CS 189 Introduction to Machine Learning
Spring 2018 HW9

1 Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to assignment on Gradescope, "HW9 Write-Up". If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results, "HW9 Code".
3. If there is a test set, submit your test set evaluation results, "HW9 Test Set".

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

(a) Who else did you work with on this homework? 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. We just want to make it *extra* clear so that no one inadverdently cheats.

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 $1, \ldots, c$ and an additional "doubt" category labeled $c+1$. Let $f : \mathbb{R}^d \to \{1, \ldots, c+1\}$ be a decision rule. Define the loss function

$$L(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } f(\mathbf{x}) = y \quad f(\mathbf{x}) \in \{1, \ldots, c\}, \\ \lambda_c & \text{if } f(\mathbf{x}) \neq y \quad f(\mathbf{x}) \in \{1, \ldots, c\}, \\ \lambda_d & \text{if } f(\mathbf{x}) = c+1 \end{cases} \tag{1}$$

where $\lambda_c \geq 0$ is the loss incurred for making a misclassification and $\lambda_d \geq 0$ is the loss incurred for choosing doubt. In words this means the following:

- When you are correct, you should incur no loss.
- When you are incorrect, you should incur some penalty λ_c for making the wrong choice.
- When you are unsure about what to choose, you might want to select a category corresponding to "doubt" and you should incur a penalty λ_d.

We can see that in practice we'd like to have this sort of loss function if we don't want to make a decision if we are unsure about it. This sort of loss function, however, doesn't help you in instances where you have high certainty about a decision, but that decision is wrong.

To understand the expected amount of loss we will incur with decision rule $f(\mathbf{x})$, we look at the risk. The risk of classifying a new data point $\mathbf{x}$ as class $f(\mathbf{x}) \in \{1, 2, \ldots, c+1\}$ is

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

(a) **Show that the following policy $f_{opt}(x)$ obtains the minimum risk:**

- Find class i such that $P(Y = i|\mathbf{x}) \geq P(Y = j|\mathbf{x})$ for all j, meaning you pick the class with the highest probability given x.
- Choose class i if $P(Y = i|\mathbf{x}) \geq 1 - \frac{\lambda_d}{\lambda_c}$
- Choose doubt otherwise.

(b) **How would you modify your optimum decision rule if $\lambda_d = 0$? What happens if $\lambda_d > \lambda_c$? Explain why this is or is not consistent with what one would expect intuitively.**

3 LDA and CCA

Consider the following random variable $\mathbf{X} \in \mathbb{R}^d$, generated using a *mixture of two Gaussians*. Here, the vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^d$ are arbitrary (mean) vectors, and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ 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 $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_L, \boldsymbol{\Sigma})$.

(a) Now given a particular $\mathbf{X} \in \mathbb{R}^d$ 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 $f(\mathbf{X}) > T$, then choose the label 1, otherwise choose the label 2. **When are these two decision rules the same?** Hint: investigate the ratio between the two likelihood functions and the ratio between the two posterior probabilities respectively.

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

Let $\mathbf{Y} \in \mathbb{R}^2$ 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 $\mathbf{\Sigma}_{XX}, \mathbf{\Sigma}_{XY}$ and $\mathbf{\Sigma}_{YY}$ as a function of $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mathbf{\Sigma}$.** Recall that the random variables are not zero-mean. Hint: when computing the covariance matrices, the tower property of the expectation is useful.

(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 $\mathbf{u} \in \mathbb{R}^d$ and $\mathbf{v} \in \mathbb{R}^2$ such that $\rho(\mathbf{u}^\top\mathbf{X}, \mathbf{v}^\top\mathbf{Y})$ is maximized.

Show that the maximizing $\mathbf{u}^*$ is proportional to $\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$. Recall that $\mathbf{u}^*$ is that "direction" of $\mathbf{X}$ that contributes most to predicting $\mathbf{Y}$. **What is the relationship between $\mathbf{u}^*$ and the function $f(\mathbf{X})$ computed in part (a)?**

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

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

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

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.

