Lab 11, Classification and Clustering

Due date: Friday, May 29, 11:59pm.

Overview

For this lab you will work on two problems.

Classification. You will finish the implementation of the K-nearest neighbors (KNN) classification method and will use it to classify data from a number of datasets and find the best k for each dataset.

Clustering. You will finish the implementation of the K-means clustering algorithm and will use it to cluster data from a number of datasets and find the best k (number of clusters) for each dataset.

Assignment Preparation

This is an individual assignment.

For this assignment you will build a collection of Python functions, and a number of notebooks (at least one per question studied). Everything you build shall be from scratch, i.e., no notebooks are provided to you. You will be able to use the instructor's notebooks that step through KNN and K-Means clustering algorithms, and you are welcome to reuse any code in those notebooks. You cannot "borrow" code from anywhere else though.

See below for a more detailed list of deliverables.

Running code in Jupyter. This assignment asks you to prepare in Python files and import it into Jupyter notebooks. Unfortunately, Python's `import` statement does not "pick up" Python files from Jupyter's working directory.

However, Jupyter has a `%run` directive that fixes this issue. To run a Python file `myfile.py` located in the working directory of your Jupyter notebook, include the following command in the notebook:

```
%run myfile.py
```

You can look up the documentation on the `%run` directive at

> https://ipython.org/ipython-doc/3/interactive/magics.html#magic-run

Lab Assignment

You have to complete the instructor's implementations of two algorithms, the K-Nearest Neighbors classification algorithm and the K-Means clustering algorithm. You will then develop appropriate studies to determine how well these algorithms can analyze the data from two datasets.

Classification

You have to complete the following tasks.

Step 1. Complete the K-Nearest Neighbors classification algorithm. You can use the code from the instructor's KNN notebook as a starting point, or you can build your code from scratch. In either case, create a Python file `knn.py` containing all functions you need for KNN to run.

Recall that KNN classifier does not have a *fit* stage - it is a lazy evaluation classifier that only uses the classification step. Represent the classification step of the KNN classifier using the following Python function:

```
def knn(data, point, k)
```

Here, `data` is a (NumPy) array of data points (points in rows, features in columns), `point` is the data point to be classified, and `k` is the parameter of the method: the number of nearest neighbors to consider. If your `data` array contains no class labels, you can change the function to

```
def knn(data, labels, point, k)
```

where `labels` is an array of class labels for the data points in `data`.

Distance Functions. You shall implement at least the Eucledian distance function for distance between two points. If you believe a better distance function (or a similarity function - you can use those as well to find nearest neighbors) exists - implement it as well and test your KNN method with different distance/similarity functions.

Step 2. Prepare the code for evaluating the accuracy of your KNN implementation. Develop an implementation of M-fold cross validation for KNN. You will use this implementation at least twice (once for each of the datasets you study).

While, in general, it is OK to submit two notebooks with the same basic code copied from one to another, you may want to have only one version of the code, place it in your `knn.py` file and use it everywhere.

The cross-validation procedure shall posses the following properties:

- Input. The procedure shall be provided the dataset (already placed in a NumPy array, to separate the data acquisition from the evaluation) which is to be classified. Together with the dataset, shall come an array of class labels. It can either be a separate array, or the last column in the array of data points - select how you want to pass this information and stick to it.

  The second input is M: the number of folds in the cross-validation process.

- Folds. There are different ways to split the data into M folds. You are allowed to use a simple procedure and split deterministically with the first N/M data points (assuming the dataset of size N) in fold 1, the next N/M data points in fold 2, and so on. If you want to, you can implement a more intricate and less biased procedure for creating folds, but it is not required in this case.

  (The simplest way to do this is to carefully shuffle the data in place once, and then split the array linearly into folds. Make sure data labels get shuffled exactly the same way).

- All-but-one. If the parameter M provided as input to your processs is equal to the size of the dataset, your cross-validation code shall proceed with *all-but-one* validation.

- Output. For classification that involves more than two classes, report the overall classification accuracy and compute and report the confusion matrix.

  If the classifier is binary, compute precision, recall and $F-measure$ as well as overall accuracy and the confusion matrix (for binary classifiers, indicate to your cross-validation procedure which class you consider positive).

Step 3. Analysis and Visualization. For each dataset, create a notebook that analyzes the accuracy of the KNN method, allows you to properly select the best K (number of nearest neighbors to compare the point with), and visualizes the classified data.

The goal of the notebook is to answer two key questions:

1. What is the accuracy of KNN on the given dataset?

2. What is the best value of K?

You also need to present the results and visualize the data to the best of your ability. The data may come to you with a number of dimensions that is greater then 2 or 3. It is your job to find a good way to visualize the dataset using `MatPlotLib`.

