Homework3

March 25, 2024

1 Homework 3

25 March, 2024

1.1 Section 1 - Loading Dataset

[19]:
```
#1.1. Link: https://www.kaggle.com/datasets/akshaydattatraykhare/
 ↪diabetes-dataset

#1.2: Description: The objective of the dataset is to diagnostically predict␣
 ↪whether a patient has diabetes, based on certain diagnostic measurements␣
 ↪included in the dataset.

#1.3: Fields/Attributes/Predictors:
#Pregnancies: To express the Number of pregnancies
#Glucose: To express the Glucose level in blood
#BloodPressure: To express the Blood pressure measurement
#SkinThickness: To express the thickness of the skin
#Insulin: To express the Insulin level in blood
#BMI: To express the Body mass index
#DiabetesPedigreeFunction: To express the Diabetes percentage
#Age: To express the age
#Outcome: To express the final result 1 is Yes and 0 is No

#1.4: Import Libraries:

import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
1
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

```python
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures

#1.5: Load dataset into a Pandas dataframe
data = pd.read_csv('diabetes.csv')
```

[21]:
```python
#1.6: Display the first 10 rows
data.head(10)
```

[21]:
```
   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
5            5      116             74              0        0  25.6
6            3       78             50             32       88  31.0
7           10      115              0              0        0  35.3
8            2      197             70             45      543  30.5
9            8      125             96              0        0   0.0

   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
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
9                     0.232   54        1
```

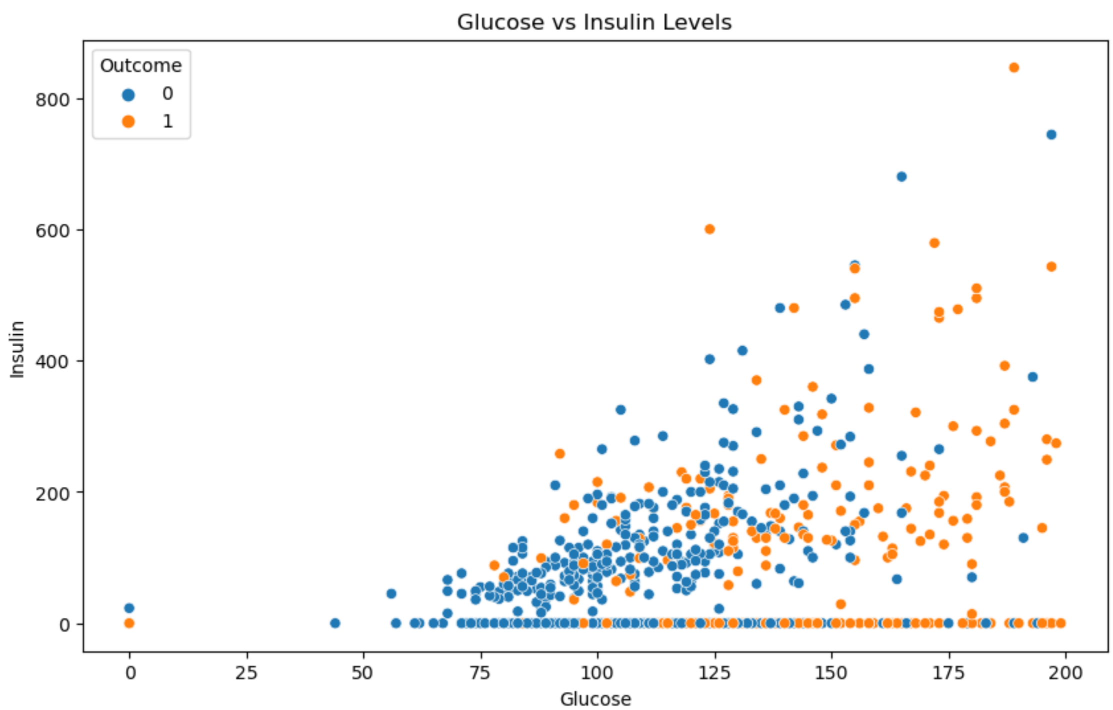
[23]:
```python
#1.7: Show the dataframe information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

1.2 Section 2 - Logistic Regression

[31]:
```
#2.1: Prepare data for logistic regression
data = data.dropna(subset=['Outcome'])
X = data.loc[:, data.columns != "Outcome"]
X = sm.add_constant(X)
y = data['Outcome']

#2.2: Visualizing the data
# Scatter plot with color coded output
plt.figure(figsize=(10,6))
sns.scatterplot(x='Glucose', y='Insulin', hue='Outcome', data=data)
plt.title('Glucose vs Insulin Levels')
plt.show()
```



