Feature selection package with IHT

Konstantin Pakulev [1] **Viacheslav Pronin** [1]

Abstract

Sparse feature selection takes an important place in numerous signal processing applications. There are many feature selection methods available, however, all of them are implemented in different libraries and lack a unified API. This makes it hard to compare the performance of different methods. In this work, we present an easily extendable framework for comparison of different feature selection methods. We also provide implementations of of two methods: permutation importance and Iterative Hard Thresholding.

1. Introduction

Selecting an optimal subset of sparse features for a model is of a great importance for many signal processing applications as source separation (Davies & Mitianoudis, 2004) or de-noising (Févotte & Godsill, 2006). Likewise, it plays a critical role in many data mining applications that handle high-dimensional data.

The problem of feature selection can be formulated as follows. Let $A \in R^{n \times p}$ be a design matrix, $y \in R^n$ be a set of observations, $x \in R^p$ be a set of parameters and $\epsilon \in R^n$ be some noise. Assuming that observation are generated via $y = Ax + \epsilon$ we want to find such $\hat{x} \in R^p$ that satisfies optimization problem 1 where s_1 is some non-zero value.

$$\underset{\hat{x}}{minimize} \quad \frac{1}{2}||A\hat{x} - y||_2^2 \tag{1}$$

where x is:

$$subject\ to \quad \sum_{j=1}^{p} I(|\hat{x}_j| \neq 0) \leq s_1 \tag{2}$$

Sometimes it also can be beneficial to select feature groups along with features (Xiang et al., 2014), i.e. when the data has some grouping structures. In such cases the optimization problem is complemented by an additional constraint 3 where the entities of x are separated into $|G|$ mutually exclusive groups, G_j denotes indices that belong to the j-th group and s_2 is some non-zero value.

$$subject\ to \quad \sum_{j=1}^{|G|} I(||\hat{x}_{G_j}||_2 \neq 0) \leq s_2 \tag{3}$$

Solving the optimization 1 with constraints 2 and/or 3 is a combinatorial problem for which no efficient algorithm exists for the general case. However, with certain algorithms a sub-optimal solution can be calculated in a reasonable time (Blumensath & Davies, 2010; Xiang et al., 2014; Friedman et al., 2010). Those methods can be separated into two categories. Methods form the first one attempt to find continuous surrogates for the discrete functions and re-formulating the problem in a form of convex or non-convex optimization (Friedman et al., 2010). Solving the problem in form of the convex optimization usually results in a sub-optimal solution, however non-convex methods require much more computational resources. Methods from the second category search for the solution via greedy algorithm to avoid tackling the discrete optimization (Blumensath & Davies, 2010; Xiang et al., 2014). In our work, we focus only on the methods based on greedy algorithms as they are comparatively fast and are suitable for large scale problems.

With the large number of methods being presented there hasn't been a framework or a benchmark that would've allowed to evaluate all of them under the same settings. Our project aims to mitigate this issue by presenting a framework for evaluating different feature selection methods. This problem is of utmost importance as a fair comparison that compares the methods under different settings is a must for selecting the method that suits the production needs the best.

The main contributions of this report are as follows:

- We present an easily scalable framework for evaluating various feature selection methos
- We provide the utilities for generating a synthetic dataset that can be used for evaluation of different methods

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Konstantin Pakulev <Konstantin.Pakulev@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

- We provide a baseline permutation importance (Breiman, 2001) method (using implementation from sklearn)
- We provide the implementations of Normalised Iterative Hard Thresholding (Blumensath & Davies, 2010)
- We present evaluation results on the synthetic dataset obtained with our pipeline for permutation importance method and Normalised Iterative Hard Thresholding

The code of the project is available here: https://github.com/adasegroup/FES-feature-selector

2. Related work

Normalised Iterative Hard Thresholding (Blumensath & Davies, 2010) is a method that uses a greedy strategy to search for the sub-optimal solution. In particular, the algorithm searches for $\hat{x}^{n+1}$ starting from $\hat{x}^0 = 0$ by applying the iterative procedure 4 where $H_K()$ is a non-linear operator that sets all but the top-k largest elements by their magnitude to zero and μ is computed adaptively as described in original work.

$$\hat{x}^{n+1} = H_K(\hat{x}^n + \mu A^\top (y - A\hat{x}^n)) \tag{4}$$

A huge advantage of the algorithm is that it gives theoretical guarantees on convergence to a local minimum of the cost function as well as it doesn't depend on the scaling of the design matrix A. The drawback is that those guarantees on convergence exist only if A satisfies restricted isometry property. Also, during our experiments we've found that the algorithm is very sensitive to noise of the observations (see section 4) and requires regularization to not overfit to noised data.

