PDS Challenge 1:
Weighted and Regularized kNN for Classification

SR+ITL

September 29, 2023

Abstract

In this first challenge, we embark on the development and analysis of a specialized k-nearest neighbors (kNN) algorithm designed for classification. Our exploration delves into two nuanced variations of the kNN algorithm: (i) a version using a weighted distance for the feature coordinates, where the weights are chosen to maximize classification accuracy, and (i) a version that considers the marginal distribution of the labels and preserves this distribution during inference.

1 Problem Setting

The challenge presents a dataset divided into training and testing sets, with an 80% - 20% split ratio, denoted as $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, respectively. Each entry in these datasets consists of a tuple $(\mathbf{x}_k, y_k)$, where $\mathbf{x}_k \in \mathbb{R}^d$ represents feature vectors, and $y_k \in \mathcal{C}$ is a categorical label variable with C possible labels.

The core objective is to develop a modified k-nearest neighbors (kNN) algorithm for classification, aimed at maximizing the classification accuracy between the predicted variable $\hat{y}_k$ and the true variable y. The algorithm undergoes training on the dataset $\mathcal{D}_{\text{train}}$ and is assessed for performance on the dataset $\mathcal{D}_{\text{test}}$.

For this first assignment, you will be working with the Asteroid Dataset, which is taken from Kaggle. The data contains 45 features in total, including your target label. Note that not all features might be useful to your kNN model. The features are as follows:

- **id**: Object database entry id.
- **spkid**: JPL SPK-ID numbers are unique identifiers assigned to celestial objects.
- **full name**: object full name or designation.
- **pdes**: Primary Designation.
- **name**: Object IAU name.
- **prefix**: comet designation prefix.
- **neo**: Near-Earth Object flag.
- **pha**: Potentially Hazardous Asteroid flag.
- **H**: absolute magnitude parameter, a measure of intrinsic brightness of the asteroid.
- **diameter**: object diameter (from equivalent sphere) in km.
- **albedo**: geometric albedo, a measure of how reflective a celestial body is.
- **diameter sigma**: 1 sigma uncertainty in the measurement of the object's diameter.
- **orbit id**: a unique identifier assigned to a particular orbit solution for a celestial object.

- **epoch**: epoch of osculation in Julian day form, a specific moment in time at which the orbital elements of a celestial object are defined.
- **equinox**: equinox of reference frame.
- **e**: eccentricity, a measure of how elliptical an orbit is.
- **a**: semi-major axis, average distance from the sun.
- **q**: perihelion distance, closest distance to the sun as it travels its orbit.
- **i**: inclination, angle with respect to x/y ecliptic plane in degrees.
- **om**: mean anomaly, a measure of how far the celestial body is in its orbit.
- **w**: aphelion distance, furthest distance a celestial body gets from the sun.
- **moid**: earth minimum orbit intersection distance, smallest distance betwixt the orbit of two astronomical bodies, given in AU.
- **class**: your target for classification. The classification this particular celestial body belongs to.
- **class id**: id for the class the asteroid belongs to.

There are other features that include sigma. Just like diameter sigma is a 1 sigma uncertainty in the measurement of the diameter, other sigma features are a 1 sigma uncertainty in the their respective measurement. You can also browse the website were the data set was derived from to check yourself: JPL Nasa.

2 Classic kNN Algorithm Development

To start, it's essential to understand the foundational principles of the classic kNN algorithm for classification.

This section of the challenge will focus on the development of the classic kNN algorithm.

The classic kNN algorithm operates as follows:

1. Given a new data point with features $\mathbf{x}$, calculate the distance between $\mathbf{x}$ and all data points in the training set $\mathcal{D}_{\text{train}}$. You can use the L-2 distance or other suitable distance metrics.
2. Select the top k data points (neighbors) with the shortest distances to $\mathbf{x}$. These neighbors are the most similar data points to the one you want to classify.
3. Among these k neighbors, determine the majority class (mode) or calculate weighted class probabilities to predict the class for $\mathbf{x}$. In the case of ties, you may choose a tie-breaking strategy.

Your task:

1. Implement the classic kNN algorithm for classification from scratch using Python. Be careful to write the algorithm in an efficient way, both in terms of execution time and memory utilization.
2. Use the classic kNN algorithm to make predictions on the testing dataset $\mathcal{D}_{\text{test}}$.
3. Evaluate the performance of your classic kNN classifier using appropriate classification metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
4. Experiment with different values of k (e.g., $k = 3$, $k = 5$, $k = 7$) and observe how the choice of k affects classification performance.

