WM_home_assignment0_tasks

May 5, 2020

1 Webmining - Assignment 0

This **Home Assignment** is to be submitted and you will be given points for each of the tasks. It familiarizes you with basics of *statistics* and basics of the *sklearn* package as well as the general setup for home assignments. This first home assignment is shorter and also less difficult than upcoming ones.

1.1 Formalities

Submit in a group of 2-3 people until 11.05.2020 23:59CET. The deadline is strict!

1.2 Evaluation and Grading

General advice for programming excercises at *CSSH*: Evaluation of your submission is done semi automatically. Think of it as this notebook being executed once. Afterwards, some test functions are appended to this file and executed respectively.

Therefore: * Submit valid *Python3* code only! * Use external libraries only when specified by task. * Ensure your definitions (functions, classes, methods, variables) follow the specification if given. The concrete signature of e.g. a function usually can be inferred from task description, code skeletons and test cases. * Ensure the notebook does not rely on current notebook or system state! * Use `Kernel --> Restart & Run All` to see if you are using any definitions, variables etc. that are not in scope anymore. * Double check if your code relies on presence of files or directories other than those mentioned in given tasks. Tests run under Linux, hence don't use Windows style paths (`some\path`, `C:\another\path`). Also, use paths only that are relative to and within your working directory (OK: `some/path`, `./some/path`; NOT OK: `/home/alice/python`, `../../python`). * Keep your code idempotent! Running it or parts of it multiple times must not yield different results. Minimize usage of global variables. * Ensure your code / notebook terminates in reasonable time.

There's a story behind each of these points! Don't expect us to fix your stuff!

Regarding the scores, you will get no points for a task if: - your function throws an unexpected error (e.g. takes the wrong number of arguments) - gets stuck in an infinite loop - takes much much longer than expected (e.g. >1s to compute the mean of two numbers) - does not produce the desired output (e.g. returns an descendingly sorted list even though we asked for ascending, returns the mean and the std even though we asked for only the mean, prints and output instead of returning it!) - …

1.2.1 Isolation

Functions that are expected to run in isolation are marked with Section 1.2.1 Warning. For these additionally you are **not** allowed to: - do imports of any kind (also *not* from the python standard library) - use imported stuff (e.g. import numpy somewhere, now use numpy) - call other functions that you have defined (when you write a variance function you are not allowed to call your previously defined mean function) - use other global variables/names Think of these functions as running in a seperate scripts that is not allowed to use any import statements of any kinf

```
[1]: # credentials of all team members (you may add or remove items from the␣
      ↪dictionary)
     team_members = [
         {
             'first_name': 'Pavel',
             'last_name': 'Raschetnov',
             'student_id': 404839
         },
         {
             'first_name': 'Philipp',
             'last_name': 'Stein',
             'student_id': 397615
         },
         {
             'first_name': 'Anya',
             'last_name': 'Poudyal',
             'student_id': 391805
         },
     ]
```

1.3 Task 1

To refresh your knowledge on basic statistics we are going to implement mean, mode, median and standard deviation. All these functions should leave the input argument intact.

[Isolation] Warning: We expect that all functions for this task to work in isolation!

1.3.1 1a) Mean (0.5 points)

Write a function my_mean that takes a list of numeric values and returns the mean.

1.3.2 1b) Std (0.5 points)

Write a function my_std that takes a list of numeric values and returns the standard deviation. Divide by n and not by n-1.

1.3.3 1c) Mode (1.0 points)

Write a function my_mode that takes a list and returns the mode. If there is no unique mode, raise a ValueError.

1.3.4 1d) Median (0.5 points)

Write a function my_median that takes a list of numeric values and returns the median.

[2]:
```python
def my_mean(ar):
    return sum(ar) / len(ar)
```

[3]:
```python
def my_std(ar):
    mean = sum(ar) / len(ar)
    return (sum((x - mean) ** 2 for x in ar) / len(ar)) ** 0.5
```

[4]:
```python
def my_mode(ar):
    counts = {}
    mode, max_cnt = None, 0
    for x in ar:
        cnt = 1 if x not in counts else counts[x] + 1
        counts[x] = cnt
        if cnt > max_cnt:
            mode = x
            max_cnt = cnt
    return mode
```

[5]:
```python
def my_median(ar):
    ar = sorted(ar)
    n = len(ar)
    if n % 2 == 1:
        return ar[n // 2]
    else:
        return (ar[n // 2 - 1] + ar[n // 2]) / 2
```

1.4 Task 2:

In this task we are will explore basic classifiers and the sklearn package. ### 2a) Preprocessing (1 point) Write a function `preprocess`. It takes no input.

