Vulnerability Prediction in Android Apps

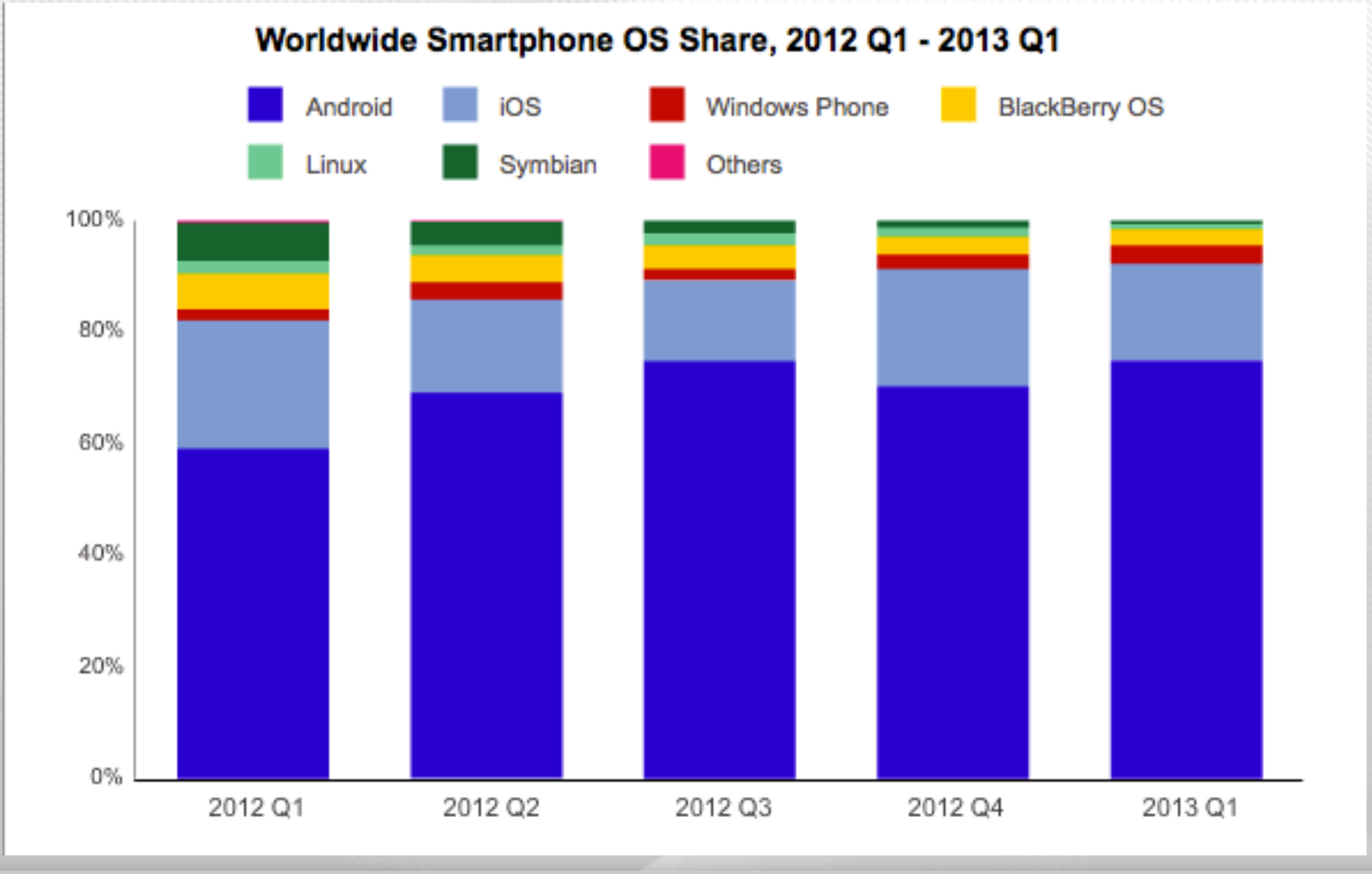
***Aram Hovsepyan*[1], *Riccardo Scandariato*[1], *James Walden*[2], *Viet Hung Nguyen*[3] *Wouter Joosen*[1]**

[1] *iMinds-DistriNet, Katholieke Universiteit Leuven*

[2] *Northern Kentucky University*, [3] *University of Trento*

Android apps are an attractive target

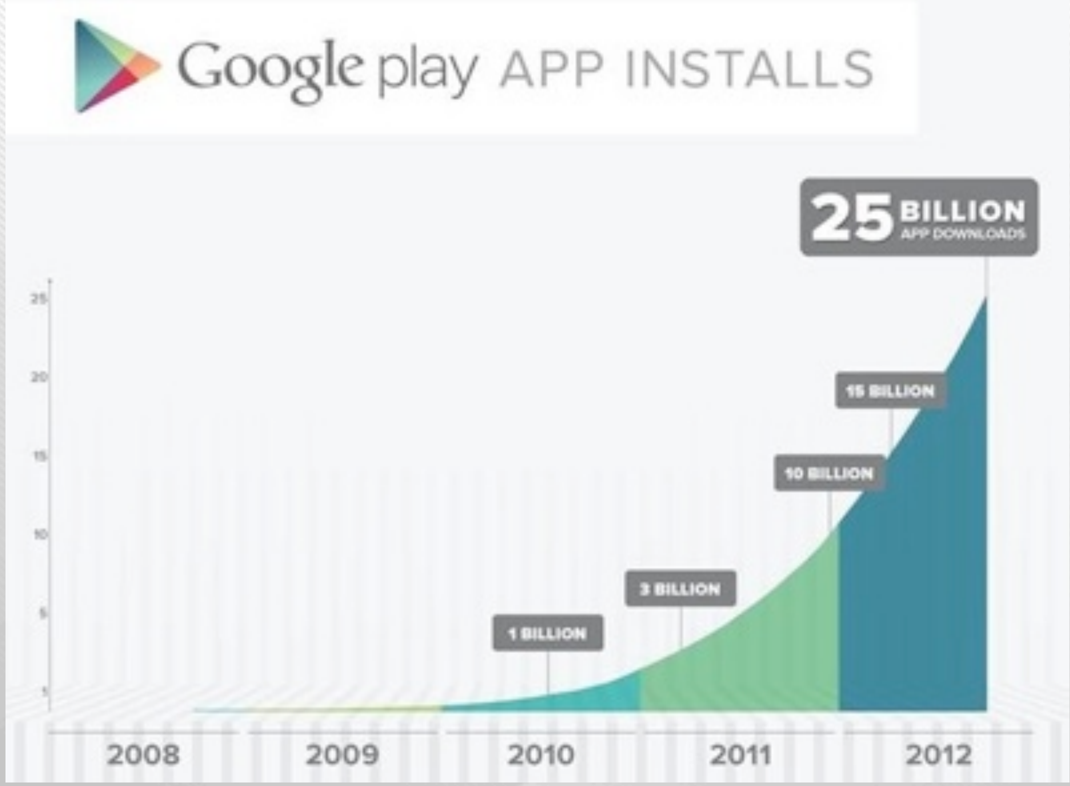
Android has 75% market share as of Q1 2013 [IDC]



Android apps are an attractive target

DistriNet

Google play has over 775K apps and over 48B total installs [IDC, Google I/O keynote]



Android apps are an attractive target

DistriNet

App security is not guaranteed by the platform provider

- Apps that are well intended, but not exploit free

A single vulnerability could affect a massive number of users

Not yet much explored

- Focused on Mozilla Firefox / RHEL

How to find vulnerabilities?

How to find vulnerabilities?

DistriNet

Code inspection

- Manual verification is not feasible
- Not all apps can afford security experts
- Even security experts cannot analyze every line of code

How to find vulnerabilities?

DistriNet

Code inspection

- Manual verification is not feasible
- Not all apps can afford security experts
- Even security experts cannot analyze every line of code

Penetration testing / security testing

How to find vulnerabilities?

DistriNet

Code inspection

- Manual verification is not feasible
- Not all apps can afford security experts
- Even security experts cannot analyze every line of code

Penetration testing / security testing

Static code analysis

How to find vulnerabilities?

