CS3244 Project 1
Group 17

Nicholas Jimmy Alden, Nixon Widjaja, and Sum Hung Yee

1 Introduction

1.1 Problem Statement

In this project we are to construct a meta-learning model for a given dataset specified in "meta-features.csv". This meta-learning model will decide, given a base-level dataset, on an appropriate supervised learning model, (decision tree, k-nearest neighbors, and draw) that would best represent the base-level dataset.

1.2 Dataset

Dataset D consists of 226 data points containing 8 beta-values frequency bin figures corresponding to 226 UCI datasets. Each data point can be assigned either one of 3 labels: '0', '1', '2'. The labels describe the most appropriate classifier for the base-level dataset, i.e. decision tree for '0', kNN for '1', and drawing results for '2'. The kNN classifier here is defined with default hyperparameters from the *sklearn* module, while the DT classifier is defined using hyperparameters *class_weight = 'balanced'* and *max_depth* = $min(v, \sqrt{u})$ where v and u are, respectively, the number of features and instances in the corresponding dataset.

Of the 226 entries, 20 are labeled "0", 148 are labeled "1", and 58 are labeled "2". Considering the imbalanced dataset, we performed oversampling to balance the proportion of each label in the dataset. We import SMOTE (Synthetic Minority Oversampling Technique) and RandomOverSampler (ROS) from the *imblearn* package and applied both to D, resulting in separate datasets which we will call D-SMOTE and D-ROS, respectively.

The ROS class selects and duplicates random data points from minority classes, which are then appended to the resulting dataset. That is, the process balances the proportion of the classes without adding new information to the dataset. SMOTE, on the other hand, creates new data points inferred from existing points in the minority classes.

Using the ROS and SMOTE oversampler, we created data points with labels 0 and 2 such that we have around a 30%, 40%, and 30% distribution for labels 0, 1, and 2 respectively, giving us 113, 148, and 113 data points for each label. This is to avoid creating too many artificial data points. We also retained the original dataset to compare with the results after oversampling.

2 Methodology

We first imported the libraries we need, including *pandas*, *NumPy*, k-Nearest Neighbor (kNN), and Decision Tree (DT) with various model selections such as *train_test_split*, *cross_val_score*, and *GridSearchCV* from *sklearn*.

2.1 Generating $\mathbf{h_{meta} = h^*_D}$

We trained D with kNN and DT and used *GridSearchCV* to find the best hyperparameter for each model. We then determined h^*_D as the model that has higher accuracy among kNN and DT. As further analysis to compensate for the imbalanced dataset D, we repeated the same process with dataset D-SMOTE and D-ROS.

2.2 Preparing the Data

Firstly, we split columns of D into x and y, with x the 8 beta-values and y the labels corresponding to the beta-values. Then, we used *train_test_split* to split the dataset into a training set and a test set with a 90-10 proportion, so at the end of each model training, we can evaluate our models with the test data.

2.3 K-Nearest Neighbor

We first checked the accuracy of default kNN with k = 5 to verify whether we have obtained an increase in accuracy after optimizing the hyperparameters. To find the most optimal parameter, i.e., to obtain the best accuracy, we opted for *GridSearchCV* from *sklearn,* which evaluates the accuracy of the model via 10-fold cross-validation. In our parameter grid, we included *'distance*' for *'weights'*, *['braycurtis', 'canberra', 'chebyshev', 'cityblock', 'cosine', 'euclidean']* for *'metric'*, and discrete values from 1 to 60 (inclusive) for k. We choose 60 as the upper bound because the smallest class contains only 20 data points and a larger k value would result in the smallest class always being overlooked. We then evaluated kNN with the hyperparameters returned by *GridSearchCV* using the test data.

