First Home Assignment - Data Mining - Group 2

March 19, 2023

Ana Silva nº 39074, 15 hours; **Cláudia Afonso** nº 36273, 15 hours; **Martim Silva** nº 51304, 15 hours; **Rita Rodrigues** nº 54859 15 hours

---

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import f1_score, matthews_corrcoef, confusion_matrix, precision_score,␣
 ↪recall_score, explained_variance_score, mean_squared_error, accuracy_score, max_error,␣
 ↪mean_absolute_error
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
from scipy.stats import pearsonr
```

0.1 Exploratory data analysis

[2]:
```
train = pd.read_csv("train.csv")
```

[3]:
```
unique = pd.read_csv("unique_m.csv")
unique = unique.drop(columns = ['critical_temp', 'material'])
```

Both datasets ("train" and "unique") were combined into a single dataset ("merged_df") before performing dimensionality reduction and creating the classification/regression models. In this way, we are using the full dataset without losing any information or repeating the same lines of code for the two isolated datasets.

[4]:
```
merged_df = pd.merge(train, unique, left_index=True, right_index=True)
# Checking if there are missing values in the dataset
merged_df.isnull().sum().sum()
```

[4]:
```
0
```

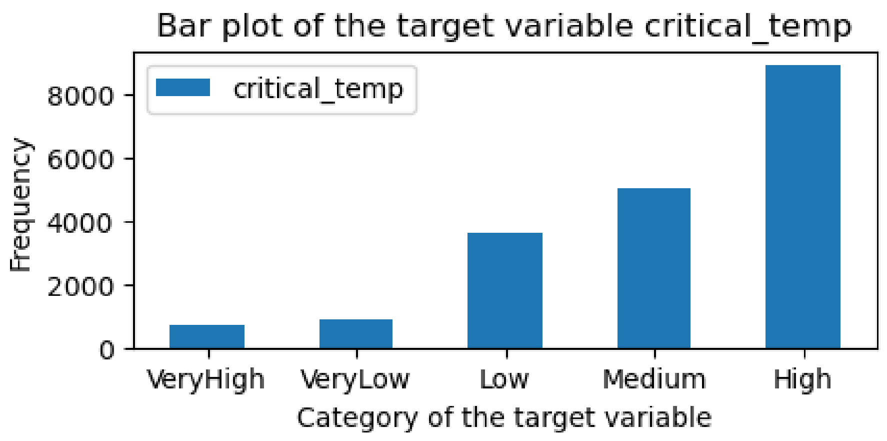
[5]:
```
# Split the trainining and testing sets
train_data = merged_df.drop('critical_temp', axis=1)
train_labels = merged_df['critical_temp']

# Split the data into a training set, a validation set, and a testing set
X_trainval, X_test, y_trainval, y_test = train_test_split(train_data, train_labels,␣
 ↪test_size=0.1, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, test_size=0.25,␣
 ↪random_state=42)
```

In the following code we will map the dependent variable, "critical_temp", into classes using the provided intervals.

```
[6]: bins = [0.0, 1.0, 5.0, 20.0, 100.0, 200.0]
     labels = ["VeryLow", "Low", "Medium", "High", "VeryHigh"]
     y_train_class = pd.cut(y_train, bins = bins, labels = labels, right = False, include_lowest =␣
      ↪True)
     y_val_class = pd.cut(y_val, bins = bins, labels = labels, right = False, include_lowest =␣
      ↪True)
```

