MACHINE LEARNING IN AUTOMOTIVE SOFTWARE DEVELOPMENT

OPPORTUNITIES AND CHALLENGES

MIROSLAW STARON

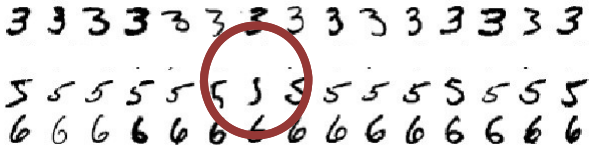
SOFTWARE CENTER,
SOFTWARE ENGINEERING
COMPUTER SCIENCE AND ENGINEERING
CHALMERS | GÖTEBORGS UNIVERSITET

Machine learning in the development of automotive software

- The concept of machine learning in software engineering
- Examples of code classification and defect clustering
- The future of applying machine learning to automotive software

Why ML is a paradigm shift…

5



This is number 5

There is:
- 60 % probability that this is number 5
- 30 % probability that this is number 3
- 10 % probability that this is number 1

Automation vs machine learning

Automation	Machine learning
• Input: manual task	• Input: manual task
• Output: computer program that executes the task	• Output: computer program that executes the task
• Development: programming (e.g. C++)	• Development: teaching (e.g. providing examples)
• Validation: testing – According to spec or not	• Validation: statistics – % of correctly classified instances – % of consistency

MACHINE LEARNING IN AUTOMOTIVE SE

EXAMPLE 1: FINDING CODING VIOLATIONS (E.G. MISRA)

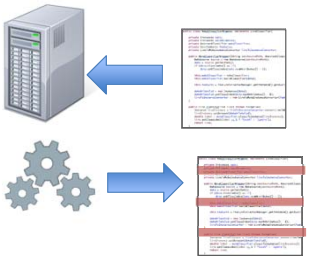
WORK DONE TOGETHER WITH DR. MIROSLAW OCHODEK, CHALMERS | UNIVERSITY OF GOTHENBURG & POZNAN UNIVERSITY OF TECHNOLOGY

Problem formulation

- Motivating example
  - Imagine we need to find MISRA violations
  - AND we do not know the grammar of the programming language
  - NEITHER we know exactly how to find the patterns manually
  - BUT we can provide an example of violations
- Example MISRA rules
  - Rule 2.2 (required): Source code shall only use /* … */ style comments.
  - Rule 2.3 (required): The character sequence /* shall not be used within a comment.
- Standard solution
  - MISRA C compliance checker (specific program)

Machine learning approach to MISRA compliance checks

```
brackets <= 0 AND
freq_override <= 0 AND
cs_case = false AND
class = false AND
freq_colon <= 0 AND
length <= 3: Ignore (979.0/2.0)
```

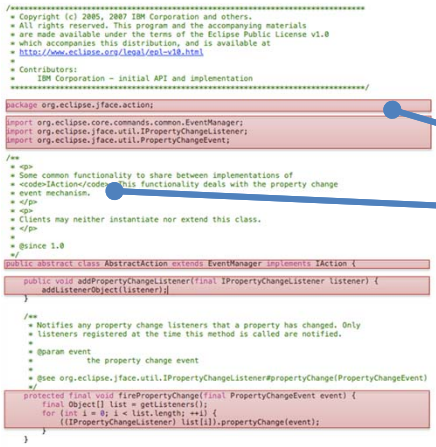


1. Mark example code where rules are violated

2. Let ML discover the pattern

3. Let ML apply this pattern on the whole code base

Pattern discovery – translating programs into ML format



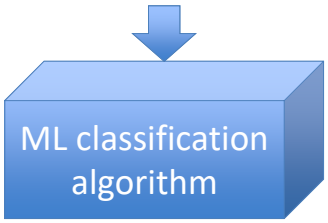
Features, attributes

#Characters	";"	"."	Comment	...	Decision
50	1	4	0	...	Count
13	0	1	1	...	Ignore
...	...	...	...	...	...

Each line is "encoded" into a format which a ML algorithm can understand

Pattern discovery – which lines violate the rule

#Characters	";"	"."	comment	...	Decision
50	1	4	0	...	Count
13	0	1	1	...	Ignore
...	...	...	...	...	...



