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

Student name: José Abraham Torres Juárez
Student id: 79507828

## Instructions

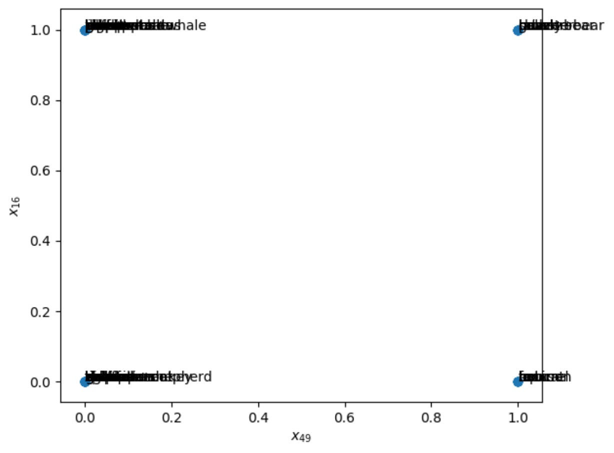
Rubric: {mechanics:5}

**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

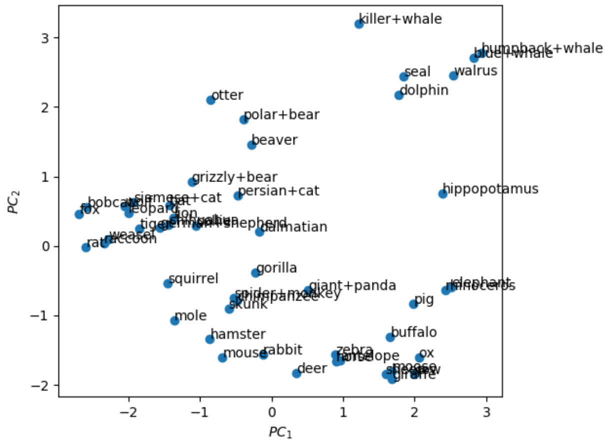
If you run `python main.py -q 1`, it will load the animals dataset and create a scatterplot based on two randomly selected features. We label some points, but because of the binary features the scatterplot shows us almost nothing about the data. One such scatterplot looks like this:



### 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 image above shows the 2 principal components of the provided data set. It is a much better representation of the data, than just the visualization of two random features. By using PCA we can see a remarkable separation between acuatic and terrestrial animals. We can even notice that animals that live on both water and land are on the boundaries, like otter, beaver, polar bear, hippopotamous.
But this representation doesn't make a good work separating the terrestrial animal, we can see grizzly bear, closer to some cats, rather than the polar bear.

## 1.2 Data Compression

Rubric: {reasoning:2}

1. How much of the variance is explained by our 2-dimensional representation from the previous question?
   By using two components we can explain 32.31% of the variance.
2. How many PCs are required to explain 50% of the variance in the data?
   Using 4 components we can represent 48.62% of the variance, furthermore by using 5 components we can represent 54.46% of the data.

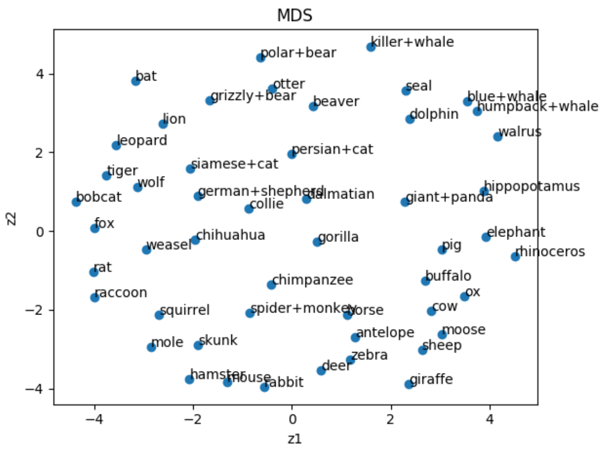
## 1.3 Multi-Dimensional Scaling

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?

