UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Rain Vagel

Developing a Scikit-learn Module for a Novel Data Partition for Machine Learning

Bachelor's Thesis (9 ECTS)

Supervisor: Raul Vicente Zafra, PhD
Supervisor: Kristjan Korjus, MMath

Tartu 2017

Developing a Scikit-learn Module for a Novel Data Partition for Machine Learning

Abstract: Machine learning is the field of using data and statistical models to make predictions. With the help of data partitioning schemes, researchers are able to efficiently test and report accuracies or error values of their models with limited data. Depending on the partitioning scheme, other helpful results, such as hyper-parameters of the model, can be returned. A new data partitioning scheme, cross-validation & cross-testing, has been discovered. However it is not yet widely used due to the fact that currently no open-source machine learning library has a function for it. In this thesis we will publish scikit-learn compatible function on Github and also implement it on different tasks. This new function can be used by anybody under an open-source license. Our tests showed that this new partitioning scheme might perform slightly worse on regression tasks, than was previously thought. For this we must study cross-validation & cross-testing further, to better understand and to further facilitate its use.

Keywords: machine learning, data partitioning, scikit-learn

CERCS: P170

Scikit-learni mooduli arendamine uue masinõppe andmejaotuse jaoks

Lühikokkuvõte: Masinõpe on ala, kus tehakse andmete ja statistiliste mudelite põhjal ennustusi. Andmejaotuse abil saavad arendajad efektiivselt testida ja raporteerida enda mudelite täpsust või veamäära piiratud andmehulkade puhul. Andmejaotusest olenevalt tagastavad need meetodid ka erinevaid mudelit kirjeldavaid näitajaid, näiteks hüper-parameetreid. On avastatud uus andmejaotamise meetod nimega rist-valideerimine & rist-testimine. Kuid see pole hetkel laialdast kasutust leidnud, sest ükski avatud lähtekoodiga masinõppe teek ei kaasa seda. Selle töö raames arendame me scikit-learni jaoks sobiva mooduli ning rakendame seda erinevatele ülesannetele. Arendatud moodul on varustatud avatud lähtekoodi litsentsiga, mis tähendab, et kõik saavad seda vabalt kasutada. Esmased katsed näitavad, et uus andmejaotuse meetod võib regressiooni ülesannetel anda halvemaid tulemusi, kui alguses ootasime. Selleks peab rist-valideerimist & rist-testimist rohkem uurima, et paremini mõista ja rohkem kasutada seda uut andmejaotuse

skeemi.

Võtmesõnad: masinõpe, andmete jaotamine, scikit-learn

CERCS: P170

Contents

1 Introduction

Machine learning has steadily become more popular due to the large amounts of data that is being gathered each day. This data can be explored and used to make predictions on future events with the help of machine learning models. However when a machine learning method or model is used on data, the researcher must be able to make sure that the model will generalise well to new data, while also not overfitting with the data that was used to train it. This is why data partitioning schemes like cross-validation and splitting datasets into training and testing data is done. Leaving a part of the dataset to be used for testing purposes can tell us how good the model is at what it is supposed to do. This however leaves a smaller portion of the data for training purposes. While not so much of a problem in fields where data is cheap, it is a problem in biological fields, were data is often extremely expensive to gather.

This is why Korjus et al. [1] have proposed a new data partitioning method, cross-validation & cross-testing. This methods aims to be a middle ground between the two already widely used data partitioning schemes, cross-validation & testing and nested cross-validation. It has the advantages and disadvantages of both.

The authors objective is to encourage use of this novel data partitioning scheme by implementing it according to scikit-learn guidelines, documenting the code, illustrating it with examples and also by applying it to real datasets on which we perform different classification and regression tasks. The code with the documentation and examples will be made public and will hopefully have a place at scikit-learns list of Related Projects, if not added into the core package.

This thesis is divided into six sections. The first section 1 is this one, which introduced the thesis. The second section 2 is Background and related work and will lay the ground for understanding this thesis and the current situation in machine learning tools similar to scikit-learn. The third section 3 is Methods and Materials. It will explain the different machine learning methods, data partitioning schemes and datasets that will be used in the thesis. The fourth section 4 is Results. In that section the implementation of cross-validation & cross-testing will be explained and the results of the tests will be written. The fifth section 5 is Discussion. There we will take a closer look at the results and implementation of cross-validation & cross-testing. The sixth and final section 6 is Conclusion. It will present a brief summary of the thesis.

2 Background and related work

In this section there are three distinct subsections. The first is for machine learning in which we take a closer look at what machine learning is and why there is such an interest for it. In the second section we will take a look at scikit-learn, what it is and the design philosophy behind it. In the third section we will take a look at machine learning libraries and programmes that are similar to scikit-learn, yet all have their specific differences.

2.1 Machine learning

Machine learning is both widely used and gaining in popularity year after year. It makes use of the huge amounts of different data that is being gathered every day on the internet. In principal machine learning is all about building a statistical model with data that we already have to then use the model to make predictions on new data that it has not seen before. There are three major machine learning types.

Supervised learning is when a researcher has a dataset that consists of not only the input data, but also the output data. A model is trained with both the input data and output data, so that for certain feature values it will know what the output will be. However as for any model, it is important that it does not overfit, as otherwise it will be unable to make accurate predictions on new data.

In semi-supervised learning the model will learn by first making some predictions, before it is told what went right and what did not. The same process is then repeated until the training is completed and it can be used to make predictions.

In unsupervised learning the model must learn totally by itself on training data, it will not see any correct data and will not be assisted in any way.

In order to know if the model works correctly, it is usually tested on a different test set, that was not used for training. While cross-validation & testing is the most widely used to make the most out of the available, there are two classical alternatives to cross-validation & cross-testing as described by Korjus et al. [1] and further analysed in this thesis. In a way all of them are similar, yet answer different questions and and make use of the data (for training and testing) in a different way, resulting in different predictive power [1]. The first partitioning scheme is Cross-validation & testing, which is the easiest method the implement and use. It is also the only one of the three which returns the predicted accuracy, statistical significance and a model that can be used in the future [1]. However with this method, not all of the available data is used to train the classifier and as such its performance suffers.

The second classical partitioning scheme is nested cross-validation. This partitioning scheme uses all of the available data in order to train and test its models. However due to the nature of the method, it will not return any model that would generalise well to unknown data [1]. This method is used mainly to show if there is a statistical dependence on the class labels and dataset. It is also impossible to give the accuracy or statistical significance of any single model, since each time there is a new set of parameters and a new training set.

2.2 Scikit-learn

Since the goal of this thesis is to develop a scikit-learn module that would make the use of cross-validation & cross-testing easy for users to implement, scikit-learn will also play a center role in developing the module and running tests on the three different data partitioning schemes.

