CS 5350/6350: Machine Learning Fall 2018

Homework 5

Handed out: 20 November, 2018
Due date: 6 December, 2018

General Instructions

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 10 pages**. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework on Canvas.

1 Logistic Regression

We looked Maximum A Posteriori (MAP) learning of the logisitic regression classifier in class. In particular, we showed that learning the classifier is equivalent to the following optimization problem:

$$\min_{\mathbf{w}} \left\{ \sum_{i=1}^{m} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)) + \frac{1}{\sigma^2} \mathbf{w}^T \mathbf{w} \right\}$$

In this question, you will derive the stochastic gradient descent algorithm for the logistic regression classifier.

1. [5 points] What is the derivative of the function $g(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$ with respect to the weight vector?

$$\begin{aligned} \frac{dg(\mathbf{w})}{d\mathbf{w}} &= \frac{\exp(-y_i \mathbf{w}' x_i)}{1 + \exp(-y_i \mathbf{w}' x_i)} (-y_i x_i) \\ &= \frac{-y_i x_i}{1 + \exp(y_i \mathbf{w}' x_i)} \end{aligned}$$

2. [5 points] The inner most step in the SGD algorihtm is the gradient update where we use a single example instead of the entire dataset to compute the gradient. Write down the objective where the entire dataset is composed of a single example, say $(\mathbf{x}_i, y_i)$. Derive the gradient with respect to the weight vector.

   The objective:

$$J(\mathbf{w}) = \frac{1}{\sigma^2}\mathbf{w}'\mathbf{w} + \log(1 + \exp(-y_i\mathbf{w}'\mathbf{x}_i))$$

   Taking derivative

$$\nabla J(\mathbf{w}) = \frac{2}{\sigma^2}\mathbf{w} - \frac{y_i\mathbf{x}_i}{1 + \exp(y_i\mathbf{w}'\mathbf{x}_i)}$$

3. [10 points] Write down the pseudo code for the stochastic gradient algorithm using the gradient from previous part.

   Hint: The answer to this question will be an algorithm that is similar to the SGD based learner we developed in the class for SVMs.

   (a) Initialize $\mathbf{w} \in \mathbb{R}^J$

   (b) For epoch = 1...T:

      i. Shuffle the training set

      ii. For each traning example $(\mathbf{x}_i, y_i)$

$$\mathbf{w}^{t+1} \leftarrow (1 - \tfrac{2r}{\sigma^2})\mathbf{w}^t + \tfrac{ry_i\mathbf{x}_i}{1+\exp y_i\mathbf{w}'x_i}$$

   (c) Return $\mathbf{w}$

2 Experiments

For this question, you will have to implement and compare different learning strategies: SVM, logistic regression (from your answer to the previous question), the naive Bayes classifier, and a variant of random forests that combines SVMs and decision trees.

2.1 The task and data

The data for this homework is adapted from the UCI credit card dataset. The goal is to predict whether a bank customer will default on their credit card payment. For more details about the data, see

Yeh, I. C., & Lien, C. H. (2009). *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients.* Expert Systems with Applications, 36(2), 2473-2480.

We have transformed the original features into a collection of binary features and have split the data into the usual training, testing and cross-validation splits. The data file contains:

1. `train.liblinear`: The full training set, with 20,000 examples.
2. `test.liblinear`: The test set, with 10,000 examples.
3. To help with cross-validation, we have split the training set into five parts `training00.data` - `training04.data` in the folder `CVSplits`.

All the data files are in the same liblinear format as we have used in the previous homeworks.

2.2 Implementation and Evaluation Notes

Each algorithm has different hyper-parameters, as described below. Use 5-fold cross-validation to identify the best hyper-parameters as you did in the previous homework.

The positive and negative labels are not balanced in this data. With such unequally distributed labels, we usually measure precision, recall and F-scores because the accuracy of a classifier could be misleading.

To compute these quantities, you should count the number of true positives (that is, examples that your classifier predicts as positive and are truly positive), the false positives (i.e, examples that your classifier predicts as positive, but are actually labeled negative) and the false negatives (i.e., examples that are predicted as negative by your classifier, but are actually positive).

