CS 5350/6350: Machine Learining Fall 2015

Homework 4
Christopher Mertin

Handed out: Nov 3, 2015
Due date: Nov 17, 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 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. 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 Warm up: Support Vector Machines

In class you have seen support vector machines (SVMs). Here we will find the optimal classifier of a simple dataset by hand. Assume that we are given the following dataset:

$$\mathcal{D}_+ = \begin{bmatrix} 3 & 1 \\ 3 & -1 \\ 6 & 1 \\ 6 & -1 \end{bmatrix} \qquad \mathcal{D}_- = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix}$$

where $\mathcal{D}_+$ are positive examples and $\mathcal{D}_-$ are negative examples.

1. [9 points] Find the optimal hyperplane and the corresponding maximum margin. Which training points are the support vectors?

    - The solution is the hyperplane about $y = 2$, which can be seen in Figure 1, where the support vectors are from the positive as $(3, 1)$ and $(3, -1)$ as they both have the same distance to the hyperplane, and for the negatives $(1, 0)$. It is the maximized margin between both points (the midpoint between them) as the distance of each closest point to the hyperplane is 1.

2. [2 points] You are given a new point $x = [1.8, 1]$ with true label -1. Does your SVM correctly classify this point?

   - Yes, please refer to Figure 2

3. [4 points] Now let's compare this classifier to the vanilla Perceptron algorithm. Assume that we have the point $x = [1.9999, 1]$ with label -1. If we allow the Perceptron algorithm to run until it achieves 0% error on the training set will it be guarenteed to classify this point correctly? Why or why not justify your answer.

   - I talked with Dr. Srikumar and he said that this question is to be taken as you *learn* on the original 8 points, and the $(1.999, 1)$ point is to be the "test" point. The vanilla Perceptron algorithm will not necessairly classify the last point correctly. The normal Perceptron algorithm does not care about the resulting weight vector, only that the training set is classified correctly. Figure 3 shows and example of how the training set could be classified correctly, but it wouldn't classify the test point $(1.999, 1)$ correctly.

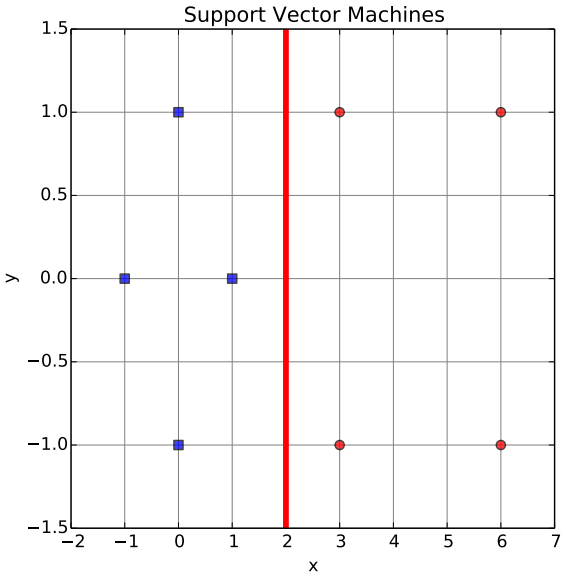

Figure 1: Solution to Problem 1

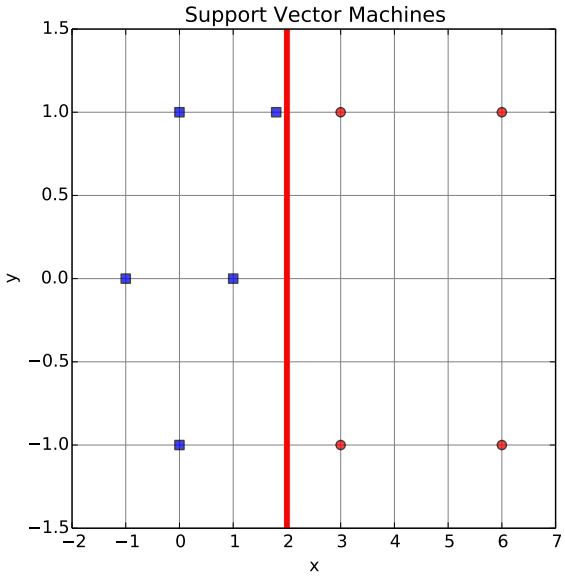

Figure 2: Solution to Problem 2

2 Kernels and the Perceptron algorithm

In the class, we saw how the idea of kernels can be applied to increase the expressiveness of the SVM classifier. Kernels is more broadly applicable than just SVMs. In this question, we will see how kernels can be used with the Perceptron algorithm.

