Practical session 2

I don't like bugs, okay? They freak me out! **- Raj in The Big Bang Theory**

Development Tools for Scientific Computing 2024/2025

Pasquale Claudio Africa, Dario Coscia

18 Feb 2025

Part 1: Background on Classifiers

A binary classifier is a type of machine learning model that categorizes data into one of two distinct classes, and it is one of the most basic and commonly used tasks in scientific applications. For example, in a spam detection system, the binary classifier classifies emails as either spam (1) or not spam (0).

The menu of today includes:

1. Building a k-Nearest Neighbors (k-NN) classifier from scratch.
2. Setting up a collaborative and reproducible development environment on GitHub.
3. Creating and run tests to ensure that our k-NN classifier works correctly and meets quality standards.

We will use the Ionosphere dataset which contains features obtained from radar signals focused on the ionosphere layer of the Earth's atmosphere. The task is to determine whether the signal shows the presence of some object, or just empty air.

Notes on k-NN Algorithm

We are given a set of data $\mathcal{D} = \{(X_1, y_1), \ldots, (X_n, y_n)\}$, where each data point $(X, y) \in \mathbb{R}^d \times \{0, 1\}$. The variable X indicates the *features* which we use to classify (e.g. pixel values, covariates, ...), while y indicates the class label (e.g. cats or dogs). Our objective is to find the class y^* of a given new point $x = X^*$.

Simple (expensive) Algorithm

1. *Calculate Distance*: Measure the distance between the input point and all other points in the dataset.
2. *Find Nearest Neighbors*: Identify the k nearest points based on the calculated distances.
3. *Classify*: Assign the most common class label among the k nearest neighbors to the input point.

Part 2: Set up repo and Python functions

Given the `devtools_scicomp_project_2025` of the first Pratical Lecture from the `main` branch, modify the README file and add your name, email address and the course you are enrolled in. Add the change, amend the last commit and push.
Create a new branch titled `knn_classifier` from the `main branch`. This is the branch we will work on. **Do not use python libraries for today session, but only built in data types.**

1. **Set up the code**:

- Activate the conda environment `devtools_scicomp` and install `PyYAML`. Add it to `requirements.txt` file.
- Inside `src/pyclassify/utils.py` file implement a function called `distance`. This function should:
    - Take two inputs: `point1` and `point2`, both of type `list[float]`.
    - Return the square of the Euclidean distance between `point1` and `point2`. You can refer to the Euclidean distance formula for this or write some test cases.

- Inside `src/pyclassify/utils.py` file implement a function called `majority_vote`. This function should:
  - Take one input: `neighbors`, which is a `list[int]` of class labels.
  - Return the majority class among the neighbors.
- Inside the `src/pyclassify/` directory, create a file called `classifier.py`. Inside it, create a Python class named `kNN`. The `kNN` class should:
  - Be initialized with an integer variable `k`, which specifies the number of nearest neighbors.
  - Contain a method called `_get_k_nearest_neighbors`, which takes `X`, `y` (the dataset values), and `x` (a new point to be classified). This method should return a list of `y` values (labels) of the `k` nearest neighbors of `x`.

- Inside the class `kNN` override the `__call__` method. It takes two inputs: `data`, a tuple containing `X` (the feature matrix) and `y` (the labels); `new_points`, a list of new points to be classified. It should return a list of predicted classes for all points in `new_points`. The main algorithm is as follows: for each point in `new_points`:
    - Call the `_get_k_nearest_neighbors` method to get the neighbors.
    - Perform majority voting using the `majority_vote` function from `utils.py`.
    - Store the predicted class for each point.
- Finally, inside the `src/pyclassify/__init__.py` file import `kNN`

```python
__all__ = [
    'kNN'
    ]

from .classifier import kNN
```

2. **Set up the tests**:

   - Inside `test/test_.py` file implement functions called `test_distance`, `test_majority_vote`. The `test_distance` function should test the distance properties, while `test_majority_vote` should test that the algorithm implemented is correct (for example given `[1, 0, 0, 0]` the algorithm should return `0`).
   - Inside `test/test_.py` check the constructor of `kNN` (valid types).

3. **Set up experiments**:

   - Inside `src/pyclassify/utils.py` add the following function use to read `yaml` files:

```python
def read_config(file):
    filepath = os.path.abspath(f'{file}.yaml')
    with open(filepath, 'r') as stream:
        kwargs = yaml.safe_load(stream)
    return kwargs
```

```
- Inside `shell/submit.sh` file write the following line of code, which downloads the [Ionosphere](https://archive.ics.uci.edu/dataset/52/ionosphere) dataset and put it a directory called `./data`. Explore the dataset and in `src/pyclassify/utils.py` create a function named `read_file` which reads the dataset file and returns the features and labels as separate lists.
```

```bash
URL="https://archive.ics.uci.edu/static/public/52/ionosphere.zip"
DEST_DIR="data"
ZIP_FILE="ionosphere.zip"
echo "Downloading ionosphere.zip from $URL..."
curl -o $ZIP_FILE $URL
# Step 2: Create the 'data' directory if it doesn't exist
if [ ! -d "$DEST_DIR" ]; then
   echo "Creating directory: $DEST_DIR"
   mkdir $DEST_DIR
fi
echo "Extracting $ZIP_FILE..."
unzip $ZIP_FILE
if [ -f "ionosphere.data" ]; then
   echo "Moving ionosphere.data to $DEST_DIR"
   mv ionosphere.data $DEST_DIR/
else
   exit 1
fi
echo "Cleaning up: Removing $ZIP_FILE"
rm $ZIP_FILE
echo "Download and extraction completed successfully."
rm Index ionosphere.names
```

- Inside `experiments/config.yaml` insert the following:

```
k: 5
dataset: ./data/ionosphere.data
```

- Inside `scripts/run.py` import `kNN` and `read_config` (remember the package is called `pyclassify`). The run file needs to:
  i. Read the parameters in the `config.yaml` file
  ii. Divide with a $80-20$ percent split the dataset (no need to shuffle it is a plus) into test and train respectively
  iii. Perform a `kNN` classification on test data and print compute accuracy.
  iv. Add, commit and push the changes using the commit message "practical2".

Note

Remember to install the package before running the `run.py`!

Solutions

The repository with the right structure and commits is reported here: GitHub repo