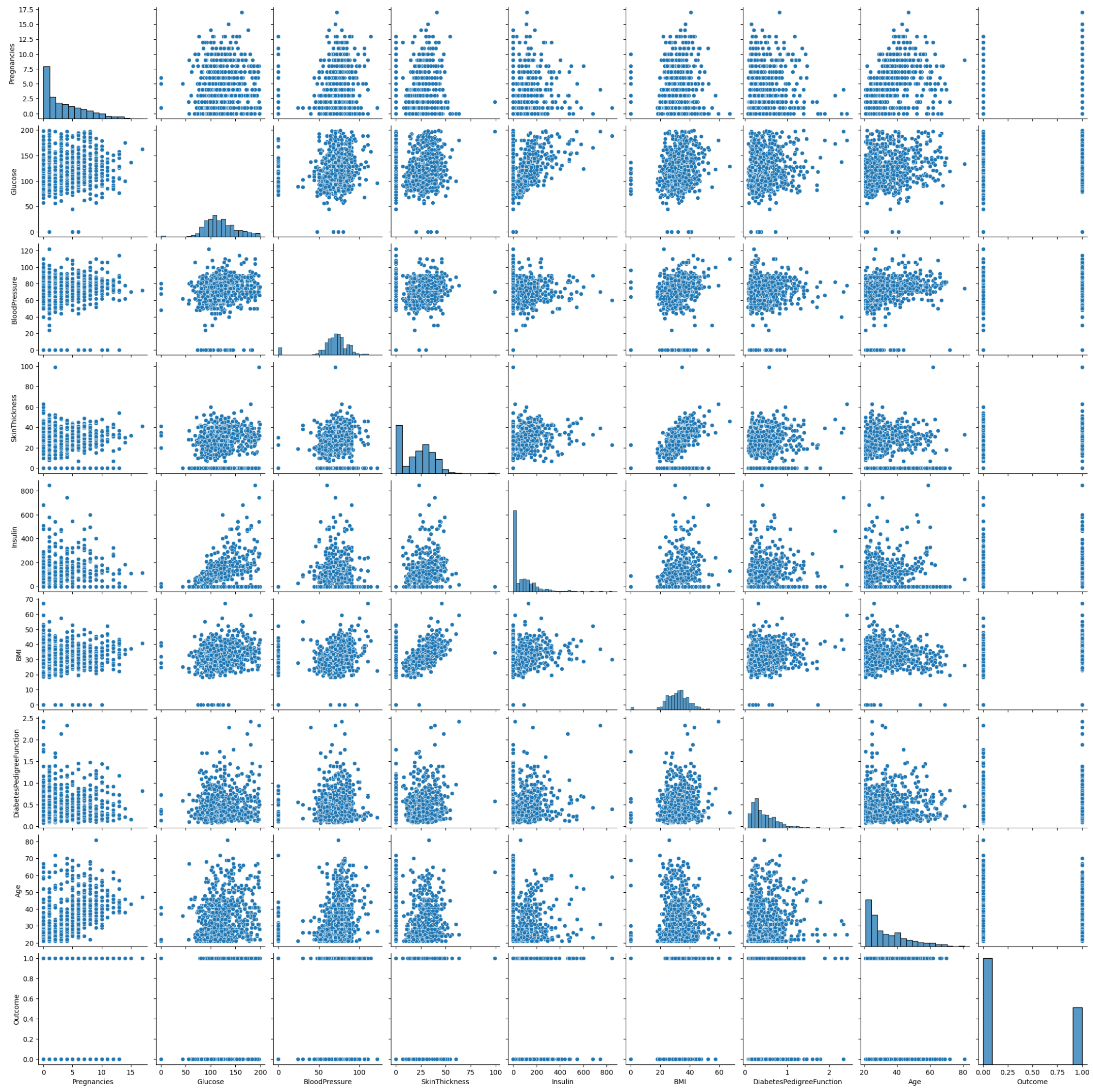
DiabetesPedigreeFunction: The distribution is heavily right-skewed, with most values concentrated towards 0, indicating that a large portion of the data has low diabetes pedigree function values (lower likelihood of diabetes based on family history).

