Feature Selection Package Implementation Plan

Week 6

November 23, 2014

1 Supported Data Format

CSV Format A really important thing to know is that scikit-learn estimators only take in continuous data in the form of NumPy arrays. There's a function in NumPy called loadtxt() that can read in a CSV file and convert it to an array with ease.

```
import numpy as np
glass_data = np.loadtxt('../data/glass_data.csv', delimiter=',')
glass_target = np.loadtxt('../data/glass_target.csv')
```

Svmlight/Libsvm Format Scikit-learn includes utility functions for loading datasets in the svmlight / libsvm format. In this format, each line takes the form <label> <feature-id>:<feature-value> <feature-id>:<feature-value> .... This format is especially suitable for sparse datasets. In this module, scipy sparse CSR matrices are used for X and numpy arrays are used for y.

```
from sklearn.datasets import load_svmlight_file
X_train, y_train = load_svmlight_file("/path/to/train_dataset.txt")
```

MATLAB Format Sci.io can also deal with MATLAB structure and cells.

```
import scipy.io
mat = scipy.io.loadmat('file.mat')
```

ARFF Format

```
>>> from scipy.io import arff
>>> from cStringIO import StringIO
```

```
>>> content = """
... @relation foo
... @attribute width  numeric
... @attribute height numeric
... @attribute color  {red,green,blue,yellow,black}
... @data
... 5.0,3.25,blue
... 4.5,3.75,green
... 3.0,4.00,red
... """
>>> f = StringIO(content)
>>> data, meta = arff.loadarff(f)
>>> data
array([(5.0, 3.25, 'blue'), (4.5, 3.75, 'green'), (3.0, 4.0, 'red')],
      dtype=[('width', '<f8'), ('height', '<f8'), ('color', '|S6')])
>>> meta
Dataset: foo
        width's type is numeric
        height's type is numeric
        color's type is nominal, range is ('red', 'green', 'blue', 'yellow', 'black')
```

2 PREPROCESSING IN SKLEARN

- Standardization, or mean removal and variance scaling: standard normally distributed data: Gaussian with zero mean and unit variance, Scaling features to a range, Centering kernel matrices
- Normalization: scaling individual samples to have unit norm
- Binarization:
- Encoding categorical features

3 EXISTING FEATURE SELECTION METHODS IN SKLEAN

- Removing features with low variance: removes all features whose variance doesn't meet some threshold.
- Univariate feature selection: selecting the best features based on univariate statistical tests, e.g., chi-2, f-test
- L1-based feature selection: for mono-output tasks and multi-output tasks
- Tree-based feature selection

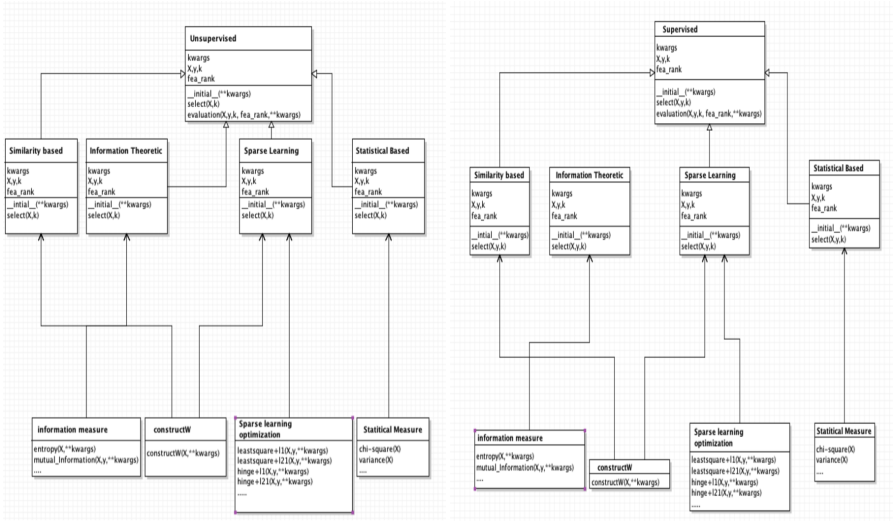
4 NOTATION

Throughout this documentation, matrices are written as boldface capital letters and vectors are denoted as boldface lowercase letters. For an arbitrary matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, M_{ij} denotes the (i, j)-th entry of $\mathbf{M}$ while $\mathbf{m}_i$ and $\mathbf{m}^j$ means the i-th row and j-th column of $\mathbf{M}$ respectively. $||\mathbf{M}||_F$ is the Frobenius norm of $\mathbf{M}$ and Tr($\mathbf{M}$) is the trace of $\mathbf{M}$ if $\mathbf{M}$ is square. $\langle \mathbf{A}, \mathbf{B} \rangle$ equals Tr($\mathbf{A}^T\mathbf{B}$), which is the standard inner product between two matrices. $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is a vector whose elements are all 1. The $l_{2,1}$-norm is defined as $||\mathbf{M}||_{2,1} = \sum_{i=1}^{m} ||\mathbf{m}_i|| = \sum_{i=1}^{m} \sqrt{\sum_{j=1}^{n} M_{ij}^2}$).

In particular, we use $\mathbf{X} \in \mathbf{R}^{n \times d}$ to denote the data matrix, where n is the number of instances and m is the number of features. For each data set, we use $f_1, \ldots, f_m$ to denote the m features, and $\mathbf{f}_1, \ldots, \mathbf{f}_m$ are the corresponding feature vectors, where $\mathbf{f}_i \in \mathbf{R}^n$ and $\mathbf{X} = (\mathbf{f}_1, \ldots, \mathbf{f}_m)$.

5 ARCHITECTURE

5.1 CLASS DIAGRAM



(a) (b)

Figure 5.1: (a) Unsupervised feature selection algorithms; (b) Supervised feature selection algorithms.

5.2 UNSUPERVISED AND SUPERVISED FEATURE SELECTION

There will be two parent class, UFeatureSelection class and FeatureSelection class. Both UFeatureSelection and FeatureSelection class has some attributes input data

(**X**), true label (**y**), number of selected features (k), feature ranking (feaRank) that will be used by its children class.

Attributes:

- **X**: numpy array, shape = [nsamples, nfeatures]
- **y**: numpy array, shape = [nsamples,1]
- k: int, number of selected features
- feaRank: int array, shape = [nfeatures, 1]

Each specific feature selection algorithm is a child class that inherits either the UFeatureSelection or FeatureSelection class, and it has an attribute kwargs which is a dictionary in python to store the needed parameters for this feature selection algorithm. All unsupervised feature selection algorithms and supervised feature selection algorithms also have a corresponding evaluation method to evaluate the selected features on clustering and classification task.

