CS 189: Introduction to Machine Learning - Discussion 2

1. Gradient descent

   Gradient descent is a very popular method of optimization, in particular, for finding the minimum value of a convex function. Suppose you wish to find the point at which a function $f : \mathbb{R}^d \to \mathbb{R}$ attains its minimum. The gradient descent algorithm begins with an initial "guess" x^0. Then at every time $t \in \{1, 2, 3, \dots\}$ the algorithm computes an updated estimate x^t as

$$x^t = x^{t-1} - \eta \nabla f(x) \mid_{x=x^{t-1}} .$$

   Here, $\eta > 0$ is some real valued parameter known as the "learning rate" or the "step size". The expression $\nabla f(x) \mid_{x=x^{t-1}}$ denotes the gradient of the function f at the point x^{t-1}. If you do not know what a gradient is, please look it up. In this question, however, we will deal with the case when $d = 1$ and when the function f is differentiable. In this case, the gradient simply equals the derivative of f.

   The point of this exercise is to evaluate the effect of the choice of the parameter η on the performance of the gradient descent algorithm. Suppose

$$f(x) = x^2 - 3x + 4.$$

   Suppose you run the gradient descent algorithm to find the minimum of this function with the starting point as $x^0 = 10$. Evaluate the first 5 steps of the gradient descent algorithm when: *(a)* $\eta = 5$, *(b)* $\eta = 0.5$, *(c)* $\eta = 0.05$. Try to draw inferences about the performance of the algorithm each time.

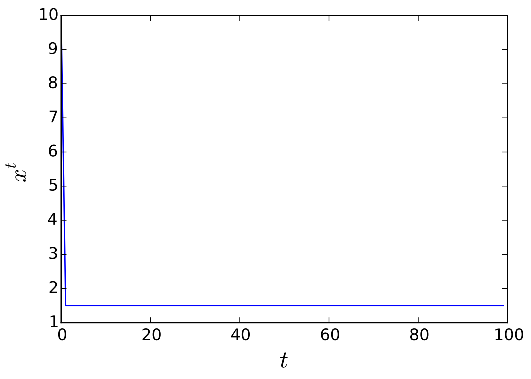
   Now suppose the parameter η is not fixed for the entire algorithm, but is allowed to be changed at every iterations. In other words, we have a parameter η_t associated with iteration t of the algorithm and $x^t = x^{t-1} - \eta_t \nabla f(x) \mid_{x=x^{t-1}}$. Evaluate the first 5 steps of the gradient descent algorithm when *(d)* $\eta_t = \frac{1}{t^2}$, *and (e)* $\eta_t = \frac{1}{t}$. Again, try to understand, at a conceptual level, the effect of various choices of η_t.

Solution:

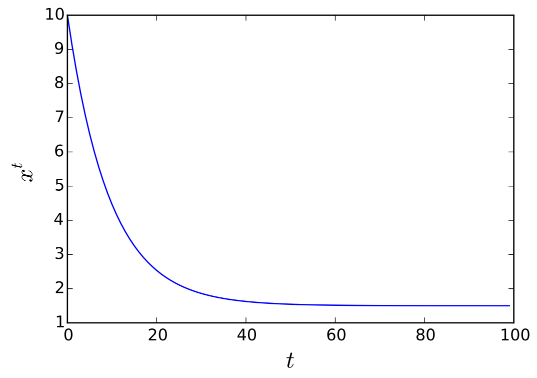
a 10, -75, 690, -6195, 55770, -501915,...

Diverges to infinity.

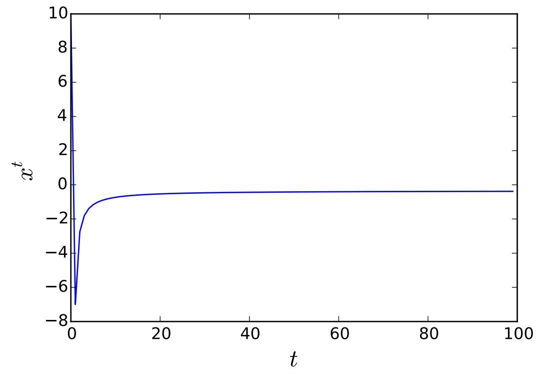
b 10, 1.5, 1.5, 1.5, 1.5, 1.5,...



