The Journal of Open Source Software

LFSpy: A Python Implementation of Local Feature Selection for Data Classification with scikit-learn Compatibility

Kiret Dhindsa[1, 2, 3], **Oliver Cook**[1], **Thomas Mudway**[1], **Areeb Khawaja**[1], **Ron Harwood**[1], and **Ranil Sonnadara**[1, 2, 3]

1 Research and High Performance Computing, McMaster University **2** Vector Institute **3** Department of Surgery, McMaster University

DOI: 10.21105/joss.01958

Software

- Review
- Repository
- Archive

Editor: Dan Foreman-Mackey

Reviewers:

- @effigies
- @sauln

Submitted: 12 December 2019
Published: 10 May 2020

Background

Successful machine learning depends on inputting features, or variables, that provide information that is useful for solving the problem at hand. Supervised classification, where the goal is to label or categorize new data based on patterns identified in labeled training data, is the most commonly applied task in machine learning. For these problems, the features used to train a machine learning model must help discriminate between different categories of data samples. However, it is not always possible to know a priori which of the available features are informative, and which are not. The presence of uninformative features can contribute noise and reduce the robustness and performance of classification models. Therefore, an important step in machine learning is the selection of informative features and the omission of uninformative features.

Summary

Where typical feature selection methods find an optimal feature subset that is applied globally to all data samples, Local Feature Selection (LFS) finds optimal feature subsets for each local region of a data space. In this way, LFS is better able to adapt to regional variability and non-stationarity in a sample space. In addition, the method is paired with a simple classifier based on class similarity which can account for the fact that different samples may be modeled using different feature subsets.

Local feature selection is performed by promoting class-wise clustering in the neighbourhood around each point, i.e., by finding the subset of available features that minimizes the average distance between points belonging to the same class, while maximizing the distances between classes. Thus, a feature subspace is identified that maximizes classifiability locally around each point. However, since this feature space can be different for each local region, standard classifiers cannot be readily applied. Instead, a notion of similarity between samples is introduced that intuitively lends itself to classification. Since LFS defines a local region for each sample, the regions are overlapping. Therefore, each point is represented in a number of feature spaces. Therefore, a class label can be assigned by accumulating the class labels of the nearest neighbours to a sample in each of these feature spaces. Full details of LFS, including experimental results demonstrating its effectiveness compared to other feature selection and classification methods on several datasets, are given in (Armanfard, Reilly, & Komeili, 2015) and (Armanfard, Reilly, & Komeili, 2017).

LFSpy was designed to be used for researchers working on any supervised learning problem, but is especially powerful for data that are non-stationary, non-ergodic, or that otherwise do not cluster well into classes. A prominant example of data with these properties is electroencephalography (EEG) time-series. In this field, LFS has been used to continuously detect characteristic brain responses to auditory stimuli in coma patients (Armanfard, Komeili, Reilly, & Connoly, 2018), and is being tested for detection of traumatic brain injury (Boshra et al., 2019).

Usage

LFSpy is a Python implementation of LFS that follows the `scikit-learn` (Pedregosa et al., 2011) class structure, and can therefore be used as part of a `scikit-learn` Pipeline like other classifiers. Open-source code, full documentation, and a demo using sample data are available at https://github.com/McMasterRS/LFSpy. Using LFS to train a model and test on new data is made simple with LFSpy and can be done in just a few lines of code. Usage follows the standard format used for `scikit-learn` classifiers. First, an LFS object is created to hold the model configuration, parameters, and once trained, the trained model itself. The implementation is flexible in that it gives the user control over a number of optional parameters, including for example, the size of the local region used for feature selection. The following training and testing functions can then be called from that object:

- `lfs.fit`: trains an LFS model given training data and corresponding training labels
- `lfs.predict`: for a trained model, outputs class label predictions given testing data
- `lfs.score`: outputs the classification error for the testing data in total, and by class, given testing data and ground truth testing labels

Given training and testing data that are compatible with `scikit-learn` models, a typical example of model training and testing is as follows:

```
from LFSpy import LocalFeatureSelection
lfs = LocalFeatureSelection(alpha=19,
                            gamma=0.2,
                            tau=2,
                            sigma=1,
                            n_beta=20,
                            nrrp=2000,
                            knn=1)
lfs.fit(training_data, training_labels)
predicted_labels = lfs.predict(testing_data)
total_error, class_error = lfs.score(testing_data, testing_labels)
```