Denote true positives, false positive and false negative as TP, FP and FN respectively. The precision (p), recall (r) and f-value F_1 are defined as:

$$\begin{aligned} p &= \frac{TP}{TP+FP} \\ r &= \frac{TP}{TP+FN} \\ F_1 &= 2\frac{p \cdot r}{p+r} \end{aligned}$$

For all your classifiers, you should report measure precision, recall and F_1. During cross-validation, use the average F_1 instead of average accuracy.

2.3 Algorithms to Compare

1. [15 points] **Support Vector Machine**

   Implement the simple stochastic sub-gradient descent version algorithm SVM as described in the class. Assume that the learning rate for the t^{th} epoch is

$$\gamma_t = \frac{\gamma_0}{1+t}$$

For this, and all subsequent implementations, you should choose an appropriate number of epochs and justify your choice. One way to select the number of epochs is to observe the value of the SVM objective over the epochs and stop if the change in the value is smaller than some small threshold. You do not have to use this strategy, but your report should specify the number of epochs you chose.

Hyper-parameters:

(a) Initial learning rate: $\gamma_0 \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

(b) The regularization/loss tradeoff parameter: $C \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

Implementation & Discussion:

(a) Language/Package: Python + Pandas + Numpy

(b) Details: Since the dimension is manageable, I use numpy.arrays to represent features. The training set is stored as a pandas.DataFrame, shape = (n_smaples, 2), columns =['y', 'X']. Example i is labeled +1 if $w'x > 0$. For each set of γ and C, I tried epoch from 1 to 50 with a step of 2. I would like to try a more granular step but the computation time is prohibitive.

(c) Discussion. I find that SVM/Logistic performs generally worse than NB and SVMTrees, and removing regularization (or set it to a small number) and increasing C improve the F score. It's largely due to that the dataset is highly non-linearly separable, where the model tend to always predict "-1". Therefore, to make the model more learnable, we need to increase C, i.e., increase the model's tolerance for "bad points" (the "+1" points). In my experiments a large γ and a large C give me the best shoot. Actually if I keep increasing C to exceed 10, I'll get better performance, though we're not required to do so in the homework.

2. [15 points] **Logistic regression**

Implement the Logistic Regression learner based on your algorithm in the Question 3.

Hyper-parameters:

(a) Initial learning rate: $\gamma_0 \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

(b) Tradeoff: $\sigma^2 \in \{10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^4\}$

Implementation & Discussion:

(a) Language: Language/Package: Python + Pandas + Numpy

(b) Details: Since the dimension is manageable, I use numpy.arrays to represent features. The training set is stored as a pandas.DataFrame, shape = (n_smaples, 2), columns =['y', 'X']. I iterate over each row to update the weight. Example i is labeled +1 if $\Pr(+1|X) > 1/2$. I didn't try parameter combination if $2\gamma/\sigma^2 <= 1$, because in this case the weights will go infinity.

(c) Discussion: Based on the discussion in the SVM model above, a small regularizer $(1 - \frac{2\gamma}{\sigma^2})$ and a big loss $(\frac{\gamma y_i \mathbf{x}_i}{1+\exp(y_i \mathbf{w}' \mathbf{x}_i)})$ would better fit our non-linearly separable data, hence we need a relatively big γ and a small $\frac{\gamma}{\sigma}$ ratio. That's what I observed in the experiments.

3. [15 points] **Naive Bayes**

Implement the simple Naive Bayes learner. You need to count the features to get the likelihoods one at a time. To get the prior, you will need to count the number of examples in each class.

For every feature x_i, you should estimate its likelihood of taking a value for a given label y (which is either + or −) as:

$$P(x_i|y) = \frac{Count(x_i, y) + \lambda}{Count(y) + S_i\lambda}.$$

Here, S_i is the number of all possible values that x_i can take in the data. (In the data provided, each feature is binary, which should simplify your implementation a lot.)

The hyper-parameter λ is a smoothing term. In example we saw in class, we set $\lambda = 1$. But, in this experiment, you should choose the best λ based on cross-validation.

Hyper-parameter: Smoothing term: $\lambda \in \{2, 1.5, 1.0, 0.5\}$

Implementation & Discussion:

(a) Language: Language/Package: Python + Pandas + Numpy

(b) Details: Since the dimension is manageable, I use numpy.arrays to represent features. The training set is stored as a pandas.DataFrame, shape = (n_smaples, 2), columns =['y', 'X']. I iterate over each row to update $\Pr(x_j|y)$. Example i is labeled +1 if $\Pr(+1|X) > \Pr(-1|X)$.