ML algorithm finds patterns of which lines violate the rules

```
freq_semi_colon > 0 AND
comment = false: Count (534.0/3.0)

brackets <= 0 AND
freq_override <= 0 AND
cs_case = false AND
class = false AND
freq_colon <= 0 AND
length <= 3: Ignore (979.0/2.0)

comment = true: Ignore (517.0/5.0)

: Count (372.0)

Number of Rules  :     4
```

Finding code violations in the entire code base

```
freq_semi_colon > 0 AND
comment = false: Count (534.0/3.0)

brackets <= 0 AND
freq_override <= 0 AND
cs_case = false AND
class = false AND
freq_colon <= 0 AND
length <= 3: Ignore (979.0/2.0)

comment = true: Ignore (517.0/5.0)

: Count (372.0)

Number of Rules  :     4
```

```
/**
 * Notifies any property change listeners that a property has changed. Only
 * listeners registered at the time this method is called are notified. This
 * method avoids creating an event object if there are no listeners
 * registered, but calls
 * <code>firePropertyChange(PropertyChangeEvent)</code> if there are.
 *
 * @param propertyName
 *            the name of the property that has changed
 * @param oldValue
 *            the old value of the property, or <code>null</code> if none
 * @param newValue
 *            the new value of the property, or <code>null</code> if none
 *
 * @see org.eclipse.jface.util.IPropertyChangeListener#propertyChange(PropertyChangeEvent)
 */
protected final void firePropertyChange(final String propertyName,
        final Object oldValue, final Object newValue) {
    if (isListenerAttached()) {
        firePropertyChange(new PropertyChangeEvent(this, propertyName,
                oldValue, newValue));
    }
}

public void removePropertyChangeListener(
        final IPropertyChangeListener listener) {
    removeListenerObject(listener);
}

}
```

PRACTICE:

FINDING CODING VIOLATIONS IN INDUSTRIAL CODE

Analysis: one declaration per line

- Rule:

Only one variable declaration per line of code

Good example:
```
uint32_t * foo;
uint32_t bar;
```

Bad example:
```
uint32_t* foo, bar;
```

- Method
  - Use this example to teach ML -> analyze one component
  - Take false-positives -> add to teaching set -> repeat

- Exit criteria: fewer false-positives than true positives

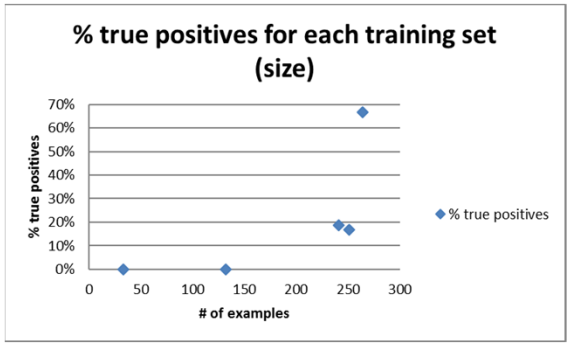
Results

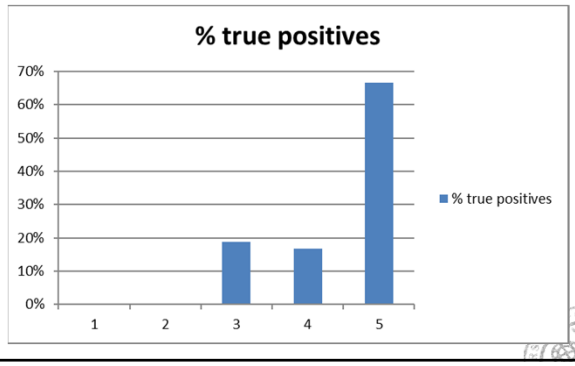
The size of the training set (example) is one of two major factors determining accuracy.

