Homework 4

The questions below are due on Tuesday October 10, 2017; 11:00:00 PM.

1) INTRO TO LINEAR REGRESSION

So far, we have been looking classification, where predictors are of the form

$$y = sign(\theta^T x + \theta_0)$$

making binary classification as to whether example x belongs to the positive or negative class of examples.

In many problems, we want to predict a real value, such as the actual gas mileage of a car, or the concentration of some chemical. Luckily, we can use most of a mechanism we have already spent building up, and make predictors of the form:

$$y = \theta^T x + \theta_0$$

This is called a *linear regression* model.

We would like to learn a linear regression model from example. Assume X is a d by n array (as before) but that Y is a 1 by n array of floating-poing numbers (rather than +1 or -1). Given data (X, Y) we need to find θ, θ_0 that does a good job of making predictions on new data drawn from the same source.

We will approach this problem by formulating an objective function. There are many possible reasonable objective functions that implicitly make slightly different assumptions about the data, but they all typically have the form:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda R(\theta, \theta_0)$$

For regression, we most frequently use *squared loss*, in which

$$L_s(x, y, \theta, \theta_0) = (y - \theta^T x - \theta_0)^2$$

We might start by simply trying to minimize the average squared loss on the training data; this is called the *empirical risk*:

$$J_{emp}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0)$$

Later, we will add in a regularization term.

We will see later in this assignment that we can find a closed form matrix formula (requiring a matrix inverse) for the optimal θ in a linear regression formula, Being able to solve a machine-learning problem in closed form is very awesome! But inverting a matrix is computationally expensive (a bit less than $O(n^3)$) and so, as our data sets get larger, we will need to find some more efficient ways to approach it.

Let's start by thinking about gradient descent:

1) What is the gradient of the empirical risk with respect to θ? We can see that it is of the form:

$$\nabla_\theta J_{emp}(\theta, \theta_0) = -\frac{1}{n}\sum_{i=1}^{n} g(x^{(i)}, y^{(i)})$$

where $x^{(i)}, y^{(i)}$ are the i^{th} data point and its label.

Write an expression for $g(x, y)$ using the symbols: `x`, `y`, `theta` and `theta_0`. Remember that you can use `@` for matrix product, and you can use `transpose(x)` to transpose a vector.

$g(x, y) =$ `(2*(x@transpose(y)) - (2*x@transpose(x)@theta) - (`

Ask for Help

2) What is the gradient of the empirical risk with respect to θ_0? We can see that it is of a similar form:

$$\nabla_{\theta_0} J_{emp}(\theta, \theta_0) = -\frac{1}{n}\sum_{i=1}^{n} g_0(x^{(i)}, y^{(i)})$$

Write an expression for $g_0(x, y)$ using the symbols: `x`, `y`, `theta` and `theta_0`.

$g_0(x, y) =$ `2*(y - transpose(theta)@x - theta_0)`

Ask for Help

2) LISA AND ANDREY

Lisa Squares and Andrey Tikhonov are trying to find a good feature set for a linear regression problem. They have 100 training examples. Lisa suggests using the raw features, which are 2 dimensional. Andrey favors applying a polynomial feature function, which ends up with 50 dimensions. Which of the following are true:

Lisa's approach, without regularization, will tend to have low estimation error. True

Ask for Help

Andrey's approach, without regularization, will tend to have low estimation error. False

Ask for Help

Lisa's approach, without regularization, will tend to have low structural error. False

Ask for Help

Andrey's approach, without regularization, will tend to have low structural error. True

Ask for Help

Regularization will be especially important in Lisa's approach. False

Ask for Help

Regularization will be especially important in Andrey's approach. True

Ask for Help

3) ADDING REGULARIZATION

Note!!! Please do this problem after the next one (Minimizing Empirical Risk). It will make more sense.

Although we don't have the same notion of margin maximization as with the SVM formulation for classification, there is still a good reason to *regularize* or put pressure on the weight vector to prevent it from fitting the training data too closely, especially in cases where we have few data points and many features.

And, as it happens, this same regularization will help address a problem that you might have anticipated when finding the analytical solution for θ, which is that XX^T might not be invertible (where we are using the definition of X as in problem 4, where each column of X is a d-length vector representing a d-feature sample point).

Consider this matrix:

```
X = np.array([[1, 2], [2, 3],[3, 5], [1, 4]])
```

Is XX^T invertible? If not, what's the problem? Mark all that are true.

☐	It is invertible
☐	It is not invertible because X is not square
✓	It is not invertible because two columns of X are linearly dependent
✓	It is not invertible because the rows of XX^T are linearly dependent
✓	It is not invertible because n is smaller than d
☐	We cannot compute the transpose of X

Ask for Help

4) MINIMIZING EMPIRICAL RISK

We can also solve regression problems analytically.

Remember the definition of *squared loss*,

$$L_s(x, y, \theta, \theta_0) = (y - \theta^T x - \theta_0)^2$$

and *empirical risk*:

$$J_{emp}(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0)$$

Later, we will add in a regularization term.