Unsupervised feature selection evaluation:
evaluation(**X**, **y**, k, **kwargs)
Input:

- **X**: numpy array, shape = [nsamples, nSelectedFeatures]
- **y**: numpy array, shape = [nsamples,1]
- k: int, number of selected features
- kwargs: dictionary
  “clusteringMethod": kmeans(default), etc
  “measure": accuracy(default), NMI, etc

Output:

- result: float

Supervised feature selection evaluation:
evaluation(**X**, **y**, k, **kwargs)
Input:

- **X**: numpy array, shape = [nsamples, nSelectedFeatures]
- **y**: bumpy array, shape = [nsamples,1]
- k: int, number of selected features
- kwargs: dictionary
  “classificationMethod": SVM(default), etc

"measure": accuracy(default), NMI, etc

Output:

- result: float

Besides the evaluation method, each child unsupervised feature selection class has a common function select(**X**, k) which selects the first k features and then sorts them in a descending order, storing the feature ranking in the feaRank attributes of the class.

Unsupervised feature selection common methods:
select(**X**, k)
Input:

- **X**: numpy array, shape = [nsamples, nSelectedFeatures]
- k: int, number of selected features

Output:

- feaRank: int array, shape = [nfeatures, 1]

Details:
1. Choose the top k features
2. Rank the features and store the ranking vector to feaRank attributes

Similar to the unsupervised feature selection algorithms, each child supervised feature selection class has a common function select(**X**, **y**, k) which selects the first k features and then sorts them in a descending order, storing the feature ranking in the feaRank attributes of the class.

Supervised feature selection common methods:
select(**X**, **y**, k)
Input:

- **X**: numpy array, shape = [nsamples, nSelectedFeatures]
- **y**: bumpy array, shape = [nsamples, 1]
- k: int, number of selected features

Output:

- feaRank: int array, shape = [nfeatures, 1]

Details:
1. Choose the top k features
2. Rank the features and store the ranking vector to feaRank attributes

6 AUXILIARY FUNCTIONS

6.1 CONSTRUCT WEIGHT MATRIX

The construction can be in a supervised or an unsupervised way, we also consider different similarity measure, it can be built based on sklearn.metrics.pairwise. For the constructW method, if the user does not specify any parameters, it will use the default the settings. Otherwise, the user can specify the parameters in the parameter kwargs and call the function constructW(**X**, **kwargs).

Weight matrix construction method:
construct(**X**, **kwargs)
Input:

- **X**: numpy array, shape = [nsamples, nSelectedFeatures]
- kwargs: dictionary
  "fisherScore": false (default) – boolean, build **W** according to Fisher Score
  "relief": false (default) – boolean, build **W** according to relief method
  "trueLabel": **y** (default None) – numpy array, shape = [nsamples, 1]
  "metric": "euclidean", "cosine" (default) – string
  "neighborMode": "knn" (default), "supervised" – string
  'weightMode": "binary" (default), "heatKernel", "cosine" – string
  "k": 5 (default) – int, the parameter needed for inn neighborMode
  "t": 1 (default) – float, parameter for "heatKernel" weightMode
  "bNormalized": false (default) – boolean, effective under "cosine" metric

Output:

- **W**: weight matrix – numpy array, shape = [nsamples, nsamples]

Details:
Detailed pseudo code is listed as follows:

7 SIMILARITY BASED FEATURE SELECTION

Similarity based feature selection algorithms are composed of three steps:

- Step1: calculate the similarity matrix **W** according to certain criterion
- Step2: for each feature $\mathbf{f}_i$, calculate its score $SC(\mathbf{f}_i) = \mathbf{f}_i^T \mathbf{W} \mathbf{f}_i$
- Step3: rank the features according to the score in descending order and select the top ranked features

Algorithm 1 constructW(**X**, **kwargs) – Initialization Part

```
if “metric" not in kwargs.keys(): then
    kwargs[“metric"] = “cosine"
end if
if kwargs[“metric"] == “cosine" and “bNormalized" not in kwargs.keys(): then
    kwargs[“bNormalized"] = false
else if kwargs[“metric"] != (“cosine" or “euclidean")  then
    error(“metric not exist”) and return
end if
if “neighborMode" not in kwargs.keys(): then
    kwargs[“neighborMode"] = “knn"
end if
if kwargs[“neighborMode"] ==“knn" and k not in kwargs.keys(): then
    k = 5
else if kwargs[“neighborMode"] ==“supervised" and k not in kwargs.keys(): then
    k = 5
else if kwargs[“neighborMode"] ==“supervised" and “trueLabel" not exist: then
    error(“trueLabel must be provided") and return
else if kwargs[“neighborMode"] != (“knn" or “supervised") then
    error(“neighborMode not exist") and return
end if
if “weightMode" not in kwargs.keys(): then
    kwargs[“weightMode"] = “binary"
end if
if kwargs[“weightMode"] == “heatKernel": then
    if kwargs[“metric"]!=“euclidean": then
        kwargs[“metric"] = “euclidean”
    end if
    if t not in kwargs.keys(): then
        t = 1
    end if
else if kwargs[“weightMode"] == “cosine": then
    if kwargs[“metric"]!=“cosine": then
        kwargs[“metric"] = “cosine”
    end if
    if “bNormalized" not in kwargs.keys(): then
        bNormalized = false
    end if
else
    error(“weightMode not exist") and return
end if
```

Algorithm 2 constructW(**X**, **kwargs) – continued (supervised mode)

nSamples = number of data samples, Label = vector of unique labels, nLabel = number of distinct labels
if kwargs[“neighborMode"] == “supervised": **then**
    if kwargs[“fisherScore"] == true: **then**
        for m = 1 : nLabel **do**
            if y(i)==y(j)==Label(m) **then**
                W(i,j) = 1/(number of samples with label == Label(m))
            end if
        end for
        return **W**
    end if
    if kwargs[“relieff"] == true: **then**
        if i == j **then**
            W(i,j) = 1
        else if **X**(i,:) is among the k nearest points to **X**(j,:) with the same class or **X**(j,:) is among the k nearest points to **X**(i,:) with the same class **then**
            W(i,j) = -1/k
        else if **X**(i,:) is among the k nearest points to **X**(j,:) with different class or **X**(j,:) is among the k nearest points to **X**(i,:) with different class **then**
            W(i,j)=1/(nLabel-1)k
        end if
        return **W**
    end if
    if kwargs[“weightMode"] == “binary" **then**
        for m = 1: nLabel **do**
            if **X**(i,:) is among the k nearest points (euclidean or cosine) to **X**(j,:) or **X**(j,:) among the k nearest points to **X** (i,:) within the same class Label(m) **then**
                W(i,j) = 1
            end if
        end for
    end if
    if kwargs[“weightMode"] == “heatKernel" **then**
        for m = 1: nLabel **do**
            if **X**(i,:) is among the k nearest points to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) within the same class Label(m) **then**
                W(i,j) = $e^{-\frac{||\mathbf{X}(i,:)-\mathbf{X}(j,:)||^2}{t}}$
            end if
        end for
    end if
    if kwargs[“weightMode"] == “cosine" **then**
        if kwargs[“bNormalized"]==false **then**
            for n = 1:nSamples **do**
                X(n,:)=**X**(n,:)/$\sqrt{||\mathbf{X}(n,:)||^2}$
            end for
        end if
        for m = 1: nLabel **do**
            if **X**(i,:) is among the k nearest points to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) within the same class Label(m) **then**
                W(i,j) = **X**(i,:)**X**(j,:)
            end if
        end for
    end if
end if

