LFSpy

May 07, 2020

Contents:

Localized feature selection (LFS) is a supervised machine learning approach for embedding localized feature selection in classification. The sample space is partitioned into overlapping regions, and subsets of features are selected that are optimal for classification within each local region. As the size and membership of the feature subsets can vary across regions, LFS is able to adapt to local variation across the entire sample space.

This repository contains a python implementation of this method that is compatible with scikit-learn pipelines. For a Matlab version, refer to https://github.com/armanfn/LFS

The LFS approach was developed by Nargus Armanfard. For further information please refer to:

- N. Armanfard, JP. Reilly, and M. Komeili, "Local Feature Selection for Data Classification", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 6, pp. 1217-1227, 2016.
- N. Armanfard, JP. Reilly, and M. Komeili, "Logistic Localized Modeling of the Sample Space for Feature Selection and Classification", IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1396-1413, 2018

CHAPTER 1

Statement of Need

LFSpy offers an implementation of the Local Feature Selection (LFS) algorithm that is compatible with scikit-learn, one of the most widely used machine learning packages today. LFS combines classification with feature selection, and distinguishes itself by its flexibility in selecting a different subset of features for different data points based on what is most discriminative in local regions of the feature space. This means LFS overcomes a well-known weakness of many classification algorithms, i.e., classification for non-stationary data where the number of features is high relative to the number of samples.

CHAPTER 2

Installation

LFSpy is available on the pypy distribution platform at https://pypi.org/project/LFSpy/.

To install LFSpy along with its dependacies run the command:

```
pip install lfspy
```

CHAPTER 3

Dependencies

LFS requires:

- Python 3
- NumPy>=1.14
- SciPy>=1.1
- Scikit-learn>=0.18.2
- pytest>=5.0.0

CHAPTER 4

Testing

We recommend running the provided test after installing LFSpy to ensure the results obtained match expected outputs.

pytest may be installed either directly through pip (`pip install pytest`) or using the test extra (`pip install LFSpy[test]`).

```
pytest --pyargs LFSpy
```

This will output to console whether or not the results of LFSpy on two datasets (the sample dataset provided in this repository, and scikit-learn's Fisher Iris dataset) are exactly as expected.

So far, LFSpy has been tested on Windows 10 with and without Conda, and on Ubuntu. In all cases, results have been exactly the expected results.

CHAPTER 5

Usage

To use LFSpy on its own:

```
from LFSpy import LocalFeatureSelection

lfs = LocalFeatureSelection()
lfs.fit(training_data, training_labels)
predicted_labels = lfs.predict(testing_data)
total_error, class_error = lfs.score(testing_data, testing_labels)
```

To use LFSpy as part of an sklearn pipeline:

```
from LFS import LocalFeatureSelection
from sklearn.pipeline import Pipeline

lfs = LocalFeatureSelection()
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)
```

CHAPTER 6

Tunable Parameters

- alpha: (default: 19) the maximum number of selected features for each representative point
- gamma: (default: 0.2) impurity level tolerance, controls proportion of out-of-class samples can be in local region
- tau: (default: 2) number of passes through the training set
- sigma: (default: 1) adjusts weightings for observations based on their distance, values greater than 1 result in lower weighting
- n_beta: (default: 20) number of beta values to test, controls the relative weighting of intra-class vs. inter-class distance in the objective function
- nrrp: (default: 2000) number of iterations for randomized rounding process
- knn: (default: 1) number of nearest neighbours to compare for classification

CHAPTER 7

Authors

- Oliver Cook
- Kiret Dhindsa
- Areeb Khawajaby
- Ron Harwood
- Thomas Mudway

CHAPTER 8

Acknowledgments

- N. Armanfard, JP. Reilly, and M. Komeili, “Local Feature Selection for Data Classification”, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 6, pp. 1217-1227, 2016.
- N. Armanfard, JP. Reilly, and M. Komeili, “Logistic Localized Modeling of the Sample Space for Feature Selection and Classification”, IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1396-1413, 2018.

