Assignment 1: Forest Cover-Type Prediction with Scikit-Learn

Arash Vahdat

CMPT 733

Fall 2015

Readings You are highly recommended to read the following readings while doing this assignment:

- Support Vector Machines: Ch. 12, Statistical elements of machine learning, Hastie et al.
- Random Forest: Ch. 15.1, 15.2, Statistical elements of machine learning, Hastie et al.
- Zero-mean unit-variance normalization: `https://en.wikipedia.org/wiki/Feature_scaling`

Provided files An archive with all necessary files for this assignment can be found at the following location on the RCG network:

`/cs/vml2/avahdat/CMPT733_Data_Sets/Assignment1/`

1 Introduction

The Scikit-learn[1] python package contains a handful of basic machine learning algorithms ranging from regression/classification to unsupervised clustering/dimensionality reduction models. The package contains many algorithms, however, the algorithms proposed were not designed for scalable solutions. Nonetheless, this package is very useful in the case of relatively small data (that can fit on a single machine) or for prototyping ideas. This assignment will serve as a review of three standard classification models (SVM, kernel SVM, Random Forests) using Scikit-lean in order to get familiar with the training/validation/test protocols used in this package.

In this assignment, we will work on a Kaggle challenge[2] designed for the problem of "Forest Cover-Type Prediction". As stated on the competition page "in this competition you are asked to predict the forest cover-type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover-type for a given 30×30 meter cell was determined from the US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from the data obtained from the US Geological Survey and the USFS. The data is in raw form (not scaled) and contains columns of binary values for independent qualitative variables such as "wilderness areas" or "soil type"."

[1] http://scikit-learn.org/

[2] https://www.kaggle.com/c/forest-cover-type-prediction

In the archive, you are provided with the file `forest_data.npz`. This file that can be loaded using `numpy.load` contains the following numpy variables:

- **`data_training`**: a matrix of size 25,000 $\times$ 54 which contains 25,000 training samples in each row. Each sample is represented with a 54 dimensional feature vector. Please refer to the description of the forest cover-type challenge for detailed information regarding these features.
- **`label_training`**: a one-dimensional vector of class labels for the training samples. Therefore, its length is 25,000 and it contains labels in $\{1,2,\ldots,7\}$
- **`data_val`**: a matrix of size 5000 $\times$ 54 which contains 5,000 samples that will be used for validation.
- **`label_training`** contains the class labels for the validation set.
- **`data_test`**: a matrix of size 550,000 $\times$ 54 which contains 550,000 samples used for test. This matrix can be ignored in this assignment.

2 Linear Support Vector Machine (SVM)

Multi-class linear SVM is an extension of binary linear SVM for multi-class classification problems. Let's represent a set of N training examples using $X = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$. This notation represents x_i as the i^{th} training sample and $y_i \in \mathbb{Y}$ the corresponding class label. It is assumed that each example is represented using a feature vector of $\phi(x_i)$. Note that in our forest cover-type classification problem, ϕ can be considered as the 54-dimensional representation of each region (training sample) and the label set $\mathbb{Y}$ corresponds to $\{1,2,\ldots,7\}$. In multi-class linear SVM, the model's parameters are trained by solving the following optimization problem:

$$\begin{aligned} &\min_{\{w_j\},\{\xi_j\}} \frac{1}{2}\sum_{j=1}^{|\mathbb{Y}|}||w_j||^2 + C\sum_{i=1}^{N}\xi_i \\ &s.t. \quad w_{y_i}^T\phi(x_i) - w_y^T\phi(x_i) > 1-\xi_i \quad \forall y \neq y_i \end{aligned} \tag{1}$$

In this case, the classifier is represented as a set of hyperplanes parametrized using w_j for each class ($j \in \{1,2,\ldots,|\mathbb{Y}|\}$). The optimization aims at finding a hyperplane per class such that the score of the ground-truth annotation ($w_{y_i}^T\phi(x_i)$) for each training sample is greater than the score for any other class ($w_y^T\phi(x_i)$ where $\forall y \neq y_i$).

Question 1: (5 marks) The main hyper-parameter in the optimization of a multi-class SVM is the trade-off parameter C. Briefly explain how changing C from 0 to infinity can impact the optimization problem in Eq.1.

Question 2: (10 marks) Use the `sklearn.svm.LinearSVC` implementation of multi-class linear SVM to train a linear SVM model using different C parameters. Validate your classifier on the **validation** dataset. Use the multi-class classification accuracy as validation measure [3]. Visualize your accuracy on the validation set for different C values. Explain the pattern observed when C varies. Include your visualization in the report.

