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

Instructions

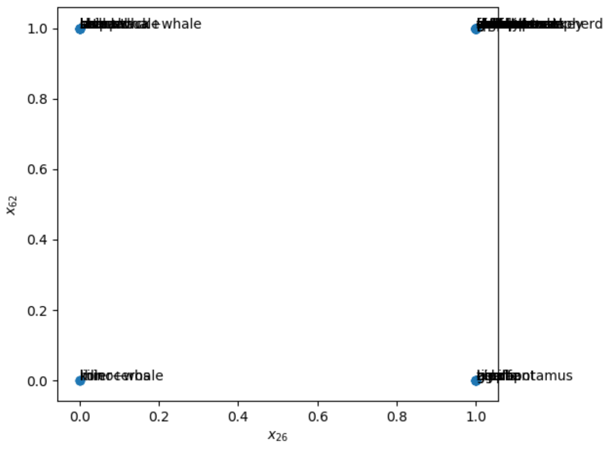
Rubric: {mechanics:5}

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:

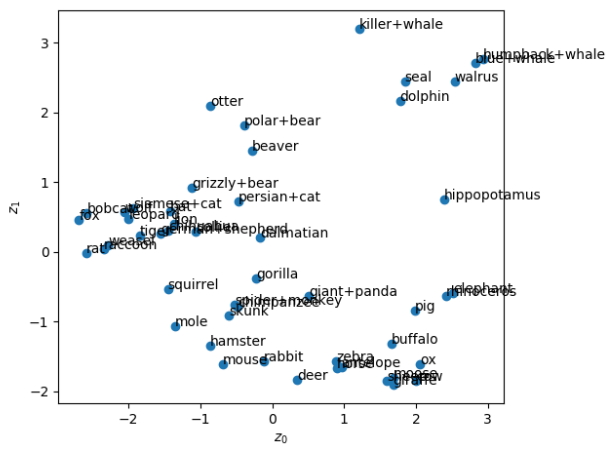


1.1 PCA for visualization

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).

Answer: The 2 dimensional plot looks like a good depiction of the dataset because similar animals are close to one another, for example seal, dolphin and walrus are close to one another and hamster, mouse and rabbit are close to one another.



1.2 Data Compression

Rubric: {reasoning:2}

1. How much of the variance is explained by our 2-dimensional representation from the previous question? Answer: 0.32313181181376877 of the variance is explained by our 2-dimensional representation
2. How many PCs are required to explain 50% of the variance in the data? Answer: 5 PCs are required to explain 50% of the variance in the data.

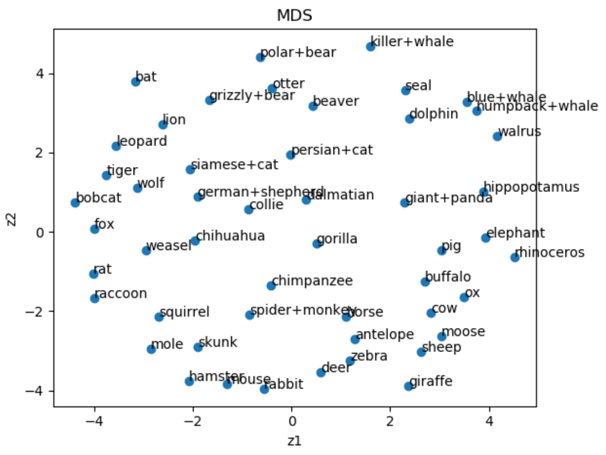
1.3 Multi-Dimensional Scaling

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):

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

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: It is indeed lower for the MDS solution since the function compress in manifold.py initializes Z to be the PCA's solution then applies the minimizer. The minimizer's solution by definition gives an error less than or equal to the initial value which, in this context, is PCA's solution.

