PyCP: An Open-Source Conformal Predictions Toolkit

Vineeth Balasubramanian, Aaron Baker, Matthew Yanez, Shayok
Chakraborty, Sethuraman Panchanathan

HAL Id: hal-01459631

https://hal.inria.fr/hal-01459631

Submitted on 7 Feb 2017

PyCP: An Open-Source Conformal Predictions Toolkit

Vineeth N Balasubramanian, Aaron Baker, Matthew Yanez, Shayok
Chakraborty, and Sethuraman Panchanathan

Center for Cognitive Ubiquitous Computing
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe AZ 85282 USA
{vineeth.nb,aaron.j.baker,mmyanez,schakr10,panch}@asu.edu

Abstract. The Conformal Predictions framework is a new game-theoretic
approach to reliable machine learning, which provides a methodology
to obtain error calibration under classification and regression settings.
The framework combines principles of transductive inference, algorithmic randomness and hypothesis testing to provide guaranteed error calibration in online settings (and calibration in offline settings supported
by empirical studies). As the framework is being increasingly used in
a variety of machine learning settings such as active learning, anomaly
detection, feature selection, and change detection, there is a need to develop algorithmic implementations of the framework that can be used
and further improved by researchers and practitioners. In this paper,
we introduce PyCP, an open-source implementation of the Conformal
Predictions framework that currently provides support for classification
problems within transductive and Mondrian settings. PyCP is modular,
extensible and intended for community sharing and development.

Keywords: Conformal predictions, Open-source software

1 Introduction

The Conformal Predictions (CP) framework is a game-theoretic approach to reliable machine learning, which provides a methodology to obtain error calibration
under classification and regression settings [1]. The framework combines principles of transductive inference, algorithmic randomness and hypothesis testing to
provide guaranteed error calibration in online settings [2] (and calibration in offline settings supported by empirical studies [3]). The framework can be applied
to various classification and regression algorithms, making it very generalizable.
In recent years, the framework has also been applied to other machine learning
settings including active learning [4], anomaly detection [5], feature selection [6],
change detection [7] and quality estimation [8]. The framework has found application in a variety of domains including medical diagnosis [9], bioinformatics
[10], biometrics [11], and network analysis [12].

The growing relevance of the CP framework has generated a need to develop algorithmic implementations of the framework that can be shared among

researchers and practitioners in the community. Machine learning researchers have recently begun to promote development of usable, inter-operable, flexible, and scalable machine learning software. As the discipline of machine learning has matured, the community has developed an extensive body of learning algorithms that are useful in diverse applications. Sharing source code improves usability and interoperability because it leads to quicker detection and correction of errors, better experimental reproducibility, incremental advancements of previous work, combinations of distinct theoretical advances, and more percolation of technical methods into other disciplines and industry [13].

In this paper, we present PyCP, the first open-source implementation of the CP framework, in the Python scripting language. Python has several advantages for use in an open-source implementation for scientific application. As a scripting language with white space syntax, it is relatively simple and has many widely-used libraries for scientific computing (e.g., NumPy, SciPy, Matplotlib). These libraries have contributed to a strong increase in the popularity of Python as a programming tool in machine learning research [14]. In order to avoid duplication of code of machine learning methods, PyCP has been developed on top of the popular SciKitLearn toolkit [1] [15], an open-source machine learning software. PyCP is currently available for use of the CP framework in classification settings with options for classifiers (k-Nearest Neighbor, Support Vector Machines, Decision Trees, Logistic Regression) as well as the framework setting (Transductive, Mondrian). It has been designed in a modular fashion to allow for easy extensibility and further development. The remainder of this paper is organized as follows: a brief background of the CP framework is presented in Section 2, the design of PyCP is presented in Section 3, examples of its use and availability are provided in Section 4, and we conclude with a discussion of future work in Section 5.