By computing the objective function with the PCA solution the value is 4942.1954, and we can notice a big improvement by calling it on the MDS solution with a value of 1776.8183. It was reduced by more than one half.

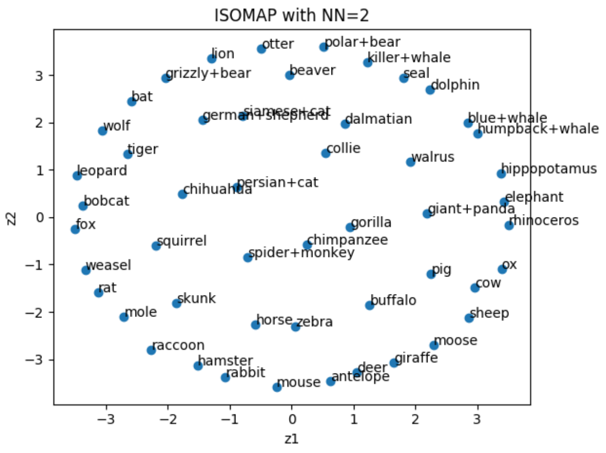
## 1.4 ISOMAP

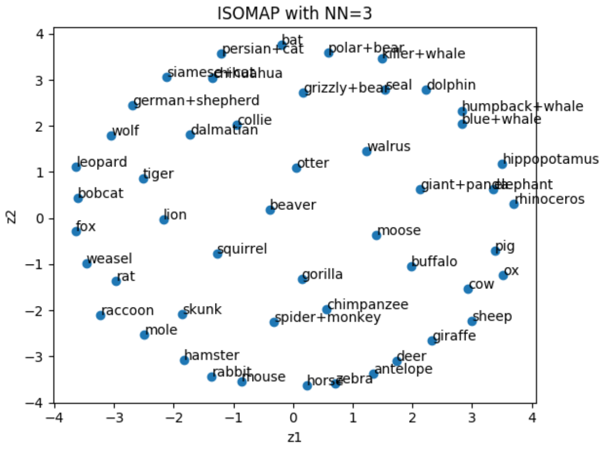
Rubric: {code:10}

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.)



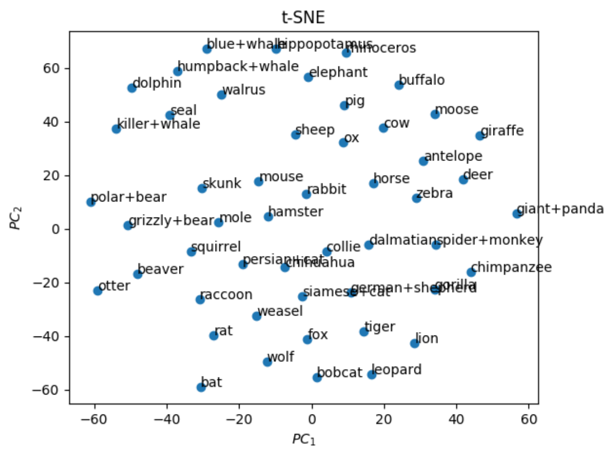


## 1.5 t-SNE

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.



ISOMAP yields a better solution, each animal is being surrounded of all the animals with the similar characteristics, we can also notice that some animals that do not have too similar neighbours are being pushed away to the center. Like the otter, its closest neighbour is the beaver but also the walrus and a collie.

### 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.
The only one of the previous methods that is sensitive to initialization is the t-SNE. This is because t-SNE does a work similar to k-means. It depends on how the data points are randomly projected on to the reduced space, which yield different results on each run. On the other hand PCA, MDS, ISOMAP are deterministic they will always yield the same solution under the same data set and parameters.