1.4 ISOMAP

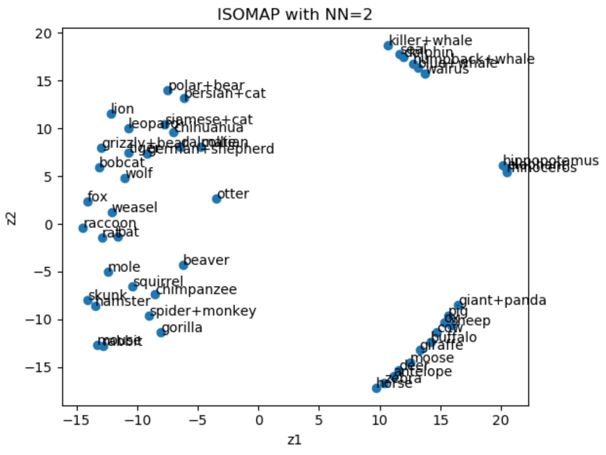
Rubric: {code:10}

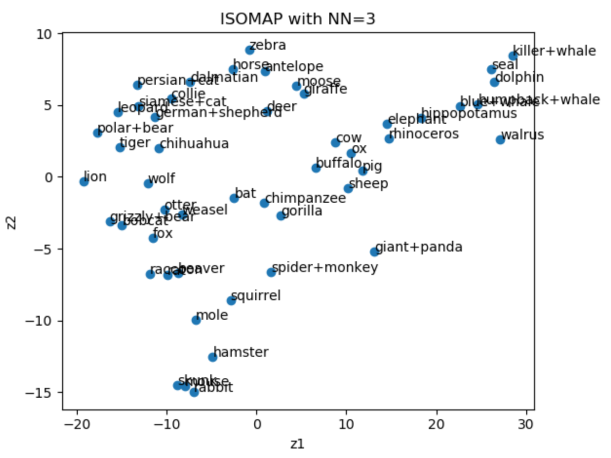
Euclidean distances between very different animals are unlikely to be particularly meaningful. However, since related animals tend to share similar traits we might expect the animals to live on a low-dimensional manifold. This suggests that ISOMAP may give a better visualization. Fill in the class *ISOMAP* 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.

```
# TODO: Convert these Euclidean distances into geodesic distances
G = np.zeros((n,n))
for j in range(n):
    # Find indices of nn + 1 nearest neighbours (including itself)
    ind = np.argpartition(D[j,:], (self.nn+1))
    ind = ind[:(self.nn + 1)]
    print(print(j), print(ind))
    for l in ind:
        G[j][l] = D[j][l]
        G[l][j] = D[l][j]

for j in range(n):
    for l in range(j, n):
        D[j][l] = utils.dijkstra(G, j, l)
        D[l][j] = D[j][l]
```

Answer: Below are the plots for NN = 2 and NN = 3





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.

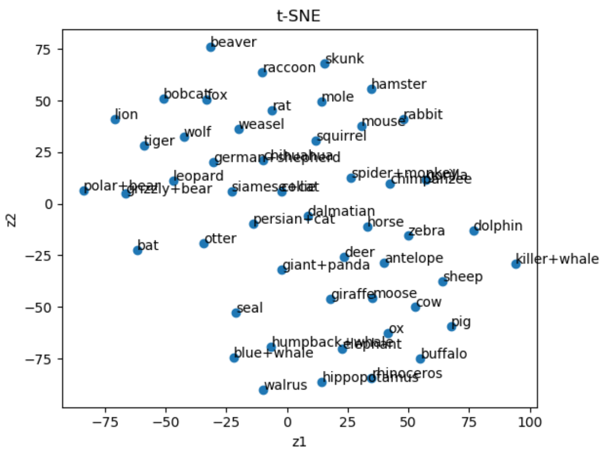
The function *utils.dijskstra* can be used to compute the shortest (weighted) distance between two points in a weighted graph. This function requires an $n \times n$ matrix giving the weights on each edge (use 0 as the weight for absent edges). Note that ISOMAP uses an undirected graph, while the k-nearest 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.



Answer: Visually it seems that PCA and ISOMAP gave more concentrated clusters, while MDS and t-SNE gave much more scattered points. In my opinion, PCA did the best job because it grouped together animals that are very similar. So, if we were to consider the most natural thing to do with this data set which is clustering in order to tell how many species or subspecies there are then PCA does a better job than the rest.

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 give different results when I reran the code. This does not match up with what we discussed in lectures because MDS and ISOMAP (which is based on MDS) are also supposed to be sensitive to initialization, but in the implementation of the given MDS class the initialization was set to be PCA's solution, which is fixed.

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 $\hat{y}_i$?

Answer: $z_i = Wx_i = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

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

$\hat{y}_i = v^T h(z_i) = \frac{3}{2} + \frac{e}{e+1} = \frac{5e+3}{2e+2}$

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.

