CS 475 Machine Learning: Homework 2
Supervised Classifiers 2

Due: Tuesday October 4, 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 Perceptron with Margin

In Homework 1 you implemented the Perceptron algorithm. In this homework you will implement a Perceptron with margin. You may reuse your Perceptron code from the previous assignment.

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 only on mistakes (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 update itself remains unchanged. 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 $k = 1 \ldots I$:
    (a) For each example $i = 1 \ldots N$:
        i. Receive an example $\mathbf{x}_i$
        ii. If $y_i(\mathbf{w} \cdot \mathbf{x}_i) < 1$, make an update to $\mathbf{w}$: $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$
```

1.1.1 Deliverables

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

1.1.2 Learning Rate and number of training iterations

The meaning of learning rate and number of training iterations here are the same as in Homework 1. We still use **–online-learning-rate** and **–online-training-iterations** to specify the learning rate and number of training iterations here as in Homework 1, with the same default values.

1.1.3 How Your Code Will Be Called

To train a model we will call:

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

There are some additional parameters which your program must support during training:

```
--online-learning-rate eta // sets the online learning rate, default = 1.0
--online-training-iterations t  // sets the number of SGD iterations, default = 5
```

All of these parameters are *optional*. If they are not present, they should be set to their default values.

To make predictions using a model we will call:

```
python classify.py --mode test --algorithm margin_perceptron \
        --model-file speech.margin_perceptron.model \
        --data speech.test \
        --predictions-file speech.test.predictions
```

1.1.4 Note about grading

You will be building on your code from the previous assignment before it has been graded. This means that you may have errors in your code from the previous homework that propagate to this assignment. As a general rule: we deduct points for errors on only a single assignment. If you had an error in a previous homework for which you lose points, you will not lose points in a later homework for the same error. You will have the opportunity to correct the error to regain the points. You will learn more about this when we discuss homework regrades.

1.2 Pegasos

In this assignment you will implement a simple but effective stochastic gradient descent algorithm to solve a Support Vector Machine for binary classification problems. The approach is called *Primal Estimated sub-GrAdient SOlver for SVM* (Pegasos).[1] The number of iterations required by Pegasos to obtain a solution of accuracy ϵ is $O(1/\epsilon)$, while previous stochastic gradient descent methods require $O(1/\epsilon^2)$.

1.2.1 SVM

A Support Vector Machine constructs a hyperplane in high dimensional space, which separates training points of different classes while keeping a large margin with regards to the training points closest to the hyperplane. SVMs are typically formulated as a constrained quadratic programming problem. We can also reformulate it as an equivalent unconstrained problem of empirical loss plus a regularization term. Formally, given a

[1]Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. Mathematical programming, 127(1), 3-30.

training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^M$ and $y_i \in \{+1, -1\}$, we want to find the minimizer of the problem:

$$\min_{\mathbf{w}} \lambda \frac{1}{2} ||\mathbf{w}||^2 + \frac{1}{N} \sum_{i=1}^{N} l(\mathbf{w}; (\mathbf{x}_i, y_i)) \tag{1}$$

where λ is the regularization parameter,

$$l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle\} \tag{2}$$

and $\langle \cdot, \cdot \rangle$ is the inner product operator. As you might have noticed, we omit the parameter for the bias term throughout this homework (i.e., the hyperplane is always across the origin).

1.2.2 Pegasos

Pegasos performs stochastic gradient descent on the primal objective Eq. 1. The learning rate decreases with each iteration to guarantee convergence. We adopt a simple strategy to decreasing the learning rate: we make the learning rate a function of the time steps.

On each time step Pegasos operates as follows. Initially, we set $\mathbf{w}_0 = 0$. On time step t of the algorithm (starting from $t = 1$) we first choose a random training example $(\mathbf{x}_{i_t}, y_{i_t})$ by picking an index $i_t \in \{1, \ldots, m\}$ uniformly at random (see Section 1.2.6). We then replace the objective in Eq. 1 with an approximation based on the training example $(\mathbf{x}_{i_t}, y_{i_t})$, yielding:

$$f(\mathbf{w}; i_t) = \lambda \frac{1}{2} ||\mathbf{w}||^2 + l(\mathbf{w}; (\mathbf{x}_{i_t}, y_{i_t})) \tag{3}$$

We consider the sub-gradient of the above approximate objective, given by:

$$\frac{\partial f(\mathbf{w}; i_t)}{\partial \mathbf{w}_t} = \lambda \mathbf{w}_t - \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1] y_{i_t} \mathbf{x}_{i_t} \tag{4}$$

where $\mathbb{1}[\cdot]$ is the indicator function which takes a value of 1 if its argument is true, and 0 otherwise. We then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \frac{\partial f(\mathbf{w}; i_t)}{\partial \mathbf{w}_t}$ using a step size of $\eta_t = 1/(\lambda t)$. Note that this update can be written as:

$$\mathbf{w}_{t+1} \leftarrow (1 - \frac{1}{t}) \mathbf{w}_t + \frac{1}{\lambda t} \mathbb{1}[y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1] y_{i_t} \mathbf{x}_{i_t} \tag{5}$$

After a predetermined number of iterations T, we output the last iterate $\mathbf{w}_T$.

You will note that the above update rule changes $\mathbf{w}$ even for cases where the value in $\mathbf{x}_i$ is 0. This is a non-sparse update: every feature is updated every time. While there are sparse versions of Pegasos, for this homework you should implement the non-sparse solution. This means that every time you update, every parameter must be updated. This will make training slower on larger feature sets (such as NLP) but it should still be reasonable. We suggest you focus testing on the smaller and thus faster datasets.

1.2.3 Offset Feature

None of the math above mentions an offset feature (bias feature) $\mathbf{w}_0$, that corresponds to a $x_{i,0}$ that is always 1. It turns out that we don't need this if our data is centered. By centered we mean that $E[y] = 0$. For simplicity, assume that the data used in this assignment is centered (even though this may not be true). Do not include another feature that is always 1 (x_0) or weight ($\mathbf{w}_0$) for it.

1.2.4 Convergence

In practice, SGD optimization requires the program to determine when it has converged. Ideally, a maximized function has a gradient value of 0, but due to issues related to your step size, random noise, and machine precision, your gradient will likely never be exactly zero. Common practice is to check that the L_p norm of the gradient is less than some δ, for some p. For the sake of simplicity and consistent results, we will not do this in this assignment. Instead, your program should take a parameter **–online-training-iterations** which is *exactly* how many iterations you should run (not an upper bound). An iteration is a single pass over every training example. **Note that the "t" in Section 1.2.2 indexes a time step, which is different from the "iterations" defined here.** The default of **–online-training-iterations** should be 5. Since you added this argument in the previous assignment, no further changes should be needed.

1.2.5 Regularization Parameter

Pegasos uses a regularization parameter λ to adjust the relative strength of regularization. Your default value for λ should be 10^{-4}. 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 `get_args` function in `classify.py`.