Developing the classic kNN algorithm will provide you with a strong foundation to compare and contrast with the weighted and regularized kNN variations explored in other sections of this challenge. It's an essential step in understanding the behavior and capabilities of kNN classifiers.

3 Weighted kNN

In our classroom discussions, we explored the utilization of the L-2 distance metric for computing the distance between two feature vectors, $\mathbf{x}_k$ and $\mathbf{x}_j$, as expressed by:

$$d(\mathbf{x}_k, \mathbf{x}_j) = \|\mathbf{x}_k - \mathbf{x}_j\|_2^2 = \sum_{l=1}^{d} \left(\mathbf{x}_k(l) - \mathbf{x}_j(l)\right)^2 . \tag{1}$$

However, for this project, we introduce a novel distance metric:

$$d(\mathbf{x}_k, \mathbf{x}_j) = \sum_{l=1}^{d} w_l \left(\mathbf{x}_k(l) - \mathbf{x}_j(l)\right)^2 , \tag{2}$$

where $w_l > 0$ and $\sum_{l=1}^{d} w_l = 1$.

Your task is to determine how to select the set of parameters $\{w_l\}_{l=1}^{d}$ in a manner that minimizes the cross-entropy of the estimator. Furthermore, you are required to repeat the optimization of the weight parameters w_l for kNN, considering different numbers of neighbors: $k = \{2, 4, 6\}$.

4 Weighted kNN Algorithm Development

In our previous sections, we've explored the classic kNN algorithm and discussed its utilization of the L-2 distance metric for calculating distances between feature vectors, $\mathbf{x}_k$ and $\mathbf{x}_j$, as represented by:

$$d(\mathbf{x}_k, \mathbf{x}_j) = \|\mathbf{x}_k - \mathbf{x}_j\|_2^2 = \sum_{l=1}^{d} \left(\mathbf{x}_k(l) - \mathbf{x}_j(l)\right)^2 . \tag{3}$$

However, in this section, we introduce an intriguing twist - the concept of weighted kNN. Instead of relying solely on the classic L-2 distance, we will explore a modified distance metric:

$$d(\mathbf{x}_k, \mathbf{x}_j) = \sum_{l=1}^{d} w_l \left(\mathbf{x}_k(l) - \mathbf{x}_j(l)\right)^2 , \tag{4}$$

where each weight parameter $w_l > 0$ and $\sum_{l=1}^{d} w_l = 1$.

When evaluating the performance of your k-nearest neighbors (kNN) algorithm for classification in this assignment, we will focus on a straightforward and intuitive metric: the "classification accuracy".

Classification accuracy measures the proportion of correctly classified instances in the test dataset $\mathcal{D}_{\text{test}}$.

Mathematically, classification accuracy is computed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Where:

- The "Number of Correct Predictions" is the count of instances in $\mathcal{D}_{\text{test}}$ that were classified correctly by your kNN classifier.
- The "Total Number of Predictions" is the total number of instances in $\mathcal{D}_{\text{test}}$.

Using classification accuracy as the evaluation metric for this assignment provides a straightforward way to assess the performance of your algorithm in terms of its ability to correctly classify data points. It's a fundamental metric for classification tasks and provides an excellent starting point for your machine learning journey.

1. Determine an optimal set of weight parameters $\{w_l\}_{l=1}^{d}$ that minimizes the classification accuracy. You should be creative in this task, since this problem might not have a closed form solution!

2. Implement the weighted kNN algorithm based on the optimized weight parameters. You can choose to implement this algorithm from scratch or use existing machine learning libraries that support custom distance metrics.

3. Utilize your weighted kNN classifier to make predictions on the testing dataset $\mathcal{D}_{\text{test}}$.

4. Evaluate the performance of your weighted kNN classifier using a range of classification metrics, including accuracy, precision, recall, F1-score, and a confusion matrix. Compare these results to those obtained using the classic kNN algorithm developed in a previous section.

5. Extend your exploration to consider different numbers of neighbors, e.g., $k = 3$, $k = 5$, $k = 7$. Investigate how varying k impacts the performance of your weighted kNN classifier in comparison to the classic kNN approac

Your journey into developing the weighted kNN algorithm will provide you with valuable insights into feature importance, distance metrics, and their influence on classification accuracy. This exploration is fundamental in advancing your understanding of machine learning techniques and their practical applications.

