CS 475 Machine Learning: Homework 3
Non-linear Methods
Due: Wednesday October 12, 2016, 11:59pm
100 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Programming (50 points)

1.1 Instance-based Learning

A simple non-parametric classifier is the k-Nearest Neighbor algorithm, which you read about in Chapter 1 of "Murphy". k-NN attempts to classify points based on the closest k training examples in the feature space. There are many variations of k-NN learning. You will implement two versions of k-Nearest Neighbor with one distance metric.

1. Standard k-NN

   Find the k nearest neighbors of the test example using the provided similarity metric. The prediction rule is

   $$\hat{y} = f(\mathbf{x}) = \arg\max_{\hat{y}'} \sum_{i \in d_k(\mathbf{x}, X)} [y_i = \hat{y}']$$

   where $d_k(\mathbf{x}, X)$ is the set of the k elements in X closest to $\mathbf{x}$. The notation $[y_i = \hat{y}']$ is the Kronecker delta, which is a function that returns 1 if the condition inside is true, 0 otherwise.

   This method should correspond to `--algorithm knn`.

2. Distance weighted k-NN

   Find the k nearest neighbors of the test example and weight the vote of each example by its distance to the test example, given by the similarity metric. The prediction rule is

   $$\hat{y} = f(\mathbf{x}) = \arg\max_{\hat{y}'} \sum_{i \in d_k(\mathbf{x}, X)} \frac{1}{1 + d(\mathbf{x}, \mathbf{x}_i)^2} [y_i = \hat{y}']$$

   This method should correspond to `--algorithm distance_knn`.

1.1.1 Distance Metric

For this assignment you will implement Euclidean distance as your distance metric. Specifically, for vectors x and x', Euclidean distance is defined as:

$$d_E(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x'_i)^2}$$

1.1.2 Number of Neighbors k

Add a command line option for specifying the number of neighbors for k-NN by adding the following code block to the `get_args` function in `classify.py`.

```
parser.add_argument("--knn", type=int, help="The value of K for KNN classification.",
                    default=5)
```

Your default value for `--knn` should be 5.

1.1.3 Beyond Binary Classification

In addition to binary classification, your K-NN classifier must support multi-class classification. For multi-class classification, you will count the nearest neighbors as before, taking the label that receives the most votes. We will provide you with a new multi-class data set.

1.1.4 Tied Prediction

In the case of a tie, you must return the label tied for the most votes that has the lowest label index. For example, if label 0 received 3 votes, label 1 received 4 votes and label 2 received 4 votes, you would return label 1 since it has the smallest label index of the labels tied for the most votes.

1.1.5 New Features

You may encounter new features at test time that aren't in the training data. For simplicity, only consider features that appear in the training data when evaluating neighbor similarity.

1.1.6 Deliverables

You need to implement both Standard k-NN and Distance weighted k-NN. Your predictors will be selected by passing the string `knn` and `distance_knn` as the argument for the algorithm parameter.

1.1.7 How Your Code Will Be Called

You may use the following commands to test your algorithm.

```
python classify.py --mode train --algorithm knn --model-file speech.knn.model \
                    --data speech.train --knn 5
```

To run the trained model on development data:

```
python classify.py --mode test --model-file speech.knn.model --data speech.dev
```

1.2 Boosting

You will implement the AdaBoost algorithm for binary classification. AdaBoost takes a weak learner and boosts it into a strong learner. The weak learner you will be using will be a decision stump: a linear classifier **in one dimension of the data**. This is essentially a decision tree that can only take a single feature (hence a stump). For the weak learner you can choose any cutoff in one dimension (more details below).

The hypothesis set H is the set of all one dimensional linear classifiers:

$$H = \{h_{j,c} : j \text{ is the feature index and } c \text{ is the cutoff}\}$$

where

$$h_{j,c}(\mathbf{x}_i) = \begin{cases} \arg\max_{\hat{y}} \sum_{\forall k:\mathbf{x}_{kj}>c} [y_k = \hat{y}] & \text{if } \mathbf{x}_{ij} > c \\ \arg\max_{\hat{y}} \sum_{\forall k:\mathbf{x}_{kj}\leq c} [y_k = \hat{y}] & \text{otherwise} \end{cases}$$

h_t will be used to describe the optimal $h_{j,c}$ at iteration t.