Simultaneous Feature and Feature Group Selection through Hard Thresholding (Xiang et al., 2014) is another algorithm that uses the same iterative scheme as IHT. But contrary to IHT it admits a globally optimal solution using dynamic programming. The algorithm employs the iterative procedure 5 where f is the objective loss function, L is found by line search and $SGHT()$ stands for Sparse Group Hard Thresholding that is solved by dynamic programming as described in (Xiang et al., 2014).

$$\hat{x}^{n+1} = SGHT(\hat{x}^n - \frac{1}{L}\nabla f(\hat{x}^n)) \tag{5}$$

The method provides theoretical guarantees on convergence to a local minimum of the cost function that is at least within $c||y - Ax^*||_2$ radius from globally optimal solution x^* for a certain constant c. However, like with the IHT agorithm the guarantees on convergence exist only if A satisfies restricted isometry property.

The work of (Efron et al., 2004) proposed least angle regression selection (LARS). The method proceed as follows. At first, a path of the solution that is indexed by a tuning parameter is built. Next, the final model is chosen on the path by a cross-validation procedure. One of the advantages of LARS is that the solution provided by it is piecewise linear and can be computed efficiently. However, as was shown in (Yuan & Lin, 2006) LARS tend to make selection based on the individual input values rather than the groups of input values which results in selection more features than necessary.

Another example of a method that follows a greedy matching strategy is Compressed Sensing Matching Pursuit (Do et al., 2008). It is a modification of the LARS algorithm that provides a significant boost in performance comparison with other methods of the same category. However, it also retains the drawbacks of LARS.

There also has been a study that focuses of evaluation of both sparse and sparse group feature selection methods (Yuan & Lin, 2006). The authors presented a set of extensions for several iterative methods (e.g. LARS) that give superior performance to the traditional step-wise backward elimination method in feature selection problem. Likewise, the work presents a comprehensive study of the methods including protocols for evaluation of both synthetic and real data.

3. Algorithms and models

We proceed with describing the main contributions of our work. Firstly, we describe the algorithm of Iterative Hard Thresholding that we've implemented. Secondly, we explain the details about our synthetic dataset that we use for the evaluation of the methods. Finally, we describe the metrics that we use for evaluation and explain their choice.

3.1. Iterative Hard Thresholding algorithm

Iterative Hard Thresholding algorithm is summarized in Alg. 1 and is identical to the version presented in the original paper (Blumensath & Davies, 2010). During the implementation we followed the pseudo-code of Alg. 1 and used Julia implementation suggested by TA's only as a reference. For a more detailed description, please, refer to the original paper (Blumensath & Davies, 2010) or our implementation.

3.2. Synthetic dataset

We generate the synthetic data following the procedure explained in (Xiang et al., 2014). Namely, the data are generated using a linear model $y = A\hat{x} + \epsilon$ where the design matrix A as well as noise ϵ follow a normal distribution.

Algorithm 1 Iterative Hard Thresholding

Initialise $x^1 = 0$, $\Gamma^1 = supp(H_K(\Phi^\top y))$
repeat
  $g^n = \Phi^\top(y - \Phi x^n)$
  $\mu^n = \frac{g^\top_{\Gamma^n} g_{\Gamma^n}}{g^\top_{\Gamma^n} \Phi^\top_{\Gamma^n} \Phi_{\Gamma^n}} g_{\Gamma^n}$
  $\tilde{x}^{n+1} = H_K(x^n + \mu^n g^n)$
  $\Gamma^{n+1} = supp(\tilde{x}^{n+1})$
  if $\Gamma^{n+1} = \Gamma^n$ **then**
    $x^{n+1} = \tilde{x}^{n+1}$
  end if
else if $\Gamma^{n+1} \neq \Gamma^n$ **then**
  if $\mu^n \leq (1-c)\frac{||\tilde{x}^{n+1} - x^n||_2^2}{||\Phi(\tilde{x}^{n+1} - x^n)||_2^2}$ **then**
    $x^{n+1} = \tilde{x}^{n+1}$
  end if
else if $\mu^n > (1-c)\frac{||\tilde{x}^{n+1} - x^n||_2^2}{||\Phi(\tilde{x}^{n+1} - x^n)||_2^2}$ **then**
  repeat
    $\mu^n = \mu^n / \kappa(1-c)$
    $\tilde{x}^{n+1} = H_K(x^n + \mu^n \gamma^n)$
  until $\mu^n \leq (1-c)\frac{||\tilde{x}^{n+1} - x^n||_2^2}{||\Phi(\tilde{x}^{n+1} - x^n)||_2^2}$
until stopping criterion is met

We additionally establish polynomial relationships between features using the procedure from (Yuan & Lin, 2006). To better understand the stability of each method to the outliers in the observations we vary the noise levels.

