Stage_3

November 6, 2021

1 Stage 3: Machine Learning

SID: 500497375

[1]:
```
#Import libraries to undertake machine learning task.
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
from math import sqrt
from sklearn import metrics, neighbors
from sklearn.model_selection import train_test_split
```

Using a k-nearest neighbors regression machine learning model and the dataset used in stage 2, we aim to use Universal health coverage (UHC) service coverage index metrics to predict the mortality rate attributed to cardiovascular disease, cancer, diabetes or chronic respiratory disease.

[2]:
```
df = pd.read_csv('S3DS.csv')
#Slice dataframe for predictive model
X = df[['Universal health coverage (UHC) service coverage index']].values
y = df['Mortality rate attributed to cardiovascular disease, cancer, diabetes␣
 ↪or chronic respiratory disease (probability):::BOTHSEX'].values
#Split data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
neighbour_for_test = neighbors.KNeighborsRegressor(n_neighbors = 4).
 ↪fit(X_train, y_train)
```

1.1 Predictions

[3]:
```
#Create sample data for sample prediction
sample = [92]
sample_prediction = neighbour_for_test.predict([sample])
print('-----Sample test case-----')
print('Universal health coverage: ', sample[0])
print('Predicted mortality rate: ', int(sample_prediction))
print('--------------------------')
```

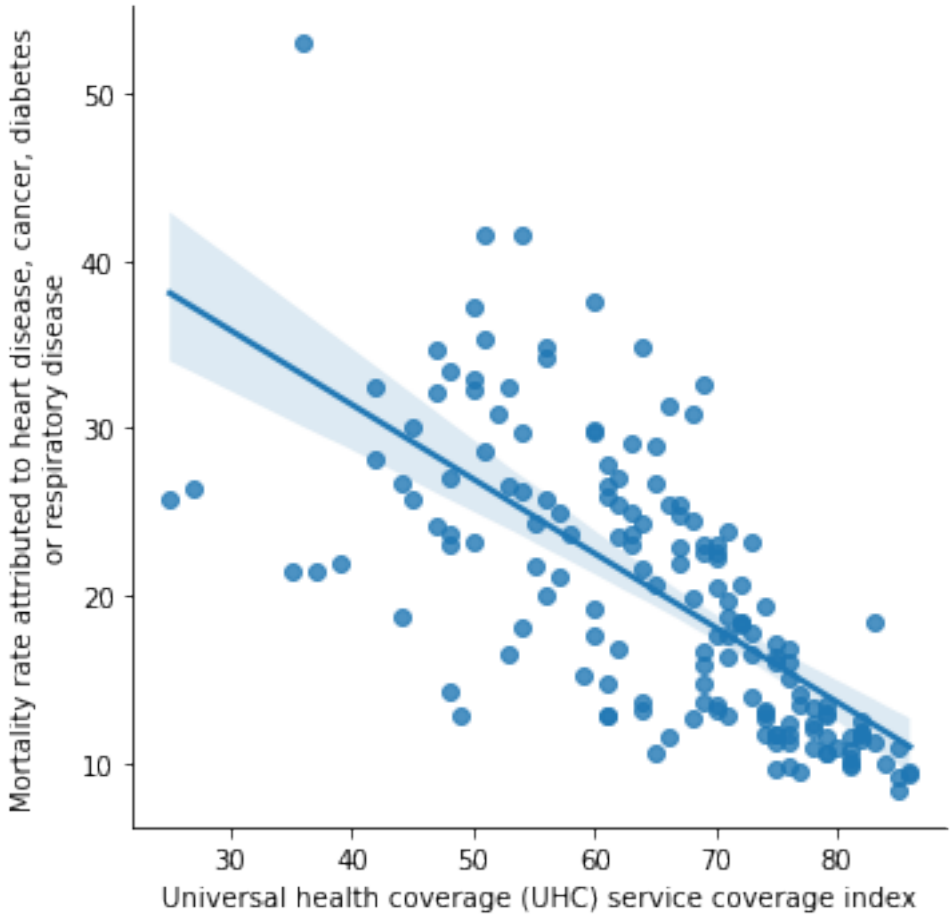
```
-----Sample test case-----
Universal health coverage:  92
Predicted mortality rate:  9
--------------------------
```

[4]:
```
y_prediction = neighbour_for_test.predict(X_test)
mean_square_error = metrics.mean_squared_error(y_test, y_prediction)
print("MSE: ", mean_square_error)
print("RMSE: ", sqrt(mean_square_error))
#An r-squared score of 1 means points in the scatterplot are very close to the␣
 ↪regression line
print("R-squared score: ", metrics.r2_score(y_test, y_prediction))
```

```
MSE:  32.01339015151515
RMSE:  5.658037659075375
R-squared score:  0.47457306005821154
```

The R-squared score explains the proportion of variance in the dependent variable that is able to be explained by the independent variable. As the R-squared value is 0.47, we can determine that there is a weak relationship between the two variables. The scatterplot below is included with a regression line to help us see the relationship between the two variables we used for our predictive model. This gives us a graphical interpretation and helps provide context of the results of our predictive model.