```
[7]: df_class_counts = pd.cut(y_trainval, bins = bins, labels = labels, right = False,␣
      ↪include_lowest = True).value_counts(ascending=True).to_frame()
     bar_plot = df_class_counts.plot.bar(y='critical_temp', figsize = (5,2))
     bar_plot.set_xticklabels(bar_plot.get_xticklabels(), rotation=0)
     plt.title('Bar plot of the target variable critical_temp')
     plt.xlabel('Category of the target variable')
     plt.ylabel('Frequency')
     plt.show()
```



When plotting a bar plot of the "critical_temp" target variable (where this variable is discretized into several categories), it is possible to observe that there is an imbalance in the distribution of this varible by category in the instances of the training and validation sets. The most frequently encountered category of the target variable for an instance in the training and validation sets is "High", followed by "Medium" and "Low". In contrast, the less frequently encountered category of the target variable is "VeryHigh" followed by "VeryLow".

0.2 Dimensionality reduction

A recurrent issue in many sets of multivariate data is the vast number of variables present, which is problematic when attempting to perform an assessment of the data. This problem is typically known as the curse of dimensionality. In this context, the dimensionality reduction technique known as principal component analysis (PCA) was applied to the dataset in order to find data representations that are more economical and that can maintain the essential characteristics of the data.

The main aim of PCA is to reduce the number of dimensions in a dataset while still considering as much of the original variation as possible. This is achieved by transforming the original correlated variables into a new set of uncorrelated variables termed principal components. Although p components are required to reflect the total variability of a dataset, this variability can frequently be explained by a smaller number k of the principal components. These result from linear combinations of the original variables and are derived in decreasing order of importance, so that the first few components account for most of the variance present in all the original variables. Thus, k principal components can replace the initial p variables and the initial dataset composed of n measurements on p variables is reduced to one of n measurements on k principal components.[1,2]

The principal components of a set of variables depend immensely on the scales used to measure the variables.

Therefore, it is necessary to scale the data before performing dimensionality reduction with PCA (Scikit-learn's PCA implementation does not scale the data beforehand). Since the nature and the scales of measurement of the independent variables are quite different, the scaling of the data is also necessary before applying the regression and classification models in the full dataset without dimensionality reduction.

[8]:
```python
# Create a scaler with training set
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

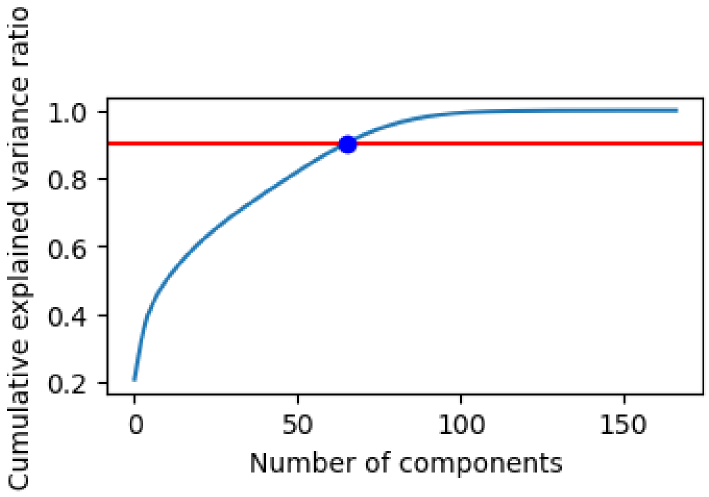
[9]:
```python
# Fit PCA to the scaled data
pca = PCA().fit(X_train_scaled)

# Calculate the cumulative sum of explained variance ratios
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Plot the cumulative sum
plt.figure(figsize=(4,2))
plt.plot(cumulative_variance_ratio)
plt.axhline(y=0.90, color='r', linestyle='-')
# Find index where cumulative explained variance ratio exceeds 0.90
n_components = np.argmax(cumulative_variance_ratio >= 0.90)

# Add marker for point where cumulative explained variance ratio equals 0.90
plt.plot(n_components, cumulative_variance_ratio[n_components], 'bo')
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance ratio')
plt.show()

print('The number of PCs at the point where the cumulative explained variance ratio is 0.9 is:
 ↪', n_components+1)
```



```
The number of PCs at the point where the cumulative explained variance ratio is
0.9 is: 66
```

[10]:
```
# Perform dimensionality reduction with PCA
pca = PCA(n_components+1)
pca.fit(X_train_scaled)
tve=0
for i, ve in enumerate(pca.explained_variance_ratio_):
    tve+=ve
    # The following if statements were added to print only the first and last 3 retained PCs.
    if i < 3:
        print("PC%d - Variance explained: %7.4f - Total Variance: %7.4f" % (i, ve, tve))
    if i > 62:
        print("PC%d - Variance explained: %7.4f - Total Variance: %7.4f" % (i, ve, tve))
```

```
PC0 - Variance explained:  0.2087 - Total Variance:  0.2087
PC1 - Variance explained:  0.0582 - Total Variance:  0.2669
PC2 - Variance explained:  0.0534 - Total Variance:  0.3203
PC63 - Variance explained:  0.0051 - Total Variance:  0.8934
PC64 - Variance explained:  0.0050 - Total Variance:  0.8984
PC65 - Variance explained:  0.0049 - Total Variance:  0.9033
```