2   Background

2.1   The Conformal Predictions Framework

The theory of conformal predictions was developed by Vovk, Shafer and Gammerman [1] based on the principles of algorithmic randomness, transductive inference and hypothesis testing. This theory is based on the relationship derived between transductive inference and the Kolmogorov complexity of an i.i.d. (identically independently distributed) sequence of data instances. Hypothesis testing is subsequently used to construct conformal prediction regions, and obtain reliable measures of confidence. The CP framework brings together principles of hypothesis testing and traditional machine learning algorithms through the definition of a non-conformity score, which is a measure that quantifies the conformity of a data point to a particular class label, and is defined suitably for each classifier. As an example, the non-conformity measure of a data point x_i

---

[1] http://scikit-learn.org/stable/

for a k-Nearest Neighbor classifier is defined as:

$$\alpha_i^y = \frac{\sum_{j=1}^{k} D_{ij}^{y}}{\sum_{j=1}^{k} D_{ij}^{-y}} \tag{1}$$

where D_i^y denotes the list of sorted distances between a particular data point x_i and other data points with the same class label, and D_i^{-y} denotes the list of sorted distances between x_i and data points with any other class label. D_{ij}^y is the jth shortest distance in the list of sorted distances, D_i^y. Figure 1 illustrates the idea.
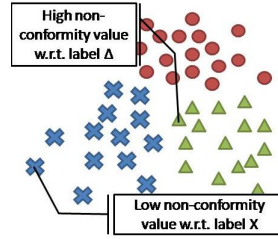
The methodology for applying the CP framework in a classification setting is as follows. Given a new test data point, say x_{n+1}, a null hypothesis is assumed that x_{n+1} belongs to the class label, say, $y^{(p)}$. The nonconformity measures of all the data points in the system so far are recomputed assuming the null hypothesis is true. A p-value function is defined as:

Fig. 1. An illustration of the nonconformity measure defined for k-NN

$$p(\alpha_{n+1}^{y^{(p)}}) = \frac{count\left\{i \in \{1, \ldots, n+1\} : \alpha_i^{y^{(p)}} \geq \alpha_{n+1}^{y^{(p)}}\right\}}{n+1} \tag{2}$$

where $\alpha_{n+1}^{y^{(p)}}$ is the non-conformity measure of x_{n+1}, assuming it is assigned the class label $y^{(p)}$. It is evident that the p-value is highest when all non-conformity measures of training data belonging to class $y^{(p)}$ are higher than that of the new test point, x_{n+1}, which points out that x_{n+1} is *most conformal* to the class $y^{(p)}$. This process is repeated with the null hypothesis supporting each of the class labels, and the highest of the p-values is used to decide the actual class label assigned to x_{n+1}, thus providing a transductive inferential procedure for classification. If p_j is the highest p-value and p_k is the second highest p-value, then p_j is called the *credibility* of the decision, and $1 - p_k$ is the *confidence* of the classifier in the decision. Given a user-specified confidence level, ϵ, the output conformal prediction regions, Γ_ϵ, contain all the class labels with a p-value greater than $1-\epsilon$. These regions are *conformal* i.e. the confidence threshold, $1-\epsilon$ directly translates to the frequency of errors, ϵ in the online setting [2]. The methodology is summarized in Algorithm 1. The CP framework can be used in association with any classifier, with the suitable definition of a non-conformity measure. Sample non-conformity measures for various classification algorithms are presented in Table 1 [1].

2.2   Mondrian Conformal Predictors

The prediction regions output by the CP framework are *valid*, i.e. the frequency of errors is calibrated by the user-defined confidence level. However, in cases of

---

Algorithm 1 Conformal Predictors for Classification

Require: Training set $T = \{(x_1, y_1), ..., (x_n, y_n)\}$, $x_i \in X$, Number of classes M, $y^{(i)} \in Y = \left\{ y^{(1)}, y^{(2)}, \ldots, y^{(M)} \right\}$, Classifier Ξ, Confidence level ϵ

1: Get new unlabeled example x_{n+1}.
2: **for** all class labels, $y^{(j)}$, where $j = 1, \ldots, M$ **do**
3:     Assign label $y^{(j)}$ to x_{n+1}.
4:     Update the classifier Ξ, with $T \cup \left\{ x_{n+1}, y^{(j)} \right\}$.
5:     Compute non-conformity measure value, $\alpha_i^{y^{(j)}} \forall i = 1, \ldots, n+1$ to compute the p-value, p_j, w.r.t. class $y^{(j)}$ (Equation 2).
6: **end for**
7: Output the conformal prediction regions $\Gamma_\epsilon = \left\{ y^{(j)} : p_j > 1 - \epsilon, y_j \in Y \right\}$.