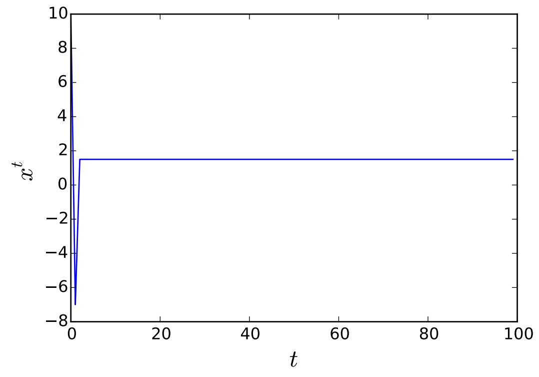
c 10, 9.15, 8.385, 7.6965, 7.07685, 6.519165,...



d 10, -7, -2.75, -1.80555556, -1.39236111, -1.16097222,...



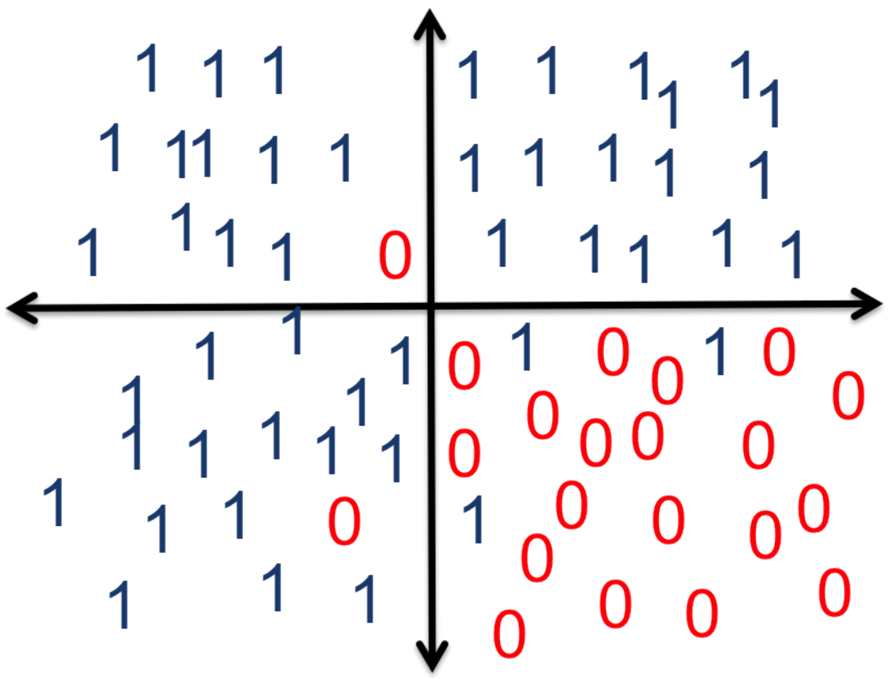
e 10. -7. 1.5, 1.5, 1.5, 1.5,...



You see that the step size should be chosen carefully. If it is too small (e.g., .5 or $\frac{1}{t^2}$) then the algorithm may not reach the minimum value, or reach it very slowly. On the other hand, if the step size is too large (e.g., 5) then the algorithm may diverge.

2. Overfitting, model selection and regularization

Consider the set of training points in the following figure.



As you can see from the figure, there are two classes 0 and 1 and two features. The goal is to build a classifier such that the decision boundary is *piecewise linear* in these two features. Let us first describe the ground truth to allow for an evaluation of the various classifier we will study. Note that in practice, you will not know this ground truth, and this ground truth is what your classification algorithm needs to infer. Points in the fourth quadrant belong to Class 0 while the points in the first three quadrants belong to class 1. When an observation is made, class label for a point near the boundary may get flipped because of some random noise.

Coming back to the real world where we only have observations and not the ground truth, let n denote the number of training samples, and for the k^{th} sample, let x^k denote the feature vector and y^k its label. For any piecewise linear classifier $f : \mathbb{R}^2 \to \{0, 1\}$, we will measure its performance on the training set in terms of the following *penalty*:

$$\sum_{k=1}^{n} \mathbf{1}\{f(x^k) \neq y^k\} + \lambda \, \texttt{num\_components}(f).$$

Here, $\texttt{num\_components}(f)$ denotes the number of piecewise linear components of the decision boundary given by f.