```
parser.add_argument("--pegasos-lambda", type=float, help="The regularization parameter for Pegasos.",
                    default=1e-4)
```

Be sure to add the option name exactly as it appears above. You can then use the value read from the command line in your main method by referencing it as `args.pegasos_lambda`. Note that the dashes have been replaced by underscores.

1.2.6 Sampling Instances

Pegasos relies on sampling examples for each time step. For simplicity, we will NOT randomly sample instances. Instead, on round i you should update using the ith example. An iteration involves a single pass in order through all the provided instances. You will make several passes through the data based on `--online-training-iterations`.

1.3 Implementation Notes

1. Many descriptions of SGD call for shuffling your data before learning. This is a good idea to break any dependence in the order of your data. In order to achieve consistent results for everyone, we are requiring that you **do not shuffle your data**. When you are training, you will go through your data in the order it appeared in the data file.

2. In SVM, $y \in \{+1, -1\}$, but the class labels in our data files are 0/1 valued. To resolve this inconsistency, convert class label 0 to -1 before training.

3. During prediction, a new instance $\mathbf{x}$ is predicted by the following rule:

   $\hat{y}_{new} = 1$ if $\langle \mathbf{w}, \mathbf{x} \rangle \geq 0$
   $\hat{y}_{new} = 0$ otherwise

4. Initialize the parameters $\mathbf{w}$ to 0.

1.3.1 Deliverables

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

1.3.2 How Your Code Will Be Called

To train a model we will call:

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

There are some additional parameters which your program must support during training:

```
--pegasos-lambda lambda   // sets the the constant scalar, default = 1e-4
--online-training-iterations t  // sets the number of SGD iterations, default = 5
```

All of these parameters are *optional.* If they are not present, they should be set to their default values.

To make predictions using a model we will call:

```
python classify.py --mode test --algorithm pegasos \
        --model-file speech.pegasos.model \
        --data speech.test \
        --predictions-file speech.test.predictions
```

Remember that your output should be 0/1 valued, not real valued.

2 Analytical (50 points)

The following problems consider a standard binary classification setting: we are given n observations with m features, $x_1, ..., x_n \in \mathbb{R}^m$.

1) Overfitting (10 points) Consider a kernel logistic regression classifier. The classifier has regularization parameter λ, as well as a kernel parameter γ which is used by a non-linear kernel (the exact kernel is unimportant. These parameters can be determined by tuning on held-out development data, or by using cross validation.

(a) Suppose we use cross validation to determine λ and γ and find that the values (λ_1, γ_1) and (λ_2, γ_2) achieve the same cross validation error. What other factors should you consider in determining which are the appropriate parameters to select for the final trained model?

(b) Just as with SVMs, we can fit a linear logistic regression classifier using either the primal or dual form. As we know, the primal form has m parameters to learn (number of features), while the dual form has n parameters to learn (number of examples). When $m \gg n$, will the dual form reduce over-fitting since it has fewer parameters? Explain your answer.

2) Hinge Loss (12 points) Linear SVMs using a square hinge loss can be formulated as an unconstrained optimization problem:

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} H(y_i(w^T x_i)) + \lambda \|w\|_2^2, \tag{6}$$

where λ is the regularization parameter and $H(a) = \max(1-a, 0)^2$ is the square hinge loss function. The hinge loss function can be viewed as a convex surrogate of the 0/1 loss function $I(a \leq 0)$.

(a) Compared with the standard hinge loss function, what do you think are the the advantages and disadvantages of the square hinge loss function?

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

(c) The function $L(a) = \max(-a, 0)^2$ can also approximate the 0/1 loss function. What is the disadvantage of using this function instead?

(d) We can choose a different loss function $H'(a) = \max(0.5 - a, 0)^2$. How will switching to H' from H effect the solution of the objective function? Specifically, the new objective becomes:

$$\hat{w}' = \arg\min \sum_{i=1}^{n} H'(y_i(w^T x_i)) + \lambda' \|w\|_2^2. \tag{7}$$

Explain your answer in terms of the relationship between λ and λ'.

3) Kernel Trick (10 points) The kernel trick extends SVMs to learn nonlinear functions. However, an improper use of a kernel function can cause serious over-fitting. Consider the following kernels.

(a) Inverse Polynomial kernel: given $\|x\|_2 \leq 1$ and $\|x'\|_2 \leq 1$, we define $K(x, x') = 1/(d - x^\top x')$, where $d \geq 2$. Does increasing d make over-fitting more or less likely?

(b) Chi squared kernel: Let x_j denote the j-th entry of x. Given $x_j > 0$ and $x'_j > 0$ for all j, we define $K(x, x') = \exp\left(-\sigma \sum_j \frac{(x_j - x'_j)^2}{x_j + x'_j}\right)$, where $\sigma > 0$. Does increasing σ make over-fitting more or less likely?

We say K is a kernel function, if there exists some transformation $\phi : \mathbb{R}^m \to \mathbb{R}^{m'}$ such that $K(x_i, x_{i'}) = \langle \phi(x_i), \phi(x_{i'}) \rangle$.

(c) Let K_1 and K_2 be two kernel functions. Prove that $K(x_i, x_{i'}) = K_1(x_i, x_{i'}) + K_2(x_i, x_{i'})$ is also a kernel function.

4) Predictions with Kernel (6 points) Let us compare primal linear SVMs and dual linear kernel SVMs in terms of computational complexity at prediction time.

(a) What is the computational complexity of prediction of a primal linear SVM in terms of the numbers of the training samples n and features m? If each sample contains at most q nonzero features, what can we do to accelerate the prediction? What is the resulting computational complexity?

(b) What is the computational complexity of prediction of a dual kernel SVM in terms of the numbers of the training samples n, features m, and support vectors s? If each sample contains at most q nonzero features, what can we do to accelerate the prediction? What is the resulting computational complexity?

5) Dual Perceptron (6 points)

(c) You train a Perceptron classifier in the primal form on an infinite stream of data. This stream of data is not-linearly separable. Will the Perceptron have a bounded number of prediction errors?

(c) Switch the primal Perceptron in the previous step to a dual Perceptron with a linear kernel. After observing T examples in the stream, will the two Perceptrons have learned the same prediction function?

(c) What computational issue will you encounter if you continue to run the dual Perceptron and allow T to approach ∞? Will this problem happen with the primal Percepton? Why or why not?

6) Robust SVM (6 points) For SVMs, the hinge loss can deal with some examples that are not linearly separable. However in some cases, these examples may be outliers: data points that are so inconsistent with the rest of the data that we suspect they are incorrect. A good classifier should be robust to those outliers. Is it possible for us to modify the hinge loss to achieve our goal? What is the disadvantage? (Hint: is it possible to choose a nonconvex function?)

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.