In collaboration with:

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITÄT KOBLENZ · LANDAU

Feature-based defect prediction
using machine learning methods

Interim report

Winter semester 2019 / 2020

February 12th, 2020

Stefan Strüder

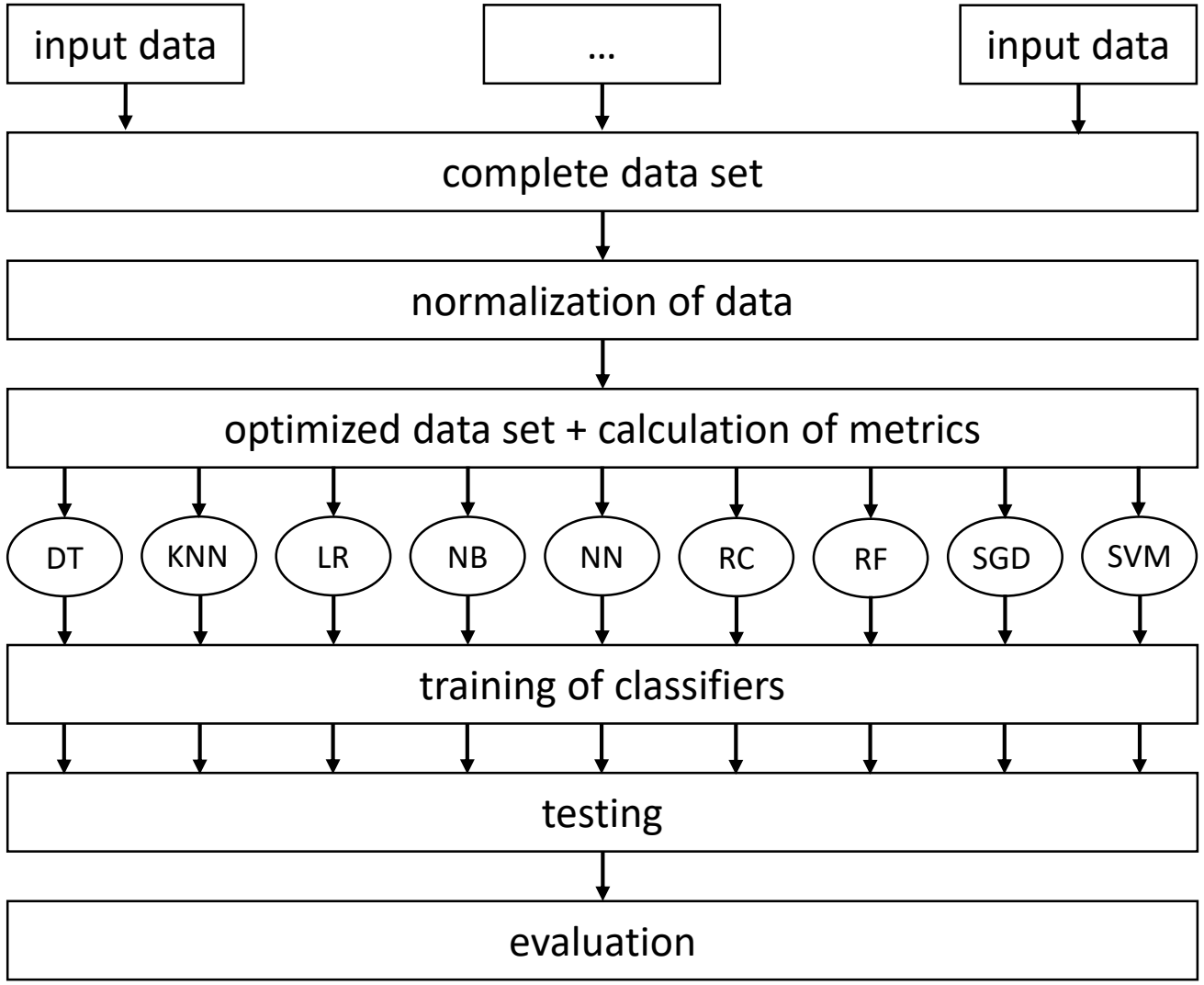
Summary

- aim of the thesis
  - prediction technique for software defects
  - in consideration of software features
  - based on methods of machine learning
- data basis: commits of versioning systems (Git)
  - faulty and defect-free commits for training of classifiers
- three research objectives
  - 1. creation of data set | 2. training of classifiers | 3. evaluation of classifiers