Scikit-learn was developed as a comprehensive machine learning library for Python with some very important design principles in mind. As Buitinck et al. [2] have written, the library must be easy to use even for beginners [3], while also having flexibility and a lot of customisation options and it must also be very efficient. This is the reason why they have decided to stick to an API and have very strict developing requirements.

All of the different machine learning models or supporting functions in this thesis are implemented in scikit-learn and are ready to use out-of-the-box, or have been developed with the help of different scikit-learn functions. For ease of use, all scikit-learn functions that require user input have default values that have been designed so that the function is as useful in a wide variety of situations as possible.

Another design philosophy of scikit-learn is that almost everything must be fit into the pipeline for ease of use through the API. This would allow an user to with ease switch out a preprocessing step without having to touch the rest of the pipeline. This makes the library easy and fast to use.

2.3 Related work

Although scikit-learn is one of the most widely used Python machine-learning libraries, there are some competitors for it. Some, for example Weka [4] or Orange [5] offer APIs, but focus mainly on the graphical user interface, which allows programming novices to easily and quikcly implement their desired algorithms. Scikit-learn on the other hand has decided that their target audience is capable enough in programming to be able to use their API [2].

Other packages are developed as command-line tools, which sometimes do not offer any API to the users, at all. Some of the for example are SofiaML [6] and Vowpal Wabbit. These types of packages do not explicitly require the use of any

programming languages, but do require the users to program their input/output. While scikit-learn is only usable in Python, these command-line tools can be used with a very wide variety of different languages.

There are also domain specific languages such as Matlab or R. They are very powerful when it comes to working on data, doing computations and plotting the results. However scikit-learn has the advantage of being implemented in, and using, Python, a general purpose programming language that is used for a very wide variety of applications. It also has the advantage of being open-source and free to use while Matlab requires the purchase of a license.

3 Methods and Materials

In this section we will be looking at the different datasets and methods that will be used in this thesis. We will describe all of the datasets and methods, while also giving the reader an overview of the different data partitioning schemes, their strengths and weaknesses. By the end of this section the reader should have enough material to understand and implement the different partition schemes and apply them on the described datasets.

3.1 Datasets

In this subsection we will look at the different datasets that will be used with the different data partitioning schemes. We will be using the Spikes dataset used in the article by Korjus et al. [1], the Iris dataset, the Boston housing dataset and a simulated dataset that has been generated according to a given input-output relation and used to validate the results.

3.1.1 Spike train data

This dataset was one of the classification datasets used by Korjus et al. and it contains multiple single unit recordings of different neurons in a region of the brain of a rat known to be involved in processing spatial information and navigation (CA1 region of the hippocampus)[1]. The data includes spikes of 61 neurons and a sample of 2000 data points while the rat navigated in a square-shaped arena. The arena was artificially divided into to areas to assign different types of labels according on which side of the arena the rat was currently in. The Spike dataset is accessible from crcns.org[1, 7].

This dataset contains a total of 61 features and 2000 samples in each class. It is used to validate Korjus et al.'s results with a data set that was used in their paper, thus proving that the code is valid and can be used in new situations.

3.1.2 Iris dataset

The iris dataset is very popular dataset that comes with Scikit-learn. This classification dataset was first introduced by Fisher[8]. As the name suggests, it has four features, all of which describe the specific iris flower and three classes, each of which consists of 50 samples.

This data set will be used to validate Korjus et al.'s results with a data set that was not used in that paper.

3.1.3 Simulated data

The simulated data set is generated by the author with the following formula:

$$y = x^3 - x^2 + U(0, 1) \tag{1}$$

Where x is sampled uniformly between 0 and 2, and $U(0, 1)$ denotes a random number sampled from the interval (0,1). It is added in order to add an element of randomness to the formula.

With the formula, a dataset of one feature (x) and 280 samples was generated to investigate the performance of regression models using the different partition methods.

This data set is used to generate quite a simple single feature polynomial data set on which regressions models of different polynomial degree can be tested on.

3.1.4 Boston dataset

The Boston housing dataset, as it is formally called, contains information collected by the U.S Census Service concerning housing in the Boston Mass area and was first published in 1978 [9]. This dataset is very commonly used for benchmarking and for educational purposes, as it is already built into scikit-learn and as such is easy to access.

The dataset has 14 attributes, out of which two can be predicted. One attribute is of boolean value, the Charles River dummy variable.

Two different attributes can be predicted with the dataset. First is the nox, nitric oxides concentration (parts per 10 million). Second is the median value of a home in thousands. In this thesis we predict the housing price.

3.2 Machine learning tools

In this subsection we will be looking at the different machine learning methods and statistical models that will be used in this thesis.

3.2.1 LASSO

LASSO stands for *Least Absolute Shrinkage and Selection Operator*. As it deals with the absolute value of coefficients and also aims to use the least amount of features possible, which is why we decided to use it for our Boston dataset, as it has a large number of features.

Lasso performs L1 regularisation, which in principle is a technique to reduce model overfitting. Overfitting is bad, because although during training and testing

the model reports a low mean square error, it will not be practically useful, as it conforms too much to the existing data points. Regularisation prevents the model weights from becoming very small or large, which would mean that the prediction curve is very complex and the model will overfit.

Regularisation penalises model weight values by adding those weight values to the calculation of the error term. L1 regularisation penalises weight values by adding the sum of their absolute values to the error term.

3.2.2 Support vector machine

Support vector machines are commonly used supervised learning tools that in scikit-learn come in three groups: classification, regression and outliers. However only two groups are used in this thesis, classification and regression.

As explained by Bennett and Bredensteiner [10]. A support vector machine must find the best separating line between two or more classes. Lets have an example where we only have two classes. The machine finds the best separating line between the two classes by first finding the points from the two classes which are closest to one another and then uses vector subtraction to draw a line that connects them. It will then declare the line that bisects the line between the two points as the best separating line.

According to the scikit-learn website they have the following advantages [11]:

- Effective in high dimensional spaces - High dimensional means that the data set has a high number of features.
- Effective in cases where number of dimensions is greater then the number of samples.
- Uses a subset of training points in the decision function, which means that it is memory efficient.
- Different kernel functions can be specified for the decision function. It has a set of kernels provided and the user can specify custom kernels - A kernel is basically a similarity function that takes two inputs and computes how similar they are. Depending on the type of kernel it will have different formulas.

They also have the following disadvantages [11]:

- Will give poor performance if the number of features is much greater than the number of samples.
- SVMs do not provide probability estimates, they are instead calculated with an expensive five-fold cross-validation

3.2.3 Grid search

In scikit-learn each estimator object has parameters that are called hyper-parameters. These hyper-parameters describe the estimator object. Each estimator object has default hyper-parameter values so in principle they do not have to be changed at all in order for the estimator to work. However they frequently have quite a large impact on the performance of the estimator so the user should aim to optimise them for the data at hand.