Age: The distribution appears to be roughly normal, with a peak around 30-40 years of age, suggesting that the dataset covers a range of ages.

Outcome: The distribution is bimodal, with two distinct peaks representing the two classes (0 for no diabetes, 1 for diabetes). The smaller peak indicates that there are fewer instances of diabetes in the dataset compared to non-diabetes cases.

[6]:
```
# Pairwise visualization - Pairplot to observe relationships between features
sns.pairplot(df)
plt.show()
```

```
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\navee\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

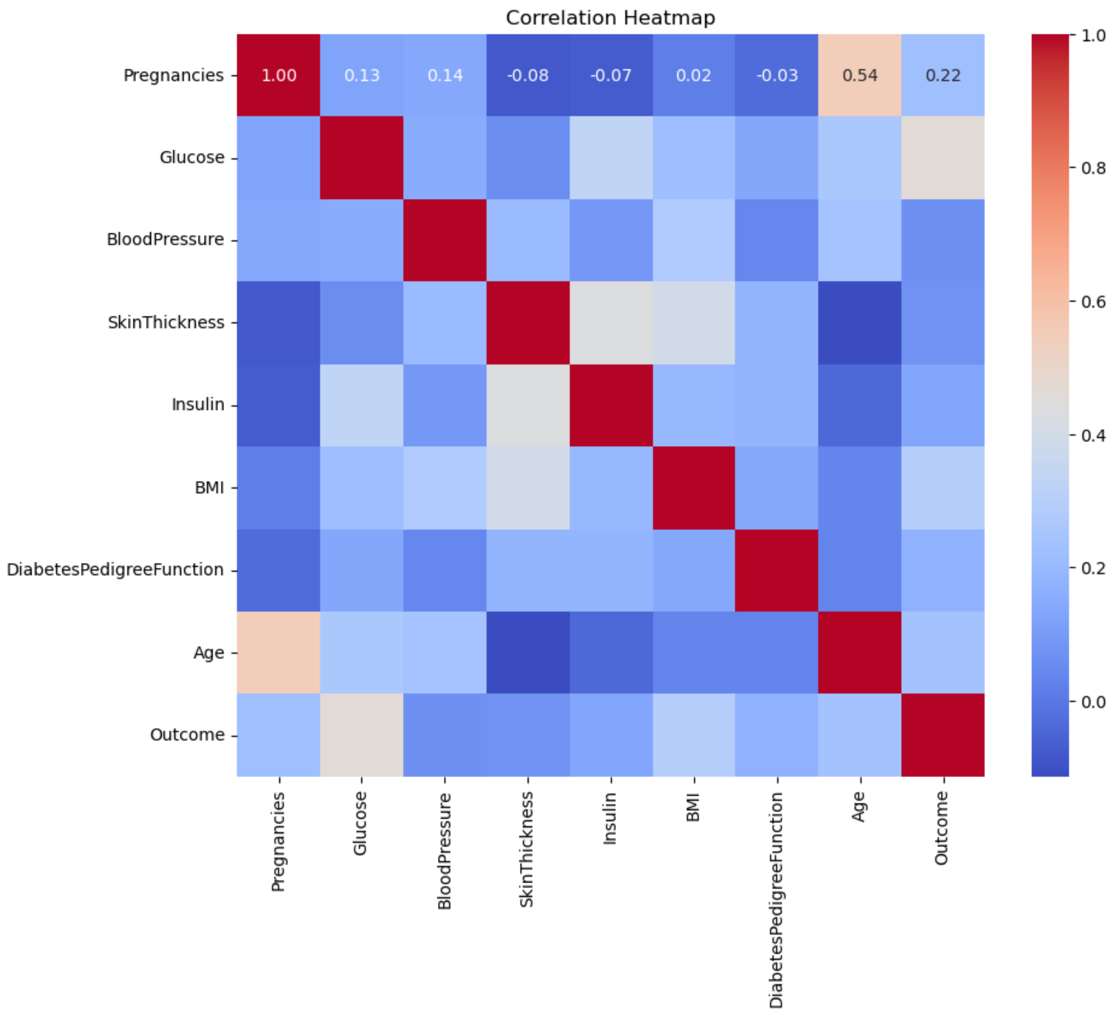


Observations:

```
1. From the pariplots, we can infer that there is positive correlation between Glucose and Insu
2. SkinThickness also seems to increase with an increase in BMI.
3. Blood Pressure does not seem to have any correltion with other features
4. The other plots does not seem to indicate any correlation
```

[7]:
```python
corr_matrix = df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



0.1 Data Cleaning

[8]:
```
# Check for initial zero values
zero_counts = (df == 0).sum()
print("Percentage of zero in each column:\n", zero_counts / len(df) * 100)

# Check for initial missing values
initial_missing_values = df.isnull().sum()
print("Initial missing values in each column:\n", initial_missing_values /␣
  ↪len(df) * 100)

# Replace zeroes with NaN in specified columns
columns_with_invalid_zeros = ['Glucose', 'BloodPressure', 'SkinThickness',␣
  ↪'Insulin', 'BMI']
```

```python
df[columns_with_invalid_zeros] = df[columns_with_invalid_zeros].replace(0, np.
 ↪nan)

# Show the count and percentage of missing values after replacement
missing_values_after_replacement = df.isnull().sum()
missing_values_percentage = (missing_values_after_replacement / len(df)) * 100

print("\nPercentage of missing values in each column after replacing zeroes␣
 ↪with NaN:\n", missing_values_percentage)
```

```
Percentage of zero in each column:
 Pregnancies                 14.453125
Glucose                      0.651042
BloodPressure                4.557292
SkinThickness               29.557292
Insulin                     48.697917
BMI                          1.432292
DiabetesPedigreeFunction     0.000000
Age                          0.000000
Outcome                     65.104167
dtype: float64
Initial missing values in each column:
 Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64

Percentage of missing values in each column after replacing zeroes with NaN:
 Pregnancies                  0.000000
Glucose                      0.651042
BloodPressure                4.557292
SkinThickness               29.557292
Insulin                     48.697917
BMI                          1.432292
DiabetesPedigreeFunction     0.000000
Age                          0.000000
Outcome                      0.000000
dtype: float64
```

Observation:

```
1. I will impute 'Glucose', BloodPressure', BMI' as they have low percentages of missing value
```

```
2. I can either impute or delete the 'SkinThickness' and 'Insulin' columns as they have high ar
3. I left out pregnancies and outcome column while replacing '0' with NAN because those columns
```

[9]:
```python
# Calculate the median for each column
medians = df.median()

# Fill missing values with the median of each column
df = df.fillna(medians)

# Calculate the percentage of missing values after imputation
missing_values_percentage = (df.isnull().sum() / len(df)) * 100
print("\nPercentage of missing values in each column after imputing:\n",␣
 ↪missing_values_percentage)
```

```
Percentage of missing values in each column after imputing:
 Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
SkinThickness               0.0
Insulin                     0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
```

0.2 Partition the data and prepare it.

e).

[10]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

# Partition the data into train/test sets
X = df.drop(columns='Outcome')
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Normalize numeric data
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Explaination

```
1. There are no categorical variables that needs to be encoded here. There are some columns tha
2. normalization has been done using standardScaler from sklearn.
    x = (x-mean(x)) / standard_deviation(x)
3. fit_transform() computes and transforms the training data, while transform() uses the previ
```