---

imbalanced datasets, where the number of data instances in one class is significantly greater than the numbers in the others, it is possible that all errors may manifest within a single minority class (e.g., cancerous class in cancer prediction problems), limiting the practical applicability of the obtained predictions. In order to overcome this issue, Vovk et al. [1] proposed the Mondrian conformal predictors, where the p-value is computed by comparing the non-conformity score of the new test data instance against only training instances from the same label class as the current hypothesis for the new instance (3) (we address the earlier method as captured in Algorithm 1 as the *Transductive* setting.)

$$p(\alpha_{n+1}^{y^{(p)}}) = \frac{count \left\{ i \in \{1, \ldots, n+1\} : y_i = y^{(p)} \text{ and } \alpha_i^{y^{(p)}} \geq \alpha_{n+1}^{y^{(p)}} \right\}}{count \left\{ i \in \{1, \ldots, n+1\} : y_i = y^{(p)} \right\}} \qquad (3)$$

3   PyCP: An Open-Source Conformal Predictions Toolkit

We now introduce *PyCP*, the first open-source implementation of the CP framework. In the currently available implementation (please see Section 4 for availability), we chose to use the Python 2.7 programming language. (Although adoption of the more recent Python 3 is high, certain critical Python 2 libraries are not yet supported in Python 3.) As mentioned earlier, in order to avoid code duplication of standard machine learning algorithms, we developed *PyCP* on top of the popular *scikit-learn* toolkit [15], an open-source machine learning library for the Python programming language. *scikit-learn* features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, k-means and DBSCAN; and is designed to interoperate with the Python numerical and scientific libraries, *NumPy* and *SciPy*. *scikit-learn* is under active development, and is being used by machine learning researchers around the world.

Classifier	Non-conformity measure	Description
k-NN	$$\frac{\sum_{j=1}^{k} D_{ij}^{y}}{\sum_{j=1}^{k} D_{ij}^{-y}}$$	Ratio of the sum of the distances to the k nearest neighbors belonging to the same class as the hypothesis y, and the sum of the distances to the k nearest neighbors belonging to all other classes (See [1] Chapter 4).
Support Vector Machines	Lagrange multipliers, or a suitable function of the distance of a data point from the hyperplane	Data points on the correct side of the hyperplane have low non-conformity scores, while data points on the margin or the incorrect side have high non-conformity scores [16].
Decision Trees	$$\frac{1}{N} \sum_{i \in J} d(x_j, x_i)$$	Average distance of a data point to all other points sharing the same label. The distance between two points is defined as the length of the shortest path between the terminal nodes containing them. If two points share the same terminal node the distance is 0 (adapted from [1]).
Logistic Regression	$$\begin{cases} -w.x & ,\text{y}=1 \\ w.x & ,\text{y}=0 \end{cases}$$	A monotonic transformation of the reciprocal of the estimated probability of the observed y given the observed x for a given data instance. w is the weight vector typically computed using Maximum Likelihood Estimation (See [1] Chapter 4).

