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?
   Using chain rule for differentiation with respect to weight vector w:

$$g'(\mathbf{w}) = \frac{d}{dw} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

$$g'(\mathbf{w}) = \frac{1}{(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))} * \frac{d}{dw}(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

$$g'(\mathbf{w}) = \frac{1}{(1+\exp(-y_i\mathbf{w}^T\mathbf{x}_i))} * [(-y_i\mathbf{x}_i)\exp(-y_i\mathbf{w}^T\mathbf{x}_i)]$$

On further simplification we get:

$$g'(\mathbf{w}) = \frac{-y_i\mathbf{x}_i}{(1+\exp(y_i\mathbf{x}_i))}$$

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.
   We need to optimize the minimizing function in the question.
   As we have seen before on perceptron or any other convex functions, gradient descent update is done by the expression:

$$w^{t+1} = w^t - r\nabla J(w^t)$$

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

$$\nabla J(w^t) = \frac{dJ(w)}{dw}$$

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

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.

   - Initialize weight vector w = 0
   - For each epoch in range 1 ... N:
     - Shuffle the training set
     - Pick a ramdom example and assume this represents the entire training set.
     - We update the weight vector if $y_i w^T x_i <= 1$. If the current iteration is t and learing rate is r, the update expression is given by:

$$w_t = w_{t-1} - r\nabla J(w_{t-1})$$

$$w_t = w_{t-1} + r\frac{y_i x_i}{1+\exp(y_i\mathbf{x}_i)} - \frac{2r}{\sigma^2}\mathbf{w}$$

   - return the final weight vector $\mathbf{w}$

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 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: $C \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

   Ans: Implementation involved modifying the weights update method on the simple perceptron code. Rest all remaining the same. Cross validation was done on learning rate and balancer C, in a decreasing order. Prediction cross product * label ≥ 0 is labeled positive else negative. And if for any set of hyper parameter average is less than 1/3rd of max average then I skip the hyper parameter values.

   Cross Validation results:
   Learning rate 0.01, balancer 10
   Averages:
   F1 = 0.442605
   Precision = 0.685459
   Recall = 0.326856
   Accuracy = 81.86%
   Test Run Results using learning rate 0.01 and load balancer 10:
   F1 = 0.449118
   Precision = 0.722388
   Recall = 0.325853
   Accuracy = 82.19%

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 based on Perceptron. Weight vector updates are based on 1b - stochastic gradient descent. Updates are made when dot product of weight and example * label is less than or equal to 0. Initial weight vector is assignment to 0, but there was no difference when assigned with random weights between -1 to 1. Best Parameters:

Learning rate 0.001 sigma squared 100

Cross Validation:

Avg. F1 = 0.403011

Avg. Precision = 0.391119

Avg. Recall = 0.428492

Avg. Accuracy = 71.93%

Test:

F1 = 0.381619

Precision = 0.475553

Recall = 0.318671

Accuracy on Test = 76.99%

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: $\lambda \in \{2, 1.5, 1.0, 0.5\}$

Ans: Implemented the algorithm completely from the scratch. Training is basically finding probability for every value which every feature could take for every labels in the data. I used logarithm of the probabilities to help with floating point precision issues on multiplication. With log probabilities we can just add the values and predict. Prediction is done for every label and highest probability label is used as the prediction. To calculate the probability we add log of raw prob of the label and then add rest of the log probability based on the value of the features present. Train phase saves all the log probabilities in a dictionary

Cross validation results:

Best smoothing term = 2

average F1 = 0.508315

average Precision = 0.497278
average Recall = 0.520284
average accuracy = 77.82%
Test run results using smoothing term = 2
F1 = 0.526526
Precision = 0.520861
Recall = 0.532316
Accuracy = 78.67%

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: $d \in \{10, 20, 30\}$

Ans: For fixing the cross validation I assumed few things.
Due to time constraints I did cross validation with 20 trees. I built 20 decision trees sampling 10% of the training set. Used these trees to transform features on both the cross validation training and testing set.

After generating all the cross validation training and testing sets, used it on an svm to cross validate the hyper parameters of learning rate and load balancer.

Initial run with 20 trees for cross validation gave best parameters as:
limiting depth = 10
learning rate = 1
balancer = 0.1
With averages as follows:
F1 = 0.394505
Precision = 0.677340
Recall = 0.278241
Accuracy = 81.18%
Final Run was done using learning rate 1 and load balancer as 0.1 with 200 trees on the test set.
Results are as follows:
limiting depth = 10
learning rate = 1
balancer = 0.1
With averages as follows:
F1 = 0.4
Precision = 0.719953
Recall = 0.27693
Accuracy = 81.49%

Observation: For this data set the F1 averages are very close to one another except for few hyper parameter values like learning rate 1 balancer 10. So every time it is executed we get a different value for learning rate and balancer, but the values for different stats remains almost the same.

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:

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

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]

Ans: Combining the 2 equations in the question we have the values:

$$\frac{P(y=1)}{P(y=0)} \prod_{j=0}^{d} \frac{P(x_j|y=1)}{P(x_j|y=0)} \geq 1$$

From the question we know each value of y and j has mean $\mu_{j,y}$ and variance σ^2
Hence, lets assume for $y = 1$ and for a particular j we have mean as μ_{j,y_1}. Lets represent the prior probability of $P(y = 1)$ as p. Therefore, the prior probability of $P(y = 0)$ will be $(1 - p)$.
Substituting these values making use of the equations above we get the decision boundary:

$$\frac{p}{1-p} \prod_{j=0}^{d} \frac{\frac{1}{\sqrt[2]{2\pi\sigma^2}} e^{-\frac{(x_j-\mu_{j,y_1})^2}{2\sigma^2}}}{\frac{1}{\sqrt[2]{2\pi\sigma^2}} e^{-\frac{(x_j-\mu_{j,y_0})^2}{2\sigma^2}}} \geq 1$$

$$\frac{p}{1-p} \prod_{j=0}^{d} e^{\frac{1}{2\sigma^2}(\mu_{j,y_1}-\mu_{j,y_0})(2x_j-\mu_{j,y_0}-\mu_{j,y_1})} \geq 1$$

Taking log on both sides and rearranging the terms:

$$\log(\frac{p}{1-p}) + \frac{1}{2\sigma^2} \sum_{j=0}^{d} (\mu_{j,y_1} - \mu_{j,y_0})(-\mu_{j,y_1} - \mu_{j,y_0}) + \frac{1}{2\sigma^2} \sum_{j=0}^{d} 2x_j(\mu_{j,y_1} - \mu_{j,y_0}) \geq 0$$

Mean values for each individual distribution is particular for that distribution and is always a consistent/constant value with respect to x_j. Also it is stated that the variance is exactly same for each and every distribution. Therefore from the previous stated expression we can isolate the consistent term as bias:

$$bias = b = \log(\frac{p}{1-p}) - \frac{1}{2\sigma^2} \sum_{j=0}^{d} (\mu_{j,y_1}^2 - \mu_{j,y_0}^2)$$

and also we have

$$weights, w_j = \frac{1}{\sigma^2}(\mu_{j,y_1} - \mu_{j,y_0})$$

Therefore, the decision boundary value can be written as:

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

Thus, we can conclude that our classifier is linear in nature.