

## ICS273A: HW3

Due: Feb 20

**Problem I:** Gaussians. Recall the multivariate gaussian for a vector  $x \in R^n$ .

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{(n/2)}|\Sigma|^{1/2}} \exp -\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu) \quad (1)$$

- A Let  $y = v_i^T x$ , where  $v_i$  is an eigenvector of  $\Sigma$  with an eigenvalue of  $\lambda_i$ . Write down the probability density for  $p(y)$ . You can use the fact that linear transformations of Gaussians are still Gaussian.
- B Let  $x$  be a zero mean gaussian random vector with a isotropic covariance ( $\Sigma = I$ ). Let  $y = Ax + b$ . Compute the mean and variance of  $y$ .
- C (MATLAB) Generate 500 random samples from a 2 dimensional gaussian with an isotropic  $\Sigma$  using the matlab command `randn`. Transform the data as above with  $b = \begin{bmatrix} .5 \\ 1 \end{bmatrix}$ , and  $A = \begin{bmatrix} -5 & 5 \\ 1 & 1 \end{bmatrix}$ . Plot the original and transformed points. In practice, this is often how one generates samples from a multivariate Gaussian with a non-diagonal  $\Sigma$ .
- D (Maximum likelihood estimation). Assume that we observe  $m$  iid samples of  $x^{(i)}$  from some Gaussian distribution. We can write the log likelihood of the  $m$  samples as a function of the Gaussian parameters  $\theta = (\mu, \Sigma)$ .

$$l(\theta) = \sum_i \log \frac{1}{(2\pi)^{(n/2)}|\Sigma|^{1/2}} \exp -\frac{1}{2}(x^{(i)} - \mu)^T \Sigma^{-1}(x^{(i)} - \mu) \quad (2)$$

The maximum likelihood estimate (MLE) of parameters  $\theta$  can be found by setting the derivative of (2) to zero. Show that (i) the maximum likelihood estimate of  $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$  and (ii) the maximum likelihood estimate of  $\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$ . Hints: Use the trace rules from HW1. For (ii), take the derivative of  $l(\theta)$  with respect to  $\Sigma^{-1}$  and use the following facts:  $\frac{\partial \log |A|}{\partial A} = (A^{-1})^T$  and  $|A|^{-1} = |A^{-1}|$ .

- E Verify the formulas from (B,C,D) by computing the maximum likelihood estimates of the transformed points from the above MATLAB problem.

**Problem II:** Lagrangian optimization. We are going to write out a formulation for a novelty detection algorithm. Given a set of points  $\{x^i\}$ , our algorithm will compute the smallest possible sphere containing the data.

- A Formulate this problem as a quadratic program optimization problem. What is being minimized and what are the constraints? Note that you may need to write convex constraints, instead of affine or linear ones. In certain conditions (including this case), one can still solve the problem via its dual formulation.
- B Using Lagrangian techniques, write down the dual optimization for this problem.
- C Show how the computed sphere can be used to classify a new point  $x^{new}$  as regular or “novel”.
- D Redo the problem allowing for some of the data to lie outside of the sphere, with the fraction controlled by a parameter  $C$  - you will have to use *slack* variables. What are the “support vectors” for this problem?
- E Show how this algorithm can be kernalized so that the sphere is calculated in a feature space  $\Phi(x)$ , rather than the input space of  $x$ .

**Problem III:** (MATLAB) Cross validation and regularization. In this problem we’ll look at the issue of regularization and model selection using our old friend, linear regression. Recall the objective function for *ridge* regression  $J(\Theta) = \frac{1}{2} \sum_{i=1}^m (\Theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\Theta\|^2$ .