LFSpy is also fully compatible with the `scikit-learn` Pipeline method:

```
from LFSpy import LocalFeatureSelection
from sklearn.pipeline import Pipeline
lfs = LocalFeatureSelection(alpha=19,
                            gamma=0.2,
                            tau=2,
                            sigma=1,
                            n_beta=20,
                            nrrp=2000,
                            knn=1)
```

```
pipeline = Pipeline([('lfs', lfs)])
pipeline.fit(training_data, training_labels)
predicted_labels = pipeline.predict(testing_data)
total_error, class_error = pipeline.score(testing_data, testing_labels)
```

The dependencies for `LFSpy` are as follows:

- Python 3
- NumPy>=1.14
- SciPy>=1.1
- Scikit-learn>=0.18.2

Comparison to Other Classifiers

A comparison of classification accuracies obtained with LFS and two standard `scikit-learn` pipelines are shown below. The Random Forest classifier (RFC) and a linear Support Vector Machine (SVM) with univariate feature selection using the F-statistic are used for comparison. For all tests, we use default settings. For consistency, none of the methods are provided with a priori information about the number of informative features to select. Both LFS and RFC choose the number of appropriate features internally. The SVM must be given a number of features to choose, so we set the number of features to 25% of the total number of available features for this example.

Results are obtained with two sample datasets that are representative of the intended use case of LFS. The first is a sample dataset used to illustrate the utility of LFS. This dataset is synthetically generated with 100 training samples and 108 test samples. The number of informative vs. uninformative features in this dataset are unknown. The second dataset is the Iris dataset included with `scikit-learn`, which contains 100 samples and four features (50 are used for training, and 50 are used for testing; all four features are informative). To illustrate the value of LFS, we show the classification accuracy of each method after appending increasing numbers of up to 1000 non-informative Guassian features. Each feature was randomly generated with zero mean and a standard deviation between 0 and 3, sampled from a uniform distribution.

It can be seen that with both datasets LFS outperforms the other two methods, particularly when the number of non-informative features becomes large. LFS remains relatively stable in classification performance, whereas RFC and SVM experience significant degradation as the number of non-informative features grows well past 100.

Classification accuracies with the sample synthetic dataset

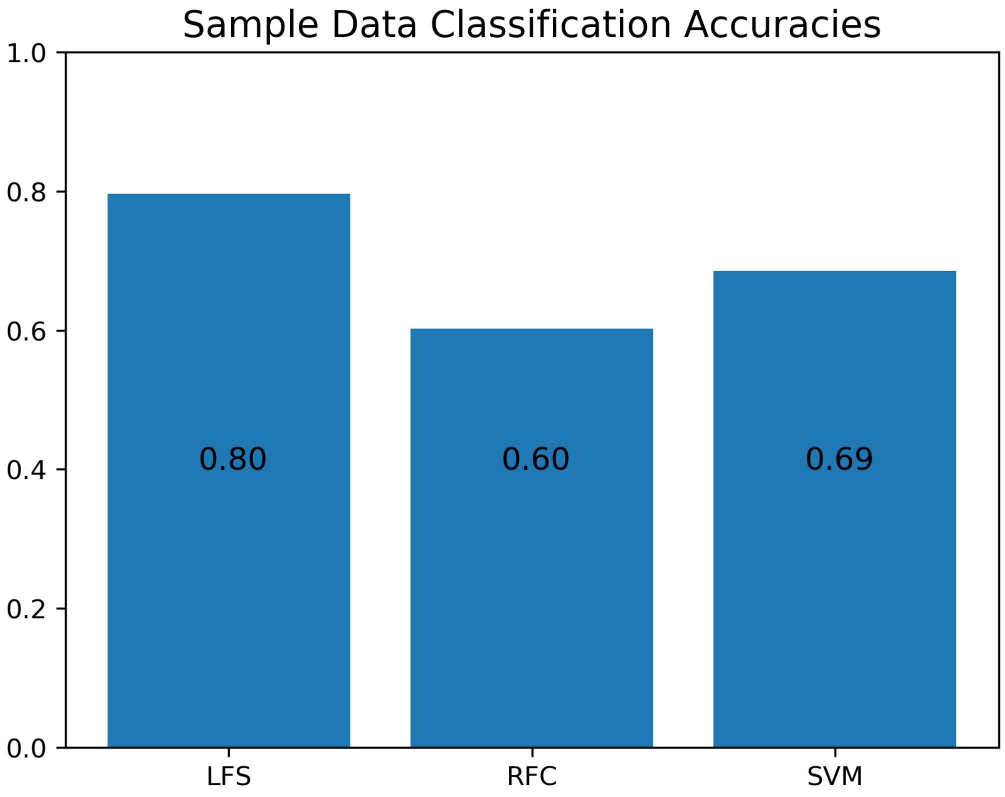

Figure 1: Comparison of classification accuracies obtained with different classifiers using the sample synthetic dataset available with the LFSpy package.