1 Time scheduling

Time scheduling

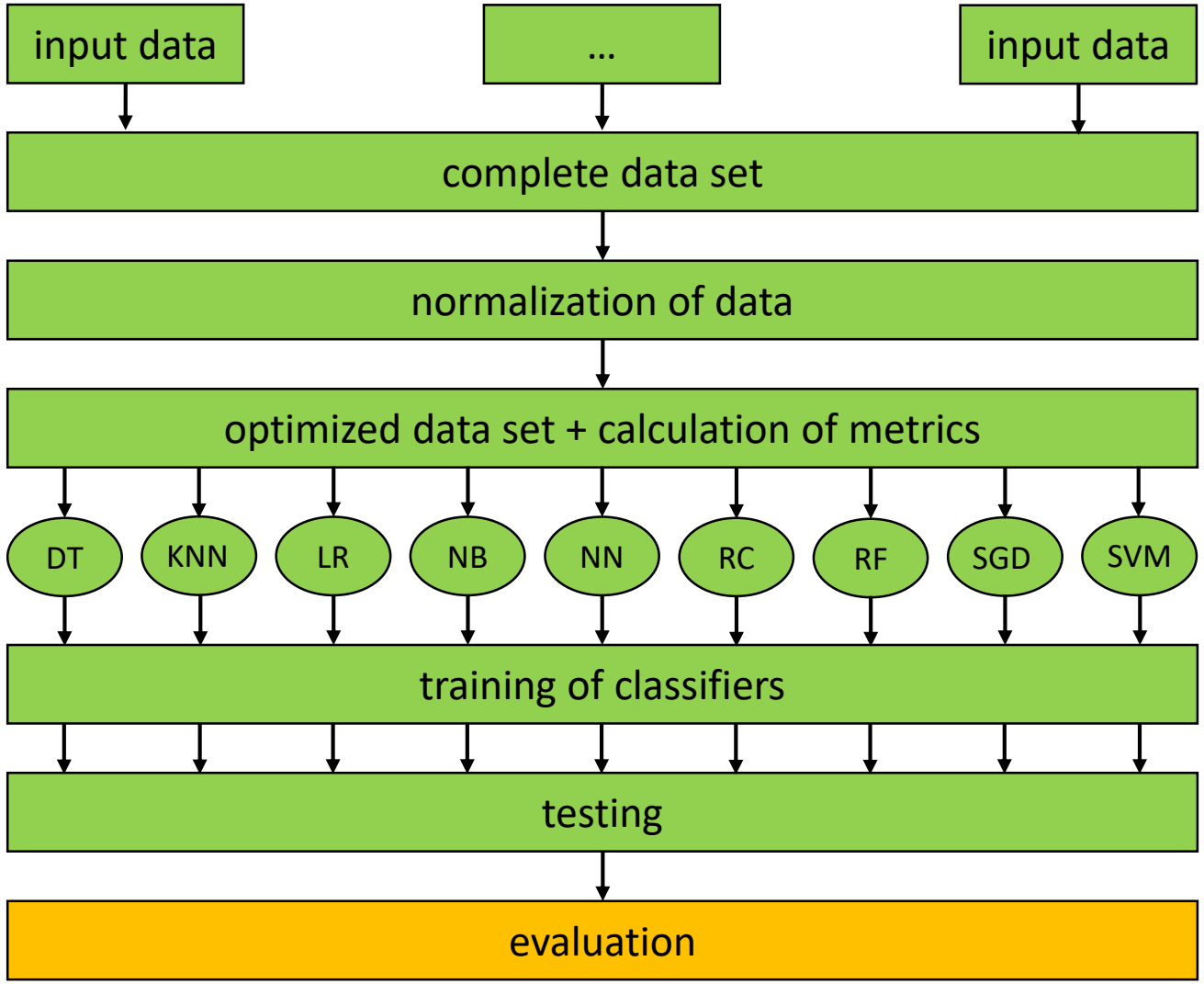
applied machine learning method



(Ceylan, Kutlubay, & Bener, 2006)