DistriNet

Code inspection

- Manual verification is not feasible
- Not all apps can afford security experts
- Even security experts cannot analyze every line of code

Penetration testing / security testing

Static code analysis

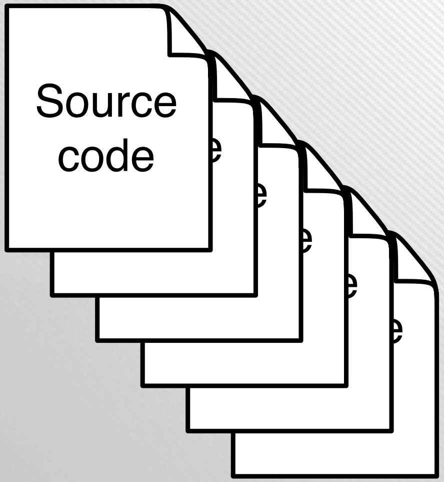
Magic

- Vulnerability prediction models

Vulnerability prediction model

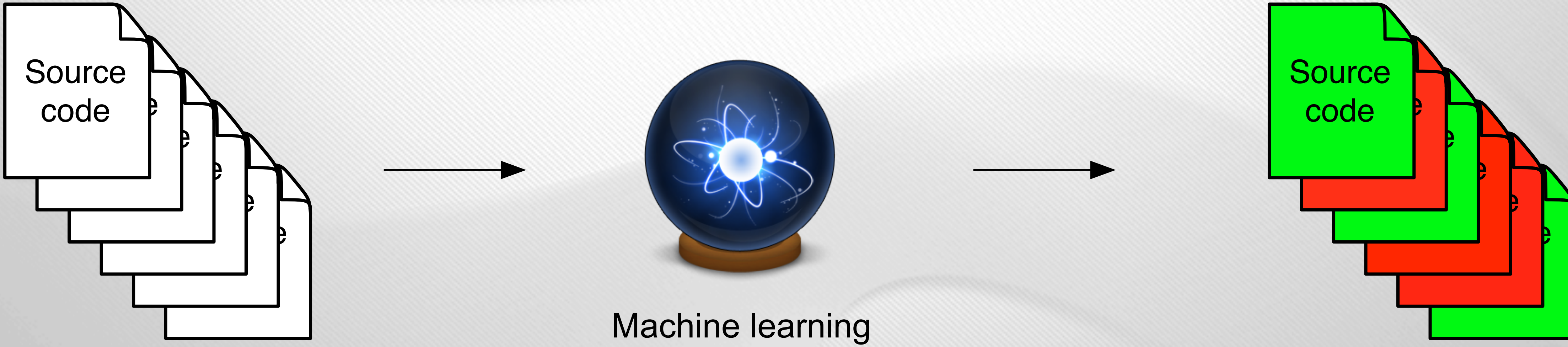
DistriNet

Vulnerability prediction model



Vulnerability prediction model

Vulnerability prediction model



Our research

Predict vulnerable Java files in Android apps!

Predict vulnerable C++ components in Chrome/Firefox

- ongoing

Predict vulnerable PHP files

- summer work

Outline

DistriNet

Existing tools and techniques

- Vulnerability prediction models

Our approach

Results

Conclusions and future research

Vulnerability prediction models

Vulnerability prediction models

Vulnerability prediction models

Vulnerability prediction models

DistriNet

Start from a hunch = feature

- e.g., larger components are more likely to be vulnerable

Vulnerability prediction models

Start from a hunch = feature

- e.g., larger components are more likely to be vulnerable

Fetch the features from the components

- e.g., calculate the size for each component

Vulnerability prediction models

DistriNet

Start from a hunch = feature

- e.g., larger components are more likely to be vulnerable

Fetch the features from the components

- e.g., calculate the size for each component

Determine the vulnerabilities

- e.g., National Vulnerability Database, MFSA

Vulnerability prediction models

DistriNet

Start from a hunch = feature

- e.g., larger components are more likely to be vulnerable

Fetch the features from the components

- e.g., calculate the size for each component

Determine the vulnerabilities

- e.g., National Vulnerability Database, MFSA

Investigate the correlation

- Use machine learning techniques

Vulnerability prediction models

Vulnerability prediction models

DistriNet

Typical “hunches”

- Use size and complexity metrics
- Leverage developer activity metrics
- Leverage code churn metrics
- Leverage design churn metrics
- Number of import statements

Vulnerability prediction models

DistriNet

Typical “hunches”

- Use size and complexity metrics
- Leverage developer activity metrics
- Leverage code churn metrics
- Leverage design churn metrics
- Number of import statements

Inspired on the defect prediction work

- Vulnerabilities are actually defects, but much more scarce (“needle in a haystack”)

Vulnerability prediction models

DistriNet

The existing models are fairly complex

- Typically several versions are necessary to collect all metrics
- Developer activity metrics are required
- Code evolution metrics are required

Biased to the underlying "hunch" of the researcher

Outline

DistriNet

Existing tools and techniques

- Static code analysis
- Vulnerability prediction using metrics

Our approach

Results

Conclusions and future research

Our approach

DistriNet

Use the source code itself in a tokenized form

Use the token frequency as features

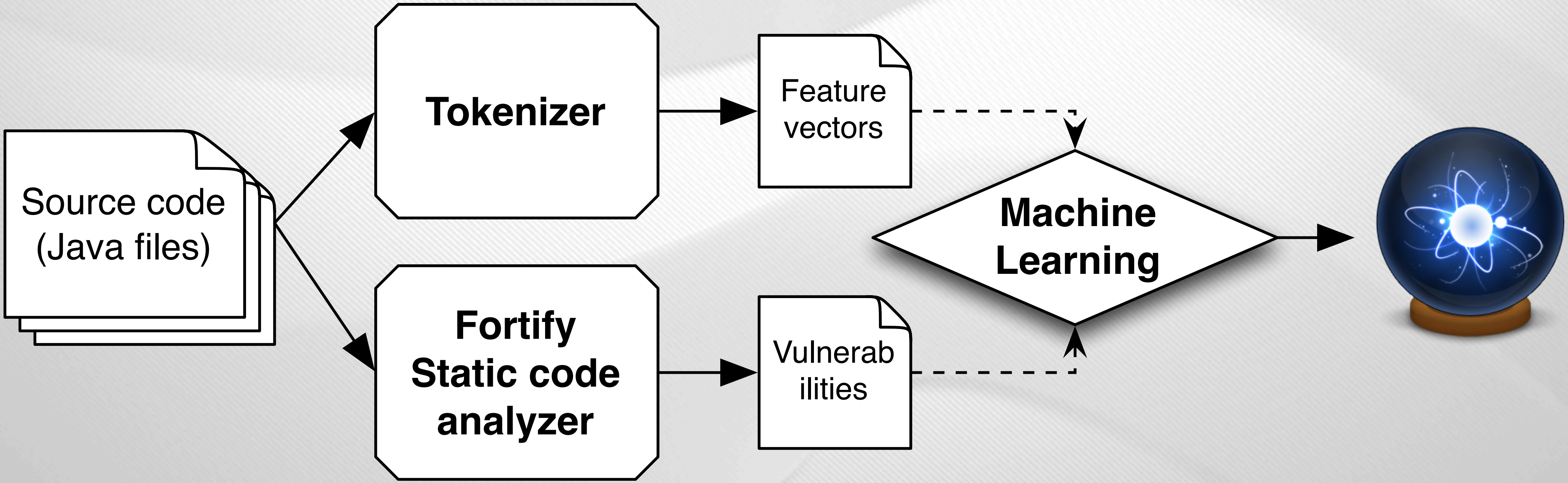
- Simplicity
- No explicit assumptions regarding the code characteristics