It does:

- read the credit_g dataset assume into a pandas dataframe. The file is located in the same folder as the notebook and called `credit-g.csv`
- compute the boolean target vector (True if 'class' is 'good')
- remove the target column from the dataframe
- convert the categorical variables to numeric ones using pd.get_dummies

- perform a (80/20) train/test split using sklearn.model_selection.train_test_split with a seed 123456
- returns the results of the train test split in order

1.4.1 2b) Train linear SVM classifier (0.5 points)

Write a function `train_LinearSVM_classifier` that trains a Linear Support Vector classifier.

It takes two arguments, the first one is the train dataset, the second the target array. It returns the trained classifier. Use the Linear support vector classifier from sklearn with seed of 123456.

1.4.2 2c) Train logistic regression classifier (0.5 points)

Write a function `train_LogisticRegression_classifier` that trains a logistic regression classifier.

It takes two arguments, the first one is the train dataset, the second the target array. It returns the trained classifier. Use the logistic regression classifier from sklearn with seed of 123456.

1.4.3 2d) Evaluate the results (1 point)

Write a function `get_scores` that computes the precision, recall, accuracy and F1 scores. It takes three arguments. The first one is a trained classifier, the second one is the test dataset to evaluate the classifier on, the third is the ground truth target vector. The function returns a dictionary like this:

```
{'accuracy' : accuracy,
 'recall' : recall,
 'precision' : precision,
 'F1' : F1}
```

[Isolation] Warning! We expect this function (2d) to work in isolation

1.4.4 2 e) Bringing it all together (0.25 points each)

Write two functions: `run_SVM` and `run_Log` that use the above functions to train and evaluate a SVM classifier and Logistic regression classifier respectively. It therefor

1. loads the dataset & performs a train test split
2. trains the respectiv classifier
3. returns the scores dictionary

Thereby use the functions `preprocess`, `train_LinearSVM_classifier`, `train_LogisticRegression_classifier`, `get_scores` you defined above.

[6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

[7]:
```
def preprocess():
    df = pd.read_csv('credit-g.csv')
    X,y = df.drop('class', axis=1), df['class'] == 'good'
    X = pd.get_dummies(X)
    return train_test_split(X, y, test_size=0.2, random_state=123456)
```

[8]:
```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
```

[9]:
```
def train_LinearSVM_classifier(train, target):
    clf = LinearSVC(random_state=123456)
    clf.fit(train, target)
    return clf
```

[10]:
```
def get_scores(clf, X_test, target):
    pred = clf.predict(X_test)
    TP = sum(pred & target)
    TN = sum(~pred & ~target)
    FP = sum(pred & ~target)
    FN = sum(~pred & target)
    scores = {}
    scores['accuracy'] = (TP + TN) / (TP + FP + TN + FN)
    recall = TP / (TP + FN)
    precision = TP / (TP + FP)
    scores['recall'] = recall
    scores['precision'] = precision
    scores['F1'] = 2 * precision * recall / (precision + recall)

    return scores
```

[11]:
```
def train_LogisticRegression_classifier(train, target):
    clf = LogisticRegression(random_state=123456)
    clf.fit(train, target)
    return clf
```

[12]:
```
def run_Log():
    X_train, X_test, y_train, y_test = preprocess()
    clf = train_LogisticRegression_classifier(X_train, y_train)
    return get_scores(clf, X_test, y_test)

def run_SVM():
    X_train, X_test, y_train, y_test = preprocess()
    clf = train_LinearSVM_classifier(X_train, y_train)
    return get_scores(clf, X_test, y_test)
```

[13]:
```
run_Log()
```

```
/home/pavel/miniconda3/envs/ml_env/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:938: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
[13]: {'accuracy': 0.775,
       'recall': 0.8970588235294118,
       'precision': 0.7973856209150327,
       'F1': 0.8442906574394464}
```

```
[14]: run_SVM()
```

```
/home/pavel/miniconda3/envs/ml_env/lib/python3.8/site-
packages/sklearn/svm/_base.py:946: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

```
[14]: {'accuracy': 0.68, 'recall': 1.0, 'precision': 0.68, 'F1': 0.8095238095238095}
```