Algorithm 3 constructW(**X**, **kwargs) – continued (knn mode)

if kwargs[“neighborMode"]==“knn": **then**
  if kwargs[“weightMode"] ==“binary" **then**
    if kwargs[“metric"]==“euclidean" **then**
      if **X**(i,:) is among the k nearest points (euclidean) to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) **then**
        W(i,j)=1
      end if
    end if
    if kwargs[“metric"]==“cosine" **then**
      if **X**(i,:) is among the k nearest points (cosine) to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) **then**
        W(i,j)=1
      end if
    end if
  end if
  if kwargs[“weightMode"] ==“heatKernel" **then**
    if **X**(i,:) is among the k nearest points (euclidean) to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) **then**
      W(i,j)=$e^{-\frac{||\mathbf{X}(i,:)-\mathbf{X}(j,:)||^2}{t}}$
    end if
  end if
  if kwargs[“weightMode"] == “cosine" **then**
    if kwargs[“bNormalized"]==false **then**
      for n = 1:nSamples **do**
        X(n,:)=**X**(n,:)/$\sqrt{||\mathbf{X}(n,:)||^2}$
      end for
    end if
    if **X**(i,:) is among the k nearest points (cosine) to **X**(j,:) or **X**(j,:) is among the k nearest points to **X**(i,:) **then**
      W(i,j) = **X**(i,:)**X**(j,:)
    end if
  end if
end if
return **W**

7.1 LAPLACIAN SCORE

Laplacian Score is proposed in [1] to select features that retain sample locality specified by an affinity matrix **L**. In [2], it is reformulated as:

$$SC_L(\mathbf{f}_i) = \hat{\mathbf{f}}_i \mathbf{W} \hat{\mathbf{f}}_i, \quad \hat{\mathbf{f}}_i \triangleq (\mathbf{f}_i - \mu \mathbf{1})/\sigma \tag{7.1}$$

where $\mu = \frac{\mathbf{f}_i^T \mathbf{D1}}{\mathbf{1}^T \mathbf{D1}}$ and $\sigma = \sqrt{(\mathbf{f}_i - \mu \mathbf{1})^T \mathbf{D} (\mathbf{f}_i - \mu \mathbf{1})}$. The problem can be solved by greedily picking the top k features which have the maximal $SC_L(\mathbf{f}_i)$.

7.2 SPEC

SPEC is an extension for Laplacian score to make it more robust to noise [3]. Given **W** and the normalized Laplacian matrix $\mathscr{L}$, three evaluation functions are proposed.

$$\max_{F_{sub}} \sum_{\mathbf{f} \in \mathbf{F}_{sub}} \hat{\mathbf{f}}^T \mathbf{K} \hat{\mathbf{f}} \tag{7.2}$$

where

$$\text{evaluation 1:} \quad \mathbf{f}_{(1)} = \mathbf{f}/||\mathbf{D}^{1/2}\mathbf{f}||, \quad \hat{\mathbf{K}}_{(1)} = \mathbf{D}^{1/2}\mathbf{U}(\mathbf{I} - \gamma(\Sigma))\mathbf{U}^T\mathbf{D}^{1/2} \tag{7.3}$$

$$\text{evaluation 2:} \quad \mathbf{f}_{(2)} = (\mathbf{f} - \mu\mathbf{1})/||\mathbf{D}^{1/2}\mathbf{f}||, \quad \hat{\mathbf{K}}_{(2)} = \mathbf{D}^{1/2}\mathbf{U}(\mathbf{I} - \gamma(\Sigma))\mathbf{U}^T\mathbf{D}^{1/2} \tag{7.4}$$

$$\text{evaluation 3:} \quad \mathbf{f}_{(3)} = \mathbf{f}/||\mathbf{D}^{1/2}\mathbf{f}||, \quad \hat{\mathbf{K}}_{(3)} = \mathbf{D}^{1/2}\mathbf{U}_k(\gamma(2\mathbf{I}) - \gamma(\Sigma_k))\mathbf{U}_k^T\mathbf{D}^{1/2} \tag{7.5}$$

And (i) ndenotes the i-th criterion defined in SPE. **U** and Σ contain the singular vectors and singular values of the normalized Laplacian matrx, $L = \mathbf{U}\Sigma\mathbf{U}^T$. IN this case, the origninal similarity matrix **K** is mapped to $\hat{\mathbf{K}}$ by the operation $\gamma(\cdot)$ on Σ for the noise reduction.

7.3 FISHER SCORE

As shown in [1], when **W** is defined as

$$\mathbf{K}_{ij} = \begin{cases} \frac{1}{n_l} & \text{if } y_i = y_j = l \\ 0 & \text{otherwise} \end{cases} \tag{7.6}$$

where n_l is the number of samples that belong to class l. Laplacian Score and Fisher Score are equivalent and have the following relationship:

$$SC_F = \frac{1}{SC_L} - 1 \tag{7.7}$$

7.4 RELIEF/RELIEFF

Assume that the training data has c classes with l instances in each class; there are k instances in both NH(**x**) and NM(**x**); and all features have been normalized to unit length. NH(**x**) or NM(**x**,y) denotes a set of nearest points to **x** with the same class of

x, or a different class (the class y), respectively. According to [2], ReliefF is equivalent constructing a similarity matrix as follows:

$$W_{ij} = \begin{cases} 1, & \text{for } i = j \quad (7.8) \\ -\frac{1}{k}, & \text{for } x_j \in NH(x_i) \quad (7.9) \\ \frac{1}{(c-1)k}, & \text{for } x_j \in NM(x_i, y) \quad (7.10) \end{cases}$$

Then the ReliefF score is

$$SC_R(f) = -1 + \mathbf{f}^T \mathbf{W} \mathbf{f} \quad (7.11)$$

common function: RankFeatures(X, **kwargs)

Input:

- X: discrete data matrix
- W: the similarity matrix

Output:

- index: the rank of features in descending order

Algorithm 4 RankFeature

Input: X, **kwargs
Output: The rank of feathers in descending order

1: **for** each feature f_i **do**
2: calculate score(f_i) = $f^T * W * f$
3: **end for**
4: rank score of each feature in descending order
5: return the index

Algorithm 5 Laplacian Score

Input: X, Y, **kwargs
Output: The rank of feathers in descending order

1: construct the W by calling the constructW(X,Y,**kwargs), specify **kwargs
2: RankFeature(X, W = W)
3: rank score of each feature in descending order
4: return the index

8 Sparse Learning Based Feature Selection

Most of sparse learning based feature selection can be written as

$$\min_{\mathbf{W}} c(\mathbf{W}, \mathbf{X}) + \alpha \cdot p(\mathbf{W}) \quad (8.1)$$

Algorithm 6 FisherScore

Input: X, Y, **kwargs
Output: The rank of feathers in descending order
1: construct the W by calling the constructW(X,Y,**kwargs), specify **kwargs
2: RankFeature(X, W = W)
3: rank score of each feature in descending order
4: return the index

where $c(\mathbf{W},\mathbf{X})$ is the cost function and $p(\mathbf{W})$ is a penalty function with sparsity constraint.