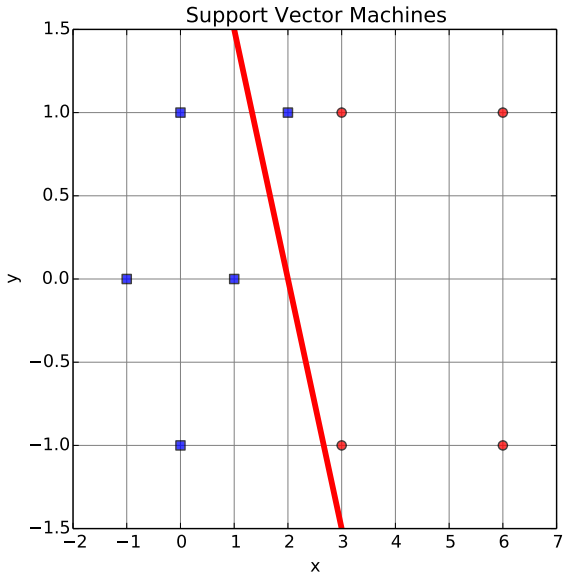

Figure 3: Solution to Problem 3

Suppose, we wish to learn a k-DNF (not necessarily monotone) using a kernel version of Perceptron. Recall that a k-DNF is a Boolean function that is a disjunction of conjunctive clauses, where each conjunctive clause has exactly k literals. For example, the following Boolean functions are 2-DNFs:

- $(x_1 \wedge x_2) \vee (\neg x_2 \wedge x_3)$
- $(\neg x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee (\neg x_4 \wedge x_1)$

In order to learn a k-DNF, we will define a feature transformation $\phi(\mathbf{x})$ that maps examples $\mathbf{x} \in \{0,1\}^n$ to a new space of k-conjunctions and then define a kernel Perceptron algorithm for learning. That is, each element of $\phi(\mathbf{x})$ corresponds to the value of a different k-conjunction and ϕ enumerates over all k-conjunctions. After this transformation, the classification of the Perceptron is via the sign of the dot product of the weight vector and the example $\mathbf{w}^T\phi(x)$. The goal is to represent this dot product using a kernel $K(w, x)$.

1. [5 points] Show that any k-DNF is linearly separable after the feature transformation.

   - As was proven in the last homework, k-Decision Lists (k-DL) are linearly separable. Therefore, this can be proven by showing that k-DNF functions are similar to that of k-DL's.

     Assume we can have an arbitrary k-DNF of the form $f = \vee_{i=1}^{N} c_i$, where c_i is a k-conjunction. We can turn this into a k-DL by the following form $C = \{(c_1, 1), (c_2, 1), \ldots, (c_N, 1), 0\}$. From this definition, it is easy to see that *both* C and f are satisfied as long as one of the k-conjunctions are true. Therefore, the k-DL and k-DNF behave in the same way, so the new k-DL can be used to make it linearly separable.

2. [10 points] Let C be the set of all conjunctions containing exactly k different literals. (Recall that a literal is a Boolean variable or its negation.) For any $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$, define

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2) = \sum_{c \in C} c(\mathbf{x}_1) c(\mathbf{x}_2).$$

Here $c(\mathbf{x})$ is the value of a conjunction c using the values in $\mathbf{x}$. Show that this function can be computed efficiently without explicitly computing the values of $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$.

- The function can be efficiently computed in *linear time* as follows.
  There are $\mathcal{O}(N)$ k-conjunctions $\in C$, which would be the upper bound on simply calcualting each of the conjunctions istead of worrying about mapping to the new space and taking the dot product. Since they are conjunctions, $c(\mathbf{x}_1)\, c(\mathbf{x}_2) = 1$ when $c(\mathbf{x}_1)$ *and* $c(\mathbf{x}_2)$ are *both* true. Therefore, instead of mapping all of the data to the new feature space and taking the dot product, *at most* linear time can be achieved by simply looking at the ones with the same number of active features between the two sets and summing over those which agree.