However the parameter space for each estimator is essentially endless. The parameter space is all of the possible values for all of the hyper-parameters. The user can not be expected to manually try out every possible combination of hyper-parameters and their valuations. For this scikit-learn has implemented hyper-parameters optimisation functions. One of them is GridSearchCV, which works on cross-validation and is used in this thesis as well.

The following paragraph is referred from scikit-learns documentation on GridSearchCV [12]. GridSearchCV is an exhaustive hyper-parameter grid search. What this means is that the user gives it a parameter grid or a list of parameter grids. Each parameter grid in turn consists of the hyper-parameter names and a list of possible valuations for each hyper-parameter. GridSearchCV will then explore all possible combinations of hyper-parameter valuations with the help of cross-validation and will save the best parameters and the best estimator.

3.3 Data partitioning schemes

In this subsection we will look more closely into what cross-validation actually is and currently the two main schemes that employ it, cross-validation & testing and nested cross-validation. We will also describe what cross-validation & cross-testing is, as well as give examples and figures in order to help understand these schemes. By the end of this subsection the reader should have full understanding of the pro's and con's of the different data partitioning schemes, as well as have enough knowledge to implement them on his own.

3.3.1 Cross-validation

Cross-validation uses data near optimally to train and test classifiers on different subsets of data in each iteration. This makes sure that all of the data is used to train the classifier at least once and as such uses all of the data for training and testing, but the same data is never in the training and testing set at the same time.

For example if we have a 5-fold cross-validation, then in each iteration the data is split so that 80% is in the training set and 20% is in the testing set. This split is

repeated differently five times until all of the data has been in the training set at least once. This assures that the best parameters for the given classifier have been found. Usually about five to ten fold cross-validation is used as it strikes the best balance between over-fitting and giving the best training size for the classifier. [13]

All of the data partitioning schemes in this paper make use of cross-validation in one way or the other and it is one of the most generally used methods in machine learning.

3.3.2 Cross-validation & testing

The most used and general scheme that uses cross-validation is cross-validation & testing. In this scheme cross-validation is used in order to find the best parameters for the machine learning model. As Korjus et al. [1] put it, the objective is to build a model that performs well in real life applications. This means that the model must generalise well to unseen data. To achieve this, the researcher must the largest training set that they can, while also making sure that the model does not overfit. This means that they also need a sizeable test set to find any possible overfitting. See Figure 1 for reference.

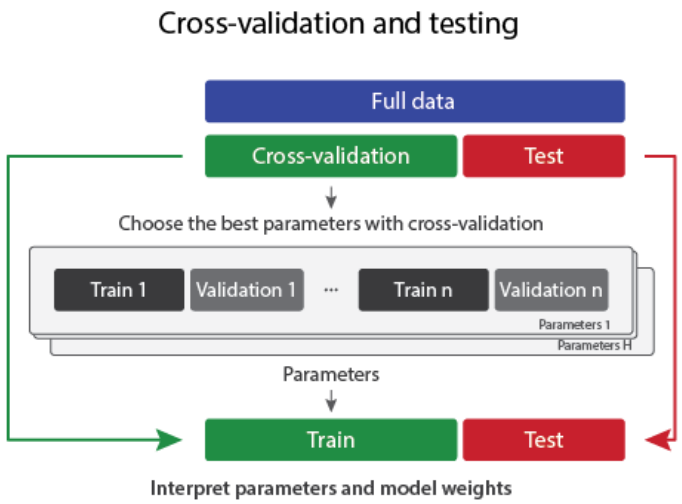

Figure 1: Cross-validation & testing[1]. First the best hyper-parameters are chosen with cross-validation and then a model is trained with them. The trained model is tested with the test set, which was split off at the start of the process.

An example of an implementation would be as follows. First the full data set is split into a cross-validation set and a test set. A test set size of about

10% - 20% is usually large enough to detect any overfitting, while also giving a sufficiently sized training set for the model [13]. The test set is left alone for now and cross-validation is done on the cross-validation set and for each iteration, the cross-validation set is again split into a training set and a validation set. This split is done by the five to ten fold rule mentioned above. From this we get a model in which we know what the best hyper-parameters and parameters of the model are and can then use the entire cross-validation set to train it, after which we will use the until now unused test set to test the model.

With this scheme we get a model that typically generalises well to unseen cases. We can also get interpretable parameters and hyper-parameters of the model with this scheme.

However an important issue is that it does not use the entirety of the data set for training the model. Instead a sizeable chunk is left out in order to later test it. This is an issue with data that is expensive to gather and compile, for example biological data, especially neurological [1]. The next scheme fixes this issue with a certain cost.

3.3.3 Nested cross-validation

Nested cross-validation aims to solve the issue of not using all of the available data to train a statistical model by nesting an internal, or nested, cross-validation into an external cross-validation. This method makes full use of all the available data, but will not return a statistical model, because it will each time pick a different model. See Figure 2 for reference.

The external cross-validation will split the full dataset into a cross-validation set and a test set. This is done with the help of any number of different splitting methods. An inner cross-validation is then performed on the cross-validation set. In this cross-validation step the best parameters are chosen and then a model is trained on the entire cross-validation set and tested on the test set. This will then be repeated for each iteration in the outer cross-validation step, each time giving a different cross-validation and test set for the inner cross-validation to use.

While this method uses all of the data for model training, it has some very specific drawback. It does not have a separate test set and chosen models and hyper-parameters and model weights might change with every outer cross-validation iteration. This means that it is impossible to pick one specific model that could be used on new data [14], since in each iteration a new model is possibly being used.

This cross-validation approach, while not returning any models or parameters that could be used in the future, is however useful as it can be used to show that there are indeed dependencies between target values or classes and the dataset.

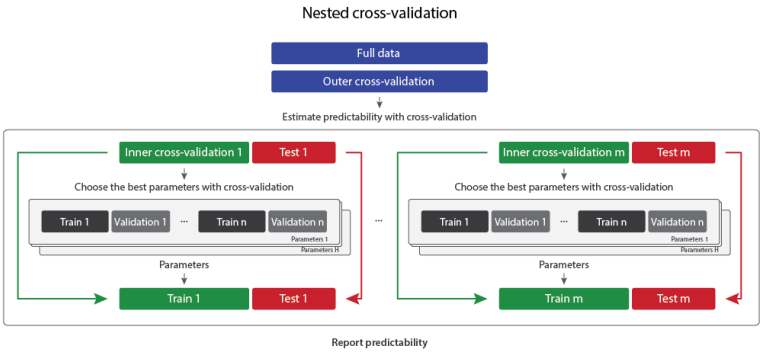

Figure 2: Nested cross-validation[1]. First an outer cross-validation is performed to get the cross-validation set and the test set. Then an inner cross-validation is performed on the new cross-validation set to get the best hyper-parameters and model coefficients.

