Applied Machine Learning HW3 (20%)

Due Wed 5/30 @ 11:59pm on Canvas

Instructions:

1. This homework is about SVMs, and it should be done using `numpy` and `sklearn`, both of which are installed on ENGR machines (`ssh access.engr.oregonstate.edu` or `ssh pelican.eecs.oregonstate.edu`). Pelican machines are considerably faster; give it a try if you find your own machines too slow for SVM.
2. Use the same data from HW1. We've provided a reference `hw1.py` for the basic part of HW1 online.
3. You should submit a single `.zip` file containing `hw3.pdf`, `income.test.predicted`, and all your code. Again, LaTeX'ing is recommended but not required.

0 Conceptual Questions (assume hard-margin SVM)

1. How many support vectors can there be for SVM in d dimensions? (start with $d = 1, 2, 3$)
2. State two equivalent formulations for SVM: (a) maximize geometric margin, given functional margin at least 1; (b) minimize weight norm square, given functional margin at least 1.
3. What will happen if (a) we replace the minimum functional margin of 1 by 10 (in the constraint)? (b) we require the geometric margin to be at least 1?
4. True or False? a) if an example has functional margin of 1, it must be in the support vector set. b) if an example is in the support vector set, it must have a functional margin of 1.
5. List two reasons why the convex hull approach is not used to solve SVMs in practice.
6. For a separable dataset, how many support vectors can there be for perceptron? What are the α values for the support vectors in perceptron?
7. What is the leave-one-out cross-validation bound for SVM? (Hint: related to the number of support vectors and the size of the training set) Why does it make sense?

1 Starting Point (from HW1)

You can re-use your HW1 code on feature maps. Let's start with the very basic fully binarized feature map, which resulted in 231 *observed* features (including the bias dimension) on `income.train.txt.5k`. Running perceptron for 5 epochs (with everything else also being default) should result in these train/dev error rates (here only showing the averaged versions, and only showing the best epoch):

```
avg_perc epoch 4 updates: 1170 (23.4%) training_err 16.1% (+:20.3%) dev_err 14.7% (+:19.3%)
```

Remember (from HW1) that the update ratio (23.4%) is different from training error (16.1%) in two ways: (1) the updates are decided by the unaveraged weights while the training error (and dev error) are measured using averaged weights, and (2) more importantly, the updates are decided by an *ever-changing* weight vector during an epoch (like a running average), while training error is evaluated at the end of an epoch, using a single (averaged) weight vector. In practice, we often use something similar to update ratio or running average rather than the exact training error, because the latter involves another pass over the training set, which slows down

training by 2x, and this was the reason why we did not ask you to report training errors in HW1. But the training data here is relatively small so we could afford to report them if we want.

If your numbers are exactly the same as the above, you can use your HW1 code, otherwise it's better to use my very simple reference `hw1.py` (on the course homepage). The TA will release a more comprehensive solution that includes various feature engineering but we won't use those here, because (see below) the kernel trick does (most of the) feature engineering for you.

2 Basic (Linear) SVM from `sklearn`

1. Train SVM using `sklearn.svm.SVC` on the 5k training set, using linear kernel (`kernel='linear'`) and the default $C = 1$. What's your training and dev error rates?

   Hint: the train error should be better than avg perceptron, but the dev error should be slightly worse.

   Note: this training might take a while depending on the computer. It took about 4 secs on my laptop (2015 MacBook Pro) and about 3 secs on `pelican` machines but about 22 secs on `flip` machines (default ENGR machines if you `ssh access.engr.oregonstate.edu`)! These `flip` machines are not meant for computing jobs; use `pelican` instead (`ssh pelican.eecs.oregonstate.edu`) if you want.

   Also report the time it takes to train.

2. How many support vectors are there? (using `clf.n_support_`)

   Find another way to verify your result (Hint: $\alpha > 0$).

3. How many support vectors have $\alpha = C$?

   How many support vectors have margin violations (i.e., non-zero slacks)? Use $\xi > 0.001$, i.e., (functional) margin is less than 1: $y(\mathbf{x} \cdot \mathbf{w}) < 1 - 0.001$.

   How many support vectors are misclassified?

   How many examples have the exact (functional) margin of 1? Use $|y(\mathbf{x} \cdot \mathbf{w}) - 1| \leq 0.001$.

   Do the results make sense?

4. Calculate the total amount of margin violations (i.e., slacks): $\sum_i \xi_i$ and the objective: $\frac{1}{2}\mathbf{w}^2 + C\sum_i \xi_i$.

5. For both (positive, negative) classes, list the top five most violated training examples and their slacks.

6. C is the only hyperparameter to tune on dev set. Vary your C: 0.1, 0.5, 1, 2, 4, 8, 20, 50.

   Report for each C: (a) training time, (b) training error, (c) dev error, and (d) number of support vectors.

7. Do you observe any trends from these? Are these trends consistent with the theory?

8. Make a plot from the above, with x-axis being C (**in a log scale**), and y-axis being training and dev error rates (i.e., two curves on one plot).

9. Make a plot of training time for 5, 50, 500, and 5000 training examples (x-axis: number of examples, y-axis: seconds) (fixing $C = 1$). Can you figure out the time complexity of default SVM training algorithm (with respect to the number of training examples) from this figure by curve fitting?

10. What if you change C to a very big number, e.g., 1000000? Why?

    Generally speaking, with increasing C, does training slow down? Explain.

3 Quadratic Kernels

1. In `sklearn`, replace linear kernel with quadratic kernel (`kernel='poly', degree=2, coef0=1`).

   Train with $C = 1$. Report the training time, and train/dev error rates.

2. Why do we need to set `coef0=1`?

3. Tune C in a way you see fit, up to $C = 10,000$. Report for each C: (a) training time, (b) training error, (c) dev error, and (d) number of support vectors.

4. Do you observe any trends from these? Are these trends consistent with the theory?

5. Make a plot from the tuning results (again, x-axis is C, in log-scale, and y-axis are train/dev errors).

6. What differences do you observe between linear and quadratic kernels, esp. when C is large? Hint: think about the training speed, training error, and dev error.

7. Why you can't access `clf.coef_` any more?

4 Final

Collect your best model and predict on `income.test.txt` to `income.test.predicted`. The latter should be similar to the former except that the target (`>=50K, <50K`) field is added.

Q: what's your best error rate on dev, and which algorithm (and settings) achieved it?
Q: what's your % of positive examples on dev and test according to this best model?

Debriefing (required):

1. Approximately how many hours did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did you work on it mostly alone, or mostly with other people?
4. How deeply do you feel you understand the material it covers (0%–100%)?
5. Any other comments?