CS 461: Machine Learning Principles
Fall 2021

Problem Set #3 (80 Points)

Out: Tuesday, October 19

Due: Tuesday, November 2, 8:00pm

Instructions

How and what to submit? Please submit your solutions electronically via Canvas. Please submit two files:

1. A PDF file with the written component of your solution including derivations, explanations, etc. You should create this PDF in LaTeX. Please name this document ⟨`firstname-lastname`⟩`-sol3.pdf`.

2. The empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures) in a Jupyter notebook file. Rename the notebook ⟨`firstname-lastname`⟩`-sol3.ipynb`.

Late submissions: there will be a penalty of 20 points for any solution submitted within 24 hours past the deadline. No submissions will be accepted past then.

What is the required level of detail? See below.

(a) When asked to **derive** something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations.

(b) When asked to **plot** something, please include in the `ipynb` file the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or in a caption.

(c) When asked to **provide a brief explanation or description**, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide it precisely (and accompany by succinct wording, if appropriate).

(d) When **submitting code (in Jupyter notebook)**, please make sure it's reasonably well-documented, runs, and produces all the requested results. If **discussion** is required/warranted, you should include it directly in the notebook (using the markdown cell).

Collaboration policy: collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, and (2) specify names of student(s) you collaborated with in your writeup.

1 Support Vector Machines (SVMs)

In this section, we will consider a kernelized SVM (with bias). It assumes a choice of kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ corresponding to the inner product in some feature space $\mathcal{F}$. That is, $K(x, x') = \langle \phi(x), \phi(x') \rangle$ where $\phi : \mathbb{R}^d \to \mathcal{F}$ is an implicit feature function. Given training data $(x_1, y_1) \dots (x_N, y_N) \in \mathbb{R}^d \times \{\pm 1\}$, we solve the soft SVM objective

$$\begin{aligned} w^*, b^*, \xi^* = \underset{w \in \mathbb{R}^d,\, b \in \mathbb{R},\, \xi \in \mathbb{R}^N}{\arg\min} \; & \frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i(\langle w, \phi(x_i) \rangle + b) \geq 1 - \xi_i && \forall i = 1 \dots N \\ & \xi_i \geq 0 && \forall i = 1 \dots N \end{aligned}$$

In the lecture, we saw that we can instead solve the following dual problem

$$\begin{aligned} \alpha^* = \underset{\alpha \in \mathbb{R}^N}{\arg\max} \; & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C && \forall i = 1 \dots N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{1}$$

where it holds that $w^* = \sum_{i=1}^{N} \alpha_i^* y_i \phi(x_i)$. The beauty of the kernel trick is that we never have to work with the parameter/features directly either in training or inference. Given a new point $x \in \mathbb{R}^d$, the prediction is made by

$$\begin{aligned} \hat{y} = \textbf{sign}\left(\langle w^*, \phi(x) \rangle + b^*\right) &= \textbf{sign}\left(\left\langle \sum_{i=1}^{N} \alpha_i^* y_i \phi(x_i), \phi(x) \right\rangle + b^*\right) \\ &= \textbf{sign}\left(\sum_{i=1:\, \alpha_i^* > 0}^{N} \alpha_i^* y_i K(x_i, x) + b^*\right) \end{aligned} \tag{2}$$

Problem 1 **[20 points]**

The objective (1) is an instance of constrained quadratic program, which can be "fed" to an off-the-shelf quadratic program solver. These solvers are usually constructed to handle certain standard formulations of the objective and constraints. The **canonical form** of a quadratic program with linear constraints is (the vector inequality and equality here are elementwise):

$$\begin{aligned} u^* = \underset{u \in \mathbb{R}^M}{\arg\min} \; & v^\top u + \frac{1}{2} u^\top H u \\ \text{subject to} \quad & Au \leq a \\ & Bu = b \end{aligned} \tag{3}$$

where M is the number of variables, $v \in \mathbb{R}^M$ and $H \in \mathbb{R}^{M \times M}$ specify the objective, $A \in \mathbb{R}^{L \times M}$ and $a \in \mathbb{R}^L$ specify L inequality constraints, and $B \in \mathbb{R}^{T \times M}$ and $b \in \mathbb{R}^T$ specify T equality constraints. Precisely specify the values of v, H, A, a, B, b so that the solution u^* in (3) is equal to the solution α^* in (1).

End of problem 1

Problem 2 **[10 points]**

Assume that you have solved the objective (1). But this only gives you $\alpha^* \in \mathbb{R}^N$: to make the prediction (2) we need the value of $b^* \in \mathbb{R}$ as well. Show that we can in fact recover b^* from α^* and the data. You must give an exact formula for b^*. You may assume that there exists some training point $k \in \{1 \dots N\}$ such that $0 < \alpha_k^* < C$.

End of problem 2

2 Twitter Sentiment Analysis with SVMs

Binary sentiment classification In this section we will consider the problem of predicting *sentiment* from tweets. We will formulate this as a binary classification problem where one class consists of positive or neutral tweets and the other class consists of negative tweets.

Dataset The tweets are regarding US airline service quality.[1] The provided dataset of a few thousand tweets has been manually labeled to reflect these binary sentiment labels.

Vector representation of input text We will represent each tweet by a feature vector based on word occurrences. This representation for documents is sometimes called "bag of words" since it discards the ordering of words and treats them as interchangeable "tokens" in a bag (document). To skip non-trivial engineering issues, we will rely on existing packages to do the low-level processing and feature extraction for us. The details are addressed in the notebook.

Problem 3 **[10 points]**
Fill in missing pieces of the linear SVM code, specifically the `forward` function of `LinearSVM`. Pass the unit test. Generate figures (try various hyperparameter values).

End of problem 3

[1] As may be expected for this topic, the data contains some unsavory expressions, left unfiltered.

Problem 4 **[20 points]**

Fill in missing pieces of the kernel SVM code, specifically in `construct_kernel` and in the margin/update computation of `pegasos_kernelized`. If your implementation is correct, you should be able to fit nonlinear data almost perfectly. Generate figures (try various hyperparameter values).

End of problem 4

Problem 5 **[20 points]**

Using your code, train your tweet classifier on the training set (`train.csv`) and tune on the validation set (`val.csv`).

- Experiment with the basic linear SVM classifier; feel free to experiment with hyperparameters (training duration, regularization).
- (Optional) Experiment with kernel SVM: Given the high feature dimensionality of our primitive text processing, we do not recommend using kernel SVM here. It could take a long time to train, unless you find a way to reduce the feature dimensionality.
- What is your best *val* performance? What hyperparameter values lead to that?
- Evaluate your single best model (based on *val*) on the development-test set (`devtest.csv`).
- Do you see a large gap between the *val* and *devtest* accuracies? (They are drawn from the same distribution, so if you see sizable differences, that might be due to overfitting to the validation set in your hyperparameter tuning.)

Write your answers directly in markdown cells (and clearly mark that these are your answers).

End of problem 5