CPSC 340 Assignment 6 (due Friday April 3rd at 11:55pm)

Instructions

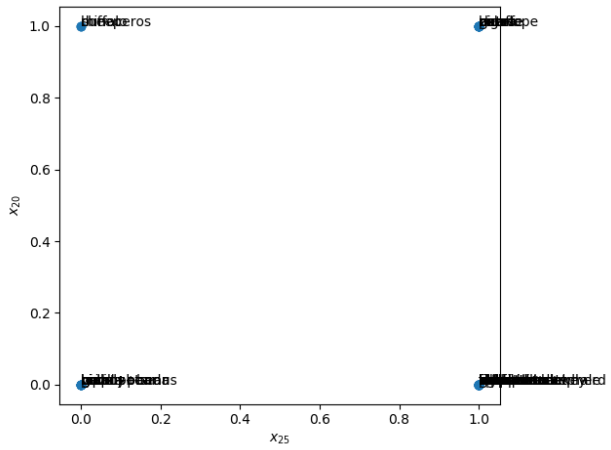
Rubric: {mechanics:5}

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

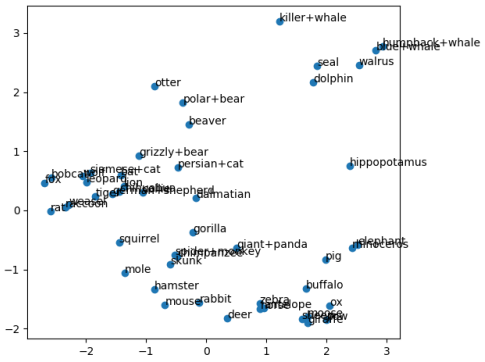
If you run `python main.py -q 1`, it will load the animals dataset and create a scatterplot based on two randomly selected features. We label some points, but because of the binary features the scatterplot shows us almost nothing about the data. One such scatterplot looks like this:



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).



The scatterplot seems to show similar animals to be clustered closer together. For example, whales, dolphins, and seals are obviously more related to each other than they are to cats, leopards, and tigers.

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?

1. 0.19019081 + 0.132941 = 0.32313181
   2D representation explains about 32% of the variance.
2. At least 5-dimensions, which explains about 54% of the variance.

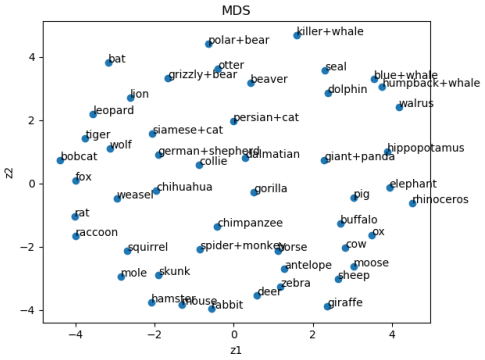
1.3 Multi-Dimensional Scaling

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?

The MDS objective with the PCA solution gives 4942.1954. The MDS solution gives 1776.8183, which is indeed lower than the PCA solution.

