CPSC 440/550 Machine Learning (Jan-Apr 2024)
Assignment 1 – due Friday January 19th at **5pm**

IMPORTANT!!!!! Please carefully read the submission instructions that will be posted on Piazza. We will deduct up to 50% on assignments that do not follow the instructions.

Most of the questions below are related to topics covered in CPSC 340, or other courses listed on the prerequisite form. There are several notes available on the webpage which can help with some relevant background.

If you find this assignment to be overly difficult, that is an early warning sign that you may not be prepared to take CPSC 440 at this time. Future assignments may be longer and more difficult than this one.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

Cite any sources outside of course materials (including people) that you refer to, as discussed on the syllabus slides. Do not use ChatGPT, Copilot, or other AI assistance tools. (Something like Grammarly is okay, but not tools that suggest "content.") Even if you've written code for some of these questions before (e.g. in 340), **do it again**.

1 Matrix Notation, Quadratics, Convexity, and Gradients [20 points]

Some of the notes on the course page might be useful to refresh some mathematical tools used in CPSC 340; in particular, your instance of 340 may not have covered positive semi-definite matrices, so check those notes out if needed. Each part is worth [2 points].

[1.1] Consider the function

$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j + \sum_{i=1}^{n} b_i x_i + c,$$

where x is a vector of length n with elements x_i, b is a vector of length n with elements b_i, and A is an $n \times n$ matrix with elements a_{ij} (not necessarily symmetric). Write this function in matrix notation so that it uses A and b, and does not have summations or references to indices i.

Answer: TODO

[1.2] Write the gradient of f from the previous question, in matrix notation.

Answer: TODO

[1.3] Show that f is convex if A is a symmetric, positive semi-definite matrix.

Answer: TODO

[1.4] When A is *strictly* positive definite, give a linear system whose solution minimizes f in terms of x.

Answer: TODO

[1.5] Suppose that A is only positive *semi*-definite. Will a "good" optimization algorithm, e.g. gradient descent with an appropriate step size schedule, necessarily find one of the solutions to your linear system from the last part? If not, where else could it go? *Hint: An example of a matrix that is psd but not strictly pd is $A = 0$.*

Answer: TODO

[1.6] Consider weighted linear regression with an L2 regularizer, with regularization strength $1/\sigma^2$,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v^{(i)}(y^{(i)} - w^\mathsf{T} x^{(i)})^2 + \frac{1}{2\sigma^2}\sum_{j=1}^{d} w_j^2,$$

where $\sigma > 0$ is finite and we have a vector v of length n containing the elements v^i. As usual, w is the weight vector, $x^{(i)}$ the ith input point, and $y^{(i)} \in \mathbb{R}$ its label. Write this function in matrix notation.

Note: you can use $\mathbf{X} \in \mathbb{R}^{n\times d}$ as the matrix with rows $(x^{(i)})^\mathsf{T}$, y as the vector containing the elements $y^{(i)}$, and V as a diagonal matrix containing the vector v along the diagonal.

Answer: TODO

[1.7] Assuming that $v^i \geq 0$ for all i, show that f from the previous part is convex. *(You can use Question [1.3] or not, as you prefer.)*

Answer: TODO

[1.8] Assuming that we have $v^i \geq 0$ for all i, give a linear system whose solution minimizes f in terms of w. *(Again, use the previous results or not, your choice.)*

Answer: TODO

[1.9] If we fit a linear classifier with the exponential loss (used in older boosting algorithms, among other places), it would take the form

$$f(w) = \sum_{i=1}^{n} \exp(-y^{(i)} w^\mathsf{T} x^{(i)}).$$

Derive the gradient of this loss function.

Answer: TODO

[1.10] The support vector regression objective function is

$$f(w) = \sum_{i=1}^{n} \max\left\{0, \left|w^\mathsf{T} x^{(i)} - y^{(i)}\right| - \epsilon\right\} + \frac{\lambda}{2}\left\|w\right\|^2$$

where ϵ is a non-negative hyper-parameter. It is similar to the L1 robust regression loss, but "doesn't care" if your prediction is less than ϵ from the target (which can reduce overfitting). Show that this non-differentiable function is convex.

Answer: TODO

2 Machine Learning Model Memory and Time Complexities [18 points]

Answer the following questions using big-O notation, and a brief explanation. Your answers may involve n, d, and perhaps additional quantities defined in the question. As an example, the (linear) least squares model has $\mathcal{O}(d)$ parameters, requires $\mathcal{O}(d)$ time for prediction, and requires $\mathcal{O}(nd^2 + d^3)$ time to train.[1]

Each part is worth [2 points].

