CS 189 Introduction to Machine Learning
Fall 2017 HW9

This homework is due **Monday, October 30 at 10pm.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

2 Classification Policy

Suppose we have a classification problem with classes labeled $1,\dots,c$ and an additional "doubt" category labeled $c+1$. Let $f:\mathbb{R}^d \to \{1,\dots,c+1\}$ be a decision rule. Define the loss function

$$L(f(x),y) = \begin{cases} 0 & \text{if } f(x) = y \quad f(x) \in \{1,\dots,c\}, \\ \lambda_s & \text{if } f(x) \neq y \quad f(x) \in \{1,\dots,c\}, \\ \lambda_r & \text{if } f(x) = c+1 \end{cases} \tag{1}$$

where $\lambda_s \geq 0$ is the loss incurred for making a misclassification and $\lambda_r \geq 0$ is the loss incurred for choosing doubt. Hence, the risk of classifying a new data point x as class $f(x) \in \{1,2,\dots,c+1\}$ is

$$R(f(x)|x) = \sum_{i=1}^{c} L(f(x),i)\,P(Y=i|x).$$

(a) **Show that the following policy obtains the minimum risk:**

(a) Choose class i if $P(Y=i|x) \geq P(Y=j|x)$ for all j and $P(Y=i|x) \geq 1-\frac{\lambda_r}{\lambda_s}$.

(b) Choose doubt otherwise.

(b) **What happens if $\lambda_r = 0$? What happens if $\lambda_r > \lambda_s$? Explain why this is consistent with what one would expect intuitively.**

3 LDA and CCA

Consider the following random variable $X \in \mathbb{R}^d$, generated using a *mixture of two Gaussians*. Here, the vectors $\mu_1,\mu_2 \in \mathbb{R}^d$ are arbitrary (mean) vectors, and $\Sigma \in \mathbb{R}^{d\times d}$ represents a positive definite (covariance) matrix. For now, we will assume that we know all of these parameters.

Draw a label $L \in \{1,2\}$ such that the label 1 is chosen with probability π_1 (and consequently, label 2 with probability $\pi_2 = 1-\pi_1$), and generate $X = \mathcal{N}(\mu_L,\Sigma)$.

(a) Now given a particular $X \in \mathbb{R}^d$ generated from the above model, we wish to find its label. **Write out the decision rule corresponding to the following estimates of L:**

- MLE
- MAP

Your decision rule should take the form of a threshold: if some function $f(X) > T$, then choose the label 1, otherwise choose the label 2. **When are these two decision rules the same?**

(b) You should have noticed that the function f is *linear* in its argument X, and takes the form $w^\top(X-v)$. We will now show that CCA defined on a suitable set of random variables leads to precisely the same decision rule.

Let $Y \in \mathbb{R}^2$ be a one hot vector denoting the realization of the label ℓ, i.e. $Y_\ell = 1$ if $L = \ell$, and zero otherwise. Let $\pi_1 = \pi_2 = 1/2$. **Compute the covariance matrices Σ_{XX}, Σ_{XY} and Σ_{YY} as a function of μ_1, μ_2, Σ.** Recall that the random variables are not zero-mean.

(c) Let us now perform CCA on the two random variables. Recall that in order to find the first canonical directions, we look for vectors $u \in \mathbb{R}^d$ and $v \in \mathbb{R}^2$ such that $\rho(u^\top X, v^\top Y)$ is maximized.

Show that the maximizing u^* is proportional to $\Sigma^{-1}(\mu_1 - \mu_2)$. Recall that u^* is that "direction" of X that contributes most to predicting Y. **What is the relationship between u^* and the function $f(X)$ computed in part (a)?**

Hint: The Sherman-Morrison formula for matrix inversion may be useful:

Suppose $A \in \mathbb{R}^{d\times d}$ is an invertible square matrix and $a, b \in \mathbb{R}^d$ are column vectors. Then,

$$(A + ab^T)^{-1} = A^{-1} - \frac{A^{-1}ab^TA^{-1}}{1 + b^TA^{-1}a}.$$

(d) **Repeat the above part for arbitrary probabilities π_1, π_2.**

(e) Now assume you don't know the parameters μ_1, μ_2, Σ, but are given samples of training data $x_1, x_2, \ldots, x_n$ drawn from the above distribution. You are then given a new X_{test}, which you wish to classify. Using your intuition from CCA, **write down a procedure that predicts the label of X_{test}.**

4 Sensors, Objects, and Localization (Part 2)

Let us say there are n objects and m sensors located in a $2d$ plane. The n objects are located at the points $(x_1, y_1), \ldots, (x_n, y_n)$. The m sensors are located at the points $(a_1, b_1), \ldots, (a_m, b_m)$. We have measurements for the distances between the objects and the sensors: D_{ij} is the measured distance from object i to sensor j. The distance measurement has noise in it. Specifically, we model

$$D_{ij} = ||(x_i, y_i) - (a_j, b_j)|| + Z_{ij},$$

where $Z_{ij} \sim N(0, 1)$. The noise is independent across different measurements.