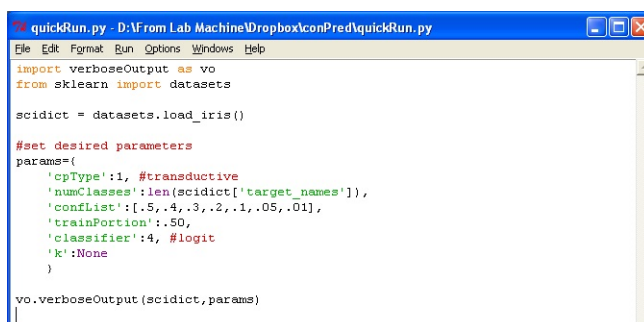**Table 1.** Non-conformity measures for various classifiers
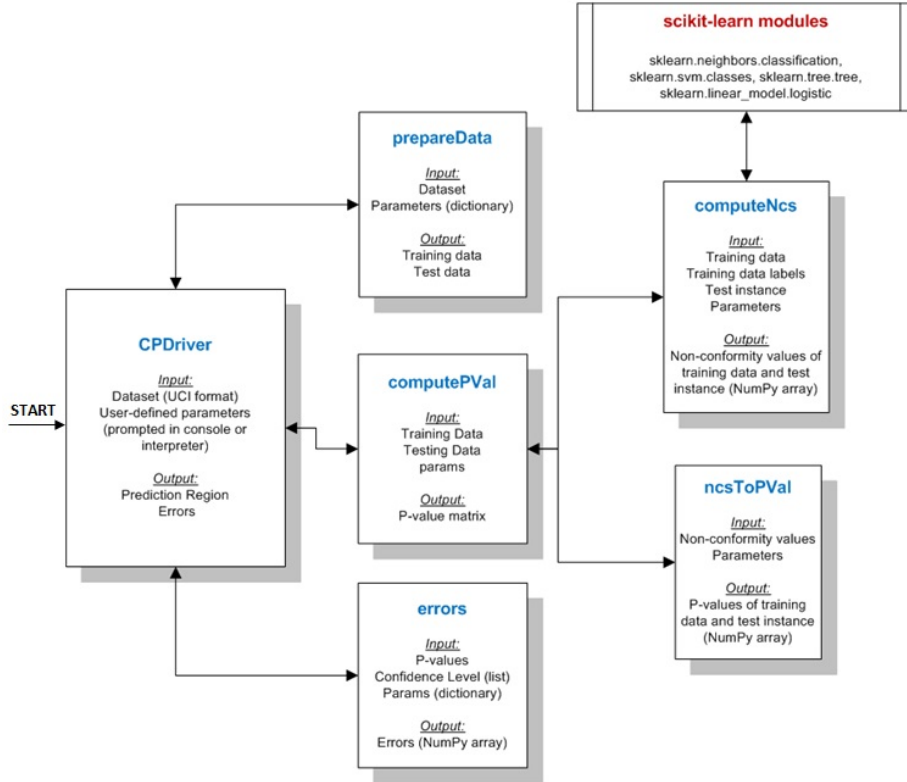
Fig. 2. Overall design of *PyCP*

The overall design of *PyCP* is presented in Figure 2. *PyCP* was implemented with a modular design to allow for easy extensibility. The `prepareData` module randomly samples the data by class and returns a partition of the original dataset into two new datasets: a training set and a test set, based on a user-specified percentage split. Each dataset contains the same proportion of class labels as the original data. The `computeNcs` module computes the non-conformity scores for each of the classifiers as listed in Table 1, by invoking the respective modules for each of the classifiers (`sklearn.neighbors.classification`, `sklearn.svm.classes`,`sklearn.tree.tree`, and `sklearn.linear_model.logistic`) from *scikit-learn*. In case of Support Vector Machines, we use the one-versus-one model for multi-class classification, i.e., if $\alpha_i^{j,k}$ denotes the non-conformity score of point x_i in the $y^{(j)}$ vs $y^{(k)}$ model, the non-conformity score $\alpha_i^{y^{(j)}}$ is computed as the average of the non-conformity scores from all the models involving $y^{(j)}$, i.e. $\frac{1}{n-1}\sum_{k\neq j}\alpha_i^{j,k}$. The `ncsToPVal` module computes the *p*-value of a new data instance with respect to each class label, based on a user-specified choice between the *Transductive* and *Mondrian* settings of the framework. The `computePVal` module iterates over the test instances, invoking the `computeNcs` and `ncsToPVal` modules with appropriate input parameters, and consolidates the p-values for each class label and each of these test instances. The `CPdriver` module serves as a primary reference point, and initiates the implementation of the complete framework by calling all the modules as required. The `errors` module is used to present the final output including errors, empty predictions, multiple predictions and graph plots. We note that the current version of *PyCP* implements CPs in the online setting. While the *PyCP* toolkit supports only the classifiers listed above, the underlying *scikit-learn* toolkit provides a robust set of options for these and other classifiers. To incorporate a classifier not listed here, the modular design of *PyCP* only requires a user to compute an $n \times 1$ non-conformity vector (where n is the number of data points) with a desired non-conformity measure for the new classifier, and store it in a *Numpy* array object within the `computeNcs` module. The classifier can then be invoked from the options in the `CPdriver` module.

