Mechine Learning第一次实验报告

58122204 谢兴

1. **数据集预处理**
2. **离散属性连续化**

**为了将离散属性转换为连续属性，我们对数据集中的分类特征进行了One-Hot编码处理。One-Hot编码是一种常见的数据预处理方式，它可以将分类数据转换成机器学习算法易于处理的形式。在我们的数据集中，time\_signature 和 key 这两个分类特征经过One-Hot编码后转变为多个二进制列，使得线性模型可以使用这些属性。**

27. categorical\_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle\_unknown='ignore'))])

1. **连续属性归一化normalization**

**通过MinMaxScaler实现，该过程保持了数据的稀疏结构，同时减少了不同属性值范围的差异。**

26. numeric\_transformer = Pipeline(steps=[('scaler', MinMaxScaler())])

1. **共线性的检测与处理**

**在模型训练之前，我们进行了共线性的检测，这是通过计算变量膨胀因子（VIF）来实现的。VIF值提供了一个指标，表明一个特征与其他特征的线性依赖程度。一个高VIF值（通常大于5）表明高共线性，可能会削弱模型的性能。在我们的模型中，当检测到高VIF值时，我们选择应用主成分分析（PCA）来降维，减少特征之间的多重共线性。**

51. # Function to calculate VIF scores

52. def calculate\_vif(X):

53. vif\_data = pd.DataFrame()

54. vif\_data["Feature"] = X.columns

55. vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

56. return vif\_data

57.

58. vif\_scores = calculate\_vif(X\_train\_preprocessed\_df)

59.

60. # Check VIF and conditionally add PCA to the pipeline

61. if np.any(vif\_scores["VIF"] > 5):

62. print("Applying PCA due to multicollinearity")

63. pipeline.steps.insert(-1, ('pca', PCA(n\_components=0.95)))

64.

65. # Fit the model pipeline, possibly with PCA

66. pipeline.fit(X\_train, y\_train)

1. **实验设置**
2. **实验评估方法**

单次留出法：用Scikit-Learn中train\_test\_split将song\_data.csv中随机划分80%为训练集，20%为测试集。

45. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

1. **性能度量**

**用**Scikit-Learn中mean\_squared\_error函数来计算均方误差（MSE）；

用r2\_score来计算模型在训练集和测试集上的决定系数。

77. # Predictions and model evaluation

78. y\_pred\_train = pipeline.predict(X\_train)

79. y\_pred\_test = pipeline.predict(X\_test)

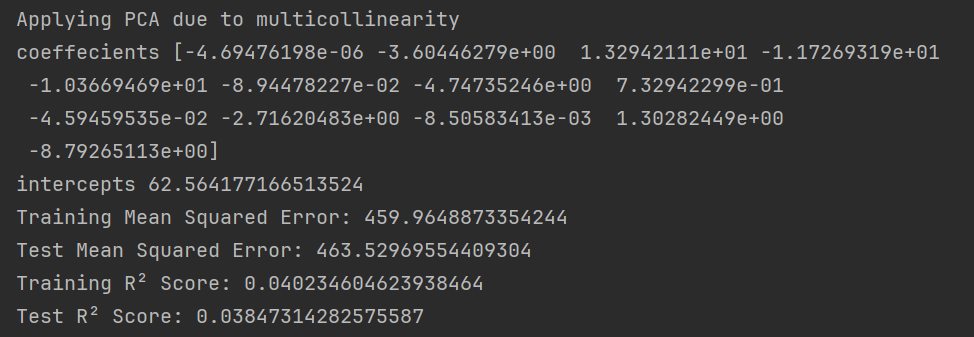
80. train\_error = mean\_squared\_error(y\_train, y\_pred\_train)

81. test\_error = mean\_squared\_error(y\_test, y\_pred\_test)

82. train\_r2 = r2\_score(y\_train, y\_pred\_train)

83. test\_r2 = r2\_score(y\_test, y\_pred\_test)