The other factor is the algorithm (not shown in the diagram)

The first trials did not find anything

Trial #5 resulted in finding
- all violations
- one false-positive (non-violation)





Practical implications

- Analyzing 3 MLOC takes ca. 20 minutes
- One needs to run 1-5 trials to find the right learning set
  - We can automate that process too
- Since this is based on simple scripts, we can build this as part of tool chains

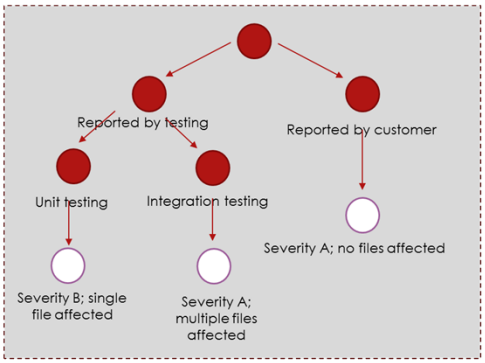
MACHINE LEARNING IN AUTOMOTIVE SE

EXAMPLE 2: WHICH DEFECT SHOULD WE FIX FIRST?

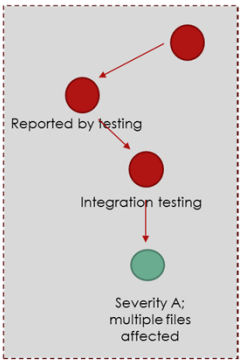
WORK DONE TOGETHER WITH DR. MIROSLAW OCHODEK, CHALMERS | UNIVERSITY OF GOTHENBURG & POZNAN UNIVERSITY OF TECHNOLOGY

Which defect should be prioritized?

Our knowledge discovery algorithms find patterns in defect descriptions and clusters them to find which defects affect many files



Our tree map describes the rules for assigning defect priorities and effort



Product management gets to decide which defects to prioritize

Defects database

Product: **large > 10 MLOC**
Period: 2010-17
Total records: ~14K
Different filters …

Main tools:

Problem formulation

- How can we predict the severity of the defect?
  - Imagine we discover a bug
  - We need to quickly assess if this bug should be fixed in this release or not
  - We need to assess if this is going to be a lot of work
- Today's solution
  - Architect and quality engineer make the assessment

Mining association rules for defect prioritization

{phaseFound=**PRODUCT VALIDATION TESTING**
answerCode=**B2 - To be corrected in this release**,
Importance=**30**}

=> {Severity=A}

supp=0.0016 confidence=0.83 lift=9.95

{phaseFound=**Customer**,
answerCode=**B2 - To be corrected in this release**,
submittedOnSystemPart=**VERY IMPORTANT PART**}

=> {Severity=A}

supp=0.0011 confidence=0.88 lift=10.45

{phaseFound=**PRODUCT VALIDATION TESTING**
answerCode=**B2 - To be corrected in this release,**
FollowUpOn=,
ClonedToAllReleases=**YES**
submittedOnSystemPart=**LI**}

=> {Severity=A}

supp=0.0013 confidence=0.80 lift=9.55

Can we distinguish Severity A defects from others?

Decision tree: J48 (Weka) + ClassBalancer

J48 pruned tree (example)

VerificationLevelRequired =
| phaseFound = : A (1.62)
| phaseFound = Customer: A (60.88/12.3)
| phaseFound = Design Test (DT): Other (38.48/8.1)
| phaseFound = Document review (CPI): Other (11.75/1.62)
| phaseFound = FOA: A (28.66/10.85)
| phaseFound = Function Test (FT): Other (228.56/40.48)
| phaseFound = PRODUCT VALIDATION TEST: Other (6.86/3.24)
| phaseFound = INTERNAL TEST: Other (5.79)
| phaseFound = Requirement Review: Other (5.06)
| phaseFound = System Test (ST): Other (148.34/61.53)
VerificationLevelRequired = Customer: A (3.24)
VerificationLevelRequired = Design Test (DT): A (22.67)
VerificationLevelRequired = Function Test (FT): A (66.39)
VerificationLevelRequired = PRODUCT VALIDATION TEST: A (6.48)
VerificationLevelRequired = Requirement Review: A (4.86)
VerificationLevelRequired = System Test (ST): A (66.39)

Number of Leaves : 16
Size of the tree : 18

Accuracy = 77.70 %

True Positive(A) = 0.642
False Positive(A) = 0.088
F-Score(A) = 0.742

True Positive (Other) = 0.912
False Positive (Other) = 0.358
F-Score(B) = 0.804

Can we distinguish Severity A defects from others?

