Applied Machine Learning
Initial Project

Georgios Sevastakis

May 2024

1 Classification

Firstly, for all cases I imported the train data from the file *'AppML_InitialProject_train.csv'* and the test data from the *'AppML_InitialProject_test_classification.csv'* and standardized them using the *fit_transform* method from the *StandardScaler* from the *sklearn.preprocessing* library.

1.1 XGBoost

First, I split the data into train and validation data ($test\_size = 0.25$, $random\_state = 42$)). To find the 20 most important features, I used the *feature_importances_* method of XGboost having defined and trained a classifier Then, I used the hyperopt library for bayesian hyperparameter optimization. For the hyperparameter optimization, cross–validation ($cv = 5$) was used at each trial. These were found to be:

```
Best Hyperparameters: {'colsample_bytree': 0.7838522963083432, 'gamma': 0.7018822402097628, 'learning_rate': 0.05306630350590696, 'max_depth': 8, 'min_child_weight': 9.827462508018204, 'n_estimators': 700, 'subsample': 0.6858906365814332}
```

Listing 1. Best hyperparameter values.

Plugging the hyperparameter's optimal values to a *XGBClassifier*, I computed the *logloss* of the validation data with crossvalidation ($cv = 5$) . The reported average *logloss* was approximately 0.07. The elapsed time was approximately 30 minutes.

1.2 Tensorflow

As before, I split the data into train and validation data ($test\_size = 0.25$, $random\_state = 42$). Then I build and defined and trained the model using *keras*. Subsequenlty, I used the model to find the *shap* values to find the 20 most important variables. Then I defined a space to search for the optimal values of the hyperparameters using *hyperopt* for bayesian optimization The best values of the hyperparameters (cross–validation was also applied here on each trial with $cv = 5$) were found to be:

```
Best Hyperparameters: {'dropout': 0.1767806339326483, 'lr': 0.01576289899456416, 'num_hidden_layers': 1, 'units': 64}
```

Listing 2. Best hyperparameter values.

Next, I used these hyperparameters (*epochs* and *batch size* were manually set to 10 and 32 respectively, while the activation function was *'relu'* for the initial and hidden layer and *'sigmoid'* for the output layer and the Adam optimizer was utilized) on the validation data using cross–validation($cv = 5$, $random\_state=42$). The reported mean logloss was 0.115 rounded to the third decimal point and the elapsed time was approximately 6 hours and 30 minutes.

2 Regression

2.1 XGBoost

First, I split the data into train and validation data (*test_size* = 0.25, *random_state* = *42*) and I defined a xgb.XGBRegressor initially. Here, I used the *feature_importances_* of XGBoost to find the 25 most important variables. Subsequenlty, I defined the hyperparameter search space. By utilizing *hyperopt* I applied bayesian optimization (cross–validating on each trial with *cv* = *5*) and computed the optimal values for the hyperparameters

```
Best Hyperparameters: {'colsample_bytree': 0.9673993118371037, 'gamma':
    0.12211782867486987, 'learning_rate': 0.12028705977003203, 'max_depth': 9, '
    min_child_weight': 9.687097601465405, 'n_estimators': 400, 'subsample':
    0.953096656192528}
```

Listing 3. Best values of the hyperparameters.

Then, I used the computed optimal hyperparameter values to train the XGBRegressor and the average (cross–validation was applied with cv = 5) *Mean Absolute Error* was found to be 0.135. By rescaling the results back, I computed the predicted energies from the test data. The elapsed time was about 3 minutes.

2.2 Pytorch

Here, I used permutation importance to find the most important 25 variables. Then, I defined a hyperparameter space and implemented *random search* (50 trials). At each trial I applied 5–fold cross–validation and ended up with the following hyperparameters values:

```
Best hyperparameters: {'hidden_dim1': 179, 'hidden_dim2': 6, 'learning_rate':
    0.01595187774544761, 'batch_size': 42}
```

Listing 4. Optimal hyperparameter values.

By applying 5–fold cross–validation with the model containing the optimal hyperparameter values, I ended up with an average mean absolute error of 0.2106. The number of epochs was set to 20 for all training models. The elapsed time was 4 minutes. By inspecting the output of the predicted data, I fear that the Pytorch implementation was not successful. I believe I have overtrained the model.

3 Clustering

Firstly, for all cases I imported the data from the file *'AppML_InitialProject_test_clustering.csv'* and standardized them using the *fit_transform* method from the *StandardScaler* from the *sklearn.preprocessing* library. Also, for all cases, I calculated the Laplacian scores to find the 10 most important features.

3.1 KMeans

Having restricted the data to those with the 10 most important features, I computed the optimal number of clusters by computing the CH index from the *calinski_harabasz_score* method. I did that for varying number of clusters, namely from 3 to 25. The **optimal number of clusters** was found to be 10. The labeling was ultimately achieved by applying *KMeans* to the data. The elapsed time was approximately 67 seconds.

3.2 Hierarchical clustering: Agglomerative

In this case, I used the elbow method to determine the optimal number of clusters. I used the *clusteval* library not only for the elbow method, but also to implement the agglomerative algorithm for the clustering. The optimal number of clusters was found to be 5 and the elapsed time was approximately 63 seconds.

3.3 DBSCAN

This method required more experimenting due to the *eps* and *min_samples* parameters. By utilizing the *k-Distance Graph Method*, the parameter *eps* was found to be approximately 1, while the *min_samples* was chosen to initially be 11 (same as the number of features). By experimenting with values around those, I ultimately chose $eps = 0.66$ and $min\_samples = 10$, which yielded 6 clusters excluding the noise (2362 data points were considered noise in the end). The elapsed time was approximately 64 seconds.