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

Instructions

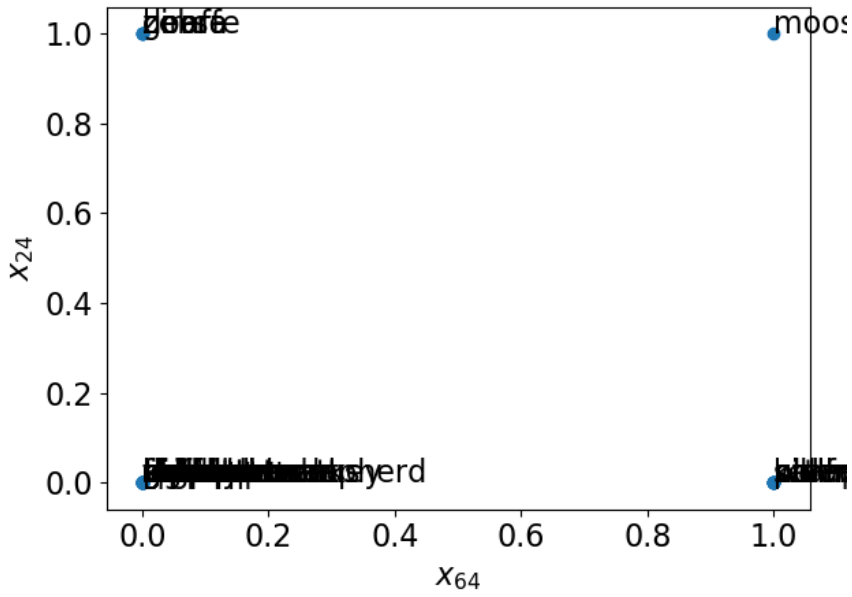
Rubric: {mechanics:5}

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

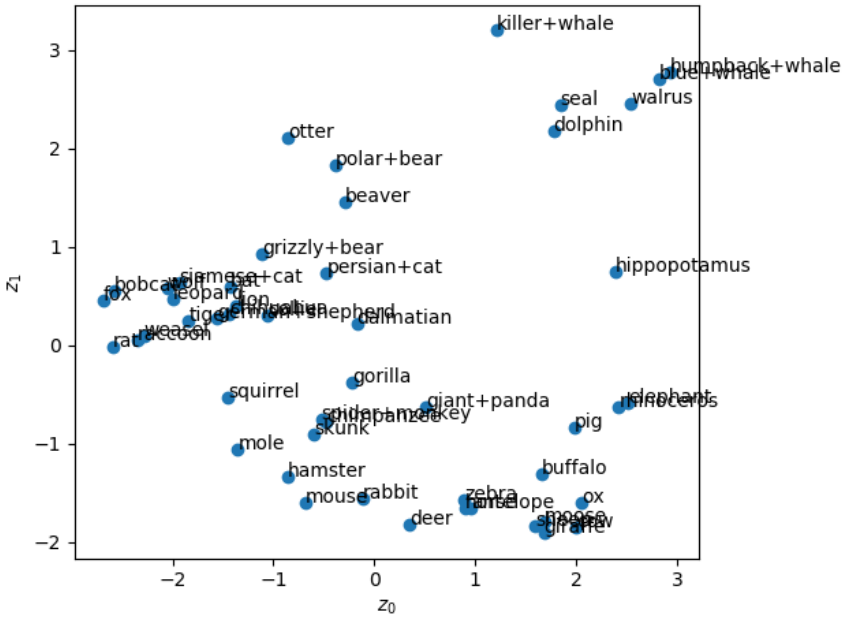
If you run `python main.py -q 1`, it will load the animals dataset and create a scatterplot based on two randomly selected features. We label some points, but because of the binary features the scatterplot shows us almost nothing about the data. One such scatterplot looks like this:



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 upper right corner has the aquatic animals such as dolphin and walrus and the lower left corner has the land animal such as squirrel and gorilla. The points spread through the whole plane and the classification is not very accurate.

1.2 Data Compression

Rubric: {reasoning:2}

1. How much of the variance is explained by our 2-dimensional representation from the previous question?
   The variance explained is 0.323
2. How many PCs are required to explain 50% of the variance in the data?
   We need 5 or more PCs to explain 50% of the variance in the data.