During our experiments we fix the size of the design matrix A to the size of 100×200 and generate 100 observations. The rest of the parameters such as noise level, the ratio of informative features (i.e. the true number of sparse features) as well as the degree of polynomial dependency between features are defined via configuration files of our pipeline. See the documentation in our repository for examples on how to change these parameters.

3.3. Metrics

To evaluate performance of different methods we employ a combination of protocols from different works (Blumensath & Davies, 2010; Xiang et al., 2014; Yuan & Lin, 2006). Namely, we report mean squared error and R^2 score of the methods being evaluated as well as the estimate of an oracle to which the noise values are known. The latter is required in order to estimate the ability of the methods to deal with the noise. Likewise, we report the noise level, the true number of generate informative features, and the degree of the polynomial dependency between features.

3.4. The structure of the framework

To set up the structure of our framework as well as ensure reproducibility and the ease of maintenance of our project we use Kedro. The underlying structure of this framework allows for the easy introduction of new datasets as well as methods and evaluation routines.

In order to add an existing dataset to our pipeline, it is enough to declare the dataset in Kedro catalog and its parameters inside a configuration file (see Figure 1) and create a node that takes the dataset as an input and processes it into a desirable format as illustrated in Figure 2. The usage of the configuration file allows having multiple instances of a dataset with different parameters.

The addition of a model and the evaluation for it can be done in a manner similar to the one explained before. The parameters for the evaluation can be setup in a manner shown in Figure 3. In Figure 4 the registration of a linear regression model and the permutation importance method is shown. The inputs for those nodes are taken from the pre-processing of the dataset in a declarative manner. Any dataset, as long as it is able to provide the data in the format required by the pipeline shown in Figure 4 can be used for evaluation without any change of the code.

Finally, to provide the functionality of running different models on different datasets in an easy way the pipeline registry is created as shown in Figure 5.

```yaml
# Whether to test on sparse features selection or on sparse group selection
option: 'sparse'
# Number of observations to generate
n: 100
# Number of parameters of the model
m: 200
# The noise level
noise_std: 1
# The rate of the informative features
redundancy_rate: 0.5
# What values to fill features with: 'normal' - from normal distribution, 'const' - with constants
features_fill: 'normal'
# Th degree of polynomial dependencies between features
poly_degree: 1
# Seed for reproducing the results
seed: 54
```

Figure 1. Dataset declaration via config.

```python
# Here now is only one pipeline for synthetic dataset creation, configured during the run or in the parameter file
def synth_test_data_pipeline(**kwargs):
    return Pipeline(
        [
            node(
                func=arrange_synth_test_data,
                inputs="parameters",
                outputs=["y", "X", "w", "y_true", "features_mask"],
                name="synth_test_data_node",
            ),
        ]
    )
```

Figure 2. Setting up the dataset processing pipeline in the code.

4. Experiments and results

We've conducted several tests to examine the properties of each method. We study how each method behaves when: there are non-linear dependencies between observations and features, high SNR and small number of actual informative

```yaml
# Explanation rate of the model. Used to determine the proposed number of informative features
explanation_rate: 0.95

# Permutation Importance parameters
# The number of time to repeat permutation
n_repeats: 30

# The number of features to select
k: 150
# Tolerance threshold for the termination
tol: 0.001
# The maximum number of iterations for IHT
max_iter: 1000
# Whether to provide output during IHT run
verbose: False
```

Figure 3. Evaluation parameters.

```python
def perm_importance_pipeline(**kwargs):
    return Pipeline(
        [
            node(
                func=fit_model,
                inputs=["y", "X"],
                outputs="regressor",
                name="fit_model_node",
            ),
            node(
                func=evaluate_perm_importance,
                inputs=["regressor", "y", "X", "w", "y_true", "features_mask", "parameters"],
                outputs=None,
                name="evaluate_perm_importance_node",
            ),
        ]
    )


def iht_pipeline(**kwargs):
    return Pipeline([
        node(
            func=evaluate_iht,
            inputs=["y", "X", "w", "y_true", "features_mask", "parameters"],
            outputs=None,
            name="evaluate_iht_node"
        )
    ])
```

Figure 4. Linear regression fitting and evaluation with the permutation importance method as well as IHT

features. The data generation procedure follows the one described in section 3.2.

