CS 5350/6350: Machine Learining Spring 2015

Homework 5

Handed out: Mar 31, 2015
Due date: Apr 13, 2015

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 (and program) your own solution. Please keep the class collaboration policy in mind.
- Feel free ask questions about the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. Your assignment should be **no more than 10 pages**. X points will be deducted if your submission is X pages beyond the page limit.
- Handwritten solutions will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework as a pdf on Canvas.
- Some questions are marked **For 6350 students**. Students who are registered for CS 6350 should do these questions. Of course, if you are registered for CS 5350, you are welcome to do the question too, but you will not get any credit for it.

1 Kernel Perceptron

In the class, we have seen the idea of kernels applied to support vector machines. In this question, you will derive the kernel version of the Perceptron learner. For the purpose of this question, we will assume that instances are classified using a feature representation $\phi(x)$. That is, classification using Perceptron is the sign of the dot product of the weight vector and the example $w^T\phi(x)$. The goal is to, instead, represent this using a kernel $K(w, x)$.

Note that, with this feature representation, the perceptron update is $w \leftarrow w + ry\phi(x)$, whenever there is a mistake (i.e. $yw^T\phi(x) < 0$). For this problem you may assume the learning rate $r = 1$.

1. Assume that the initial weight vector is the zero vector. Then, show that $w = \sum_{(x_i,y_i)\in M} y_i\,\phi(x_i)$. Here $M = \{(x_i, y_i)\}$ is the set of examples on which the learning algorithm made mistakes.

2. Let K be a kernel function defined as $K(u, v) = \phi(u)^T\phi(v)$.

   Using the fact that the weight vector can be written as in the previous question, write the classification step $y = sgn\left(w^T\phi(x)\right)$ using the kernel function $K(u, v)$ instead of using the explicit feature representation $\phi(u)$.

3. In pseudo code, write the full kernel perceptron algorithm.

   (Hint: Instead of storing a weight vector, you will store a list of examples the algorithm made a mistake on.)

2 Naïve Bayes

Consider the Boolean function $f_{TH(3,7)}$. This is a threshold function defined on the 7 dimensional Boolean cube as follows: given an instance x, $f_{TH(3,7)}(x) = 1$ if and only if 3 or more of x's components are 1.

1. Show that $f_{TH(3,7)}$ has a linear decision surface over the 7 dimensional Boolean cube.
2. Assume that you are given data sampled according to the uniform distribution over the Boolean cube $\{0,1\}^7$ and labeled according to $f_{TH(3,7)}$. Use naïve Bayes to learn a hypothesis that predicts these labels. What is the hypothesis generated by the naïve Bayes algorithm? (You do not have to implement the algorithm here. You may assume that you have seen all the data required to get accurate estimates of the probabilities).
3. Show that the hypothesis produced in (b) does not represent this function.
4. Are the naïve Bayes assumptions satisfied by $f_{TH(3,7)}$? Justify your answer.

3 Ensembles of decision trees

Support Vector Machines

Recall from class that an SVM is a technique for learning a linear classifier by minimizing the following loss function:

$$E(w) = \frac{1}{2} w^T w + C \sum_i \max(0, 1 - y_i w^T x_i)$$

where C is a hyper-parameter that controls the relative importance of the regularization term $\frac{1}{2} w^T w$ with respect to the error term. As always, the inputs x_i are real valued vectors and $y_i \in \{-1, +1\}$. This formalism is commonly referred to as L2 regularization and L1 loss.

Stochastic Gradient Descent

The concept behind SGD is to do gradient decent, but only calculate the gradient using a single example. In practice, it can be helpful to shuffle the order of the data for each epoch.

1. Initialize $w = \vec{0}$, $t = 0$
2. for epoch $1 \dots T$:
   (a) for example (x_i, y_i) for every i (random order)

i. r_t = Learning rate at t
ii. $w = w - r_t \nabla E(w, x_i, y_i)$
iii. $t = t + 1$

Here, the gradient is defined as follows:

$$\nabla E(w, x, y) = \begin{cases} w - Cyx & \text{if } yw^T x \leq 1 \\ w & \text{otherwise} \end{cases}$$

(Refer to the lecture slides for the full description of the algorithm.)

The learning rate is often stated as just r, but in practice it is better to scale it down as the algorithm converges. In your implementation r will be a function of the initial learning rate, ρ_0, and the example number, t. In the case of the SVM loss function the best function to choose is

$$r(t, \rho_0) = \frac{\rho_0}{1 + \rho_0 t / C}.$$

Here, t should be initialized to zero at the start and incremented for each example. Note that t should not be reset at the start of the epoch.

Cross Validation

In class we have seen cross validation is used to select hyper-parameters for learning algorithms. Some of the training data is put aside, and when training is finished, the resulting classifier is tested on the held out data. This allows you get get an idea of how well the particular choice of hyper-parameters does. Since you did not train on your whole dataset you may have introduced a bias. To correct for this, you will need to train many classifiers with different subsets of the data removed.