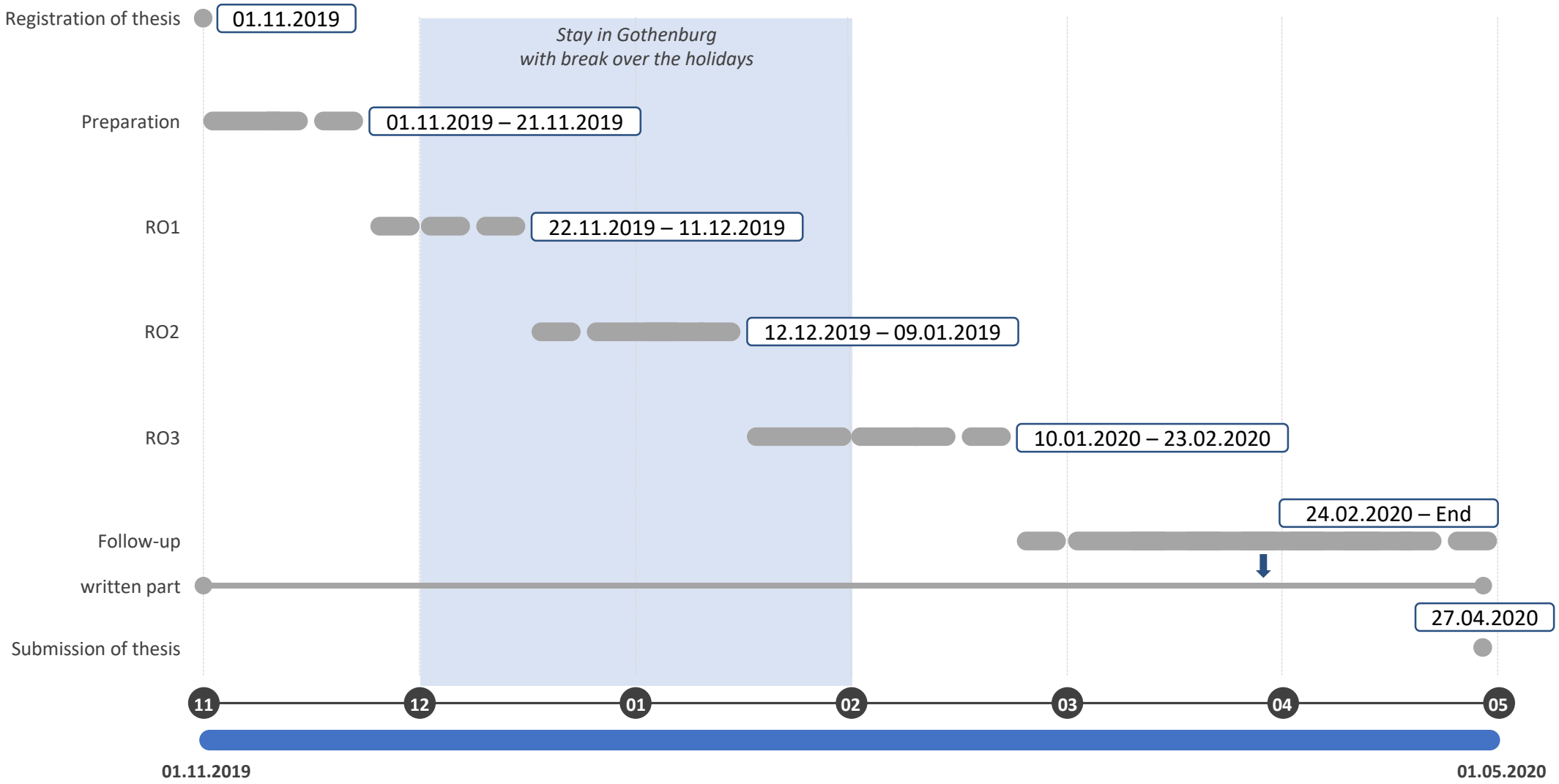
Time scheduling

applied machine learning method

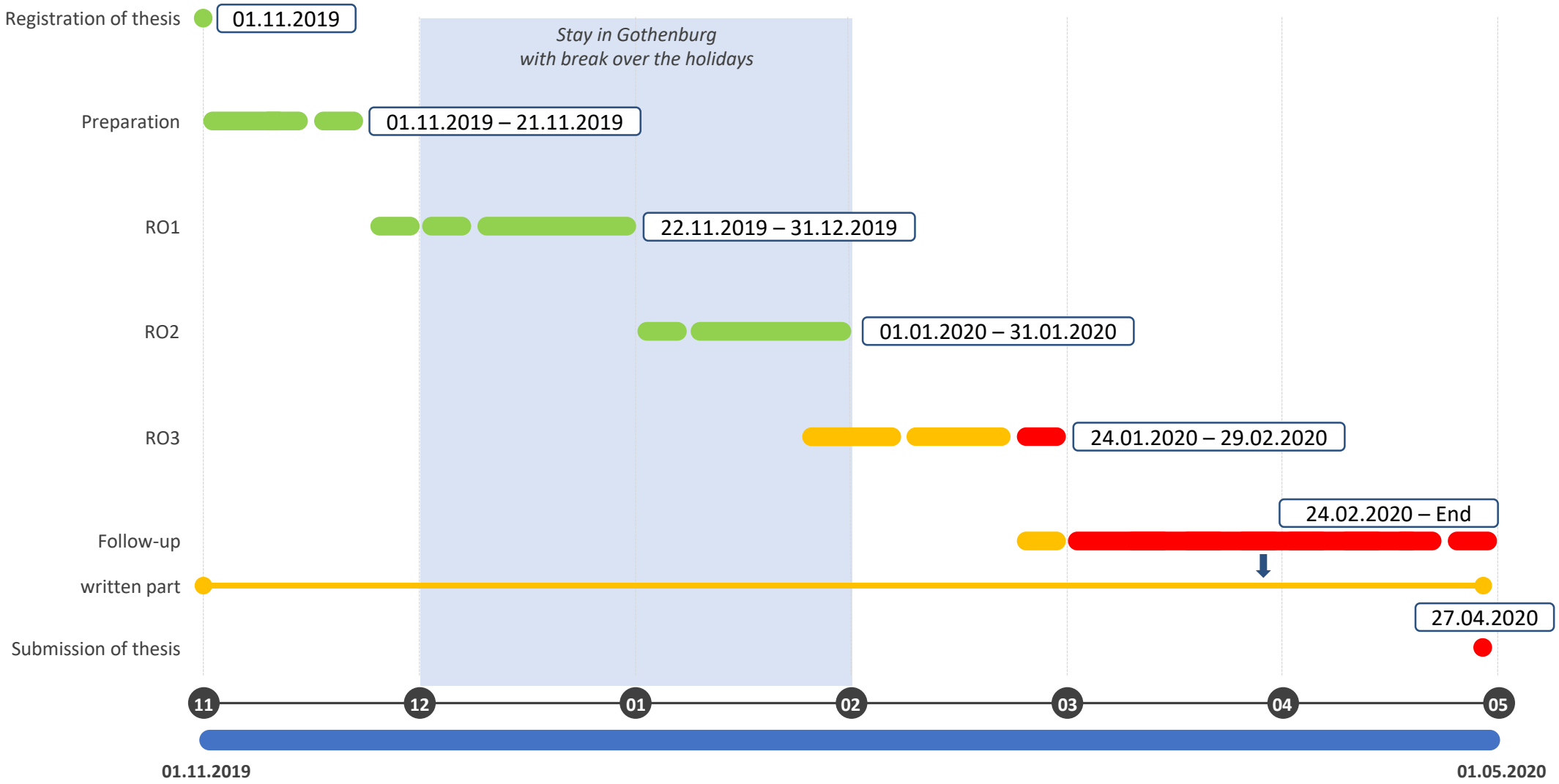


(Ceylan, Kutlubay, & Bener, 2006)

Time scheduling



Time scheduling



2 Creation of the data set

Creation of the data set

- usage of PyDriller for repository mining
  - Python framework by (Spadini, Aniche, & Bacchelli, 2018)
  - available on GitHub under Apache License 2.0 licence
  - extraction of commits, developers, diffs and source code (and more)
  - well documented (https://pydriller.readthedocs.io)

```
1  for commit in RepositoryMining("link_to_repo").traverse_commits():
2          for m in commit.modifications:
3                  print(
4                        "Author {}".format(commit.author.name),
5                        " modified {}".format(m.filename),
6                        " with a change type of {}".format(m.change_type.name),
7                        " and the complexity is {}".format(m.complexity)
8                  )
```

Creation of the data set

- historical data of 13 software projects
  - feature-based software (based on preprocessor conditionals)
  - selection criterion: previous use in literature
    (Hunsen et al., 2016; Liebig et al., 2010; Queiroz et al., 2017)
  - extracted from Git, GitHub, GitLab and Sourceforge repositories
  - divided into commits per release
    - based on tag structure of Git repositories

Blender	Busybox	Emacs	GIMP	Gnumeric	gnuplot	Irssi
3D modelling tool	*UNIX tool package*	*text editor*	*photo editor*	*spreadsheet*	*plotter*	*IRC client*

libxml2	lighttpd	MPSolve	Parrot	Vim	xfig
XML parser	*webserver*	*polynomial solver*	*VM*	*text editor*	*graphics editor*

Creation of the data set

UNIVERSITÄT KOBLENZ · LANDAU

- data was stored in MySQL database
  - one table for each software project

Column	Description	Column	Description
change_type	type of change (added, deleted, modified, renamed)	filename	name of changed file
commit_author	responsible developer	lines_added	number of lines added to file
commit_hash	unique identifier of commit	lines_removed	number of lines removed from file
commit_msg	commit message	name	software name
cycomplexity	cyclomatic complexity of changed file	nloc	lines of code of file
diff	diff of changed file	release_number	associated release number based on tags
feature	*features, that were modified*	*status*	*normal (false) or corrective (true) commit*

Creation of the data set

- further processing of data
  - extract modified features from diffs
    - usage of regular expressions to detect `#IFDEF` and `#IFNDEF`
  - identify bug-fixing commits
    - does commit-message contain "bug", "error", "fail" or "fix"?
  - identify bug-introducing commits
    - application of the SZZ-algorithm (Śliwerski, Zimmermann, & Zeller, 2005)
  - remove "false" features manually
    - e.g. from `#IFDEFs` found in comments

Creation of the data set

- further processing of data
  - exclusion of "header-features"
    - *in certain C-patterns:* header files included with `#IFDEFs` like features
    - identifiable by featurename**_h_**
    - previously considered falsely as features
    - metrics calculation and classifier training had to be redone
- all done via SQL queries and / or Python scripts