In Figure 6 evaluation results for linear regression model with permutation importance (PI) and IHT feature selection are presented. The results of Test 1 (see 6) show that with high SNR PI gives good results both in MSE and R^2 as well as selects a number of features that is close to the ground-truth. IHT requires the true number of informative features to be provided so we do not report it. It can be seen that IHT gives significantly better results but outperforms oracle which means that it has also overfitted to the noise. If features are polynomial with the degree of 3 then MSE significantly increases as shown in Test 2 for PI, but for IHT there is no change at all. If the noise is too high and SNR is lower then 0 then although MSE and R^2 are both low the model is apparently over-fitting to the data as it performs significantly better than the oracle. The same holds true for IHT. In the case when the actual number of features is four times lower than the dimensionality of the feature space permutation importance fails to recover the

```python
def register_pipelines() -> Dict[str, Pipeline]:
    """Register the project's pipelines.

    Returns:
        A mapping from a pipeline name to a ``Pipeline`` object.
    """
    synth_dataset = dpp.synth_test_data_pipeline()
    perm_importance = dsp.perm_importance_pipeline()
    iht_importance = dsp.iht_pipeline()

    return {
        "__default__": synth_dataset + perm_importance,
        "synth_pi": synth_dataset + perm_importance,
        "synth_iht": synth_dataset + iht_importance
    }
```

Figure 5. A dictionary with different combinations of datasets and models. The selection of particular parameters for a combination can be done via command line or the config file.

number of informative features. IHT performs decently in this scenario.

To conclude, permutation importance works well when SNR is high, there is no non-linear dependencies between features and the number of informative features is not lower than 50%. When working with IHT it is better for the data to be noise-free or to employ some sort of regularization. Also, IHT needs the desired number of features to be provided.

	PI	IHT	PI	IHT	PI	IHT	PI	IHT
	Test 1		Test 2		Test 3		Test 4	
SNR (dB)	15.475		21.145		-0.088		13.303	
Inform. feat.	105		105		105		56	
Poly deg.	1		3		1		1	
Oracle MSE	1.055		0.987		37.992		1.073	
Oracle R^2	0.992		0.998		0.700		0.984	
Sel. feat.	107	-	102	-	111	-	92	-
MSE	3.440	0.029	13.272	0.029	5.855	0.128	1.707	8.308
R^2	0.972	1.000	0.968	1.000	0.960	0.992	0.976	0.882

Figure 6. Evaluation on synthetic dataset with different SNR, number of informative features and polynomial degree of dependencies between features. For evaluation the number of selected features, MSE and R^2 are reported along with oracle's estimates of errors.

5. Conclusion

In this work we have presented a framework for evaluation of the methods for sparse feature selection. The framework contains the implementation of the Iterative Hard Thresholding method along with the baseline permutation importance method. We have also showcased the results of the evaluation of the algorithms on our synthetic dataset.

References

Blumensath, T. and Davies, M. E. Normalized iterative hard thresholding: Guaranteed stability and performance. *IEEE Journal of selected topics in signal processing*, 4 (2):298–309, 2010.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Davies, M. and Mitianoudis, N. Simple mixture model for sparse overcomplete ica. *IEE Proceedings-Vision, Image and Signal Processing*, 151(1):35–43, 2004.

Do, T. T., Gan, L., Nguyen, N., and Tran, T. D. Sparsity adaptive matching pursuit algorithm for practical compressed sensing. In *2008 42nd Asilomar conference on signals, systems and computers*, pp. 581–587. IEEE, 2008.

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. Least angle regression. *Annals of statistics*, 32(2):407–499, 2004.

Févotte, C. and Godsill, S. J. Sparse linear regression in unions of bases via bayesian variable selection. *IEEE Signal Processing Letters*, 13(7):441–444, 2006.

Friedman, J., Hastie, T., and Tibshirani, R. A note on the group lasso and a sparse group lasso. *arXiv preprint arXiv:1001.0736*, 2010.

Xiang, S., Yang, T., and Ye, J. Simultaneous feature and feature group selection through hard thresholding. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 532–541, 2014.

Yuan, M. and Lin, Y. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68 (1):49–67, 2006.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Konstantin Pakulev 1 (97% of work)

- Reviewing literate on the topic (5 papers)
- Coding the algorithms (permutaion importance, Iterative Hard Thresholding)
- Preparing the synthetic dataset
- Preparing evaluation results
- Preparing the GitHub Repository
- Preparing the report
- Preparing peer reviews

Viacheslav Pronin (3% of work)

- Preparing a small section of GitHub Repository (see the commits)
- Preparing sections 'Real data' and 'Permutation importance' of the original version of the report (see the GitHub repository)
- Preparing 1.5 peer reviews during the first round of peer reviews

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.
   - ☐ Yes.
   - ☑ No.
   - ☐ Not applicable.

   Students' comment: None

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.
   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.
   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.
   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.
   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.
   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.
   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.
   - ☑ Yes.
   - ☐ No.
   - ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.
   - ☐ Yes.
   - ☐ No.
   - ☑ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.
    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.
    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.
    - ☐ Yes.
    - ☑ No.
    - ☐ Not applicable.

    Students' comment: None

13. A description of the computing infrastructure used is included in the report.
    - ☐ Yes.
    - ☑ No.
    - ☐ Not applicable.

    Students' comment: None