0.2.1 The documentation of the SVM algorithm to identify the hyperparametined.

[11]:
```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
import pickle

# Instantiate the SVM model
svc = SVC(random_state=42)

svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print('Accuracy here: ', svm.score(X_test, y_test))

# Define hyperparameters to tune
param_grid = {
    'C': [0.1, 1, 10, 100],  # Tuning best regularization parameter
    'kernel': ['linear','rbf', 'sigmoid']  # Kernel to be used. Using kernels␣
 ↪as they might capture non linear relation if it exists
}

# Use GridSearchCV to tune hyperparameters
grid = GridSearchCV(svc, param_grid, cv=6, scoring='precision', n_jobs=-1,␣
 ↪return_train_score=True)
grid.fit(X_train, y_train)

# Display the best model/parameters and the best score obtained
print("Best parameters found: ", grid.best_params_)
print("Best score of GridSearch: ", grid.best_score_)

# Testing the performance on test set
best_model = grid.best_estimator_
test_score = best_model.score(X_test, y_test)
print(f"Test set precision of GridSearch: {test_score:.3f}")


# Evaluate the model on the test set
y_pred = grid.predict(X_test)
print("Test Set Accuracy of the Model: ", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```python
# Extract results from GridSearchCV
results = grid.cv_results_

# Print mean train and test scores along with standard deviation for each␣
↪parameter combination
for mean_train_score, std_train_score, mean_test_score, std_test_score, params␣
↪in zip(
        results['mean_train_score'], results['std_train_score'],
        results['mean_test_score'], results['std_test_score'],
        results['params']):
    print(f"Params: {params}")
    print(f"Mean train score: {mean_train_score:.3f} (std: {std_train_score:.
↪3f})")
    print(f"Mean test score: {mean_test_score:.3f} (std: {std_test_score:.
↪3f})\n")


# Save the best model and grid search results
with open('best_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
with open('grid_search_results.pkl', 'wb') as f:
    pickle.dump(grid, f)
```

```
Accuracy here:  0.7402597402597403
Best parameters found:  {'C': 0.1, 'kernel': 'rbf'}
Best score of GridSearch:  0.7504563492063493
Test set precision of GridSearch: 0.734
Test Set Accuracy of the Model:  0.7337662337662337
              precision    recall  f1-score   support

           0       0.74      0.90      0.81       100
           1       0.70      0.43      0.53        54

    accuracy                           0.73       154
   macro avg       0.72      0.66      0.67       154
weighted avg       0.73      0.73      0.71       154

Params: {'C': 0.1, 'kernel': 'linear'}
Mean train score: 0.758 (std: 0.011)
Mean test score: 0.741 (std: 0.047)

Params: {'C': 0.1, 'kernel': 'rbf'}
Mean train score: 0.828 (std: 0.014)
Mean test score: 0.750 (std: 0.039)

Params: {'C': 0.1, 'kernel': 'sigmoid'}
```

```
Mean train score: 0.739 (std: 0.010)
Mean test score: 0.740 (std: 0.049)

Params: {'C': 1, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.010)
Mean test score: 0.738 (std: 0.036)

Params: {'C': 1, 'kernel': 'rbf'}
Mean train score: 0.841 (std: 0.015)
Mean test score: 0.723 (std: 0.058)

Params: {'C': 1, 'kernel': 'sigmoid'}
Mean train score: 0.573 (std: 0.021)
Mean test score: 0.618 (std: 0.050)

Params: {'C': 10, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.011)
Mean test score: 0.739 (std: 0.049)

Params: {'C': 10, 'kernel': 'rbf'}
Mean train score: 0.890 (std: 0.009)
Mean test score: 0.660 (std: 0.083)

Params: {'C': 10, 'kernel': 'sigmoid'}
Mean train score: 0.527 (std: 0.017)
Mean test score: 0.549 (std: 0.088)

Params: {'C': 100, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.011)
Mean test score: 0.739 (std: 0.049)

Params: {'C': 100, 'kernel': 'rbf'}
Mean train score: 0.973 (std: 0.007)
Mean test score: 0.567 (std: 0.029)

Params: {'C': 100, 'kernel': 'sigmoid'}
Mean train score: 0.538 (std: 0.029)
Mean test score: 0.523 (std: 0.090)
```