For problems with small data sets, a popular method is the leave-one-out approach. For each example, a classifier is trained on the rest of the data and the chosen example is then evaluated. The performance of the classifier is the average accuracy on all the examples. The downside to this method is for a data set with n examples you must train n different classifiers. Of course, this is not practical for the data set you will use in this problem, so you will hold out subsets of the data many times instead.

Specifically, for this problem, you should implement k-fold cross validation to identify the hyper-parameters C and the initial learning rate ρ_0. The general approach for k-fold cross validation is the following: Suppose you want to evaluate how good a particular hyper-parameter is. You split the training data D into k parts. Now, you will train your model on $k - 1$ parts with the chosen hyper-parameter and evaluate the trained model on the remaining part. You should repeat this k times, choosing a different part for evaluation each time. This will give you k values of accuracy. Their *average cross-validation accuracy* gives you an idea of how good this choice of the hyper-parameter is. To find the best value of the hyper-parameter, you will need to repeat this procedure for different choices of the hyper-parameter.

Data

The data is the badges data (again). This dataset is similar to the badges data that we have seen in class and in a previous homework. The decision function, however, is a rather complex one this time. You can download the dataset from the course website in a file called `badges.tar.gz`. This archive consists for four files: `badges-train.txt` and `badges-test.txt` are the new training and test files. In addition, we have also extracted indicator features from the first five characters of the first and last names. The feature extracted train and test data is in `badges-train-features.txt` and `badges-test-features.txt` respectively.

This feature extracted data is in the libSVM data format which we used in homework 2. Recall from the description from homework 2 that in this format, each line is a single training example. The format of the each line in the data is

```
<label> <index1>:<value1> <index2>:<value2> ...
```

Here `<label>` denotes the label for that example. The rest of the elements of the row is a sparse vector denoting the feature vector. For example, if the original feature vector is $[0, 0, 1, 2, 0, 3]$, this would be represented as `3:1 4:2 6:3`. That is, only the non-zero entries of the feature vector are stored.

Experiment

You will train on the new badges data available on the class web page.

1. Implement SVM using SGD as a training algorithm.

2. Using the provided features, run 10-fold cross validation to find the best values for the hyper-parameters ρ_0 and C. Try out all combinations of $\rho_0 \in \{0.001, 0.01, 0.1, 1\}$ and $C \in \{0.1, 1, 10, 100, 1000\}$. Feel free to expand the set of parameters if you have enough time.

   Show a table including the 5 best parameters with three columns: ρ_0, C and the average cross validation accuracy for that choice of hyper-parameters.

   (Since all you care about here is the relative accuracy to other training runs it is not necessary for the weight vector to converge. To make cross validation faster, only run 10 epochs of SGD.)

   Use the best value of ρ_0 and C to train a classifier on the entire training set. Run at least 30 epochs of SGD. Report the performance of this classifier on the test set.

3. Next, you will use the ID3 algorithm to train decision trees on sub-samples of the data. You can use your own implementation of decision trees from the earlier homework for this part. The one important change to the tree growing algorithm is a depth parameter. This is a number that restricts the tree from growing beyond a certain depth. If the depth of the tree reaches the limit, your algorithm should create a leaf with the majority label.

From the full training set, randomly sample half the examples. Grow a decision tree of maximum depth 4 on this sample. This decision tree can now predict a + or − for any example.

Train hundred different decision trees of maximum depth four on the entire training set. *Note: To get a hundred* **different** *decision stumps, you need to repeatedly sample 50% of the training set and train a decision tree on the sub-sample.*

At this point, you should have 100 trees, each of which can predict + or − for any example. These trees will create your new feature set. Report the mean and variance of the accuracy of these trees on the test set.

Create a new 100-dimensional dataset using the hundred decision trees as follows: For each example in the data set, the value of the i^{th} feature will be the prediction of the i^{th} decision tree. This will give you a new feature representation for the data. Using this new feature set, train a linear separator with the SGD algorithm and evaluate with 10-fold cross-validation as in part 2.

Remember: Make sure that you only sample from the training set to generate the decision stumps, otherwise you might contaminate the training set with examples from the test set and this will skew your results.

4. Repeat this with depth limits of 8 and 20.

3.1 Evaluation and report

You should compare seven models: SGD for SVM on the original feature set, three depths of decision trees, and SVM over an ensemble of decision trees with depth limits four, eight and twenty. Of course, this is just the minimum. Feel free to experiment with more parameter combinations (decision tree depth, learning rate for the SGD and fraction of the data used to train the trees.) In each setting, you should use cross-validation to get the best SVM hyper-parameters.

Report the following for each case: The best cross validation accuracy and the test set performance. For decision trees show the variance instead of cross validation accuracy. Rank the different methods in terms of their performance. In the end, your conclusion will be that a particular algorithm (or set of algorithms) performed the best. Briefly state the assumptions that this conclusion is based on.

As mentioned in previous homeworks, you may use any programming language for your implementation. Upload your code along with a script so the TAs can run your solution in the CADE environment.