#text analysis

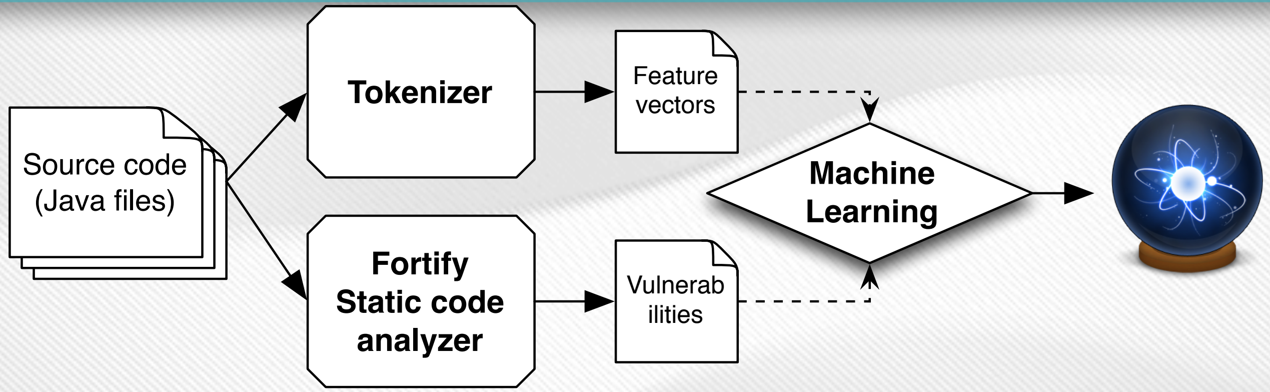
#SPAM filtering

#machine learning

Our approach

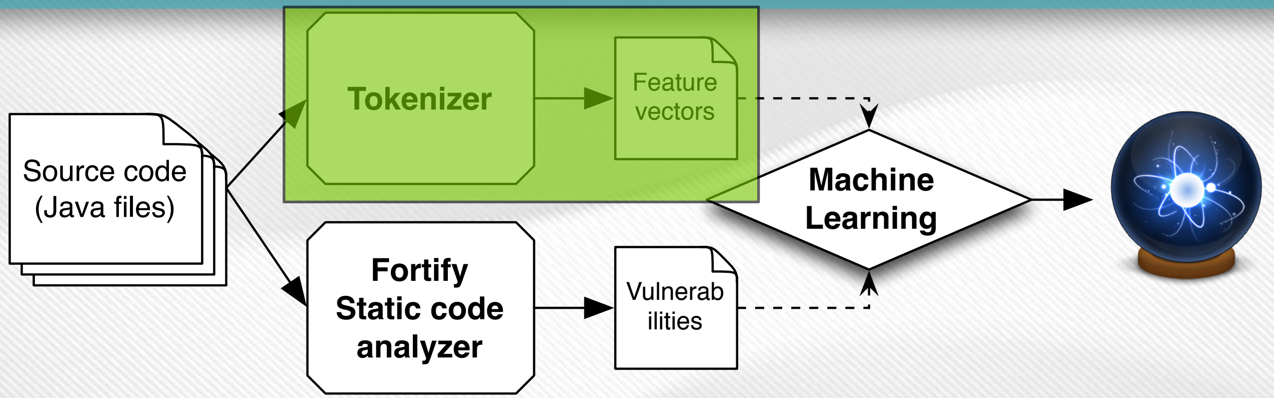


Tokenizer



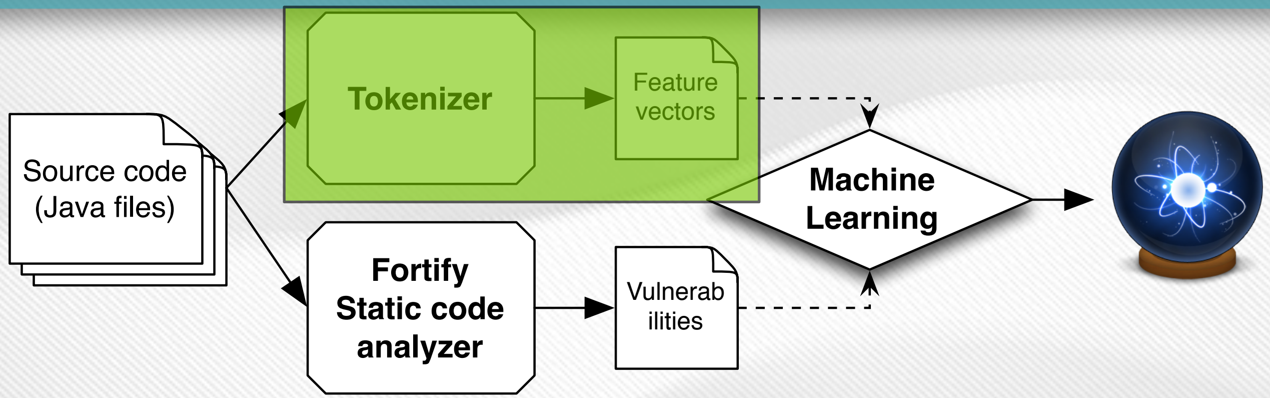
Monday 10 June 13 week

Tokenizer



Monday 10 June 13 week

Tokenizer



Transform each source code token into a feature vector

- each token ("monogram") is a feature
- tokenize by delimiters, mathematical and logical operations
  . , ! < > [ ] = + - ^ * / etc.
- each feature has a count assigned to it

Feature vector

Feature vector

```
package com.fsck.k9;
import android.text.util.Rfc822Tokenizer;
import android.widget.AutoCompleteTextView.Validator;
public class EmailAddressValidator implements Validator
{
    public CharSequence fixText(CharSequence invalidText)
    {
        return "";
    }
    public boolean isValid(CharSequence text)
    {
        return Rfc822Tokenizer.tokenize(text).length > 0;
    }
}
```

Feature vector

DistriNet

```
package com.fsck.k9;
import android.text.util.Rfc822Tokenizer;
import android.widget.AutoCompleteTextView.Validator;
public class EmailAddressValidator implements Validator
{
    public CharSequence fixText(CharSequence invalidText)
    {
        return "";
    }
    public boolean isValid(CharSequence text)
    {
        return Rfc822Tokenizer.tokenize(text).length > 0;
    }
}
```

package: 1

Feature vector

DistriNet

```
package com.fsck.k9;
import android.text.util.Rfc822Tokenizer;
import android.widget.AutoCompleteTextView.Validator;
public class EmailAddressValidator implements Validator
{
    public CharSequence fixText(CharSequence invalidText)
    {
        return "";
    }
    public boolean isValid(CharSequence text)
    {
        return Rfc822Tokenizer.tokenize(text).length > 0;
    }
}
```

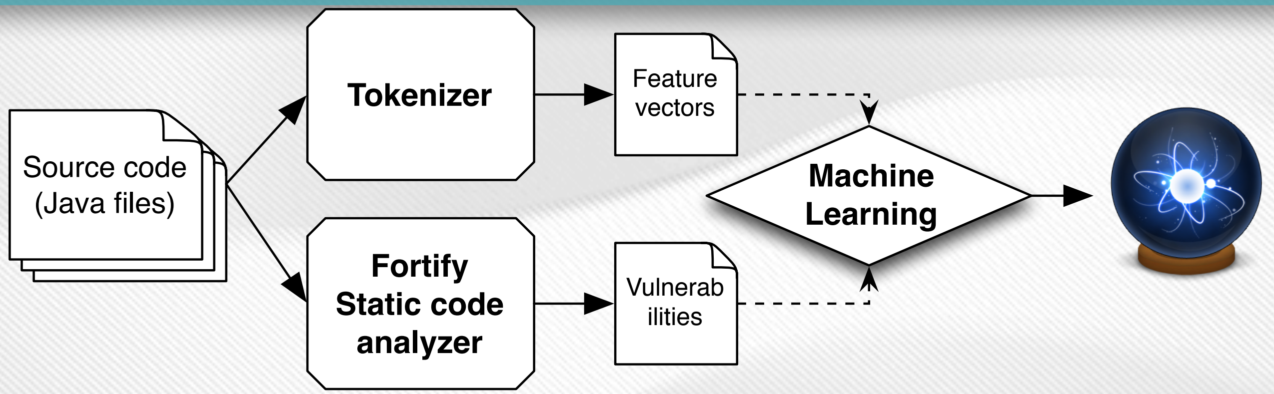
package: 1, com: 1

Feature vector

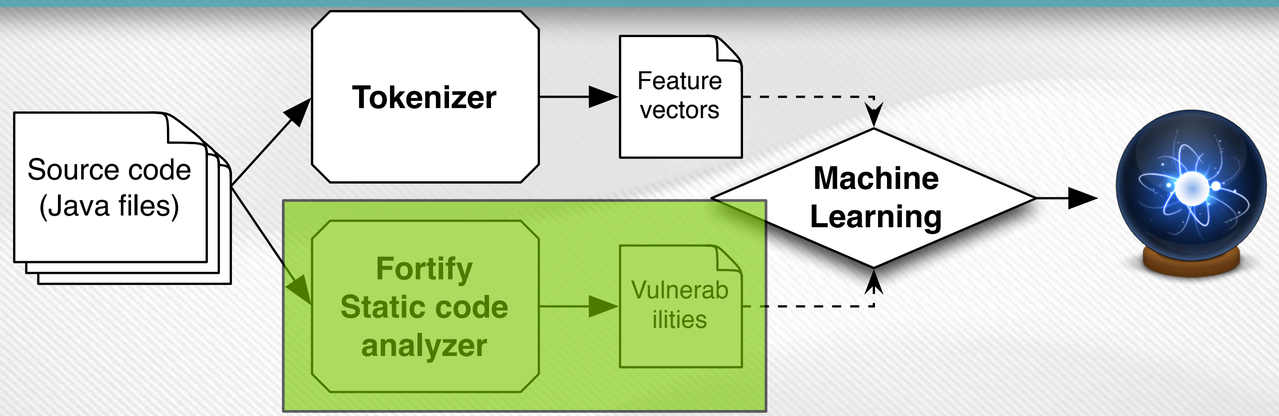

```java
package com.fsck.k9;
import android.text.util.Rfc822Tokenizer;
import android.widget.AutoCompleteTextView.Validator;
public class EmailAddressValidator implements Validator
{
    public CharSequence fixText(CharSequence invalidText)
    {
        return "";
    }
    public boolean isValid(CharSequence text)
    {
        return Rfc822Tokenizer.tokenize(text).length > 0;
    }
}
```

package: 1, com: 1, fsck: 1, k9: 1, import: 2, android: 2, text: 2, util: 1, Rfc822Tokenizer: 2, widget: 1, AutoCompleteTextView:1, Validator: 2, public: 3, class: 1, EmailAddressValidator: 1, implements: 1, CharSequence: 2, fixText: 1, invalidText: 1, return: 2, tokenize: 1, length: 1

