Fall 2018
EE 236A
Christina Fragouli
TA D. Basu

EE236A: Linear Programming

Extra Project
Due: Friday Dec. 7th

Project Description:
In this project, we will use some of the linear programming tools that we have developed to understand some aspects of nearest neighbor classification. In order to do that we must first look at set cover optimization as a clustering method. This will be followed by formulating an optimization problem to obtain a sparse data set which can be very well used for the purpose of nearest neighbor classification.

Prototype Selection for Nearest Neighbor Classification

1 Background

Nearest neighbors is a popular pattern recognition technique used for both classification and regression purposes. It is based on the notion that averaging over nearby elements of a data set can help us obtain a good classification or a prediction for a test point. This is known to give good results in numerous applications however it is not scalable when we have large data sets and in addition it holds less interpretable meaning. Rather what would be more useful is to have a few points which represent the data set and can be useful for domain specialists in making sense of the data set.

Given a data set $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^p$ with corresponding class labels $y_1, \ldots, y_n \in \{1, \ldots, L\}$, we want to identify prototype sets $\mathcal{P}_l \subseteq \mathcal{X}$ for each class l. Bien et al [1] formulate an integer optimization problem with the goal of trying to identify such prototypes sets which conform to some desirable properties. In general, every training sample should have a prototype of its same class in its neighborhood, no point should have a prototype of a different class in the neighborhood and finally there must be as few prototypes as possible. Before proceeding we must define the neighborhood of a point as $B(\boldsymbol{x}) = \{\boldsymbol{x}' \in \mathbb{R}^p : d(\boldsymbol{x}', \boldsymbol{x}) < \epsilon)\}$ which is a ball of radius ϵ centered at $\boldsymbol{x}$, ($d(\cdot, \cdot)$ is a distance metric). $B(\boldsymbol{x})$ is a set of all points lying within an ϵ radius of $\boldsymbol{x}$. For now we will only look at the euclidean distance i.e. $d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2$ for defining the neighborhood of a point.

1.1 Set Cover Integer Program

We have a set of points $\mathcal{X}$ and a collection of sets $\{B(\boldsymbol{x})\, \forall \boldsymbol{x} \in \mathcal{X}\}$ which form a cover of $\mathcal{X}$. We seek the smallest subcover of $\mathcal{X}$. In other words we wish to find the smallest subset of points $\mathcal{P} \subset \mathcal{X}$ such that $\{B(\boldsymbol{x})\ \forall \boldsymbol{x} \in \mathcal{P}\}$ covers $\mathcal{X}$. This problem can be formulated as an integer program with variable $\alpha_j = 1$ if $\alpha_j \in \mathcal{P}$ otherwise $\alpha_j = 0$.

$$\begin{array}{rll} \text{minimize} & \sum_{j=1}^{n} \alpha_j & \\ \text{subject to} & \sum_{j:\boldsymbol{x}_i \in B(\boldsymbol{x}_j)} \alpha_j \geqslant 1 & \forall \boldsymbol{x}_i \in \mathcal{X} \\ & \alpha_j \in \{0,1\} & \forall \boldsymbol{x}_j \in \mathcal{X} \end{array}$$

Refer [1], section `2.2` for more details.

1.2 Prototype Selection

Taking set cover optimization as a starting point we will formulate an integer program which expresses the previously mentioned desirable properties of our prototype set as linear and integer constraints. The variable $\alpha_j^{(l)} \in \{0,1\} = 1$ when $\boldsymbol{x}_j \in \mathcal{P}_l$ otherwise 0. To adhere to properties we would like

1. $\sum_{j:\boldsymbol{x}_i \in B(\boldsymbol{x}_j)} \alpha_j^{(y_i)} \geqslant 1 \quad \forall \boldsymbol{x}_i \in \mathcal{X}$ which means every point must be covered by at least one prototype from the same class, however we associate a slack variable ξ_i that we would like to be 0 more often.

