CO3091 - Computational Intelligence and Software Engineering

Lecture 20

Software Defect Prediction and Class Imbalance Learning

Leandro L. Minku

Image from: http://blog.qatestlab.com/wp-content/uploads/2012/04/Preventing-of-Software-Bugs-and-Bug-Sources-550x158.gif?width=550

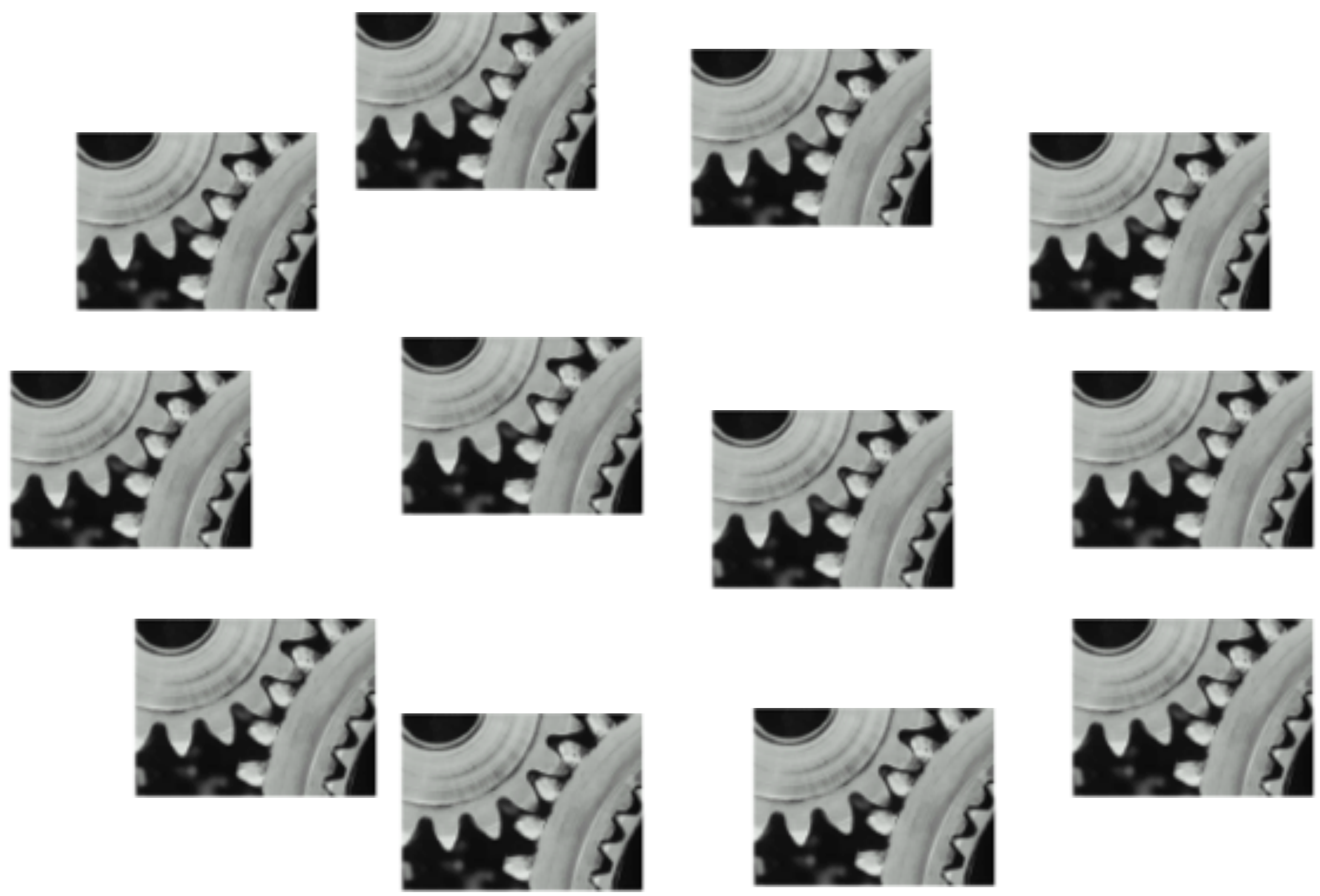
Overview

- What is software defect prediction?
- What is class imbalance?
- How to deal with class imbalance?
- How to evaluate predictive models when there is class imbalance?

