CS 475 Machine Learning: Homework 3
Supervised Classifiers 2

Due: Wednesday March 5, 2014, 12:00pm

100 Points Total Version 1.0

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

1 Programming (75 points)

In this assignment you will write three learning algorithms: a Perceptron classifier with Margin, a Dual Perceptron classifier with Margin, and Margin Infused Relaxation Algorithm. All the classifiers need only support binary prediction: the first two enforce large margins for linear and non-linear classification respectively, and the last one computes a different learning rate for each update ensuring that the example is correct after an update.

1.1 Perceptron with Margin

Perceptron is a mistake-driven online learning algorithm. It takes as input a vector of real-valued inputs $\mathbf{x}$ and makes a prediction $\hat{y} \in \{-1, +1\}$ (for this assignment we consider only binary labels). Predictions are made using a linear classifier: $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$. The term $\mathbf{w} \cdot \mathbf{x}$ is the dot product of $\mathbf{w}$ and $\mathbf{x}$ computed as $\sum_i x_i w_i$. Updates to $\mathbf{w}$ are made only when a prediction is incorrect: $\hat{y} \neq y$. The new weight vector $\mathbf{w}'$ is a function of the current weight vector $\mathbf{w}$ and example $\mathbf{x}$, y. The weight vector is updated so as to improve the prediction on the current example. Note that Perceptron naturally handles continuous and binary features, so no special processing is needed.

The basic structure of the algorithm is:

```
1. Initialize $\mathbf{w}$ to $\mathbf{0}$, set learning rate $\eta$ and number of iterations $I$
2. For each training iteration $k = 1 \ldots I$:
   (a) For each example $i = 1 \ldots N$:
       i. Receive an example $\mathbf{x}_i$
       ii. Predict the label $\hat{y}_i = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i)$, where $\text{sign}(\sum_i x_i \cdot w_i) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$
       iii. If $\hat{y}_i \neq y_i$, make an update to $\mathbf{w}$: $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$
```

Note that there is no bias term in this version and you should *not* include one in your solution. Also observe the definition of "sign" to account for 0 values. While sign returns -1 and 1, you must output as predictions the actual label values, which are 0 and 1.

This is the general form of the Perceptron, but in this assignment we will consider two changes: adding a margin and using the dual formulation.

1.1.1 Margin

By enforcing a margin during learning, we ensure that a training example is not only labeled correctly but it is labeled correctly with a margin. While the standard Perceptron updates on mistakes only (i.e. $\hat{y}_i \neq y_i$), the Perceptron with margin will update whenever the example is not classified correctly with at least a margin (i.e. $y_i(\mathbf{w} \cdot \mathbf{x}_i) < 1$). The actual update remains identical. This small change ensures that the algorithm keeps learning even when all examples are labeled correctly.

The algorithm for Perceptron with Margin is:

```
1. Initialize $\mathbf{w}$ to $\mathbf{0}$, set learning rate $\eta$ and number of iterations $I$
2. For each training iteration $k = 1 \dots I$:
    (a) For each example $i = 1 \dots N$:
        i. Receive an example $\mathbf{x}_i$
        ii. If $y_i(\mathbf{w} \cdot \mathbf{x}_i) < 1$, make an update to $\mathbf{w}$: $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$
```

1.1.2 Deliverables

You need to implement a single Perceptron algorithm with Margin. Your Perceptron predictor will be selected by passing the string `margin_perceptron` as the argument for the algorithm parameter.

1.1.3 Learning Rate

Perceptron uses a learning rate η, where $0 < \eta \leq 1$. Your default value for η should be 1. You *must* add a command line argument to allow this value to be adjusted via the command line.