2. $\sum_{j:\boldsymbol{x}_i \in B(\boldsymbol{x}_j), l \neq y_i} \alpha_j^{(l)} \leqslant 0 \quad \forall \boldsymbol{x}_i \in \mathcal{X}$ which means every point should not be covered by prototypes of a different class. However we associate a slack variable η_i which we would like to be 0 more often. Refer [1] sections `2.1 and 2.3`.

Finally the integer program for this purpose is

$$\begin{array}{l} \text{minimize}_{\alpha_j^{(l)},\xi_i,\eta_i} \displaystyle\sum_i \xi_i + \sum_i \eta_i + \lambda \sum_{j,l} \alpha_j^{(l)} \\ \displaystyle\sum_{j:\boldsymbol{x}_i \in B(\boldsymbol{x}_j)} \alpha_j^{(y_i)} \geqslant 1 - \xi_i \qquad \forall \quad \boldsymbol{x}_i \in \mathcal{X} \\ \displaystyle\sum_{j:\boldsymbol{x}_i \in B(\boldsymbol{x}_j), l \neq y_i} \alpha_j^{(l)} \leqslant 0 + \eta_i \qquad \forall \quad \boldsymbol{x}_i \in \mathcal{X} \\ \alpha_j^{(l)} \in \{0,1\} \qquad \forall\ j,l \qquad \xi_i, \eta_i \geqslant 0 \qquad \forall i \end{array}$$

$\lambda \geqslant 0$ specifies the cost of adding a prototype. λ and ϵ are the hyper-parameters that must be fixed before solving the integer program. Refer [1] section `2.3` for more details on how to choose these hyper-parameters. Ideally we will always set $\lambda = 1/n$.

2 Project Details

While we know that the set cover optimization problem formulated above for identifying a prototype set is NP-hard there exist algorithms which will help us arrive at good feasible solutions with guarantees on how far they are from the optimal solution.

In this part of the project we will implement one out of two of these algorithms which have been mentioned in the paper [1] section 3. The first one being LP relaxation and randomized rounding to get to a good feasible solution. Implement this approach so as to mimic the algorithm exactly as in [1] section 3.1. The disadvantage of this approach is that it requires solving an LP which is slow and memory intensive for a large data set. To combat the negatives of the previous method, one can also think of using the greedy approach for solving the integer optimization problem. Refer [1] section 3.2 to read about the greedy heuristic as proposed by Bien et al. However you are only required to implement the first approach.

`Part 3` of this document talks about the data sets that you will work with. `Part 4` contains guidelines on the implementation which will go into the python files. `Part 6` contains the grading schema. Some useful modules would be predict(test_points) and objective_value(). The first one can be used to make predictions for test points both from in and out of the data set using the single nearest neighbor rule. The second one will be useful for comparing the objective of the LP relaxations followed by randomized rounding approach for different values of ϵ (or number of prototypes). Comment on the computational performance across different data sets. In `part 6` you will find suggested changes which can help improve the performance of the algorithm for nearest neighbor classification. Make some suggested changes to the optimization problem. Compare the performance with and without making the changes.

Note on cross validation: In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

3 Data set

We will be using the public data sets available in the `sklearn` package of python (refer `link`). For sanity checks through the coding process you can make use of the iris data set. This data set consists of 3 classes of 50 instances each, where each class refers to a type of iris plant. Each data point has 4 attributes.

This will be followed by experiments on the breast cancer Wisconsin data set which has 569 instances each with 30 attributes which represent different aspects of the cell nucleus ranging from concavity, perimeter, area, smoothness, texture etc. The classes are benign and malignant.

Eventually we will use the digits data set for prototype selection. The digits data set is made up of 1797 (8×8) images of handwritten digits belonging to 10 different classes from 0 to 9. We will expect to see different computational speeds for the different data sets.