2.4 Decision Tree

Similar to how we worked on kNN, the *GridSearchCV* function is once again utilized to evaluate the viability of multiple combinations of hyperparameters. The hyperparameters that are searched over are as follows: *['gini', 'entropy']* for *'criterion', [None, 'balanced]* for *'class_weight',* and discrete values from 1 to 8 (inclusive) for *'max_depth'*. The upper limit 8 is chosen as this is the number of features in D. Again, the search rates the accuracy of each combination of hyperparameters through 10-fold cross-validation.

2.5 Generating Meta Features

We now treat our original dataset as a base-level dataset. From this dataset, we formulated an algorithm to obtain an array of meta-features. First, using *numpy.linalg*, we defined a function that calculates the Euclidean distance between 2 points. This gives us an easy way to calculate the inverse distance, or *invd*. We then calculate the beta values of each point by creating a distance matrix, or a 2-dimensional array that records the distance between any 2 points. Each row in the distance matrix contains information about the data point in the same row as the dataset. Note that the values along the diagonal are 0 and that this matrix is symmetric. Each row, therefore, contains the distance between the data point at the specific row, and every other point.

For each row/data point, the beta value is calculated by taking a ratio of 2 values:

For the first value, we need to find the number of data points with the same label as our current data point. This is done easily through the use of a dictionary, which stores labels as keys and frequencies as values. Suppose there are k data points with the same label, including itself. We then find the kth smallest distance in the appropriate row of our distance matrix. Now, recalling that each entry at column j in row i corresponds to the distance from the datapoint at row i to the datapoint at row j of our dataset, we sum up all inverse distances where the distance from i to any j is less than k, where $j \neq i$, if the labels of the data points at j and i are the same.

For our second value, we simply map all the distances in row i of our distance matrix to the corresponding inverse distances and sum them up. We subtract 1 from this sum, taking into account the entry along the distance-matrix diagonal, which is mapped to $\frac{1}{1+0} = 1$.

With the beta values for each datapoint generated, we sort them into 8 equal bins divided along the interval $[0, 1]$. Each bin contains the proportion of points contained within a specific interval.

2.6 Generating h'_D and Comparing with h^*_D

This can be done by feeding the D meta-features (an array of 8 values) into our trained optimal model to predict an appropriate label (the optimal classifier) for dataset D. The result it gives is h'_D. We then feed D into h^*_D and h'_D and compare the accuracy using 10-fold cross-validation.

3 Results and Evaluation

3.1 K-Nearest Neighbors

Table 1. kNN Accuracy and Optimal Hyperparameters

Dataset	**D**	**D-SMOTE**	**D-ROS**
Default accuracy	0.6057	0.5593	0.6725
Training accuracy	0.6857	0.6906	0.8749
Test accuracy	0.7391	0.6316	0.8421
Optimal distance metric	Chebyshev	Cosine	Canberra
Optimal k	53	4	1

From this result, we can observe that the optimal distance metric may not always be Euclidean distance as there may be more optimal similarity measures that fit the data.

3.2 Decision Trees

Table 2. DT Accuracy and Optimal Hyperparameters

Dataset	D	D-SMOTE	D-ROS
Default accuracy	0.5169	0.4433	0.5443
Training accuracy	0.6760	0.6010	0.7829
Test accuracy	0.7391	0.6316	0.7368
Optimal splitting criterion	Entropy Change	Entropy Change	Entropy Change
Optimal depth	2	8	8

3.3 Choosing h_{meta}

The model with the highest accuracy is kNN with D-ROS. However, the optimal k of 1 indicates that it is an overfitted model, which explains the high accuracy, therefore we discard the model.

kNN and DT with dataset D have the same test accuracy, but we discard these 2 models as the inherent data imbalance might affect the classification process done by the model.

The most valid kNN model appears to be kNN with D-SMOTE. With a test accuracy of 0.63, it is the second highest within kNN algorithms. Furthermore, a k value of 4 is not too large, nor is it too small to cause overfitting. However, the test accuracy is deemed to be too low.

Therefore, out of the 3 remaining valid models, we choose the one with the highest accuracy, which is DT with D-ROS, as our h^*_D, and hence our h_{meta}.