(c) Discussion: It's worth noting that shifting the smooth parameter from 0.5 to 2 makes no difference to the performance. I think that's because $\Pr(x_j|y)$ for some j is very large, therefore adding counts to the less frequent features didn't change the MLE in a significant way.

4. [25 points] **SVM over trees**

In class we have learned how the bagging and random forest algorithms work. In this setting, you are going to build a different ensemble over depth-limited decision trees that are learned using the ID3 algorithm.

First, using the training set, you need to build 200 decision trees. To construct a decision tree, you need to sample 10% of the examples *with replacement* from the training set (i.e. 2000 examples), and use this subset to train your decision tree with a depth limit d. Repeating this 200 times will get you 200 trees.

Usually, the final prediction will be voted on by these trees. However, we would like to train an SVM to combine these predictions. To do so, you should treat the 200 trees as a feature transformation and construct a new dataset by applying the transformation. That is, suppose your trees were $tree_1, tree_2, \cdots, tree_{200}$. Each of these are functions that can predict a label for an example that is either -1 or $+1$. Instead of just predicting the label, treat them as a feature transformation $\phi(x)$ that is defined as:

$$\phi(x) = [tree_1(x), tree_2(x), \cdots, tree_N(x)]$$

In other words, you will build an N dimensional vector consisting of the prediction (1 or -1) of each tree that you created. Thus, you have a *learned* feature transformation.

Now, you can train an SVM on these transformed features. (Don't forget to transform the test set before making your final evaluations.)

Hyper-parameters:

(a) Initial learning rate $\gamma_0 \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

(b) Tradeoff $C \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

(c) Depth: $d \in \{10, 20, 30\}$

Implementation & Discussion

(a) Language: Language/Package: Python + Pandas + Numpy

(b) Details: Since the dimension is manageable, I use numpy.arrays to represent features. The training set is stored as a pandas.DataFrame, shape = (n_smaples, 2), columns =['y', 'X']. To shorten the computation time, I use the builtin "multiprocessing" module, which gives me a 7x boost on a eight core machine. To avoid data loss, I write the trained rules incrementally to disk with the "pickle" module. It took 8 hours to transform the features in the training set and another 4 hours for the testing set.

(c) Discussion: The SVMTree model outperforms the original SVM, but still slightly worse than NB. I think the main reason is that by transforming the features using a non-linear (Decision Tree) way, we make the features which are originally non-linearly separable into a linearly separable space, pretty much like the "activation" function in Neural Networks. The SVM is insensitive to the transformed features, where most of the parameter combinations yield similar outcome. I think that's because the features are of the same scale ("+1" and "-1") to the label after transformation.

2.4 What to report

1. For each algorithm above, briefly describe the design decisions that you have made in your implementation. (E.g, what programming language, how do you represent the

vectors, trees, etc.)

I set the initial weights to a small random number vector $(0.02 \times [0, 1] - 0.01)$. For other design decisions please refer to the **Implementation & Discussion** part in the above section for each algorithm.

2. Report the best hyper-parameters, the average precision, recall and F_1 achieved by those hyperparameters during cross-validation and the precision/recall/F_1 on the test set. You can use the table 1 as a template for your reporting.

   For reasons of choice of hyper-parameters, please also refer to the **Implementation & Discussion** part in the above section for each algorithm.

	Best hyper-parameters	Average Cross-validation P/R/F1	Test P/R/F1
SVM	$\gamma = 0.1$, $C = 10$, epoch $= 15$	0.333/0.565/0.370	0.641/0.256/0.366
Logistic regression	$\gamma = 1$, $\sigma = 1000$, epoch $= 6$	0.454/0.398/0.399	0.284/0.529/0.370
Naive Bayes	$\lambda = 1$	0.498/0.539/0.518	0.512/0.547/0.529
SVM over trees	$depth = 10$, $treenumber = 200$, $\gamma = 0.01$, $c = 10$	0.699/0.404/0.512	0.718/0.399/0.513

