CPSC 340 Assignment 6 (due April 1st at 11:55pm)

We are providing solutions because supervised learning is easier than unsupervised learning, and so we think having solutions available can help you learn. However, the solution file is meant for you alone and we do not give permission to share these solution files with anyone. Both distributing solution files to other people or using solution files provided to you by other people are considered academic misconduct. Please see UBC's policy on this topic if you are not familiar with it:
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,959`
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,960`

Instructions

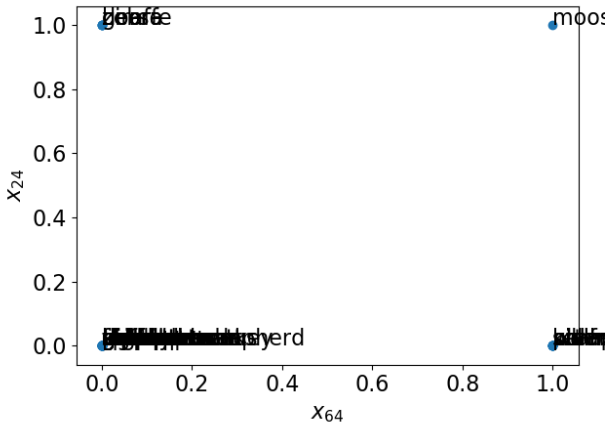
Rubric: {mechanics:5}

IMPORTANT!!! Before proceeding, please carefully read the general homework instructions at `https://www.cs.ubc.ca/~fwood/CS340/homework/`. The above 5 points are for following the submission instructions. You can ignore the words "mechanics", "reasoning", etc.

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

1 Data Visualization

If you run `python main.py -q 1`, it will load the animals dataset and create a scatterplot based on two randomly selected features. We label some points, but because of the binary features the scatterplot shows us almost nothing about the data. One such scatterplot looks like this:

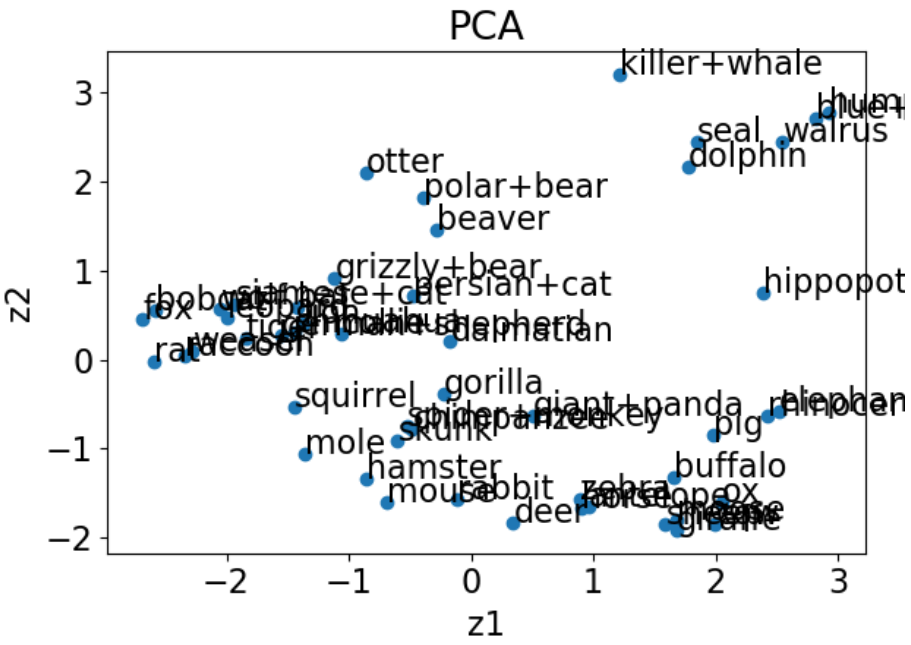


1.1 PCA for visualization

Rubric: {reasoning:2}

Use scikit-learn's PCA to reduce this 85-dimensional dataset down to 2 dimensions, and plot the result. Briefly comment on the results (just say anything that makes sense and indicates that you actually looked at the plot).

Answer: The scatterplot should look roughly like this:



The points that are labeled could vary. (Roughly it looks like as we move from the upper-left to the lower-right we go from terrestrial to aquatic animals, and as we move from the lower-left to the upper-right the animals are getting bigger. But they are definitely exceptions to these rules that the higher-order principal components might capture, and the 'crowding' effect places some very dissimilar animals next to each other.)

1.2 Data Compression

Rubric: {reasoning:2}

1. How much of the variance is explained by our 2-dimensional representation from the previous question?
2. How many PCs are required to explain 50% of the variance in the data?

Answer: Computing the amount of variance explained by the PCs can be done using the last line of the code in the previous answer.

1. With $k = 2$ only about 30.19% of the variance is explained, so our 2D visualization is missing a lot of information.
2. We need $k \geq 5$ to explain 50% of the variance.

Note: you need to centre the columns of X before computing the variance explained with the formula from lecture. You don't need to do this for the previous part, because scikit-learn does it automatically.

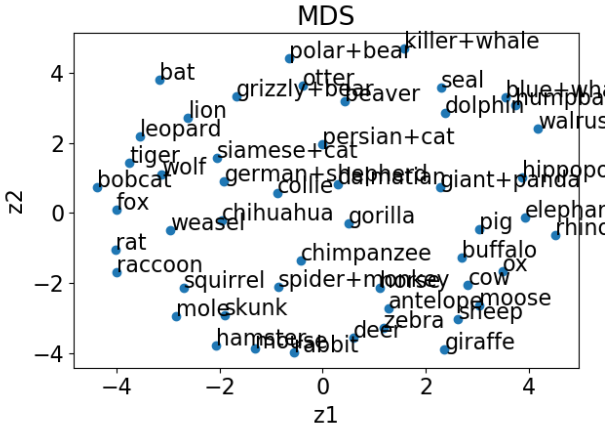
1.3 Multi-Dimensional Scaling

Rubric: {reasoning:1}

If you run `python main.py -q 1.3`, the code will load the animals dataset and then apply gradient descent to minimize the following multi-dimensional scaling (MDS) objective (starting from the PCA solution):

$$f(Z) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n}(\|z_i - z_j\| - \|x_i - x_j\|)^2. \tag{1}$$

The result of applying MDS is shown below.



Although this visualization isn't perfect (with "gorilla" being placed close to the dogs and "otter" being placed close to two types of bears), this visualization does organize the animals in a mostly-logical way. Compare the MDS objective for the MDS solution vs. the PCA solution; is it indeed lower for the MDS solution?

Answer: The MDS objective is around 1777 at the MDS solution and 4292 at the PCA solution; yes, it's lower as expected.

1.4 ISOMAP

Rubric: {code:10}

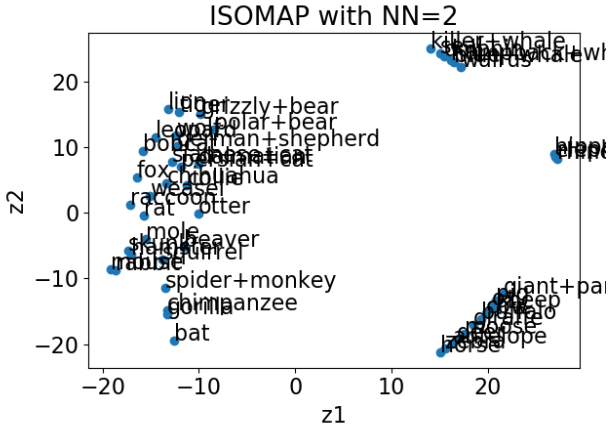
Euclidean distances between very different animals are unlikely to be particularly meaningful. However, since related animals tend to share similar traits we might expect the animals to live on a low-dimensional manifold. This suggests that ISOMAP may give a better visualization. Fill in the class *ISOMAP* so that it computes the approximate geodesic distance (shortest path through a graph where the edges are only between nodes that are k-nearest neighbours) between each pair of points, and then fits a standard MDS model (1) using gradient descent. Plot the results using 2 and using 3-nearest neighbours.

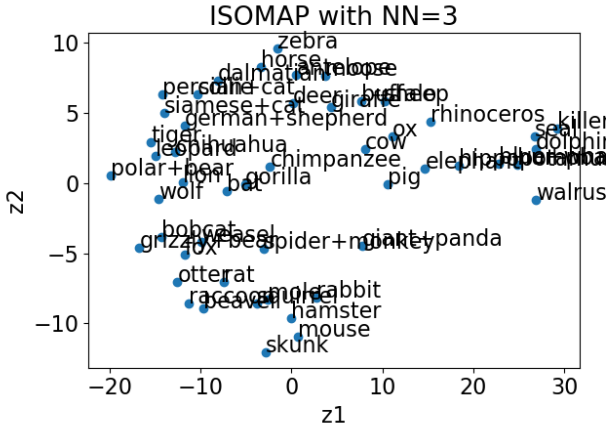
Note: when we say 2 nearest neighbours, we mean the two closest neigbours excluding the point itself. This is the opposite convention from what we used in KNN at the start of the course.