Creation of the data set

UNIVERSITÄT KOBLENZ · LANDAU

	Purpose	Data source	#Releases	#Commits	#Corrective	#Bugintrod.	#Features
Blender	3D modelling tool	GitHub mirror	11	19119	8258	1418	3394
Busybox	UNIX tool package	Git repository	14	4984	1408	142	628
Emacs	text editor	GitHub mirror	7	12805	6959	685	718
GIMP	photo editor	GitLab repository	14	7240	1703	272	204
Gnumeric	spreadsheet	GitLab repository	8	6025	1591	136	637
gnuplot	plotter	GitHub mirror	5	6619	880	1323	558
Irssi	IRC client	GitHub repository	7	253	77	1	9
libxml2	XML parser	GitLab repository	10	732	409	37	200
lighttpd	webserver	Git repository	6	2597	1202	555	230
MPSolve	polynomial solver	GitHub repository	8	668	158	69	54
Parrot	VM	GitHub repository	7	16245	3437	824	397
Vim	text editor	GitHub repository	7	9849	1033	2571	1158
xfig	graphics editor	Sourceforge	7	18	0	0	137

Creation of the data set

- calculation of metrics on **feature**- and **file**-level
  - 5 metrics taken from (Queiroz et al., 2017, marked with [q])
  - 6 additional metrics
  - calculated via SQL queries and / or via Python scripts

Metric	Description
comm [q]	number of commits dedicated to the affected feature / file in a release
adev [q]	number of developers who have worked on the affected feature / file in a release
ddev [q]	cumulative number of developers who have worked on the affected feature / file in a release
exp [q]	geometric mean of the "experience*" of all developers who have worked on the affected feature / file in a release
oexp [q]	"experience*" of the developer who has contributed most to the affected feature / file in a release

** number of added, modified and removed files*

Creation of the data set

Metric	Description
modd	"modification degree": number of edits to the affected feature / file within a release
mods	"modification scope": number of unique features / files edited in one release
nloc	average lines of code of the edits of the affected feature / file in a release
cyco	average cyclomatic complexity of modifications on the affected feature / file in a release
addl	average number of lines added to the affected feature / file in a release
reml	average number of lines deleted from the affected feature / file in a release

3 Training of the classifiers

Training of the classifiers

- usage of scikit-learn and WEKA
- scikit-learn (https://scikit-learn.org/)
  - best known Python library for machine learning
  - selection of various classification algorithms
  - comprehensive integration of further Python libraries
- WEKA (https://www.cs.waikato.ac.nz/ml/weka/)
  - developed at the University of Waikato (New Zealand)
  - comprehensive selection of classification- and attribute-selection-algorithms
  - graphical interface
- two tools to compare the results of the respective implementations of the algorithms

Training of the classifiers

scikit-learn	WEKA
Decision Trees	J48
k-Nearest-Neighbor	k-Nearest-Neighbor
Ridge Classifier	Logistic Regression
Naïve Bayes	Naïve Bayes
Neural Networks	Neural Networks
Random Forest	Random Forest
Stochastic Gradient Descent	Stochastic Gradient Descent
Support Vector Machines	Support Vector Machines

J48 is a DT-algorithm

both based on regression

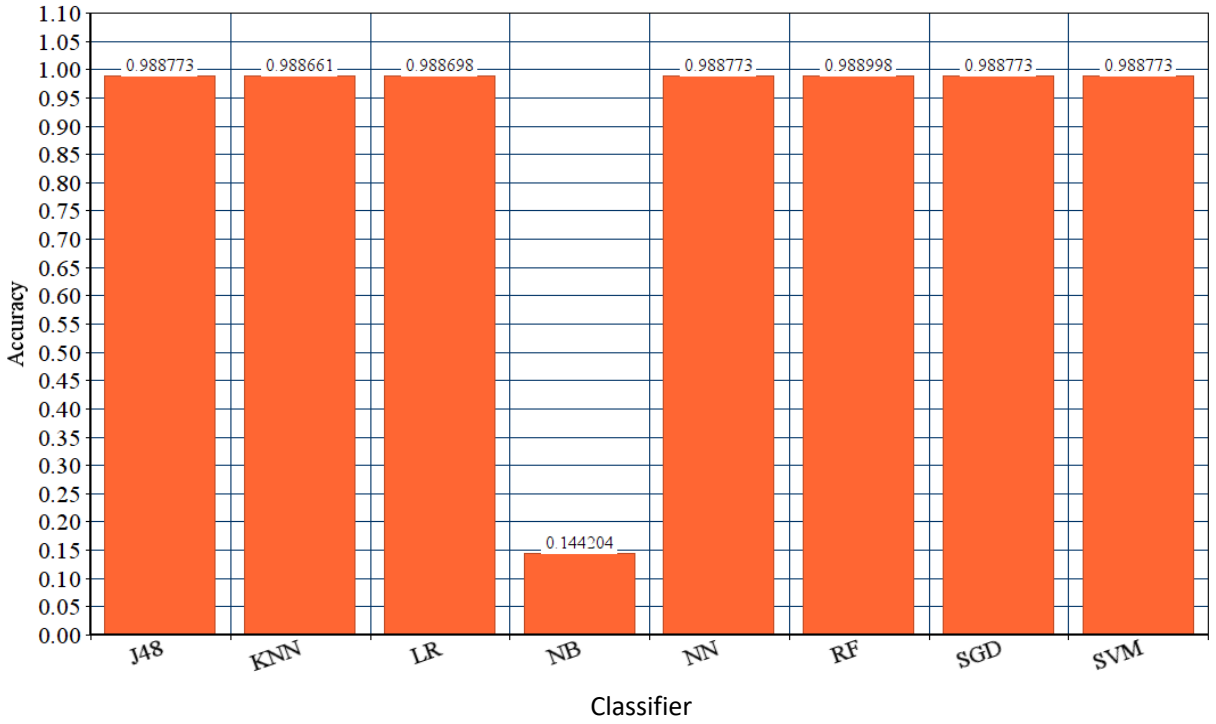
each classifier has been configured before final training

Training of the classifiers

- file dataset is imbalanced
  - label "clean" is strongly overrepresented
  - balance is a prerequisite for correct training of classifiers
  - **unbalanced data results in misleading accuracy**
    - most data records are correctly assigned to the overrepresented class
- application of the SMOTE method
  - Synthetic Minority Over-sampling Technique (Chawla, Bowyer, Hall, & Kegelmeyer, 2002)
  - oversampling of the underrepresented class
  - integrated "filter" in WEKA