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

q points, the error is measured by the average distance between the predicted object locations and the true object locations,

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

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:

- A *Generative Model*: This is basically from the earlier assignment: we first estimate sensor locations from the training data by solving the nonlinear learst squares problem using Gauss-Newton algorithm [1], and then use the estimated sensor locations to estimate the new object locations.
- A *Oracle Model*: This is the same as generative model except that we will use the ground truth sensor location rather than the estimated sensor location.
- A *Linear Model*. Using the training set, the linear model attempts to fit (X_i, Y_i) directly from the distance measurements $(D_{i1}, \ldots, D_{i7})$. Then it predicts $(X_{i'}, Y_{i'})$ from $(D_{i'1}, \ldots, D_{i'7})$ using the map that it found during training. (It never tries to explicitly model the underlying sensor locations.)
- 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.) The neural net approach also follows the principle of finding/learning a direct connection between the distance measurements and the object location.

(a) **Implement the last four models listed above in `models.py`.** Starter code has been provided for data generation and visualization to aid your explorations. We provide you a simple gradient descent framework for you to implement the neural network, but you are also free to use the TensorFlow and PyTorch code from your previous homework.

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.

(b) In the following parts, we will deal with two type of data, the "regular" data set, and the "shifted" data set. The "regular" data set has the same distribution as the training data set,

[1] This is not covered in the class but you can find the related information in `https://www.wikiwand.com/en/Gauss-Newton_algorithm`

while the "shifted" data set has a different distribution. **Run `plot0.py` to visualize** the sensor location, the distribution of the "regular" data set, and the distribution of the "shifted" data set. **Attach the plot**.

(c) The starter code generated a set of 7 sensors and 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`.

Use `plot1.py` to 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?

(d) 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.** The starter code is in `plot2.py`.

(e) 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 $n_{\text{train}} = 200$ and $n_{\text{test}} = 1000$. **What is the best choice for k (the number of layers)? Justify your answer with plots.** The starter code is in `plot3.py`.

(f) You might have seen that the neural network performance is disappointing compared to the generative model in the "shifted" data. Try increasing the number of training data and tune the hyper-parameters. Can you get it to generalize to the "shifted" test data? **Attach the "number of training data vs accuracy" plot to justify your conclusion.** What is the intuition how neural network works on predicting the D? The starter kit is provided in `plot4.py`.

5 Entropy, KL Divergences, and Cross-Entropy

Stepping back for a bit, notice that so far we have mostly considered so called *Euclidean* spaces, the most prominent example is the vector space $\mathbb{R}^d$ with inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y}$ and norm $\|\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{x}$. For example in linear regression, we had the loss function

$$f(\boldsymbol{\theta}) = \|\mathbf{X}^\top \boldsymbol{\theta} - \mathbf{y}\|_2^2 = \sum_{i=1}^{n} (\mathbf{x}_i{}^\top \boldsymbol{\theta} - y_i)^2$$

which uses precisely the Euclidean norm squared to measure the error.

In this problem we will implicitly consider a different geometric structure, which defines a metric on probability distributions. In more advanced courses, this can be developed further to show how probability distributions are naturally imbued with a curved non-Euclidean intrinsic geometry. Here, our goals are more modest — we just want you to better understand the relationship between probability distributions, entropy, KL Divergence, and cross-entropy.

Let $\mathbf{p}$ and $\mathbf{q}$ be two probability distributions, i.e. $p_i \geq 0$, $q_i \geq 0$, $\sum_i p_i = 1$ and $\sum_i q_i = 1$, then we define the Kullback-Leibler divergence

$$\mathrm{KL}(\mathbf{p}, \mathbf{q}) = \sum_i p_i \log \frac{p_i}{q_i}$$

which is the "distance" to $\mathbf{p}$ from $\mathbf{q}$. We have $\mathrm{KL}(\mathbf{p}, \mathbf{p}) = 0$ and $\mathrm{KL}(\mathbf{p}, \mathbf{q}) \geq 0$ (the latter by Jensen's inequality) as would be expected from a distance metric. However, $\mathrm{KL}(\mathbf{p}, \mathbf{q}) \neq \mathrm{KL}(\mathbf{q}, \mathbf{p})$ since the KL divergence is not symmetric.

