CS273a Homework #2

Introduction to Machine Learning: Fall 2012

Due: Friday October 19th, 2012

Write neatly (or type) and show all your work!

Problem 1: Perceptron Algorithm

In this problem, we'll explore a basic perceptron algorithm on separable and non-separable data. In the next problem, you will extend this implementation to optimize over smooth loss functions. Download the associated homework code, and replace last week's code (a few functions have been updated).

We'll start by building two binary classification problems, one separable and the other not:

```
iris=load('data/iris.txt');     % load the text file
X = iris(:,1:2); Y=iris(:,5);   % get first two features
XA = X(Y<2,:); YA=2*Y(Y<2)-1;   % get class 0 vs 1 and convert to +/- 1
XB = X(Y>0,:); YB=2*Y(Y>0)-3;   % get class 1 vs 2 and convert to +/- 1
```

(a) Show the two classes in a scatter plot and verify that one is linearly separable.

(b) Build a linear classifier using the basic perceptron algorithm I gave in class:

```
pc = perceptClassify(Xp,Yp, step,tol,nIter);
% you can explore the various options and code
```

Run it on each data set. The code should plot both the data points and the decision boundary at each iteration. Compute, and add another plot that shows, the value of the misclassification rate as a function of iteration. When you have decided on some good parameter values, show the error rate plots for both data sets and comment on their behavior and convergence.

Note that, unlike my `linearRegress` class, `perceptClassify` does add its own constant feature, so you do not need to do this to the input features.

(c) You can also set the perceptron weights manually. Instead of running the perceptron algorithm, try computing the coefficients of a linear *regression* of the data. Use these coefficients within your classifier (`setWeights`) and compare its performance.

(d) On the linearly separable data, add another point at $x = (-10, 10)$ with class $y = -1$. Try both the linear regression and perceptron algorithm again. Now how do they compare? What happened?

Problem 2: Linear classifier with logistic-MSE loss

In this problem you will extend the perceptron classifier code from Problem 1 to optimize the mean squared error loss on a "soft" logistic threshold prediction.

(a) Fill in the missing components of the `logisticMseClassify` function `train`. Your algorithm should use "online" (or "stochastic") gradient descent, in which parameters are updated for each data point, as discussed in class and on the slides. This also mirrors the behavior of the basic perceptron algorithm in `perceptClassify`, for comparison.

In class I gave the gradient for the logistic MSE loss assuming predictions of 0/1 values; here for consistency with the perceptron we're using ± 1. The logistic prediction function "scaled" to ± 1 is

$$\rho(z) = 2\sigma(z) - 1$$

where $z = \theta x^{(i)}$ is the linear response of the perceptron, and σ is the standard logistic function

$$\sigma(z) = \big(1 + \exp(-z)\big)^{-1}.$$

The logistic MSE loss function is then

$$J(\theta) = \big(\rho(z) - y^{(i)}\big)^2$$

and its gradient (up to a constant) is

$$\nabla J(\theta) = (\rho(z) - y^{(i)})\ \sigma(z)(1 - \sigma(z))\ x^{(i)}.$$

(b) To test your algorithm, you may want to start with a simple 1D data set:

```
X1 = [0.05 0.24 0.28 0.53 0.61 0.48 0.58 0.76 0.80 0.95]';
Y1 = [-1   -1   -1   -1   -1   1    1    1    1    1]';
lc = logisticMSE(X1,Y1 ... );
```

The included plotting functions should help you visualize the evolution of your classifier, so that you can see it evolve to a "visually reasonable" set of predictions.

(c) Run your algorithm on both sets `XA,YA` and `XB,YB` from the previous problem. Again, comment on their behavior, comparing to the behavior of the basic perceptron algorithm.

(d) In your code, at each full iteration of your algorithm (after each time through all the data), compute your misclassification rate and also the value of the logistic MSE loss $J(\theta)$. Plot them both as a function of the iteration on `XB,YB` and compare their behavior.

(e) Derive the gradient of the regularized hinge loss,

$$J(\theta) = \max\big[0\,,\,1 - y(i)\theta x^{(i)}\big] + \lambda\,\theta\theta'$$

and modify your training function to use this loss instead. Run your modified algorithm on `XA,YA` and `XB,YB` and compare its behavior.