8.1 General

The error function can be (for hinge loss and logistic, will add multi-class case later)

$$\begin{aligned} c(\mathbf{W},\mathbf{X}) &= ||\mathbf{X}\mathbf{W}-\mathbf{Y}||_F^2 \\ c(\mathbf{W},\mathbf{X}) &= \sum_{i=1}^{N} max(0, 1-y_i\mathbf{w}^T\mathbf{x}_i) \\ c(\mathbf{W},\mathbf{X}) &= \sum_{i=1}^{N} \log(1+\exp(-y_i(\mathbf{w}^T\mathbf{x}_i+b))) \end{aligned} \tag{8.2}$$

And the penalty can be

$$\begin{aligned} p(\mathbf{W}) &= ||\mathbf{W}||_1 \\ p(\mathbf{W}) &= ||\mathbf{W}||_{2,1} \\ p(\mathbf{W}) &= ||\mathbf{W}||_{p,\infty} \end{aligned} \tag{8.3}$$

8.2 Operators used By Proximal Gradient Descent and Alternating Direction Method of Multiplier

If using proximal gradient descent or alternating direction method of multiplier to optimize the sparse learning based feature selection method, one will result in solving the following function

$$\text{L21\_Proximal\_Operator: } \min_{\mathbf{W}} \frac{1}{2}||\mathbf{W}-\mathbf{A}||_F^2 + \rho||\mathbf{W}||_{2,1} \tag{8.4}$$

$$\text{L1\_Proximal\_Operator: } \min_{\mathbf{W}} \frac{1}{2}||\mathbf{W}-\mathbf{A}||_F^2 + \rho||\mathbf{W}||_1 \tag{8.5}$$

$$\text{LpInf\_Proximal\_Operator: } \min_{\mathbf{W}} \frac{1}{2}||\mathbf{W}-\mathbf{A}||_F^2 + \rho||\mathbf{W}||_{p,\infty} \tag{8.6}$$

base function: L21_Proximal_Operator(A, ρ)
Input:

- A: data matrix

- ρ: scalar

Output:

- W: the feature weight matrix

formula

- $\frac{1}{2}||W-A||_F^2+\rho||W||_{2,1}$

Algorithm 7 L21_Proximal_Operator

Input: A $\in \mathbb{R}^{m\times C}$, ρ
Output: W $\in \mathbb{R}^{m\times C}$

```
Initialize W to be a all zero matrix
for i = 1 : m do
    normAi = sqrt(sum_j A_ij^2)
    if normAi > ρ then
        the ith row of W = (1 - ρ / normWi) * the i-th row of A
    end if
end for
return W
```

base function: L1_Proximal_Operator(A, ρ)
Input:

- A: data matrix
- ρ: scalar

Output:

- W: the feature weight matrix

formula

- $\frac{1}{2}||W-A||_F^2+\rho||W||_1$

base function: LpInf_Proximal_Operator(A, ρ)
Input:

- A: data matrix
- ρ: scalar

Output:

- W: the feature weight matrix

Algorithm 8 L21_Proximal_Operator

Input: A $\in \mathbb{R}^{m \times C}$, ρ
Output: W $\in \mathbb{R}^{m \times C}$

```
Initialize W to be a all zero matrix
for i = 1 : m do
    for j = 1 : C do
        if A_ij > ρ then
            W_ij = A_ij − ρ
        else if A_ij < −ρ then
            W_ij = A_ij + ρ
        end if
    end for
end for
return W
```

formula

- $\frac{1}{2}||W - A||_F^2 + \rho||W||_{p,\infty}$

Algorithm 9 LpInf_Proximal_Operator

Input: A $\in \mathbb{R}^{m \times C}$, ρ
Output: W $\in \mathbb{R}^{m \times C}$

```
Initialize W to be a all zero matrix
for i = 1 : m do
    for j = 1 : C do
        if A_ij > ρ then
            W_ij = A_ij − ρ
        else if A_ij < −ρ then
            W_ij = A_ij + ρ
        end if
    end for
end for
return W
```

8.3 Proximal Gradient Descent

The objective function is

$$F(W = f(W) + \lambda\phi(W) \tag{8.7}$$

where f(W) is some convex and differentiable function and $\phi(W)$ is non-smooth but convex. In each iteration of the proximal algorithm, F(W) is linearized around the current estimate W_t, and the value of W is updated as the solution of the following

proximal gradient descent problem

$$W_{t+1} = \arg\min_{W} G_{\eta_t}(W, W_t) \tag{8.8}$$

where $G_{\eta_t}(W, W_t)$ is defined as

$$G_{\eta_t}(W, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W - W_t)\right) + \frac{\eta_t}{2}||W - W_t||_F^2 + \lambda\phi(W) \tag{8.9}$$

To be specific, the update rule of W_{t+1} is

$$W_{t+1} = \arg\min \frac{1}{2}||W - (W_t - \frac{1}{\eta_t}\nabla f(W_t))|| + \frac{\lambda}{\eta_t}\phi(W) \tag{8.10}$$

which is exactly the proximal projection operator we defined previously.

8.4 *LeastSquare_L21*

function LeastSquare_L21(X, Y, **kwargs)
formula:

$$\min_{W} F(W) = f(W) + \lambda\phi(W) = \frac{1}{2}||XW - Y||_F^2 + \frac{\beta}{2}Tr(W^T X^T LXW) + \lambda||W||_{2,1} \tag{8.11}$$

where $f(W) = \frac{1}{2}||XW - Y||_F^2 + \frac{\beta}{2}Tr(W^T X^T LXW)$ The derivative of f(W) with respect to W is:

$$\nabla f(W = X^T XW - X^T W + \beta X^T LYXW \tag{8.12}$$

8.5 *LeastSquare_L1*

function LeastSquare_L1(X, Y, **kwargs)
formula:

$$\min_{W} F(W) = f(W) + \lambda\phi(W) = \frac{1}{2}||XW - Y||_F^2 + \frac{\beta}{2}Tr(W^T X^T LXW) + \lambda||W||_1 \tag{8.13}$$

where $f(W) = \frac{1}{2}||XW - Y||_F^2 + \frac{\beta}{2}Tr(W^T X^T LXW)$ The derivative of f(W) with respect to W is:

$$\nabla f(W = X^T XW - X^T W + \beta X^T LYXW \tag{8.14}$$

8.6 *Logistic_L21*

The objective function is

$$\min_{W,b} F(W) = f(W) + \lambda\phi(W) = \sum_{i=1}^{N}\sum_{j=1}^{C} \log\left[1 + \exp\left(-Y_{ij}(x_i w^j + b_j)\right)\right] + \lambda||W||_{2,1} \tag{8.15}$$

