Part I: Multi-class Classification

Note: This Part I does not require coding and is also quite light.

Problem 1: Show the multi-class Softmax reduces to two-class Softmax when C = 2 (30 points)

Show that the multi-class Softmax cost in Equation (7.24) reduces to the two-class Softmax cost in Equation (6.37)

$$g\left(\mathbf{w}_0, \ldots, \mathbf{w}_{C-1}\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1 + \sum_{\substack{j=0 \\ j\neq y_p}}^{C-1} e^{\mathring{\mathbf{x}}_p^T\left(\mathbf{w}_j - \mathbf{w}_{y_p}\right)}\right). \tag{7.24}$$

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1 + e^{-y_p\mathring{\mathbf{x}}_p^T\mathbf{w}}\right) \tag{6.37}$$

Problem 2: Balanced accuracy in the multi-class setting (20 points)

Extend the notion of balanced accuracy to the multi-class setting. In particular, give an equation for the balanced accuracy in the context of multi-class classification that is analogous to Equation (6.79) for the two-class case.

$$\mathcal{A}_{\text{balanced}} = \frac{\mathcal{A}_{+1} + \mathcal{A}_{-1}}{2}. \tag{6.79}$$

Where A_{+1} and A_{-1} denote the accuracy computed on class +1 and -1 separately.

Part II: Unsupervised Learning

Problem 1: Producing a PCA basis (20 points)

Repeat the experiment described below on the provided dataset (2d_span_data.csv), reproducing the illustrations shown in Figure 1. You may use the Python implementation given below as a basis for your work.

Hint: you can use the pca_visualizer() function in the companion .py file to reproduce the plot exactly the same as shown below, but you are also welcome to write your own plotting code, in which case you are not required to reproduce exactly the same plot, but rather just the essentials – namely, all the data points before and after transformation and the PCs (just two indicative red lines are sufficient and the arrows are not compulsory).

Experiment
In the left panel of Figure 1 we show the mean-centered data, along with its two principal components (pointing in the two orthogonal directions of greatest variance in the dataset) shown as red arrows. In the

right panel we show the encoded version of the data in a space where the principal components are in line with the coordinate axes.

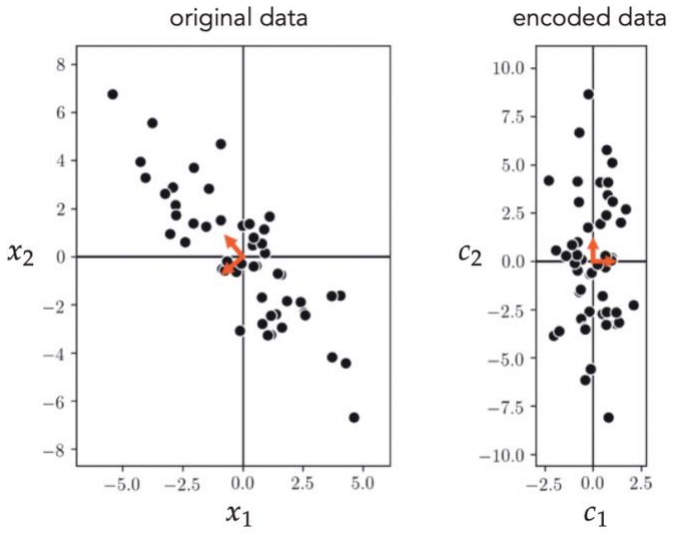

Figure 1

```python
# center an input dataset X
def center(X):
    X_means = np.mean(X,axis=1)[:,np.newaxis]
    X_centered = X - X_means
    return X_centered
```

```python
# function for computing principal components of input dataset X
def compute_pcs(X,lam):
    # create the data covariance matrix
    P = float(X.shape[1])
    Cov = 1/P*np.dot(X,X.T) + lam*np.eye(X.shape[0])

    # use numpy function to compute eigenvectors / eigenvalues
    D,V = np.linalg.eigh(Cov)
    return D,V
```

Problem 2: Perform K-means (20 points)

Implement the K-means algorithm detailed and apply it to properly cluster the dataset as shown in the left panel of Figure 2 using K = 3 cluster centroids. Visualize your results by plotting the dataset, coloring each cluster a unique color, as well as plotting the 3 centroids in a different shape than the data points. You may use the Python code given below to create a dataset.

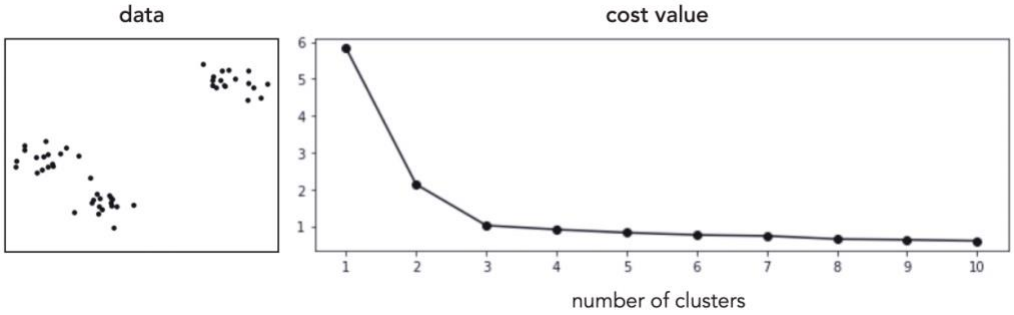

Figure 2

```
P = 50 # Number of data points
from sklearn import datasets
blobs = datasets.make_blobs(n_samples=P,centers = 3)
data = np.transpose(blobs[0])
```

Problem 3: Making a scree plot (10 points)

Repeat the experiment described below, reproducing the illustration shown in the right panel of Figure 2. You use the same procedure as described in problem 2 to create a dataset

Experiment
To determine the optimal setting of the parameter K, i.e., the number of clusters in which to cluster the data, we typically must try a range of different values for K, run the K-means algorithm in each case, and compare the results using an appropriate metric like the average intra-cluster distance in Equation (8.28). Of course, if we achieve an optimal clustering for each value of K (perhaps running the algorithm multiple times for each value of K) then the intra-cluster distance should always go down monotonically as we increase the value of K since we are partitioning the dataset into smaller and smaller chunks.

$$\text{average intra-cluster distance} = \frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{x}_p - \mathbf{c}_{k_p}\right\|_2 . \qquad (8.28)$$

For example, in Figure 2 we show the results of running ten runs of K-means ranging the value of K from K = 1, . . . , 10 and keeping the clustering that provided the lowest intra-cluster distance for each value of K for the dataset shown in the left panel of the figure. In the right panel we plot the best distance value attained for each value of K tried, a plot often referred to in the jargon of machine learning as a scree plot.

As one should expect, the intra-cluster distance decreases monotonically as we increase K. Notice, however, that the scree plot above has an elbow at K = 3, meaning that increasing the number of clusters from three to four and onwards reduces the distance value by very little. Because of this we can argue that K = 3 is a good choice for the number of clusters for this particular dataset (which also makes sense in this instance, since we can visualize the dataset and clearly see that it has three clusters) since any fewer clusters and the intra-cluster distance is comparatively large, while adding additional clusters does not decrease the total intra-cluster distance too much.

This illustrates the typical usage of the scree plot for deciding on an ideal number of clusters K for K-means. We compute and then plot the intra-cluster distance over a range of values for K, and pick the value at the ”elbow” of the plot. In practice this value is often chosen subjectively (by visual analysis of the scree plot) as was done here.