These results show the amount of variance explained by each principal component (PC) extracted from the dataset, as well as the cumulative total variance explained by all the components up to that point. PC0 explains the largest proportion of variance (20.87%) in the dataset, followed by PC1 (5.82%) and PC2 (5.34%). Together, the first three components account for 32.03% of the total variance in the dataset. PCs 3 to 65 explain increasingly smaller proportions of variance in the data, with PCs 63 to 65 explaining 1.5% of the total variance. To achieve a cumulative explained variance ratio of 0.9 it is necessary to retain 66 principal components (PCs), since the counting of the number of PCs starts from 0 (the first PC is PC0). Overall, these results suggest that the first few PCs are the most important for capturing the majority of the variation in the dataset, while the later PCs have less and less impact.

[11]:
```
# Applying PCA to the training and validation sets
X_train_transformed = pca.transform(X_train_scaled)
X_val_transformed = pca.transform(X_val_scaled)
```

0.3 Regression models

[12]:
```
def printRegStatistics(truth, preds):
    print("The RVE is: ", explained_variance_score(truth, preds))
    print("The rmse is: ", mean_squared_error(truth, preds, squared=False))
    corr, pval = pearsonr(truth, preds)
    print("The Correlation Score is is: %6.4f (p-value=%e)"%(corr,pval))
    print("The Maximum Error is is: ", max_error(truth, preds))
    print("The Mean Absolute Error is: ", mean_absolute_error(truth, preds))
```

Linear Regression

[13]:
```
# Full dataset model
reg_full = LinearRegression()
reg_full.fit(X_train_scaled, y_train)
y_pred_full = reg_full.predict(X_val_scaled)
print("Full dataset model (Linear Regression) performance:")
printRegStatistics(y_val, y_pred_full)

# PCA transformed dataset model
reg_pca = LinearRegression()
reg_pca.fit(X_train_transformed, y_train)
y_pred_pca = reg_pca.predict(X_val_transformed)
print("\nPCA transformed dataset model (Linear Regression) performance:")
printRegStatistics(y_val, y_pred_pca)
```

```
Full dataset model (Linear Regression) performance:
The RVE is:  0.7623142440304604
The rmse is:  16.681543769797024
The Correlation Score is is: 0.8732 (p-value=0.000000e+00)
The Maximum Error is is:  133.2263552143609
The Mean Absolute Error is:  12.377407204255952

PCA transformed dataset model (Linear Regression) performance:
The RVE is:  0.6630454699804071
The rmse is:  19.858368905513103
The Correlation Score is is: 0.8143 (p-value=0.000000e+00)
The Maximum Error is is:  196.40584199639807
The Mean Absolute Error is:  14.816660050740778
```

Decision Tree Regressor

[14]:
```python
# Full dataset model
tree_full = DecisionTreeRegressor()
tree_full.fit(X_train_scaled, y_train)
y_pred_full = tree_full.predict(X_val_scaled)
print("Full dataset model (Decision Tree Regressor) performance:")
printRegStatistics(y_val, y_pred_full)

# PCA transformed dataset model
tree_pca = DecisionTreeRegressor()
tree_pca.fit(X_train_transformed, y_train)
y_pred_pca = tree_pca.predict(X_val_transformed)
print("\nPCA transformed dataset model (Decision Tree Regressor) performance:")
printRegStatistics(y_val, y_pred_pca)
```

```
Full dataset model (Decision Tree Regressor) performance:
The RVE is:  0.8737884829968046
The rmse is:  12.146916816984149
The Correlation Score is is: 0.9367 (p-value=0.000000e+00)
The Maximum Error is is:  126.43
The Mean Absolute Error is:  6.157601425812784

PCA transformed dataset model (Decision Tree Regressor) performance:
The RVE is:  0.8392815549283952
The rmse is:  13.715199454123765
The Correlation Score is is: 0.9199 (p-value=0.000000e+00)
The Maximum Error is is:  125.0
The Mean Absolute Error is:  6.919971068255693
```

The results show the performance metrics for two regression models (Linear Regression and Decision Tree) on two datasets (the full dataset and a PCA-transformed dataset). For the Linear Regression model, the full dataset had a higher RVE and correlation score compared to the PCA-transformed dataset. The overall performance of both models was reasonable, with a RVE in the range of 0.66-0.76 and a correlation score in the range of 0.81-0.87. For the Decison Tree model, the full dataset had a slightly higher RVE and correlation score compared to the PCA-transformed dataset. The overall performance of both models was reasonably high, with a RVE in the range of 0.84-0.87 and a correlation score in the range of 0.92-0.94.