Comparison of WEKA-classifiers before and after applying the SMOTE method

before



after

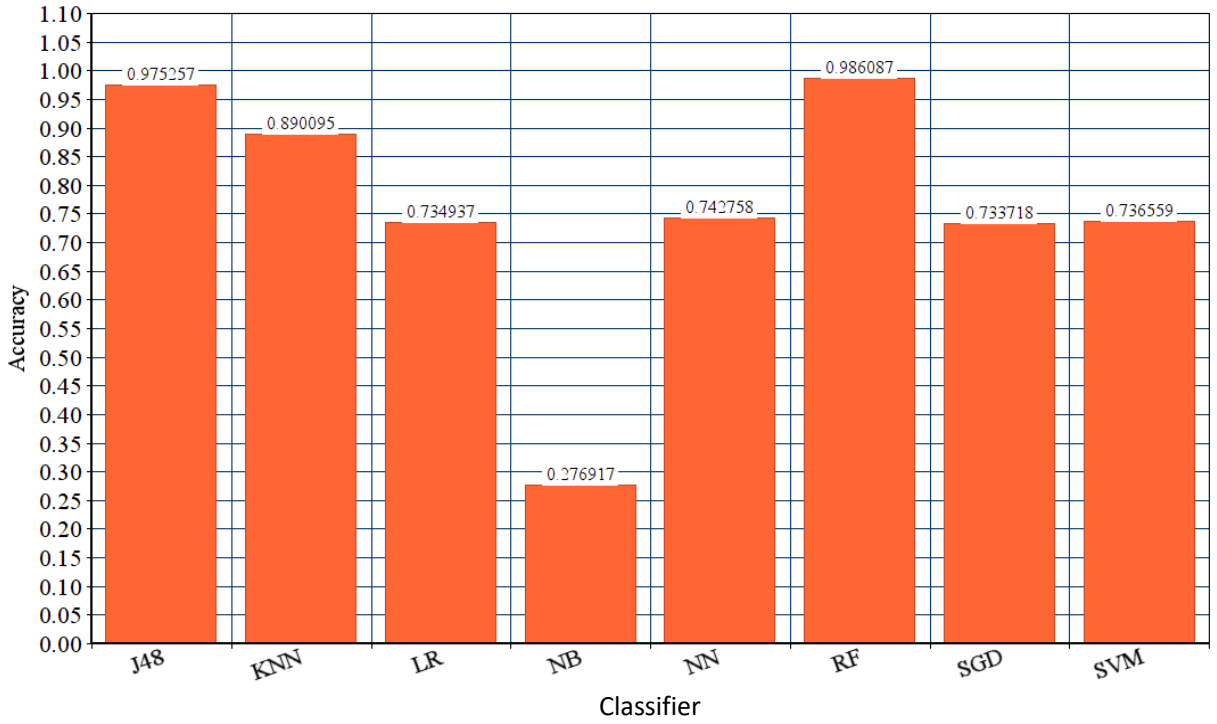
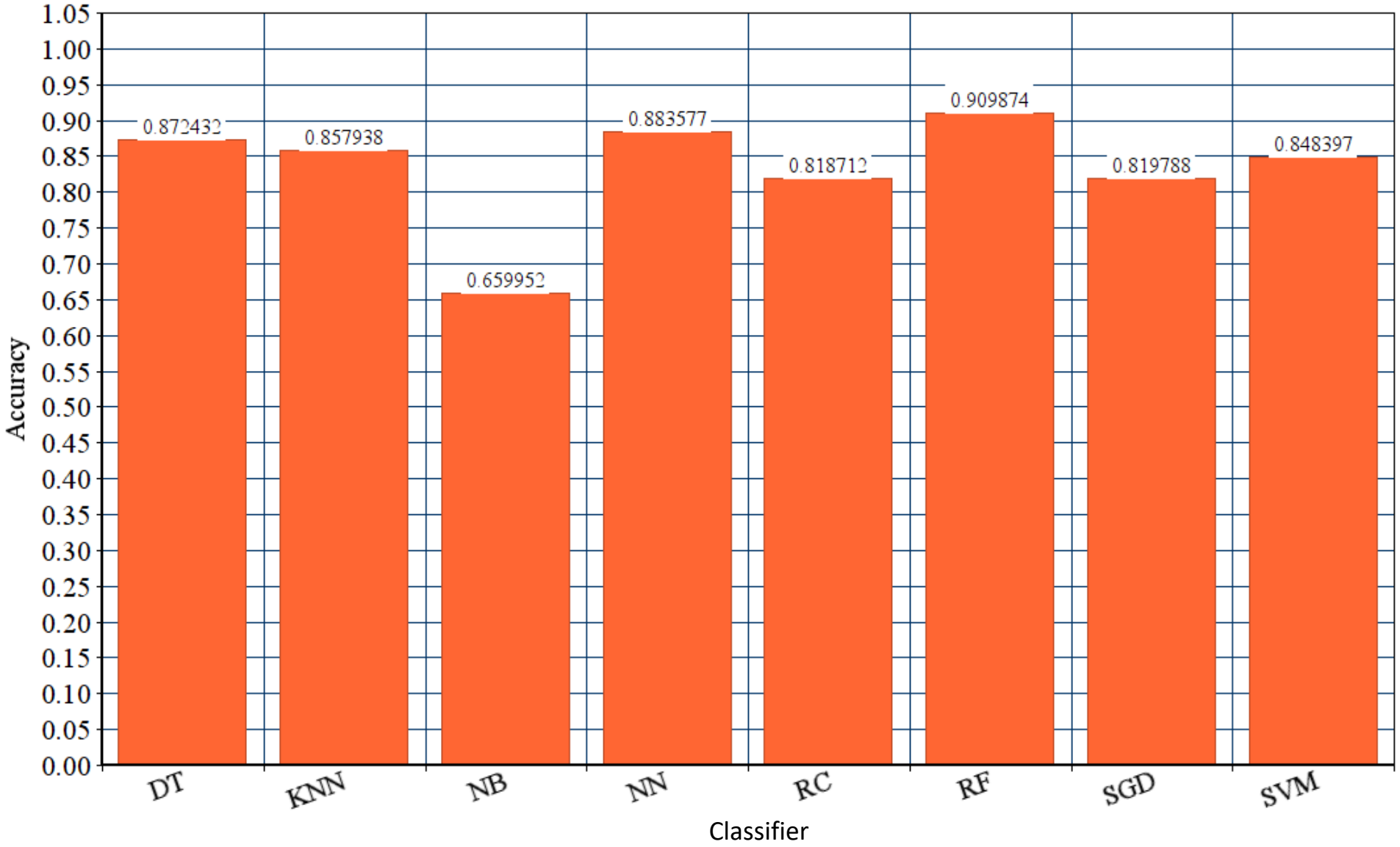


