Introduction to Machine Learning (67577)

Exercise 4
Boosting And SVM

May 2018

Contents

Submission guidelines **2**

SVM- Formulation **2**

SVM And Generalization **3**

Separate the Inseparable - Adaboost **4**

Face Classification (Bonus) **5**

Submission guidelines

- All answers to both theoretical parts and practical parts, should be submitted as a single pdf file.
- Each question is enumerated. Use those numbers when answering your questions.
- FIGURES: **Submit all the plots**. Your results should be self explanatory and readable: each figure should have a **unique informative title** and each axis should have a **meaningful label**. How to add title and axis labels? For instance, click here. Ambiguous graphs will result in point reduction for the whole answer.
- CODE: All of your Python code files used in the practical questions should be attached in the zip file. Note that if your code doesn't run on the CS environment you will not get points for this question. You allowed to use only numpy, unless we wrote something else explicitly.
- Please submit a script `ex4_runme.py` that runs your code, and generates the figures.

SVM- Formulation

1. (5 Points) Write the Hard-SVM as a QP problem. i.e. write the following problem:

$$\underset{(\mathbf{w},b)}{\operatorname{argmin}} ||\mathbf{w}||^2 \text{ s.t. } \forall i, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geqslant 1.$$

   in the following form:

$$\underset{\mathbf{v}\in\mathbb{R}^n}{\operatorname{argmin}}\left(\frac{1}{2}\mathbf{v}^\top Q\mathbf{v} + \mathbf{a}^\top \mathbf{v}\right)$$
$$\text{s.t. } A\mathbf{v} \leqslant \mathbf{d},$$

   where $Q \in \mathbb{R}^{n\times n}, A \in \mathbb{R}^{m\times n}, \mathbf{a} \in \mathbb{R}^n, \mathbf{d} \in \mathbb{R}^m$ are fixed vectors and matrices.

   Hint: Observe that: $\mathbf{w}^\top I\mathbf{w} = \|\mathbf{w}\|^2$. Another hint: in tirgul 4 we saw an example of 'Learning Halfspaces with ERM'.

   Why is it interesting? Once we formulated a problem as QP we're almost finished, since we can use generic algorithms in order to solve this kind of problems efficiently.

2. (Bonus - 3 Points) In the Soft-SVM we defined the problem:

$$\arg\min_{\mathbf{w},\{\xi_i\}} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i \text{ s.t. } \forall_i, y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \geqslant 1 - \xi_i \text{ and } \xi_i \geqslant 0$$

   Show that we can alternatively solve:

$$\arg\min_{\mathbf{w}} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{m}\sum_{i=1}^{m}\ell^{hinge}(y_i\langle \mathbf{w}, \mathbf{x}_i\rangle),$$

   where $\ell^{hinge}(a) = \max\{0, 1-a\}$, and get an optimal solution to the first formulation.

   Why is it interesting? In ex6 we will use this formulation in order to solve real world problems.

SVM And Generalization

In the lecture we claimed that SVM generalizes well. In this part we will show it empirically on a specific setting:

3. (10 Points) In order to see the quality of the SVM we need some baseline. For this purpose, we will **compare SVM to Perceptron** - a well known (naive) algorithm for finding separating a hyperplane is:

```
Batch Perceptron
input: A training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize: $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  for $t = 1, 2, \ldots$
    if $(\exists\ i$ s.t. $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0)$ then
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      output $\mathbf{w}^{(t)}$
```

Implement a perceptron. Please submit a file called perceptron.py with a class called perceptron that has the following methods:

(a) fit$(X, \mathbf{y})$ learns the weights $\mathbf{w} \in \mathbb{R}^d$ according to the perceptron algorithm. $X \in \mathbb{R}^{m \times d}$ is a matrix of m samples in $\mathbb{R}^d$ and $\mathbf{y} \in \{\pm 1\}^m$ are their classifications. See pseudocode above (replace the "for" loop by "while").

(b) predict$(\mathbf{x})$ that predicts the label for a sample $\mathbf{x}$ using the classe's current weight vector. The method returns either 1 or -1.

For the SVM classifier you will use sklearn's SVC library. When creating an object of this type there are settings you can specify, choose C=1e10, kernel='linear'. These will make your SVM learn a linear classifier (i.e. without using a kernel) that is very similar to the hard-SVM we talked about in class. Once you create the object, you can use the functions fit, score to complete the exercise.

We will now compare the SVM with the Perceptron on a specific setting. Define the distribution that generates the data points $\mathbf{x}$ to be $\mathcal{D} = \mathcal{N}(\mathbf{0}, \boldsymbol{I}_2)$, i.e. a two-dimensional Gaussian with mean vector of zeros and a unit matrix for covariance. The true labels are determined by $f(\mathbf{x}) = \text{sign}\left(\left\langle \begin{pmatrix} 0.3 \\ -0.5 \end{pmatrix}, \mathbf{x} \right\rangle + 0.1\right)$. Use numpy.random.multivariate_normal to draw the points.