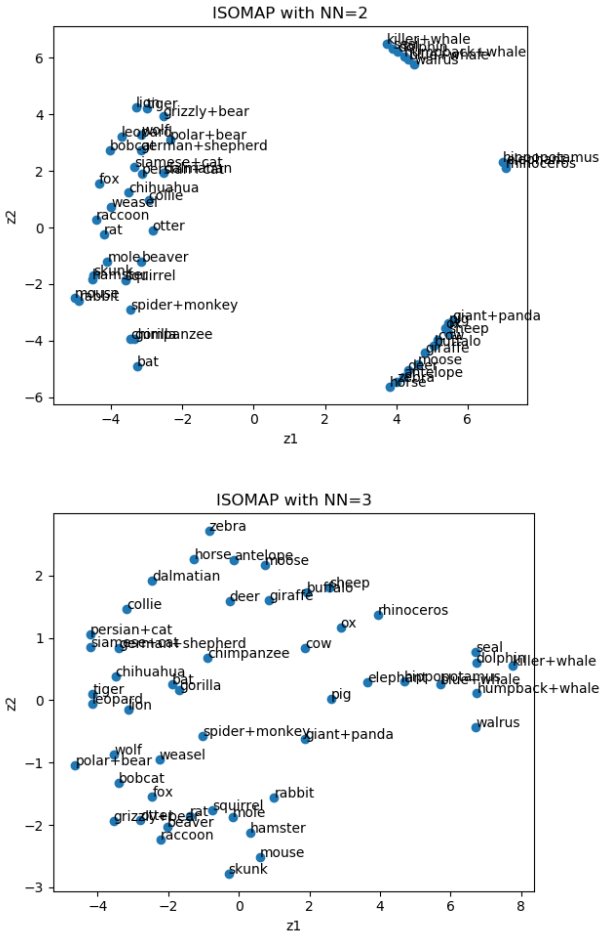
1.4 ISOMAP

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. Make sure to include the code you have written in your pdf GradeScope submission. Plot the results using 2 and using 3-nearest neighbours.

```
def compress(self, X):
    n = X.shape[0]

    D = utils.euclidean_dist_squared(X,X)
    D = np.sqrt(D)

    D_args = np.argsort(D)
    D_indx = np.argsort(D_args)

    for i in range(n):
        for j in range(n):
            if D_indx[i, j] <= self.nn:
                D_indx[j, i] = D_indx[i, j]

    adj = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            if D_indx[i, j] <= self.nn:
                adj[i, j] = 1
            if i == j:
                adj[i, j] = 0

    assert np.allclose(adj, adj.T)

    for i in range(n):
        for j in range(n):
            D[i, j] = utils.dijkstra(adj, i, j) #create the matrix

    D[np.isinf(D)] = D[~np.isinf(D)].max()

    pca = PCA(self.k)
    pca.fit(X)
    Z = pca.transform(X)
```

```
z,f = findMin(self._fun_obj_z, Z.flatten(), 500, D)
Z = z.reshape(n, self.k)
return Z
```



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.)

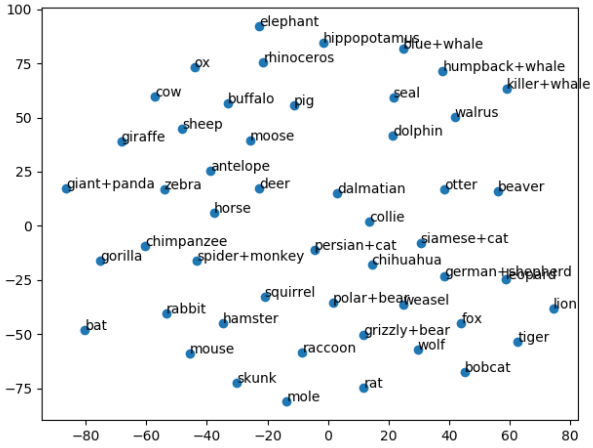
1.5 t-SNE

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.

I believe ISOMAP with nn=2 did the best job. The animals are clearly and distinctly clustered based on similarities. MDS and TSNE are also not bad, but the animals are not as differentiated.



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.

Only t-SNE gives different results when the code is re-run. This is because only t-SNE is sensitive to initialization, while the others are not. The random initializations result in different outputs.

2 Neural Networks

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.

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 $\hat{y}_i$?

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

$$h(z_i) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\hat{y}_i = v^T h(z_i) = 4$$

2.2 SGD for a Neural Network: implementation

Rubric: {code:5}

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.
Training error: 0.08448, Test error: 0.0811

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!)
The stochastic gradient implementation trains much faster (15 seconds) compared to the gradient descent implementation (82 seconds). Although the errors are a bit higher, I believe the speed of SGD makes it the better choice.

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.
Default: 0.0 training error, 0.0217 test error

activation='logistic': 0.0 training error, 0.0224 test error
The default uses the ReLU activation. With the logistic activation, the results are nearly the same. This may suggest that most inputs to the function are positive.

hidden layer sizes=500: 0.0 training error, 0.0178 test error
The default hidden layer size is 100. The test error decreases because the increase in hidden layers allows for more "accurate" and "precise" modeling of the data.

alpha=0.01: 0.0 training error, 0.021 test error
The default alpha is 0.0001. Increasing alpha increases L2 regularization, reducing overfitting, resulting in a slightly lower (likely insignificant) test error.

batch size=500: 0.0 training error, 0.0236 test error
Increasing the batch size from 200 to 500 also did not significantly change the errors. Since n=50000, the increase may have been too small to affect either training and testing.

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.

3 Very-Short Answer Questions

Rubric: {reasoning:12}

1. Is non-negative least squares convex?
   Yes, because we are taking the max of each w's against 0

2. Name two reasons you might want sparse solutions with a latent-factor model.
   Sparsity improves efficiency and solutions give us more interpretations to our model.

3. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric?
   ISOMAP is used for unsupervised learning and it is non-parametric because the model gets more complex with higher n and the final step of ISOMAP (MDS) depends on n.

4. Which is better for recommending moves to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.
   Content-Based Filtering is better because you can apply the prediction on new users.

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?
   For Collaborative filtering, the matrix (data) is not all filled in but PCA it is.

6. Are neural networks mainly used for supervised or unsupervised learning? Are they parametric or nonparametric?
   Neural networks are used for supervised learning because the model is based on the label vector. Also, neural networks are parametric because the number of weights are fixed regardless of n.

7. Why might regularization become more important as we add layers to a neural network?
   With more layers, there could be more over-fitting. Thus, regularization can combat this issue.

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.
   Stochastic gradient descent converges to a local minimum by following a random route. We do not care if one update caused the loss to increase or decrease, we only care that there is an overall decrease.

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?
   100*(10 + 3) + 100 + 3 = 1403

10. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.
    Hidden layers=0, activation='identity'

11. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?
    As more hidden layers are added to the neural net, the gradient of sigmoid loss functions converge to 0. This makes it hard to train a model with many hidden layers.

12. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?
    Convolutional networks decrease training time. In fully connected neural networks, each neuron is connected each neuron in the previous layer. Hence, the large number of weights result in higher training times.