3. [5 points] Assume that the initial weight vector for Perceptron is the zero vector. Then, show that $\mathbf{w} = \sum_{(\mathbf{x}_i, y_i) \in M} y_i\, \phi(x_i)$. Here $M = \{(\mathbf{x}_i, y_i)\}$ is the set of examples on which the learning algorithm made mistakes.

- The Perceptron algorithm only updates everytime it makes a mistake. To update the weight vector, it updates it as $\mathbf{w}_{i+1} = \mathbf{w}_i + y_i \mathbf{x}_i$. If we have a function $\phi(\mathbf{x})$ that maps $\mathbf{x}$ to a new feature space, we have $\mathbf{w}_{i+1} = \mathbf{w}_i + y_i \phi(\mathbf{x}_i)$. If the Perceptron algorithm makes M errors, the resulting weight vector can be reprsented as $\mathbf{w} = \sum_{(\mathbf{x}_i, y_i) \in M} y_i \phi(\mathbf{x}_i)$.

4. [5 points] Using the fact that the weight vector can be written as in the previous question, write the classification step $y = sgn\left(\mathbf{w}^T \phi(\mathbf{x})\right)$ using the kernel function $\kappa(\mathbf{x}_1, \mathbf{x}_2)$ instead of using the explicit feature representations.

- From the question we have

$$y = sgn\left(\mathbf{w}^T \phi\left(\mathbf{x}\right)\right)$$

Which can be generalized by plugging in for $\mathbf{w}$ from the question before as

$$y = sgn\left(\sum_{(\mathbf{x}_i, y_i) \in M} y_i\, \phi(\mathbf{x}_i)^T \phi(\mathbf{x})\right)$$

We also know from the notes that $\kappa(\mathbf{x}, \mathbf{z}) \equiv \phi(\mathbf{x})^T \phi(\mathbf{z})$. So we can use this to rewrite it in the form

$$y = sgn\left(\sum_{(\mathbf{x}_i, y_i) \in M} y_i\, \kappa(\mathbf{x}_i, \mathbf{x})\right)$$

5. [15 points] Write explicitly the pseudo code for the kernel Perceptron algorithm that uses your kernel to learn a k-DNF. (Hint: Instead of storing the weight vector, you will store a list of examples where the algorithm makes mistakes.)

   - The given problem states that the initial weight vector is the zero vector, therefore there is no bias term to account for in the update, so it will be ignored.

Algorithm 1 Zero'd Kernel Perceptron

$\mathbf{w} \leftarrow 0$
for $i = 1$ **to** M **do**
  if $y \neq sgn\left(\sum_{\mathbf{x}_i, y_i \in M} y_i \kappa(\mathbf{x}, \mathbf{x}_i)\right)$ **then**
    $\mathbf{w} \leftarrow \mathbf{w} \cup (\mathbf{x}_i, y)$
  end if
end for
return $sgn\left(\sum_{\mathbf{x}_i, y_i \in M} y_i \kappa(\mathbf{x}, \mathbf{x}_i)\right)$

3 Experiment: Training an SVM classifier

Recall from class that an SVM learns a classifier by minimizing the following loss function:

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

where C is a hyper-parameter that controls the relative importance of the regularization term $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ 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 $\mathbf{w} = \vec{0}$, $t = 0$
2. for epoch $1 \ldots T$:
   (a) for example $(\mathbf{x}_i, y_i)$ for every i (random order)
      i. r_t = Learning rate at t
      ii. $\mathbf{w} = \mathbf{w} - r_t \nabla E(\mathbf{w}, \mathbf{x}_i, y_i)$
      iii. $t = t + 1$

Here, the gradient is defined as follows:

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

(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, one successful strategy is to choose r 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.

Data

For this home work you will be working on two data sets.

1. The data files named `train0.10` and `test0.10` from data0 folder present in hw2.

2. The data set named astro included as part of this assignment. There are two variants of this dataset – `original` and `scaled`, each with train and test files. The scaled data set is a feature-normalized version of the original dataset.

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 and Reporting

1. [7 points] Implement the SVM learner using stochastic sub-gradient descent as a training algorithm. In your report, briefly describe any design choices you make in implementing the classifier.