Vulnerability assignment

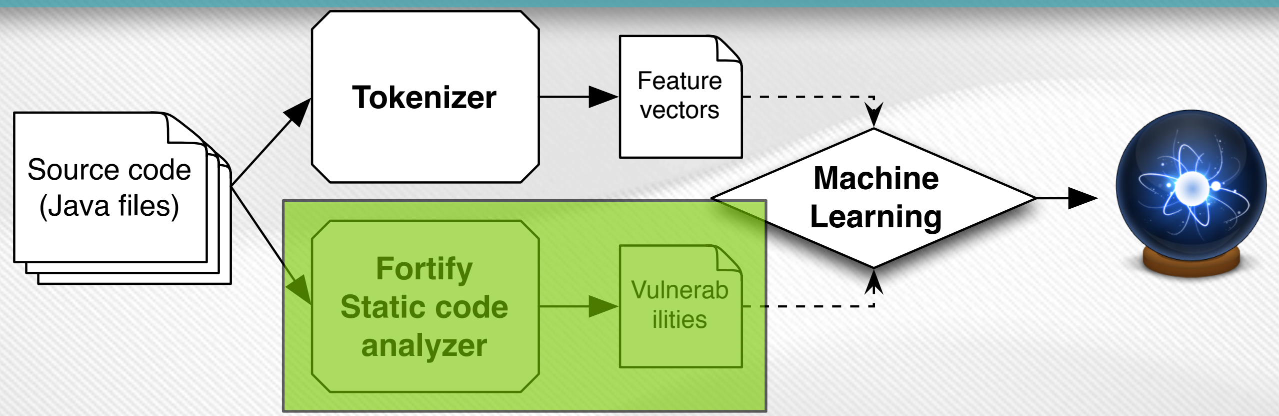


Vulnerability assignment



Monday 10 June 13 week

Vulnerability assignment

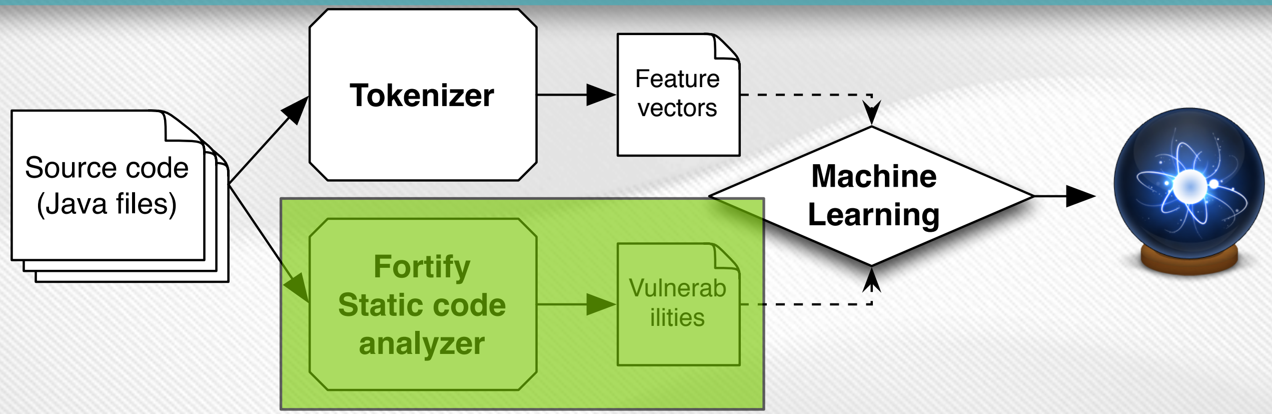


Assign vulnerability to each Java file

- use Fortify (static code analyzer) for this task
- each file is either *vulnerable* or *clean*

Monday 10 June 13 week

Vulnerability assignment

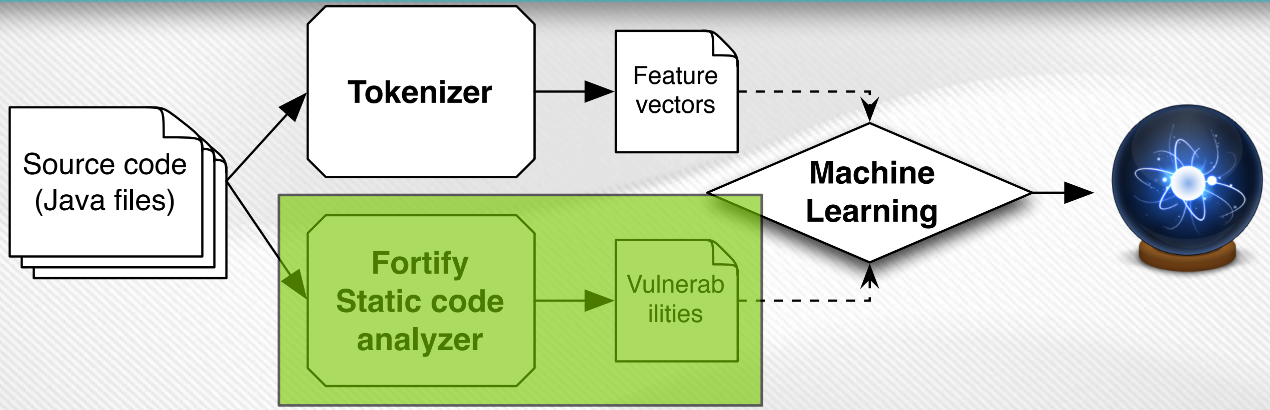


Assign vulnerability to each Java file

- use Fortify (static code analyzer) for this task
- each file is either *vulnerable* or *clean*

package: 1, com: 1, fsck: 1, k9: 1, import: 2, android: 2, text: 2, util: 1, Rfc822Tokenizer: 2, widget: 1, AutoCompleteTextView:1, Validator: 2, public: 3, class: 1, EmailAddressValidator: 1, implements: 1, CharSequence: 2, fixText: 1, invalidText: 1, return: 2, tokenize: 1, length: 1

Vulnerability assignment



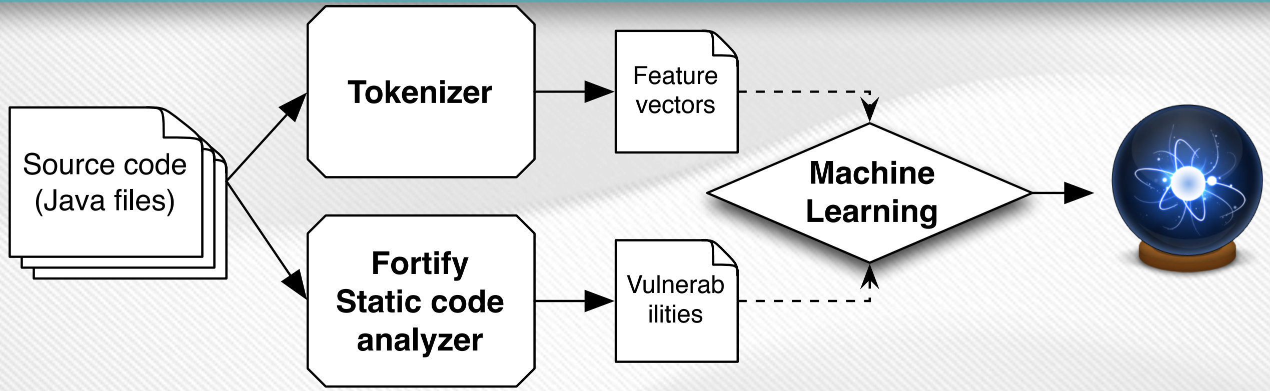
Assign vulnerability to each Java file

- use Fortify (static code analyzer) for this task
- each file is either *vulnerable* or *clean*

package: 1, com: 1, fsck: 1, k9: 1, import: 2, android: 2, text: 2, util: 1, Rfc822Tokenizer: 2, widget: 1, AutoCompleteTextView:1, Validator: 2, public: 3, class: 1, EmailAddressValidator: 1, implements: 1, CharSequence: 2, fixText: 1, invalidText: 1, return: 2, tokenize: 1, length: 1, vulnerability: 0