Add this command line option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("online_learning_rate", "double", true, "The learning rate for perceptron.");
```

Be sure to add the option name exactly as it appears above. A common mistake is to change underscores to dashes.

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
double online_learning_rate = 1.0;
if (CommandLineUtilities.hasArg("online_learning_rate"))
    online_learning_rate = CommandLineUtilities.getOptionValueAsFloat("online_learning_rate");
```

1.2 Dual Perceptron with Margin

The above algorithm only learns linear classifiers. However, some data can be highly non-linear, which is impossible to learn using a linear classifier. In class, we learned about the kernel trick, which uses the dual of a linear classifier for kernel learning. Kernels project data into high dimensional spaces in which a linear separator may exist. The decision boundary learned by the linear classifier is linear in the high dimensional space but non-linear in the original feature space.

A kernel is a function which maps the input data into a new space using a basis function and computes the inner product between two basis functions. A kernel function is defined as:

$$K(x, x') = (\phi(x) \cdot \phi(x')^T)$$

In this assignment you will implement the Dual Perceptron so that you can learn with a kernel. We derived the updates for the dual perceptron in class. Specifically, we can rewrite the prediction rule for the Dual Perceptron as:

$$\hat{y} = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x)$$

Instead of learning $\mathbf{w}$, you will learn α_i for each example $\mathbf{x}_i$. The new update increments α_i for each time the prediction for $\mathbf{x}_i$ is incorrect (for many training iterations):

$$\alpha_i += 1 \ \text{ iff } \ \hat{y}_i \ \text{ is incorrect } .$$

In this case, since we are implementing Perceptron with a margin, updates will happen not only when the prediction is incorrect, but when it is correct without a margin.

1.2.1 Deliverables

You need to implement a Dual Perceptron algorithm with Margin. Your algorithm should both enforce a margin and support kernels. Your Dual Perceptron predictor will be selected by passing a different string for each kernel for the algorithm parameter as described below.

1.2.2 Kernel Functions

You will implement two kernel functions for use with the Dual Perceptron. We strongly suggest building a single class for the `DualPerceptron` that supports different `Kernel` objects, rather than duplicate code. In fact, you can write a single class that supports the perceptron with margin above and the dual perceptron.

1. Linear Kernel: $K(x, x') = (x \cdot x'^T)$

   The command line argument for the `algorithm` parameter should be `perceptron_linear_kernel`.

2. Polynomial Kernel: $K(x, x') = (1 + (xx'^T))^d$

   The command line argument for the `algorithm` parameter should be `perceptron_polynomial_kernel`. The parameter d is explained below.

Note that the naive implementation may compute $K(x_i, x_j)$ many times during training. To improve training efficiency you should store (cache) computed values of $K(x_i, x_j)$ in a Matrix, $G_{ij} = K(x_i, x_j)$. G is called a Gram Matrix.

The parameters d for the polynomial kernel is set using command line options. Add this command line option by adding the following code block to the `createCommandLineOptions` method of `Classify`.

```
registerOption("polynomial_kernel_exponent", "double", true, "The exponent of the polynomial kernel.");
```