    - A major design choice that was made, was that the transformed files such as `astro/original/train.transform` don't actually exist. They are calculated correctly, but they are not written to a file and instead everything is stored in memory. This was to keep the computation time down as file I/O is a major bottleneck.

      The SVM that was implemented uses Python and the NumPy library so that mathematical operations on vectors would already be implemented. This reduced the size of the code and the possibility for bugs in the mathematics.

2. [3 points] Construct a feature transformation that maps all points from the original space to the space of all products of features. That is, for every pair of original features x_i and x_j, you will have feature $x_i x_j$ in the transformed set. (This will include the square terms such as x_i^2 as well.) Apply the transformation to the `original` and the `scaled` datasets. So you will have four cases: `original`, `scaled`, `original.transformed`, and `scaled.transformed`.

   Report the distance of the farthest data point from the origin for each of these four datasets.

   - The points that were found to be furthest from the origin are as follows:
     - `astro/original/train`:
       ```
       [  1.34611000e+02   5.81073100e+02  -1.44179400e-01   1.66310800e+02]
       Distance from origin: 619.214
       ```
     - `astro/original/train.transform`:
       ```
       [  1.81201213e+04   7.82188311e+04  -1.94081332e+01   2.23872631e+04
          7.82188311e+04   3.37645948e+05  -8.37787709e+01   9.66387321e+04
         -1.94081332e+01  -8.37787709e+01   2.07876994e-02  -2.39785914e+01
          2.23872631e+04   9.66387321e+04  -2.39785914e+01   2.76592822e+04]
       Distance from origin: 383425.372
       ```
     - `astro/scaled/train`:
       ```
       [-0.980394 -0.91751   0.932665  0.992551]
       Distance from origin: 1.913
       ```
     - `astro/scaled/train.transform`:
       ```
       [ 0.9611724   0.8995213  -0.91437917 -0.97309105
         0.8995213   0.8418246  -0.85572946 -0.91067547
        -0.91437917 -0.85572946  0.869864    0.92571758
        -0.97309105 -0.91067547  0.92571758  0.98515749]
       Distance from origin: 3.658
       ```

3. [20 points] Run the following experiment for the above 4 datasets and also `data0.10`.

   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 $C \in \{0.001, 0.01, 0.1, 1, 10\}$. Feel free to expand the set of hyperparameters.

   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 during cross-validation.)

   - The table below summarizes the results of the best hyper-parameters from cross validation

Table 1: Results for each file with hyper-parameters

	original/train	original/train.transform	scaled/train	scaled/train.transform	data0/train0.10
$Acc.^{(1)}$	0.833	0.688	0.692	0.592	0.812
$C^{(1)}$	0.010	10.000	100.000	100.000	10.000
$\rho_0^{(1)}$	0.001	1.000	0.100	1.000	1.000
$Acc.^{(2)}$	0.832	0.688	0.663	0.590	0.791
$C^{(2)}$	0.001	10.000	100.000	100.000	100.000
$\rho_0^{(2)}$	0.100	1.000	0.001	0.100	0.001
$Acc.^{(3)}$	0.816	0.675	0.656	0.585	0.772
$C^{(3)}$	0.010	0.001	100.000	100.000	10.000
$\rho_0^{(3)}$	1.000	0.001	0.010	0.001	0.010
$Acc.^{(4)}$	0.799	0.663	0.632	0.576	0.686
$C^{(4)}$	0.010	0.100	100.000	100.000	10.000
$\rho_0^{(4)}$	0.010	0.100	1.000	0.010	0.100
$Acc.^{(5)}$	0.776	0.659	0.597	0.415	0.679
$C^{(5)}$	0.001	10.000	10.000	10.000	10.000
$\rho_0^{(5)}$	0.100	0.010	1.000	0.001	0.100

4. [15 points] For each dataset, 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 along with the margin of the trained weight vector.

   - The table below summarizes the results

Table 2: Final results

File	*Acc.*	C	ρ_0	$Margin$
`original/train`	0.854	0.010	0.001	12.425
`original/train.transform`	0.648	10.000	0.100	151.747
`scaled/train`	0.505	100.000	0.100	0.040
`scaled/train.transform`	0.615	100.000	1.000	0.149
`data0/train0.10`	0.875	10.000	1.000	0.534

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.