5 Regularized kNN for Balanced Classification

While the maximization of accuracy is a valuable strategy in many classification tasks, it's essential to recognize scenarios where maintaining the integrity of the estimator with respect to the marginal distribution of labels becomes paramount.

This approach becomes critical when addressing potential bias that may arise due to disparities in class label frequencies. For instance, when one class significantly dominates the dataset in terms of frequency, predictions on the test set may tend to be skewed towards this more prevalent class, potentially compromising the overall performance.

In the context of our problem setting, as outlined in Section **??**, let's explore how we can adapt our approach to mitigate such bias. What modifications can we introduce to the kNN algorithm to better account for the underlying class distribution and ensure a balanced classification? In this section, we delve into these crucial considerations, exploring strategies to regularize our kNN classifier and achieve equitable treatment of diverse class labels.

1. Consider how the problem setting described in Section **??** can be modified to address potential bias arising from class label frequency disparities. Propose adjustments or extensions to the existing problem setting that ensure a more balanced classification process.

2. Explore and implement regularization techniques for your kNN classifier that take into account the relative frequencies of class labels. These techniques may include adjusting class weights or introducing custom distance metrics that mitigate bias.

3. Utilize your regularized kNN classifier to make predictions on the testing dataset $\mathcal{D}_{\text{test}}$ after applying the proposed regularization strategies.

4. Analyze and interpret the results obtained with a focus on classification accuracy. Discuss how the regularization strategies have affected the classifier's ability to handle class imbalance and potential bias.

By undertaking these tasks, you will gain a deeper understanding of regularization techniques in kNN and their impact on balanced classification. The use of classification accuracy as the evaluation metric will guide your efforts in developing a regularized kNN algorithm that not only maximizes the accuracy but also ensures equitable treatment of diverse class labels.

6 Performance visualization

This last section describes possible approaches to visualize your results that you can use in your code and video presentation. This is not a new problem, but rather a discussion of the tools you might use in the three tasks above.

In machine learning, it's essential to visualize the performance of your algorithms to gain insights and evaluate their effectiveness. For your kNN classifier, you can employ various visualization techniques to assess its performance. Here are some suggestions:

6.1 Confusion Matrix

A confusion matrix is a valuable tool to visualize the performance of a classifier, especially in multi-class classification problems like this one. It provides a summary of how well your kNN algorithm is classifying instances into different classes. Each row of the matrix represents the true class, while each column represents the predicted class. You can compute metrics like precision, recall, and F1-score based on the confusion matrix to understand the classifier's behavior.

6.2 ROC Curves and AUC

Receiver Operating Characteristic (ROC) curves are commonly used in binary classification to visualize the trade-off between true positive rate (sensitivity) and false positive rate. In multi-class classification, you can create ROC curves for each class or use techniques like One-vs-All to aggregate results. The Area Under the ROC Curve (AUC) can also be calculated to quantify the overall performance.

6.3 Class Separation Visualization

To gain an intuitive understanding of how well your kNN classifier separates different classes in the feature space, you can create 2D or 3D scatter plots. Each point represents an instance with its features, and you can use different colors or markers to represent class labels. This visualization can help identify areas of overlap between classes and assess the algorithm's ability to discriminate between them.

6.4 Hyperparameter Tuning Plots

Since you're experimenting with different values of k and weight parameters w_l, it's crucial to visualize how these hyperparameters affect the classifier's performance. You can create plots or graphs that show the accuracy or other relevant metrics as a function of these hyperparameters. This will help you choose the best hyperparameter values for your kNN classifier.

6.5 Decision Boundary Visualization

For 2D or 3D feature spaces, visualizing the decision boundary of your kNN classifier can be insightful. This allows you to see how the algorithm separates different classes and how the boundary changes with different hyperparameters. You can use contour plots or color-coded regions to represent class boundaries.

6.6 Learning Curve

A learning curve plots the classifier's performance (e.g., accuracy or cross-entropy) as a function of the training set size. By gradually increasing the training set size and evaluating the classifier's performance at each step, you can gain insights into how much data is required for your kNN algorithm to perform well. This can help with decisions related to dataset size and model complexity.

Remember to label your visualizations appropriately and provide interpretations of the results. Visualizations are powerful tools for conveying information and insights about your kNN classifier's performance.