Overall, the decision tree regressor (DTR) outperformed the linear regression model, both for the full dataset and its projection in a smaller feature space following dimensionality reduction with PCA. In addition, the Decision Tree model on the full dataset had the highest performance of all models, with a RVE of 0.87 and a correlation score of 0.94. The linear regression model did not perform as well, with lower RVE and correlation score suggesting that out of the two the DTR may be more appropriate for this dataset.

Unlike a linear regression model, decision tree regressors can capture non-linear relationships between variables.

Consequently, the fact that the DTR outperformed the linear regression model can indicate that the relationship between the independent variables and the target variable is non-linear or that the data has non-linear patterns.

0.4 Classification models

Gaussian Naïve Bayes

[15]:
```
def printClassificationStatistics(y_test, preds):
    print("The F1 score is: %7.4f" % f1_score(y_test, preds, average = "macro"))
    print("The MCC score is: %7.4f" % matthews_corrcoef(y_test, preds))
    print("The precision score is: %7.4f" %  precision_score(y_test, preds, average =␣
 ↪"macro"))
    print("The recall score is: %7.4f" %  recall_score(y_test, preds, average = "macro"))
    print("The accuracy is: ", accuracy_score(y_test, preds))
```

[16]:
```
nb_full = GaussianNB()
nb_full.fit(X_train_scaled, y_train_class)
y_pred_full_nb = nb_full.predict(X_val_scaled)
print("Full dataset model (Naive Bayes) performance:")
printClassificationStatistics(y_val_class, y_pred_full_nb)

nb_pca = GaussianNB()
nb_pca.fit(X_train_transformed, y_train_class)
y_pred_pca_nb = nb_pca.predict(X_val_transformed)
print("\nPCA transformed dataset model (Naive Bayes) performance:")
printClassificationStatistics(y_val_class, y_pred_pca_nb)
```

```
Full dataset model (Naive Bayes) performance:
The F1 score is:  0.3578
The MCC score is:  0.3322
The precision score is:  0.4538
The recall score is:  0.5652
The accuracy is:  0.46258361204013376

PCA transformed dataset model (Naive Bayes) performance:
The F1 score is:  0.3591
The MCC score is:  0.2056
The precision score is:  0.3768
The recall score is:  0.4314
The accuracy is:  0.483695652173913
```

Decision Tree Classifier

[17]:
```
tree_full_c = DecisionTreeClassifier()
tree_full_c.fit(X_train_scaled, y_train_class)
y_pred_full_tree = tree_full_c.predict(X_val_scaled)
print("Full dataset model (Decision Tree) performance:")
printClassificationStatistics(y_val_class, y_pred_full_tree)

tree_pca_c = DecisionTreeClassifier()
tree_pca_c.fit(X_train_transformed, y_train_class)
y_pred_pca_tree = tree_pca_c.predict(X_val_transformed)
print("\nPCA transformed dataset model (Decision Tree) performance:")
printClassificationStatistics(y_val_class, y_pred_pca_tree)
```

```
Full dataset model (Decision Tree) performance:
The F1 score is:  0.7995
The MCC score is:  0.7674
The precision score is:  0.8094
The recall score is:  0.7906
The accuracy is:  0.8432274247491639

PCA transformed dataset model (Decision Tree) performance:
The F1 score is:  0.7620
The MCC score is:  0.7321
The precision score is:  0.7707
The recall score is:  0.7544
The accuracy is:  0.8191889632107023
```

The results show the performance metrics for two classification models (Naïve Bayes and Decision Tree) on two datasets (the full dataset and a PCA-transformed dataset). For the Naïve Bayes model, the full dataset had higher MCC and precision scores compared to the PCA-transformed dataset. However, the overall performance of both models was not particularly strong, with F1 scores in the range of 0.36 and accuracy in the range of 0.46-0.48. For the Decision Tree model, the full dataset had higher F1, MCC, precision, recall, and accuracy scores compared to the PCA-transformed dataset. Both models had relatively strong performance, with F1 scores in the range of 0.76-0.80 and accuracy between 0.82-0.84.