4 Framework Implementation

The implementation of your new classifiers should go into the files `classifier.py`. This is an incomplete implementation of the python class for prototype selection which is based on [1]. You will be using `numpy` extensively so it would help to refer the *documentation* and this *tutorial* from time to time. Some of the details for completing the implementation are included below:

- `__init__`(): This is the constructor where you will initialize the required member variables as mentioned in `classifier.py`.
- `train_lp()`: This is where you will implement the linear programming formulation and solve using cvxpy.
- `objective_value()`: This is where you will compute the objective value of the integer optimization problem after the training phase.
- `predict()`: This function must compute the prediction for a given number of instances using the one nearest neighbor rule after the training phase.
- `cross_val()`: This function returns the average test error, average number of prototypes and average objective value for a given ϵ, λ and data set. Modify it as required. Note that it is not a member of the class `classifier()`.
- You are required to implement other functionalities in terms of modules (functions) as per the requirements. It is advisable to keep the code modular so as to enable efficient debugging and re-usability.

When implementing your classifier, make sure to make it a generic implementation that takes inputs of any number of features, and any number of classes. Although you will mainly try your classifier on the particular data sets referenced in this project we may test your classifier on a different data set with different parameters.

5 Performance Metrics

We will be using two performance metrics to evaluate your algorithm:

1. *Test error*: After the selection of prototypes, classify a point as the label of the nearest prototype. The average misclassification rate is the test error.
2. *Cover error*: After selection of prototypes, classify a point as the label of the nearest prototype if it lies in the ϵ neighborhood of the prototype. If no such prototypes exist then count it as a misclassification. The average misclassification rate is the cover error.
   Note: You may have to modify `predict()` accordingly to accommodate this.

An improvement in performance can be characterized by obtaining smaller errors with lesser prototypes which can be seen on overlay plots of error vs number of prototypes.

6 Suggested Modifications (1 point)

Out of the 5 points allotted for this project, 1 point is assigned to an analysis of minor modifications to the integer program which could be beneficial for the end goal of nearest neighbor classification. You are supposed to make at least one modification to the existing optimization problem in order to improve the performance. Creativity is welcome here however you need to provide an explanation to justify your proposed modification. Document your modifications and observations in the report. Suggested changes are:

- Restrict the regime where a point has itself but no other point in the neighborhood and yet is selected as a prototype. Report your observations after tweaking the optimization problem in the case of the randomized rounding algorithm.
- Modify the ILP so that almost every point has in its neighborhood, at least two prototypes from the same class. Does that help improve the performance of one nearest neighbor classification?
- Is it required that the ϵ ball be defined by a distance metric? Incorporate other ways of defining a neighborhood and compare their performance with each other for the randomized rounding approach.
- Instead of the third term in the ILP objective, try to enforce a strict upper bound on the number of prototypes to be selected for every label. Introduce some slack and compare results for both of the above with the original LP. Document the modified optimization problems in your report and comment on the feasibility of the solution obtained via the randomized rounding approach with or without the slack variable.

Eventually if after making the changes you are able to achieve smaller test errors with the same number of prototypes, you would have an improved algorithm however your analysis will fetch you credit.

7 Grading

- This project is for 5 points.
- You are required to implement the logic in [1] and analyze the observations:
  - (`2 points`) Complete the library classifier.py as per the annotations included in the python file and use the Iris data set for sanity checks.
  - (`1 point`) Perform 4-fold cross validation with the breast cancer data set to observe the variation of the test error and cover error with ϵ ranging between 2 percentile to

40 percentile of the inter-point distances. You can use `numpy.percentile` wherever required. Provide plots of the test error and cover error vs average number of prototypes.

  - (`1 point`) Train the classifier for the digits data set using the LP and randomized rounding approach and plot the variation of the integer program objective for different values of ϵ. Note that this is different from the optimal value of the LP that is solved in the process.
  - (`1 point`) Incorporate at least one modification which could be something you thought of or from the suggestions in `part 6` and include your observations. Plot cover error vs number of prototypes after you have modified the optimization problem.
  - Provide a report of up to 2 pages, explaining your observations.

- You should submit your implementation of `classifier.py`, any additional files that you may have created, as well as your report. You will submit this by December 7th.
- For the purposes of grading, we will check your implementation locally after submission. We may use a different data set when grading, so make sure your implementation of the classifier is generic, i.e., it takes inputs of any number of features, and any number of classes and meets the specifications mentioned under framework implementation.

References

[1] Jacob Bien, Robert Tibshirani, "Prototype Selection for Interpretable Classification", 2012