

CPSC 440/550 Machine Learning (Jan-Apr 2026)
Assignment 1 – due Friday January 16th at **11:59pm**

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. 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.”)

Question **2** and onward use code available from the course webpage; get it from **a1.zip**. You’ll need Python 3 and the packages **numpy**, **scipy**, and **matplotlib** for the first parts, then also PyTorch for Question **[3.5]** and onward. The CPU version of PyTorch is fine. If you don’t have them installed already, you can get them with **conda install**, **pip install**, or your system package manager; <https://pytorch.org/get-started/locally/> might be useful. Alternatively, you might find it easier to use <https://colab.research.google.com/> or similar cloud resources; just make sure that you don’t lose your work!

1 Matrix Notation, Quadratics, Convexity, and Gradients [30 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 [3 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 symmetric and *strictly* positive definite, give a linear system whose solution minimizes f in terms of x .

Answer: TODO

[1.5] Suppose that A isn't symmetric, but that $A + A^T$ is strictly positive definite. Characterize the minimizers of f (like the previous part), or argue that there might not be a minimizer.

Answer: TODO

[1.6] Suppose that A is symmetric and 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

[1.7] The support vector regression objective function is

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

where $\epsilon \geq 0$ and $\lambda \geq 0$ are hyperparameters. It is similar to the L1 robust regression loss (with L2 regularization), 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

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

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

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 i th input point, and $y^{(i)} \in \mathbb{R}$ its label. Write this function in matrix notation.

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

Answer: **TODO**

- [1.9] 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.)

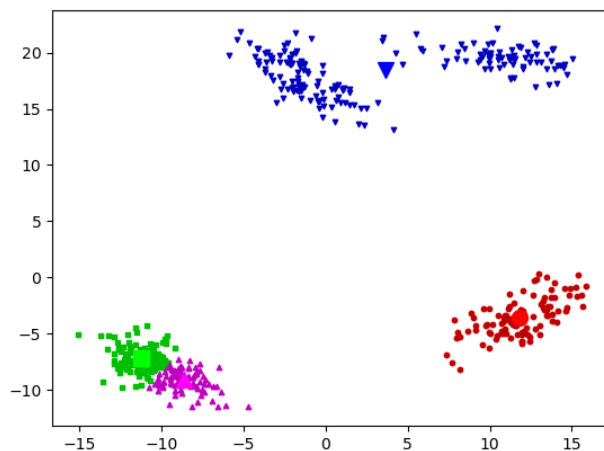
Answer: **TODO**

- [1.10] 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.)

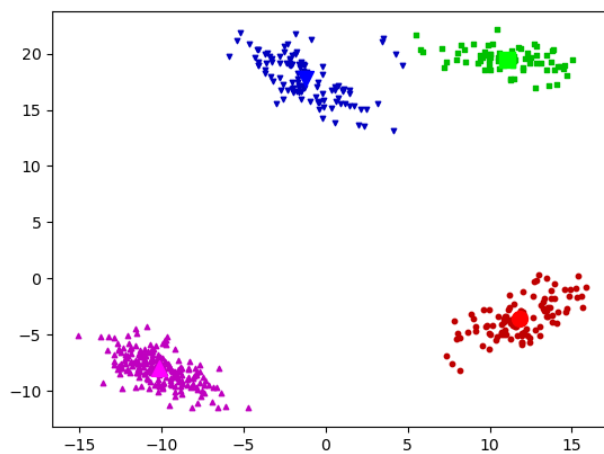
Answer: **TODO**

2 K-Means Clustering [25 points]

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 multiple candidate clusterings for a *fixed* value of k , one strategy is to minimize the sum of squared distances between examples $x^{(i)}$ and their means $w_{y^{(i)}}$,

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

where $y^{(i)} \in \arg \min_{1 \leq c \leq n} \|x^{(i)} - w^{(c)}\|$ 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.

- [2.1] [4 points] Complete the function `KMeans.loss`, inside `kmeans.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**

- [2.2] [3 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**

- [2.3] [4 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**

- [2.4] [5 points] [Explain why](#) the `KMeans.loss` function should not be used to choose k – even if you evaluate it on test data.

Answer: **TODO**

- [2.5] [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**

- [2.6] [8 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). Since the median is the point minimizing the sum of absolute distances to the data, it makes sense here to use the L1-norm version of the error (where $y^{(i)}$ now represents the closest centre in the L1-norm),

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

`main.py kmedians-outliers` 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**

3 Various Kinds of One-Dimensional Regression [45 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.

- [3.1] [5 points] Finish the `LeastSquaresBias` class, in `least_squares.py`. Using this class should fit and predict with the model

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

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**

- [3.2] [10 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*. Put an RBF feature at each data point; you can include an intercept or not, your choice. `main.py lsq-rbf` will then run it for you.

Use `lam` for the L2 regularization parameter¹, and `sigma` for the lengthscale of the Gaussian features. Your L2 regularization should correspond to minimizing the loss function $\|\mathbf{X}w - y\|^2 + \lambda \|w\|^2$, and not the version that puts a $\frac{1}{n}$ and/or a $\frac{1}{2}$ 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**

- [3.3] [10 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 \$\lambda\$ and \$\sigma\$, and the plot you obtain by refitting on the full dataset with the best values of \$\lambda\$ and \$\sigma\$.](#)

Answer: **TODO**

- [3.4] [5 points] Take a look at the `ComboModel` fit by `main.py lsq-combo`. [What advantage might this model have over a plain `LeastSquaresRBFL2`?](#)

Hint: Try thinking about, and maybe plotting using the `plot_lims` parameter to `test_and_plot`, what these models would do for a test point $\hat{x} = -6$.

Answer: **TODO**

¹We 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...

The RBF features here mean that once we've fit a model to n data points, it takes $\mathcal{O}(n)$ time to predict. We can get similar predictions, but potentially much faster, if we don't allow ourselves so many features. While there are many ways to do this, we might as well optimize the location of these features. In the file `torch_net.py`, there's an implementation of a simple one-hidden-layer ReLU network; `main.py relu` will run it for you and plot the results.

- [3.5] [8 points] One-dimensional ReLU networks are “piecewise-linear.” `main.py relu-hinges` has scaffolding code to plot the locations of these “hinges” through optimization, the points where the network changes from one linear function to another. Fill in the code to find the location; hand in the code and a justification of why this is the correct thing to do.

Don't try to access the model as a black box predictor; go into its parameters. `list(thing.parameters())` will get you the parameters of a `nn.Module` or other network layer.

Answer: TODO

- [3.6] [7 points] `main.py relu-combos` will run two different `ComboModels` (as in Question [3.4]) combining a ReLU network with an RBFL2 net. Hand in the generated `combos.png`; the two predicted functions are both pretty good, but have some qualitative differences. Identify at least two or three qualitative differences in the curves, and explain why they make sense.

Answer: TODO