The function *utils.dijskstra* can be used to compute the shortest (weighted) distance between two points in a weighted graph. This function requires an $n \times n$ matrix giving the weights on each edge (use 0 as the weight for absent edges). Note that ISOMAP uses an undirected graph, while the k-nearest neighbour graph might be asymmetric. One of the usual heuristics to turn this into a undirected graph is to include an edge i to j if

i is a KNN of j or if j is a KNN of i. (Another possibility is to include an edge only if i and j are mutually KNNs.)

Answer: We can implement ISOMAP by changing the distance function. See *manifold.ISOMAP*. This assumes that we add an edge if i is a KNN of j or vice versa, but other distance functions in the spirit of constructing a KNN graph are also acceptable. The result of using this weighting, for 2 and 3 neighbours respectively, is:





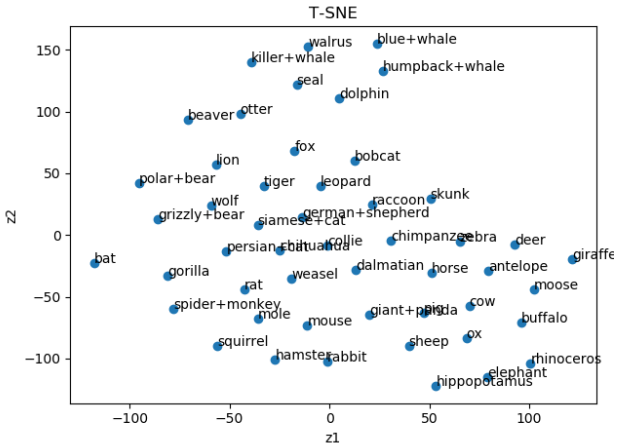
1.5 t-SNE

Rubric: {reasoning:1}

Try running scikit-learn's t-SNE on this dataset as well. Submit the plot from running t-SNE. Then, briefly comment on PCA vs. MDS vs. ISOMAP vs. t-SNE for dimensionality reduction on this particular data set. In your opinion, which method did the best job and why?

Note: There is no single correct answer here! Also, please do not write more than 3 sentences.

Answer: This is a matter of opinion and the goal of the question is to make sure you at least looked at the plots. I like ISOMAP with 2 neighbours because it clearly separates some sensible groups. One question though is that giant panda is away from the rest of the bears. t-SNE figure is below:



1.6 Sensitivity to Initialization

Rubric: {reasoning:2}

For each of the four methods (PCA, MDS, ISOMAP, t-SNE) tried above, which ones give different results when you re-run the code? Does this match up with what we discussed in lectures, about which methods are sensitive to initialization and which ones aren't? Briefly discuss.

Answer: Only t-SNE gives different results each time. PCA is not sensitive to initialization. MDS and ISOMAP are, but we're not initializing them randomly, we're using the (deterministic) PCA solution. scikit-learn is presumably using a random initialization for t-SNE.

2 Neural Networks

2.1 Neural Networks by Hand

Rubric: {reasoning:5}

Suppose that we train a neural network with sigmoid activations and one hidden layer and obtain the following parameters (assume that we don't use any bias variables):

$$W = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix}, v = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

Assuming that we are doing regression, for a training example with features $x_i^T = \begin{bmatrix} -3 & -2 & 2 \end{bmatrix}$ what are the values in this network of z_i, $h(z_i)$, and $\hat{y}_i$?

Answer:

$$z_i = Wx_i = \begin{bmatrix} -2 & 2 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} -3 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} (-2)(-3) + 2(-2) + (-1)2 \\ 1(-3) + (-2)(-2) + 0(2) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$h(z_i) = \begin{bmatrix} \frac{1}{1+\exp(-0)} \\ \frac{1}{1+\exp(-1)} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{1+1/e} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{e}{1+e} \end{bmatrix}.$$

$$\hat{y}_i = v^T h(z_i) = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{e}{1+e} \end{bmatrix} = 3/2 + e/(1+e).$$

2.2 SGD for a Neural Network: implementation

Rubric: {code:5}

NOTE: before starting this question you need to download the MNIST dataset from `http://deeplearning.net/data/mnist/mnist.pkl.gz` and place it in your *data* directory.

