CS4622 - Machine Learning

Lab 01 - Feature Engineering

190377T

Introduction

We were provided with 3 CSV files. The first file is a training dataset containing 28,520 rows and columns. Each row represents a set of data, while there are 256 features associated with each row. Additionally, there are 4 target labels present for each row in this dataset. The second file is a validation dataset comprising 750 rows and columns. Similar to the training dataset, each row in the validation dataset has 256 features and 4 target labels. The third file is the test dataset file

The initial 256 columns hold values that form a vector representing the speaker's characteristics for each audio file within the dataset. These values are known as speaker embedding vectors. The final 4 columns contain labels that are connected to the speaker's information corresponding to each respective speaker embedding vector.

- Label 1 - Speaker ID
- Label 2 - Speaker age
- Label 3 - Speaker gender
- Label 4 - Speaker accent

Process

Initially, I define the labels and features that the dataset consists of. The labels represent different classes or categories, while the features are attributes associated with each data point. The DataFrames are then loaded from CSV files located in the specified paths.

To handle missing values in the dataset, specifically in label_2, I perform a data cleaning step. For the 'label_2' category, I filter out the rows with missing values using the notna() function, ensuring that only complete data points are considered for this label. Subsequently, I apply feature engineering by standardizing the feature values using the StandardScaler from scikit-learn. This ensures that all features have the same scale, which often aids in improving the performance of machine learning models.

```
from sklearn.preprocessing import StandardScaler

x_train = {}
y_train = {}

x_valid = {}
y_valid = {}

x_test = {}

for label in labels:
  tr_df = train_df[train_df['label_2'].notna()] if label == 'label_2' else
train_df
  vl_df = valid_df

  scaler = StandardScaler()
  x_train[label] = pd.DataFrame(scaler.fit_transform(tr_df.drop(labels,
axis = 1)), columns = features)
  y_train[label] = tr_df[label]

  x_valid[label] = pd.DataFrame(scaler.transform(vl_df.drop(labels, axis =
1)), columns = features)
  y_valid[label] = vl_df[label]

  x_test[label] = pd.DataFrame(scaler.transform(vl_df.drop(labels, axis =
1)), columns = features)
```

I utilized a Support Vector Machine (SVM) with a linear kernel to build a predictive model for the 'label_1' and 'label_3' which pertain to speaker identification and gender labels, respectively. SVM's proficiency in identifying optimal class-separating hyperplanes makes it apt for classification. I specifically chose the linear kernel for its suitability when data can be separated by a straight line.

```
from sklearn import svm

clf = svm.SVC(kernel= 'linear')
clf.fit(x_train['label_3'], y_train['label_3'])
```

For the Speaker accent label 4 classification, I adopted the K-Nearest Neighbors (KNN) approach utilizing scikit-learn's KNeighborsClassifier. KNN is a robust algorithm for classifying data based on the majority class of its neighboring data points. In this case, it suits the task of discerning speaker accents. With the parameter n_neighbors set to 3, the KNN classifier considers the three nearest neighbors when making predictions. This choice helps balance noise reduction and capturing local patterns.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train['label_4'], y_train['label_4'])
```

For the Speaker age label regression task, I employed scikit-learn's KNeighborsRegressor in combination with the K-Nearest Neighbors (KNN) algorithm. KNN regression is well-suited for predicting numerical values by considering the average or weighted average of the target values of its neighboring data points. With the parameter n_neighbors set to 3, the KNN regressor accounts for the three closest neighbors when generating predictions. This balance between considering multiple neighbors and avoiding overfitting is integral to accurate age predictions.

```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors=3)
knn.fit(x_train['label_2'], y_train['label_2'])
```

Feature Engineering Techniques

In feature reduction, I applied two distinct techniques:

- SelectKBest
- Principal Component Analysis (PCA)

Firstly, I employed SelectKBest, a method from scikit-learn's feature_selection module, utilizing the ANOVA F-statistic (f_classif) to evaluate feature relevance. By specifying k the top k features with the highest F-scores were chosen. This approach aimed to retain the most informative variables for prediction.

- Label 1: k = 130
- Label 2: k = 200
- Label 3: k = 15
- Label 4: k = 60

```
from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(f_classif, k=130)
x_train_kBest = selector.fit_transform(x_train['label_1'],
y_train['label_1'])
x_valid_kBest = selector.transform(x_valid['label_1'])
x_test_kBest = selector.transform(x_test['label_1'])
```

Subsequently, I employed PCA, a dimensionality reduction technique, using the PCA class from scikit-learn with a target explained variance of 95%. This approach aims to capture a significant proportion of the original data's variance within a reduced feature set. The svd_solver='full' parameter choice ensures that full Singular Value Decomposition is employed for optimal results.

After PCA:

- Label 1: 54 features
- Label 2: 63 features
- Label 3: 12 features
- Label 4: 37 features

```
from sklearn.decomposition import PCA

pca = PCA(n_components= 0.95, svd_solver='full')
pca.fit(x_train_kBest)
x_train_trf = pd.DataFrame(pca.transform(x_train_kBest))
x_valid_trf = pd.DataFrame(pca.transform(x_valid_kBest))
```

For label 1, 3 and 4, The confusion matrix was printed to visualize the classification results. Additionally, I calculated accuracy using metrics.accuracy_score, while precision and recall were computed using metrics.precision_score and metrics.recall_score. These metrics offer insights into the model's ability to correctly classify and differentiate classes. They collectively contribute to understanding its overall performance and effectiveness in handling the specific task at hand.

For label 2, I used mean absolute error, mean squared error which provides insight into the average squared differences between predictions and actual values and the coefficient of determination (R-squared) which measures the proportion of variance in the dependent variable that the model explains to evaluate the performance.

Accuracy after PCA:

- Label 1: accuracy_score - 0.9573333333333334
- Label 2: R-squared - 0.9842107285505826
- Label 3: accuracy_score - 0.9826666666666667
- Label 4: accuracy_score - 0.9853333333333333

Colab Notebook Links

Label 1

https://colab.research.google.com/drive/1gzF2t-EYku4QJG2PjjhB7NanogDoFkp8?usp=sharing

Label 2

https://colab.research.google.com/drive/1A_BHIXLz_ST9Zx9ij9vbnkxUhVrwQiWP?usp=sharing

Label 3

https://colab.research.google.com/drive/1F15Gx-oKmoTulV9HDmWO9I-vFvQpQzM4?usp=sharing

Label 4

https://colab.research.google.com/drive/1YdB3mhlcnsRY3sLpqo3Vv5jEQUaWnStN?usp=sharing