[2.1] What is the training time for least squares (linear regression by solving the normal equations) with L2 regularization?

Answer: TODO

[2.2] What is the prediction cost for a trained linear model on t test examples?

Answer: TODO

[2.3] What is the storage space required to make predictions with a linear regression model using Gaussian RBF features (with lengthscale, aka bandwidth, σ)?

Answer: TODO

[2.4] What is the prediction time for linear regression with Gaussian RBFs (with lengthscale, aka bandwidth, σ) as features on t test examples?

Answer: TODO

[2.5] What is the cost of evaluating the support vector regression objective function from Question [1.10]?

Answer: TODO

[2.6] What is the cost of trying to minimize the exponential loss from Question [1.9] by running t iterations of gradient descent?

Answer: TODO

[2.7] What is the cost of trying to minimize the exponential loss by running t iterations of stochastic gradient descent?

Answer: TODO

[2.8] What is the storage space required for the k-means clustering algorithm?

Answer: TODO

[2.9] What is the cost of clustering t examples using an already-trained k-means model?

Answer: TODO

Many people got very low grades on variants of this question from past years. If you're not sure how to answer questions like this, get help!

[1] In this course, we assume matrix operations have the "textbook" cost where the operations are implemented in a straightforward way, e.g. with `for` loops. For example, we'll assume that multiplying two $n \times n$ matrices or computing a matrix inverse simply costs $\mathcal{O}(n^3)$, rather than the $\mathcal{O}(n^\omega)$ where ω is more like 2.37, which while true is only relevant for extremely huge matrices; $\mathcal{O}(n^3)$ is closer to the behaviour observed for actual practical matrix sizes.

3 MAP Estimation [12 points]

In 340, we showed that under the assumptions of a Gaussian likelihood and Gaussian prior,

$$y^{(i)} \sim \mathcal{N}(w^\top x^{(i)}, 1), \quad w_j \sim \mathcal{N}\left(0, \frac{1}{\lambda}\right),$$

the MAP estimate is equivalent to solving the L2-regularized least squares problem

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2,$$

in the "loss plus regularizer" framework. For each of the alternate assumptions below, write it in the "loss plus regularizer" framework (simplifying as much as possible):

[3.1] [4 points] Gaussian likelihood with a separate variance $\nu^{(i)} > 0$ for each training example, and Laplace prior with a separate scale $1/\lambda_j > 0$ for each variable,

$$y^{(i)} \sim \mathcal{N}(w^\mathsf{T} x^{(i)}, \nu^{(i)}), \quad w_j \sim \mathcal{L}\left(0, \frac{1}{\lambda_j}\right)$$

where a Laplace distribution $\mathcal{L}(\mu, b)$ has density $\frac{1}{2b}\exp\left(-\left|x-\mu\right|/b\right)$.

Answer: TODO

[3.2] [4 points] Robust student-t likelihood and a uniform prior for w,

$$p(y^{(i)}|x^{(i)}, w) = \frac{1}{\sqrt{\nu}B\left(\frac{1}{2}, \frac{\nu}{2}\right)}\left(1 + \frac{(w^\mathsf{T} x^{(i)} - y^{i})^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \quad w \sim \text{Uniform}\left(\{w : \|w\| \leq C\}\right),$$

where u is $d \times 1$, B is the "Beta" function, and the parameter ν is called the "degrees of freedom."[2] You can use V_d as the volume of the d-dimensional unit ball $\{w : \|w\| \leq 1\}$. *Hint: Think carefully about what a uniform prior density means for the posterior density of w.*

Answer: TODO

[3.3] [4 points] If $y^{(i)}$ represents counts, we can use a Poisson-distributed likelihood, and take a Gaussian prior on w centered at u:

$$p(y^{(i)}|x^{(i)}, w) = \frac{\exp(y^{(i)} w^\mathsf{T} x^{(i)})\exp(-\exp(w^\top x^{(i)}))}{y^{(i)}!}, \quad w_j \sim \mathcal{N}\left(u_j, \frac{1}{\lambda}\right)$$

Answer: TODO

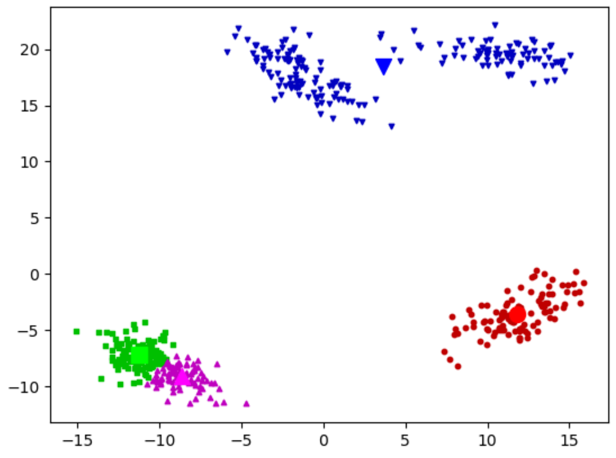
[2]This likelihood is more robust than the Laplace likelihood, but leads to a non-convex objective.