The AdaBoost algorithm is as follows:

1. Input: $(y_1, \mathbf{x}_1), ..., (y_n, \mathbf{x}_n)$ where $\mathbf{x}_i \in \mathbb{R}^m$ and $y_i \in \{-1, 1\}$
2. Initialize: $D_1(i) = \frac{1}{n}$
   where $D_t(i)$ is the weight of instance $(y_i, \mathbf{x}_i)$ at iteration t
3. For each iteration $t \in \{1, 2, ..., T\}$ do:
   (a) $h_t = \arg\min_{h' \in H} \epsilon_t(h')$
   where $\epsilon_t(h) \equiv \sum_{i=1}^{n} D_t(i)[h(\mathbf{x}_i) \neq y_i]$
   (b) $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t(h_t)}{\epsilon_t(h_t)}$
   (c) $D_{t+1}(i) = \frac{1}{Z_{t+1}} D_t(i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$
   where $Z_{t+1} = \sum_{i=1}^{n} D_t(i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$ is the normalizing factor

Note for the log used in calculating α_t, the natural log should be used, which is `math.log()` in Python (do `import math` before using the log function). This algorithm will be selected with the value `adaboost` for the argument `algorithm` on the command line.

1.2.1 Choosing $h_{j,c}$

Remember that when you are finding the best $h \in H$, you only need to check up to $n-1$ values of c per dimension. This is because more values will have no affect on your training set. You are only choosing values of c that partition the examples into non-empty sets.

To this end, you should choose values of c in line with something like the *max-margin principle*, where the observable error does not distinguish values of c. That is, assuming you are choosing c for $h_{j,c}$ (i.e. in dimension j) with examples sorted in ascending order of value in j: choose from values of c that will partition the data into $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$ and $\{\mathbf{x}_{k+1}, ..., \mathbf{x}_n\}$ in dimension j as $c \in \{\frac{1}{2}(\mathbf{x}_{k+1,j} + \mathbf{x}_{k,j}) : k \in \{1, 2, ..., n-1\}\}$. Note that the number of distinct values of c may be less than $n-1$ in cases where there are duplicate values in dimension j for some instances. You will choose one dimension j and one of these (up to) $n-1$ values for c that minimizes the error of $h_{j,c}$:

$$h_t = \arg\min_{h_{j,c}} \epsilon_t(h_{j,c}) = \arg\min_{h_{j,c}} \sum_{i=1}^{n} D_t(i)[h_{j,c}(\mathbf{x}_i) \neq y_i]$$

1.2.2 Making Predictions

In AdaBoost, you make your predictions as a weighted vote of the classifiers learned on each iteration:

$$\hat{y} = f(x) = \arg\max_{\hat{y}'} \sum_{t=1}^{T} \alpha_t [h_t(x) = \hat{y}]$$

Keep in mind that each h_t is the $h_{j,c}$ at iteration t.

1.2.3 Boosting Iterations

The number of boosting iterations to run will be given as a command line flag. Add this command line option by adding the following code block to the `get_args` function in `classify.py`.

```
parser.add_argument("--num-boosting-iterations", type=int, help="The number of boosting iterations to run.",
                    default=10)
```

The default number of iterations is 10.

1.2.4 Stopping Criteria

In some cases, your error ϵ_t will reach 0. In this case, α_t will become infinite. Therefore, we will place a stopping criteria on boosting. When you encounter an ϵ_t that is close to 0, where close to 0 is measured as less than 0.000001, stop boosting and do not use the current hypothesis (since we cannot set α_t for this hypothesis). You should stop even if you have not performed every boosting iteration requested by `--num-boosting-iterations`.