Software Defect Prediction

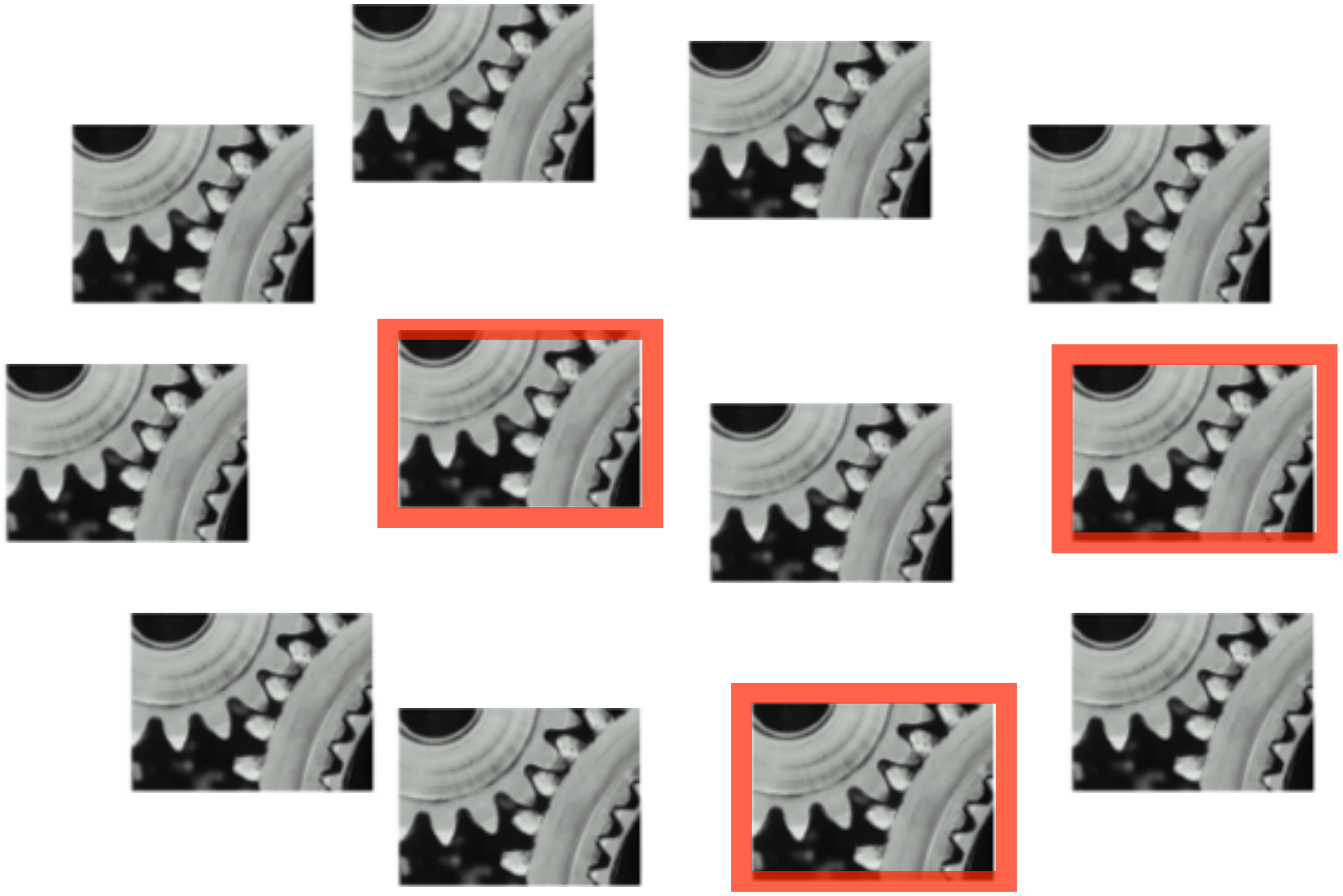
Software is composed of several components.