- Potentially valuable features (using filter):
  - phaseFound
  - Keywords headline: branch, test case, underscore
  - Kyewords desc: descr_info, descr_requirement, descr_test, descr_debug, descr_log…
  - DaysUntilAssigned
- Records = **6342**

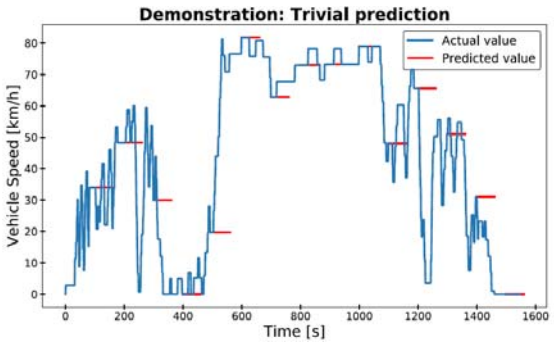
- Features = **49**
  - Directly available
  - Time periods between changes of states
  - Keywords appearance in description and header

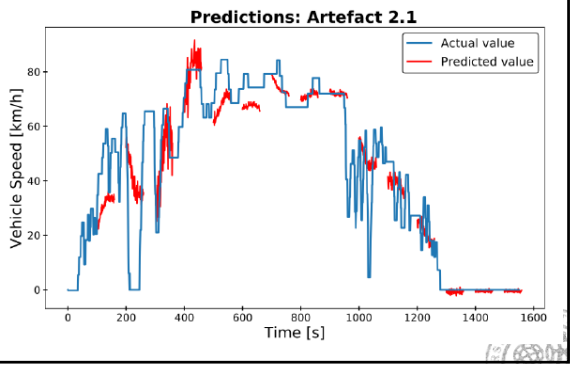
EXAMPLE 3: PREDICTING ENERGY USAGE BASED ON DRIVING PROFILES

MASTER THESIS WORK OF ERIK TEVELL, CHALMERS

Predicting vehicle speed for electrical vehicles

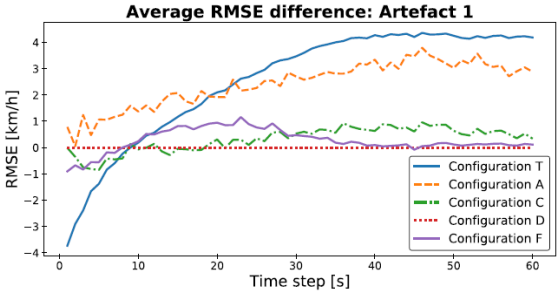
- Predicting speed is needed to
  - Predict energy consumption
  - Predict the available driving distance
- Using formal models is not possible because of the variability of the scenarios
- Machine learning can help to predct the speed

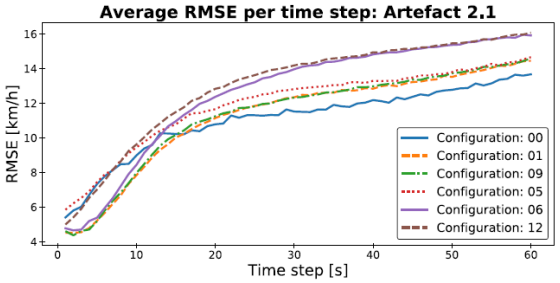




Accuracy of complex model outruns trivial models?