[3] sklearn.metrics.accuracy_score

Question 3: (5 marks) Rescale all features using the zero-mean unit-variance normalization scheme. How does this normalization step affect your performance on the validation set when using different C values?

3 Kernelized Support Vector Machine (SVM)

Kernelized Support Vector Machine solves the optimization problem in the Eq.1 in its dual form and uses the kernel trick to replace $\phi(x_i)^T\phi(x_j)$ with a kernel $K(x_i,x_j)$ that measures the dot product of these two vectors in a (potentially) high dimensional space. In general a kernelized SVM model tends to achieve a better performance than linear SVM.

Question 4: (10 marks) The `sklearn.svm.SVC` class provides a general implementations of SVM that can be used for training and testing both, kernelized and linear SVM classifiers. Use this class to train a kernelized SVM with the Radial Basis Function (RBF) kernel ($K(x_i,x_j) = \exp(\frac{||x_i-x_j||^2}{2\sigma^2})$). In addition to the C parameter, find the best scale parameter σ for the RBF kernel. Visualize the performance of your classifier on the validation set using different C and σ values. Report your figures.

Note: You will notice that training a kernelized SVM using all 25000 training samples is very slow. Use a small subset of training examples for this question.

Question 5: (5 marks) Briefly explain why training a kernelized SVM classifier is significantly slower than training a linear SVM as observed in the previous section.

Question 6: (5 marks) In the previous question, you probably noticed that training a kernelized SVM using all training examples is very slow. One approach to speed up the process is to precompute the kernel matrix. This way the same kernel matrices will be reused while cross-validating over the C parameter. Train a kernelized SVM classifeir using precomputed RBF kernel. For this question use as many as training data that you can and visualize the performance of your classifier on the validation set using different C values. You can use `rbf_kernel` from `sklearn.metrics.pairwise` to compute RBF kernel.

Note: For this question, you can use the same value for σ selected from the first question. But you will still need to cross-validate the C parameter. Why cannot we fix the C parameter to the best value from the first question?

Question 7: (5 marks) The main advantage of using the precomputed kernel option in SVM is that you can define your own kernel. The following link lists a set of common kernels: `http://crsouza.com/2010/03/kernel-functions-for-machine-learning-applications/#kernel_functions`. Replace the RBF kernel from the previous question with a variant of the Rational Quadratic Kernel as follows:

$$k(x_i,x_j) = 1 - \frac{||x_i - x_j||}{||x_i - x_j|| + const} \tag{2}$$

Visualize the performance of your classifier on the validation set, using different C and *const* values. Report your figures. Note that $||x_i - x_j||$ represents the Euclidean distance between x_i and x_j which can be computed efficiently using `euclidean_distances` from `sklearn.metrics.pairwise`.

4 Random Forest

Ensemble models are another group of popular classifiers that are widely used in Big Data applications. The main advantage of these methods is their great scalability which comes from their parallelizable formulation. In this section, we will use the Scikit-learn implementation of a Random Forest classifier as an example of ensemble methods. This implementation can be easily applied on small datasets and on a standalone machine. Later on in this course, we will see more scalable implementations of Random Forests.

Question 8: (15 marks) The Random Forest classifier discussed during lectures can be created using the `sklearn.ensemble.BaggingClassifier` class with `sklearn.tree.DecisionTreeClassifier` as the base estimator. Use these two classes to train a Random Forest classifier on all training samples of the forest cover-type problem and validate it on the validation set. There are three main parameters that should be set before training a Random Forest classifier: 1) n the number of training examples sampled for training each decision tree. 2) m the number of sampled features used for each decision tree. 3) T the number of trees used in the model. Set all these parameters via cross-validation on the validation set. Visualize your accuracy on the validation set for different values of these parameters. Report your figures.

Note: You can validate different values for one parameter (e.g. n) while holding the two other parameters (e.g. m and T) fixed. This will reduce the number of experiments needed to set all three parameters. Also, if your machine has multiple cores make sure to set n_jobs in the $BaggingClassifier$ classes to speedup the training process.

5 Other Classifiers

Question 9: (10 marks) Implement another classifier of your own choice using any available package. Try to understand your selected model, and, design experiments to tune its parameters. In your report, briefly describe your classifier and its main parameters and report your performance. Your classifier does not need to outperform classifiers used in this assignment, however, it is important to design experiments to set its parameters properly.

6 Submission

Your assignment should be submitted online at `https://courses.cs.sfu.ca/`. You should submit two files for this assignment :

1. **Report:** Create your report in PDF format including your answers and figures to all the above questions.
2. **Code:** Create an archive with all scripts used in this assignment. You should create your scripts such that your results can be reproduced. Please include a small readme.txt file in your archive briefly explaining which script was used for each question.