Project 1 - SF2957 Statistical Machine Learning

Due November 28 2024, 19:00

1 Stochastic subgradients for training support vector machines

In this first part of the project you will design and implement a classifier for handwritten digit recognition. The dataset is made up of 1797 images of size 8×8. Each image is of a low-resolution hand-written digit. Each image is first transformed into a feature vector with length 64. To display images in the data set, run the Python 3 script `display_digits.py`.

The classifier is represented using a matrix

$$\Theta = [\theta_1 \ \theta_2 \ \cdots \ \theta_K] \in \mathbb{R}^{d \times K},$$

where there are $K = 10$ classes $\{0, 1, \ldots, 9\}$. Given Θ the predicted class for a data vector (image) $x \in \mathbb{R}^d$ is

$$\text{argmax}_{j=1,\ldots,K} \left(\sum_{i=1}^{d} x_i \Theta_{ij} \right) = \text{argmax}_{j=1,\ldots,K} [\Theta^T x]_j.$$

The data is represented as pairs $(x, y) \in \mathbb{R}^d \times \{0, 1, \ldots, 9\}$, where x is the vector representation of the 8×8 image and y is the label. To train the classifier we minimize the empirical risk using a multi-class hinge-loss

$$L(\Theta, (x, y)) = \max_{j \neq y} \left[1 + \sum_{i=1}^{d} x_i (\Theta_{ij} - \Theta_{iy}) \right]_+,$$

where $t_+ = \max(t, 0)$ is the positive part. More precisely, we aim to minimize

$$R^{\text{emp}}(\Theta) = \frac{1}{n} \sum_{k=1}^{n} L(\Theta, (x_k, y_k)).$$

(a) Show that $R^{\text{emp}}(\Theta)$ is convex.

(b) For a given pair (x, y), representing (image, label), provide an expression for a subgradient G of $R^{\text{emp}}(\Theta)$.

(c) Python 3 code for training the classifier is provided in the file `svmclassifier.py`. Two functions are left un-implemented: `svmsubgradient` and `sgd`. The first function implements the calculation of the subgradient in (b) for a given data point. The function `sgd` implements a projected stochastic gradient descent method, where you should take the stepsize ϵ_k proportional to $1/\sqrt{k}$ and project Θ to the ball

$$B_r = \{\Theta \in \mathbb{R}^{dk} : \sum_{ij} \Theta_{ij}^2 \leq r^2\}.$$

Implement the functions `svmsubgradient` and `sgd`.

(d) Evaluate the performance of the classifier, using training sets of size 20, 50, 100, 500, 1000, and 1500, and a test set of 100 images.

(e) Investigate other choices of the learning rate, say $\epsilon_k \propto k^{-\alpha}$ for $\alpha \in (0, 1]$, on the performance of the classifier.

2 Gaussian process classification

The `scikit-learn` package available at `https://scikit-learn.org/` provides basic machine learning functionality in Python. In this, the second part of the project, we will use their implementation of Gaussian process classification and regression. You will use the same data set of labeled handwritten digits as in the previous section.

(a) In the file `gpclassify.py` a Gaussian process multi-class classifier is implemented for claissifying the digits data set. Note that in the scikit implementation several binary one-versus-rest classifiers are fitted, it does not implement a true multi-class Laplace approximation for GP multi-class classification. Investigate the classification performance of the GP classifier with respect to size of the training data and the choice of covariance function.

(b) A rather simple way to produce a multi-class classifier with Gaussian processes is to use Gaussian process regression, as a replacement of Gaussian process classification. Using a one-hot representation of the labels, one can treat the classification problem as a regression problem with a K-dimensional response variable. GP regression is used to predict numerical values of the K-dimensional one-hot-vector and the classification is done by predicting the class with the largest predicted value. Use the functionality in the scikit class `GaussianProcessRegressor` to construct a classifier based on GP regression and evaluate its performance in terms of size of the training data and choice of the covariance function.

3 Convolutional neural network classification

In the third part of the project you will use a simple convolutional neural network to perform the classification of the digits data set using TensorFlow or PyTorch. Follow the instructions at `https://keras.io` and `https://www.tensorflow.org/guide/keras`, or `https://pytorch.org/get-started/locally/` to install TensorFlow with Keras or PyTorch into your Python environment. A useful tutorial on building a simple convolutional neural network for classifying the mnist data set is available at `https://www.tensorflow.org/tutorials/` or at `https://pytorch.org/tutorials/`.

(a) Build and train a simple convolutional neural network model (of one or two convolutional layers) to classify the images in the digits data set. Evaluate its performance.

(b) Investigate the performance of the convolutional neural network in terms of the size of the training data and the design of the layers. Try to find a network with as few parameters as possible, without sacrificing performance.

(c) (Optional) Try to implement a convolutional ceural network classifier on a more complex dataset. For example the 28 by 28 pixel digits dataset, found here: `https://keras.io/api/datasets/mnist/` or here: `https://pytorch.org/vision/stable/generated/torchvision.datasets.MNIST.html`.

4 Comparison and discussion

Compare the performance of all the classifiers for the digits data set studied in this project. Discuss their potential benefits and limitations.