INF554: MACHINE LEARNING I
ÉCOLE POLYTECHNIQUE
Lab 6: Kernels, SVM, and Adaboost

Stratis Limnios, Jesse Read, Nikolaos Tziortziotis and Michalis Vazirgiannis

November 5, 2018

1 Description

The goal of this lab is to study

- Kernels (Linear, Gaussian)
- Support Vector Machines
- Boosting and Bagging methods

2 Support Vector Machines

In the second part this lab, we will apply support vector machines (SVMs) on three different 2D artificial datasets. Applying SVM[1] in these datasets will help us gain an intuition of how SVMs work. Additionally, you will try using different values of the C parameter with SVMs. Actually, the C parameter is a positive value that controls the penalty for misclassified training examples. A large C parameter tells the SVM to try to classify all the examples correctly. C plays a role similar to $\frac{1}{\lambda}$, where λ is the regularization parameter that we were using previously for logistic regression. Additionally, we will implement and examine two different types of kernel function, the linear kernel and the Gaussian kernel.

2.1 SVM with linear kernels

We will begin with a simple dataset which is linear separable. The script `main1.py` will plot the training data (see Fig.1). In this dataset, the positions of the positive examples (indicated with o) and the negative examples (indicated with +) suggest a natural separation indicated by the gap. However, notice that there is an outlier positive example o on the far left at about $(0.1, 4.1)$. As part of this exercise, you will also see how this outlier affects the SVM decision boundary.

In this point, we need to implement a linear kernel. The linear kernel between a pair of instances, $(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is given as:

$$K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \sum_{k=0}^{m} x_k^{(i)} x_k^{(j)}, \tag{1}$$

where m is the number of features.

[1] In our code, we will use the SVM classifier implemented at `scikit-learn`.

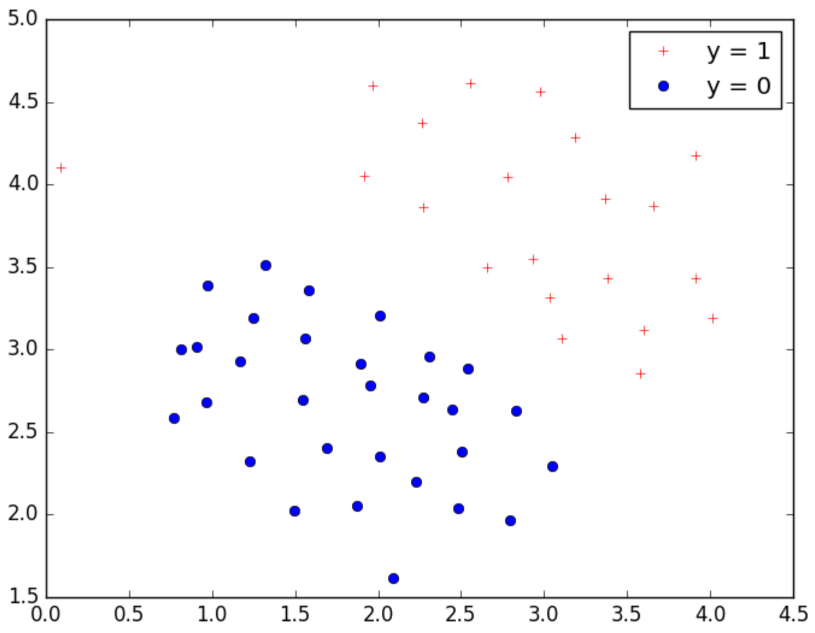

Figure 1: Dataset 1

2.1.1 Tasks to be performed

1. Fill in the code in `linearKernel.py` function to compute the linear kernel between two examples, $(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.

2. Try different values of C on this dataset. Specifically, you should select the value of C in the script `main1.py` to $C = \{1, 100\}$ and run the SVM training. When $C = 100$, you should observe that the SVM now classifies every single example correctly, but has a decision boundary that does not appear to be a natural fit for the data. When $C = 1$, you should find that the SVM puts the decision boundary in the gap between the two datasets and misclassifies the data point on the far left.