4 K-Means Clustering [15 points]

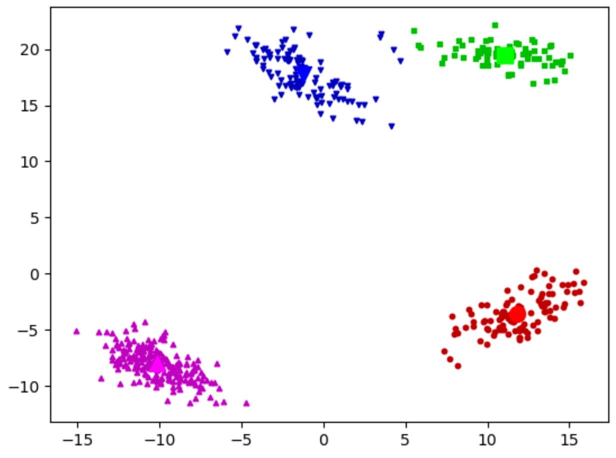
This and following questions use code available from the course webpage; get it from `a1.zip`.

You'll need Python 3 and the packages `numpy`, `scipy`, and `matplotlib`. If you don't have them installed already, install them with `conda install`, your system package manager, or `pip install numpy scipy matplotlib`.

If you run `main.py kmeans`, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:



(Note that the colours are arbitrary due to the label switching problem.) But the "correct" clustering (that was used to make the data) is something more like this:



If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples $x^{(i)}$ and their means $w_{y^{(i)}}$,

$$f(w^{(1)}, w^{(2)}, \ldots, w^{(k)}, y^{(1)}, y^{(2)}, \ldots, y^{(n)}) = \sum_{i=1}^{n} \left\| x^{(i)} - w^{(y^{(i)})} \right\|_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} \left(x_j^{(i)} - w_j^{(y^{(i)})} \right)^2 .$$

where $y^{(i)} \in \arg\min_{c\in[n]} \left\|x^{(i)} - w^{(c)}\right\|$ is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the $w^{(c)}$ and the $y^{(i)}$ values.

[4.1] [3 points] Complete the function `KMeans.loss`, inside `k_means.py`, that takes in a data matrix $\mathbf{X}$, a vector of corresponding cluster assignments y, and a matrix of cluster means W, and computes this objective function. Hand in your code.

Answer: TODO

[4.2] [1 points] Modify your code to, instead of/in addition to printing the number of labels that change on each iteration, print the value of `KMeans.loss` after each iteration. What trend do you observe? (No need to hand in code or specific loss values for this, just describe the trend.)

Answer: TODO

[4.3] [2 points] `main.py kmeans-best` will call the function `q_kmeans_best()` in `main.py`, which calls the `best_fit` function to rerun k-means 50 times and take the one with the lowest error. Complete the `best_fit` function, and hand in your code and the resulting plot.

Answer: TODO

[4.4] [3 points] Explain why the `KMeans.loss` function should not be used to choose k – even if you evaluate it on test data.

Answer: TODO

[4.5] [1 points] The data in `cluster_data_2.npz` is exactly the same as `cluster_data.npz`, except that it has four outliers that are far away from the data. `main.py kmeans-outliers` will run your code from above to find the best of 50 runs on this data and save it as `figs/kmeans-outliers.png`. Hand in this plot; are you satisfied with it?

Answer: TODO

[4.6] [5 points] Implement the *k-medians* algorithm in `kmedians.py`, which assigns examples to the nearest $w^{(c)}$ in the L1-norm, and then updates all the $w^{(c)}$ by setting them to the "median" of the points assigned to the cluster (defining the d-dimensional median as the concatenation of the medians along each dimension). For this algorithm it makes sense to use the L1-norm version of the error (where $y^{(i)}$ now represents the closest centre in the L1-norm),

$$f(w^{(1)}, w^{(2)}, \dots, w^{(k)}, y^{(1)}, y^{(2)}, \dots, y^{(n)}) = \sum_{i=1}^{n} \left\|x^{(i)} - w^{(y^{(i)})}\right\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{d} |x_j^{(i)} - w_j^{(y^{(i)})}|,$$

`main.py kmedians` will find the best of 50 models with $k = 4$ for you, once you've finished the `KMedians` class. Hand in your code and plot. Is this better?

