Machine Learning HW2 (10%, in groups of 2 or 3)

Instructor: Liang Huang

Due Sun Oct 29 @ 11:59pm

Instructions:

1. Singleton groups are not allowed. Please reply to the Canvas discussion thread to find teammates.
2. This HW should be done using `numpy` and `sklearn`, both of which have been installed on ENGR machines (`ssh access.engr.oregonstate.edu` or `ssh pelican.eecs.oregonstate.edu`). If you'd like to install them on your own computers, we suggest installing `anaconda` which comes with both packages.
3. Use the same data from HW1. However, since SVM training (by default) is *much* slower than perceptron/MIRA, let's just use the first 5,000 training examples (dev/test remain unchanged):

   ```
   head -5000 income.train.txt > income.train.txt.5k
   ```

4. You should submit a single `.zip` file containing `hw1.pdf`, `income.test.predicted`, and all your code. Again, LaTeX'ing is recommended but not required.

0 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 and Aggressive MIRA (0.9) for 5 epochs on these 5,000 examples (with everything else also being default) should result in these train/dev error rates (here only showing the averaged versions):

```
perceptron avg: train_err 16.16% (+:20.3%)   dev_err 16.11% (+:19.5%) at epoch 3.40
AMIRA 0.9  avg: train_err 16.38% (+:21.0%)   dev_err 16.05% (+:18.9%) at epoch 0.60
```

I confirmed that the error rates from the unaveraged versions are slightly worse than those trained on the full training set in HW1, but the averaged versions are almost identical (which is reasonable, given that both averaged perceptron/MIRA achieved their best error rates before a full epoch on HW1). So we will use the smaller 5k training set instead of the whole HW1 set for faster SVM training.

If your numbers are too far from these, you may use my reference HW1 code (on course homepage).

1 Basic (Linear) SVM from `sklearn`

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

   Note: the train error rate should be better than averaged perceptron/AMIRA, and the dev error rate should be similar to averaged AMIRA.

   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? (`clf.n_support_`)

   How many among them have margin violations (i.e., non-zero slacks)?

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

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

5. C is the only hyperparameter to tune on dev set. Vary your C: 0.01, 0.1, 1, 2, 5, 10.

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

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

   Note: I observed $C = 5$ resulting in the best dev error rate.

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

8. 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?

2 Online SVM (Pegasos)

1. The default SVM solver is dual and batch and you have seen how slow it is. Now implement the very simple primal and online (sub-gradient descent) Pegasos algorithm (from the slides and extra reading on the course website), for linear kernel and $C = 1$. The update rule is very similar to perceptron/MIRA.

   Let it run until (reasonable) convergence (until the objective does not change significantly or until max. epoch). For each epoch, report (a) objective, (b) training error, (c) dev error.

2. Does the objective eventually converge to the one returned by `sklearn`? Is yours faster or slower?

3. Make two plots from the above experiment: the objective vs. epoch, and train/dev error rates vs. epoch.

4. How many support vectors are there in the final model? How does it compare with `sklearn`

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

   Note: I observed (significantly) better train/dev error rates than linear kernel by tuning C.

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

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

6. Can you adapt Pegasos for quadratic kernels? (Just describe the algorithm, but no need to implement it!)

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?