Algorithm 10 LeastSquare_L21

Input: X, Y, **kwargs (λ, β, L)

Output: feature in descending order by $||w_i||$

Initialize $\eta_0, W_1, \alpha_1 = 1, t = 1, V_1 = W_1, learning_rate$
while t < maxIter **do**
    calculate $\nabla f(W_t)$ as defined in Eq. (8.11)
    calculate $F(W_t)$ where F() is defined in Eq.(8.11)
    calculate W_{tmp} by calling function L21_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
    calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_{2,1}$
    while $F(W_t) > G_{\eta_{t-1}(W_{tmp}, W_t)}$ **do**
        $\eta_{t-1} = 2\eta_{t-1}$
        calculate W_{tmp} by calling function L21_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
        calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_{2,1}$
    end while
    $\eta_t = \eta_{t-1}$
    $W_{t+1} = L21\_Proximal\_Operator(V_t - \frac{1}{\eta_t}\nabla f(V_t), \eta_t)$
    $\alpha_{t+1} = \frac{1+\sqrt{1+4\alpha_t^2}}{2}$
    compute $V_{t+1} = W_t + \frac{\alpha_t - 1}{\alpha_{t+1}(W_{t+1} - W_t)}$
end while
sort feature in descending order by $||w_i||$
return sorted features

Algorithm 11 LeastSquare_L1

Input: X, Y, **kwargs (λ, β, L)

Output: feature in descending order by $||w_i||$

Initialize $\eta_0, W_1, \alpha_1 = 1, t = 1, V_1 = W_1, learning_rate$
while t < maxIter **do**
    calculate $\nabla f(W_t)$ as defined in Eq. (8.13)
    calculate $F(W_t)$ where F() is defined in Eq.(8.13)
    calculate W_{tmp} by calling function L1_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
    calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_1$
    while $F(W_t) > G_{\eta_{t-1}(W_{tmp}, W_t)}$ **do**
        $\eta_{t-1} = 2\eta_{t-1}$
        calculate W_{tmp} by calling function L1_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
        calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_1$
    end while
    $\eta_t = \eta_{t-1}$
    $W_{t+1} = L21\_Proximal\_Operator(V_t - \frac{1}{\eta_t}\nabla f(V_t), \eta_t)$
    $\alpha_{t+1} = \frac{1+\sqrt{1+4\alpha_t^2}}{2}$
    compute $V_{t+1} = W_t + \frac{\alpha_t - 1}{\alpha_{t+1}(W_{t+1} - W_t)}$
end while
return W

where x_i means the i-th row of X anfd w^j means j-th column of W. And f(W) is

$$f(W) = \sum_{i=1}^{N} \sum_{j=1}^{C} \log\left[1 + \exp\left(-Y_{ij}(x_i w^j + b_j)\right)\right] \tag{8.16}$$

Then the partial derivative of f(W) with respect to W is

$$\frac{\partial f(W)}{\partial W_{mn}} = \sum_{i=1}^{N} -Y_{in} X_{im} \frac{\exp\left(-Y_{in}(x_i w^n + b_n)\right)}{1 + exp\left(-Y_{in}(x_i w^n + b_n)\right)} \tag{8.17}$$

The partial derivative of f(W) with respect to b is

$$\frac{\partial f(W)}{\partial b_n} = \sum_{i=1}^{N} -Y_{in} \frac{\exp\left(-Y_{in}(x_i w^n + b_n)\right)}{1 + exp\left(-Y_{in}(x_i w^n + b_n)\right)} \tag{8.18}$$

Algorithm 12 Logistic_L21

Input: X, Y, **kwargs (λ)
Output: feature in descending order by $||w_i||$

Initialize $\eta_0, W_1, \alpha_1 = 1, b_1, t = 1, V_1 = W_1, learning_rate$
while t < maxIter **do**
  calculate $\nabla f(W_t)$ as defined in Eq. (8.17)
  calculate $F(W_t)$ where F() is defined in Eq.(8.15)
  calculate W_{tmp} by calling function L21_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}} \nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
  calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2} ||W_{tmp} - W_t||_F^2 + \lambda ||W_{tmp}||_{2,1}$
  while $F(W_t) > G_{\eta_{t-1}(W_{tmp}, W_t)}$ **do**
    $\eta_{t-1} = 2\eta_{t-1}$
    calculate W_{tmp} by calling function L21_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}} \nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
    calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2} ||W_{tmp} - W_t||_F^2 + \lambda ||W_{tmp}||_{2,1}$
  end while
  $\eta_t = \eta_{t-1}$
  $W_{t+1} = L21\_Proximal\_Operator(V_t - \frac{1}{\eta_t} \nabla f(V_t), \eta_t)$
  $\alpha_{t+1} = \frac{1 + \sqrt{1 + 4\alpha_t^2}}{2}$
  compute $V_{t+1} = W_t + \frac{\alpha_t - 1}{\alpha_{t+1}(W_{t+1} - W_t)}$
  compute b as $b_n = b_n - learning_rate * \frac{\partial f(V_t)}{\partial b_n}$
end while
sort feature in descending order by $||w_i||$
return sorted features

8.7 *Logistic_L1*

The objective function is

$$\min_{W,b} F(W) = f(W) + \lambda\phi(W) = \sum_{i=1}^{N}\sum_{j=1}^{C} \log\left[1 + \exp\left(-Y_{ij}(x_i w^j + b_j)\right)\right] + \lambda||W||_1 \quad (8.19)$$

Algorithm 13 Logistic_L1

Input: X, Y, **kwargs (λ)
Output: feature in descending order by $||w_i||$

Initialize $\eta_0, W_1, \alpha_1 = 1, b_1, t = 1, V_1 = W_1$
while t < maxIter **do**
  calculate $\nabla f(W_t)$ as defined in Eq. (8.17)
  calculate $F(W_t)$ where F() is defined in Eq.(8.19)
  calculate W_{tmp} by calling function L1_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
  calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_1$
  while $F(W_t) > G_{\eta_{t-1}(W_{tmp}, W_t)}$ **do**
    $\eta_{t-1} = 2\eta_{t-1}$
    calculate W_{tmp} by calling function L1_Proximal_Operator($W_t - \frac{1}{\eta_{t-1}}\nabla f(W_t), \frac{\lambda}{\eta_{t-1}}$)
    calculate $G_{\eta_{t-1}}(W_{tmp}, W_t)$ as $G_{\eta_{t-1}}(W_{tmp}, W_t) = f(W_t) + Tr\left(\nabla f(W_t)^T (W_{tmp} - W_t)\right) + \frac{\eta_{t-1}}{2}||W_{tmp} - W_t||_F^2 + \lambda||W_{tmp}||_1$
  end while
  $\eta_t = \eta_{t-1}$
  $W_{t+1} = L21\_Proximal\_Operator(V_t - \frac{1}{\eta_t}\nabla f(V_t), \eta_t)$
  $\alpha_{t+1} = \frac{1+\sqrt{1+4\alpha_t^2}}{2}$
  compute $V_{t+1} = W_t + \frac{\alpha_t - 1}{\alpha_{t+1}(W_{t+1} - W_t)}$
  compute b as $b_n = b_n - learning_rate * \frac{\partial f(V_t)}{\partial b_n}$
end while
return W