1. **实验结果**
2. **模型展示**

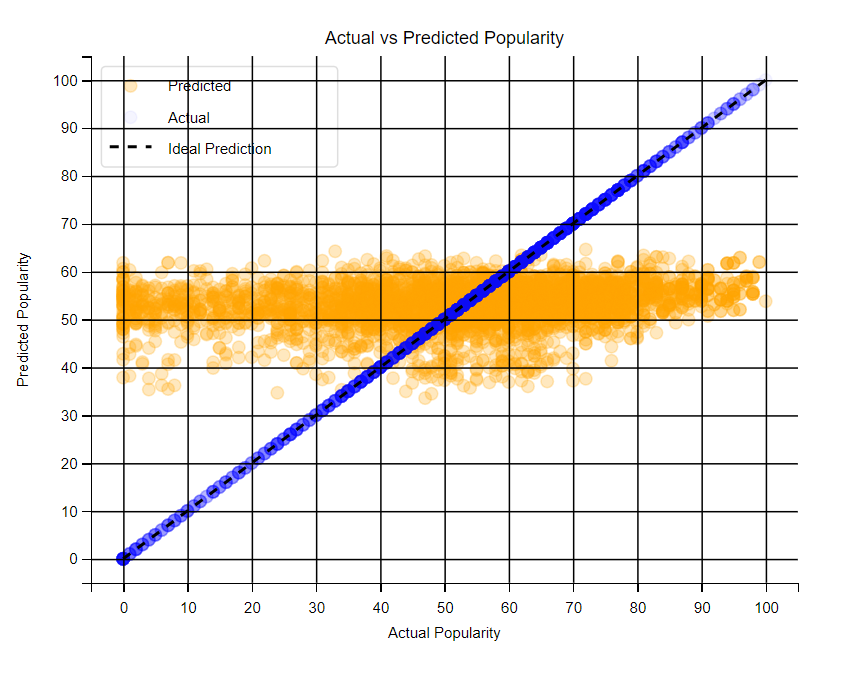
****

1. **可视化结果展示**

**（1）对数据集的展示**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**（2）对模型的展示**

****

1. **结果分析**

**均方误差 (MSE)**

**本实验中，训练集和测试集的MSE值均较高，表明模型预测值与实际值之间存在较大差异，预测准确性较低。**

**决定系数 (R² Score)**

**训练集和测试集上的R²值均低于0.05，表明模型几乎无法解释目标变量的变异性，表现较差。**

**结论**

**模型的预测准确性和解释能力均不理想，需要进一步改进。**

1. **代码附录**

1. import pandas as pd

2. import numpy as np

3. from sklearn.model\_selection import train\_test\_split

4. from sklearn.linear\_model import LinearRegression

5. from sklearn.metrics import mean\_squared\_error, r2\_score

6. from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

7. from sklearn.compose import ColumnTransformer

8. from sklearn.pipeline import Pipeline

9. import matplotlib.pyplot as plt

10. from sklearn.decomposition import PCA

11. import seaborn as sns

12. from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

13. import os

14.

15. # Load the dataset

16. data = pd.read\_csv('song\_data.csv')

17.

18. # Identifying categorical and numerical columns (excluding 'song\_popularity' from numerical)

19. categorical\_cols = data.select\_dtypes(include=['object', 'category']).columns.drop('song\_name')

20. numerical\_cols = data.select\_dtypes(include=[np.number]).columns.drop('song\_popularity')

21.

22. # Dropping 'song\_name' as it's irrelevant for modeling

23. data = data.drop('song\_name', axis=1)

24.

25. # Define transformers for numerical and categorical data

26. numeric\_transformer = Pipeline(steps=[('scaler', MinMaxScaler())])

27. categorical\_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle\_unknown='ignore'))])

28.

29. # Combine transformers into a ColumnTransformer

30. preprocessor = ColumnTransformer(transformers=[

31. ('num', numeric\_transformer, numerical\_cols),

32. ('cat', categorical\_transformer, categorical\_cols)

33. ])

34.

35. # Define the model pipeline

36. pipeline = Pipeline(steps=[

37. ('preprocessor', preprocessor),

38. # Initially, do not include PCA in the pipeline; it may be added conditionally

39. ('linear\_regression', LinearRegression())

40. ])

41.

42. # Splitting the dataset into training and testing sets

43. X = data.drop('song\_popularity', axis=1)

44. y = data['song\_popularity']

45. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

46.

47. # Fit the preprocessing to compute VIF on the training set

48. X\_train\_preprocessed = preprocessor.fit\_transform(X\_train)

49. X\_train\_preprocessed\_df = pd.DataFrame(X\_train\_preprocessed, columns=[f"feature\_{i}" for i in range(X\_train\_preprocessed.shape[1])])

50.

51. # Function to calculate VIF scores

52. def calculate\_vif(X):

53. vif\_data = pd.DataFrame()

54. vif\_data["Feature"] = X.columns

55. vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

56. return vif\_data

57.

58. vif\_scores = calculate\_vif(X\_train\_preprocessed\_df)

59.

60. # Check VIF and conditionally add PCA to the pipeline

61. if np.any(vif\_scores["VIF"] > 5):

62. print("Applying PCA due to multicollinearity")

63. pipeline.steps.insert(-1, ('pca', PCA(n\_components=0.95)))

64.

65. # Fit the model pipeline, possibly with PCA

66. pipeline.fit(X\_train, y\_train)

67.

68.

69. LR = LinearRegression().fit(X\_train, y\_train)

70. train\_pred = LR.predict(X\_train)

71. test\_pred = LR.predict(X\_test)

72.