[33]:
```
#2.3: Build the logistic regression model
model = sm.Logit(y,X).fit()
```

```python
#2.4: Display model summary
model.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.470993
         Iterations 6
```

[33]:

Dep. Variable:	Outcome	**No. Observations:**	768
Model:	Logit	**Df Residuals:**	759
Method:	MLE	**Df Model:**	8
Date:	Mon, 25 Mar 2024	**Pseudo R-squ.:**	0.2718
Time:	21:14:30	**Log-Likelihood:**	-361.72
converged:	True	**LL-Null:**	-496.74
Covariance Type:	nonrobust	**LLR p-value:**	9.652e-54

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	-8.4047	0.717	-11.728	0.000	-9.809	-7.000
Pregnancies	0.1232	0.032	3.840	0.000	0.060	0.186
Glucose	0.0352	0.004	9.481	0.000	0.028	0.042
BloodPressure	-0.0133	0.005	-2.540	0.011	-0.024	-0.003
SkinThickness	0.0006	0.007	0.090	0.929	-0.013	0.014
Insulin	-0.0012	0.001	-1.322	0.186	-0.003	0.001
BMI	0.0897	0.015	5.945	0.000	0.060	0.119
DiabetesPedigreeFunction	0.9452	0.299	3.160	0.002	0.359	1.531
Age	0.0149	0.009	1.593	0.111	-0.003	0.033

[39]:
```python
#Locate an interesting datapoint
y_filtered = y[(data['BMI'] > 30) & (data['Pregnancies'] > 10)]
y_filtered
data.iloc[[518]]
```

[39]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
518           13       76             60              0        0  32.8

     DiabetesPedigreeFunction  Age  Outcome
518                      0.18   41        0
```

[46]:
```python
#2.5 Prediction

# Select a row and convert it into a Dataframe with one row
test = X.iloc[[518]].copy()

# Modify for this instance
test['Glucose'] = 100
test['BloodPressure'] = 90

# Make a prediction using the adjusted test DataFrame
prediction = model.predict(test)
```

```
percentage_probability = prediction.iloc[0] * 100
binary_predictions = (prediction >= 0.5).astype(int)
class_prediction = binary_predictions.iloc[0]

# Print the results
print(f"Probability of default: {percentage_probability:.2f}%")
print(f"Class: {class_prediction}")
test
```

```
Probability of default: 31.83%
Class: 0
```

[46]:

```
     const  Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
518    1.0           13      100             90              0        0  32.8

     DiabetesPedigreeFunction  Age
518                      0.18   41
```

1.3 Section 3 - LDA - Linear Discriminant Analysis

[68]:

```
# Data setup
data['pregnancy_class'] = 0
data.loc[(data['Pregnancies'] > 0) & (data['Pregnancies'] < 8),␣
 ↪'pregnancy_class'] = 1
data.loc[data['Pregnancies'] > 8, 'pregnancy_class'] = 2
data['pregnancy_class'].value_counts()
```

[68]:

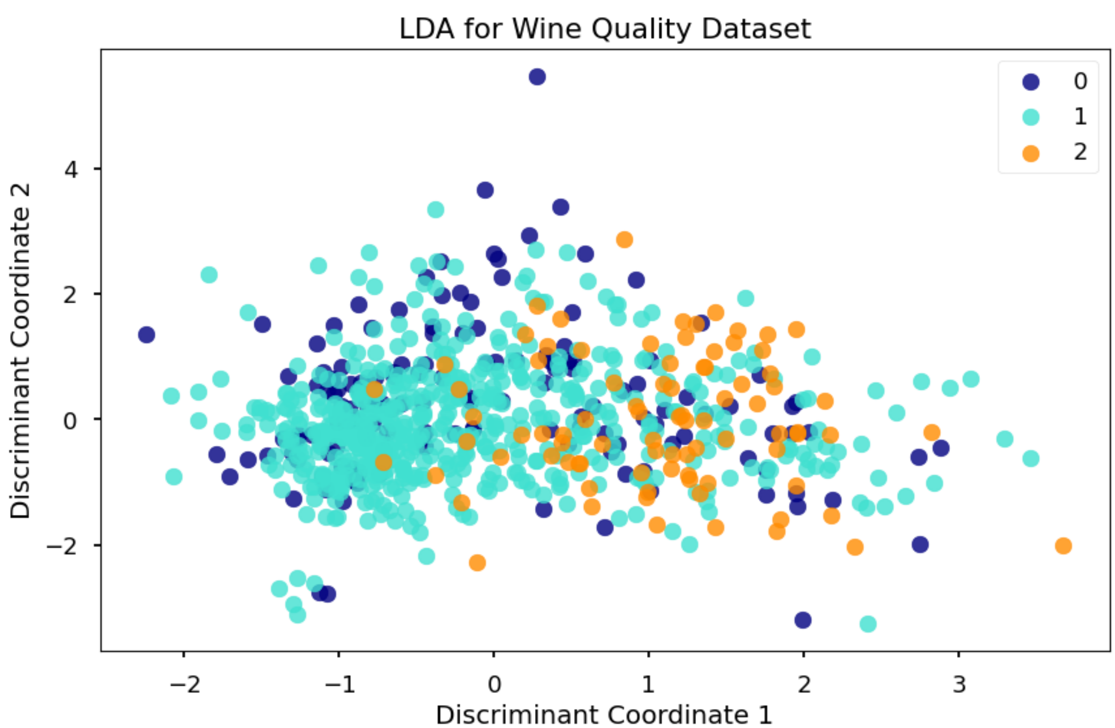
```
pregnancy_class
1    533
0    149
2     86
Name: count, dtype: int64
```

[72]:

```
X = data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',␣
 ↪'DiabetesPedigreeFunction', 'Age', 'Outcome']]
X = sm.add_constant(X)
y = data['pregnancy_class']

# Fit the LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)

# LDA Variance ratio

print(lda.explained_variance_ratio_)

# Transform the data using the fitted LDA in the new space
```

```
X_r_lda = lda.transform(X)
target_names = ['0','1','2']

# Set the style and create the figure

with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(figsize=[10,6])
    colors = ['navy', 'turquoise', 'darkorange']
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):
        ax.scatter(X_r_lda[y == i, 0], X_r_lda[y == i, 1], alpha=.
 ↪8,label=target_name, color=color)
        ax.set_title('LDA for Wine Quality Dataset')
        ax.set_xlabel('Discriminant Coordinate 1')
        ax.set_ylabel('Discriminant Coordinate 2')
        ax.legend(loc='best')
plt.show()
```

```
[0.87716609 0.12283391]
```



[74]:
```
## Section 4 - K-fold Cross-validation

X = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Outcome']]
```

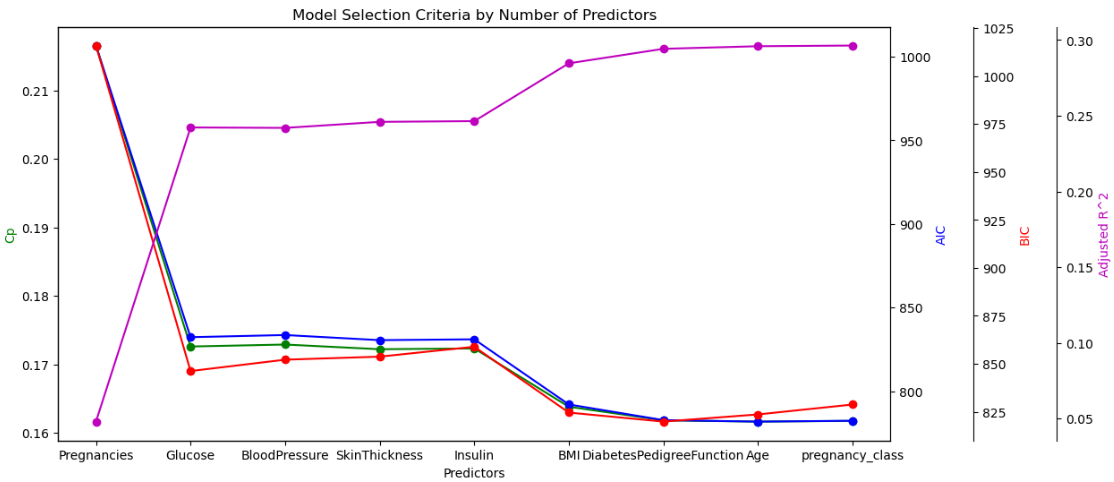
```
y = data['Age']

# Initialize Linear Regression model
lr = LinearRegression()

# Define a 5-fold cross-validation split
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_scores = []

# Manually loop through each fold
for train_index, test_index in kf.split(X):
    # Split data into train and test based on current fold
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train the model on the current fold
    lr.fit(X_train, y_train)

    # Make predictions
    y_pred = lr.predict(X_test)

    # Calculate MSE and store
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)

    print(f"MSE for fold {len(mse_scores)}: {mse:.4f}")

# Print results
print(f"Mean MSE from 5-fold CV using Linear Regression: {np.mean(mse_scores):.
 ↪4f}")
print(f"Standard deviation of MSE: {np.std(mse_scores):.4f}")
```