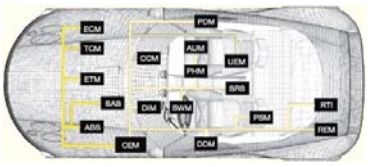
- Trivial models are not always the worst ones
- ML can be costly in terms of time, and is much better in certain scenarios, but worse in other scenarios

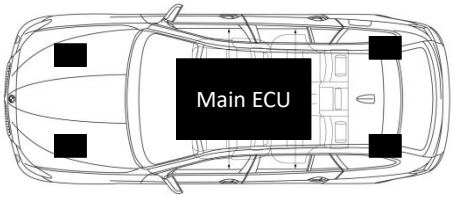




CHALLENGES

Integrated architectures instead of distributed architectures





Distributed architecture == multiple data sets

Multiple data sets == multiple formats

Multiple data sets == multiple keys

Multiple data sets == multiple time stamps

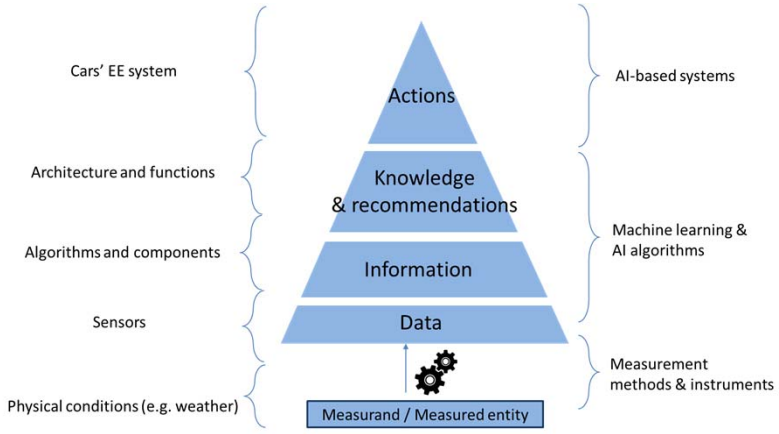
Integrated architecture == integrated data sets

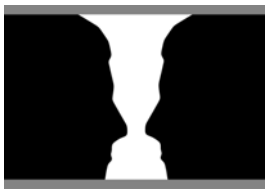
Integrated data sets == single format

Integrated data sets == one key

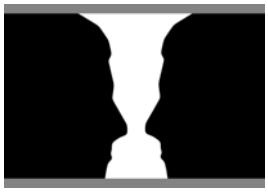
Integrated data sets == one time stamp

Data quality limits ML / AI





Label 1: Two persons



Label 2: Vase

Verification and validation of machine learning algorithms

- White-box ML testing can still be assessed using coverage-based criteria

```
J48 pruned tree (example)
------------------
VerificationLevelRequired =
|   phaseFound = : A (1.62)
|   phaseFound = Customer: A (60.88/12.3)
|   phaseFound = Design Test (DT): Other (38.48/8.1)
|   phaseFound = Document review (CPI): Other (11.75/1.62)
|   phaseFound = FOA: A (28.66/10.85)
|   phaseFound = Function Test (FT): Other (228.56/40.48)
|   phaseFound = PRODUCT VALIDATION TEST: Other (6.86/3.24)
|   phaseFound = INTERNAL TEST: Other (5.79)
|   phaseFound = Requirement Review: Other (5.06)
|   phaseFound = System Test (ST): Other (148.34/61.53)
VerificationLevelRequired = Customer: A (3.24)
VerificationLevelRequired = Design Test (DT): A (22.67)
VerificationLevelRequired = Function Test (FT): A (66.39)
VerificationLevelRequired = PRODUCT VALIDATION TEST: A (6.48)
VerificationLevelRequired = Requirement Review: A (4.86)
VerificationLevelRequired = System Test (ST): A (66.39)

Number of Leaves :      16
Size of the tree :      18
```

- Coverage-based measurement of test progress makes no sense for black-box ML algorithms