Answer:
Fitting took 12 seconds
Training error = 0.08606
Test error = 0.0841

```
def fitWithSGD(self, X, y, alpha = 0.001, minibatch_size = 500, epochs
    = 10):
    if y.ndim == 1:
        y = y[:,None]

    self.layer_sizes = [X.shape[1]] + self.hidden_layer_sizes + [y.
        shape[1]]
    self.classification = y.shape[1]>1 # assume it's classification
        iff y has more than 1 column

    # random init
    scale = 0.01
    weights = list()
        for i in range(len(self.layer_sizes)-1):
            W = scale * np.random.randn(self.layer_sizes[i+1],self.
                layer_sizes[i])
            b = scale * np.random.randn(1,self.layer_sizes[i+1])
            weights.append((W,b))
            weights_flat = flatten_weights(weights)

    for j in range(epochs):
        ind = np.random.permutation(X.shape[0])
        for i in range (0, X.shape[0], minibatch_size):
            l = min(minibatch_size, X.shape[0] - 1 - i)
            X_rand = X[ind[i: i + l], :]
            y_rand = y[ind[i: i + l]]
            f, g = self.funObj(weights_flat, X_rand, y_rand)
            weights_flat = weights_flat - alpha *  g

    self.weights = unflatten_weights(weights_flat, self.layer_sizes)
```

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: The gradient descent implementation for this neural network is better because it often gives less errors than stochastic gradient, and it did not take much more time for this dataset. For example:
Gradient descent took 80 seconds with Training error = 0.04132 and Test error = 0.0455
Stochastic gradient descent took 10 seconds with Training error = 0.08522 and Test error = 0.0805

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.

Answer: hidden_layers_size changed to $(100, 100)$ from 100 (default value) and the test error decreases from 0.0222 to 0.0211, the training error decreases from 2×10^{-5} to 0. Based on this we can conclude that the model was probably underfitting. On the otherhand, if we change α (the L2 regularization parameter) from 0.0001 to 0.1 then the training error increases from 2×10^{-5} to 0.0056, but the test error decreases from 0.0222 to 0.0217. This can be explained by the increase of the regularization parameter so there is a much more focus on regularization. This makes the training error increase and the test error decrease.

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? Answer: Non-negative least squares is convex.

2. Name two reasons you might want sparse solutions with a latent-factor model. Answer: Because sparsity makes operations on the solutions much more efficient. It is also cheaper to store.

3. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric? Answer: ISOMAP is mainly used for unsupervised learning and it is 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 filtering because for a new user, we cannot use collaborative filtering to see which users this new user is similar to since we do not have any responses (e.g. ratings) that this new user gives for the movies. Content filtering is better because it is based on concreate features, so it can ask for the new user's features (e.g. preferences) and recommend movies based on the feature-oriented model that it obtains based on the other users' features and the movies they watched.

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 regularized while original PCA is not regularized. Collaborative filtering tries to fill in the missing entries of the matrix so that the loss sum over the entries (i, j) is minimized (e.g. so that similar users rate similar movies with similar ratings). In PCA, all entries are given and we try to find the components of our examples and express the examples as a linear combination of these components.

6. Are neural networks mainly used for supervised or unsupervised learning? Are they parametric or nonparametric? Answer: 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? Answer: The more layers we add to the neural network the more overfitting it is because the complexity increases.

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 and on average the loss Answer: It is extra cost to check whether the loss went up/down after each update and this defeats the purpose of stochastic gradient descent since it would take a higher number of iterations and the cost would accumulate.

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: We have k = 100, d = 10. W is k x d so there are 1000 parameters in W. v is k x 3 (since we are doing 3-class classification) so there are 300 parameters in v. For biases, there are k neurons in the hidden layers and 3 in the output so there are 103 bias components. In total we have **1103 parameters**.

10. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.
Answer: We can use features from a latent-factor model by setting $z_i = Wx_i$ and then we can make predictions using a linear model $y_i = v^T z_i$. Then we aim to minimize the convex loss function

$$f(W, v) = \frac{1}{2}\sum_{i=1}^{n}(v^T z_i - y_i)^2.$$

11. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?
Answer: Away from the origin, a sigmoid function has gradient close to 0. In deep neural networks, the gradient can be nearly zero almost everywhere.

12. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification? Answer: The number of parameters in a regular neural network can get very huge very quickly, and this can cause storage and overfitting problems.