4   Availability, Use and Examples

PyCP is currently available for download at: `http://www.public.asu.edu/~v-nallure/conformalpredictions/pycp.html`, and is distributed as open-source under the BSD-2 license. It can be downloaded as a `.zip` file containing the aforementioned `.py` modules, which can be compiled and run through IDLE (the Integrated Development Environment developed for Python) or a similar shell program. On IDLE, *PyCP* is initiated by specifying the options and parameters in `quickRun.py` file. Figure 3 illustrates a sample `quickRun.py` file. As evident, the user can specify the framework setting in `cpType` (*Mondrian*= 0 or *Transductive*= 1), the list of confidence levels at which results are sought in `confList`, the portion of the dataset to be used for training in `trainPortion`, the classifier option in `classifier` (*k*-NN = 1; SVM = 2; Decision Trees = 3;

Logistic Regression = 4), and other classifier-specific paramters such as `k` for the k-NN classifier. Once the parameters are specified, `quickRun.py` can be run on IDLE to obtain the predictions for the test data points. Instructions for using *PyCP* are also available in the `README.txt` included in the downloaded `.zip` file. Figure 4 shows the IDLE shell when `quickRun.py` is executed to obtain conformal predictions for the test data points. *PyCP* currently operates with datasets specified in the `.csv` (comma-separated value) format, as used commonly in the UCI Machine Learning Repository [17].
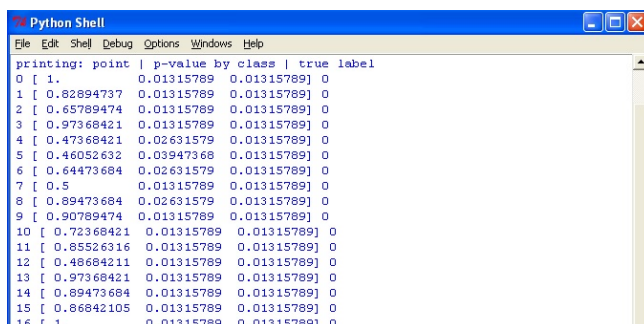
Fig. 3. Specifying options and parameters on `quickRun.py` within an IDLE shell

Fig. 4. Running `quickRun.py` within an IDLE shell

PyCP can also be executed using the IDLE shell console, where the program is driven by a series of prompts requesting user input. Organized console prints walk the user through each input stage and explicitly indicate compatible inputs. Prompting user input in this fashion minimizes the need for tutorial documentation, as the user is presented with every option available in the toolkit at each step. Figure 5 illustrates this mode of execution.

Fig. 5. Running the *CPdriver* module from within an IDLE shell

The user can then select either *Mondrian* or *Transductive* prediction settings (Figure 6), the portion of data to be used for training (Figure 7), the classifier (Figure 8), and any classifier paramters. The `CPdriver` then provides output predictions on the test data points as illustrated in Figure 4.

Fig. 6. The user selects the Mondrian conformal prediction framework.

Fig. 7. The user selects a portion of data to be used for training.

Fig. 8. The user selects the *k*-nearest neighbors classifier.

PyCP returns error counts as three lists; the first lists empty errors by class, the second lists multiple errors, and the third, prediction errors. This helps study both the *validity* and *efficiency* properties of the CP framework (as described in [1]). Finally, *PyCP* also provides plotting capabilities using the *Matplotlib* library. Figure 9 shows a sample plot obtained using *PyCP*.
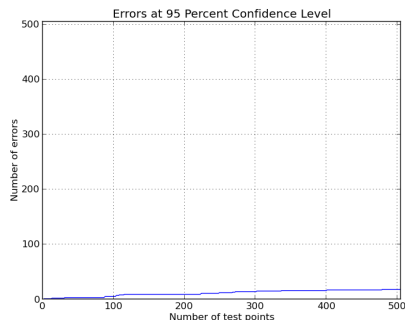
Fig. 9. Sample plot obtained in *PyCP* on the Breast Cancer dataset from the UCI Machine Learning repository [17] using Decision Trees classifier

5   Conclusions and Future Work

In this paper, we have introduced *PyCP*, the first open-source implementation of the Conformal Predictions (CP) framework. *PyCP* is currently available under the BSD-2 license for machine learning researchers and practitioners to apply the CP framework to datasets in their respective domains, as well as for further development by the community. *PyCP* currently provides support for applying the CP framework under classification settings, with four classifiers: k-Nearest Neighbors, Support Vector Machines, Decision Trees and Logistic Regression. *PyCP* also supports the Mondrian setting (for class-conditional validity), in addition to the standard transductive setting of the framework. *PyCP* has been developed using Python 2.7 on top of the popular *scikit-learn* toolkit, an open-source machine learning software developed in Python and used by machine learning researchers across the world. It has been designed in a modular fashion to allow for easy extensibility, and will lead to better experimental reproducibility, incremental advancements of previous work, and more percolation of technical methods into other disciplines and industry.