Datasets. You will run KNN on two datasets:

1. **The Iris Dataset.** It is available at

   `/home/dekhtyar/data/iris.csv`

   Recall that a single nearest neighbor already properly classifies all but 6 irises. Can you do better by increasing the number of neighbors?

2. **User Knowledge dataset.** This is a five-dimensional dataset from the UCI Machine learning repository. The original dataset is located at

   https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling

   A CSV version of the dataset containing only the data (and the line with the column names) is available on the Jupyter server at

   `/home/dekhtyar/UserKnowledge.csv`

The dataset comes from a study of user knowledge based on five features[1]:

- STG: The degree of study time for goal object materails
- SCG: The degree of repetition number of user for goal object materails
- STR: The degree of study time of user for related objects with goal object
- LPR: The exam performance of user for related objects with goal object
- PEG: The exam performance of user for goal objects

The class variable, UNS: the knowledge level of the user, has four different values: Very Low, Low, Medium and High.

All five features are measured as real numbers between 0 and 1.

Clustering

For the clustering part of your assignment perform the following steps.

[1] https://archive.ics.uci.edu/ml/datasets/User+Knowledge+Modeling

Step 1. Complete K-Means Clustering implementation. Complete the implementation of the K-Means clustering algorithm. You can use the instructor's code as a starting point, or you can build your own code. Either way, create a Python file `clustering.py` and implement the K-means clustering algorithm as the following Python function:

```
def kmeans(data, k)
```

where `data` is the array of data points and `k` is the number of clusters.

If you want to have a bit more fine tuned control, you can implement this function as

```
def kmeans(data, k, distance, stoppage)
```

where `distance` is the name of the distance function (shall be implemented in `clustering.py`) to use, and `stoppage` is the flag controlling your stoppage condition - set up as you see fit.

The function shall return an array of cluster assignments to the input data points

Datasets. You will use the same two datasets, **Iris** and **User Knowledge** to search for clusters in the data. Note that while the class labels are available to you and can be used to validate your clustering performance, their presence does not mean that the dataset contains that specific number of clusters. In fact, it is quite possible that some classes in the datasets consist of multiple clusters (e.g., subjects achieve high knowledge through distinctly different approaches to learning). Because of this, investigate the carefully, how many clusters the datasets contain.

Step 2. Evaluation of clustering alogithms. You will develop code to evaluate the performance of the clustering algorithms on the data you received. Your evaluation will compute two types of metrics for a given clustering: *cluster purity* and *cluster compactness.*

The **cluster purity** metrics are possible because both datasets come with class labels. A cluster is considered to be 100% pure if all its data points have the same label[2]. In general, the purity of a cluster is the *percent of data points in the cluster that carry the plurality label.* The overall *purity of a clustering* is the percentage of pure data points in the cluster, i.e., the percentage of points that carry plurality labels in their respective clusters.

Purity can be measured in a few ways (the above definition is just one), but essentially, cluster/clustering purity scores evaluate accuracy.

The **cluster compactness** measures look at the size of the cluster and the relationships between intercluster distances and intracluster distances.

[2]Notice that a pure cluster does not have to equal to the entire class – two or more clusters can split all data points in a single class.

You can implement any of the cluster compactness measures discussed in class, just document properly what you are computing.

Step 3. Clustering experiments. For each of the datasets, prepare a Jupyter notebook that analyzes the use of K-means clustering algorithm to cluster the data. For each dataset, the key two questions to answer are:

- How good is K-means clustering in clustering the data from the dataset?
- What is the "right" number of clusters in the dataset, and why?

Your notebook shall conduct a grid search for a good value for k – the nubmer of clusters. It shall output the results of evaluating each run of K-means clustering, and attempt to visualize them to the best of your ability.

At the end, your notebook shall contain your analysis of the overall accuracy, and the selection of k and your reasons for it.

Deliverables and submission

You shall build two notebooks, `KNN.ipynb` and `KMEANS.ipynb` and two Python files, `knn.py` and `clustering.py` containing your implementations. Submit these four files together with any other files you need.

You can download the instructor's notebooks by running the

```
$ nbgrader fetch --course=dekhtyar Lab11
```

command. The instructor's notebooks are the copies of the KNN and K-means notebooks shown in class, packaged into the `Lab11` directory for your convenience.

Submit your lab using `nbgrader`:

```
$ nbgrader submit --course=dekhtyar Lab11
```

References

[1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2), 133-151. July 2001.

[2] Ken Goldberg, Anonymous Ratings Data from the Jester Online Joke Recommender System, http://www.ieor.berkeley.edu/~goldberg/jester-data/.