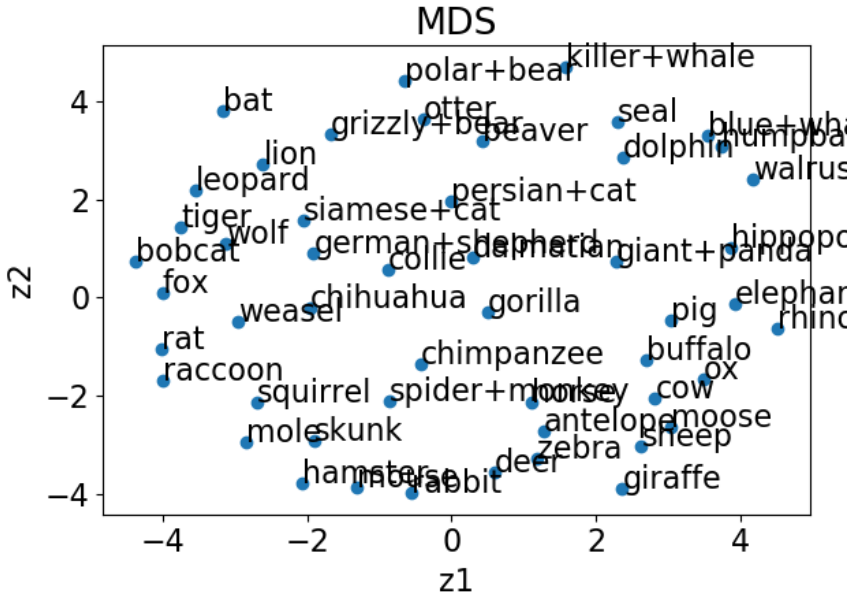
1.3 Multi-Dimensional Scaling

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):

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

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 for the PCA solution is 4942.19545302 and the loss for the MDS solution is 1776.81831128 so it is indeed lower for the MDS solution because the MDS solution comes from optimizing the MDS objective.