73. #Print Models paremeter

74. print('coeffecients', LR.coef\_)

75. print('intercepts', LR.intercept\_)

76.

77. # Predictions and model evaluation

78. y\_pred\_train = pipeline.predict(X\_train)

79. y\_pred\_test = pipeline.predict(X\_test)

80. train\_error = mean\_squared\_error(y\_train, y\_pred\_train)

81. test\_error = mean\_squared\_error(y\_test, y\_pred\_test)

82. train\_r2 = r2\_score(y\_train, y\_pred\_train)

83. test\_r2 = r2\_score(y\_test, y\_pred\_test)

84.

85. # Print metrics

86. print(f'Training Mean Squared Error: {train\_error}')

87. print(f'Test Mean Squared Error: {test\_error}')

88. print(f'Training R² Score: {train\_r2}')

89. print(f'Test R² Score: {test\_r2}')

90.

91.

92. # Visualizing Actual vs Predicted Popularity for test set

93.

94. plt.figure(figsize=(10, 6)) # Set the size of the figure

95. plt.scatter(y\_test, y\_pred\_test, alpha=0.5, label='Predicted', color='orange') # Increase alpha for more solid color

96. plt.scatter(y\_test, y\_test, alpha=0.2, label='Actual', color='blue') # Plot the actual values for comparison

97.

98. # Ideal prediction line

99. plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2, label='Ideal Prediction')

100.

101. # Axes and title

102. plt.xlabel('Actual Popularity')

103. plt.ylabel('Predicted Popularity')

104. plt.title('Actual vs Predicted Popularity')

105.

106. # Legend

107. plt.legend(loc='upper left') # Position the legend in the upper left corner

108.

109. plt.grid(True)

110.

111. # Show the plot

112. plt.show()

113.

114. # Additional visualization: PCA Component Analysis (Cumulative Explained Variance)

115. pca = PCA().fit(preprocessor.transform(X\_train))

116. plt.figure(figsize=(10, 6))

117. plt.plot(np.cumsum(pca.explained\_variance\_ratio\_), marker='o')

118. plt.xlabel('Number of Components')

119. plt.ylabel('Cumulative Explained Variance')

120. plt.title('Explained Variance by Components')

121. plt.axhline(y=0.95, color='r', linestyle='--')

122. plt.text(0.5, 0.9, '95% explained variance', color='red')

123. plt.show()

124.

125.

126. continuous\_features = [col for col in X.columns if X[col].dtype in ['float64', 'int64']]

127.

128. # Function to save plots

129. def save\_plots(data, X\_train, y\_train, model, continuous\_features, save\_path):

130. for feature in continuous\_features:

131. # Distribution

132. plt.figure(figsize=(10, 4))

133. sns.histplot(data[feature], kde=True)

134. plt.title(f'Distribution of {feature}')

135. plt.xlabel(feature)

136. plt.ylabel('Frequency')

137. distribution\_path = os.path.join(save\_path, f'distribution\_{feature}.png')

138. plt.savefig(distribution\_path)

139. plt.close()

140.

141. # Correlation with target

142. plt.figure(figsize=(10, 4))

143. sns.scatterplot(x=data[feature], y=data['song\_popularity'])

144. plt.title(f'{feature} vs. Song Popularity')

145. plt.xlabel(feature)

146. plt.ylabel('Song Popularity')

147. correlation\_path = os.path.join(save\_path, f'correlation\_{feature}.png')

148. plt.savefig(correlation\_path)

149. plt.close()

150.

151. # Model visualization for a specific feature

152. feature\_name = continuous\_features[0]

153. feature\_index = X\_train.columns.get\_loc(feature\_name)

154.

155. # Training set scatter plot and regression line

156. plt.figure(figsize=(10, 6))

157. plt.scatter(X\_train[feature\_name], y\_train, color='blue', label='Actual', alpha=0.5)

158. predictions = model.predict(X\_train)

159. plt.scatter(X\_train[feature\_name], predictions, color='red', label='Predicted', alpha=0.5)

160. sorted\_idx = np.argsort(X\_train[feature\_name])

161. plt.plot(X\_train[feature\_name].iloc[sorted\_idx], predictions[sorted\_idx], color='green', label='Regression Line')

162. plt.title(f'Model Predictions vs. Actual for {feature\_name}')

163. plt.xlabel(feature\_name)

164. plt.ylabel('Song Popularity')

165. plt.legend()

166. model\_prediction\_path = os.path.join(save\_path, f'model\_prediction\_{feature\_name}.png')

167. plt.savefig(model\_prediction\_path)

168. plt.close()

169.

170. # save the visiual results

171. save\_plots(data, X\_train, y\_train, pipeline, continuous\_features, "visilual\_results")

172.