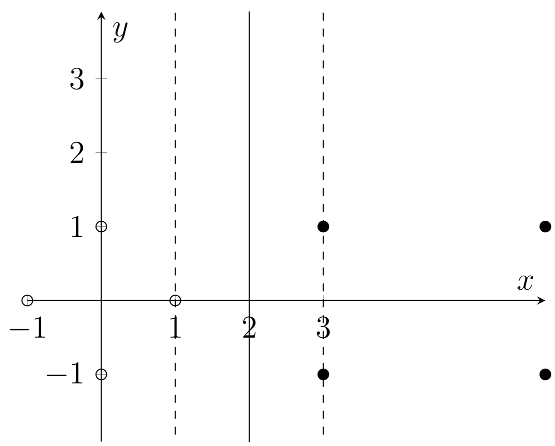
CS 5350/6350: Machine Learining Fall 2015

Homework 4

Handed out: Nov 3, 2015
Due date: Nov 17, 2015

1 Warmup: Support Vector Machines

1. 

Solution: $\mathcal{D}_+$ are represented by filled dots and $\mathcal{D}_-$ are represented by empty dots.The optimal hyperplane is represented by the line at $x = 2$. The maximum margin is 1.
The points which lie on the margin are the support vectors. The points are: $(1, 0), (3, 1)$ and $(3, -1)$.

2.

Solution: The new point $x = [1.8, 1]$ with true label -1 will lie on the negative labels side. Hence SVM correctly classifies it.

3.

Solution: If a vanilla perceptron gives a classifier which achieves 0% error on the training set , we can not guarantee that it will correctly classify the point $x = [1.9999, 1]$ with label -1. This is because the perceptron may give classifier which linearly separates the training set, but the classifier may not necessarily be $x = 2$. We can still get a classifier which linearly separates the training set and and is not $x = 2$. For e.g. if we get a classifier: $x = c$ where c is constrained by the condition $1 < c < 1.8$. Any of these classifiers will misclassify the given point even after giving 0% error on training set.

Any classifier can be given as perceptron does not have a regularization term to maximize the margin.

2 Kernels and the Perceptron Algorithm

1.

Solution: In a k-DNF, there are exactly k literals. As we know that a k-DNF is a disjunction of the conjunctions of a set of exact k literals. We know that any disjunction is linearly separable with a unit weight vector. Similary if we take all the conjunctions in the DNF as features for the feature transformation we will get a simple disjunction, that is linearly separable by a unit weight vector.

2.

Solution: We are given that C is the set of all conjunctions containing exactly k different literals. We know that for any $\mathbf{x}_1, \mathbf{x}_2 \in \{0,1\}^n$

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

The value of conjunctions is given by $c(\mathbf{x})$. Therefore, when $c(\mathbf{x}_1) = 1$ and $c(\mathbf{x}_2) = 1$, only then will they contribute to the value of $K(\mathbf{x}_1, \mathbf{x}_2)$ as their product will also give 1.
We can also say that c is defined as where the value of conjunctions from both $\mathbf{x}_1$ and $\mathbf{x}_2$ are true. The number of elements which contribute to $K(\mathbf{x}_1, \mathbf{x}_2)$ is defined by $Same(\mathbf{x}_1, \mathbf{x}_2)$. We now pick k elements from this set and sum over them to compute the value of $K(\mathbf{x}_1, \mathbf{x}_2)$. Hence:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{c \in C} \binom{Same(\mathbf{x}_1, \mathbf{x}_2)}{k}$$

It will take linear time as there are n comparisons at maximum to find the elements in each set which have the same value. Hence this function can be efficiently computed without explicitly computing the values of $\phi(\mathbf{x}_1)\phi(\mathbf{x}_2)$.

3.

Solution: In perceptron algorithm weight vector is updated for every mistake for learning rate 1, defined by, $w_{t+1} = w_t + y_i x_i$

As the features are transformed, represent the feature transformation of a single example by $\phi(x_i)$. Hence, the above equation can be rewritten as , $w_{t+1} = w_t + y_i \phi(x_i)$

Now taking the initial weight vector as a zero vector, we can perform a manual run on the algorithm.

For mistake 1, $w_1 = 0 + y_1\phi(x_1)$

For mistake 2, we can write,
$w_2 = w_1 + y_2\phi(x_2) = y_1\phi(x_1) + y_1\phi(x_1)$

Going on till t mistakes we get,
$w_t = w_{t-1} + y_t\phi(x_t)$
$w_t = \sum_i^{t-1} y_i\phi(x_i) + y_t\phi(x_t)$,
So we can simplify the above equation to:

$\mathbf{w} = \sum_{(\mathbf{x}_i, y_i) \in M} y_i \, \phi(x_i)$

4.