[5]:
```
universal_health_coverage = "Universal health coverage (UHC) service coverage␣
 ↪index"
physician_density = 'Mortality rate attributed to cardiovascular disease,␣
 ↪cancer, diabetes or chronic respiratory disease (probability)::::BOTHSEX'

ax = sns.lmplot(x = universal_health_coverage, y = physician_density, data = df)
ax.set(xlabel = "Universal health coverage (UHC) service coverage index",␣
 ↪ylabel = "Mortality rate attributed to heart disease, cancer, diabetes\nor␣
 ↪respiratory disease")
plt.show()
```

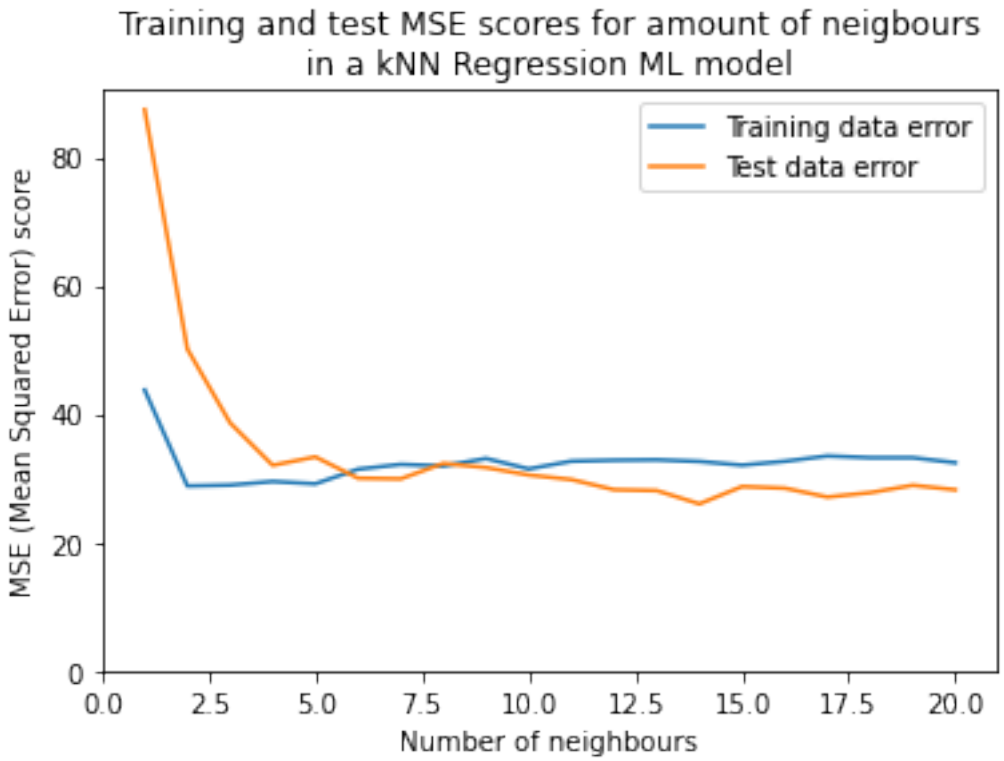


Given the intense scattering of the points in the scatterplot, it is safe to assume that the r-squared score of around 0.47 makes sense, as there are not that many points close to the regression line and a significant amount of outliers in the data. In comparison, a r-squared score of one/near one indicates a more compact scatterplot where all the points are near the regression line and a stronger relationship between the two variables.

1.2 Model Evaluation

[7]:
```
train_error_scores = []
test_error_scores = []
values = [i for i in range(1, 21)]

for i in values:
    error_model = neighbors.KNeighborsRegressor(n_neighbors = i)
    error_model.fit(X_train, y_train)

    training_prediction = error_model.predict(X_train)
    train_error = metrics.mean_squared_error(y_train, training_prediction)
```

```
    train_error_scores.append(train_error)

    test_prediction = error_model.predict(X_test)
    test_error = metrics.mean_squared_error(y_test, test_prediction)
    test_error_scores.append(test_error)

plt.title("Training and test MSE scores for amount of neigbours\nin a kNN␣
 ↪Regression ML model")
plt.plot(values, train_error_scores, label = "Training data error")
plt.plot(values, test_error_scores, label = "Test data error")
plt.xlabel("Number of neighbours")
plt.ylabel("MSE (Mean Squared Error) score")
plt.xlim(xmin = 0)
plt.ylim(ymin = 0)

plt.legend()
plt.show()
```



The lower the number of neighbours needed for consensus, the more likely the model will be overfit to the training data, as there is a decision boundary around every single point instead of being based on groups of points. This is able to explain why the mean squared error scores are so high for low neighbour counts, with the testing data running a MSE of around 87 at the beginning, and 42 for the testing data. As the number of neighbours increase however, this becomes less of a

factor and the model now runs a risk of underfitting. This is evident at the cross over point where number of neighbours is equal to 8.