Note: since we use non homogeneous hypothesis your perceptron should consider also non homogeneous hypotheses. So do not forget to add columns of ones to the data when you use the perceptron (the SVM add ones by itself).

4. (10 Points) For each $m \in \{5, 10, 15, 25, 70\}$, draw m training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. Plot the points, using a different color for each of the two labels. Add the true hypothesis, the SVM hypothesis, and the perceptron hypothesis (the hyperplane, not the vector) to your plot. Hand in the 5 plots, one for each m.

5. (15 Points) Consider the following procedure:

- Draw training points $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ from a distribution $\mathcal{D}$ and classify them according to a true hypothesis f (i.e. with labels $y_1, \dots, y_m$). **Note:** The training data should always have points from two classes. So if you draw a training set where no point has $y_i = 1$ or no point has $y_i = -1$ then just draw a new dataset instead, until you get points from both types.
- Draw k test points $\{\mathbf{z}_1, \dots, \mathbf{z}_k\}$ from the same distribution $\mathcal{D}$ and calculate their true labels as well.
- Train a Perceptron classifier and an SVM classifier on $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$.
- Calculate their accuracy (the fraction of test points that is classified correctly) on $\{\mathbf{z}_1, \dots, \mathbf{z}_k\}$.

For each $m \in \{5, 10, 15, 25, 70\}$, repeat the above procedure 500 times with $k = 10000$ and save the accuracies (or just keep the mean accuracy) of each classifier. Finally, hand in a plot of the mean accuracy as function of m with one curve for SVM and another for Perceptron.

6. (5 Points) Which classifier did better? why do you think that happened? No need for a formal argument, just explain what are the properties of the classifiers that cause these results.

Separate the Inseparable - Adaboost

In this part we will boost a Decision Stump classifier by Adaboost and analyze its performance.

- As we saw in class, one of the main motivations for boosting is computational complexity. Try to solve the following questions in a reasonable runtime. Two tips to help reducing runtime:
  - Try to **Avoid for loops**. If you are not familiar with the concept of vectorization in numpy, this is the time for you to read about it. You can find many tutorials on the web, for example **this**. (loops are allowed as long as your code runs in reasonable time).
  - You can use **line_profiler**. It is a tool that helps in finding the slower parts of your code. After you located the bottleneck try to think if you can accelerate it.
- Take a look at the module ex4_tools which contains some useful tools for this exercise.

7. (10 Points) The file adaboost.py contains a template for implementation of the adaboost classifier. Fill in the template and implement the Adaboost algorithm as learned in class, where you assume WL is a weak-learner class that can be used as follows:

   h = WL(D,X,y) - constructs a weak classifier trained on X, y weighted by D.
   h.predict(X) - returns the classifier's prediction on a set X.

   In ex4_tools you are provided with such an implementation for a Decision Stump classifier, called DecisionStump.

   You may add methods as you see fit, but please implement the functions that are declared in the template.

8. (10 Points) In ex4_tools you are provided with the function generate_data. Use it to generate 5000 samples without noise (i.e. noise_ratio=0). Train an Adaboost classifier over this data. Use the DecisionStump weak learner mentioned above, and $T = 500$. Generate another 200 samples without noise ("test set") and plot the training error and test error, as a function of T. Plot the two curves on the same figure.

9. (10 Points) Plot the decisions of the learned classifiers with $T \in \{5, 10, 50, 100, 200, 500\}$ together with the test data. You can use the function decision_boundaries together with plt.subplot for this purpose.

10. (5 Points) Out of the different values you used for T, find $\hat{T}$, the one that minimizes the test error. What is $\hat{T}$ and what is its test error? Plot the decision boundaries of this classifier together with the training data.

11. (Bonus- 4 Points) Look into the AdaBoost: Take the weights of the samples in the last iteration of the training (D^T). Plot the training set with size proportional to its weight in D^T, and color that indicates its label (again, you can use decision_boundaries). Oh! we cannot see any point! the weights are to small... so we will normalize them: D = D / np.max(D) * 10. What do we see now? can you explain it?

12. Repeat 8,9,10,(11- for the bonus) with noised data. Try noise_ratio=0.01 and noise_ratio=0.4.
    - (5 Points) Add all the graphs to the pdf.
    - (5 Points) Describe the changes.
    - (5 Points) Explain 8 in terms of the bias complexity tradeoff.
    - (5 Points) Explain the differences in 10.

Face Classification (Bonus)

In this last section we will try to tackle a computationally 'heavy' problem. In order to solve it in a reasonable runtime, we should take into account some technical aspects of the program, as we did earlier. Where mentioned, you should avoid for loops altogether.

