Machine Learning
Programming Assignment II-b

Ujjwal Sharma and dr. Stevan Rudinac

The following assignments will test your understanding of topics covered in the first four weeks of the course. These assignments **will count towards your grade** and should be submitted through Canvas by **30.11.2018 at 12:59 (CET)**. You can choose to work individually or in pairs. You can get at most 4 points for these assignments, which is 4% of your final grade.

Submission

You can submit either a Jupyter Notebook (*.ipynb) or a python program file (*.py). To test the code we will use Anaconda Python 3.6. Please state the names and student ids of the authors (at most two) at the top of the submitted file.

Please make sure to submit only a single notebook or source file for homeworks II-a and II-b. You can use comments to indicate the problem number.

1 Data

For this week, we go back to the data from Assignment I which is available here. You can reuse any utilities *you wrote* to read or preprocess that data. In the zip file, you will find:

- A training data file titled `webStats_train.csv`.
- Other files in the zip package (with file-name in the form `webStats_test-*.csv`) contain test data.

The supplied training data consists of 280 *features* and a single positive integer-valued label denoting the number of likes for the page. The last column of the training data is the label and should be extracted before training. The test data has the same structure. You will find the `pandas` library extremely helpful in working with this data.

In the first week, there were multiple convergence issues, mainly with SVM classifiers, due to the bad scaling of data. In this homework, we will use newly-gained knowledge about data pre-processing and complex classifiers to mitigate those issues while attempting to build better-performing models. We will also attempt to understand the effects of missing data on estimator effectiveness (accuracy score) and experiment with measures that can be taken to mitigate these unwanted effects.

In this assignment, we limit ourselves to the classification task from Assignment I, albeit with added issues.

2 Classification

The likes on an article are often indicative of its popularity and by proxy user-engagement. Imagine that the domain expert decided to divide the data into three categories based on the number of likes:

✉ u.sharma@uva.nl, s.rudinac@uva.nl

Popularity	Likes
Not-Popular	0
Somewhat-Popular	1
Very-Popular	≥ 2

Table 1: Likes to Popularity Labels

Before starting with the classification task, you will need to convert the numerical like data (i.e. target variable) into class labels.

Additionally, you are required to use the `pipeline` estimator functionality from sklearn (like in Homework II-a) for all your models.

2.1 Data Preprocessing

Data in this example problem has large variations in scale and could pose issues for the classifier. To mitigate this, we can add a data scaling step before feading it to the classifier. This can be easily done using a `Pipeline`.

For this task, experiment with the `StandardScaler`, `MinMaxScaler` and the `MaxAbsScaler` to ascertain which of these works best for this data.

2.2 Classification

In this section, we will focus on classifying articles into the above-mentioned three popularity categories based on their features. For this task, you will implement the following classifiers:

1. A k-nearest neighbor (k-NN) model. The model is available from `sklearn.neighbors`
2. A linear SVM model with an optimal C selected using `GridSearchCV`. Available from `sklearn.svm`
3. A Random Forests classifier. The classifier is available from `sklearn.ensemble`

TASK : Fit all three models to the data. Once completed, report the *accuracy score* averaged over all test data blocks.

Your final pipeline should contain the following steps:

INPUT → SCALING → CLASSIFICATION

TASK : For all the pipelines, report your (averaged) accuracy scores. Additionally, in no more than 100 words, present your broad observations on the performance of these classification models.

2.3 Missing features and Imputation

Missing data can have strong effects on the ability to train the aforementioned models. These effects can be mitigated to a certain extent by data imputation where missing data are replaced by substitute values. In order to simulate these effects, we can randomly set input data to `NaN`.

To do this end you can use the function below:

```
def add_nan_to_data(train_inputs, miss_prob=0.2):
    """
    Randomly flips a numpy ndarray entry to NaN with a supplied
    probability.

    WARNING: Do not try to add missing values to labels. This is
```

```
    not unsupervised learning.

        :param train_inputs: Numpy ndarray of dims (num_examples, feature_dims)
        :param miss_prob=0.2: Probability that a bit is flipped to NaN.
    """
    mask = np.random.choice(2, size=train_inputs.shape, p=[miss_prob,1-miss_prob]).astype(bool)
    input_shape = train_inputs.shape

    #Flatten Inputs and Mask
    train_inputs = train_inputs.ravel()
    mask = mask.ravel()
    train_inputs[~mask] = np.nan

    # reshape inputs back to the original shape.
    missing_train_inputs = np.reshape(train_inputs, newshape=input_shape)

    return missing_train_inputs
```

You can copy this function from here.

Once you have the data with missing entries, you can experiment with data imputation. In this homework, we will limit ourselves to the `SimpleImputer` function from `sklearn.impute` that replaces a missing value by the mean, median or mode of the given feature. In general, more sophisticated imputation methods, such as k-NN, may yield better performance.

TASK : Repeat your experiments from the previous section, with an additional Imputation module. Your pipeline should look like:

DATA (MISSING INPUTS) → IMPUTATION → SCALING → CLASSIFICATION

TASK : For all pipelines, report your averaged accuracy scores. Additionally, in no more than 100 words, present your broad observations on the effects of missing data entries on the performance of the classification models.