Testing all these components can be very expensive.



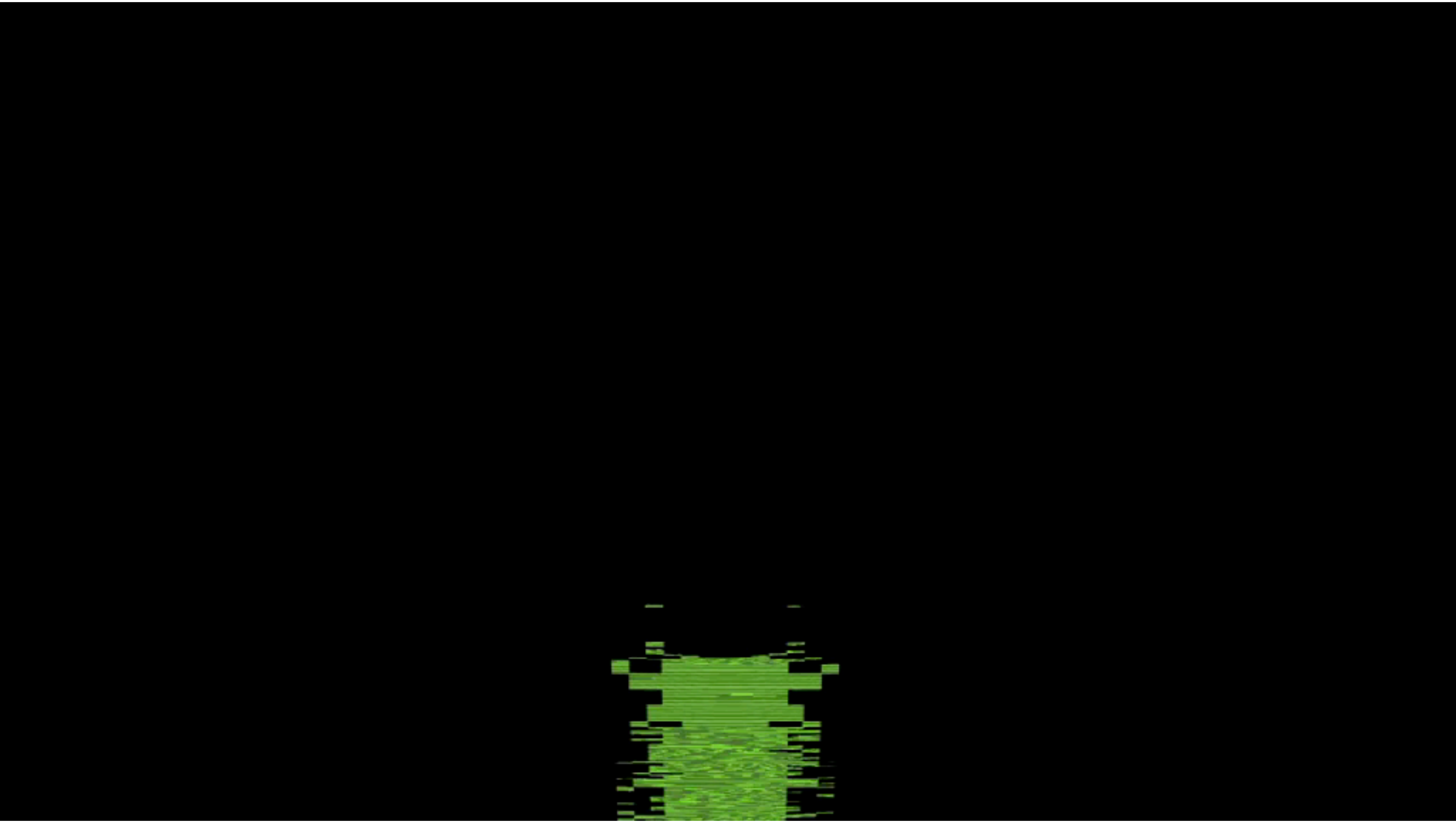
Software Defect Prediction

If we know which components are more likely to be defective (buggy), we can increase testing cost-effectiveness.



