Machine Learning A (2024)
Home Assignment 1

Carlo Rosso rkm957

Contents

1 Make Your Own (10 points)

I am going to follow the order of the question and I am going to answer them one by one, follows the answers to the first question.

1. I would collect, the grade to each assignment, their grades' mean, their study program and their attendance to the lectures. Let n_a be the number of assignments and G the set of grades you can take in each assignment (considering they can only be the same, for semplicity), S the set of study programs and let the attendance be the ratio between attended class over the total number of class this far. Then $\mathcal{X} = G^{n_a} \times [2, 12] \times S \times [0, 1]$.

2. The label space $\mathcal{Y}$ is the set of all possible grades that can be taken in the final exam, so $\mathcal{Y} = G$ (referring to the previous question).

3. I would define the lost function as the mean squared error between the predicted grade and the real grade, so $\updownarrow(y', y) = (y' - y)^2$.

4. I define the distance measure as the absolute difference ($|x_1 - x_2|$) in each dimension which already defines such an operation and I would use the following formula for the computation of the distance in the S dimension:

$$d(s_1, s_2) = \begin{cases} 0 & \text{if } s_1 = s_2 \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

5. I think it is a quite basic algorithm, so I suppose it would obtain some kind of result, but I am not able to guess any result, I don't think I am very good at guessing. I think it can be considered a good starting point for the model, but I also think you should try many different lost functions and distance measures to see which one works the best.

6. There could be privacy issue on how you get access to the information to build the model. Data can be too sparse or too noisy.

2 Digits Classification with K Nearest Neighbors (40 points)

2.1 Task #1

```
def distance(x, y):
    return np.sum((x - y) ** 2)

def compute_predictions(distances, training_labels):
  sorted_distances = np.argsort(distances)
  predictions = np.cumsum(training_labels[sorted_distances])
  return [-1 if x < 0 else 1 for x in predictions]

def loss(predictions, labels):
  return np.sum((labels != predictions)) / len(predictions)

def knn(training_data, training_labels, test_data, test_labels):
  distances = [[distance(x, x_train) for x_train in training_data] for x
    in test_data]
  predictions = np.array([compute_predictions(x, training_labels) for x
    in distances]).T
  return [loss(x, test_labels) for x in predictions]
```

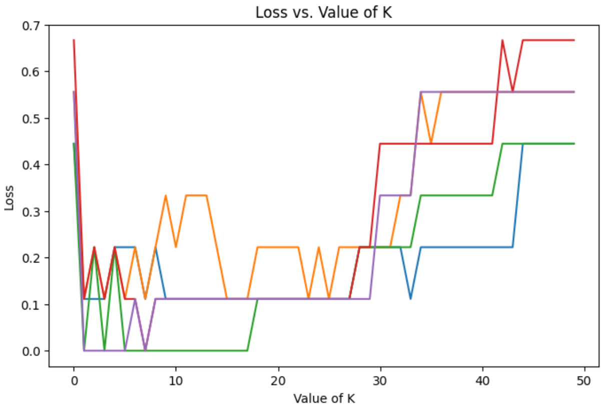

Figure 1: Validation test of size 10: influence of k over the loss

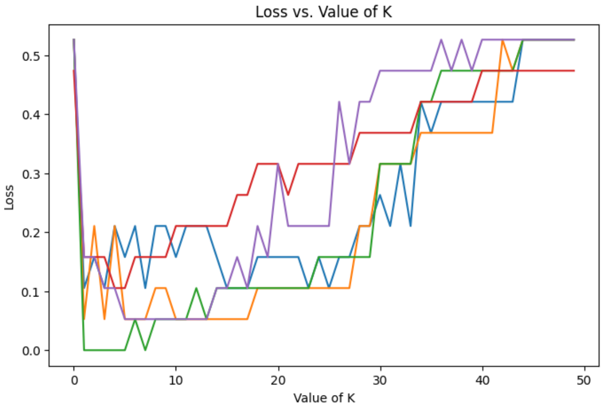

Figure 2: Validation test of size 20: influence of k over the loss

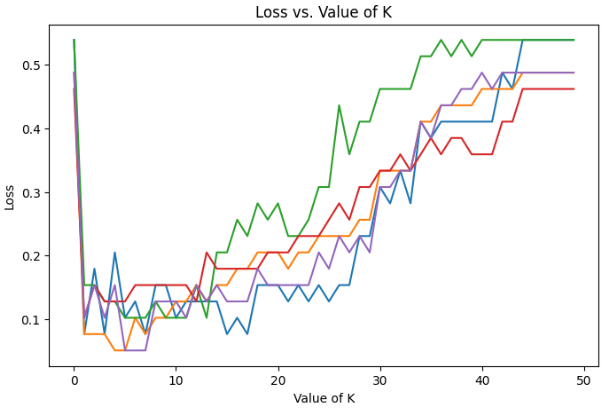

Figure 3: Validation test of size 40: influence of k over the loss

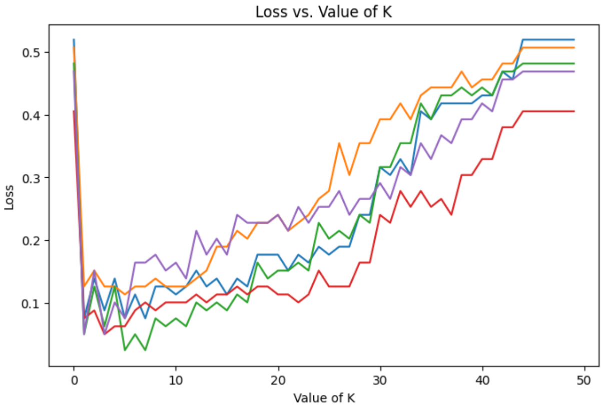

Figure 4: Validation test of size 80: influence of k over the loss

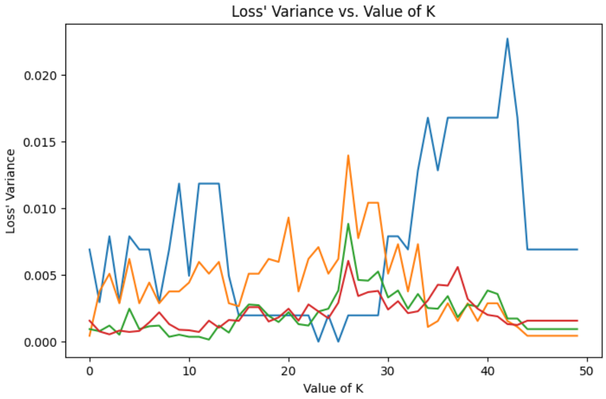

Figure 5: Influence of k over validation errors' variance

2.1.1 What can you say about fluctuations of the validation error as a function of n?

The fluctuations of the validation error seems to decrease with the number of samples in the test set. This is probably due to the fact that the more samples you have, the more precise we compute the loss.

2.1.2 What can you say about the prediction accuracy of K-NN as a function of K?

It looks like it doesn't really matter the size of the test set, the accuracy is going to be at the highest as long as $1 < k < 15$, then it starts to decrease. In addition, I suppose it decreases linearly with k up to $k = 50$, since that is the maximum value of k we have tested. When $k = 50$ the accuracy gets to be about 50%. Noting that this is a binary classification problem, the algorithm at $k = 50$ is as good as a random guess.