[12]:
```
import pickle
from sklearn.metrics import classification_report, accuracy_score

# Load the best model
with open('best_model.pkl', 'rb') as f:
    best_model = pickle.load(f)
```

```python
# Load the grid search results
with open('grid_search_results.pkl', 'rb') as f:
    grid = pickle.load(f)

# Testing the performance on test set
test_score = best_model.score(X_test, y_test)
print(f"Test set precision of GridSearch: {test_score:.3f}")

# Evaluate the model on the test set
y_pred = grid.predict(X_test)
print("Test Set Accuracy of the Model: ", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Extract results from GridSearchCV
results = grid.cv_results_

# Print mean train and test scores along with standard deviation for each␣
 ↪parameter combination
for mean_train_score, std_train_score, mean_test_score, std_test_score, params␣
 ↪in zip(
        results['mean_train_score'], results['std_train_score'],
        results['mean_test_score'], results['std_test_score'],
        results['params']):
    print(f"Params: {params}")
    print(f"Mean train score: {mean_train_score:.3f} (std: {std_train_score:.
 ↪3f})")
    print(f"Mean test score: {mean_test_score:.3f} (std: {std_test_score:.
 ↪3f})\n")
```

```
Test set precision of GridSearch: 0.734
Test Set Accuracy of the Model:  0.7337662337662337
              precision    recall  f1-score   support

           0       0.74      0.90      0.81       100
           1       0.70      0.43      0.53        54

    accuracy                           0.73       154
   macro avg       0.72      0.66      0.67       154
weighted avg       0.73      0.73      0.71       154

Params: {'C': 0.1, 'kernel': 'linear'}
Mean train score: 0.758 (std: 0.011)
Mean test score: 0.741 (std: 0.047)

Params: {'C': 0.1, 'kernel': 'rbf'}
Mean train score: 0.828 (std: 0.014)
Mean test score: 0.750 (std: 0.039)
```

```
Params: {'C': 0.1, 'kernel': 'sigmoid'}
Mean train score: 0.739 (std: 0.010)
Mean test score: 0.740 (std: 0.049)

Params: {'C': 1, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.010)
Mean test score: 0.738 (std: 0.036)

Params: {'C': 1, 'kernel': 'rbf'}
Mean train score: 0.841 (std: 0.015)
Mean test score: 0.723 (std: 0.058)

Params: {'C': 1, 'kernel': 'sigmoid'}
Mean train score: 0.573 (std: 0.021)
Mean test score: 0.618 (std: 0.050)

Params: {'C': 10, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.011)
Mean test score: 0.739 (std: 0.049)

Params: {'C': 10, 'kernel': 'rbf'}
Mean train score: 0.890 (std: 0.009)
Mean test score: 0.660 (std: 0.083)

Params: {'C': 10, 'kernel': 'sigmoid'}
Mean train score: 0.527 (std: 0.017)
Mean test score: 0.549 (std: 0.088)

Params: {'C': 100, 'kernel': 'linear'}
Mean train score: 0.756 (std: 0.011)
Mean test score: 0.739 (std: 0.049)

Params: {'C': 100, 'kernel': 'rbf'}
Mean train score: 0.973 (std: 0.007)
Mean test score: 0.567 (std: 0.029)

Params: {'C': 100, 'kernel': 'sigmoid'}
Mean train score: 0.538 (std: 0.029)
Mean test score: 0.523 (std: 0.090)
```

Reference:

```
1. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
```

Observations:

```
1. When GridSearchCV is used it automatically uses cross validation, in here cv = 5 means 5 fl
```

2.Out of all actual class 0 instances, 90% were correctly identified as 'No diabetes'.
3. Out of all actual instances of "Diabetes," only 43% were correctly identified by the model a
4. Out of all instances predicted as "No Diabetes," 74% were actually "No Diabetes."
5. Out of all instances predicted as "Diabetes," 70% were actually "Diabetes."

Bais and Variance

1. The model may be overfitting as the training accuracy is high and the test accuracy has drop
2. The model achieved an accracy of 75% in the training. The 75% suggests that while the predi
3. The model achieved an accuracy of 73 % in the test set meaning that whatever the model has l

Model

1. Given its performance on training and test set, the classifier seems to be doing a decent jo
2. As we only have an accuracy of 73%, the model can help medical industry as subordinate in as

[ ]: