Machine Learning Methods
Exercise 2

Last updated: 18.1.24

1 General Description

In this exercise, you will explore the practical application of two fundamental machine learning concepts: Distance-Based Methods and Decision Trees. We will use a dataset of 2D map locations of cities in the USA, which consists of 30K data samples. Using different methods, you will examine how cities are classified into states and identify anomalous cities.

You are expected to hand in a report, no longer than 8 pages, describing your experiments and answers to the questions in the exercise. The submission guidelines are described in Sec. 8, please read them carefully.

Note 1: You **must** explain your results in each question! Unless specified otherwise, an answer without explanation will **not** be awarded marks.
Note 2: When you are asked to plot something, we expect that plot to appear in the report.

2 Seeding

As in the previous exercise, we would like to reproduce your experiments. This can be done by random seeding. Specifically, all that it takes is to put the following line of code at the beginning of your main file: *np.random.seed(0)* (where *np* stands for the *numpy* library).

3 Provided Code Helpers

You are free to choose how to implement this exercise. We do provide you with some helpful helper functions. These helpers include visualizations, data processing and demos for working with specific models. They can be found in `helpers.py`. These functions are meant to help you and we recommend you to use them. Saying that, they are optional, meaning **if you find them confusing you may simply ignore them**.

We also provide you with a skeleton code for the kNN clasifier, found in `knn.py`. Unlike the prevoius helpers, this skeleton is mandatory and you must fill-in the blanks for Sec. 5.

4 Data

The dataset used for our exploration consists of three distinct files: *train.csv, validation.csv, test.csv*. Each file is structured as a table, with three columns: longitude (`long`), latitude (`lat`) and `state`. The samples in our dataset are represented as 2D coordinates of cities in the USA, with the corresponding labels indicating the state. For instance, if we consider a 2D city coordinate such as $(41.8375, -87.6866)$ – representative of Chicago – its label would be *'IL'* (short for *'Illinois'*). To make it easier, we already encoded the `state` column into integers.

The samples in our dataset can be denoted as $\mathcal{X} \in \mathbb{R}^{N \times d}$, where N represents the number of samples, and d is the representation dimension. In our specific case, $d = 2$ as we are dealing with longitude and latitude coordinates.

We will use this spatial dataset for exploring distance-based methods and decision trees, by classifying cities into states based on their geographical coordinates.

5 Classification with k-Nearest Neighbors (kNN)

kNN embodies the principle that similar instances tend to belong to the same class. However, the choice of the hyperparameter k, representing the number of neighbors considered, can significantly influence classification outcomes and requires careful consideration.
To facilitate your implementation of supervised kNN classification, we offer a pseudo-code template in Algorithm. 1. This pseudo-code outlines the key steps involved in the kNN algorithm, making it a valuable reference for your implementation.
Moreover, we recommend exploring the `faiss` library, which offers an efficient kNN implementation. The library is particularly useful when dealing with large datasets or scenarios where computational efficiency is critical. Additionally, you can find an example of how to extract kNN distances for a given test sample in the provided code helper. To install `faiss` make sure you already have the `numpy` library installed, then simply run:
`pip install faiss-cpu`.

5.1 Task

Implement the `KNNClassifier` class given in the `knn.py` file. The required inputs and outputs are given in the code. The kNN search should be done using `faiss` (in a similar way to the code you saw in class).

Algorithm 1: kNN Classification

Data: Training dataset $\mathcal{X}_{train} = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^2$ is the feature vector and y_i corresponds to its class label; Test instance $\hat{x}_{\text{test}}$; Number of neighbors k.

Result: Predicted class label $\hat{y}_{\text{test}}$ for the test instance.

for *each training instance* (x_i, y_i) **do**

  Calculate the distance d_i between x_i and x_{test}.

Sort the distances d_i in ascending order and select the first k instances.

  for *each class* c **do**

    Count the occurrences of c in the selected k instances.

Assign the class label $\hat{y}_{\text{test}}$ to the one with the highest count.

return $\hat{y}_{test}$

1. Your code should allow for two distance metrics, L_1 and L_2, it should also allow the user to specify the number of neighbors k.

2. Use *train.csv* as training data *test.csv* as the testing data.

3. Now, try every combination for $k \in \{1, 10, 100, 1000, 3000\}$ and distance metrics in $\{L_1, L_2\}$ (feel free to use nested loops). Present the results in a table of dimensions 5×2, containing the test accuracy scores for each combination of k and distance metric. Report this table in your PDF.

You should save all 10 kNN models and accuracies. For each model you should also save its hyper-parameters (distance metric, k) and its test accuracy. We will later choose between the models, according to these values.

5.2 Questions

1. Look at the 5×2 table of results from task 3. What is the trend you see when the number of k increases? Does it changes between different distance metrics?

2. We will now visualize the differences between k values and distance metrics:

   - Choose the k value with the highest test accuracy when using the L_2 distance metric. We will call it k_{max}.
   - Choose the k value with the lowest test accuracy when using the L_2 distance metric. We will call it k_{min}.
   - Using the given visualization helper (`plot_decision_boundaries`), plot the test data and color the space according to the prediction of the following 3 models, each in a separate plot (overall 3 plots): (i) distance_metric $= L_2$, $k = k_{max}$. (ii) distance_metric $= L_2$, $k = k_{min}$. (iii) distance_metric $= L_1$, $k = k_{max}$.

NOTE: Running `plot_decision_boundaries` may take up to a few minutes when using high k values (e.g. 1000, 3000). If it takes substantially more (15 minutes or more) your `predict` implementation is probably not efficient enough.

(a) Look at the plots of the (i) k_{max} with L_2 and (ii) k_{min} with L_2. What is different between the way each one divides the space? Why does k_{max} results in better accuracy? Explain.

(b) Look at the plots of the k_{max} with L_2 distance metric and k_{max} with L_1 distance metric. How does the choice of distance metric affect the classification space? Explain.

5.3 Anomaly Detection Using kNN

In this phase, our exploration turns towards anomaly detection, utilizing the kNN algorithm to identify anomalies. For this specific task, we deviate from supervised classification. Instead, we treat the training set as a single class, ignoring individual class labels. The primary objective is to identify anomalies within the test set by calculating kNN distances.
More specifically:

- You were given an additional test file specifically for this task, named: *AD_test.csv*. Your train set remains unchanged.
- Find the 5 nearest neighbors from the train set for each test sample of *AD_test.csv* using `faiss`. Use the L_2 distance metric. Save the distances to the neighbors as well.
- Sum the 5 distances to the nearest neighbors for each test sample. We will refer to these summations as the anomaly scores.
- Find the 50 test examples with the highest anomaly scores. We will define these points as anomalies, while the rest of the points will be defined as normal.
- Using the `matplotlib` library, plot the points of *AD_test.csv*. According to your prediction, color the normal points in blue, and the anomalous points in red. Additionally, in the same plot, include the data points from *train.csv* colored in black and with an opacity of 0.01 (the parameter controlling the opacity is named `alpha`). You might find the `plt.scatter` function useful for this visualization.

5.4 Questions

1. What can you tell about the anomalies your model found? How are they different from the normal data? Explain.

6 Decision Trees

In this part of the exercise you will be classifying the points to states using decision trees. The main purpose of the part is for you to understand the axis aligned nature of decision trees, and visualize their decision space. **NOTE:** Q8 is a bonus question, you do NOT have to do it. As it is very short, and does not require any explanations, we highly recommend you to do it.

We will use the `sklearn` library to implement our decision trees. To make things clearer, we provided you with an example (`decision_tree_demo`) for training and predicting with a decision tree using `sklearn`.

6.1 Task

Train 24 decision trees using the provided training data, classifying the long.-lat. points into states. The 24 trees are all combinations for the following values for these two hyper-parameters:

- Maximal depth (named `max_depth`): (1, 2, 4, 6, 10, 20, 50, 100)
- Maximal leaf nodes(named `max_leaf_nodes`): (50, 100, 1000)

You should save all 24 tree models. For each tree you should also save its hyper-parameters (`max_depth`, `max_leaf_nodes`) and all 3 accuracies (training, validation and test). We will later choose between the trees, according to these values.