Abb.	Classifier	Abb.	Classifier
J48	Decision Trees	NN	Neural Networks
KNN	k-Nearest-Neighbor	RF	Random Forest
LR	Logistic Regression	SGD	Stochastic Gradient Descent
NB	Naïve Bayes	SVM	Support Vector Machines

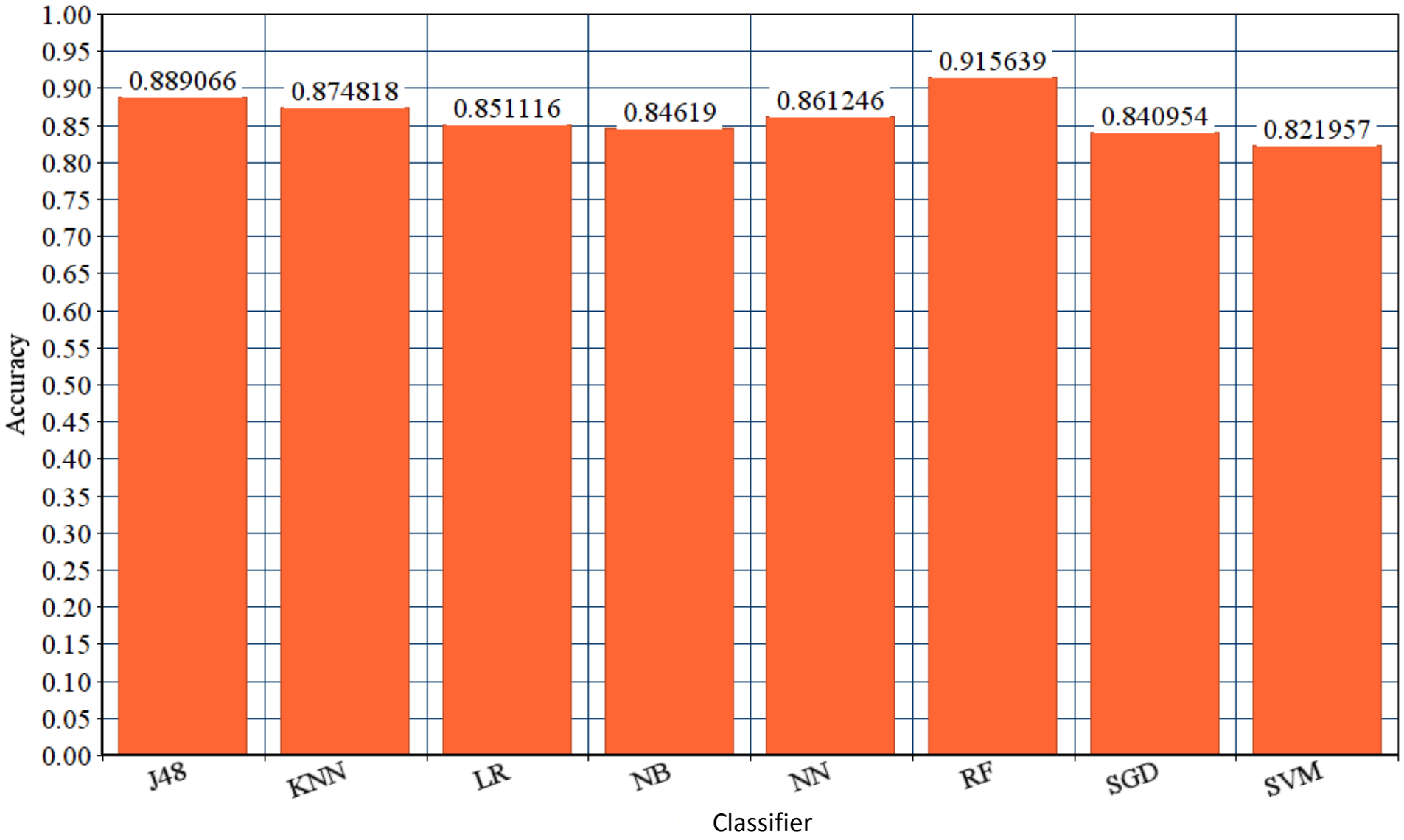
Final analysis of the feat-classifiers of scikit-learn

Abb.	Classifier
DT	Decision Trees
KNN	k-Nearest-Neighbor
RC	Ridge Classifier
NB	Naïve Bayes
NN	Neural Networks
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines



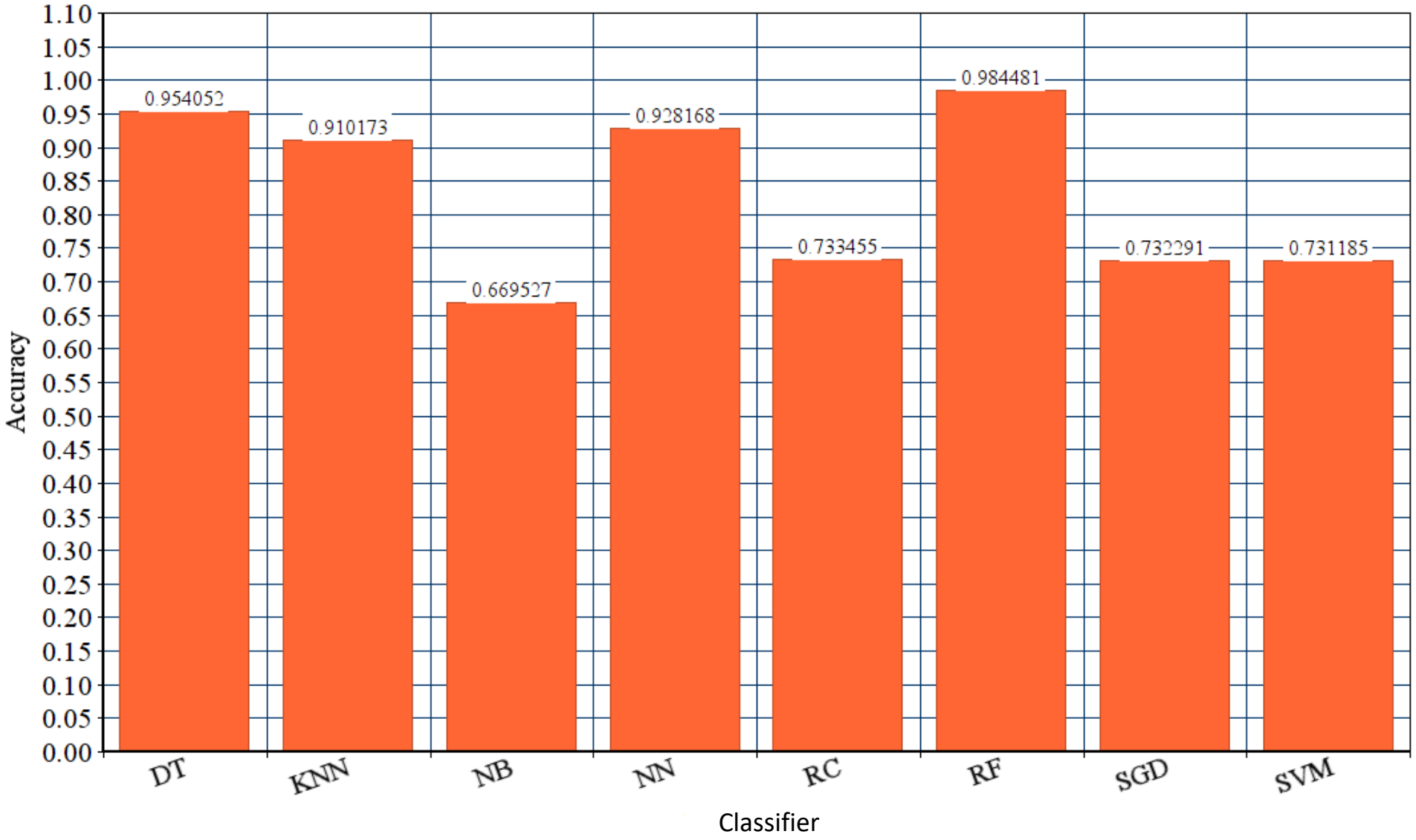
Final analysis of the feat-classifiers of WEKA

Abb.	Classifier
J48	Decision Trees
KNN	k-Nearest-Neighbor
LR	Logistic Regression
NB	Naïve Bayes
NN	Neural Networks
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines



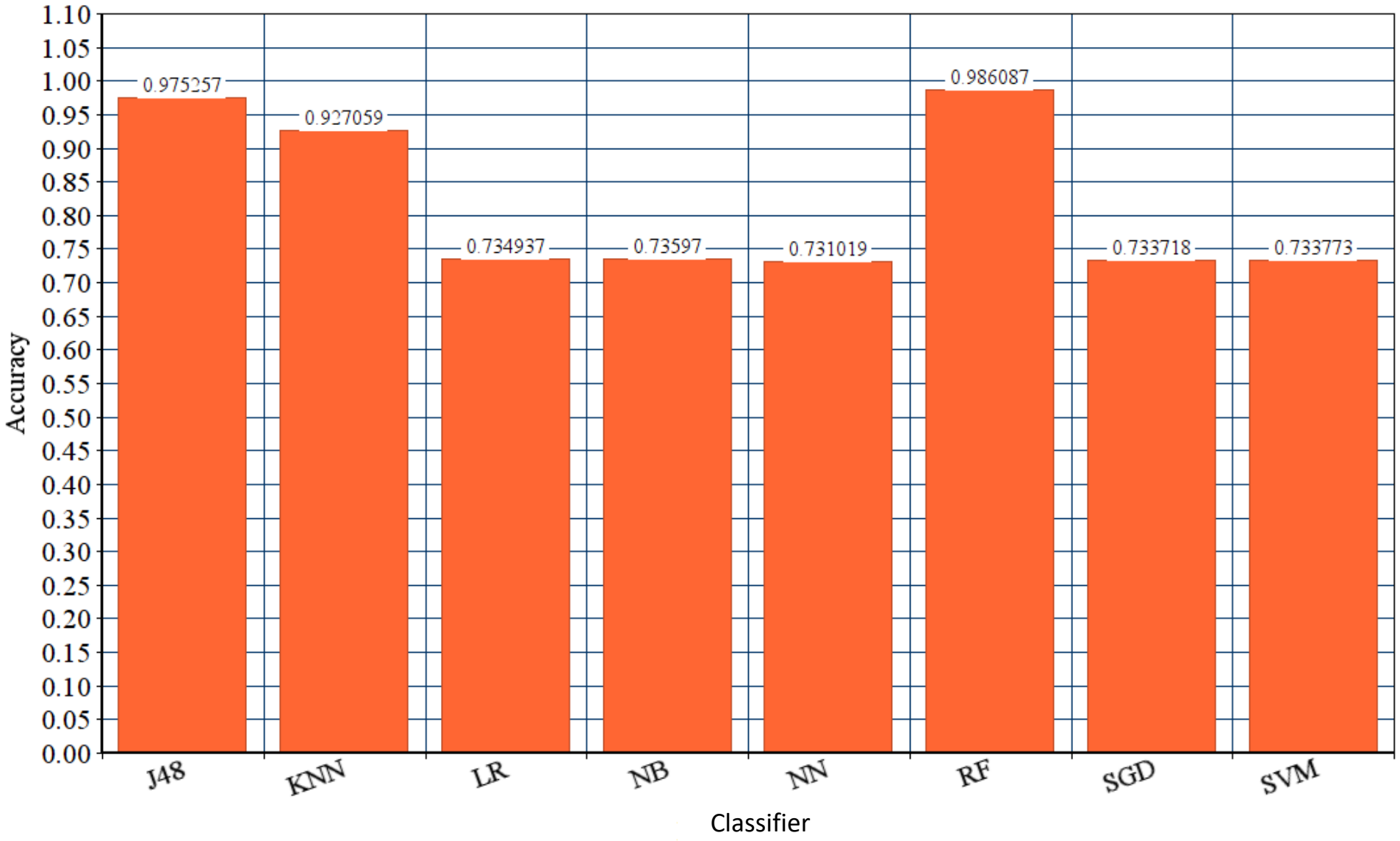
Final analysis of the file-classifiers of scikit-learn