For each of the following values of λ, find a piecewise-linear classifier that leads to the minimum penalty on the training set.

a $\lambda = 0.01$

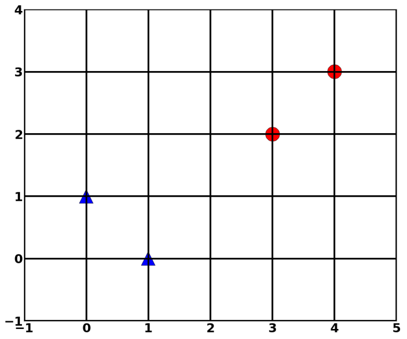
b $\lambda = 10$

c $\lambda = 1$

How do these classifiers compare to the ground truth? What inferences did you draw about "overfitting" and "model selection" from this exercise?

Solution: When $\lambda = 0.01$, there is very little penalty on a more complex model. The optimal piecewise-linear decision boundary here is the one which exactly fits the training data. Thus there is no error in the training data but the test data will contain errors due to this overfitting. When $\lambda = 10$, there is a large enough penalty on the number of components to mandate a linear classifier. There will thus be many errors at test time. This choice of λ prevents the choice of a good model. Finally, when $\lambda = 1$, an optimal decision boundary is the true boundary. In this example, this choice of λ offers a good compromise between the fitting and the model complexity.

3. Fun with the SVM margin. (Do not worry if you do not understand every part of this question completely. We will be doing SVMs in detail later in the semester.)
   a. We typically frame an SVM problem as trying to *maximize* the margin. Explain intuitively why a bigger margin will result in a model that will generalize better, or perform better in practice.
   b. Show that the width of an SVM margin with linearly separable data is $\frac{2}{\|w\|}$.
   c. You're presented with the following set of data (triangle = +1, circle = -1):



   Find the equation (by hand) of the hyperplane $\vec{w}^T x + b = 0$ that would be used by an SVM classifier. Which points are support vectors?

Solution:

a One intuition is that if points are closer to the border, we are less certain about their class. Thus, it would make sense to create a boundary where our "certainty" is highest about all the training set points. Another intuition involves thinking about the process that generated the data we are working with. Since it's a noisy process, if we drew a boundary close to one of our training points of some class, it's very possible that a point of the same class will be generated across the boundary, resulting in an incorrect classification. Therefore it makes sense to make the boundary as far away from our training points as possible.

b The width of the margin is defined by the points that lie on it, also called support vectors. Let's say we have a point, $\vec{x}'$, which is a support vector. The distance between $\vec{x}'$ and the separating hyperplane can be calculated by projecting the vector starting at the plane and ending at $\vec{x}$ onto the plane's unit normal vector. The equation of the plane is $\vec{w}^T\vec{x} + b = 0$. Since $\vec{w}$ by definition is orthogonal to the hyperplane, we want to project $\vec{x}' - \vec{x}$ onto the unit vector normal to the hyperplane, $\frac{\vec{w}}{\|\vec{w}\|}$.

$$\frac{\vec{w}^T}{\|\vec{w}\|}(\vec{x}' - \vec{x}) = \frac{1}{\|\vec{w}\|}(\vec{w}^T\vec{x}' - \vec{w}^T\vec{x}) = \frac{1}{\|\vec{w}\|}(\vec{w}^T\vec{x}' + b - \vec{w}^T\vec{x} - b)$$

Since we set $\vec{w}^T\vec{x}' + b = 1$ (or -1) and by definition, $\vec{w}^T\vec{x} + b = 0$, this quantity just turns into $\frac{1}{\|\vec{w}\|}$, or $\vec{1}\|\vec{w}\|$, so the distance is the absolute value, $\frac{1}{\|\vec{w}\|}$.

Since this is half the margin, we double it to get the full width of $\frac{2}{\|\vec{w}\|}$.

c The equation of the hyperplane will pass through point $(2, 1)$, with a slope of -1. The equation of this line is $x_1 + x_2 = 3$. We know that from this form, $w_1 = w_2$. We also know that the at the support vectors, $w^T x + b = \pm 1$. This gives us the equations:

$$1w_1 + 0w_2 + b = 1$$

$$3w_1 + 2w_2 + b = -1$$

Solving this system of equations, we get $\vec{w} = [-\frac{1}{2}, -\frac{1}{2}]^T$ and $b = \frac{3}{2}$.

The support vectors are $(1, 0), (0, 1)$, and $(3, 2)$.