CIS6060/Bioinformatics : W21 : Assignment 2 : Simple Pipeline

This is an assignment I gave out while teaching the CIS 6060 course in Winter 2021 at the University of Guelph School of Computer Science. It will give you an overview of how to set up a simple machine learning data analysis pipeline and provide you a starting place for similar work.

License

Introduction

This assignment will give you a taste of running a machine-learning pipeline using scikit-learn, numpy, and pandas in the Python 3 environment.

To be sure that you have the required libraries in your Python environment, you will want to be sure that you have them installed. Here are the install lines to use if you are using pip

```
python3 -m pip install pandas
python3 -m pip install numpy
python3 -m pip install scikit-learn
python3 -m pip install matplotlib
python3 -m pip install seaborn
```

Tasks

For this assignment, we wish to do three things:

1) we want to extend out machine-learning pipeline possibilities with another data set
2) we want to extend a “support vector machine” machine-learning pipeline to also attempt the use of “logistic regression”
3) based on the logistic regression results, we wish to analyze the performance on each data set

As in the last assignment, the main deliverable will be report based on your findings. See details on what should be in the report below

Data Sets

For the machine-learning tools in `scikit-learn` the input data needs to be formatting in a tabular organization, where each row is a "sample" that includes a "label" to be learned.

Consider this table for data for the relationship "logical AND", where the "`Result`" is 1 (true) if both of the inputs "`A`" and "`B`" are true:

```
A,B,Result
0,0,0
0,1,0
1,0,0
1,1,1
```

If we load this table from a `.csv` file using `pandas`, we can extract the "label" column "`Result`" and use the other two "feature data" columns to describe the rows capturing the relationships in the observed data.

Typically, we may refer to this division as the matrix X of input features, and the vector y of labels.

For the scikit-learn algorithms, all the values in both the matrix X and the vector y need to be numbers. For y it is probably best if these are integers.

Any data that you choose for your own problem of interest will need to be converted into this form. Take a look at the provided data files to see examples of this format.

Abalone data set

The Abalone data set in `src-data-abalone-UCI` provides a relatively small example. In this problem X gives measures of a shellfish from Tasmania where the objective is to distinguish the juveniles (class "I") from the adults (class "M" and "F") – these values I/M/F values have been mapped to 0/1/2 in the y label column, which in this problem is called "Sex".

In the Abalone data set, the X matrix has 4177 samples, and 7 columns.

There is a script called `convert_data_to_standard_tabular_format.py` in the directory `src-data-abalone-UCI` that with very small adaptation can be used to convert any of the UCI Machine Learning Repository datasets to this standard form.

Cancer by Gene Expression

This data set in `src-data-cancer-by-gene-expression` provides a table where X contains data for 7129 gene expressions (the columns of the table) as collected from 72 patients. The y label values from this problem come from the column called "cancer".

The fact that the table is very wide but quite short means that we will have to do "dimensionality reduction" on this data – see PCA below.

Tools

There are tools provided for you to run an analysis of both the *Abalone* and *Cancer by Gene Expression* data sets, and a simple training/testing split of the data in *Cancer by Gene Expression*

These are provided in files for "train-vs-test" runs and "cross-validation":

- train -vs- test runs:
  - `tools/runpipe_abalone_test_train.sh` : This is the simplest pipeline. It divides the data provided in the data set up into a portion for testing and separate portions for training and "validation".

    Dividing the data up in this way is important, as if the same values are presented to the learning system to "test" as were used to "train" then it is much the same situation as giving exactly the same problems to students to learn a topic and evaluate their learning – we cannot tell the difference between students who "actually learned" the material and those who simply memorized the answers. Machine learning systems will defintely "memorize the answers" if they can.

    We divide into three portions as we want to figure out what the best tuning parameters are for the learning system, and we have the same need for indepenent sets of problem data for tuning (*a.k.a.* validation data) and for the actual training data to learn the problem.

    This pipeline therefore divides the data up as follows:

    - 25% for testing, and the remaining 75% divided into
      - 50% for validation (so 37.5% of the total)
      - 50% for training (so 37.5% of the total)

    Training the system on this gives us a single estimate of how well the system was able to learn the problem.
  - `tools/runpipe_cancer_test_train.sh` : This pipeline performs the same "train -vs- test" analysis as the previous example, but on the *Cancer* data. As the *Cancer* data has so few rows relative to the number of columns, PCA is used as a dimensionality reduction technique.

    Note that because there are few rows in this problem, it runs *considerably faster* than the *Abalone* problem.

  The result of running these pipelines will be placed in directories named `abalone-train-vs-test` and `cancer-train-vs-test`.
- cross validation runs:. The tools below run a full 5-fold cross validation analysis on the two data sets in order to provide 5 estimates for each evaluation. This will allow

an estimate of the variability in the performance to be calculated, and also a paired t-test to be used to compare performances between algorithms (more below)

- `tools/runpipe_abalone_5_fold.sh` : run a cross validation pipeline on the *Cancer* data
- `tools/runpipe_cancer_5_fold.sh` : run a cross validation pipeline on the *Abalone* data

The result of running these pipelines will be placed in directories named `abalone-folded-*` and `cancer-folded-*`.

Tools used in pipelines

Each of these calls several python3 scripts to do the work. These are:

- `tools/create_test_train_split.py` and `tools/create_folds.py` : These scripts divide up the rows of the provided X and y values into independent sets of samples, ensuring that the resulting sets are "stratified" according to the labels in y.
- `tools/calculate_data_projection.py` : this takes the data for a particular analysis as produced by one of the previous scripts, and projects it into a standardized space. There are two steps to this:
  - individually scale the values in each of the columns in X so that the have zero mean and unit standard deviation based on the portion set aside for training. (This removes unit problems, and ensures that the learning system treats each column as of equal potential importance.)
  - (optionally) projects the data to a lower-dimensional space using Principal Components Analysis. While making the data more difficult to interpret, it has several advantages:
    - reduces the number of dimensions, allowing tables that are wider than they are tall to be learned (as in the *Cancer* problem)
    - reduces the number of dimensions, reducing the training time
    - allows two-dimensional plots to be made of the two "most important" dimensions
- `tools/evaluate_svn.py` : this script trains and provides analysis of the ability to learn a data problem using the popular "Support Vector Machine Classifier" algorithm. While most people refer to this as an "SVM", scikit-learn calls this an "SVC". It is the same thing.

To add another algorithm, I suggest copying `tools/evaluate_svn.py` as a starting place, and then adding your new script to each pipeline.

Note that the Logistic Regression tool in scikit-learn documentation is available online.

Analysis and Report

Once you have run the evaluations, and in particular the 5 fold cross validation evaluations, you will have multiple estimates of the performance of the various classifiers on the data sets.

In your report, you should provide an analysis that lets you approach the following questions:

- What is the mean and standard deviation of the key performance metrics for each classifier for each problem:
  - accuracy
  - precision
  - recall
  - F-measure
- Does one classifier perform significantly better than another in terms of accuracy for a given problem?

  Approach this by calculating a *paired t-test* of the performance measures for a pair of classfiers. As *exactly the same decisions* are being made for data in the same fold by different classfiers, these values can be paired together in this type of test.
- What do the best parameters chosen in validation based grid search tell us about these problems?

Write your analysis up in a short report to explain what you have learned.