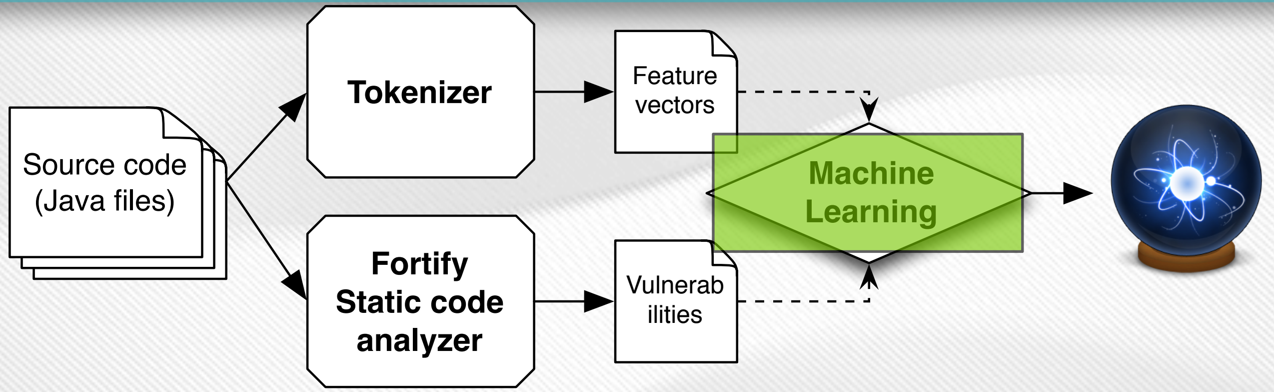
Monday 10 June 13 week

Machine learning



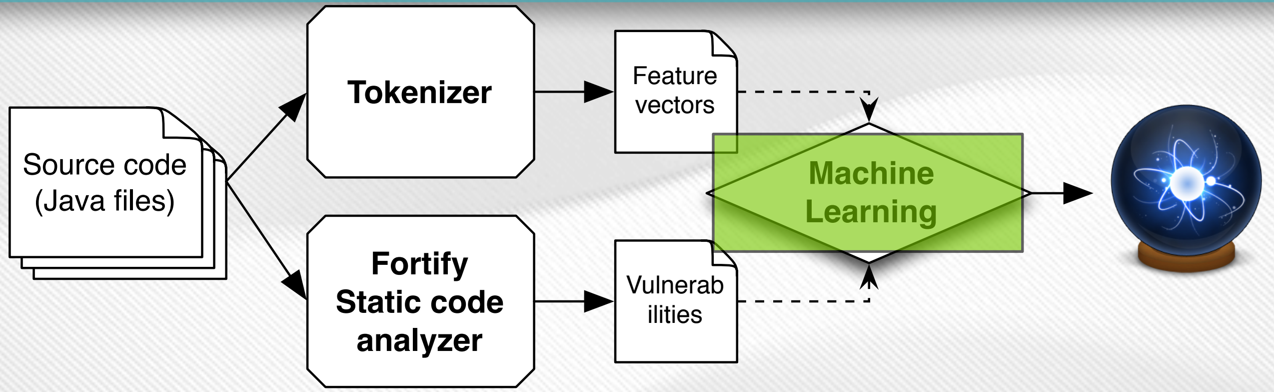
Monday 10 June 13 week

Machine learning



Monday 10 June 13 week

Machine learning



Leverage machine learning techniques to build a prediction model

- Training set -> the data used to train the model
- Testing set -> the data used to validate the model

Various techniques available (SVM, Naive Bayes, Random Forest, CART, kNN)

Experiment 1

Experiment 1

DistriNet

Can we predict future versions of an app based on its first version?

Experiment 1

DistriNet

Can we predict future versions of an app based on its first version?

- **Training set** - the first version (v0) of an app

Experiment 1

DistriNet

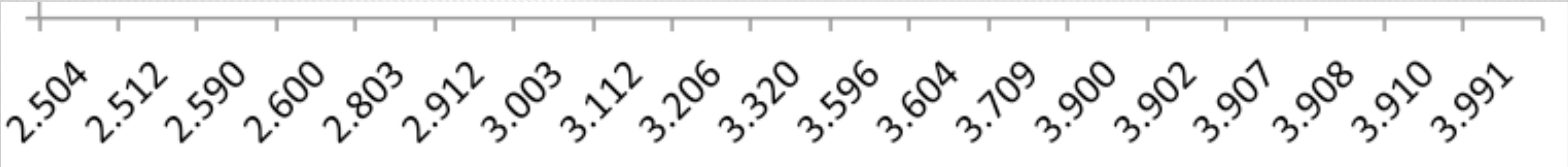
Can we predict future versions of an app based on its first version?

- **Training set** - the first version (v0) of an app
- **Testing set** - all subsequent versions of that app

Experiment 1

Can we predict future versions of an app based on its first version?

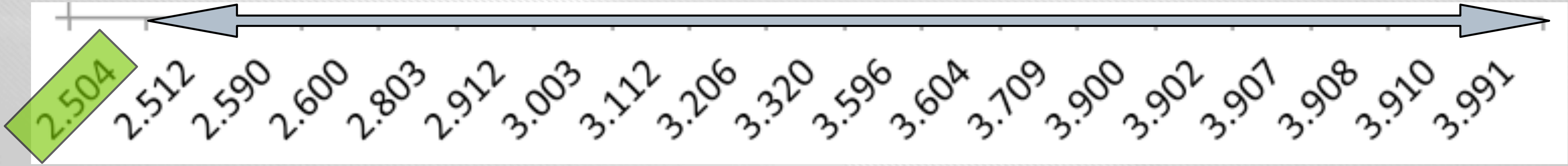
- **Training set** - the first version (v0) of an app
- **Testing set** - all subsequent versions of that app



Experiment 1

Can we predict future versions of an app based on its first version?

- **Training set** - the first version (v0) of an app
- **Testing set** - all subsequent versions of that app

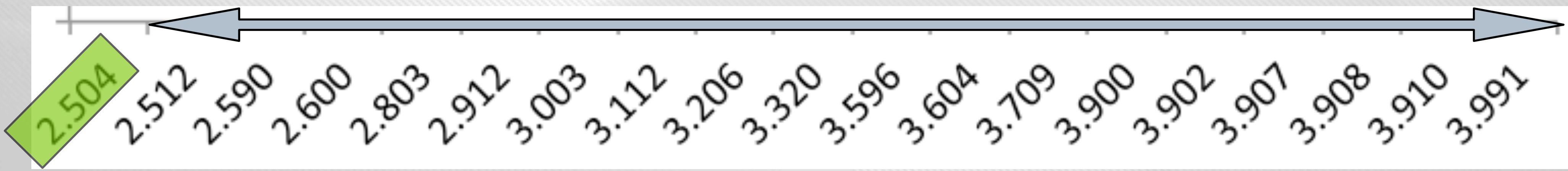


Monday 10 June 13 week

Experiment 1

Can we predict future versions of an app based on its first version?

- **Training set** - the first version (v0) of an app
- **Testing set** - all subsequent versions of that app



- Repeat for all apps

Experiment 2

Experiment 2

DistriNet

Can we build a generalized predictor that works on all apps?

- **Training set** - the first version (v0) of an app
- **Testing set** - first versions of all other apps

Applications (data from early 2012)