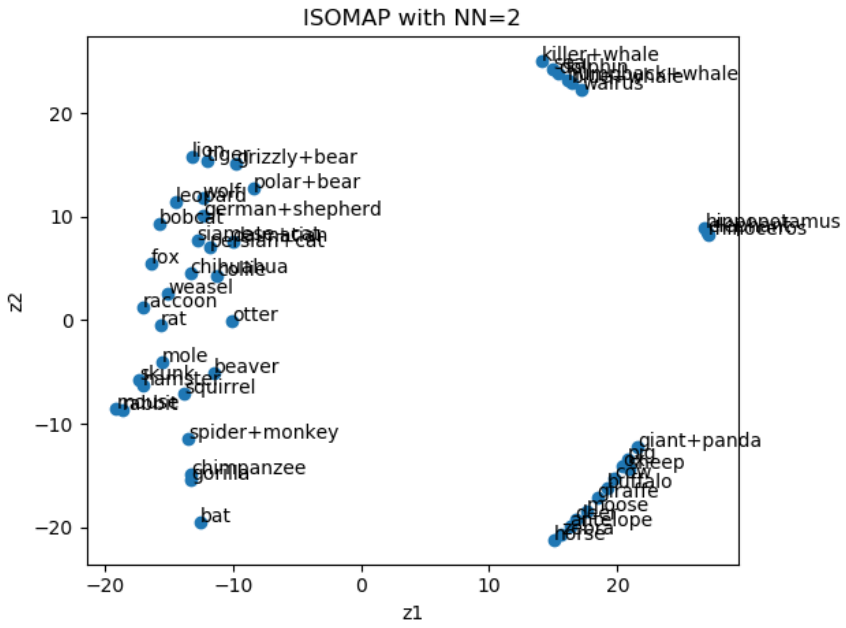
1.4 ISOMAP

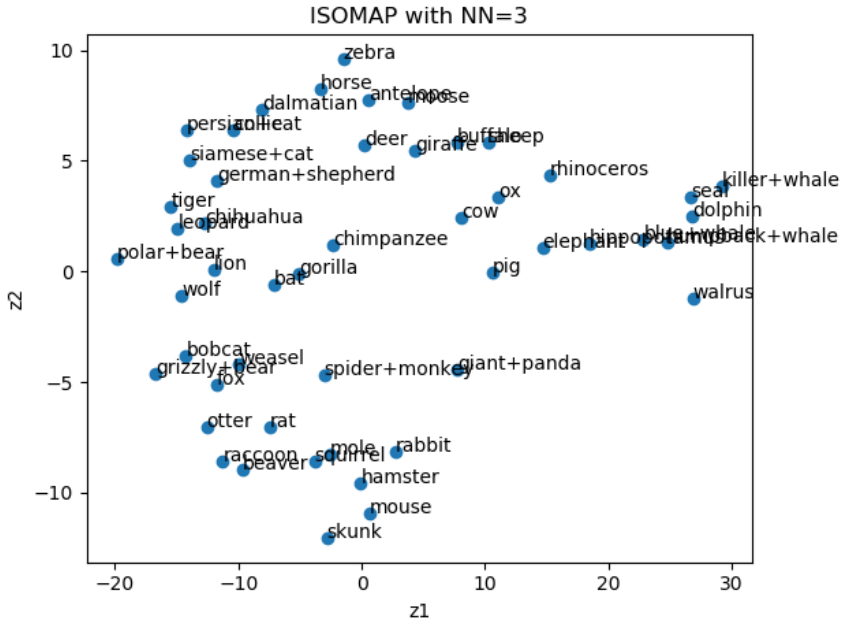
Rubric: {code:10}

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.)





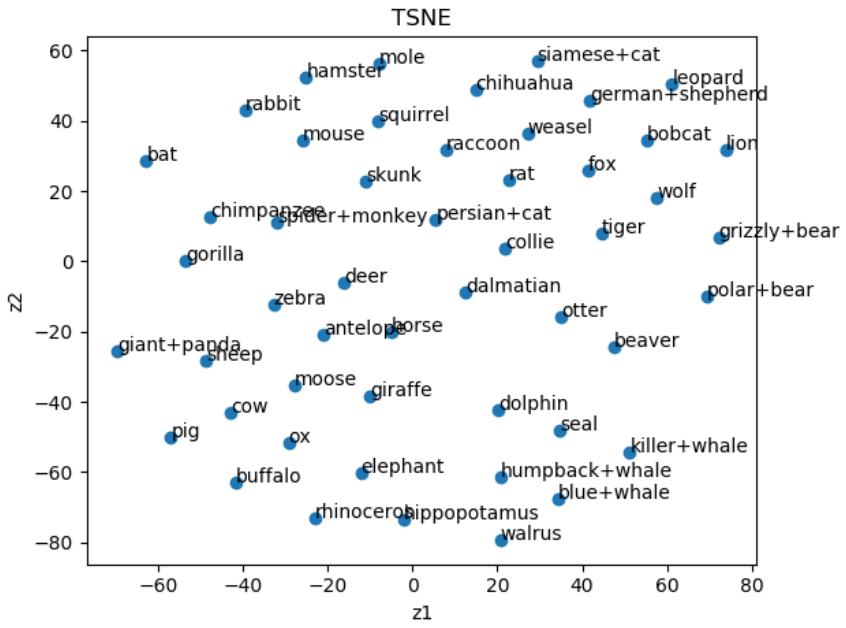

```
class ISOMAP(MDS):

    def __init__(self, n_components, n_neighbours):
        self.k = n_components
        self.nn = n_neighbours

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

```
# Compute Euclidean distances
D = utils.euclidean_dist_squared(X,X)
D = np.sqrt(D)

# TODO: Convert these Euclidean distances into geodesic distances
G = np.zeros([n, n])
for i in range(n):
    neighbours = np.argsort(D[i])[:self.nn + 1]
    for j in neighbours:
        G[i, j] = D[i, j]
        G[j, i] = D[j, i]
D = utils.dijkstra(G, i=None, j=None)

# If two points are disconnected (distance is Inf)
# then set their distance to the maximum
# distance in the graph, to encourage them to be far apart.
D[np.isinf(D)] = D[~np.isinf(D)].max()

# Initialize low-dimensional representation with PCA
pca = PCA(self.k)
pca.fit(X)
Z = pca.transform(X)

# Solve for the minimizer
z,f = findMin(self._fun_obj_z, Z.flatten(), 500, D)
Z = z.reshape(n, self.k)
return Z
```

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?



In my opinion, ISOMAP (NN = 2) did the best job because it clearly classified object into several groups, even if it is not completely accurate.

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

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 for the t-SNE method the result changes when I re-run the code. PCA with SVD is not sensitive to initialization but PCA with other methods such as gradient descent and SGD are sensitive to initialization. MDS, ISOMAP, and t-SNE are all sensitive to initialization because the for unsupervised learning the algorithms do not know the labels. The reason that in our case MDS, ISOMAP and t-SNE are not sensitive is because they all use PCA with SVD for initialization.

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 $\hat{y}_i$?

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

$h(z_i) = \begin{bmatrix} \frac{1}{1+exp(-0)} \\ \frac{1}{1+exp(-1)} \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.731 \end{bmatrix}$

$\hat{y}_i = v^T h(z_i) = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.731 \end{bmatrix} = 2.231$

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 = 0.08464
Test error = 0.0808