Abb.	Classifier
DT	Decision Trees
KNN	k-Nearest-Neighbor
RC	Ridge Classifier
NB	Naïve Bayes
NN	Neural Networks
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines



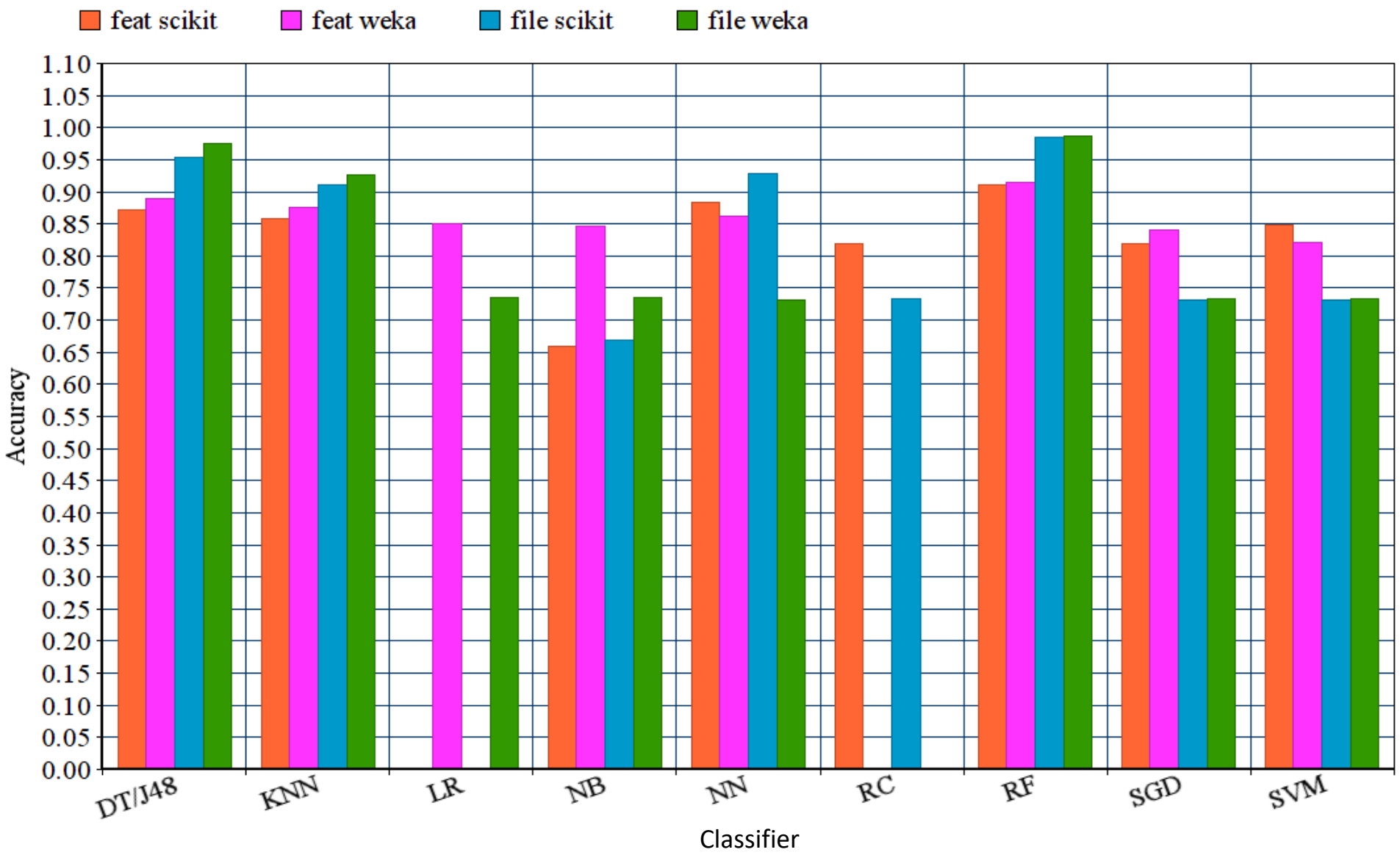
Final analysis of the file-classifiers of WEKA

Abb.	Classifier
J48	Decision Trees
KNN	k-Nearest-Neighbor
LR	Logistic Regression
NB	Naïve Bayes
NN	Neural Networks
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines



Comparison of the classifiers

Abb.	Classifier
DT	Decision Tree
J48	Decision Tree
KNN	k-Nearest-Neighbor
LR	Logistic Regression
NB	Naïve Bayes
NN	Neural Networks
RC	Ridge Classifier
RF	Random Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines



4 Outlook

Outlook

- evaluation of classifiers
  - research of suitable evaluation metrics
  - calculate evaluation metrics
- evaluation and comparision based on classical metrics found in literature
- continue written part of thesis

Thank you for your attention.

Question time.

Bibliography

- Ceylan, E., Kutlubay, F. O., & Bener, A. B. (2006). Software defect identification using machine learning techniques. Proceedings - 32nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA, 240–246. https://doi.org/10.1109/EUROMICRO.2006.56
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16, 321–357. https://doi.org/10.1613/jair.953
- Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O., Becker, M., & Apel, S. (2016). Preprocessor-based variability in open-source and industrial software systems: An empirical study. In Empirical Software Engineering (Vol. 21). https://doi.org/10.1007/s10664-015-9360-1
- Liebig, J., Apel, S., Lengauer, C., Kästner, C., & Schulze, M. (2010). An analysis of the variability in forty preprocessor-based software product lines. Proceedings - International Conference on Software Engineering, 1, 105–114. https://doi.org/10.1145/1806799.1806819
- Queiroz, R., Passos, L., Valente, M. T., Hunsen, C., Apel, S., & Czarnecki, K. (2017). The shape of feature code: an analysis of twenty C-preprocessor-based systems. Software and Systems Modeling, 16(1), 77–96. https://doi.org/10.1007/s10270-015-0483-z
- Śliwerski, J., Zimmermann, T., & Zeller, A. (2005). When do changes induce fixes? *Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005*, 1–5. https://doi.org/10.1145/1082983.1083147
- Spadini, D., Aniche, M., & Bacchelli, A. (2018). PyDriller: Python framework for mining software repositories. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018, 908–911. https://doi.org/10.1145/3236024.3264598