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
double polynomial_kernel_exponent = 2;
if (CommandLineUtilities.hasArg("polynomial_kernel_exponent"))
   polynomial_kernel_exponent = CommandLineUtilities.getOptionValueAsFloat("polynomial_kernel_exponent");
```

The default value for d should be 2.

1.3 Margin Infused Relaxation Algorithm

While the Perceptron described above benefits from a margin, it still makes no guarantees about the correctness of an example after an update.

This idea is captured in a class of algorithms called Passive-Aggressive Learning. These algorithms are passive in that they ignore examples that are classified correctly (with a margin). However, when an example is classified without a margin, they become aggressive and update to ensure that the examples are now classified with at least a margin.

One particular Passive-Aggressive algorithm is called the Margin Infused Relaxation Algorithm (MIRA). MIRA minimizes the hinge-loss by updating each example to enforce a margin. Specifically, MIRA solves the following optimization algorithm on every round:

$$\begin{aligned} \mathbf{w}' &= \arg\min_{\mathbf{w}'} \frac{1}{2}||\mathbf{w} - \mathbf{w}'||^2 \\ \text{s.t.} &\ \ y_i(\mathbf{w}'\mathbf{x}_i) \geq 1 \end{aligned}$$

Solving this optimization problem yields an additive update for $\mathbf{w}$

$$\mathbf{w}' = \mathbf{w}_i + \tau y_i \mathbf{x}_i \ ,$$

where $\mathbf{w}'$ is the new weight vector, $\mathbf{w}$ is the current weight vector, and τ is defined as:

$$\tau = \frac{1 - y_i(\mathbf{w}\mathbf{x}_i)}{||\mathbf{x}_i||^2}$$

The end result is an algorithm that is very similar to the Perceptron. Binary predictions are made using the standard prediction rule: $\text{sign}(\mathbf{w}\mathbf{x})$. The updates are both additive and modify $\mathbf{w}$ by adding $y\mathbf{x}$ times some learning rate. In the case of Perceptron, the learning rate was given as η. In this case, τ can be thought of as a dynamic learning rate, in which the scale of the update depends on how incorrect the prediction was. Finally, updates are made when the current example has non-zero loss. For (regular) Perceptron, this is based on the 0/1 loss. For MIRA, it is based on the hinge-loss, i.e. update when $y_i(\mathbf{w}\mathbf{x}_i) < 1$.

1.3.1 Deliverables

In this assignment, you will implement MIRA for binary linear classification. Your MIRA predictor will be selected by passing the string `mira` as the argument for the algorithm parameter.

1.4 Number of training iterations

For all the three algorithms above, since we will be running these online methods in batch mode, you can iterate multiple times over the data. This can improve performance by increasing the number of updates each algorithm makes. We will define the number of times each algorithm iterates over all of the data by the parameter `online_training_iterations`. You *must* define a new command line option for this parameter. Use a default value of 5 for this parameter.

You can add this option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("online_training_iterations", "int", true, "The number of training iterations for online methods.");
```

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
int online_training_iterations = 5;
if (CommandLineUtilities.hasArg("online_training_iterations"))
    online_training_iterations = CommandLineUtilities.getOptionValueAsInt("online_training_iterations");
```

Both MIRA and Perceptron with Margin must support this command line option. As in previous assignments, you should not change the order of examples. You must iterate over examples exactly as they appear in the data file, i.e. as provided by the data loader.

1.5 Numerical Precision

For all numerical calculations involving floating point numbers, use the `double` type and NOT the `float` type to store values. This will help in achieving numerical precision.

1.6 Data Sets

We are providing a new synthetic data set for this assignment called `Circle`. It may be helpful for testing your non-linear methods.

2 Analytical (25 points)

1) Kernels (10 points) We say K is a kernel function if there exists some transformation $\phi : \mathbb{R}^m \to \mathbb{R}^{m'}$ such that $K(x, x') = \langle \phi(x), \phi(x') \rangle$. Let K_1 and K_2 be two kernel functions.

(a) Prove that $K(x, x') = K_1(x, x')K_2(x, x')$ is a kernel function.

(b) Prove that $K(x, x') = K_1(x, x') + K_2(x, x')$ is a kernel function.

2) Logistic Loss (10 points) Linear SVMs can be formulated in an unconstrained optimization problem

$$\min_{w,b} \sum_{i=1}^{n} H(y_i(w^T x_i)) + \lambda \|w\|_2^2, \tag{1}$$

where λ is the regularization parameter and H is the well known logistic loss function:

$$H(a) = \log(1 + exp(-a))$$

The logistic loss function can be viewed as a convex surrogate of the 0/1 loss function, which can be written using the identity function as $I(a \leq 0)$.

(a) Prove that $H(a)$ is a convex function of a.

(b) The function $H(a) = \exp(-a)$ can also approximate the 0/1 loss function. How does this compare with the logistic loss function?

3) Margin (5 points) The SVM objective uses a margin value of 1 in the constraints ($\gamma = 1$.) Show that we can replace 1 with any arbitrary constant $\gamma > 0$ and that the solution for the maximum margin hyperplane is unchanged.

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `library.zip`. **You must submit source code (.java 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.

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 (library.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 the class discussion board: http://bb.cs475.org.