3.3.4 Cross-validation & cross-testing

This novel cross-validation & testing approach is proposed by Korjus et al. [1] and aims to make more efficient use of the entire dataset while fitting a model, but it will also maintain the interpretability of hyper-parameters [1]. See Figure 3 for reference.

Like cross-validation & testing, it has a smaller cross-validation size than nested cross-validation, for choosing the hyper-parameters, but uses all of the available data to fit a model. This is done by the "cross-testing" method, that is being used instead of the normal testing method in cross-validation & testing and within nested cross-validation. Similar to cross-validation, cross-testing will cycle through the testing set, each time splitting off a piece and putting it into the training set. This continues until all of the test set data has been in the training set at least once.

Because in each cross-testing iteration the model is being fitted with new data, the approach can not return a single statistical model, that could be then used to make predictions on unseen data. However it can return the average score of the hyper-parameters and tell the researcher if there are dependencies between the data and the results. However most importantly it returns the best hyper-parameters that it could find for that data. This means that the researcher is able to describe the model and use those hyper-parameters to train the model on the full dataset.

In terms of advantages and disadvantages, this approach is a middle ground between cross-validation & testing and nested cross-validation. It makes a more

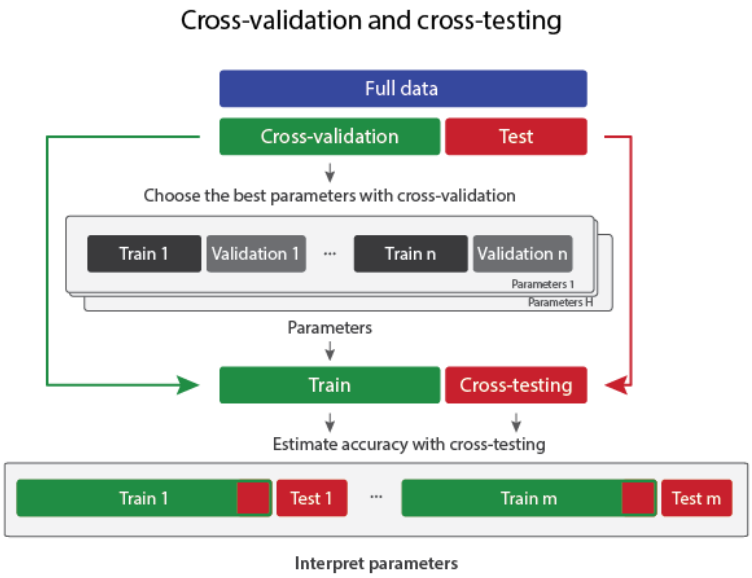

Figure 3: Cross-validation & cross-testing[1]. Everything is the same until the testing phase. In the testing phase, we perform cross-testing, which is very similar to cross-validation in which we iterate through the test set until all the data in the test set has been in the training set.

efficient use of data, which means that the statistical dependency and accuracy of its models is better than cross-validation & testing, but it can not return a model that can be later used on unseen data [1]. However unlike nested cross-validation, it does not use all of the data for model training and such is not as accurate, however it can return interpretable hyper-parameters.

4 Results

In this section we will first discuss the requirements that scikit-learn has on any kind of contributions that are submitted to its core library and our implementation and publishing of cross-validation & cross-testing. Finally the results from applying the different data partitioning schemes and machine learning methods on the four datasets will also be shown here.

4.1 Implementing cross-validation & cross-testing

The main objective of this thesis was to create an easily usable implementation of cross-validation & cross-testing that would be compatible with scikit-learn. For this the module must follow strict scikit-learn guidelines about coding style, documentation and test coverage. We will look at the different requirements before taking a more specific look at the code.

4.1.1 Coding style

Since scikit-learn is a very large open-source Python project, it has a large number of people constantly making contributions. However, it has a small group of core developers or maintainers of the code base and before any contribution makes its way into the codebase, it will be checked by two different core developers.

Strict coding guidelines that everybody must follow make working on open-source projects very easy. Following a guideline insures that everybody is writing their code the same way, this makes it easier and faster for developers to read each others code which in turn makes it easier for the core developers to check contributions before they are accepted into the core codebase. It also makes it much simpler for new people to get started contributing to the project.

As such, scikit-learn has enforced strict coding guidelines that are based on the PEP8 guidelines published by the Python Software Foundation [15]. However they have added a few guidelines in order to make it more suitable for their needs as from the scikit-learn webpage:

1. Underscores must be used to separate words in non class names: n_samples instead of nsamples.

2. Must avoid multiple statements on one line and preferably use a line return after a control flow statement.

3. Use relative imports for references inside scikit-learn, except for unit tests. These should use absolute imports, as client code would.

4. import * must not be used in any case as it is considered harmful by the official Python recommendations. This would make the code harder to read as the origins of symbols is no longer explicitly referenced and it prevents the use of static analysis tools, which are used in scikit-learn to automatically find bugs,

5. All docstrings must use the numpy docstring standard.

All code that is being submitted to scikit-learn must follow those strict guidelines in order to be accepted.

4.1.2 Documentation

For an open-source project the size of scikit-learn, good documentation is important to make it easier to use and to contribute to. This is why everything must be documented with at least one paragraph of narrative documentation. This documentation should include links to references in literature. If applicable, algorithms or mathematical equations should be included in the documentation, but they must be accompanied with narrative text that helps the reader get an intuitive understanding of the algorithm.

Scikit-learn suggests the following documentation structure to make their project uniform and easy to understand:

1. A small paragraph with a hand-waving explanation of what the method does.

2. Point out why the feature is useful and when it should be used. Also include here big O complexities of the algorithm or rules of thumb, if the big O complexities are not available.

3. A generated figure from an example.

4. Mathematical equations followed by references. Adding them later makes the documentation more friendly for users who are just interested in what the feature will do.

5. "See also" section that lists references to related classes or functions

Following the above formula while writing your documentation will make it easy to read and give it a uniform look.

4.1.3 Testing

Scikit-learn requires high-quality unit testing in their core library. For this they use the nose package, a unit test framework made for python that can run doc tests, unit tests and "no boilerplate" tests. Code coverage of new features is expected to be at least 90%.

4.1.4 Implementation

For this thesis we have chosen to implement the cross-testing section of the cross-validation & cross-testing method. Cross-validation has already been made available and incorporated into multiple different functions. This and scikit-learns design principles of ease of use and composition [2] resulted in the decision of a separate cross-testing function.