For simplicity in this section, assume that we are handling the constant term, θ_0, by adding a dimension to the input feature vector that always has the value 1.

Let data matrix $Z = X^T$ be n by d, let output vector $T = Y^T$ be n by 1, and recall that θ is d by 1. Then we can write the whole linear regression prediction as $Z\theta$.

Note: this problem is usually formulated using a n by d matrix and an n by 1 vector, which are our Z and T above. The notes call this matrix and vector X and Y, but this usage is inconsistent with our previous definition of X; so we will stick with X being d by n.

1) Let T be the n by 1 vector of target values. Write an equation expressing the mean squared loss of θ in terms of Z, T, n, and θ. Hint: note that this loss ($J(\theta)$) is a scalar, a sum of squared terms divided by n; we can write it as $(W^T W)/n$ for a column vector W.

Enter your answer as a Python expression. You can use symbols `Z`, `T`, `n` and `theta`. Recall that `transpose(x)` for transpose of an array, `inverse(x)` for the inverse of an array, `norm(x)` for the length(norm) of a vector, and `x@y` to indicate a matrix product of two arrays.

$J(\theta) =$ `(transpose(T-Z@theta)@(T-Z@theta))/n`

Ask for Help

Now, how can we find the minimizing θ, given Z and T? Take the gradient (yes, even with a matrix expression), set it to zero(s) and solve for θ.

2) What's

$$\nabla_\theta J(\theta)$$

You can use matrix derivatives or, look at the elementwise derivation in the notes and deduce the matrix form.

$\nabla_\theta J(\theta) =$ `2*(transpose(Z)@Z@theta - 2*(transpose(Z)@T`

Ask for Help

3) What if you set this equation to 0 and solve for $\theta*$, the optimal θ? Hint: It's ok to ignore the constant scaling factor.

$\theta^* =$

○	$(Z^T T)^{-1}(Z^T Z)$
●	$(Z^T Z)^{-1} Z^T T$
○	$(Z Z^T)^{-1} Z T^T$

Ask for Help

4) Just converting back to the data matrix format we have been using (not transposed), we have

$\theta^* =$

○	$(XY^T)^{-1}(XX^T)$
○	$(X^TX)^{-1}X^TY$
●	$(XX^T)^{-1}XY^T$

Ask for Help

4) Given the data matrix `X` and outputs `Y` defined below, compute θ^*, using `np.dot`, `np.transpose` and `np.linalg.inv`.

```
# Enter an expression to compute the set th to the optimal theta
th = np.dot(np.linalg.inv(np.dot(X,X.T)), np.dot(X,Y.T))
```

Ask for Help

5) FLASHBACK TO CLASSIFICATION

We're gearing up to address both regression and classification using stochastic gradient descent. So, let's work on writing the gradient of the SVM loss function in a similar style in Python. Here's the objective function:

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_h(y^{(i)}(\theta^T x^{(i)} + \theta_0)) + \lambda\|\theta\|^2$$

where

$$L_h(v) = \begin{cases} 1 - v & \text{if } v < 1 \\ 0 & \text{otherwise} \end{cases}$$

We will need to compute $\frac{d}{d\theta}L_h(v)$. You can describe this gradient as a function of v and $\frac{d}{d\theta}v$. In the python below, `v` is the value of v and `dv` is $\frac{d}{d\theta}v$.

Here are definitions of the components of the svm objective. Note that these definitions use some numpy "tricks" that are worth understanding in detail.

```
def hinge(v):
    return np.where(v >= 1, 0, 1-v)
def hinge_loss(x, y, th, th0):
    return hinge(y*(np.dot(th.T, x) + th0))
def svm_obj(x, y, th, th0, lam):
    return np.mean(hinge_loss(x, y, th, th0), axis = 1, keepdims = True) + lam * np.linalg.norm(th)**2
```

Now let's compute the gradients with respect to θ and θ_0, making sure that they work for data matrices and label vectors. You can write one function at a time---some of the checks will apply to each function independently.

```
# Write a function that returns the gradient of hinge(v) in terms
# assume values of v and dv as input.
def d_hinge(v, dv):
    return np.where(v>=1, 0, -dv)

# Write a function that returns the gradient of hinge_loss(x, y,
# with respect to th
def d_hinge_loss_th(x, y, th, th0):
    return d_hinge(hinge_loss(x, y, th, th0))

# Write a function that returns the gradient of hinge_loss(x, y,
# with respect to th0
def d_hinge_loss_th0(x, y, th, th0):
    return np.where(y*(np.dot(th.T, x) + th0)>=1, 0, -y*th0)

# Write a function that returns the gradient of svm_obj(x, y, th,
# respect to th
def d_svm_obj_th(x, y, th, th0, lam):
    return d_hinge_loss_th(x, y, th, th0) + 2*lam*np.linalg.norm(

def d_svm_obj_th0(x, y, th, th0, lam):
    return d_hinge_loss_th0(x,y,th, th0)
```

Ask for Help