E0-270 - Machine Learning
Assignment 2

Submission Deadline: February 26, 2018, 8 pm

General Instructions

1. Follow the folder structure that is given in the accompanying material.
2. The code has to be submitted via Canvas, submit a single .zip file.
3. The report has to be submitted in room 205.
4. Do not use MATLAB. We wanted to introduce you to PyTorch and hence the entire assignment (including the code template) is in Python.
5. Do not change function definitions in the code template unless asked otherwise.

Question 1: (Support Vector Machines)

(a) Generating Synthetic Data

Nothing beats the fun of playing around with synthetic data. Synthetic data not only provides you with an easy problem to start with, it also helps in gaining an insight into the behaviour of the algorithm that will operate on this data. Edit `Q1/ans1a.py` to generate 1000 data points, $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{1000}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^2$ and $y^{(i)} \in \{+1, -1\}$ using `f = dummy_fn`. Plot the data and use different colors to differentiate between points belonging to different classes. Is the data linearly separable? **5 Marks**

(b) Implementing Hard SVM

Edit `Q1/ans1b.py` to implement the `svm_train` and `svm_predict` functions. The task is to implement a hard SVM (as explained in class). Generate 1000 data points as in part (a) and train your SVM with linear kernel using 80% of this data and test on the remaining 20% data. Report the training and test accuracy. Plot the decision regions (using the function provided in `Q1/ans1b.py`) and submit the plot. Note that for solving the dual problem, you should use the `cvxopt` package as explained in `Q1/ans1b.py`. **10 Marks**

(c) Implementing Soft SVM

Copy your solution for part (b) to create the file `Q1/ans1c.py`. Modify `svm_train` and `svm_predict` functions to implement a soft SVM. The function `svm_train` should now take a new parameter C as input for soft SVM. Generate synthetic data as in part (a). Now do the following:

```
for $C \in \{0.1, 1, 10\}$ do
  for $p \in \{0, 0.1, 0.3\}$ do
    Flip each training label in $Y$ with probability $p$ to obtain $\hat{Y}$
    Train SVM using training data $X$, training labels $\hat{Y}$ and $C$ (linear kernel)
    Report accuracies on original training labels $Y$ and test set against $C$ and $p$
  end for
end for
```

5 Marks

(d) Generating More Complex Synthetic Data

More complicated synthetic data can be generated by providing different functions as input to `data_gen` function in `Q1/ans1a.py`. Edit `Q1/ans1d.py` to write one such function: `complex_fn`. `complex_fn` should give an output $+1$ if the norm of input x is less than 3 and -1 otherwise. Generate 1000, two-dimensional synthetic data points using this function. Plot the data and use different colors to differentiate between points belonging to different classes. Is the data linearly separable? **5 Marks**

(e) Using Kernels

Implement the `rbf_kernel` function in `Q1/ans1e.py`. Generate 1000 synthetic data points as in part (d) and train the SVM using implementation from part (b) on 80% of the data (the remaining 20% is test set). Report the accuracies on training and test set. Show the decision regions. **5 Marks**

(f) Dealing with Real Data

Create a new file `Q1/ans1f.py`. Use the spam classification dataset given in `Q1/data`, to train your SVM implementation from part (c). Use RBF kernel (from part (e)). Use 5-fold cross validation to select the right value of C from the set $\{0.01, 0.1, 1, 10, 100\}$. Report accuracies on each fold for different values of C in a table. Using the chosen value of C retrain the model on full training set given in `Q1/data/SpambaseFull` and test it on the test set given in the same folder. Report the training and test accuracy. As an optional-ungraded challenge, you might want to explore other kernels for this problem. **10 Marks**

Question 2: (Neural Networks)

You will probably never write backpropagation from scratch in practical applications (thanks to automatic differentiation!). However, it is important to understand how it works and why it works because this will provide an insight into the learning process of neural networks. In this question, you will be guided through the process of writing backpropagation for a general fully connected neural network.