Intuitively, the function works as follows:

1. Split the test set according to whatever cross-validation function the user has chosen, if none is chosen, a standard 3-fold cross-validation will be used.
2. Add the nth fold to the training set, while the $n - 1$ folds make up the test set.
3. Train the model given to the function with the new training set.
4. Test the model with the new test set.
5. Output the average score and hyper-parameters.

Do note that if the user does not input any hyper-parameters into the function, then the hyper-parameters that the statistical model already has, will be used and returned instead. Because by design everything that requires user-defined parameters in scikit-learn has appropriate default values [2], it will always return some hyper-parameters.

For the sake of keeping the documentation as uniform as possible between different scikit-learn functions, the documentation below of both the inputs and the outputs is very heavily based on the documentation of the GridSearchCV function [12].

The inputs of the function are as follows:

estimator : estimator object.

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function or scorer must be passed.

X_train : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vectors that are part of the training set, where n_samples is the number of samples and n_features is the number of features [12].

X_test : array-like, shape (n_samples,)

Target values that are part of the training set (class labels in classification, real numbers in regression)

Y_train : {array-like, sparse matrix}, shape (n_samples, n_features)

Training vectors that are part of the test set, where n_samples is the number of samples and n_features is the number of features.

Y_test : array-like, shape (n_samples,)

Target values that are part of the test set (class labels in classification, real numbers in regression)

cv : "int, cross-validation generator or an iterable, optional

Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 3-fold cross-validation,
- Integer, to specify the number of folds in a (Stratified)KFold,
- An object to be used as a cross-validation generator,
- An iterable yielding train, test splits.

For integer/None inputs, if the estimator is a classifier and the target is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used." - as from the GridSearchCV documentation [12].

scorer : callable or None, default=None

Scorer callable object / function with the signature scorer(esimator, X, y). If None, the score method of the estimator is used.

model_params : dict, optional

Hyper-parameters to pass to the estimator.

The function returns the following value:

scores : list

A list of scores.

The module is available publicly under the BSD 3-clause license on Github at: `https://github.com/RainVagel/cross-val-cross-test`. Examples on how to use it are in the doc string in the source code and the repository also has an example on the Iris data set.

4.2 Applications to datasets

We applied machine learning models to four datasets in total: Spike, Iris, Simulated and Boston datasets. In this section we will be looking into how the different partitioning schemes and machine learning methods behaved on the above mentioned datasets. We will run classification models on Spike and Iris, regression on Simulated and Boston. For a more detailed description of the datasets, please see section 3.1. Nested cross-validation will not be run on tests where the dataset size is constant, while the test set size changes, as nested cross-validation always uses all of the dataset for testing purposes.

The models will be run 200 times on each dataset with the specific dataset variables. In particular, we explore the performance of different data partition schemes when the test set size and the total size of the dataset are varied. Different possibilities of test set sizes were: 5, 10, 20, 25, 30, 35, 40, 45, 50 percent. The total size of the dataset depended on the number of samples from the complete dataset that we sampled it from.

4.2.1 Spike dataset

In order to validate the code of this thesis, we chose to try and replicate Korjus et al.'s implementation as closely as possible. Here we used SVM as a classifier of the rat location as a function of its neuronal features.

The hyper-parameters consisted in the value C (penalty for misclassification) and the kernel (transformation applied to the features before classification). In particular C was scanned at values: 0.0001, 0.01, 1.0 and since we did not use any form of preprocessing on this data, we picked the other hyper-parameter to be the kernel, with the possibilities being: linear, polynomial or radial basis functions. We applied it to dataset sizes ranging between 20 and 260, with a step of 10.

Changing dataset size. On Figure 4 we see that cross-validation & cross-testing outperforms cross-validation & testing. It also for the most part outperforms nested cross-validation. However nested cross-validation has the lowest deviation throughout the figure. We see that in terms of performance when comparing the three partitioning schemes, they were similar to what Korjus et al. [1] reported.

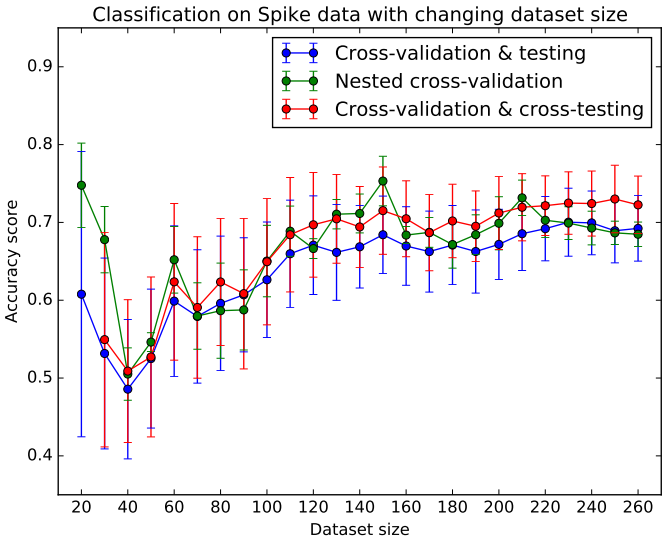

Figure 4: Accuracy score of a support vector machine implementation on the Iris dataset plotted against changing dataset size.

What is noteworthy is that cross-validation & cross-testing failed to report an accuracy score at a dataset size of 20 samples.

Changing test set size. On Figure 5 we see that cross-validation & cross-testing reported a higher accuracy score throughout the entire test.

However we see the usual curve in which the higher the test set size is relative to the dataset size, the lower the reported accuracy goes.

4.2.2 Iris dataset

On this dataset we applied the classification support vector machine. The parameter grid was kept the same as we used on the Spikes data set. This was to run identical programs with identical parameters on the different datasets. However the Spikes dataset has 61 features, while the Iris dataset only has four.

Since the Iris data set is much smaller, having a maximum of 150 samples, we scanned the datasets size by subsampling the original data to produce datasets between 20 to 150 samples, with a step of 10.

Changing dataset size. The three different cross-validation methods delivered similar results, staying quite close to one another. As can be seen from Figure

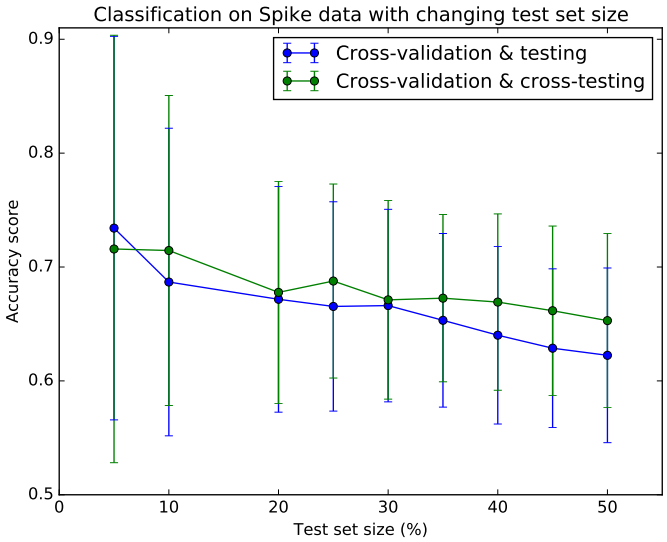

Figure 5: Accuracy score of a support vector machine implementation on the Spikes dataset plotted against changing test set size.

