1 About Task 1

In task 1, datasets were splitted into 2 parts: 80% of data for training and validation, and 20% of data for testing. I used Support Vector Classifier (SVC), Gaussian Naive Bayes, Decision Tree, and Random Forest to train on the 4 datasets, as suggested by the example in the assignment description. All models, except Gaussian Naive Bayes, were trained multiple times with Grid Search to fine tune the hyper parameters, which were hand picked randomly. Gaussian Naive Bayes does not have hyper parameters that would make a significant difference after tuning. Grid Search was performed with cross-validation of 5 folds, which is also the default behavior of scikit-learn's GridSearch class.

To run the script, first create a virtual environment and get into it:

```
python -m venv ./pyenv
source ./pyenv/bin/activate
```

Then do the following to install dependencies:

```
pip install -r requirements.txt
```

Then you can run the script:

```
python task1.py
```

After installing dependencies, you can also run the script for task 2:

```
python task2.py
```

Hyper parameters

For SVC, the regularization parameter, C, was chosen between 1 and 10; the degree was chosen from 1 through 5; and the kernel was chosen from 'poly', 'rbf', and 'sigmoid'. Among all the scikit-learn provided kernels, 'linear' was not chosen because it was found to be running extremely slow with our dataset compared to other kernels; and conceptually, 'poly' would almost already cover the case of 'linear' when degree was set to 1.

For Decision Tree, criterion for splitting was chosen from 'gini', 'entropy', and 'log_loss'; the strategy used to choose the split at each node was chosen between 'best' and 'random'; the maximum number of features when seeking for the best split was chosen from the total number of features, square root of the total number of features, and the log of the total number of features base 2.

For Random Forest, aside from all the available hyper parameter choices succeeded from Decision Tree model, the number of estimators were tuned ranging from 20 to 200.

2 Results on test set

Data Class

Model	Accuracy	F1-Score	Precision	Recall	Avg Training Time (s)
SVC	84.08%	0	0	0	0.096
GaussianNB	59.24%	0.44	0.28	1.0	0.00092
Decision Tree	73.25%	0.045	0.053	0.04	0.0024
Random Forest	74.52%	0.13	0.14	0.12	0.21

Feature Envy

Model	Accuracy	F1-Score	Precision	Recall	Avg Training Time (s)
SVC	83.97%	0	0	0	0.049
GaussianNB	79.49%	0.53	0.42	0.72	0.00063
Decision Tree	71.79	0.083	0.087	0.080	0.0024
Random Forest	71.79%	0.15	0.15	0.16	0.24

God Class

Model	Accuracy	F1-Score	Precision	Recall	Avg Training Time (s)
SVC	85.99%	0	0	0	0.026
GaussianNB	87.89%	0.58	0.57	0.60	0.00051
Decision Tree	75.79%	0.050	0.056	0.045	0.0025
Random Forest	75.79%	0.050	0.056	0.045	0.22

Long Method

Model	Accuracy	F1-Score	Precision	Recall	Avg Training Time (s)
SVC	79.49%	0	0	0	0.055
GaussianNB	75.64%	0.46	0.42	0.5	0.00064
Decision Tree	66.03%	0.070	0.080	0.063	0.0029
Random Forest	64.10%	0.034	0.038	0.031	0.24

3 Bias in datasets

As noticed in part 2, all models tend to have very low f1 score despite of relatively high accuracy. Looking closer, many of the models tend to have both very low precision and very low recall, which indicates that the model might tend not to predict true overall. We can do some statistics on the data:

Imports

In [1]:
```
import numpy as np
import pandas as pd

from task1 import get_data_np
```

In [2]:
```
dc_data, dc_labels = get_data_np('data-class.arff', 'is_data_class')
fe_data, fe_labels = get_data_np('feature-envy.arff', 'is_feature_envy')
gc_data, gc_labels = get_data_np('god-class.arff', 'is_god_class')
lm_data, lm_labels = get_data_np('long-method.arff', 'is_long_method')
```

Data class

In [4]:
```
print(pd.Series(dc_labels).value_counts())
```

```
False    644
True     140
dtype: int64
```

Feature envy

In [5]:
```
print(pd.Series(fe_labels).value_counts())
```

```
False    648
True     128
dtype: int64
```

God class

In [6]:
```
print(pd.Series(gc_labels).value_counts())
```

```
False    654
True     130
dtype: int64
```

Long method

In [7]:
```
print(pd.Series(lm_labels).value_counts())
```

```
False    647
True     129
dtype: int64
```

So on average only about 20% of the total data has the label of true. This could very well confuse the model and let the model tend to predict false. The datasets was biased to have too much good code, and resulted in models failing to recognize real code smells and just predict false anyway.