1.2.5 Deliverables

You need to implement AdaBoost. Your predictor will be selected by passing the string `adaboost` as the argument for the `algorithm` parameter.

1.2.6 How Your Code Will Be Called

You may use the following commands to test your algorithm.

```
python classify.py --mode train --algorithm adaboost --model-file speech.adaboost.model \
                   --data speech.train --num-boosting-iterations 10
```

To run the trained model on development data:

```
python classify.py --mode test --model-file speech.adaboost.model --data speech.dev
```

1.3 Data Sets

Because the complexity of both k-NN and AdaBoost depend heavily on the number of features in the data, we will not test your algorithms on the NLP data set. Remember to test your k-NN implementation on the multi-class data set as well as binary data sets.

2 Analytical (50 points)

1) Decision Trees (12 points) Consider a binary classification task with the following set of training examples:

x_1	x_2	x_3	x_4	x_5	classification
0	1	1	-1	-1	+
0	1	1	-1	-1	+
0	1	1	1	1	-
0	-1	1	1	1	+
0	-1	1	-1	-1	-
0	-1	1	-1	-1	-

(a) Can this function be learned using a decision tree? If so, provide such a tree (describe each node in the tree). If not, prove it.

(b) Can this function be learned using a logistic regression classifier? If yes, give some example parameter weights. If not, why not.

(c) What is the entropy of the labels in the training data?

(d) What is the information gain of x_2 relative to labels?

2) Decision Tree (12 points) Let's investigate the accuracy of decision tree learning. We start by constructing a unit square ($[0;1] \times [0;1] \times [0;1]$). We select n samples from the cube, each with a binary label ($+1$ or -1), such that no two samples share either x, y, or z coordinates. Each feature can be used multiple times in a decision tree. At each node we can only conduct a binary threshold split using one single feature.

(a) Prove that we can find a decision tree of depth at most $\log_2 n$, which perfectly labels all n samples.

(b) If the samples can share either two coordinates but not all three, can we still learn a decision tree which perfectly labels all n samples with the same depth as (a)? Why or why not?

3) Adaboost (14 points) There is one good example at $x = 0$ and two negative examples at $x = \pm 1$. There are three weak classifiers are

$$h_1(x) = 1 \cdot \mathbf{1}(x > 1/2) - 1 \cdot \mathbf{1}(x \leq 1/2),$$
$$h_2(x) = 1 \cdot \mathbf{1}(x > -1/2) - 1 \cdot \mathbf{1}(x \leq -1/2)$$
$$h_3(x) = 1.$$

Show that this data can be classified correctly by a strong classifier which uses only three weak classifiers. Calculate the first two iterations of AdaBoost for this problem. Are they sufficient to classify the data correctly?

4) Ensemble Methods (12 points) Consider the following binary classification Boosting algorithm.

1. Given $\{\mathbf{x}_i, y_i\}_{i=1}^N$, number of iterations T, weak learner f.
2. Initialize $\mathcal{D}_0$ to be a uniform distribution over examples.
3. For each iteration $t = 1 \dots T$:
   (a) Train a weak learner f on the data given $\mathcal{D}_t$ to produce hypothesis h_t.
   (b) Compute the error of h_t as $\epsilon_t = P_{\mathcal{D}_t}[h_t(\mathbf{x}_i) \neq y_i]$
   (c) Compute $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
   (d) Update $\mathcal{D}$ as:
   $$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t + (T-t)/T) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t + (T-t)/T) & \text{otherwise} \end{cases}$$
4. Output final hypothesis $H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^T \alpha_t h_t(\mathbf{x})\right\}$

Z_t is a normalization constant so that $\mathcal{D}$ is a valid probability distribution.

Describe the difference between this algorithm and the AdaBoost algorithm we learned about in class. What problem of AdaBoost is this change designed to fix? How does changing the algorithm's user provided parameter affect this behavior?

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `code.zip`. **You must submit source code (.py files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you. We will include the libraries specific in `requirements.txt` but nothing else.
2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (code.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: https://piazza.com/class/it1vketjjo71l1.