3.4 Generating Meta-Features

As explained previously, each data point in the dataset is assigned a beta-value. Here are the results of the first 10 data points.

```
num: 87.3547, denom: 165.2596, bounding dist: 0.5346, beta: 0.5286
num: 85.9979, denom: 164.5758, bounding dist: 0.5046, beta: 0.5225
num: 85.6723, denom: 162.0568, bounding dist: 0.4466, beta: 0.5287
num: 86.9847, denom: 166.4523, bounding dist: 0.4953, beta: 0.5226
num: 84.1723, denom: 162.8415, bounding dist: 0.499, beta: 0.5169
num: 86.042, denom: 166.1092, bounding dist: 0.4906, beta: 0.518
num: 74.1227, denom: 149.567, bounding dist: 0.561, beta: 0.4956
num: 84.0517, denom: 161.9932, bounding dist: 0.4993, beta: 0.5189
num: 86.3347, denom: 164.5378, bounding dist: 0.5371, beta: 0.5247
num: 85.9912, denom: 166.0343, bounding dist: 0.4953, beta: 0.5179
```

Fig. 1. Beta values for the first 10 data points.

Each beta value is placed into 1 of 8 equal bins divided along the interval [0,1]. Each bin *b* will contain the proportion of beta values that fall under the

interval $[\frac{b}{8}, \frac{(b+1)}{8})$. The distribution of beta values are as follows:

```
Values for interval:
(0.0, 0.125): 0.3451327433628323
(0.125, 0.25): 0
(0.25, 0.375): 0
(0.375, 0.5): 0.15044247787610632
(0.5, 0.625): 0.5044247787610626
(0.625, 0.75): 0
(0.75, 0.875): 0
(0.875, 1.0): 0

As a vector:
          0
0  0.345133
1  0.000000
2  0.000000
3  0.150442
4  0.504425
5  0.000000
6  0.000000
7  0.000000
```

Fig. 2. Distribution of Beta values.

3.5 Predictions and Comparisons

```
Prediction: [1]
dt ROS as h*d on base dataset
h*D: 0.48735177865612644
h'D = default knn: 0.5978260869565217
default dt: 0.4254940711462451
```

Fig. 3. Predictions.

Feeding in our newly-generated meta-features into h^*_D yields a prediction of 1, and h'_D would be *sklearn* default kNN classifier with k = 5.

h^*_D gives a prediction of 1 (kNN better) for D's meta-features. Therefore, h'_D would be *sklearn* default kNN classifier with k = 5. We first verify by finding out the accuracy of the default decision tree classifier and verify that the accuracy of the default kNN classifier is higher. This means that our model correctly predicts that the default kNN is more optimal than the default decision tree classifier.

Testing this on D with *cross_val_score*, we get an accuracy mean score of 0.5978 for h'_D as opposed to h^*_D's 0.4966. Therefore, we conclude that h'_D is superior to h^*_D.

4 Conclusion

Comparing h'_D to h^*_D, our meta-learning model correctly predicts that the better default classifier is the kNN classifier as specified above, when compared to the base decision tree. However, it remains that the chosen meta-learning model does not perform as well in terms of accuracy when compared to the default kNN classifier that it chooses. In actuality, h'_D performs better than h^*_D in this context. Therefore, we conclude that meta-learning may not have been effective in this case.

References

1. Pedregosa et al.: *'Scikit-learn: Machine Learning in Python'*. JMLR 12. pp. 2825-2830 (2011)
2. Lemaitre, G., Nogueira, F., Aridas, C.K.: *'Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning'*. JMLR 18. pp 1-5 (2017)
3. Harris, C.R., Millman, K.J., van der Walt, S.J. et al.: *'Array programming with NumPy'*. Nature 585. pp. 357–362 (2020).
4. McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 56-61).
5. The pandas development team. (2020, February). pandas-dev/pandas: Pandas 1.5.3. Zenodo.
6. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017