How to know which components are more likely to be defective (buggy)?

What about “hiring” existing bugs to work for us, to help us finding more bugs!?!?



[YouTube video posted by Atlassian: https://youtu.be/4Mz1estA4MA]

Software Defect Prediction as a Machine Learning Problem

Modules of previous versions of the software

Module id	x1 = branch count	x2 = LOC	x3 = halstead	…	y = defective ?
1	18	1000	1	…	No
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
…	…	…	…	…	…

→ Machine Learning Algorithm →

New module **x** for new version of the software →

→ Yes/No

What Input Attributes Can Be Used?

- Input attributes: several different metrics could be used to describe software modules. E.g.:
  - McCabe cyclomatic complexity
  - Halstead complexity measures
  - Number of lines of code (LOC)
  - Number of lines with comments
  - …

McCabe Cyclomatic Complexity

Measures the complexity of the code based on the number of linearly independent paths in the code flow graph.

- We will have more possible paths if we have more condition statements in our code.
  - McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe Cyclomatic Complexity

- Simple conditions are conditional statements without OR or AND. E.g.:
  - If (a > b) —> this counts as one simple condition
  - While (a > b) —> this counts as one simple condition
  - For (a=b; a > b; b++) —> this counts as one simple condition
  - Do…while (a > b) —> this counts as one simple condition

McCabe Cyclomatic Complexity

- For compound conditions, count each simple condition inside it. E.g.:
  - If (a > b) OR (a > 2) —> this counts as two simple conditions

```
if (a > b) OR (a > 2)
    statement
```

```
if (a > b)
    statement
else if (a > 2)
    statement
```

McCabe Cyclomatic Complexity

- For compound conditions, count each simple condition inside it. E.g.:
  - If (a > b) OR (a > 2) —> this counts as two simple conditions
  - If (a > b) AND (a > 2) —> this counts as two simple conditions

```
if (a > b) AND (a > 2)
    statement
```

```
if (a > b)
    if (a > 2)
        statement
```

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
else
    b = a;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
if (a > b || a > 1)
    a = b;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
if (a > b && a > 1)
    a = b;
int c = a + b;
```

McCabe cyclomatic complexity =
number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
if (2 * a > b)
    a = b;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
while (a > b)
    a--;
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

McCabe Cyclomatic Complexity — Example

```
switch (a) {
    case 1:
        a += 10;
        break;
    case 2:
        a += 30;
        break;
    case 3:
        a += 60;
        break;
    default:
        a += 1;
        break;
}
int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

Count number of cases.

McCabe Cyclomatic Complexity — Example

```
int a = 1;
int b = 2;
int c = 3;
try {
    a= 10;
} catch (ExceptionType1 name) {
    b = 10;
} catch (ExceptionType2 name) {
    c = 10;
}

int c = a + b;
```

McCabe cyclomatic complexity = number of simple conditions + 1.

McCabe cyclomatic complexity = ?

Count number of catches.

McCabe Cyclomatic Complexity

One could expect a more complex piece of code (with higher number of possible paths) to be more likely to contain defects.

However, we don't know how much more likely and whether / how it interacts with other metrics.

Halstead Complexity Measures

Measures the complexity of the code based on the number of operators and operands used in the code.

- n1 = number of distinct operators.
  - E.g., !=, !, %, /, *, +, &&, ||, etc.
- n2 = number of distinct operands.
  - E.g., identifiers that are not reserved words, constants (character, number, or string constants), type (bool, char, double, etc), etc.
- N1 = total number of the operators.
- N2 = total number of operands.

Halstead Complexity Measures

- n1 = number of distinct operators
- n2 = number of distinct operands
- N1 = total number of the operators
- N2 = total number of operands

But we don't know how much more likely and whether / how each of these metrics interacts with other metrics.

