Machine Learning Package

Portfolio of Machine Learning Algorithms

Summary

- Working environment: Python3
- Requeriments: numpy, pandas, scipy, matplotlib

Base repository: https://github.com/jcapels/si

Setup: https://github.com/jcapels/si#setup

- Create a fork of the repository on your personal GitHub account;
- Clone the repository from your personal account;
- Install the requirements;
- Change the authorship in *__init__.py* (src->si->__init__.py)
- Commit (git commit) and push (git push) the changes.

- Credits: Fernando Cruz, Vítor Pereira and João Correia for the original implementation.

Summary

- Implementation of the **Dataset** class.
- Implementation of Input/Output (IO) functions
- Introduction to the implementation of base classes such as **Transformer**, **Estimator** and **Model**.
- Implementation of feature selectors – VarianceThreshold and SelectKBest

Class Dataset

- In the "*data*" folder, the "*dataset.py*" module was added, which contains the "*Dataset*" class.
- *class Dataset:*
  - Attributes:
    - X – matrix/table of features (independent variables)
    - y – vector of the dependent variable
    - features – vector of feature names
    - label – name of the dependent variable
  - Methods:
    - shape – dimensions of the dataset
    - has_label – checks if the dataset has y
    - get_classes – returns the classes of the dataset (possible values of y)
    - get_mean, get_variance, get_median, get_min, get_max – returns mean, variance, median, minimum, and maximum value for each feature/dependent variable
    - summary – returns a pandas *DataFrame* with all descriptive metrics

io sub-package

- Now, add another sub-package named "io" with two modules called "csv_file.py" and "data_file.py." We will add functions to read and write datasets.
- *def read_csv*
  - arguments:
    - filename – name/path of the file
    - sep – value separator
    - features – boolean. Does the file have feature names?
    - label – boolean. Does the file have y? (If yes, assume it's the last column)
  - expected output:
    - *Dataset object*
    - Reads the specified file and returns a Dataset object.
    - Hint: You can use packages like pandas

io sub-package

- *def write_csv*
  - arguments:
    - filename – name/path of the file
    - dataset – dataset object to write to the file
    - sep – value separator
    - features – boolean. Does the file have feature names?
    - label – boolean. Does the file have y?
  - expected output:
    - Writes the specified file with the provided arguments.
    - Hint: You can use packages like pandas

io sub-package

- *def read_data_file*
  - arguments:
    - filename – name/path of the file
    - sep – value separator
    - label – boolean. Does the file have y? (If yes, assume it's the last column)
  - expected output:
    - *Dataset object*
    - Reads the specified file and returns a Dataset object.
    - Hint: You can use modules from other packages like numpy.genfromtxt

io sub-package

- *def write_data_file*
  - arguments:
    - filename – name/path of the file
    - dataset – dataset object to write to the file
    - sep – value separator
    - label – boolean. Does the file have y?
  - expected output:
    - Writes the specified file with the provided arguments.
    - Hint: You can use modules from other packages like numpy.savetxt

Feature Selection

- Feature selection involves selecting/reducing the number of variables in the dataset.
- In our portfolio, feature selection methods can follow the structure of a **Estimator** and a **Transformer**.
- An **Estimator** is any object that can learn from data. The fit() method allows the estimator to learn patterns from the data.
- A **Transformer** is a specific type of **Estimator** used to modify or transform data. It implements a fit() method, but more importantly, it also provides a transform() and fit_transform() method that applies the learned transformation to the data.

Feature Selection

- **Estimator** Architecture:
  - parameters - a set of user-defined parameters.
  - estimated parameters - a set of parameters/attributes estimated from the data.
  - *_fit* – an abstract method that forces all classes that extend the estimator to implement for estimating parameters from the data
  - *fit* – method that calls *_fit*.

Feature Selection

- **Transformer** Architecture:
  - Extends the **Estimator**
  - *_transform* - an abstract method that forces all classes that extend the **Transformer** to implement responsible for transforming the data.
  - *transform* – method that calls *_transform*
  - *fit_transform* – method that calls both *fit* and *transform*

Class VarianceThreshold

- Create a "*feature_selection*" sub-package, add the "variance_threshold.py" module, which should contain the "VarianceThreshold" class.
- *class VarianceThreshold(Transformer):*
  - parameters:
    - threshold – cutoff line/cut-off value
  - estimated parameters:
    - variance – the variance of each feature
  - methods:
    - _fit – receives a Dataset object and estimates the variance of each feature; returns itself (self) (HINT: use np.var())
    - _transform – receives a Dataset object and selects all features with variance greater than the threshold and returns the transformed Dataset object

statistics sub-package

- Now, let's add another sub-package called "statistics" with a module named "f_classification.py." We will add a function to analyze the variance of our dataset.
- *def f_classification*
  - arguments:
    - dataset – the Dataset object
  - expected output:
    - tuple with F values + tuple with p values
  - algorithm:
    - group the samples by classes. You can use the get_classes() method of the dataset and then select the samples from each class into a list.
    - use the *scipy.stats.f_oneway* function. This function returns the F values and p values.