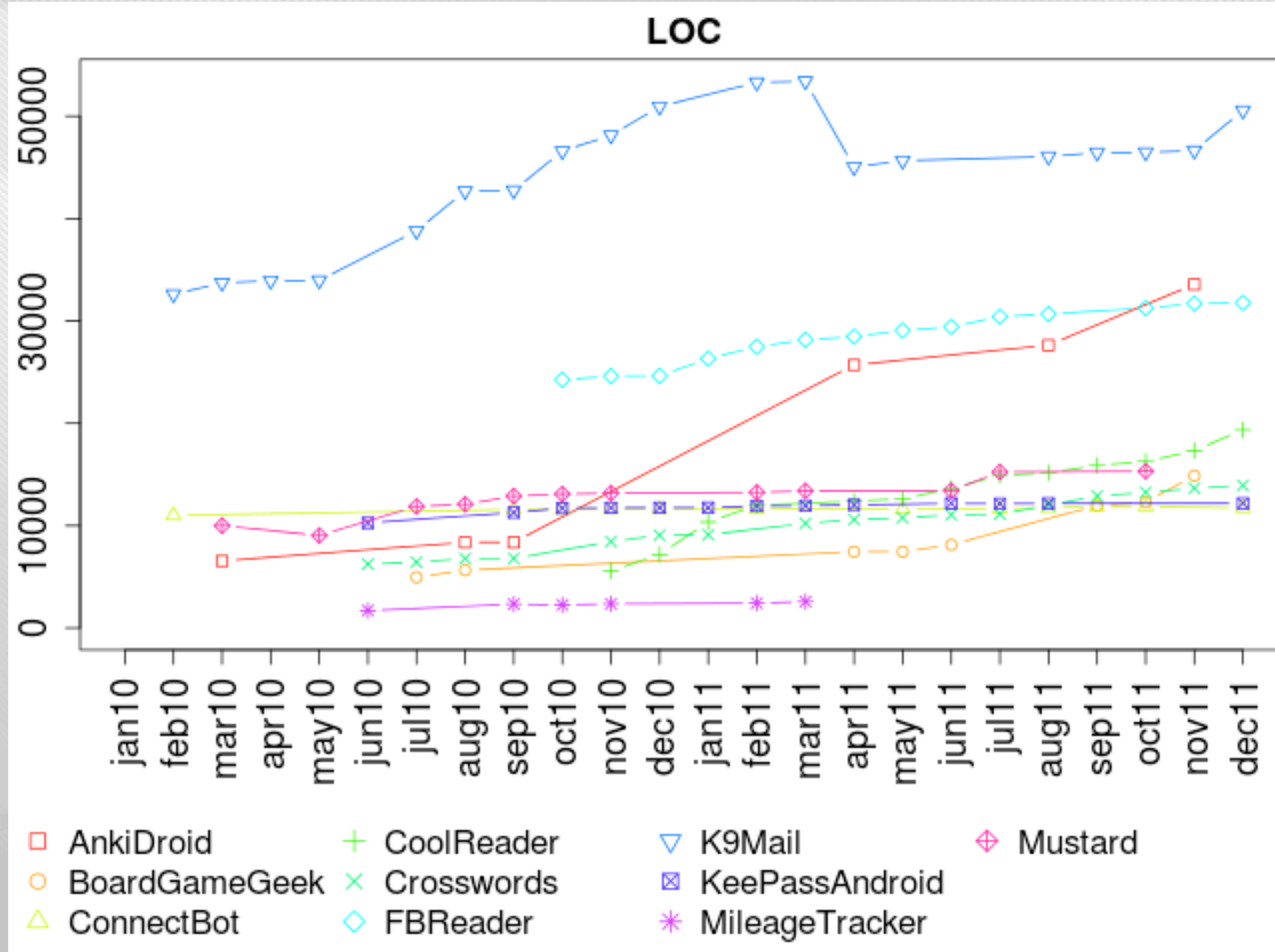
DistriNet

Application	Category	Downloads	Versions
AnkiDroid	education	100k - 500k	8
BoardGameGeek	books	10k - 50k	8
Connectbot	communication	1M - 5M	12
CoolReader	books	1M - 5M	13
Crosswords	brain & puzzle	5k - 10k	17
FBReader	books	1M - 5M	14
K9 Mail	communication	1M - 5M	19
KeePassAndroid	tools	100k - 500k	13
MileageTracker	finance	100k - 500k	6
Mustard	social	10k - 50k	12

- F-droid repository: 01/01/2010->31/12/2011
- Selection criteria: open-source, size, number of versions

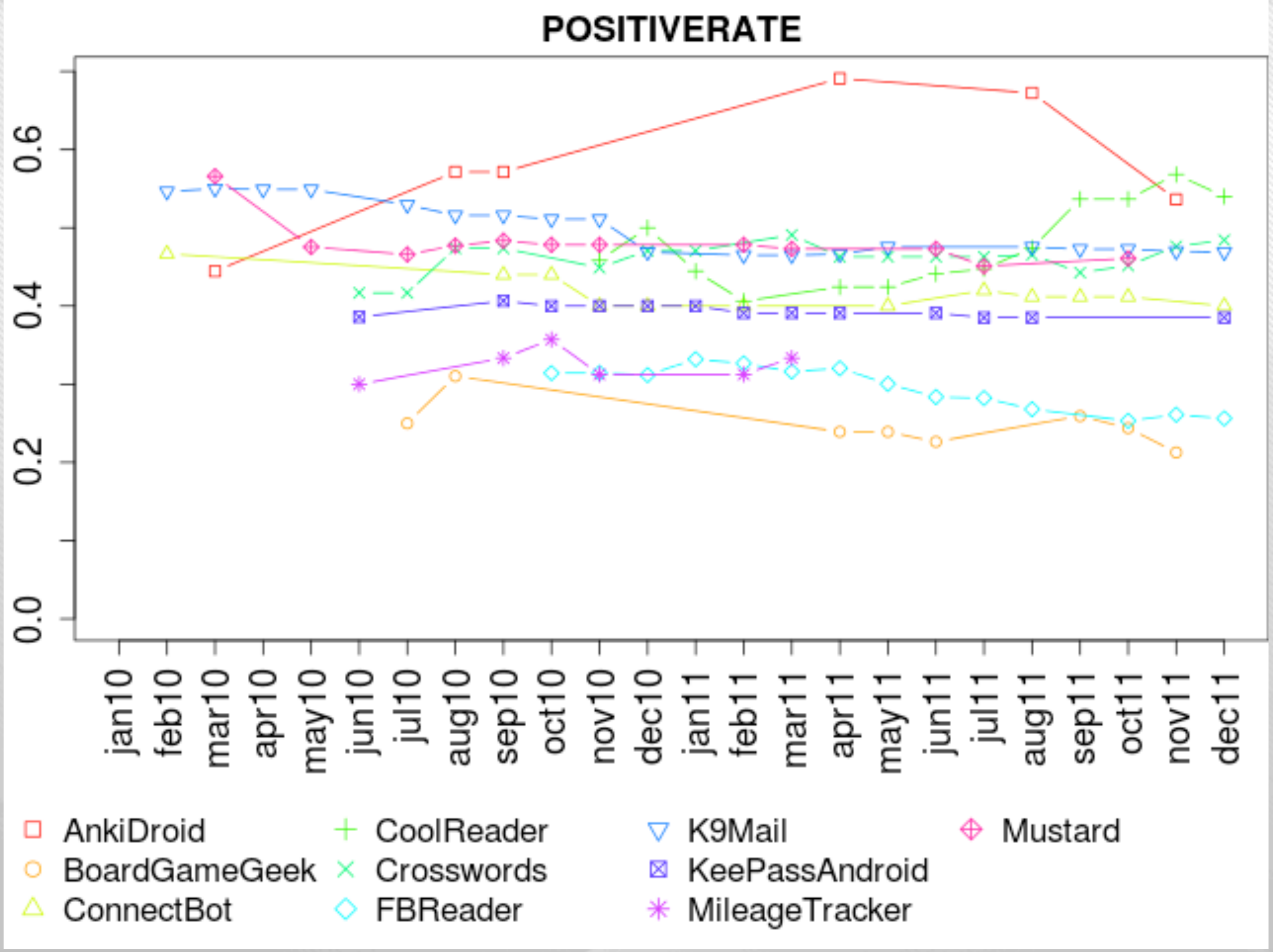
Applications: descriptive statistics

DistriNet



Monday 10 June 13 week

Applications: descriptive statistics



Outline

DistriNet

Existing tools and techniques

- Static code analysis
- Vulnerability prediction using metrics

Our approach

Results

Conclusions and future research

Performance indicators

Performance indicators

Accuracy: percentage of correctly classified files

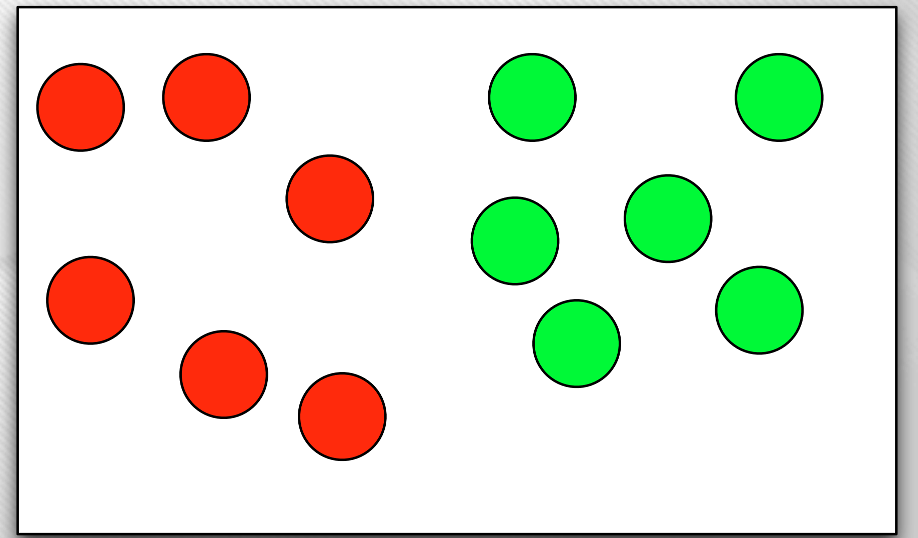
- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Performance indicators