6, the accuracy rose significantly between the data set sizes of 20 and 60, slightly dropping right after that. However note that the drop is only a few percentage points. The accuracy score of cross-validation & cross-testing always stays close to the other two and very frequently inhabits the space between the two. We can also observe that with a bigger dataset, the standard deviation for the three partitioning schemes also decreased.

Changing test set size. Here we have the data set size fixed to 100 samples and only change the size of the test set relative to the dataset. We can observe that the reported accuracy score in Figure 7 stays quite stable for the two different data partitioning schemes.

We can however see that the standard deviation slightly decreased as the test set size increases.

4.2.3 Simulated data

On the simulated data we applied a support vector machine, more specifically epsilon-support vector regression. We chose this because this model had polynomial degrees as one of its hyper-parameters. The data has one simple feature, an x value, and the answer, an y value. The y value was returned by a cubic polynomial

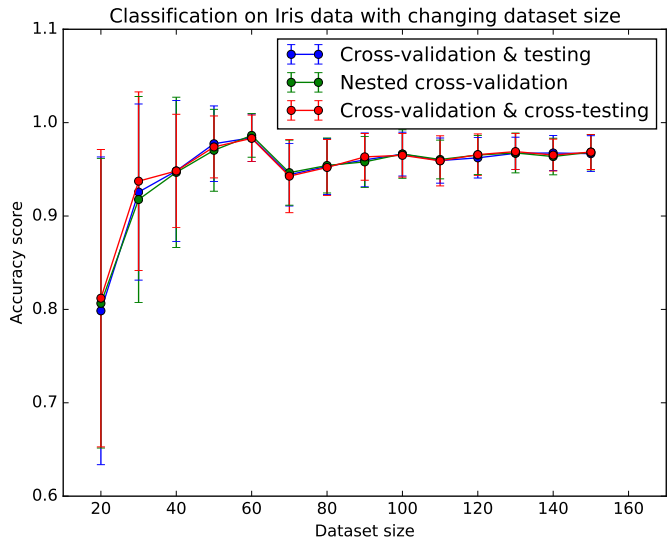

Figure 6: Accuracy score of a support vector machine implementation on the Iris dataset plotted against changing dataset size.

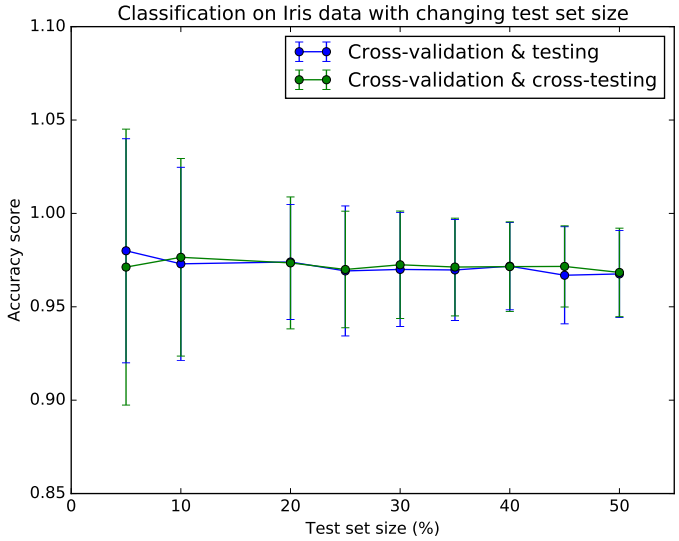

Figure 7: Accuracy score of a support vector machine implementation on the Iris dataset plotted against changing test set size.

function which took in x as a parameter. For more info on how the dataset was generated, look at section 3.1.3.

The data has a total of 260 samples, which means that the data set size varied between 20 and 260 with a step of 10.

Changing dataset size. On Figure 8, we see results in which again the three methods perform very similar to one another. They also do not lose much in median absolute error, but do lose a lot of standard deviation, so the results have converged much more on the value shown on the graph.

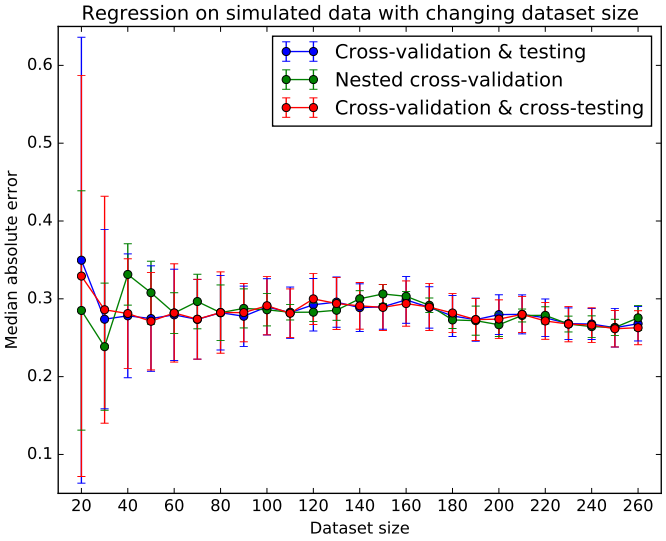

Figure 8: Median absolute error of epsilon-support vector regression implementation on Simulated data plotted against changing dataset size.

We can also see from Figure 8 that it follows the already familiar curve of reporting a lower median absolute the more data we have to work with, although the test set size is constantly 50% of the data set size. After 240 samples the three methods seem to almost converge.

Changing test set size. On Figure 9, we see that the two data partitioning schemes reported quite similar median absolute error values with neither actually really getting any meaningful headway over the other.

From Figure 9 we again see that as the test set size increased, the standard deviation slightly decreased.