- Code vocabulary: $n = n1 + n2$
- Code length: $N = N1 + N2$

Once could expect that a larger piece of code is likely to contain more bugs.

- Volume: $V = N \log_2 n$
- Difficulty: $D = n1 / 2 * N2 / n2$
- …

Once could expect that a piece of code deemed more difficult is likely to contain more bugs.

Lines of Code and Comment

- Lines of Code (LOC): number of lines containing code.

Once could expect that a larger piece of code is likely to contain more bugs.

- Lines of Comment: number of lines containing comments.

Once could expect that a more commented piece of code is likely to contain less bugs.

But we don't know how much more likely, and whether / how these metrics interact with other metrics.

Which Classifier to Use?

- Naive bayes has been showing to perform well in comparison with other approaches.
- However, it still struggles because software defect prediction is a class imbalanced problem.

What is Class Imbalance?

A machine learning problem is class imbalanced when there are much less examples of one or more classes than examples of at least one of the other class / classes.

In software defect prediction, there are typically much more examples of non-defective modules than defective ones.

In this module, we will consider the scenario where there is only 2 classes: one majority and one minority.

Effect of Class Imbalance

- Class imbalance makes it difficult for machine learning approaches to recognise examples of the minority class.
- In the worst case scenario, the classifier would classify all new instances as belonging to the majority class.
- In software defect prediction, this would mean that all new modules would be classified as non-defective!

Potential Solutions

- Undersampling:
  - Instead of using the whole training set to produce the predictive model, use a sample of this training set.
  - Training sample:
    - Minority class: get all training examples of the minority class.
    - Majority class: randomly take n examples of the majority class, where n is the number of minority class examples.
- Problem: loose a lot of information. This may be ok if the data set is very large or easy to learn.

Original training set

Module id	x1 = branch count	x2 = LOC	x3 = halstead	…	y = defective ?
1	18	1000	1	…	No
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
4	25	100	3	…	No
5	50	500	4	…	No
6	4	30	3	…	No
7	25	2000	2	…	No
8	28	3000	5	…	No
9	13	1000	10	…	No
10	25	500	12	…	No

Training Sample

Module id	x1 = branch count	x2 = LOC	x3 = halstead	…	y = defective ?
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
6	4	30	3	…	No
10	25	500	12	…	No

Potential Solutions

- Oversampling:
  - Instead of using the whole training set to produce the predictive model, use a sample of this training set.
  - Training sample:
    - Majority class: get all training examples of the majority class.
    - Minority class: consider that n is the number of majority class examples. Randomly select n examples from the minority class.
- Problems:
  - Increases chances of overfitting the minority class, as it will produce several copies of this class.
  - Increase training time.
  - Does not really acquire extra information about the class boundaries.

Original training set

Module id	x1 = branch count	x2 = LOC	x3 = halstead	…	y = defective?
1	18	1000	1	…	No
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
4	25	100	3	…	No
5	50	500	4	…	No
6	4	30	3	…	No
7	25	2000	2	…	No
8	28	3000	5	…	No
9	13	1000	10	…	No
10	25	500	12	…	No

Training Sample

Module id	x1 = branch count	x2 = LOC	x3 = halstead	…	y = defective?
1	18	1000	1	…	No
4	25	100	3	…	No
5	50	500	4	…	No
6	4	30	3	…	No
7	25	2000	2	…	No
8	28	3000	5	…	No
9	13	1000	10	…	No
10	25	500	12	…	No
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
2	30	900	10	…	Yes
2	30	900	10	…	Yes
3	20	5000	3	…	Yes
3	20	5000	3	…	Yes
2	30	900	10	…	Yes
2	30	900	10	…	Yes

Evaluating Predictive Performance

- Classification error (or accuracy) is inadequate when there is class imbalance.

  Classification error = $\frac{1}{n}\sum_{i=1}^{n}$ ($y_i \neq y_i'$)

- Consider the following scenario:
  - 10 examples of the minority class.
  - 990 examples of the majority class.
  - Predictive model predicts all examples as being of the majority class.