Classification accuracies with the Iris dataset

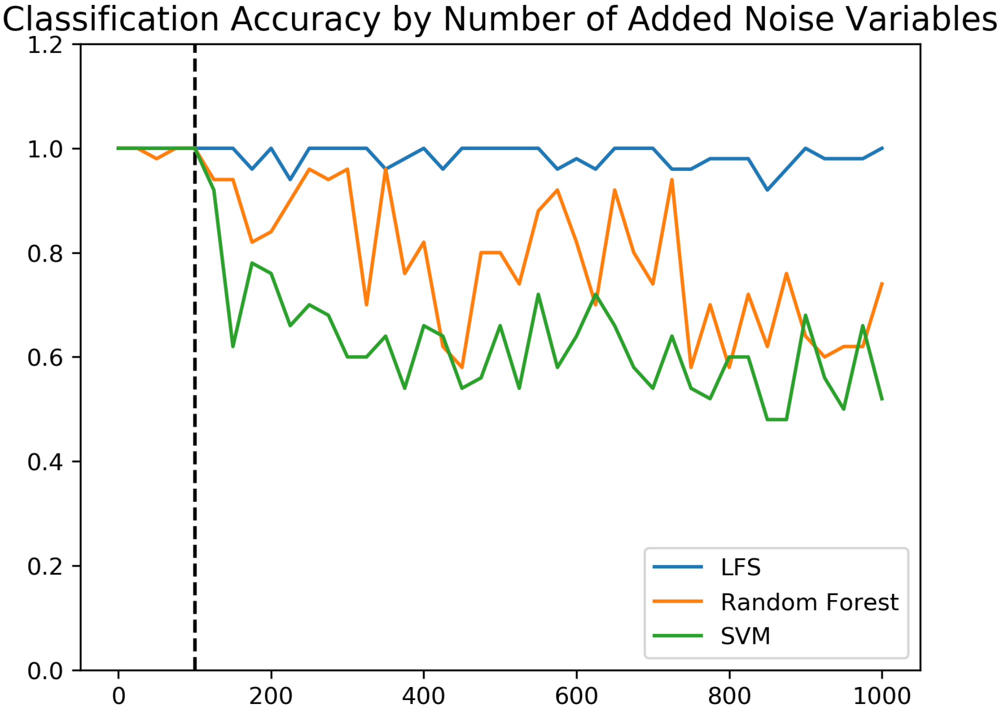

Figure 2: Comparison of classification accuracies obtained with different classifiers and increasing numbers of non-informative features using the Fisher Iris dataset available in `scikit-learn`.

Comparison using data generated with `scikit-learn`

For this comparison we use the same classifiers with the same configurations as described above, but instead make use of `make_classification` function in `scikit-learn`. This function provides us with more control so we can generate a dataset with properties that illustrate in more detail where `LFSpy` is particularly useful. In this example, we generate synthetic datasets to simulate a two-class classification problem with 50 samples (40 used for training, and 10 used for testing). In total, 12 datasets are created with varying degrees of complexity introduced through different properties. First, each class can be distributed over one, two, or three clusters (i.e., if each class is represented by three clusters, then it is made up of three distinct distributions with different statistics). Four datasets are created within each of these cases: 1) a simple problem with 5 informative features and zero non-informative features, 2) five informative features with 40 redundant features, (`r=40`) made up as combinations of the five informative features, 3) five informative features with 40 repeated features (`s=40`) made up as copies of the five redundant features, and 4) five informative features with 20 redundant features and 20 repeated features (`r=20, s=20`). All datasets were generated with 5% noisy labels (i.e., for 5% of the samples, the true class was assigned randomly), and a class separability of 1.0.