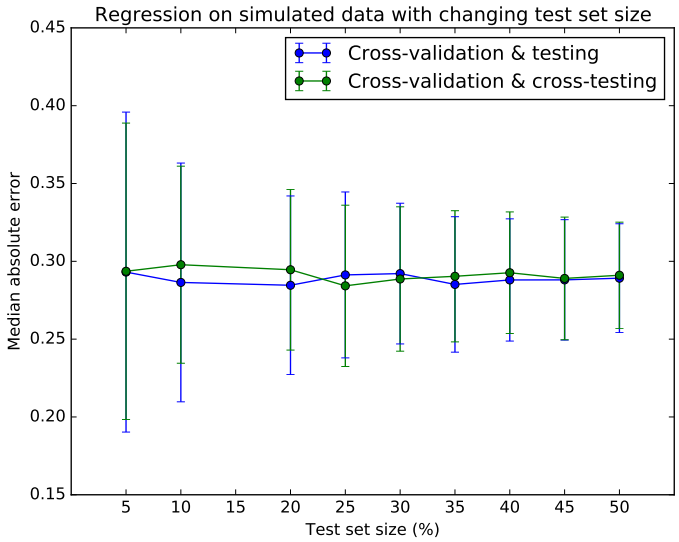

Figure 9: Median absolute error of epsilon-support vector regression implementation on Simulated data plotted against changing test set size.

4.2.4 Boston dataset

With the Boston dataset, which consists of housing information collected in the 1970's in the Boston Mass area, we dropped one of the features in order to have all of the features be floating point values. The feature that we dropped was a boolean value. The other values were all then normalized into values between 0 and 1. The model that we used was LASSO, as it works well with data sets that have bigger number of features.

Similar to the rest of the data sets, we only used a portion of the total set, with the maximum number of samples being 260 and the minimum being 20. A step was 10 samples.

Changing dataset size. We see a similar situation on Figure 10 in which all of the methods again report a high median absolute error. Cross-validation & cross-testing and cross-validation & testing report similar results and are stable through the test. However what is different in the graph is that nested cross-validation also reports stable results through the test.

In Figure 10, nested cross-validation is doing much worse then either of the other two methods by reporting a larger error.

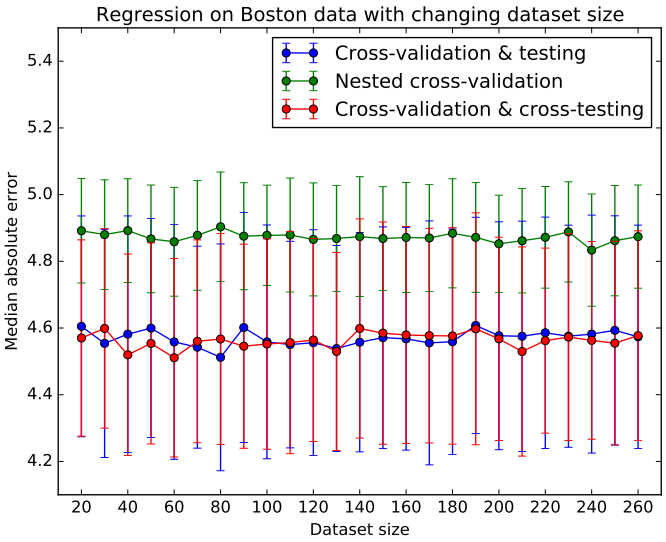

Figure 10: Median absolute error of LASSO implementation on Boston dataset plotted against changing dataset size.

Changing test set size. On Figure 11 we see that the median absolute error being reported by the two data partitioning schemes is unusually large, meaning that the model will not be useful in practice. We can also see that both cross-validation & testing and cross-validation & cross-testing report similar median absolute errors and stable scores with all the different test set sizes.

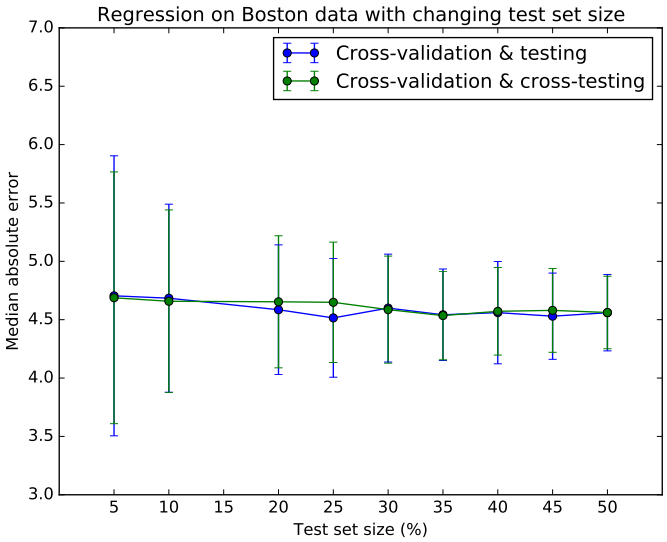

Figure 11: Median absolute error of LASSO implementation on Boston dataset plotted against a changing test set size.

5 Discussion

In this section we will take a look at the implementation of the cross-testing function and the results that the three data partitioning schemes produced when applied to the four datasets. We will also bring out the limitations, open problems and opportunities for future research.

5.1 Thesis results

For this thesis we have two separate result types. The first result was the successful implementation and publishing of a cross-validation & cross-testing function. In order to allow users to have more freedom when using the data partitioning method, we decided to only implement a function for cross-testing as there are already a large amount of different cross-validation functions in scikit-learn. A more detailed explanation of the implementation and the documentation for the function itself can be found above at section 4.1.4.

The other results are from applying the data partitioning schemes to the four datasets. Since dataset sizes are quite small, the maximum size being 260 samples, it is possible that in some cases the partitioning schemes did not have that much

of a difference in the accuracy score or median absolute error that they reported. This is the case for Figures 7, 9 and 8.

On Figures 5 and 7 we see a curve that was shown by Korjus et al. [1]. We can see on Figure 5 that the data partitioning schemes reported different accuracy scores and after a steep drop between the sizes of 20 and 40 samples, started a general trend up. Here we can also see that the standard deviation of nested cross-validation decreases as the dataset size increased. Meaning that we started getting more accurate results averaged over 200 iterations. What is not known is why nested cross-validation did not perform the best, as was shown by Korjus et al. [1]. As we can see on Figure 7, all three partitioning schemes performed very well and very close to each other. The standard deviation for all of them also greatly decreased as the dataset size increased. When comparing it to Figure 5, it is possible that the difference comes from the Spike data being more noisy due to its 61 features, than the Iris dataset, which has only 4 features. The Spike dataset is potentially more noisy because of the big amount of features that it has, most of which might not be important in predicting.

On Figures 4 and 6, we see that the accuracy score decreases while the test set size relative to the dataset size increases. Note that the dataset size for these tests, and for similar tests with regression, was fixed at 100 samples. This is because as you use a higher percentage of the total dataset for testing, you will have less data to train the model with. The accuracy scores in Figure 6 stay at a relatively high and stable level. This could be due to the amount of features that each dataset has. The Spike dataset has 61 features, all of which might not be even important. The Iris dataset however has 4 features and even if all of them are not very important for the model, it is not as noisy as the Spike dataset.