Classification error = 10 / 1000 = 1%

Classification accuracy = 100% -1% = 99%

Evaluating Predictive Performance — Confusion Matrix

Predicted class

Actual class	Positive	Negative
Positive	True positive	False negative
Negative	False positive	True negative

$$\text{True positive rate} = \frac{\text{Number of true positives}}{\text{Number of examples whose actual class is positive}}$$

Percentage of positive examples that have been correctly classified.

Evaluating Predictive Performance — Confusion Matrix

Predicted class

Actual class	Positive	Negative
Positive	True positive	False negative
Negative	False positive	True negative

$$\text{True negative rate} = \frac{\text{Number of true negatives}}{\text{Number of examples whose actual class is negative}}$$

Percentage of negative examples that have been correctly classified.

Evaluating Predictive Performance — Confusion Matrix

Predicted class

Actual class	Positive	Negative
Positive	True positive	False negative
Negative	False positive	True negative

$$\text{False positive rate} = \frac{\text{Number of false positives}}{\text{Number of examples whose actual class is negative}}$$

Percentage of negative examples that have been incorrectly classified.

Evaluating Predictive Performance — Confusion Matrix

Predicted class

Actual class	Positive	Negative
Positive	True positive	False negative
Negative	False positive	True negative

$$\text{False negative rate} = \frac{\text{Number of false negatives}}{\text{Number of examples whose actual class is positive}}$$

Percentage of positive examples that have been incorrectly classified.

Example

- Consider the following scenario:
    - 10 examples of the class positive.
    - 990 examples of the class negative.
    - Predictive model predicts all examples as being of the negative class.

$$\text{True negative rate} = \frac{\text{Number of true negatives}}{\text{Number of examples whose actual class is negative}}$$

$$\text{True negative rate} = \frac{990}{990} = 1 = 100\%$$

Example

- Consider the following scenario:
    - 10 examples of the class positive.
    - 990 examples of the class negative.
    - Predictive model predicts all examples as being of the negative class.

$$\text{False positive rate} = \frac{\text{Number of false positives}}{\text{Number of examples whose actual class is negative}}$$

$$\text{False positive rate} = \frac{0}{990} = 0 = 0\%$$

Example

- Consider the following scenario:
  - 10 examples of the class positive.
  - 990 examples of the class negative.
  - Predictive model predicts all examples as being of the negative class.

$$\text{True positive rate} = \frac{\text{Number of true positives}}{\text{Number of examples whose actual class is positive}}$$

$$\text{True positive rate} = \frac{0}{10} = 0 = 0\%$$

Example

- Consider the following scenario:
  - 10 examples of the class positive.
  - 990 examples of the class negative.
  - Predictive model predicts all examples as being of the negative class.

$$\text{False negative rate} = \frac{\text{Number of false negatives}}{\text{Number of examples whose actual class is positive}}$$

$$\text{False negative rate} = \frac{10}{10} = 1 = 100\%$$

Example

- Consider the following scenario:
  - 10 examples of the class positive.
  - 990 examples of the class negative.
  - Predictive model predicts all examples as being of the negative class.

↑ True negative rate = 100%

↓ False positive rate = 0%

↑ True positive rate = 0%

↓ False negative rate = 100%

Summary Of Last Three Lectures

- Naive Bayes has a probabilistic view of Machine Learning.
- Naive Bayes for classification problems:
  - Naive Bayes for categorical input attributes.
  - Naive Bayes for numerical input attributes.
- Naive Bayes is typically poor for regression problems.
- Naive Bayes has been used for software defect prediction.
- However, strategies to deal with class imbalance are recommended to be used in conjunction with Naive Bayes for this problem.

Further Reading

Zaheed Mahmood, David Bowes, Peter Lane and Tracy Hall.

What is the impact of imbalance on software defect prediction performance?

International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2015)

http://dl.acm.org/citation.cfm?id=2810150

Lab session today at 3pm!