In the tirgul we learned about The Viola Jones object detection framework. In this framework we use a weak learner algorithm which returns hypotheses based on Haar features. The Haar features that the algorithm uses are based on the 4 images A, B, C, D at different scales and locations:

A Haar feature is defined by:

$$f(I) = \sum_{(x,y)\in\text{White Spaces}} I(x,y) - \sum_{(x,y)\in\text{Black Spaces}} I(x,y)$$

where $I(x, y)$ is the value of the pixel in location (x, y).

A Haar feature can be calculated efficiently by using a mechanism called Integral Image. Given an input Image, the integral image is an image created from it in which the value of the pixel at any (x, y) location equals the sum of the values of all pixels that are before (above or to the left of) it in the original image.

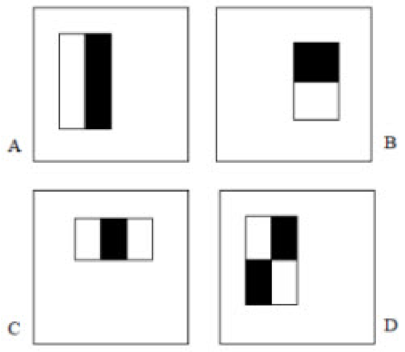

Figure 1: Haar features

5	4	3	8	3
3	9	1	2	6
9	6	0	5	7
7	3	6	5	9
1	2	2	8	3

5	9	12	20	23
8	21	25	35	44
17	36	40	55	71
24	46	56	76	101
25	49	61	89	117

Figure 2: Image on the left and the integral image on the right.

13. (Bonus - 2 Points) Show how to calculate Integral Image i.e. $S(a,b) = \sum_{x=1}^{a} \sum_{y=1}^{b} I(x,y)$ in $O(n)$, where n is the amount of pixels in the image (write a pseudocode).

14. (Bonus - 2 Points) Given Integral Image, show how to calculate the sum over a square in $O(1)$.

15. (Bonus - 1 Points) Deduce how to calculate a Haar feature in $O(1)$ by Integral Image.

16. (Bonus - 6 Points) In this question we are going to implement and boost a weak learner for the problem of face / not face classification.

    The data for this section is provided in the directory faces - it contains images of face in the sub directory FACES and other images in the sub directory NFACES.

    The file face_detection.py contains a template for a weak learner of the problem. You may add methods as you see fit, but please implement the empty functions that are declared in the template:

    (a) Use your answer from q1 in the theoretical part in order to implement the function 'integral_image(images)'.

    (b) Use your answer from q2 in the theoretical part in order to implement the function 'sum_square(integrals, up, left, height, width)'. **Avoid for loops.**

    (c) Use your answer from q3 in the theoretical part in order to implement the function 'kernel_k(self, integrals, up, left, height, width)' for k=b,c,d (see picture of the kernels in the theoretical part) take a look at 'kernel_a' for example. **Avoid for loops.**

    Our weak learner iterates over all possible kernels and for each kernel it calculates the feature value for all images. (the functions: 'train', 'evaluate_kernels_performance', 'evaluate_kernel', read them and make sure you understand!). Each feature is configured by the following parameters:

- 'kernel'- one type from the 4 kernel functions described above.
- 'up'- the upper limit of the upper-left rectangle of the kernel
- 'left'- the left limit of the upper-left rectangle of the kernel
- 'height'- the height of rectangle in the kernel
- 'width'- the width of rectangle in the kernel
- 'sign'- whether the upper-left rectangle of the kernel is white or black.

The kernel function just returns a scalar (the feature value) for each sample. We should now find a decision stump hypothesis for each feature according to the labels. In order to do that efficiently we will use the algorithm from the Tirgul on Boosting.

(d) Use example 5 from the Tirgul on Boosting in order to implement the function evaluate_feature_performance(self, kernel, feature_values, weights, labels, up, height, left, width, sign). **Take a look at the 'find_threshold' function in 'ex4_tools' if you are not sure how to avoid the for loop.**

(e) Implement the function 'predict(self, integrals)'. **Avoid for loops.**

17. (Bonus - 6 Points) Use the function 'train_images, test_images, train_labels, test_labels = load_images(path_to_data)' from the : 'ex4_tools.py' file in order to load the images. Train an Adaboost classifier over the training set. Use the WeakImageClassifier that you implemented, and $T = 50$. Plot the training error and test error, as a function of T (observe that after you trained adaboost with T classifiers you can predict with any number of classifiers $t < T$) plot both curves on the same.

18. (Bonus - 6 Points) Use the DecisionStump and train an Adaboost classifier over the same data (again $T = 50$). Plot the training error and test error, as a function of T (observe that after you trained adaboost with T classifiers you can predict with any number of classifiers $t < T$) plot both curves on the same figure. Compare to the previous results- which is better? explain both in terms of train error and test error.