(a) Forward Pass

We will begin by writing a class that implements a single linear layer. Edit `Q2/ans2a.py` to complete the implementation of class: `Linear` and functions: `sigmoid`, `sigmoid_grad`. In this part you are only supposed to implement the forward pass. **0 Marks**

(b) Backward Pass

Edit `Q2/ans2b.py` to complete the implementation of `backward` and `step` function. You have to implement the gradient calculation for a single fully connected layer. Ignore regularization for now. Following the notation from [1], each layer should take, $\delta^{(l+1)}$, which is the error associated with its output to compute the required gradients and $\delta^{(l)}$, the error associated with its input. The superscripts associated with δ might change depending on how you index your layers. **0 Marks**

(c) Putting it Together

Edit `Q2/ans2c.py` to complete the implementation of a single hidden layer neural network as explained in the comments. Run `Q2/ans2c.py`, if everything till now was correct you will see the message - "Grad-Check Successful". Your code should pass the gradient check. Attach a screenshot showing this. What do you think the `__main__` function is doing? **20 Marks**

(d) Training on Real Data

Create a file `Q2/ans2d.py`. Use the network class written in `Q2/ans2c.py` to create a network instance. Load the training and test data from `Q1/data/SpambaseFull` and train the neural network using this data. Plot the cost incurred (on y-axis) with respect to the number of iterations (on x-axis). Also report the final training and test set accuracy. Write briefly about the approach that you used to decide when to stop training the network.
. **10 Marks**

(e) Using PyTorch

Install PyTorch [2]. Since PyTorch has auto-grad, you do not need to implement backpropagation this time. Train a neural network with the same architecture and data as in part (d) using PyTorch. Study about regularization and use l_2-regularization in your network. Experiment with different values of regularization parameter λ in

[1] http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm
[2] http://pytorch.org/

the set $\{0.0001, 0.001, 0.01, 0.1, 1\}$. Plot the test and train accuracies for different values of λ (take λ values on log scale while plotting) in the same plot. In this example you might not see a big effect, but in general regularization is very useful. **10 marks**

Question 3: (Cost Sensitive Binary Classification)

Consider a cost-sensitive binary classification task in which misclassifying a positive instance as negative incurs a cost of $c \in (0, 1)$, and misclassifying a negative instance as positive incurs a cost of $1-c$. This can be formulated as a learning task with instance space $\mathcal{X}$ and label and prediction spaces $\mathcal{Y}$ and $\hat{\mathcal{Y}}$. The cost sensitive loss is defined as:

$$l_c(y, \hat{y}) = \left\{ \begin{array}{ll} 1-c, & \text{if } y = -1 \text{ and } \hat{y} = 1 \\ c, & \text{if } y = 1 \text{ and } \hat{y} = -1 \\ 0, & \hfill \text{otherwise} \end{array} \right\}$$

Let D be a joint probability distribution on $\mathcal{X} \times \{\pm 1\}$. Let μ be the marginal distribution on $\mathcal{X}$ obtained from D. Denote by $\eta(x)$, the conditional probability $P(y = 1|x)$. For a classifier $h : \mathcal{X} \longrightarrow \{\pm 1\}$, the expected loss can be calculated as:

$$L^c_D[h] = \mathbb{E}_{(x,y)\sim D}[l_c(y, \hat{y})]$$

Derive the Bayes optimal classifier in this setting, i.e. a classifier $h^* : \mathcal{X} \longrightarrow \{\pm 1\}$, such that:

$$L^c_D[h^*] = \inf_{h:\mathcal{X}\longrightarrow\{\pm 1\}} L^c_D[h]$$

How does your derivation change if the loss incurred on misclassifying a positive instance as negative is a and for misclassifying a negative instance as positive is b for some arbitrary $a, b > 0$? **10 Marks**

Question 4: (Perceptron)

Implement the perceptron algorithm to classify the dataset given in `Q1/data/SpambaseFull`. Run the algorithm on the training set for 100 iterations (each iteration involves going over all the examples in the training set once).

1. Plot the number of mis-classifications in the training set as a function of the number of iterations.
2. Report the final training and test accuracy along with all the hyper-parameters that were used by you (if any).
3. Would the graph be different if you shuffle the training set before running perceptron on it? How about the case when the features are shuffled in some fixed order across examples?
4. Look at the weights assigned by perceptron to each feature after training. Are all the weight values comparable? If no, can you list the top 5 and bottom 5 features[3] ordered according to the weight assigned to them by perceptron.

10 Marks

[3]Feature descriptions can be found at: https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.names