8.8 Algorithms Using Chris Ding's Method

8.9 Efficient and Robust Feature Selection

$$||\mathbf{XW} - \mathbf{Y}||_{2,1} + \gamma||\mathbf{W}||_{2,1} \quad (8.20)$$

Algorithm 14 LSL21_V2

Input: $X \in R^{N\times m}$, $Y \in R^{N\times K}$, **kwargs (λ, α, L)
Output: feature in descending order by $||w_i||$

if "λ" not in **kwargs **then**
    $\lambda = 0.1$
end if
if "α" not in **kwargs **then**
    $\alpha = 0$
else if " **then**L" not in **kwargs
    print error, need L
end if
if "maxIter" not in **kwargs **then**
    maxIter = 1000
end if
Initialize W to be an identity matrix or using $W = (X^T X + 0.5 * I)^{-1}(X^T Y)$
for i = 1:maxIter **do**
    Construct an identity matrix $D \in R^{m\times m}$ with $D_{ii} = \frac{1}{||w_i||}$, where $||w_i|| = EuclideanNorm(W(i,:))$
    Update W as $W = (X^T X + \alpha X^T L X + \gamma D)^{-1}(X^T Y)$
end for
return W

Algorithm 15 ERFS

Input: $X \in R^{N\times d}$, Y, **kwargs (γ)

if "γ" not in **kwargs **then**
    $\gamma = 0.1$
end if
if "maxIter" not in **kwargs **then**
    maxIter = 1000
end if

Output: feature in descending order by $||w_i||$

$A = [X, \gamma I_{N\times N}] \in R^{N\times(m)}$, where $I_{N\times N}$ is an identity matrix of size N by N
Initialize $D \in R^{m\times m}$ as an identity matrix
for i = 1:maxIter **do**
    calculate U as $U = D^{-1}A^T(AD^{-1}A^T)^{-1}Y$
    calculate D as $D_{ii} = \frac{1}{2||u_i||}$
end for
W = the first d rows of U
return W

8.10 MCFS

MCFS[4]: Unsupervised Feature Selection for Multi-Cluster Data. First calculate the spectral embedding as

$$\mathbf{Ly} = \lambda \mathbf{Dy} \tag{8.21}$$

Then do the spectral regression:

$$\begin{aligned} \min_{\mathbf{a}_k} \quad & ||\mathbf{y}_k - \mathbf{X}^T \mathbf{a}_k||_F^2 \\ s.t. \quad & |\mathbf{a}_k| \leq n \end{aligned} \tag{8.22}$$

$$MCFS(j) = \max_k |a_{k,j}| \tag{8.23}$$

Algorithm 16 MCFS

Input: $X \in R^{N \times d}$, **kwargs (L, K, n, where K is number of clusters and n is number of features to select)

Output: features in descending order of MCFS score

1: **if** "L" not in **kwargs **then**
2: print error, need L
3: **end if**
4: **if** "K" not in **kwargs **then**
5: print error, need K
6: **end if**
7: **if** "n" not in **kwargs **then**
8: print error, need n
9: **end if**
10: Solve the generalized eigend-problem in Eq. (8.21), Let $Y = [y_1, \ldots, y_K]$ contian the top K eigenvectoes with respect to the smallest eigenvalues (use scipy.linalg.eig)
11: solve K L1-regularized regression problem in Eq. (8.22) using sklearn.linear_model.LassoLars(????????????? this one doesn't satisfy the requirement)
12: compute the NCFS score for each feature according to Eq. (8.23)
13: return features in descending order of MCFS scores

8.11 UDFS

8.12 NDFS

$$\begin{aligned} &\min_{F,W} Te(F^T LF) + \alpha(||XW - F||_F^2 + \beta||W||_{2,1}) \\ &s.t. F^T F = I, F \geq 0 \end{aligned} \tag{8.24}$$

Algorithm 17 NDFS

Input: $X \in R^{N \times d}$, Y, **kwargs (α, β, γ)

if "γ" not in **kwargs **then**
    $\gamma = 5000$
end if
if "maxIter" not in **kwargs **then**
    maxIter = 1000
end if

Output: features in descending order by $||w_i||$

Initialize $F \in R^{N \times K}$ suing k-means
Set $D \in R^{d \times d}$ as an identity matrix
for i = 1:maxIter **do**
    $M = L + \alpha(I_{N \times N} - X(X^T X + \beta D)^{-1} X^T)$
    $F_{ij} = F_{ij} \frac{(\gamma F)_{ij}}{(MF + \gamma F F^T F)_{ij}}$
    $W = (X^T X + \beta D)^{-1} X^T F$
    update D with $D_[ii] = \frac{1}{||w_i||}$
end for
sort features in descending order by $||w_i||$
return features in descending order by $||w_i||$

8.13 RUFS

8.14 Adaptive Lasso based*

8.15 Group lasso based*

9 Information Theoretic Feature Selection

entropy: quantifies the uncertainty present in the distribution of a feature X

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{9.1}$$

See **scipy.stats.entropy(pk, qk=None, base=None)**

conditional entropy: the entropy of X conditioned on Y

$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y) \tag{9.2}$$

mutual information: the amount of inforamtion shared by X and Y

$$I(X;Y) = H(X) - H(X|Y) = \sum_{x\in X}\sum_{y\in Y} p(xy)\log\frac{p(xy)}{p(x)p(y)} \tag{9.3}$$

See **sklearn.metrics.mutual_info_score(labels_true, labels_pred, contingency=None)**

conditional mutual information

$$I(X;Y|Z) = H(X|Z) - H(X|YZ) = \sum_{z\in Z} p(z)\sum_{x\in X}\sum_{y\in Y} p(xy|z)\log\frac{p(xy|z)}{p(x|z)p(y|z)} \tag{9.4}$$

See **Non-Parametric Entropy Estimation Toolbox** or **pyentropy**

9.1 Criteria as Linear Combination of Shannon Information Terms

A greedy optimization process which accesses features based on a simple scoring criteria:

$$J_{cmi} = I(f_k;Y) - \beta\sum_{j\in F} I(f_j;X_k) + \gamma\sum_{j\in F} I(f_j;f_k|Y) \tag{9.5}$$

where F is the currently selected feature subset, f_k is an unselected feature, Y is label, β and γ are two parameters which depend on the feature selection algorithms.