DistriNet

Accuracy: percentage of correctly classified files

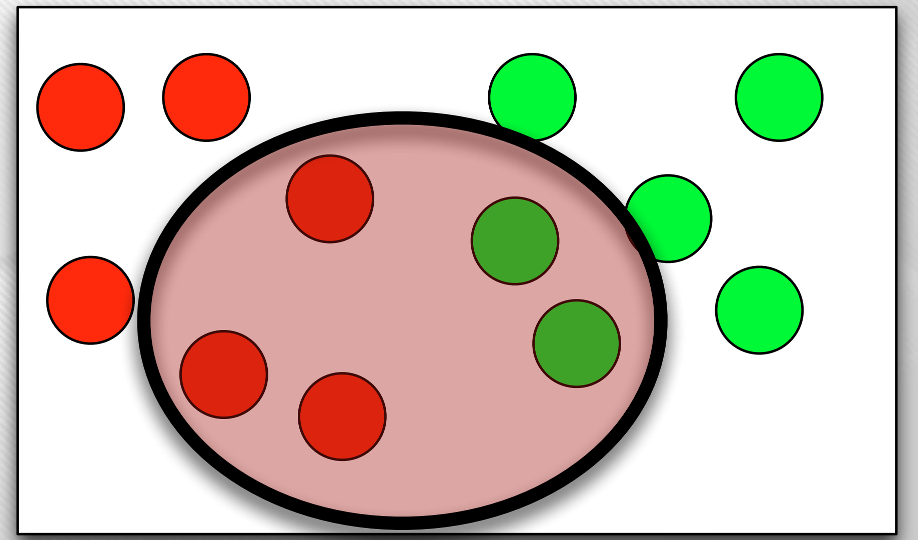
- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy



Performance indicators

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy



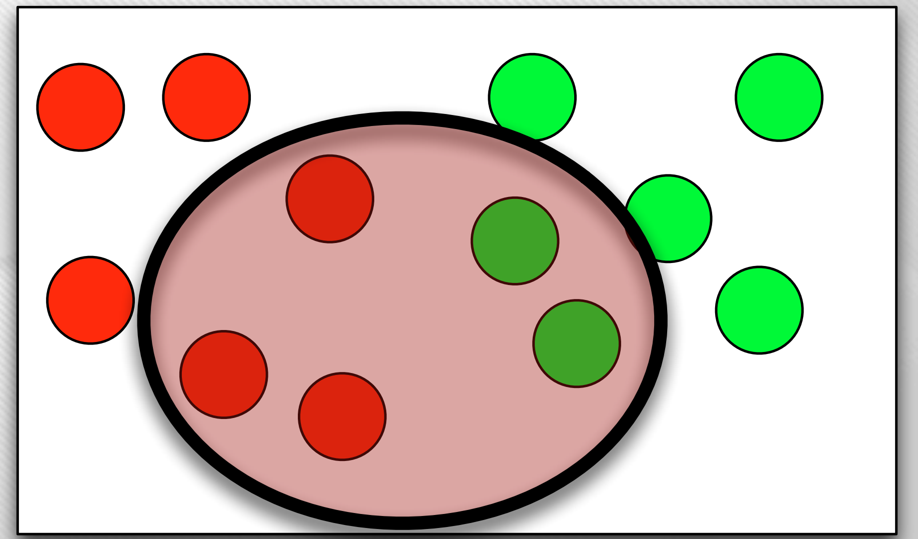
Performance indicators

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)



Monday 10 June 13 week

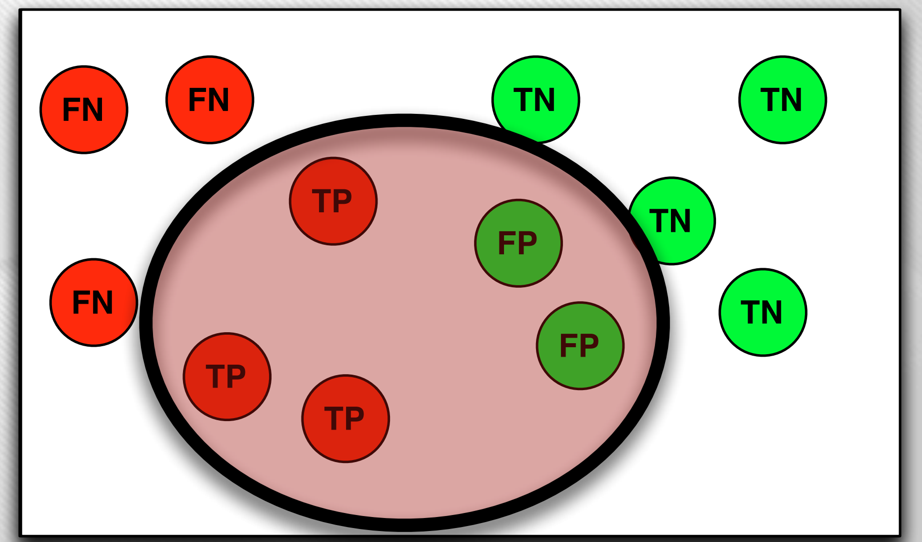
Performance indicators

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)



Performance indicators

DistriNet

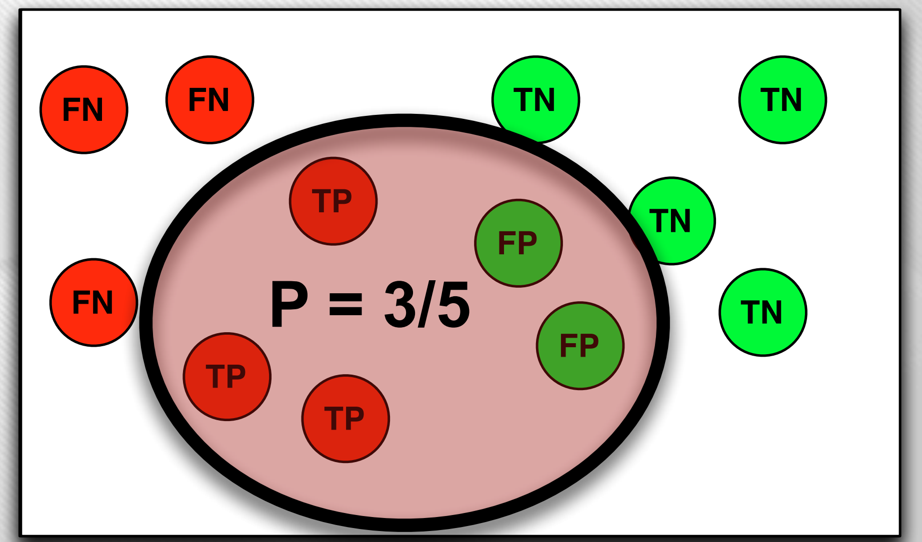
Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)

Precision: P = TP/(TP+FP)



Performance indicators

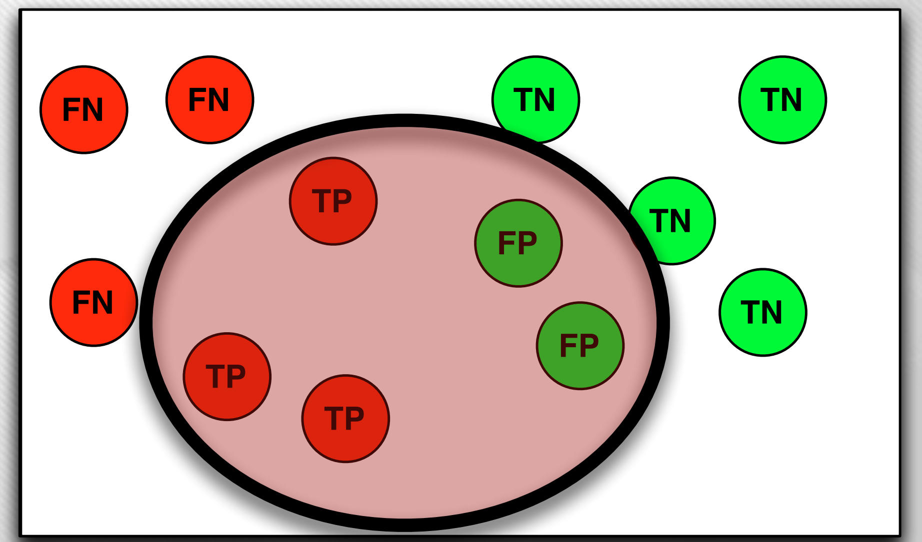
DistriNet

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)



Precision: P = TP/(TP+FP)

Monday 10 June 13 week

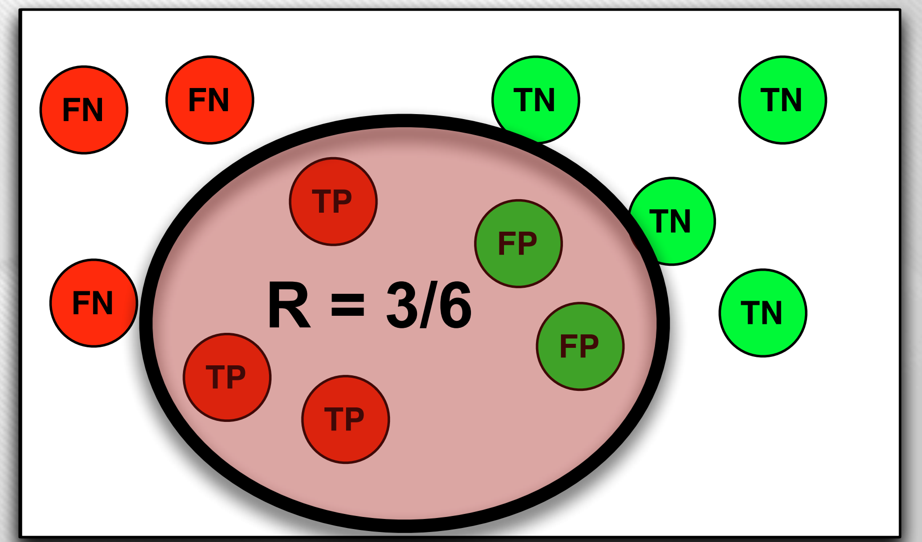
Performance indicators

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)