Starter code has been provided for data generation to aid your explorations. All Python libraries are permitted, though you must use your own implementation of a Neural Network if you want to get bonus points for implementing stochastic gradient descent. For others parts, you are free to use commercial libraries. You will need to plot ten figures in total in this problem.

Assume we observe $D_{ij} = d_{ij}$ with $(X_i, Y_i) = (x_i, y_i)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Here, $m = 7$. **Our goal is to predict $(X_{i'}, Y_{i'})$ from newly observed $D_{i'1}, \ldots, D_{i'7}$.** For a data set with n test data, the error is measured by the average distance between the predicted object locations and the true object locations,

$$\frac{1}{n}\sum_{i=1}^{n} \sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2},$$

where $(\hat{x}_i, \hat{y}_i)$ is the location of objects predicted by a model.

We are going to consider five models in this problem and compare their performance:

- In an earlier assignment, we first estimated sensor locations and then used the estimated sensor locations to estimate the new object locations. This is called a *Generative Model*.
- A *Linear Model*. Using the training set, the linear model attempts to learn (X_i, Y_i) from $(D_{i1}, \dots, D_{i7})$. Then it predicts $(X_{i'}, Y_{i'})$ from $(D_{i'1}, \dots, D_{i'7})$.
- A *Second-Order Polynomial Regression Model*. The set-up is similar to the linear model, but including second-order polynomial features.
- A *Third-Order Polynomial Regression Model*. The set-up is similar to the linear model, but including third-order polynomial features.
- A *Neural Network Model*. The Neural Network should have two hidden layers, each with 100 neurons, and use `Relu` as the non-linearity. (You are encouraged to explore on your own beyond this however. These parameters were chosen to teach you a hype-deflating lesson.)

(a) **Implement the five model listed above. Submit your code on gradescope and include it in your write-up.**

(b) **Fix a set of 7 sensors and generate the following data sets:**

- 15 training sets where n_{train} varies from 10 to 290 in increments of 20.
- A "regular" testing data set where $n_{\text{test}} = 1000$.
- A "shifted" testing data set where $n_{\text{test}} = 1000$. You can do this by setting original_dist to `False` in the function `generate_data` in `starter.py`.

The difference between the "regular" testing data and the "shifted" testing data is that the "regular" testing data is drawn from the same distribution as the training data, whereas the "shifted" testing data is farther away. **Train each of the five models on each of the fifteen training sets. Use your results to generate three figures. Each figure should include *all* of the models on the same plot so that you can compare them**:

- A plot of *training error* versus n_{train} (the amount of data used to train the model) for all of the models.
- A plot of *testing error* on the "regular" test set versus n_{train} (the amount of data used to train the model) for all of the models.
- A plot of *testing error* on the "shifted" test set versus n_{train} (the amount of data used to train the model) for all of the models.

Briefly describe your observations. What are the strengths and weaknesses of each model?

(c) We are now going to do some hyper-parameter tuning on our neural network. Fix the number of hidden layers to be two and let ℓ be the number of neurons in each of these two layers. Try values for ℓ between 100 and 500 in increments of 50. Use data sets with $n_{\text{train}}200$ and $n_{\text{test}} = 1,000$. **What is the best choice for ℓ (the number of neurons in the hidden layers)? Justify your answer with plots.**

(d) We are going to do some more hyper-parameter tuning on our neural network. Let k be the number of hidden layers and let ℓ be the number of neurons in each hidden layer. **Write a formula for the total number of weights in our network in terms of ℓ and k. If we want to keep the total number of weights in the network approximately equal to 10000, find a formula for ℓ in terms of k.** Try values of k between 1 and 4 with the appropriate implied choice for ℓ. Use data sets with $n_{\text{train}} = n_{\text{test}} = 200$. **What is the best choice for k (the number of layers)? Justify your answer with plots.**

(e) **Repeat part ((b)), but for the neural network you should use the hyper-parameters you found through hyper-parameter tuning.**

(f) (Bonus) Instead of training your neural network model using gradient descent, use batch stochastic gradient descent. **How does the neural network trained with SGD compare to the neural network trained with gradient descent? Justify your answer with plots.** (For this part, you must use your own implementation.)

(g) (Bonus) You might have seen that the neural network performance was dissapointing. Explore increasing the training data as well as playing with the hyper-parameters on your own to improve neural network performance for samples drawn from the "regular" data set. Can you get anything to generalize to the "shifted" test data? **Report your parameters and present your evidence.**

5 Tensorflow Installation

Install Tensorflow and run the tutorial.

`https://www.tensorflow.org/get_started/mnist/beginners`

By now, hopefully you have a reasonably good understanding of how Backpropagation and SGD work to train Neural Networks. Implementing these on your own in a good learning experience, but when it comes time for practical applications you are probably going to use existing packages. Starting with the next homework, you will no longer be required to use your own implementation of Neural Networks. Instead, you may use Tensorflow, where SGD and Backpropagation are implemented for you. This week, you should get a head start by installing Tensorflow.

Submit your code (it is fine to copy/paste the tutorial, as long as you get it running) and the exact value you get for the training error on MNIST (it should be approximately 92%).

6 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.