8.1 Introduction

Localized feature selection (LFS) is a supervised machine learning approach for embedding localized feature selection in classification. The sample space is partitioned into overlapping regions, and subsets of features are selected that are optimal for classification within each local region. As the size and membership of the feature subsets can vary across regions, LFS is able to adapt to local variation across the entire sample space.

This repository contains a python implementation of this method that is compatible with scikit-learn pipelines. For a Matlab version, refer to https://github.com/armanfn/LFS

The LFS approach was developed by Nargus Armanfard. For further information please refer to:

- N. Armanfard, JP. Reilly, and M. Komeili, “Local Feature Selection for Data Classification”, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 6, pp. 1217-1227, 2016.
- N. Armanfard, JP. Reilly, and M. Komeili, “Logistic Localized Modeling of the Sample Space for Feature Selection and Classification”, IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1396-1413, 2018

8.1.1 Statement of Need

LFSpy offers an implementation of the Local Feature Selection (LFS) algorithm that is compatible with scikit-learn, one of the most widely used machine learning packages today. LFS combines classification with feature selection, and distinguishes itself by its flexibility in selecting a different subset of features for different data points based on what is most discriminative in local regions of the feature space. This means LFS overcomes a well-known weakness of many

classification algorithms, i.e., classification for non-stationary data where the number of features is high relative to the number of samples.

8.2 Installation

LFSpy is available on the Python Package Index (PyPI).

To install LFSpy along with its dependencies run the command:

```
pip install lfspy
```

8.2.1 Dependencies

LFS requires:

- Python 3
- NumPy>=1.14
- SciPy>=1.1
- Scikit-learn>=0.18.2
- pytest>=5.0.0

8.3 Configuration

The localized feature selection method has a set of user configurable paramaters that can be tweaked to get your desired functionality. For a full description of each parameter refer to the papers listed in the citations section. The parameters are:

- alpha: (default: 19) the maximum number of selected features for each representative point
- gamma: (default: 0.2) impurity level tolerance, controls proportion of out-of-class samples can be in local region
- tau: (default: 2) number of passes through the training set
- sigma: (default: 1) adjusts weightings for observations based on their distance, values greater than 1 result in lower weighting
- n_beta: (default: 20) number of beta values to test, controls the relative weighting of intra-class vs. inter-class distance in the objective function
- nrrp: (default: 2000) number of iterations for randomized rounding process
- knn: (default: 1) number of nearest neighbours to compare for classification

8.4 Usage

To use LFSpy on its own:

```
from LFSpy import LocalFeatureSelection

lfs = LocalFeatureSelection()
lfs.fit(training_data, training_labels)
predicted_labels = lfs.predict(testing_data)
total_error, class_error = lfs.score(testing_data, testing_labels)
```

8.5 scikit-learn Compatability

To use LFSpy as part of an sklearn pipeline:

```
from LFS import LocalFeatureSelection
from sklearn.pipeline import Pipeline

lfs = LocalFeatureSelection()
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)
```

8.6 Testing

We recommend running the provided test after installing LFSpy to ensure the results obtained match expected outputs.

pytest may be installed either directly through pip (pip install pytest) or using the test extra (pip install LFSpy[test]).

```
pytest --pyargs LFSpy
```

This will output to console whether or not the results of LFSpy on two datasets (the sample dataset provided in this repository, and scikit-learn's Fisher Iris dataset) are exactly as expected.

So far, LFSpy has been tested on Windows 10 with and without Conda, and on Ubuntu. In all cases, results have been exactly the expected results.

8.7 Functionality

```
class LocalFeatureSelection(self, alpha=19, gamma=0.2, tau=2, sigma=1, n_beta=20,␣
↪nrrp=2000, knn=1, rr_seed=None)
```