Precision: P = TP/(TP+FP)

Recall: R = TP/(TP+FN)

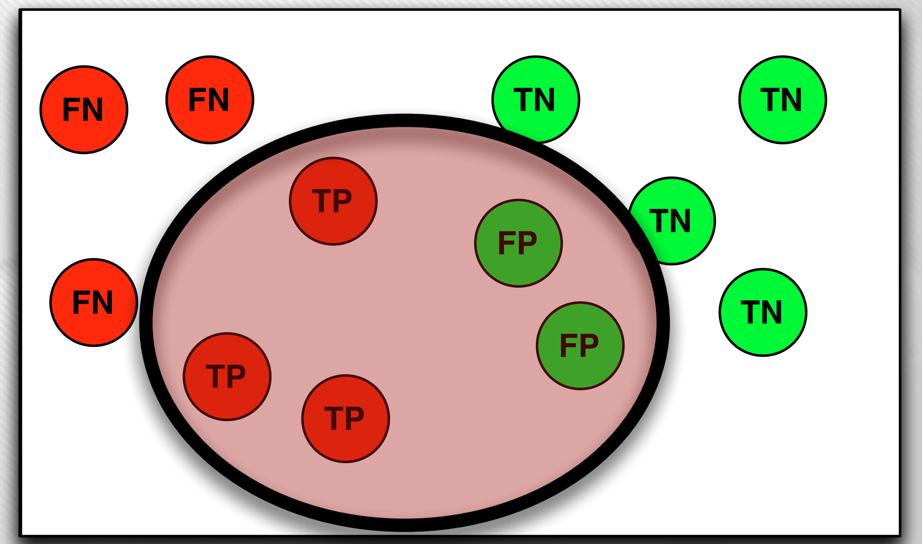
Performance indicators

Accuracy: percentage of correctly classified files

- imagine 90% of the files are clean
- saying all files are clean will achieve 90% accuracy

Prediction vs. reality

- True positive (TP)
- True negative (TN)
- False positive (FP)
- False negative (FN)



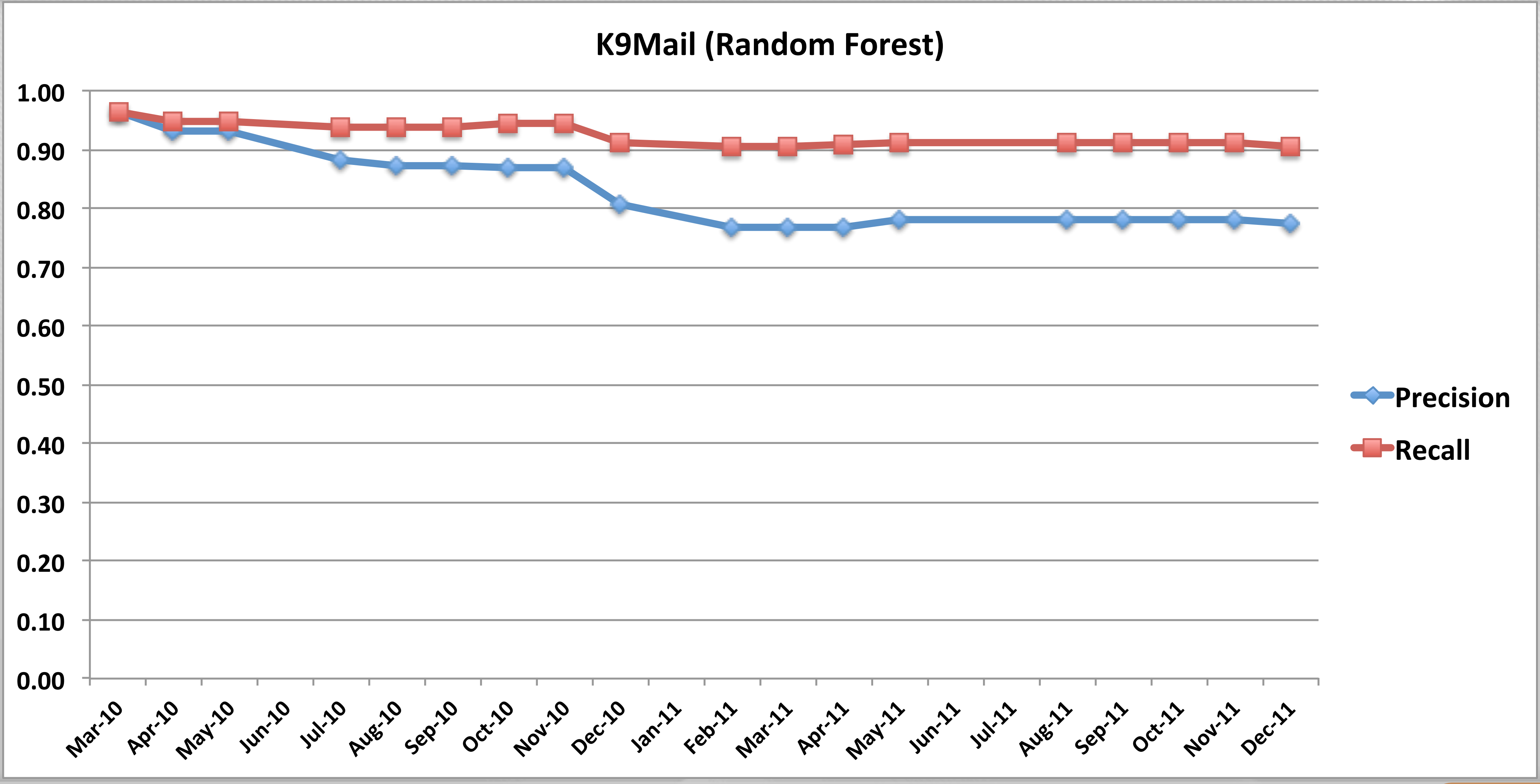
Precision: P = TP/(TP+FP)

Recall: R = TP/(TP+FN)

Performance indicators: K9

DistriNet

Performance indicators: K9



Experiment 1: future predictions

Experiment 1: future predictions

When do we need to build a new model?

- Retrain when performance indicators drop with 10%

Experiment 1: future predictions

DistriNet

When do we need to build a new model?

- Retrain when performance indicators drop with 10%

Application	Retrain (months)
AnkiDroid	--
BoardGameGeek	9
ConnectBot	--
CoolReader	10
Crosswords	2
FBReader	--
K9Mail	12
KeePassAndroid	--
MileageTracker	1
Mustard	--

-- no retraining is required

Results: most influential features

Results: most influential features

DistriNet

Most influential features

Results: most influential features

DistriNet

Most influential features

- *e, Exception, try, catch (error handling)*

Results: most influential features

Most influential features

- *e, Exception, try, catch (error handling)*
- *if (branching)*

Results: most influential features

Most influential features

- *e, Exception, try, catch (error handling)*
- *if (branching)*
- *null (pointer algebra)*

Results: most influential features

DistriNet

Most influential features

- *e, Exception, try, catch (error handling)*
- *if (branching)*
- *null (pointer algebra)*
- java, org (import statements)

Results: most influential features

Most influential features

- *e, Exception, try, catch (error handling)*
- *if (branching)*
- *null (pointer algebra)*
- java, org (import statements)
- new, Log (others)

Results: most influential features

Most influential features

- *e, Exception, try, catch (error handling)*
- *if (branching)*
- *null (pointer algebra)*
- java, org (import statements)
- new, Log (others)

Produced by InfoGain

Validity threats

DistriNet

Use of Fortify tool for vulnerability extraction

- Some research results have shown that there are strong correlations between static analysis metrics and the quality of reported vulnerabilities
- **Manual validation seems to confirm our findings (work in progress)!**
- **We are currently validating the same technique on Mozilla Firefox and the results are slightly better than the existing work**

Conclusions and future research

We have presented a novel technique for predicting vulnerable Java files in Android applications

- The obtained results are very promising

We are working in parallel on 2 additional tracks

- Vulnerability prediction for Firefox/Chrome in C++
- Vulnerability prediction for PHP

Bring your own data

We are looking to validate our technique further

If you have data you are willing to share with us, we would be glad to collaborate