2.2 SVM with Gaussian kernels

Now, we are going to use SVMs for non-linear classification. More specifically, you will be using SVMs along with Gaussian kernels on datasets where classes are not linearly separable. From the illustration of the dataset in Fig. 2, it is apparent that there is no linear decision boundary that separates the positive and negative examples for this dataset. However, by using the Gaussian kernel with the SVM, you will be able to learn a non-linear decision boundary that can perform reasonably well for the dataset.

2.2.1 Gaussian kernel

To find non-linear decision boundaries with the SVM, we need to implement a Gaussian kernel. The Gaussian kernel can be seen as a similarity function that measures the distance between a pair of instances, $(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$. The Gaussian kernel is also parameterized by a bandwidth parameter, σ, which determines how fast the similarity metric decreases (toward 0) as the examples are further apart. The

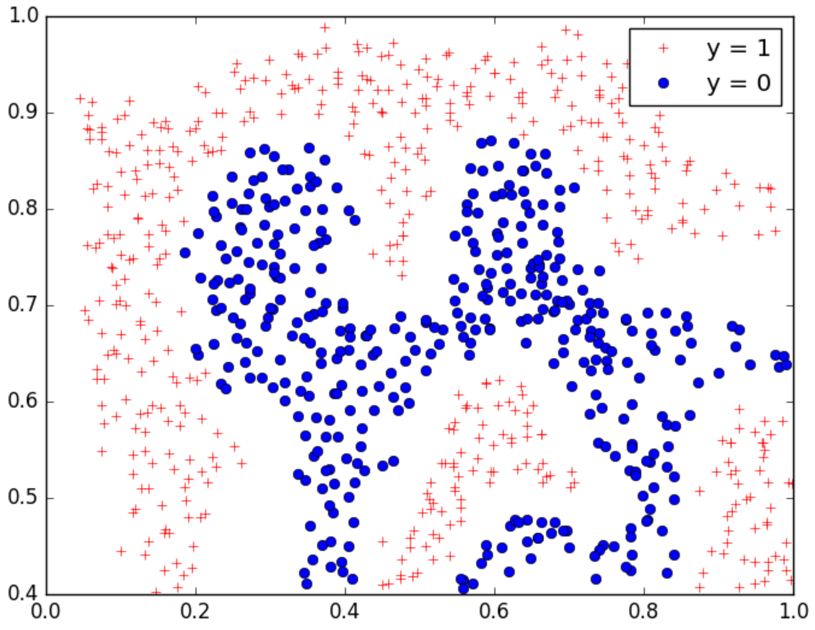

Figure 2: Dataset 2

Gaussian kernel is defined as follows:

$$K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\frac{\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|^2}{2\sigma^2}\right) \quad (2)$$

2.2.2 Tasks to be performed

1. Fill in the code in `gaussianKernel.py` function in order to compute the Gaussian kernel between two examples, $(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$.
2. Examine different values for C and σ and try to interpret the effects on the performance of SVM.

3 Adaboost

The goal of this lab is to study ensemble learning methods and in particular the one of AdaBoost algorithm. Boosting is a powerful technique for combining multiple base classifiers to produce an ensemble classification model that can significantly outperform any of the base classifiers. The most widely used method is the one of *AdaBoost* (adaptive boosting). One basic charactering of boosting learning methods is that they can give good results even if the base classifiers have performance that is only slightly better than random (this is the reason that the base classifiers are also known as weak learners).

One characteristic of the boosting method is that the base classifiers are trained in sequence and each base classifier is trained using a weighted form of the dataset in which the weighting coefficient associated with each data point depends on the performance of the previous classifiers. When all the classifiers have been trained, their predictions are combined based on a weighted majority voting rule as shown in Fig. 3.

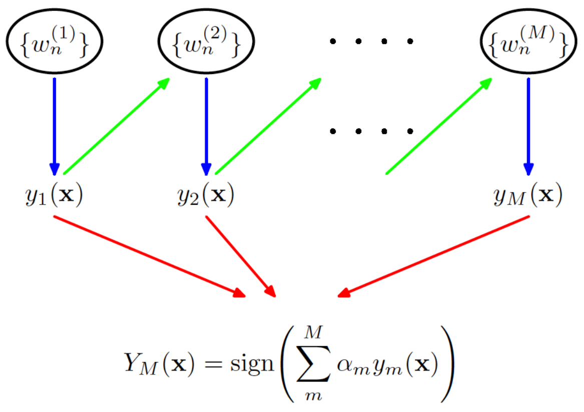

Figure 3: Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n(m)$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).