- Step1: initialize F to be empty
- Step2: for each feature not in S, calculate J_{cmi}
- Step3: put the the feature with the largest J_{cmi} to F
- Step4: go to step2

Different β and γ result in feature selection algorithm:

MIFS: Mutual Information Feature Selection, $\beta \in [0,1], \gamma = 0$
MIM: Mutual Information Maximization, $\beta = 0, \gamma = 0$
CIFE: Conditional Infomax Feature Extraction, $\beta = 1, \gamma = 1$
JMI: Joint Mutual Information, $\beta = \frac{1}{|F|}, \gamma = 0$
MRMR: Max-Relevance Min-Redundancy, $\beta = \frac{1}{|F|}, \gamma = \frac{1}{|F|}$

common function: LCSI(X, Y, **kwargs)

Input:

- X: discrete data matrix
- Y: label

- beta:
- gamma:

Output:

- F: selected features

Detail:

Algorithm 18 LCSI

Input: X, Y, **kwargs (Beta, Gamma, FunctionName)
Output: Selected Features

Initialize S to contain all the features, and F to be empty
Beta = kwargs['Beta'], Gamma = ['Gamma']
repeat
    for each $f_i \in S$ **do**
        if kwargs['FunctionName'] == 'JMI' **then**
            Beta = $\frac{1}{|F|}$
        else if kwargs['FunctionName'] == 'MRMR' **then**
            Beta = $\frac{1}{|F|}$, Gamma = $\frac{1}{|F|}$
        end if
        Calculate $J_{cmi}(f_i)$ using Eq.(9.5) by calling function 9.3 and function 9.4
    end for
    find the feature that has the largest positive J_{cmi}, say f_k
    put f_k into F: F = F $\cup$ $\{f_k\}$
    remove f_k from S
until no J_{cmi} is positive
return the selected features F

Algorithm 19 MIFS

Input: X, Y, **kwargs
Output: Selected Features

if 'Beta' not in kwargs **then**
    Beta = 0.5
else
    Beta = kwargs['Beta']
end if
F = LCSI(X, Y, Beta, Gamma=0)
return the selected features F

Algorithm 20 MIM

Input: X, Y, **kwargs
Output: Selected Features
1: F = LCSI(X, Y, Beta=0, Gamma=0)
2: return the selected features F

Algorithm 21 CIFE

Input: X, Y, **kwargs
Output: Selected Features
1: F = LCSI(X, Y, Beta=1, Gamma=1)
2: return the selected features F

Algorithm 22 JMI

Input: X, Y, **kwargs
Output: Selected Features
1: F = LCSI(X, Y, Gamma=0, FunctionName = 'JMI')
2: return the selected features F

Algorithm 23 MRMR

Input: X, Y, **kwargs
Output: Selected Features
1: F = LCSI(X, Y, FunctionName = 'MRMR')
2: return the selected features F

9.2 Criterion as Non-Linear Combinations of Shannon Information Terms

CMIM: Conditional Mutual Information Maximization

$$J_{cmim}(f_k) = I(f_k;Y) - \max_{f_j \in F}[I(f_k;f_j) - I(f_k;f_j|Y)] \tag{9.6}$$

IF: Informative Fragments, equivalent to CMIM

$$J_{if}(f_k) = J_{cmim}(f_k) \tag{9.7}$$

ICAP: Interaction Capping

$$J_{icap}(f_k) = I(f_k;Y) - \sum_{f_j \in F} \max[0, \{I(f_k;f_j) - I(f_k;f_j|Y)\}] \tag{9.8}$$

DISR: Double Input Symmetrical Relevance

$$J_{disr}(f_k) = \sum_{f_j} \frac{I(f_k f_j;Y)}{H(f_k f_j Y)} \tag{9.9}$$

Algorithm 24 CMIM

Input: X, Y, **kwargs
Output: Selected Features

Initialize S to contain all the features, and F to be empty
Please store some intermediate values that will be used repeated to accelerate the speed
repeat
    for $f_i \in S$ **do**
        for $f_j \in F$ **do**
            calculate $I(f_i; f_j)$ by calling the mutual entropy function
            calculate $I(f_i; f_j|Y)]$ by calling the conditional mutual entropy function
            $Table[f_i; f_j] = I(f_i; f_j) - I(f_i; f_j|Y)]$
        end for
        maxValue = from the row of f_i in Table, find the largest one with $f_j \in F$
        calculate $I(f_i; Y)$ by calling the mutual information function
        $J_{cmim}(f_i) = I(f_i; Y) - maxValue$
    end for
    find the feature in S with the largest J_{cmim} value, say f_k
    put f_k into F
    remove f_k from S
until
return the selected features F

Algorithm 25 IF

The same as Algorithm 9.2

Algorithm 26 ICAP

Input: X, Y, **kwargs
Output: Selected Features

Initialize S to contain all the features, and F to be empty
repeat
    for $f_i \in S$ **do**
        for $f_j \in F$ **do**
            calculate $I(f_i; f_j)$ by calling the mutual entropy function
            calculate $I(f_i; f_j|Y)]$ by calling the conditional mutual entropy function
            $Table[f_i; f_j] = I(f_i; f_j) - I(f_i; f_j|Y)]$
        end for
        maxValue = from the row of f_i in Table, find the largest one with $f_j \in F$
        calculate $I(f_i; Y)$ by calling the mutual information function
        $J_{cmim}(f_i) = I(f_i; Y) - maxValue$
    end for
    find the feature in S with the largest J_{cmim} value, say f_k
    put f_k into F
    remove f_k from S
until no $J_{ICAP} > 0$
return the selected features F

9.3 FCBF

Fast Correlation Based Filter [5] is a filter model feature selection algorithm that measure feature-class and feature-feature correlation. FCBF starts by selecting a set of features $S^{'}$ that is highly correlated to the class with SU > δ, where δ is a threshold given by the user. In FCBF, a features f_i with symmetrical uncertainty $SU_{i,c}$ to the class c will be called predominant $iffSU_{i,c} \geq \delta$ and there is no f_i such that $SU_{j,i} \geq SU_{i,c} \forall f_j \in S^{'} where(j \neq i)$. However, if there exists such feature f_j where $SU_{j,i} \geq SU_{i,c}$, then f_j will be called redundant feature to f_i. Then, this set of redundant features will be denoted as S_{P_i}, which will be further split into $S^+_{P_i}$ and $S^-_{P_i}$ where they contain redundant feature to f_i with $SU_{j,c} > SU_{i,c}$ and $SU_{j,c} \leq SU_{i,c}$ respectively. Finally, FCBF applies three heuristic on $S_{P_i}, S^+_{P_i}$, and $S^-_{P_i}$ that remove the redundant features and keep the feature that most relevant to the class. The symmetrical uncertainty is defined as:

$$SU(X, Y) = 2[\frac{IG(X, Y)}{H(X) + H(Y)}] \tag{9.10}$$

function calculateSU(f1, f2)
input

- f1: a vector

- f2: a vector of the same length as f1

output

- su

Algorithm 27 calculateSU

Input: f1, f2
Output: su score of f1 and f2
1: calculate IG(f1,f2) by calling the information gain function
2: calculate H(f1) by calling the entropy function
3: calculate H(f2) by calling the entropy function
4: return su as defined in Eq. (9.10)