Problem 3: Features

In this problem we will explore the creation of new features to increase the complexity of our classifier and its decision boundaries.

(a) Using data set `XB,YB` from before, try adding polynomial features into the classifier and visualize the new classification boundary. You can do this using an updated feature in `plotClassify2D` that enables a pre-processing function, e.g.,

```
pre = @(x) fpoly(x,degree,false);     % creates implicit function handle to call fpoly
lc = logisticMSE( pre(XB), YB, ... ); % call training function with processed features
plotClassify2D(lc,XB,YB,pre); % plot (XB,YB) with decision regions of predict(lc,pre(X),Y)
```

Evaluate a few values of the degree $(1, 2, 3)$ and compare their resulting decision boundaries. If you do not have confidence that your solution to Problem 2 is working, you can use the perceptron classifier provided from Problem 1.

(b) Load the provided MNIST data, which correspond to images of handwritten digits. We'll select out a binary classification problem between zeros and eights:

```
mnist=load(data/mnist.txt'); X=mnist(:,2:end); Y=mnist(:,1);  % load data
id=(Y==0 | Y==8); X08 = X(id,:); Y08 = Y(id);      % select out 0/8 comparison
% use reorder and split to randomize the order and split 75/25 training/test
```

Here we have 784 features and only 200 data, so we are likely to overfit. Learn a linear classifier and compare the training and test MSE of what you learn.

(c) We will use random features to explore increasing complexity with number of features. The provided function `fkitchensink` can produce various randomly generated non-linear features, including sigmoidal (logistic) responses:

```
[Xk F] = fkitchensink(X, K, 'sigmoid');  % create K random features and store as F
Xk2 = fkitchensink(X2, K, 'sigmoid', F); % generate the same features on new data
```

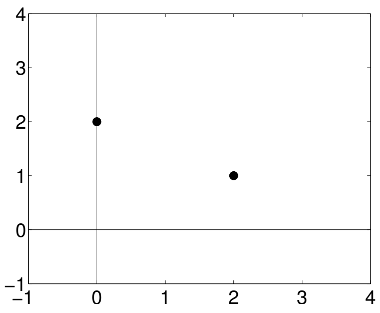
Try training a linear classifier with slowly increasing values of $K = [1, 5, 10, 25, 50, 100, 200]$ and compute their training and test errors (misclassification rates). Plot the error rates as a function of K and assess if and where you are over- and under-fitting the data. What is the best (estimated) error rate you achieve?

As a side note, these sigmoidal features can be thought of as a "poor man's" 2-layer neural network. The sigmoid form is identical to that of a single layer of a feed-forward neural network; using them as inputs to a perceptron creates a two-layer neural network. The only difference is that here, we have only trained the second layer; the first (randomly generated) layer is not updated during the learning process.

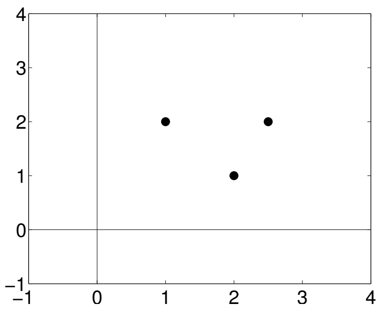
Problem 4: Shattering and VC Dimension

Consider the following learners and data points, which have two real-valued features x_1, x_2. Which of the following three examples can be shattered by each learner? Give a brief explanation / justification and use your results to guess the VC dimension of the classifier.

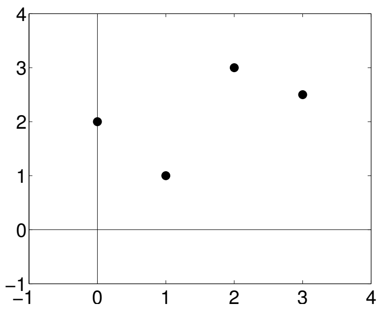
Data points:



(#1)



(#2)



(#3)

Let $T(z) = \text{sign}(z) = \begin{cases} +1 & z \geq 0 \\ -1 & z < 0 \end{cases}$. Variables a, b, c indicate parameters of the learner.

(a) $T(\,a + bx_1\,)$

(b) $T(\,(x_1 - a)^2 + (x_2 - b)^2 + c\,)$

(c) $T(\,(a * b)x_1 + (c/a)x_2\,)$