```
def findMinStochastic(funObj, w, maxEvals, X, y, verbose=0):

    funEvals = 1
    alpha = 0.001
    batch_size = 500
    n_epochs = 10


    for epoch in range(n_epochs):
        # Line-search using quadratic interpolation to
        # find an acceptable value of alpha
        X, y = shuffle(X, y)
        N, d = X.shape

        for i in range(0, N, batch_size):
            Xbatch = X[i:i+batch_size]
            ybatch = y[i:i+batch_size]

            f, g = funObj(w, Xbatch, ybatch)
            w = w - alpha * g

            funEvals += 1
```

```
        # Print progress
        if verbose > 0:
            print("%d - loss: %.3f" % (funEvals, f))


    return w, f
```

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!)
Stochastic gradient descent is better. Even though it gives a higher training and test error than gradient descent, it is significantly faster than the gradient descent.

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.
$batch_size = 100, learning_rate = "constant"$
$Trainingerror = 0.0$
$Testerror = 0.0195$
$batch_size = 100, learning_rate = "invscaling"$
$Trainingerror = 0.0$
$Testerror = 0.0207$
$batch_size = 100, learning_rate = "adaptive"$
$Trainingerror = 0.0$
$Testerror = 0.0218$
$batch_size = 50, learning_rate = "constant"$
$Trainingerror = 0.00028$
$Testerror = 0.0194$
$hidden_layer_sizes = 200, alpha = 0.0001$
$Trainingerror = 0.00076$
$Testerror = 0.0223$
$hidden_layer_sizes = 200, alpha = 0.001$
$Trainingerror = 0.0$
$Testerror = 0.0183$
$hidden_layer_sizes = 200, alpha = 0.1$
$Trainingerror = 0.00562$

$Testerror = 0.0199$
$hiddenlayersizes = 300, alpha = 0.0001$
$Trainingerror = 0.0$
$Testerror = 0.0173$
$hiddenlayersizes = 300, alpha = 0.001$
$Trainingerror = 0.0$
$Testerror = 0.0178$
$hiddenlayersizes = 300, alpha = 0.1$
$Trainingerror = 0.00644$
$Testerror = 0.022$

When the number of hidden layers increases, test error decreases but it may overfit as the training error may be exactly 0. When the batch size decreases, test error decreases but it may overfit as well. As we increase the regularization strength, the training error increases and the test error decreases so it gives better approximation error which reduces overfit.

3 Very-Short Answer Questions

Rubric: {reasoning:12}

1. Is non-negative least squares convex?
   Yes, non-negative least square is convex since the solution is the max of the least square solution or 0 and the max of 2 convex functions is convex.

2. Name two reasons you might want sparse solutions with a latent-factor model.
   A sparse solution provides cheaper predictions and has more interpretability since non-negative weights are often more interpretable.

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.

4. Which is better for recommending movies to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.
   Content-based filtering is better for recommending movies for a new user. Collaborative filtering learns about each user/movie but can't predict on new users/movies. Content-based filtering can predict on new users/movies but can't learn about each user/movie.

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?
   Collaborative filtering tries to predict(fill in unknown entries) the best x_{ij} in the matrix while PCA tries to find the low dimension feature z_i that best represent the data.

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

7. Why might regularization become more important as we add layers to a neural network?
   As we add layers to a neural network, the depth of the neural network increases. Since increase in depth results in overfitting, we need to use regularization to keep the model complexity under control.

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.
   With Stochastic gradient descent, it is possible for some example to point in the wrong direction which

results the loss to go up. There is no need to check whether the loss went up/down because the algorithm is going in the right direction on average and we don't mind if the loss is going up in some instances.

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?
   It has 11 * 100 + 101 * 3= 1100 + 303 = 1403 parameters.

10. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.
    The loss for a neural network is non-convex because the sigmoid function is non-convex. We need to choose a active function that is convex (e.g. the identity function).

11. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?
    The "vanishing gradient" problem is the underflow/overflow during gradient calculation.

12. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?
    Training on a 256 by 256 image using normal neural network with k = 10000 and 3 layers has 2 billion parameters. This hugh number of parameters results in storage issues and overfitting.