```
MSE for fold 1: 118.0754
MSE for fold 2: 83.3304
MSE for fold 3: 79.1842
MSE for fold 4: 94.1781
MSE for fold 5: 76.1809
Mean MSE from 5-fold CV using Linear Regression: 90.1898
Standard deviation of MSE: 15.2180
```

1.4 Section 5 - Linear Model Selection Statistics

[78]:
```
# Setup data
X = data.loc[:, data.columns != "Outcome"]
y = data['Outcome']

# Define function to compute Cp
```

```python
def compute_cp(model, X, y):
    mse = np.mean((model.predict(X) - y) ** 2)
    p = len(model.params) - 1 # Number of predictors
    n = len(y)
    cp = mse + 2 * p * mse / (n - p - 1)
    return cp

# Compute metrics for models with increasing numbers of predictors
predictors = X.columns
cp_values, aic_values, bic_values, adjr2_values = [], [], [], []

for k in range(1, len(predictors) + 1):
    chosen_predictors = predictors[:k]
    X_subset = X[chosen_predictors]
    X_subset = sm.add_constant(X_subset) # Add constant for intercept
    model = sm.OLS(y, X_subset).fit()
    cp_values.append(compute_cp(model, X_subset, y))
    aic_values.append(model.aic)
    bic_values.append(model.bic)
    adjr2_values.append(model.rsquared_adj)

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))
ax2 = ax1.twinx()
ax3 = ax1.twinx()
ax4 = ax1.twinx()

# Adjust the position of the third axis to be offset on the right
ax3.spines.right.set_position(("axes", 1.1))
ax4.spines.right.set_position(("axes", 1.2))

# Plot metrics on each appropriate axis
ax1.plot(predictors, cp_values, 'g-', label="Cp", marker='o')
ax2.plot(predictors, aic_values, 'b-', label="AIC", marker='o')
ax3.plot(predictors, bic_values, 'r-', label="BIC", marker='o')
ax4.plot(predictors, adjr2_values, 'm-', label="Adj R^2", marker='o')

# Set labels and title
ax1.set_xlabel('Predictors')
ax1.set_ylabel('Cp', color='g')
ax2.set_ylabel('AIC', color='b')
ax3.set_ylabel('BIC', color='r')
ax4.set_ylabel('Adjusted R^2', color='m')
plt.title('Model Selection Criteria by Number of Predictors')

# Display
plt.show()
```



1.5 Section 6 - Simple Polynomial Regression

[83]:
```
# Setup data
X = data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Outcome']]
y = data['Age']

# Split the dataset into training and testing sets for validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
↪2,random_state=42)

# 10th-degree polynomial regression
poly10_model = make_pipeline(PolynomialFeatures(10), LinearRegression())
poly10_model.fit(X_train, y_train)
y_pred_poly10 = poly10_model.predict(X_test)
mse_poly10 = mean_squared_error(y_test, y_pred_poly10)

# 4th-degree polynomial regression
poly4_model = make_pipeline(PolynomialFeatures(4), LinearRegression())
poly4_model.fit(X_train, y_train)
y_pred_poly4 = poly4_model.predict(X_test)
mse_poly4 = mean_squared_error(y_test, y_pred_poly4)

# 3rd-degree polynomial regression
poly3_model = make_pipeline(PolynomialFeatures(3), LinearRegression())
poly3_model.fit(X_train, y_train)
y_pred_poly3 = poly3_model.predict(X_test)
mse_poly3 = mean_squared_error(y_test, y_pred_poly3)

print(f"Mean Squared Error for 10th-degree Polynomial: {mse_poly10:.2f}")
```

```
print(f"Mean Squared Error for 4th-degree Polynomial: {mse_poly4:.2f}")
print(f"Mean Squared Error for 3rd-degree Polynomial: {mse_poly3:.2f}")
```

```
Mean Squared Error for 10th-degree Polynomial: 7454754185673.88
Mean Squared Error for 4th-degree Polynomial: 58484.55
Mean Squared Error for 3rd-degree Polynomial: 273.51
```

1.6 Section 7 - Discussion Forum

When completing Homework 3, there were a number of findings when analyzing the dataset I selected regarding Diabetes and its predictors. Moreover, some of the models and tools used during the homework highlighted these findings.

1. I had a hard time trying to break the data into classes for Linear Discriminant Analysis. For just about every predictor, there were either too many unique values to break it down into classes or there attribute itself didn't really sub-categorize well. I selected the number of pregnancies: the first class was 0, the second was between 1-7, and the third was over 7. 2. Using Linear Model Selection, I found that the "sweet spot" for number of predictors was 6. This included number of pregnancies, bmi, glucose level, blood pressure level, skin thickness, and insulin level. 3. Using Simple Polynomial Regression, I found that a 3rd-degree polynomial best fit the data with a Mean Squared Error (MSE) of 273.51

[ ]:

```

```