Consider a two-class classification problem, in which the training data comprises input vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ along with corresponding binary target variables $t_1, \dots, t_n$ where $t_n \in \{-1, 1\}$. Each data point is given an associated weighting parameter w_n, which is initially set $1/n$ for all data points. We shall suppose that we have a procedure available for training a base classifier using weighted data to give a function $y(\mathbf{x}) \in \{-1, 1\}$. At each stage of the algorithm, AdaBoost trains a new classifier using a data set in which the weighting coefficients are adjusted according to the performance of the previously trained classifier so as to give greater weight to the misclassified data points. Finally, when the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weight to different base classifiers. The basic steps of the AdaBoost method are presented at Alg. 1.

Algorithm 1 AdaBoost $(D_n = \{(\mathbf{x_i}, t_i)\}_{i=1}^n)$

```
Input:  Training dataset $(D_n = \{(\mathbf{x_i}, t_i)\}_{i=1}^n)$
          Base classifier $\textsc{Base}(\cdot,\cdot)$
          Number of base classifiers $T$
Output: Class label $t$ of $\mathbf{x}$

 1: $\mathbf{w}^{(1)} \leftarrow (1/n, \dots, 1/n)$        {Initial weights}
 2: for $t \leftarrow 1$ to $T$ do
 3:     $y_t \leftarrow \textsc{Base}(D_n, \mathbf{w}^{(t)})$        {Calling the base learner}
 4:     $\gamma_t \leftarrow \frac{\sum_{i=1}^n w_i^{(t)} I(y_t(\mathbf{x_i}) \neq t_i)}{\sum_{i=1}^n w_i^{(t)}}$        {$I(y_t(\mathbf{x_i}) \neq t_i)$ is the indicator variable, gives 1 when $y_i^{(t)} \neq t_i$}
 5:     $\alpha_t \leftarrow \ln\left\{\frac{1-\gamma_t}{\gamma_t}\right\}$        {coefficient of $y_t$}
 6:     for $i \leftarrow 1$ to $n$ do
 7:         $w_i^{(t+1)} \leftarrow w_i^{(t)} \exp\{\alpha_t I\big(y_t(\mathbf{x_i}) \neq t_i\big)\}$        {Re-weighting the points}
 8:     end for
 9: end for
10: return $Y_T(\mathbf{x}) = \sum_{t=1}^T \alpha_t y_t(\mathbf{x})$        {Make predicitons using the final model}
```

We see that the first base classifier $y_1(\mathbf{x})$ is trained using weighting coefficients $w^{(1)}$ that are all equal, which therefore corresponds to the usual procedure for training a single classifier. At step 7, the weighting coefficients $w^{(t)}$ are increased for data points that are misclassified and remains unchanged for data points that are correctly classified. Successive classifiers are therefore forced to place greater emphasis on points that have been misclassified by previous classifiers, and data points that continue to be misclassified by successive classifiers receive ever greater weight. The quantities γ_t represent weighted measures of the error rates of each of the base classifiers on the data set. We therefore see that the weighting coefficients α_t defined in step 5 of the algorithm give greater weight to the more accurate classifiers when computing the overall output given by equation of step 10.

4 Tasks to be done

We should implement and apply AdaBoost to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables[2].

- Fill in the code that implements the AdaBoost classifier given in Algorithm 1. As *base classifier*, we will use the decision tree offered by `scikit-learn`. The input to the algorithm is the number of decision trees T that will be used by the ensemble method and the depth of the decision trees (variable D). At the end of the process, we compute the training and test errors.
- Optimize the tree depth of AdaBoost. To do that, create a Python function that implements AdaBoost (adding your code from the previous question) and call it with different tree depths D (for simplicity, with $T = 100$ number of trees). Plot the final test error vs. the tree depth. What do you observe?

References

[1] Bishop, Christopher M. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., 2006.

[2]See `https://archive.ics.uci.edu/ml/datasets/Covertype` for details about the dataset.