Solution: Using the equation from previous solution we can replace $\mathbf{w}^T$ in the following equation

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

We know that $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2)$. Therefore applying the definition to the equation we get,

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

5.

Algorithm 1 Pseudocode of kernel Perceptron to learn k-DNF

Start with M equal to the empty set
for every (x_i, y_i) in examples **do**
  $y' = sgn\left(\sum_{(\mathbf{x}_j, y_j) \in M} y_j K(\mathbf{x}_j, \mathbf{x}_i)\right)$
  if $y' \neq y_i$ **then**
    Add (x_i, y_i) to M
    $\mathbf{w} = \sum_{(\mathbf{x}_i, y_i) \in M} y_i \phi(x_i)$
  end if
end for

Solution: Lets take a set M which will store the examples on which mistakes are made. From above question 3 and 4 we can deduce that the similar calculation happens in case of k-DNF. The pseudo code is defined above.

3 Experiment: Training an SVM classifier

1.

Solution: I have kept the number of outside the code to keep similarity between perceptron and SVM learner using stochastic sub-gradient descent. The values of C are $\{0.0001, 0.001, 0.01, 0.1, 1, 10.0, 20.0\}$ and $\rho_0 = \{0.0001, 0.001, 0.01, 0.1, 1\}$. As the data0 was linearly separable i tried out higher values of C to get perfect accuracy.

2.

Solution: For data0:
The farthest point from the origin at a distance of 2.68649 is:

1 0:1 1:0.743593 2:0.973706 3:0.923739 4:0.897686 5:0.850652 6:0.986938 7:0.79702 8:0.892886 9:0.222102 10:0.936804

For astro/original:
The farthest point from the origin at a distance of 619.214 is:

1 0:1 1:134.611 2:581.073 3:-0.144179 4:166.311

For astro/scaled:
The farthest point from the origin at a distance of 1.91259 is:

-1 0:1 1:-0.980394 2:-0.91751 3:0.932665 4:0.992551

For astro/original.transformed:
The farthest point from the origin at a distance of 362018 is:

1 0:1 1:134.611 2:581.073 3:-0.144179 4:166.311 5:18120.1 6:78218.8 7:-19.4081 8:22387.3 9:337646 10:-83.7788 11:96638.7 12:0.0207877 13:-23.9786 14:27659.3

For astro/scaled.transformed:
The farthest point from the origin at a distance of 3.4682 is:

-1 0:1 1:-0.980394 2:-0.91751 3:0.932665 4:0.992551 5:0.961172 6:0.899521 7:-0.914379 8:-0.973091 9:0.841825 10:-0.855729 11:-0.910675 12:0.869864 13:0.925718 14:0.985157

3.

Solution: For data0:

Initial Rate	**C**	**Average Accuracy**
0.001	20	1
0.01	20	0.985
0.1	20	0.97
1	20	0.969
0.0001	20	0.922

Accuracy on training set: 1

Accuracy on test set: 1

Margin: 0.0380758

For astro/original:

Initial Rate	C	Average Accuracy
0.1	1	0.840584
0.001	1	0.838636
1	20	0.837662
0.001	10	0.825974
0.0001	1	0.819805

Accuracy on training set: 0.823567

Accuracy on test set: 0.823567

Margin: 0.0166075

For astro/scaled:

Initial Rate	C	Average Accuracy
0.0001	20	0.805195
1	20	0.801299
0.01	20	0.797403
0.001	20	0.79026
0.1	20	0.771753

Accuracy on training set: 0.83943

Accuracy on test set: 0.83943

Margin: 0.000963625

For astro/original.transformed:

Initial Rate	C	Average Accuracy
1	1	0.859091
0.0001	0.1	0.854221
0.0001	0.01	0.849675
0.0001	10	0.848052
0.001	0.01	0.838312

Accuracy on training set: 0.767562

Accuracy on test set: 0.767562

Margin: 0.497844

For astro/scaled.transformed:

Initial Rate	**C**	**Average Accuracy**
0.01	20	0.857792
0.001	20	0.856494
0.1	20	0.843831
0.0001	20	0.839935
1	20	0.832468

Accuracy on training set: 0.912269

Accuracy on test set: 0.912269

Margin: 8.65407e-05