Deep Learning Practical Work 1-c
Learning a SVM classifier

Nicolas Thome Guillaume Couairon Mustafa Shukor Asya Grechka
Matthieu Cord

Homework

Data, code, and PDF version of this file are available at
`https://rdfia.github.io`

Introduction

In the previous sessions, we built a processsing pipeline that transformed a RGB image into a compact representation that is more suited to image classification. Now, each image is encoded as a BoW vector describing in some way the presence (or absence) of $p = 1000$ patterns in the image. The vector of the i-th image, thus, is $\mathbf{x}_i \in \mathbb{R}^p$. Moreover, we know that each image can belong to one of 15 categories from our dataset. Therefore in this session, we will build the ultimate part of the pipeline, an image classifier that will takes as input the BoW vector ! A lot of different machine learning models can be used at that point, and we will use the popular SVM (Support Vector Machine).

Partie 1 – Some reminders on classification and SVM

Learning. A classification model has for goal to "learn" how to discriminate data, meaning to find a function $f_{\mathbf{w}}$ with parameters $\mathbf{w}$ so that $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$ can predict the correct category/label/target $\hat{\mathbf{y}}$ given the input $\mathbf{x}$. To do so, inductive models learn on a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1..N_{train}}$. The optimization process will find the optimal parameters $\mathbf{w}^*$ that minimize a cost function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})\right] = \arg\min_{\mathbf{w}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[\mathcal{L}(\mathbf{y}, f_{\mathbf{w}}(\mathbf{x}))\right] \tag{1}$$

SVM. In our case, $\mathbf{x}$ is the BoW representation of an image ; $y = \{-1, 1\}$ indicates whether the image belong or not to a given class ; f is the SVM, thus a linear model $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b$; and the cost function is $\mathcal{L} = \max(0, 1 - y(\mathbf{w}\mathbf{x} + b)) + \lambda||\mathbf{w}||_2^2$ where λ is an *hyperparameter* of the model. Hyperparameters are determined through cross-validation by the user, but aren't found by optimization as the other parameters of the model.

Multi-class. A SVM is a binary classifier, and thus can only classify among two classes "yes/no". If we want to classify among multiple classes "class 1, class 2, class 3, etc.", we need to apply a strategy. A common one is the the *one-vs-all* strategy where we learn one SVM per class. Each classifier must determine if a input sample belong to the class or not. The final prediction is obtained by choosing the class with the highest predicted score.

Evaluation. Our model is learned on a set of examples, this set is called the **training** dataset. Multiple metrics exist to evaluate whether our model has learned correctly (roc, f1, etc.) but the most common metric is the *accuracy* : how much examples were correctly classified. Having an excellent accuracy on the training dataset doesn't mean your model is good. The model could have "learn by heart" this set and not generalize at all on new never-seen-before samples. This is called the **overfitting**. To evaluate correctly our model, the evaluation must be done on a **testing** dataset on which the model is NEVER learned. Moreover, we often use a **validation** dataset that will act as a "fake" dataset. The optimal hyperparameters will be chosen depending on the accuracy on this validation dataset.

Partie 2 – Practice

We will need BoW representation of our images. You can use the ones you computed in the previous session, or use the ones provided at the beginning of the colab in the file `15_scenes_Xy.npz`.

Load in-memory the representation with numpy,
`data = np.load("15_scenes_Xy.npz", "rb")`. You'll get `data["X"]` and `data["y"]`.

TODO

- Load the features matrix $\mathbf{X}$ and the labels vector $\mathbf{y}$.
- Split the dataset into *train* / *validation* / *test* with the following distribution 70% / 10% / 20%. Use the function `train_test_split` of scikit-learn.
- Learn a SVM with scikit-learn on the train set with different values of C (equivalent to the λ in the previous formula). Start with $C = 1$ before testing other values.
- Evaluate your SVMs on the validation set to find the optimal value of C.
- Evaluate your best SVM on the test set to obtain the final performance.
- Repeat previous steps with other hyperparameters, such as the kernel (linear, polynomial, rbf, etc.) or the strategy (one-vs-all, one-vs-one).

Questions

1. ★ ★ Discuss the results, plot for each hyperparameters a graph with the accuracy in the y-axis.
2. ★ ★ Explain the effect of each hyperparameter.
3. Why the validation set is needed in addition of the test set ?