SelectKBest Class

- In the *feature_selection* sub-package, add the "*select_k_best.py*" module which will contain the *SelectKBest* class.
- *class SelectKBest(Transformer):*
  - parameters:
    - score_func – variance analysis function (we can use *f_classification* by default)
    - k – number of features to select
  - estimated Parameters:
    - F – the F value for each feature estimated by the *score_func*
    - p – the p value for each feature estimated by the *score_func*
  - methods:
    - _fit – estimates the F and p values for each feature using the scoring_func; returns itself (self)
    - _transform – selects the top k features with the highest F value and returns the selected X

Evaluation

- Exercise 1: NumPy array Indexing/Slicing
  - 1.1) In this exercise, we will use the iris dataset. Load the "iris.csv" using the appropriate method for this file type (use the new functions from the package).
  - 1.2) Select the penultimate independent variable. What is the dimension of the resulting array?
  - 1.3) Select the last 10 samples from the iris dataset. What is the mean of the last 10 samples for each independent variable/feature?
  - 1.4) Select all samples from the dataset with values less than or equal to 6 for all independent variables/features. How many samples do you obtain?
  - 1.5) Select all samples with a class/label different from 'Iris-setosa'. How many samples do you obtain?

Evaluation

- Exercise 2: NumPy array Indexing/Slicing
  - 2.1) Add a method to the Dataset class that removes all samples containing at least one null value (NaN). Note that the resulting object should not contain null values in any independent feature/variable. Also, note that you should update the y vector by removing entries associated with the samples to be removed. You should use only NumPy functions. Method name: *dropna*

  - *def dropna*
    - arguments:
      - none
    - expected output:
      - self (modified Dataset object)

Evaluation

- 2.2) Add a method to the Dataset class that replaces all null values with another value or the mean or median of the feature/variable. Note that the resulting object should not contain null values in any independent feature/variable. You should use only NumPy functions.
  Method name: fillna

- *def fillna*
  - arguments:
    - value – float or "mean" or "median"
  - expected output:
    - self (modified Dataset object)

Evaluation

- 2.3) Add a method to the Dataset class that removes a sample by its index. Note that you should also update the y vector by removing the entry associated with the sample to be removed. You should use only NumPy functions.
  Method name: remove_by_index

- *def remove_by_index*
  - arguments:
    - index – integer corresponding to the sample to remove
  - expected output:
    - self (modified Dataset object)

Evaluation

- Optional: You can add examples of how to use these methods to the script/notebook of Exercise 1.

Evaluation

- Exercise 3: Implementing *SelectPercentile*
  - 3.1) Add the *SelectPercentile* object to the *feature_selection* sub-package. You should create a module called "*select_percentile.py*" to implement this object. The *SelectPercentile* class has a similar architecture to the SelectKBest class. Consider the structure presented in the next slide.

  - 3.3) Test the SelectPercentile class in a Jupyter notebook using the "iris.csv" dataset (classification).

SelectPercentile Class

- *class SelectPercentile(Transformer):*
  - parameters:
    - score_func – variance analysis function (*f_classification by default*)
    - percentile – percentile for selecting features
  - estimated parameters:
    - F – the F value for each feature estimated by the score_func
    - p – the p value for each feature estimated by the score_func
  - methods:
    - _fit – estimates the F and p values for each feature using the scoring_func; returns itself (self)
    - _transform – Selects a given percentage of features based on their F-values, ensuring the number of selected features adheres to the specified percentile. The method handles ties at the threshold to maintain the correct number of features. Example: Suppose you have 10 features with F-values [1.2, 3.4, 2.1, 5.6, 4.3, 5.6, 7.8, 6.5, 5.6, 3.2] and set percentile=40. The threshold is calculated as the F-value at the 60th percentile, which is 5.6. Initially, the mask selects features with F-values greater than 5.6, resulting in [7.8, 6.5]. Since we need 4 features (40% of 10), the method identifies ties at the threshold (5.6) and includes the first two of these tied features ([5.6, 5.6]) to meet the requirement. The final selected features are [5.6, 5.6, 7.8, 6.5], ensuring the output adheres to the specified percentile.