Technical Report – Assignment 1

Andrew (Hwan) Park (hwanp)

1.0. Introduction

In her research, Arcelli Et Al. reaches the conclusion that any machine learning algorithms can be used to detect code smells with high accuracy.[1] However, we know the research isn't flawless: the dataset is biased.[2] As we discussed in class, flawed datasets can be detrimental. Such datasets can easily invalidate the conclusion and methodology of the research. Thus, the colloquial expression: "garbage in, garbage out". In this assignment, I aim to replicate Arceilli Et Al.'s study with the provided and corrected datasets.

Assignment 1: Code Smell Classifier

2.1. Training the Model

We are provided with four .arff datasets: data class, feature envy, god class, and long method. I trained the Decision Tree Classification model for each dataset with user input from the command line interface. I chose to stick to one model, rather than experiment with more models, due to my unfamiliarity with machine learning in general. Rather than overwhelm myself with excessive theory, I decided to attempt to understand decision tree classification as best as I can.

For the training phase, I stuck with the same steps performed by Professor Sousa during lab 2. From a high level, we prepare and clean the datasets. Then, after fitting the model, we tune the hyperparameters with cross validation and grid search techniques.

2.2. Specific steps performed to train the models:

1. To prepare the datasets, we load the .arff files as a pandas data frame, and store the data frame as a class variable. I decided to not store the arff file itself to reduce redundancy and additional overhead in memory.
2. To clean the datasets, we translate the y-values (dependent variable) from boolean to 0/1 bit values using the preprocessing encoder.
3. Then, we fill in the missing values in the x-values (independent variables) with median values of each column. We use the SimpleImputer class to easily perform this task.
4. We must now split the dataset into training and testing sets. I maintain the 85/15 split used by Professor Sousa.
5. We now tune the hyperparameters using grid search and cross validation (10 fold). We use the same range of depths (1 ~ 20 inclusive) and number of leaves (choice between

[1] Dealing with Skewed Dataset Slides, Leo Sousa
[2] Ibid.

1, 5, 10, 20, 50, 100) used by Professor Sousa for the grid search. Although I decided to leave these parameters alone, these values are simply default values that can be overridden by named parameters.

6. The best model from the grid search is stored separately as best_model class attribute. Note that the X_train, y_train, x_test, and y_test variables are also stored as class attributes.

2.3. Calculating and Displaying the Output

- Then, we calculate the results for various metrics: accuracy, train_accuracy (accuracy on train data), f1_score, train_f1_score (f1 score on train data). These metrics are stored as class attributes.
- Then, using the brilliant pip module tabulate, we print the results in an easily comprehensible table. Depending on the actions selected by the user in the CLI (run or compare), the displayed outputs will differ. In a normal run, the program displays accuracy and f1_score. However, in a compare run, the program displays all four metrics mentioned in step 7.

2.4. How to Run the Program

- The program is run by the entry point at main.py. After running main.py, the program is entirely handled by a handy CLI created using PyInquirer. The user inputs are validated to ensure that the user selects at least one option for both “action” and “datasets”. Proceed as you would any other CLI; the instructions are explicitly told in the command line.

2.5. Caveats:

- Note that the model is trained every time the program is run. We assume that the dataset may differ from the previous run.
- Note that the program is independent of the datasets. Even if we add more datasets to the datasets/ directory, the program should be able to handle them without errors. One caveat is that the datasets should be named in hello-world.arff format. We use the dashed filename to parse the output table headers.
- The time to train all four models is less than 2 minutes. However, the run time may severely differ in less powerful computers. All models were tested on the M1 Pro base model Macbook Pro with unified 16GB memory.

3.0. Results & Performance:

smell	**accuracy**	**f1_score**
Feature Envy	85%	51%
God Class	75%	24%

Data Class	80%	51%
Long Method	83%	0%

smell	accuracy	train_accuracy	f1_score	train_f1_score
Feature Envy	85%	84%	51%	55%
God Class	75%	85%	24%	59%
Data Class	80%	85%	51%	70%
Long Method	83%	83%	0%	0%

3.1. How and Why is the Dataset Biased?

The replicated results are interesting to say the least: the accuracy is relatively high, but the f1 scores are very low. Some smell models even have 0% f1 scores. After some research, I came to the conclusion that the dichotomy between the high accuracy and low f1 score may indicate excessive actual (true) negatives in the datasets.[3] Why? The f1 score, which is the harmonic mean of precision and recall, does not account for the number of true negatives, but accuracy does![4]

A simple thought experiment brilliantly illustrates this point. Imagine I claim that I have a model that identifies terrorists trying to board flights with 99.999 percent accuracy. How did I create such a model? I simply label every single person boarding a flight “not a terrorist”. Since the US saw 19 terrorists from 2000-2017, with 800 million average passengers per year, the model achieves an amazing accuracy.[5] The large number of true negatives can heavily influence accuracy.

Once we look at Arcelli Et Al.’s smelly/non-smelly instances in the paper, we can see the skews in the dataset. Professor Sousa once mentioned that in the real world, smells are approximately 1 percent. However, in Arcelli Et Al.’s dataset, the smelly/non-smelly ratio is around 70 percent on average.[6] Thus, not only does the dataset fail to reflect reality, but it also is heavily skewed in favor of smelly instances. Thus, Arcelli Et Al.’s high performing models are misleading.

b'false'	700
b'true'	140

[3] https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
[4] Ibid.
[5] https://builtin.com/data-science/precision-and-recall
[6] Comparing and experimenting machine learning techniques for code smell detection, Arcelli Et Al.

Name: is_feature_envy, dtype: int64	
b'false'	700
b'true'	140
Name: is_data_class, dtype: int64	
b'false'	700
b'true'	140
Name: is_god_class, dtype: int64	
b'false'	700
b'true'	140
Name: is_long_method, dtype: int64	

The table above summarizes the number of true/negative values of the independent variables in our datasets. Exactly 20 percent of all rows are smelly. The remaining 80 percent of all rows are non-smelly. Naturally, we can conclude that the models generated from datasets given to us by Professor Sousa may have had a large number of true negatives that contribute to a low f1 score but high accuracy. The datasets given to us are unfortunately still skewed and do not reflect reality. Thus, we conclude that the model is unreliable, especially under real world conditions.

4.0. Bibliography

1. https://builtin.com/data-science/precision-and-recall
2. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
3. Comparing and experimenting machine learning techniques for code smell detection, Arcelli Et Al.
4. Dealing with Skewed Dataset Slides, Leo Sousa