(a) *Entropy motivation:* Let $X_1, X_2, \ldots, X_n$ be independent identically distributed random variables taking values in a finite set $\{0, 1, \ldots, m\}$, i.e. $p_j = P(X_i = j)$ for $j \in \{0, 1, \ldots, m\}$. The *empirical number of occurrences* is then a random vector that we can denote $\mathbf{F}^{(n)}$ where $F_j^{(n)}$ is the number of variables X_i that happen to take a value equal to j.

Intuitively, we can consider coin tosses with $j = 0$ corresponding to heads and $j = 1$ corresponding to tails. Say we do an experiment with $n = 100$ coin tosses, then $F_0^{(100)}$ is the number of heads that came up and $F_1^{(100)}$ is the number of tails.

Recall that the number of configurations of $X_1, X_2, \ldots, X_n$ that have $f^{(n)}$ as their empirical type is $\binom{n}{f_0^{(n)}, f_1^{(n)}, \ldots, f_m^{(n)}}$. Further notice that dividing the empirical type by n yeilds an empirical probability distribution.

Show using the crudest form of Stirling's approximation ($\ell! \approx (\frac{\ell}{e})^\ell$) that this is approximately equal to $\exp\Big(nH(f^{(n)}/n)\Big)$ where the entropy H of a probability distribution is defined as $H(\mathbf{p}) = \sum_{j=0}^{m} p_j \ln \frac{1}{p_j}$.

(b) *KL divergence motivation:* Recall that the probability of seeing a particular empirical type is given by:

$$P(\mathbf{F}^{(n)} = \mathbf{f}^{(n)}) = \binom{n}{f_0^{(n)}, f_1^{(n)}, \ldots, f_m^{(n)}} \prod_{j=1}^{m} p_j^{f_j^{(n)}}.$$

Consider the limit of large n and a sequence of empirical types so that $\frac{1}{n}\mathbf{f}^{(n)} \to \mathbf{f}$ for $n \to \infty$, where $\mathbf{f}$ is some distribution of interest.

Use Stirling's approximation to show that

$$\lim_{n\to\infty} \frac{1}{n} \log P(\mathbf{F}^{(n)} = \mathbf{f}^{(n)}) = -\operatorname{KL}(\mathbf{f}, \mathbf{p})$$

Intuitively this means that the larger $\operatorname{KL}(\mathbf{f}, \mathbf{p})$ is, the easier it is to conclude $\mathbf{f} \neq \mathbf{p}$ from empirical data since the chance that we would get confused in that way is decaying exponentially. Note also that the empirical distribution is the first argument of the KL divergence and the true model is the second argument of the KL divergence — we are going from the true model to the empirical one.

(c) **Show that for probability distributions $p(\mathbf{x}, y)$ and $q_\theta(\mathbf{x}, y) = q_\theta(y \mid \mathbf{x})q(\mathbf{x})$ with $\mathbf{x}$ from some discrete set $\mathcal{X}$ and y from some discrete set $\mathcal{Y}$ we have**

$$\operatorname{KL}(p, q_\theta) = c - \sum_{\mathbf{x}\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(\mathbf{x}, y) \log q_\theta(y \mid \mathbf{x}) \tag{2}$$

for some constant c independent of θ.

(d) In logistic regression we predict labels $y_i = +1$ or $y_i = -1$ from features $\mathbf{x}_i$ using the transition probability model

$$q_\theta(y_i \mid \mathbf{x}_i) = \frac{1}{1 + e^{-y_i \boldsymbol{\theta}^\top \mathbf{x}_i}}. \tag{3}$$

We now show that the cross-entropy loss you have seen in lectures can be formulated as minimizing the KL distance to the empirical probabilities from the probabilities induced by the model q_θ.

For convenience, we assume that all the feature $\mathbf{x}_i$ are distinct — no two training points are identical.

Use (c) to show that with the empirical distribution

$$p(\mathbf{x}, y) = \begin{cases} \frac{1}{n} & \text{if } \mathbf{x} = \mathbf{x}_i \text{ and } y = y_i \text{ for some } i = 1, 2, \ldots, n \\ 0 & \text{otherwise} \end{cases}$$

we get

$$\min_{\boldsymbol{\theta}} \operatorname{KL}(p, q_{\boldsymbol{\theta}}) = \min_{\boldsymbol{\theta}} -\frac{1}{n} \sum_i \log q_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i),$$

which is the cross entropy loss derived in lectures.

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 the 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.