- A Derive the closed form solution to the above equation using a modified form of the normal equations
- B Load in the data hw3train.dat. The first column represents a 1-dimensional feature  $x$ , and the second column is its  $y$  value. For the un-regularized case ( $\lambda = 0$ ), train a polynomial regression model of the form  $h(x) = \sum_{j=0}^{j=d} \theta_j x^j$ . Note that we are raising  $x$  to the  $j^{th}$  power, and that  $x^0 = 1$  represents a constant term. Train this for  $d = 1, 2, 3, 10$ . Plot the curves corresponding to the predictions over the  $x$ -interval from  $[0,1]$ . Compute the squared training error (the value of  $J(\Theta)$ ) for each model. Compute the corresponding squared test error using the data hw3test.dat . Which model has the lowest training error? The lowest test error?
- C For  $d = 8$ , we can *regularize* the solution by introducing a non-zero  $\lambda$ . This restricts the model space (in much the same way reducing  $d$  restricts the model). Try different values of  $\lambda$ : 1e-1, 1e-5, and 1e-15. Which has the lowest training/testing error?
- D Cross validation. In the real world, we would not have access to the test data when selecting the best model. Instead, one can split up the training data into a “train” and “validation” set, and compute the error on the validation set. Leave-one-out validation involves training the model with  $m - 1$  points, computing the error on the  $m^{th}$  point, and averaging this result over the  $m$  possible choices of the left-out point. Repeat the model selection experiment from (2) and the regularization tuning from (3) using

leave-one-out validation. How do they do compared to the “cheating” approach that tunes the model on the test data?

**Problem IV:** (MATLAB) Digit classification with neural nets. Download training and testing data from the MNIST dataset at *digitsTrain.dat* and *digitsTest.dat*. I will provide *hw3skel.m* for visualizing the digits. You will be coding a softmax neural net with 1 hidden layer with a variable number of nodes in the single hidden layer. The input features are 196 dimensional. You will use it to distinguish handwritten 1s, 2s, and 3s, and so you need to train 3 outputs  $out_1$ ,  $out_2$ , and  $out_3$ .

$$J(w) = - \sum_{i=1}^m \sum_{k=1}^3 y_k^{(i)} \log \left( out_k(x^{(i)}) \right) \quad (3)$$

$$out_k(x^{(i)}) = \frac{\exp(a_{out,k})}{\sum_j \exp(a_{out,k})} \quad (4)$$

A We will use the multi-class cross entropy error function defined above. We will train the model with stochastic gradient ascent. To implement the backpropagation equations, let us write the gradient due to a *single*  $(x, y)$  training example as  $\frac{\partial J}{\partial w_{ji}}$ . First, we will need to calculate the derivative of the error with respect to the output-node weights. The following fact will be useful

$$\frac{\partial J}{\partial a_{out,k}} = -(y_k - out_k(x))$$

for the three output nodes  $k = 1, 2, 3$ . Derive the above fact.

B (Reference; not graded) To calculate the derivative of the error with respect to the hidden-node weights, we can write the backprop equations as

$$z_j = g(a_j) \text{ where } g(z) = \frac{1}{1 + e^{-z}} \text{ and } a_j = \sum_i w_{ji} z_i + w_{j0} z_0 \quad (5)$$

$$\frac{\partial J}{\partial w_{ji}} = \delta_j z_i \text{ where } \delta_j = \frac{\partial J(w)}{\partial a_j} \quad (6)$$

$$\delta_j = g(a_j)(1 - g(a_j)) \sum_k \delta_k w_{kj} \quad (7)$$

Note that we have included a bias term in (5) by defining  $z_0 \equiv 1$ .

C Implement stochastic gradient descent and train the model with 3 nodes in the hidden layer. Include a bias for both the hidden and output nodes.

D Train the model with additional nodes in the hidden layer (try at least 5 nodes and 7 nodes) . Pick the best one with cross validation. Since leave-one-out validation is expensive, just split the training data equally

into a single “train” and “val” set. Note that the train set should contain equal examples of 1s, 2s, and 3s, so split the data accordingly. Score the misclassification performance of the selected model on the test data.

E (Not graded). You can visualize the input weights for each hidden node as an image - just as one can visualize the input features as an image in *hw3skel.m*. Visualize the features to see what the neural net has learned.