Answer: TODO

5 Regularization and Hyper-Parameter Tuning [20 points]

If you run `main.py lsq`, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Draw a figure showing the training/testing data and what the model looks like.

Unfortunately, this is not a great model of the data, and the figure shows that a linear model with a y intercept of 0 is probably not suitable.

[5.1] [5 points] Finish the `LeastSquaresBias` class, in `least_squares.py`. Using this class should fit and predict with the model

$$y^{(i)} = w^\mathsf{T} x^{(i)} + b,$$

where both w and b are fit to data with least squares; implement it however you like (there are a few reasonable options). `main.py lsq-bias` will then run it for you. Hand in your new class and the updated plot.

Answer: TODO

[5.2] [6 points] Allowing a non-zero y-intercept improves the prediction substantially, but the model still seems sub-optimal because there are obvious non-linear effects. Complete the model `leastSquaresRBFL2` that implements *least squares using Gaussian radial basis functions (RBFs) with L2-regularization*. `main.py lsq-rbf` will then run it for you.

Use `lam` for the L2 regularization parameter[3], and `sigma` for the lengthscale of the Gaussian features. Note that your L2 regularization should correspond to minimizing the loss function $\|\mathbf{X}w - y\|^2 + \lambda \|w\|^2$; some versions instead correspond to putting a $\frac{1}{n}$ in front of the loss term.

Hand in your function and the plot generated with $\lambda = 1$ and $\sigma = 1$.

Hint: The function `utils.euclidean_dist_squared`, which was also used in our k-means implementation, efficiently computes the squared Euclidean distance between all pairs of rows in two matrices.

Answer: TODO

[5.3] [6 points] Modify the script, in the function `lsq-rbf-split`, to split the training data into a "train" and "validation" set (you can use half the examples for training and half for validation), and use these to select λ and σ from some reasonable set of candidate values. You'll probably want to vary these by a few orders of magnitude either smaller or larger from 1.

(Although in real work you'd probably use some pre-coded helpers for this, code it yourself here.)

Hand in your modified function, the selected λ and σ, and the plot you obtain by refitting on the full dataset with the best values of λ and σ.

Answer: TODO

[5.4] [3 points] Consider (but you don't have to implement) a model combining the first two parts of this question,

$$y^{(i)} = w^\mathsf{T} x^{(i)} + b + v^\mathsf{T} z^{(i)},$$

where $z^{(i)}$ are the Gaussian RBFs and v is a vector of regression weights for those features. Suppose that we first fit w and b assuming that $v = 0$ as in part (a), and then we fit v with w and w_0 fixed (you

[3] We can't name it `lambda`, because that's a reserved word in Python. You could technically type λ, but there are a lot of Unicode characters for λ that all look really similar, so it gets annoying. . . .

could use your code for Question [5.3] to do this by modifying the $y^{(i)}$ values). Why would someone want to do this?

Hint: Think about how this model would behave on a test point $\tilde{x} = 10$.

Answer: TODO

6 Optimal matrix multiplication [15 points]

Suppose we have a bunch of matrices $M_1, M_2, \ldots, M_k$, where M_i is of shape $d_{i-1} \times d_i$, and we want to find the $d_0 \times d_k$ matrix $M_1 M_2 \cdots M_k$. There are lots of ways to compute this, since matrix multiplication is associative: for example, if $k = 6$ we could do $M_1(M_2(M_3(M_4(M_5 M_6))))$, $M_1((M_2 M_3)((M_4 M_5) M_6))$, etc. The number of possibilities is exponential in k.

Recall that if A is $m \times n$ and B is $n \times p$, we can compute AB with $\mathcal{O}(mnp)$ scalar operations; this is typically a "good-enough" model for how much this costs in practice for the kinds of problems that appear in ML.

Fill in the `mmul_order` function in `mmul_order.py` to use dynamic programming to find the optimal order of multiplications.

Make sure you follow the comments there as to the input and output format of your function, and that your algorithm is $\mathcal{O}(k^3)$ or better.

`main.py mmul-order` will call the function for you with a few test cases; hand in your code and the cost and order for the final part.

Answer: TODO