If you run `python main.py -q 2` it will train a one-hidden-layer neural network on the MNIST handwritten digits data set using the `findMin` gradient descent code from your previous assignments. After running for the default number of gradient descent iterations (100), it tends to get a training and test error of around 5% (depending on the random initialization). Modify the code to instead use stochastic gradient descent. Use a minibatch size of 500 (which is 1% of the training data) and a constant learning rate of $\alpha = 0.001$. Report your train/test errors after 10 epochs on the MNIST data.

Answer: See code. On one particular run of 10 epochs, the training error is 8.5% and the test error is 8.3% (strangely, the test error is actually slightly *lower* than the training error). A note about the code: the implementation provided is the more straightforward implementation which re-uses the pre-existing `funObj`. See `https://piazza.com/class/jhjgkk9un036np?cid=675` for a discussion about this issue.

2.3 SGD for a Neural Network: discussion

Rubric: {reasoning:1}

Compare the stochastic gradient implementation with the gradient descent implementation for this neural network. Which do you think is better? (There is no single correct answer here!)

Answer: After only 10 epochs, SGD does better than 10 iterations of GD (which is the equivalent amount of computation). However, if you run the SGD for 100 epochs, it does worse than 100 iterations of GD. So, the question of which is better depends on how much time you have to wait. Also, this is a fairly minimal implementation of SGD, whereas a bit more effort was put into the GD implementation in `findMin` in terms of adaptively picking the step size at each iteration.

2.4 Hyperparameter Tuning

Rubric: {reasoning:2}

If you run `python main.py -q 2.4` it will train a neural network on the MNIST data using scikit-learn's neural network implementation (which, incidentally, was written by a former CPSC 340 TA). Using the default hyperparameters, the model achieves a training error of zero (or very tiny), and a test error of around 2%. Try to improve the performance of the neural network by tuning the hyperparemeters. Hand in a list changes you tried. Write a couple sentences explaining why you think your changes improved (or didn't improve) the performance. When appropriate, refer to concepts from the course like overfitting or optimization.

For a list of hyperparameters and their definitions, see the scikit-learn `MLPClassifier` documentation: `http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`. Note: "MLP" stands for Multi-Layer Perceptron, which is another name for artificial neural network.

Answer: This question is open-ended; there is no specific correct answer.

3 Very-Short Answer Questions

Rubric: {reasoning:12}

1. Is non-negative least squares convex?

   Answer: Yes (the area above the function is still a convex set if we restrict to the positive orthant).

2. Name two reasons you might want sparse solutions with a latent-factor model.

   Answer: Interpretability, faster computation.

3. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric?

   Answer: Unsupervised; Non-parametric.

4. Which is better for recommending moves to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.

   Answer: Content-based filtering. Collaborative filtering relies on ratings of that user, but there won't be any for a new user.

5. Collaborative filtering and PCA both minimizing the squared error when approximating each x_{ij} by $\langle w^j, z_i \rangle$; what are two differences between them?

   Answer: Collaborative filtering is used for recommender systems, is solving a supervised problem, and is usually done on a matrix where we don't know all the entries. PCA assumes you have all entries of the matrix, is an unsupervised method, and is used for a variety of things like visualization and factor discovery.

6. Are neural networks mainly used for supervised or unsupervised learning? Are they parametric or nonparametric?

   Answer: Supervised; parametric.

7. Why might regularization become more important as we add layers to a neural network?

   Answer: The model is becoming more complex, which means a higher danger of overfitting. Regularization combats overfitting.

8. With stochastic gradient descent, the loss might go up or down each time the parameters are updated. However, we don't actually know which of these cases occurred. Explain why it doesn't make sense to check whether the loss went up/down after each update.

   Answer: This would require iterating over all the examples, which is exactly what we're trying to avoid with SGD.

9. Consider using a fully-connected neural network for 3-class classification on a problem with $d = 10$. If the network has one hidden layer of size $k = 100$, how many parameters (including biases), does the network have?

   Answer: Weights: $10 \times 100 + 100 \times 3 = 1300$. Biases: $100 + 3 = 103$. Total: $1300 + 103 = 1403$.

10. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.

    Answer: No hidden layers (and a convex activation at the end). Or, all activation functions are linear, so it's just linear regression.

11. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?

Answer: Gradient might be zero because of overflow/underflow in the intermediate calculations (especially when repeated across layers)

12. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?

    Answer: The number of parameters would be huge; we would overfit massively and run into prohibitive speed/memory issues.