We can observe that for both Figures 4 cross-validation & cross-testing was the best, while in Figure 6 the two data partitioning schemes performed quite similarly to one another. The reason for this is currently unknown. The results in Figure 4 is as is expected from the results of Korjus et al. [1].

On Figure 8 we see that the partitioning schemes have reported very similar and quite low median absolute errors throughout the test. While the median absolute error changes very little with the change in dataset size, was does change is the standard deviation for the partitioning schemes. This shows that they have all moved closer to their actual values.

Figure 10 shows us that a graph where not only is nested cross-validation the worst performer, but cross-validation & testing and cross-validation & cross-testing perform very similarly to one another. We can also see that throughout the test the median absolute error remains very stable for the three schemes. However nested cross-validation does seem to have the lowest standard deviation. We are unable to explain this figure at this time.

On Figures 9 and 11 we again have graphs that are quite similar to one another, although the median absolute error in Figure 11 is about 16 times higher than that of Figure 9. We can see that for both figures the standard deviation of all partitioning schemes decreases. We are currently unable to explain why the figures that plot the median absolute error against test set size report such stable results.

5.2 Limitations

Due to limited time, resources and computing power, numerous compromises had to be made during the writing of this thesis. None of them break anything, but they would of been good tools to further analyse the different partitioning schemes.

Firstly it would of been good to run tests for more than 200 iterations. This would of converged the results more closely to their actual results. However 200 iterations was a good compromise between the computing power and the limited amount of time that we had at our disposal.

Secondly comprehensive tests were not written for the cross-testing function in the Github repository. This was done mainly due to time restraints and since the objective was to get it to the "Related Projects" section, it would not have been deemed necessary by scikit-learn.

Thirdly it would of been excellent to gather data on what hyper-parameters were chosen in which partitioning scheme with what machine learning method. It would of been interesting to see in what cases some hyper-parameter values were chosen over others and how they performed.

When analysing the hyper-parameters, it would of been helpful to then iterate through many more hyper-parameters and their different values. That way there would of been a much bigger difference between the different models created during the grid-search and training processes.

The last two limitations would of required a large amount of data collecting and analysing in order to not only graph the data points, but to also interpret them.

We were unable to properly explain the results of Figures 9, 10 and 11.

5.3 Future work

The behaivior of nested cross-validation in Figures 9, 10 and 11. is up for further study. The results are especially interesting, because for both regression tasks, nested cross-validation had the worse performance and the two other partitioning schemes reported very similar results. The paper by Korjus et al. [1] says that this should not be the case, with nested cross-validation performing the best, followed

by cross-validation & cross-testing and the worst should be cross-validation & testing.

In the future correct permutation tests must also be ran in order to get the proportion of significant results. This was left undone in this thesis due to being computationally extremely expensive. Due to my weaker computer, it was not possible to get the necessary results in a reasonable amount of time. It is also possible that the strange results are not statistically significant.

The short-term goal is to get this module into the "Related projects" page on the scikit-learn webpage. This would mean that the module and in turn cross-validation & cross-testing would be quite visible to people and will also be easy to use for machine learning purposes.

The long-term goal is to have the cross-testing function accepted into the scikit-learn core library. However this requires it to be a few years old, have at least 200 citations and be widely used. These requirements are in place in order to help the core developers to keep the scikit-learn code at a high standard.

6 Conclusion

In this thesis we have given an overview on the three different data partitioning schemes, scikit-learn and what requirements there are to getting your algorithm and contribution accepted into their core library. We have also implemented the three partitioning schemes on both classification and regression tasks. The datasets for classification were Spikes and Iris, while for regression Simulated and Boston were used.

Cross-validation & testing and cross-validation & cross-testing usually performed as was expected, however from time to time nested cross-validation performed worse than was expected. While for small datasets, sometimes the different partitioning schemes give out very similar results, it is unusual that nested cross-validation would be much worse than the other two.

The main goal of this thesis was to create a function or set of functions to use cross-validation & cross-testing easily in scikit-learn. This has been successful with the fully documented and public cross-testing function on Github.

In this thesis we also reproduced the results that Korjus et al [1] showed for the Spike dataset and produced similar results with the Iris dataset. We also found out that the partitioning schemes produce different results for regression tasks. This is something that must be looked into more, especially for the new partitioning scheme, cross-validation & cross-testing.

References

[1] Korjus K, Hebart MN, Vicente R. An Efficient Data Partitioning to Improve Classification Performance While Keeping Parameters Interpretable. PLOS ONE. 2016;11:e0161788.

[2] Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, et al. API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning; 2013. p. 108–122.

[3] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. vol. 12. Microtome Publishing; 2011. p. 2825–2830.

[4] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA data mining software: an update. vol. 11; 2009. p. 10–18.

[5] Demšar J, Zupan B, Leban G, Curk T. Orange: From Experimental Machine Learning to Interactive Data Mining. In: Lecture Notes in Computer Science. Springer Nature; 2004. p. 537–539.

[6] Sculley D. Large scale learning to rank. In: NIPS Workshop on Advances in Ranking; 2009. p. 58–63.

[7] Teeters JL, Harris KD, Millman KJ, Olshausen BA, Sommer FT. Data sharing for computational neuroscience. vol. 6. Springer; 2008. p. 47–55.

[8] Fisher RA. The Use of Multiple Measurements in Taxonomic Problems. vol. 7. Wiley-Blackwell; 1936. p. 179–188.

[9] Harrison D, Rubinfeld DL. Hedonic housing prices and the demand for clean air. vol. 5. Elsevier; 1978. p. 81–102.

[10] Bennett KP, Bredensteiner EJ. Duality and geometry in SVM classifiers. In: ICML; 2000. p. 57–64.

[11] 1.4. Support Vector Machines;. (04.05.2017). `http://scikit-learn.org/stable/modules/svm.html`.

[12] sklearn.model_selection.GridSearchCV;. (11.05.2017). `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`.

[13] Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI. vol. 14. Stanford, CA; 1995. p. 1137–1145.

[14] Franklin J. The elements of statistical learning: data mining, inference and prediction. vol. 27. Springer Nature; 2005. p. 83–85.

[15] Style Guide for Python Code;. (20.04.2017). `https://www.python.org/dev/peps/pep-0008`.

Non-exclusive licence to reproduce thesis and make thesis public

I, Rain Vagel (date of birth: 7th of May 1995),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Developing a scikit-learn module for a novel data partition for machine learning

supervised by Raul Vicente and Kristjan Korjus

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu 11.05.2017