Our future efforts will include: (i) Extend emphPyCP to include inductive conformal predictors; (ii) Extend *PyCP* to other classifiers and regression settings; (iii) Allow different output options for users to choose from (text output of prediction regions, error graphs at different confidence levels, regions with a single prediction, etc.); (iv) Comprehensive documentation including an informative tutorial and in-line code-level comments; (v) Include a setup script (`setup.py`) and create a source distribution using Python Distutils (distribution utilities), which will allow *PyCP* to be hosted on the Python Package Index (PyPI) online repository; (vi) Develop a graphical interface or visual programming framework for the toolkit; (vii) Support other popular dataset formats such as the `.arff` format from *Weka*; (viii) Port the code to Python 3 when the necessary libraries are available on that platform; and (ix) Make *PyCP* available on `mloss.org`, a web portal exclusively maintained for machine learning open-source software.

References

1. Vovk, V., Gammerman, A., Shafer, G.: Algorithmic Learning in a Random World. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
2. Vovk, V.: On-line confidence machines are well-calibrated. In: 43rd Symposium on Foundations of Computer Science, Washington, DC, USA (2002) 187–196
3. Vanderlooy, S., van der Maaten, L., Sprinkhuizen-Kuyper, I.: Off-line learning with transductive confidence machines: An empirical evaluation. In: Machine Learning and Data Mining in Pattern Recognition. (2007) 310–323
4. Ho, S.S., Wechsler, H.: Query by transduction. Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(9) (2008) 1557–1571
5. Laxhammar, R., Falkman, G.: Conformal prediction for distribution-independent anomaly detection in streaming vessel data. In: Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques. StreamKDD '10, New York, NY, USA, ACM (2010) 4755
6. Bellotti, T., Luo, Z., Gammerman, A.: Strangeness minimisation feature selection with confidence machines. In: Intelligent Data Engineering and Automated Learning IDEAL 2006. (2006) 978–985
7. Ho, S.S., Wechsler, H.: A martingale framework for detecting changes in data streams by testing exchangeability. IEEE Transactions on Pattern Analysis and Machine Intelligence **32**(12) (2010) 2113–2127
8. M. Kukar: Quality assessment of individual classifications in machine learning and data mining. Knowledge Information Systems **9**(3) (2006) 364–384
9. Lambrou, A., Papadopoulos, H., Gammerman, A.: Reliable confidence measures for medical diagnosis with evolutionary algorithms. Information Technology in Biomedicine, IEEE Transactions on **15**(1) (2011) 93 –99
10. Shao, H., Yu, B., Nadeau, J.H.: Strangeness-based feature weighting and classification of gene expression profiles. In Wainwright, R.L., Haddad, H., eds.: SAC, ACM (2008) 1292–1296
11. Li, F., Wechsler, H.: Open set face recognition using transduction. IEEE Trans. Pattern Anal. Mach. Intell. **27**(11) (2005) 1686–1697
12. Dashevskiy, M., Luo, Z.: Network traffic demand prediction with confidence. In: GLOBECOM. (2008) 1453–1457
13. Sonnenburg, S., Braun, M.L., Ong, C.S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Mller, K.R., Pereira, F., Rasmussen, C.E., Rtsch, G., Schlkopf, B., Smola, A., Vincent, P., Weston, J., Williamson, R.: The need for open source software in machine learning. Journal of Machine Learning Research **8** (2007) 2443—2466
14. Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rckstie, T., Schmidhuber, J.: PyBrain. J. Mach. Learn. Res. **11** (2010) 743–746
15. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, d.: Scikit-learn: Machine learning in python. Journal of Machine Learning Research **12** (2011) 2825–2830
16. Balasubramanian, V., Gouripeddi, R., Panchanathan, S., Vermillion, J., Bhaskaran, A., Siegel, R.: Support vector machine based conformal predictors for risk of complications following a coronary drug eluting stent procedure. In: Computers in Cardiology, 2009. (2009) 5 –8
17. Bache, K., Lichman, M.: UCI machine learning repository (2013)