Overall, the decision tree model on the full dataset had the highest performance of all the classification models, with an accuracy of 0.84 and an F1 score of 0.80. The Naïve Bayes models did not perform as well, with lower accuracy and F1 scores, suggesting that out of the two the Decision Tree model may be more appropriate for this dataset.

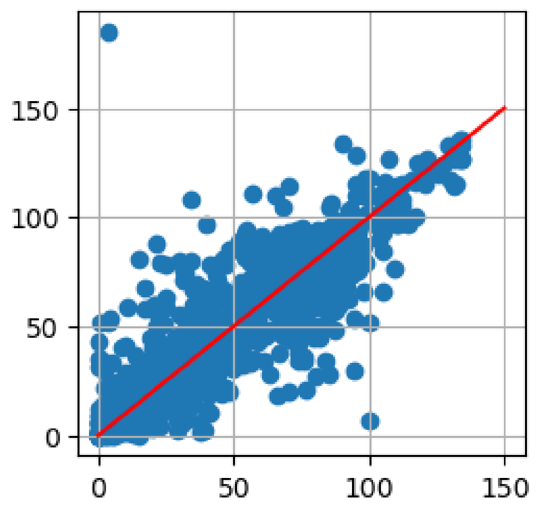
0.5 Best model - DTR with the full dataset (without prior dimensionality reduction with PCA)

Overall, we determined that the DTR with the full dataset was the best performing model for our analysis, with a RVE of 0.87, RMSE of 12.15, correlation score of 0.94, Maximum Error of 126.43, and Mean Absolute Error of 6.16. Having the best result out of the regression and classification models appears to indicate that the DTR without prior dimensionality reduction with PCA may be more appropriate for this dataset.

[18]:
```
y_pred_test = tree_full.predict(X_test_scaled)
printRegStatistics(y_test, y_pred_test)
```

```
The RVE is:  0.875068751808421
The rmse is:  11.973563818577643
The Correlation Score is is: 0.9377 (p-value=0.000000e+00)
The Maximum Error is is:  181.4
The Mean Absolute Error is:  6.168738723153971
```

[19]:
```
plt.figure(figsize=(3,3))
plt.scatter(y_test, y_pred_test)
plt.grid()
#the closer the predictions approach the 45 degree angle, the better the model
plt.plot([0, 150], [0, 150], c="r")
plt.show()
```



0.6 Conclusions

To sum up, this project focused on the application of various data mining techniques to analyze a merged dataset comprising two distinct datasets, namely “train” and “unique_m”. The analysis involved conducting dimensionality reduction using PCA and then applying linear regression, decision tree regressor, naïve bayes, and decision tree classifier to perform regression and classification tasks.

After performing dimensionality reduction using PCA, the score results for both the classification and regression models were generally worse. Since PCA works by transforming the original variables into a smaller set of variables (PCs) while preserving as much variance in the data as possible, it is possible that some information is lost in this process. In addition, the resulting PCs are produced by linear combinations of the original variables. If the relationships between the independent variables are non-linear, then the PCA may not capture the underlying non-linear structure of the data. As a result, the feature set composed of a smaller subset of variables (PCs) may lead to a decrease in model performance.

Based on the results shown here, we determined that the decision tree regressor with the full dataset was the best performing model for our analysis. The skewness of the target variable towards “High” and “Medium” categories in the training and validation sets might explain why the classification models in general performed worse than the regression models. When there is an imbalance of categories, with one having significantly fewer instances than another, classification models may struggle to perform well due to bias towards the majority class. When the data is imbalanced, the classifier may learn to simply predict the majority class, resulting in poor performance on the minority class.

As a future work we would consider the usage of Singular Value Decomposition (SVD) to perform dimensionality reduction on this dataset. In this way, it would be possible to observe the overall effect of this technique on the score results for the classification and regression models when applied to our dataset.

0.7 Bibliography

[1] Everitt, Brian, and Torsten Hothorn. An introduction to applied multivariate analysis with R. Springer Science & Business Media, 2011.

[2] Chatfield, Christopher, and Alexander J. Collins. Introduction to Multivariate Analysis. Chapman and Hall, 1980.