6.2 Questions

1. Choose the tree with the best validation accuracy. What is its test accuracy? Report it.
2. Report the training accuracy, and test accuracy of the tree from Q1. Is this tree successful in generalizing from its training examples? Look at the test accuracies of the other trees. Is our validation set sufficient to choose the best tree? Explain your answers.
3. **Are 50 nodes enough?** There are 50 states in the USA, and a leaf node predicts a single state each time. Are 50 leaf nodes enough to achieve perfect accuracy?
   - If so explain what properties of the data make this possible.
   - If not explain why more leaf nodes are needed.
4. **How trees see the world.** Visualize the data, coloring the map with the predicted class colors, according to the tree you chose in Q1. Use the given helpers for this visualization. What is the shape it creates for each class?

5. **Restricted leaves.** Choose the tree with the best validation accuracy that has only 50 leaf nodes. Visualize the data as before with this tree predictions. How does the prediction of the 50 compare to the overall best from Q1? Explain.

6. **Restricted depth.** Choose the tree with the best validation accuracy that has a maximal depth of at most 6 (i.e. `max_depth` ≤ 6). Visualize the data as Q4 with this tree predictions. What has changed in the way the space is divided compared to Q4? Explain.

7. **Random Forest.** Random forest are a simple extension to decision trees. Specifically, random forests train many small decision tree models on different subsets of the data and take their majority vote as the prediction. To our convenience, this is also implemented in the `sklearn` library. In the helpers file you can find an example for loading a random forest model, and its interface is exactly the same as the decision trees.

   Train a random forest of trees, using 300 trees with a maximal depth of 6. Visualize the data and predictions as Q1. Is this model more expressive than the one from Q1? How this visualization helped reach that conclusion? Explain.

8. **Experimenting with XGBoost (Bonus 5 pts).**

   - **A word on Boosting.** While random forests are useful and powerful models, it is usually more beneficial to use boosting methods and more specifically gradient boosting methods such as XGBoost [1]. In a nutshell, boosting methods fit a series of models to the data, each one fixing the mistakes of the previous one: They start from some base model and fit another model to its mistakes. This process is repeated many times, composing more and more "fixer models", until some stopping criteria is met. For the purpose of this bonus exercise, this intuition is enough.
   - **Operating XGBoost in Code.** We will operate xgboost using this library. Its API is similar to sklearn models, used for the trees in previous questions. Use the `XGBClassifier` model class. You can find an example for loading this class in the helpers file.
   - **Question.** Train an XGBoost model with the same parameters as the random forest from Q6 as well as `learning_rate=0.1` . Report its test accuracy and visualize the XGBoost predictions as in Q4. How are the predictions of XGBoost different from the ones of the random forest? Which algorithm is more successful in this task? Explain.

7 Ethics

We will not tolerate any form of cheating or code sharing. You may only use the `numpy`, `pandas`, `faiss`, `mathplotlib`, `sklearn` and `xgboost` libraries.

8 Submission Guidelines

You should submit your report, code and README for the exercise as $ex1\_\{YOUR\_ID\}.zip$ file. **Other formats (e.g. *.rar*) will not be accepted!**.

The README file should include your name, cse username and ID.

Reports should be in PDF format and be no longer than 5 pages in length. They should include any analysis and questions that were raised during the exercise. Please answer the questions in Sec. 5, 5.3& 6 in sequential order. **You should submit yourr code (including your modified helpers file), Report PDF and README files alone without any additional files.**

8.1 ChatGPT / Github Copilot

For this exercise specifically, we advise you to avoid ChatGPT / Github Copilot, as one of the purposes is to get familiar with **numpy**. Saying that, we do not prohibit using them. If you do choose to use them, write at the end of your report a paragraph detailing for which parts /functionalities you used them.

8.2 Submission

Please submit a zip file named $ex2\_\{YOURID\}.zip$ that includes the following:

1. A README file with your name, cse username and ID.
2. The *.py* files with your filled code.
3. A PDF report.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.