Table 1: Results table

3 [25 points, Extra Credit for the holidays] Naïve Bayes and Linear Classifiers

In this problem you will show that a Gaussian naïve Bayes classifier is a linear classifier. We will denote inputs by d dimensional vectors, $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$. We will assume that each feature x_j is a real number. Our classifier will predict the label 1 if $\Pr(y = 1|\mathbf{x}) \geq \Pr(y = 0|\mathbf{x})$. Or equivalently,

$$\frac{\Pr(\mathbf{x}|y = 1)\Pr(y = 1)}{\Pr(\mathbf{x}|y = 0)\Pr(y = 0)} \geq 1$$

Remember the naïve Bayes assumption we saw in class:

$$\Pr(\mathbf{x}|y) = \prod_{j=0}^{d} \Pr(x_j|y)$$

Suppose each $P(x_j|y)$ is defined using a Gaussian/Normal probability density function, one for each value of y and j. Each Gaussian distribution has mean $\mu_{j,y}$ and variance σ^2 (Note that they will all have same variance). As a reminder, the Gaussian distribution is represented by the following probability density function:

$$f(x_j \,|\, \mu_{j,y}, \sigma) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x_j - \mu_{j,y})^2}{2\sigma^2}}$$

Show that this naïve Bayes classifier has a linear decision boundary.
[Hint: Refer to the notes on the naive Bayes classifier and Linear models in the class website to see how to do this with binary features]

For simplicity, denote $P(y = 1)$ by p.

$$\begin{aligned}
\frac{\Pr(\mathbf{x}|y=1)\Pr(y=1)}{\Pr(\mathbf{x}|y=0)\Pr(y=0)} &= \frac{\Pr(y=1)\prod_{j=0}^{d}\Pr(x_j|y=1)}{\Pr(y=0)\prod_{j=0}^{d}\Pr(x_j|y=0)} \\
&= \frac{p}{1-p}\frac{\prod_{j=0}^{d} f(x_j|\mu_{j,1},\sigma)}{\prod_{j=0}^{d} f(x_j|\mu_{j,0},\sigma)} \\
&= \frac{p}{1-p}\prod_{j=0}^{d}\frac{\exp\{-\frac{(x_j-\mu_{j,1})^2}{2\sigma^2}\}}{\exp\{-\frac{(x_j-\mu_{j,0})}{2\sigma^2}\}} \\
&= \frac{p}{1-p}\prod_{j=0}^{d}\exp\{-\frac{(x_j-\mu_{j,1})^2}{2\sigma^2}+\frac{(x_j-\mu_{j,0})^2}{\sigma^2}\} \geq 1
\end{aligned}$$

Taking log,

$$\begin{aligned}
\text{LHS} &= \log\frac{p}{1-p}+\frac{1}{2\sigma^2}\sum_{j=0}^{d}(-x_j^2+2x_j\mu_{j,1}+\mu_{j,1}^2+x_j^2+2x_j\mu_{j,0}+\mu_{j,0}^2) \\
&= \{\log\frac{p}{1-p}+\frac{1}{2\sigma^2}\sum_{j=0}^{d}(\mu_{j,1}^2+\mu_{j,0}^2)\}+\sum_{j=0}^{d}(\{\frac{\mu_{j,1}-\mu_{j,0}}{\sigma^2}\}x_j) \geq 0
\end{aligned}$$

Let us denote $\{\log\frac{p}{1-p}+\frac{1}{2\sigma^2}\sum_{j=0}^{d}(\mu_{j,1}^2+\mu_{j,0}^2)\}$ by b, and denote $\{\frac{\mu_{j,1}-\mu_{j,0}}{\sigma^2}\}$ by w_j, we get the linear form

$$b+\sum_{j=0}^{d} w_j x_j \geq 0$$

Experiment Submission Guidelines

1. The report should detail your experiments. For each step, explain in no more than a paragraph or so how your implementation works. Describe what you did. Comment on the design choices in your implementation. For your experiments, what algorithm parameters did you use? Try to analyze this and give your observations.

2. Your report should be in the form of a *pdf* file, LaTeX is recommended.

3. *Your code should run on the CADE machines.* You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.

You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.

4. Please do not hand in binary files! We will *not* grade binary submissions.

5. Please look up the late policy on the course website.