Parameters	**alpha : integer, optional, default 19** maximum number of selected features for each representative sample **gamma : integer, optional, default 0.2** impurity level **tau : integer, optional, default 2** number of iterations **sigma : integer, optional, default 1** controls neighboring samples weighting **n_beta : integer, optional, default 20** number of distinct beta **nrrp : integer, optional, default 2000** number of iterations for randomized wandering process **knn : integer, optional, default 1** k nearest neighbours **rr_seed : integer, optional, default None** seed value for random wandering process
Attributes	**fstar : array of shape (n_features, n_features)** selected features for each sample **fstar_lin : array of shape (n_features, n_features)** fstar before applying randomized wandering process **training_data : array of shape (n_features, n_samples** The set of M by N features and observations the model was trained on **training_labels : array of shape (n_samples)** The set of N class labels the model was trained on

8.7.1 Methods

fit(self, training_data, training_labels)	
predict(self, testing_data)	
classification(self, testing_data)	
class_sim_m(self, test, N, patterns, targets, fstar)	

```
__init__(self, alpha=19, gamma=0.2, tau=2, sigma=1, n_beta=20, nrrp=2000, knn=1, rr_
↪seed=None)
```

Initialize self

```
fit(self, training_data, training_labels)
```

Fit model

Parameters	**training_data** [{array-like} of shape (n_samples, m_features)] Training data **training_labels** [{array-like} of shape (n_samples)] Class labels for each sample
Returns	

```
predict(self, testing_data)
```

Predict using the model

Parameters	**testing_data** [{array-like} of shape (n_samples, m_features)] Testing data
Returns	

```
classification(self, testing_data)
```

Internal feature classification function, called by predict function

Parameters	**testing_data** [{array-like} of shape (n_samples, m_features)] Testing data
Returns	

```
class_sim_m(self, test, N, patterns, targets, fstar, gamma, knn)
```

Internal feature classification function, called by classification function

Parameters	**test** [{array-like} of shape (n_samples, m_features)] Testing data **N: {integer}** Number of features **patterns:** Data the model was trained on **targets:** Class Labels the model was trained on **fstar:** Selected features for each samples **gamma:** Impurity Level **knn:** K nearest neighbours
Returns	

8.8 Examples

Given here is an example demonstration of localized feature selection and LFSpy for feature selection and classification using the common Iris flower data set.

For installation instructions please refer to the "Installation" section.

```
import numpy as np
from scipy.io import loadmat
from LFSpy import LocalFeatureSelection
from sklearn.pipeline import Pipeline

# Loads the sample dataset
mat = loadmat('LFSpy/tests/matlab_Data')
x_train = mat['Train'].T
y_train = mat['TrainLables'][0]
x_test = mat['Test'].T
y_test = mat['TestLables'][0]


#Trains an tests and LFS model using default parameters on the given dataset.
print('Training and testing an LFS model with default parameters.\nThis may take a␣
↪few minutes...')
lfs = LocalFeatureSelection(rr_seed=777)
pipeline = Pipeline([('classifier', lfs)])
pipeline.fit(x_train, y_train)
y_pred = pipeline.predict(x_test)
score = pipeline.score(x_test, y_test)
print('LFS test accuracy: {}'.format(score))
# On our test system, running this code prints the following: LFS test accuracy: 0.
↪7962962962962963
```

8.9 Contribution Guidelines

Contributions are welcomed and can be made to the public Git repository available at: https://github.com/McMasterRS/LFSpy

We encourage anyone looking to contribute to consult the open issues available at https://github.com/McMasterRS/LFSpy/issues

We ask that in submitting changes you consult the coding standards and pull request guidelines outlined below.

8.9.1 Contributing to the method:

This library impliments the Localized Feature Selection method outlined by Nargus Armenford. As such, changes made the method should be only done to reflect changes made to the theoretical basis.

8.9.2 Submitting a Pull Request

Please submit one pull request per feature. Before submitting a pull request ensure your code continues to pass the included tests. LFSpy uses pytest and the tests are located in the tests directory of this repository.

The tests can be run using the command:

```
pytest --pyargs LFSpy
```

8.10 Citations

1. N. Armanfard, JP. Reilly, and M. Komeili, “Local Feature Selection for Data Classification”, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 6, pp. 1217-1227, 2016.

2. N. Armanfard, JP. Reilly, and M. Komeili, “Logistic Localized Modeling of the Sample Space for Feature Selection and Classification”, IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1396-1413, 2018.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search