Algorithm 28 FCBF

Input: X = $[f_1, f_2, \ldots, f_d]$, Y, **kwargs (δ: a threshold)
Output: Selected Feature Subset
1: Initialize S_{list} to be empty
2: **for** i = 1:d **do**
3: calculate $SU_{i,Y}$ for f_i by calling the function calculateSU
4: **if** $SU_{i,Y} > \delta$ **then**
5: append f_i to S_{list}
6: **end if**
7: **end for**
8: order S_{list} in descending order of $SU_{i,c}$ value
9: Initialize F to be empty
10: **while** S_{list} not empty **do**
11: F_p = get the first element of S_{list} and remove F_p from S_{list}
12: append F_p to F
13: **for** each feature f in S_{list} **do**
14: **if** calculateSU(F_p, f) ≥ calculateSU(f,Y) **then**
15: remove f from S_{list}
16: **end if**
17: **end for**
18: **end while**
19: return the selected features F

9.4 Information Gain

$$IG(f, Y) = H(f) - H(f|Y) \tag{9.11}$$

where

$$H(f) = -\sum_i P(f_i)\log_2(P(f_i))$$
$$H(f|Y) = -\sum_j P(y_j)\sum_i P(f_i|y_i)\log_2(P(f_i|y_j)) \tag{9.12}$$

Algorithm 29 IGFS

Input: X, Y, **kwargs
Output: Feature Index in descending order of their information gain value
1: **for** i = 1:d **do** % d is number of features
2: calculate information gain of f_i by calling the information gain function
3: **end for**
4: sort the features according to their information gain value in descending order
5: return the sorted features

10 Statistics Based Feature Selection

10.1 Gini Index

$$\text{GiniIndex}(f) = 1 - \sum_{i=1}^{C}[p(i|f)]^2 \tag{10.1}$$

Algorithm 30 GIFS

Input: X, Y, **kwargs
Output: Feature Index in descending order of their Gini-index balue
1: **for** i = 1:d **do** % d is number of features
2: calculate Gini-index of f_i by calling the information gain function
3: **end for**
4: sort the features according to their information gain value in descending order
5: return the sorted features

10.2 CFS

CFS uses a correlation based heuristic to evaluate the worth of features:

$$Merit_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \tag{10.2}$$

Where $Merit_S$ is the heuristic "merit" of a feature subset S containing k features, $\overline{r_{cf}} = \sum_{f_i \in S}\frac{1}{k}SU(f_i, Y)$ is the mean feature class correlation and $\overline{r_{ff}} = \sum_{f_i, f_j \in S}\frac{1}{k(k-1)}SU(f_i, f_j)$ is the average feature inter-correlation. Here SU is the symmetrical uncertainty

function, which is defined in Eq. (9.10). The mean feature-class correlation (numerator) is an indication to how easily a class could be predicted based on the feature subset. And the average feature-feature inter correlation (denominator) determines correlation between features which indicates the level of redundancy between them. Feature correlations are estimated based on the information theory that determines the degree of association between features. The amount of information by which the entropy of Y decreases reflects the additional information about Y provided by X which is measured via Information Gain. SInce, information gain is usually biased in favor of features with more values, symmetrical uncertainty is used.

CFS explores the search space using the Best First search. It estimates the utility of a feature by considering its predictive ability and the degree of correlation (redundancy) it introduces to the selected feature set. More specifically, CFS calculates feature-class and feature-feature correlations using symmetrical uncertainty and then selects a subset of features using the Best First search with a stopping criterion of five consecutive fully expanded non-improving subsets. Merits of CFS are it does not need to reserve any part of the training data for evaluation purpose and works well on smaller data sets. It selects the maximum relevant feature and avoids the reintroduction of redundancy. But the drawback is that CFS cannot handle problems where class is numeric.

Algorithm 31 calculateCFSMerit

Input: X, Y, **kwargs
Output: CFS Merit for this feature subset

initialize rcf and rff to be 0, k = number of features
for each feature f in X **do**
    calculate rcf = rcf + calculateSU(f,Y)
end for
for each feature f_i in X **do**
    for each feature f_j in X with j > i **do**
        rff = rff + calculateSU(f_i, f_j)
    end for
end for
return $\frac{fcf}{\sqrt{k+rff}}$

10.3 T-SCORE, F-SCORE*

call existing feature selection algorithm

10.4 CHI-SQUARE

call existing feature selection algorithm

Algorithm 32 CFS

Input: X, Y, **kwargs
Output: Selected Feature Subset

1: Initialize S to be empty, U to contain all the features, and Merit an empty list
2: **repeat**
3: **for** each feature f in U **do**
4: T = $S \cup \{f\}$
5: calculate the merit by calling calculateCFSMerit(T, Y)
6: **end for**
7: put the feature with the largest merit to S, remove that feature from U and put append the corresponding merit to Merit
8: **until** the merits of five consecutive fully expanded subsets don't improve
9: return the selected feature subset

10.5 Low Variance Feature Selection

call existing feature selection algorithm

11 Recursive Feature Selection

11.1 Decision Tree Based

11.2 SVM Based

12 Evaluation

Cross Validation sklearn.cross_validation

For supervised feature selection

- ACC: sklearn.metrics.accuracy_score
- NMI: sklearn.metrics.normalized_mutual_info_score
- SVM: sklearn.svm

For unsupervised feature selection

- ACC: sklearn.metrics.adjusted_rand_score(labels_true, labels_pred)
- NMI: sklearn.metrics.adjusted_mutual_info_score(labels_true, labels_pred)
- Clustering: sklearn.cluster

Other Metrics

- Precision_Recall: sklearn.metrics.precision_recall_curve(y_true, probas_pred, pos_label=None, sample_weight=None)

- Average_Precision_Score: sklearn.metrics.average_precision_score(y_true, y_score, average='macro', sample_weight=None)
- ROC_AUC_Score: sklearn.metrics.roc_auc_score(y_true, y_score, average='macro', sample_weight=None)
- AUC: sklearn.metrics.auc(x, y, reorder=False)
- F1_Score: sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)

REFERENCES

[1] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in neural information processing systems*, 2005, pp. 507–514.

[2] Z. Zhao, L. Wang, H. Liu, and J. Ye, "On similarity preserving feature selection," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 3, pp. 619–632, 2013.

[3] Z. Zhao and H. Liu, "Spectral feature selection for supervised and unsupervised learning," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 1151–1157.

[4] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 333–342.

[5] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *ICML*, vol. 3, 2003, pp. 856–863.