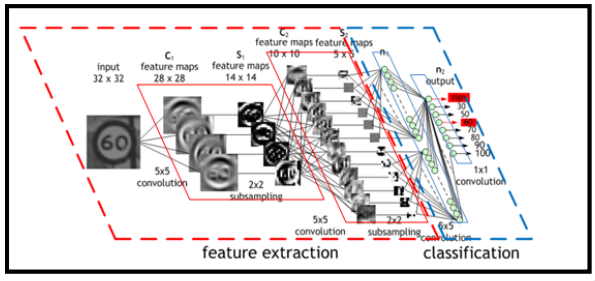
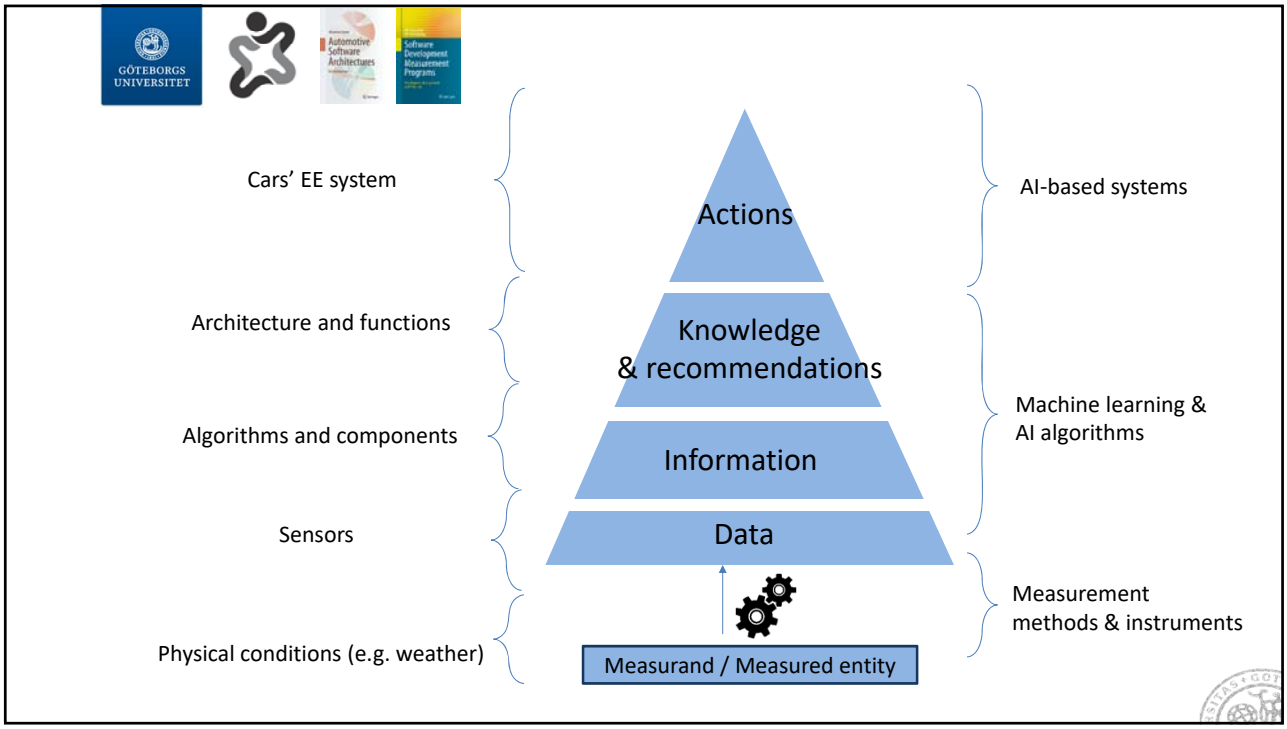


Figure source: NVIDIA

Summary

Opportunities and challenges

- Opportunities
  - Safe automation of common tasks (e.g. defect prioritization)
  - White-box, traceable recognition of patterns (e.g. source code violations)
  - Predictions with better accuracy (e.g. distance prediction)
  - <all kinds of optimizations>
- Challenges
  - High quality machine learning required integrated architecture
  - Data quality is more important than algorithm quality
  - Coverage-based safety analysis needs to be replaced with debugging of ”boundary” scenarios



Probabilistic processes mixed with deterministic processes

- Today's algorithms are based on control loops and deterministic behavior
  - Recognizing a speed limit of 60 km/h always leads to the same behavior
  - Failure to recognize the limit leads to default behavior: "ignore"

- Tomorrow's algorithms will need to work with probabilities
  - Recognizing that the road sign is 60% speed limit of 60 km/h and 40% speed limit of 80 km/h needs additional sources to validate the data
    - Enhanced data quality (multiple sources)
    - Risk management – what happens if we reduce the speed to 60 km/h while in a traffic flow with other cars driving 80 km/h