## 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 $\hat{y}_i$?

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

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

$$\hat{y}_i = \begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.731 \end{bmatrix} = 2.231$$

## 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.
On 10 epochs the training error was 0.10648, and the test error 0.0992, after several runs it is consistently finishing with errors around 10%

## 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!)
In terms of accuracy the gradient descent implementation, does a much beter obtaining a 5% error, whereas the SGD obtains an error around 10%. But in terms of efficiency SGD does an awesome work getting to 10% of error in around 4 seconds, when the gradient descent takes around 75 seconds to reach the max iterations.

In conclusion, out of the box normal GD almost ensure you good results at a high time cost. On the other hand SGD ensure you efficiency, with a proper hyperparameter tunning SGD will beat normal GD.

## 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.
The default parameters for the MLPClassifier, get a 2% test error with a running time around 60 to 70 seconds.

I tried changing the default solver from *adam* to *sdg*, with different values on exclusive parameters of *sdg* like `momentum`, but didn't do any better than the *adam* solver.

By using *adam* and the following parameters, the `batch_size = 400` (default 200) give more data to the to use on each epoch which makes the gradient a better approximation, `learning_rate_init = 0.007` (default 0.001) increase the model speed of the gradient, finally by setting `alpha = 0.0002` we prevent overfitting.

With these parameters the time was reduced around 25 to 30 seconds, to a half of the original time while still preserving a 2% testing error, but with a little increase on the training error to 0.1%.

## 3 Very-Short Answer Questions

Rubric: {reasoning:12}

1. Is non-negative least squares convex?
   Yes, it is a p norm, and we know that every p norm is a convex function.
2. Name two reasons you might want sparse solutions with a latent-factor model.
   One is to reduce the storage of our latent-factor, so we can represent the same data in less space. Another advantage is that this also reduces the time of computing operations with an sparse matrix, which can reduce the training time significantly.
3. Is ISOMAP mainly used for supervised or unsupervised learning? Is it parametric or non-parametric?
   ISOMAP is a non-parametric method that would be used for unsupervised learning, to reduce the dimensionality of the data set and make it easier to clusterize. An example of this, is run ISOMAP and with the ouput run kmeans or dbscan.
4. Which is better for recommending moves to a new user, collaborative filtering or content-based filtering? Briefly justify your answer.
   For a new user content-based filtering probably would do a better job than collaborative filtering, because the user is new, we not have any data about he has "bought" to compare it between other 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?
   PCA doesn't have regularization whereas collaborative filtering does have.
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 nonparametric.
7. Why might regularization become more important as we add layers to a neural network?
   The complexity of the model grows as we add more layers, making it easier to overfit so regularization will help us in reducing the overfitting of our model.
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.

We know that in general the gradient will point to a minima, the stochastic gradient might point on the wrong direction i.e. the training sample is an outlier. So we dont have to check if after each update the loss goes up or down, in overall after several updates we will be approaching to the minima.

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?
   There will be 11 * 101 = 1111 parameters for the features (10 + 1 bias) and the hidden layer (100 + 1 bias), plus 101 * 3 = 303 the weight for connecting each node in the hidden layer with each class. So in total 1414.

10. The loss for a neural network is typically non-convex. Give one set of hyperparameters for which the loss is actually convex.
    If we set 0 hidden layers the neural network becomes just logistic regression which has a convex loss function.

11. What is the "vanishing gradient" problem with neural networks based on sigmoid non-linearities?
    By plotting a sigmoid function we can notice that the gradient on both extremas of the graph tend to be 0, this will cause the neural network to produce smaller and smaller weights until these weights disappear.

12. Convolutional networks seem like a pain... why not just use regular ("fully connected") neural networks for image classification?
    Convolutional network allow us to extract "significant" features by applying convolutions to the data, the result of these convolutions would make it much easier to classify rather than just using the plain features in a fully connected network.

    Another advantage is that we will need less space and time to be used. So when we are training with big data sets, it will have a much better performance than using just fully connected graph.