The results of this experiment, shown below, demonstrate how `LFSpy` can have an advantage in increasingly complex classification problems with large numbers of non-informative features of different kinds. In particular, its strategy of breaking up a classification problem into many small local problems instead of attempting to find a global solution allows LFS to perform well when each class is represented by multiple clusters with different statistics.

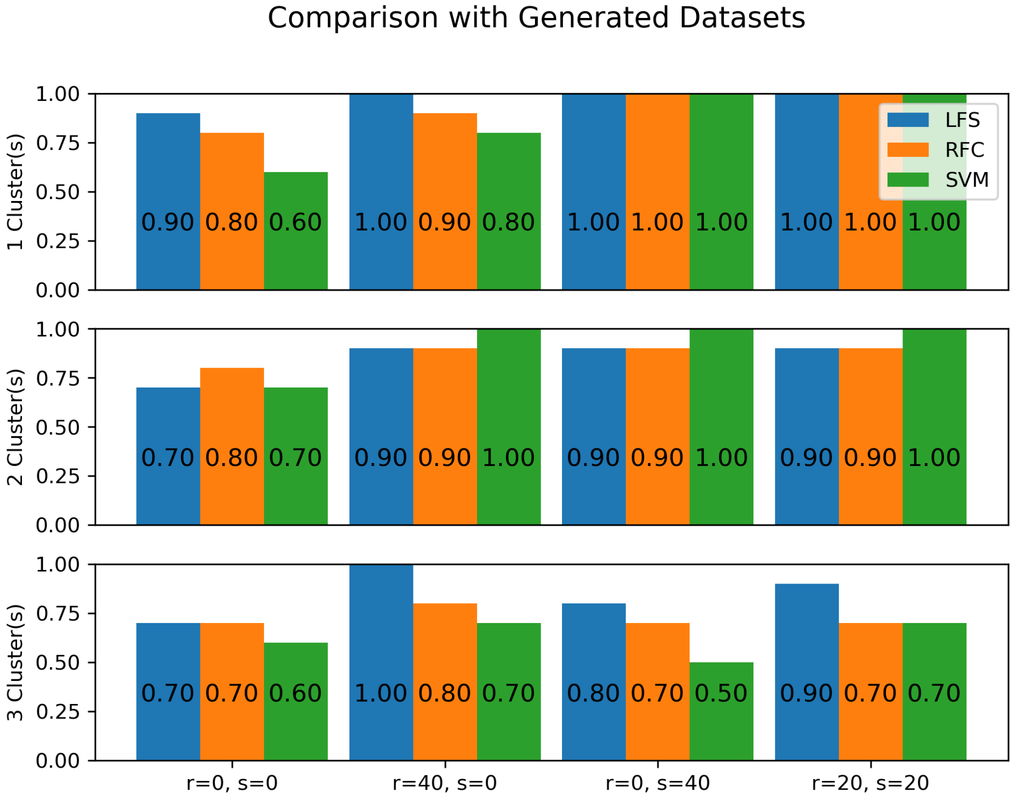

Figure 3: Comparison of classification accuracies obtained with different classifiers and different kinds of complexities using data generated with `scikit-learn.datasets.make_classification`.

Acknowledgments

Funding for this project was obtained through the CANARIE Research Software Program Local Support Initiative.

References

Armanfard, N., Komeili, M., Reilly, J. P., & Connoly, J. (2018). A machine learning framework for automatic and continuous MMN detection with preliminary results for coma outcome prediction. *IEEE journal of biomedical and health informatics*. doi:10.1109/JBHI.2018.2877738

Armanfard, N., Reilly, J. P., & Komeili, M. (2015). Local feature selection for data classification. *IEEE transactions on pattern analysis and machine intelligence*, *38*(6), 1217–1227. doi:10.1109/TPAMI.2015.2478471

Armanfard, N., Reilly, J. P., & Komeili, M. (2017). Logistic localized modeling of the sample space for feature selection and classification. *IEEE transactions on neural networks and learning systems*, *29*(5), 1396–1413. doi:10.1109/TNNLS.2017.2676101

Boshra, R., Dhindsa, K., Boursalie, O., Ruiter, K. I., Sonnadara, R., Samavi, R., Doyle, T. E., et al. (2019). From group-level statistics to single-subject prediction: Machine learning detection of concussion in retired athletes. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *27*(7), 1492–1501. doi:10.1109/TNSRE.2019.2922553

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.