In supervised classification we learn a function f mapping the feature vector to a class label. In this discussion we will go over two classification techniques:

- Instance based classification (k-NN)
- Linear classification (Softmax)

1 k-nearest neighbor classification (k-NN)

In the classification setting, the simplest incarnation of the k-NN model is to predict the target class label as the class label that is most often represented among the k most similar training examples for a given query point. In other words, the class label can be considered as the "mode" of the k training labels or the outcome of a "plurality voting." Note that in literature, kNN classification is often described as a "majority voting." While the authors usually mean the right thing, the term "majority voting" is a bit unfortunate as it typically refers to a reference value of $> 50\%$ for making a decision. In the case of binary predictions (classification problems with two classes), there is always a majority or a tie. Hence, a majority vote is also automatically a plurality vote. However, in multi-class settings, we do not require a majority to make a prediction via k-NN. For example, in a three-class setting a frequency $> \frac{1}{3}$ (approx 33.3%) could already be enough to assign a class label.

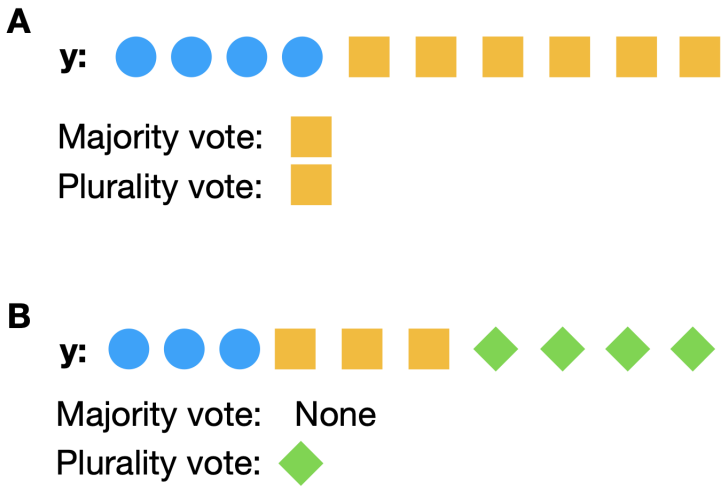

Figure 1: Illustration of plurality and majority voting.

A common distance metric to identify the k nearest neighbors $\mathcal{D}_k$ is the Euclidean distance

measure,

$$d(\mathbf{x}^{(a)}, \mathbf{x}^{(b)}) = \sqrt{\sum_{j=1}^{m} (\mathbf{x}_j^{(a)} - \mathbf{x}_j^{(b)})^2}$$

which is a pairwise distance metric that computes the distance between two data points $\mathbf{x}^{(a)}$ and $\mathbf{x}^{(b)}$ over the m input features.

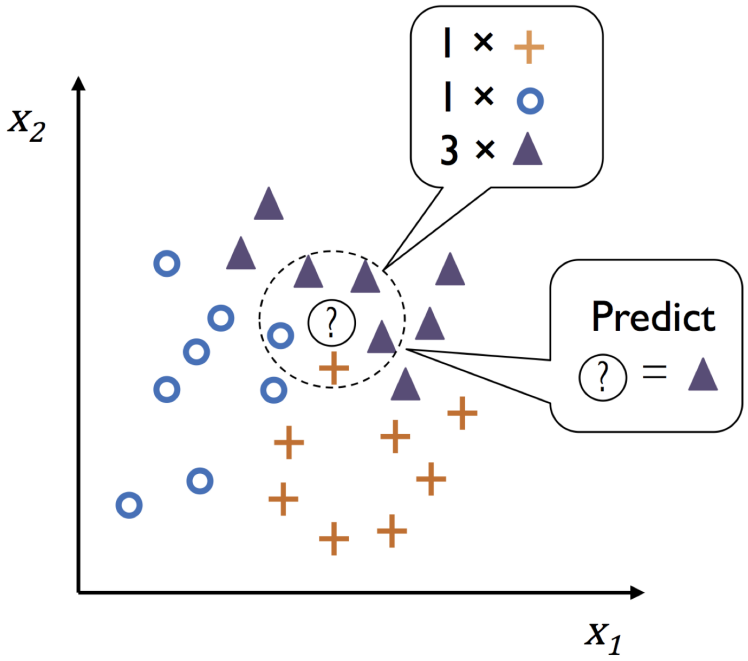

Figure 2: Illustration of k-NN for a 3-class problem with $k = 5$.

1.1 Curse of dimensionality

The k-NN algorithm is particularly susceptible to the curse of dimensionality . In machine learning, the curse of dimensionality refers to scenarios with a fixed size of training examples but an increasing number of dimensions and range of feature values in each dimension in a high-dimensional feature space.

In k-NN an increasing number of dimensions becomes increasingly problematic because the more dimensions we add, the larger the volume in the hyperspace needs to be to capture a fixed number of neighbors. As the volume grows larger and larger, the "neighbors" become less and less "similar" to the query point as they are now all relatively distant from the query point considering all different dimensions that are included when computing the pairwise distances.

For example, consider a single dimension with unit length (range [0, 1]). Now, if we consider 100 training examples that are uniformly distributed, we expect one training example located at each

0.01^{th} unit along the [0, 1] interval or axis. So, to consider the three nearest neighbors of a query point, we expect to cover $\frac{3}{100}$ of the feature axis. However, if we add a second dimension, the expected interval length that is required to include the same amount of data (3 neighbors) now increases to $0.03^{\frac{1}{2}}$ (we now have a unit rectangle). In other words, instead of requiring $0.03 \times 100\% = 3\%$ of the space to include 3 neighbors in 1D, we now need to consider $0.03^{\frac{1}{2}} \times 100\% = 17.3\%$ of a 2D space to cover the same amount of data points – the density decreases with the number of dimensions. In 10 dimensions, that's now $0.03^{\frac{1}{10}} \times 100\% = 70.4\%$ of the hypervolume we need to consider to include three neighbors on average. You can see that in high dimensions we need to take a large portion of the hypervolume into consideration (assuming a fixed number of training examples) to find k nearest neighbors, and then these so-called "neighbors" may not be particularly "close" to the query point anymore.

2 Linear classification

In the previous section we described the k-Nearest Neighbor (kNN) classifier which labels data points by comparing them to (annotated) data points from the training set. As we saw, k-NN has a number of disadvantages:

- The classifier must remember all of the training data and store it for future comparisons with the test data. This is space inefficient because datasets may easily be gigabytes in size.
- Classifying a test image is expensive since it requires a comparison to all training images.

We are now going to develop a more powerful approach to image classification that we will eventually naturally extend to entire Neural Networks and Convolutional Neural Networks. The approach will have two major components: a score function that maps the raw data to class scores, and a loss function that quantifies the agreement between the predicted scores and the ground truth labels. We will then cast this as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

2.1 Parameterized mapping from images to label scores

The first component of this approach is to define the score function that maps the pixel values of an image to confidence scores for each class. In linear classification, we define the score function that maps the raw image pixels to class scores as a linear mapping:

$$f(\mathbf{x}^{(i)}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$$

where image $\mathbf{x}^{(i)}$ has all of its pixels flattened out to a single column vector and the matrix $\mathbf{W}$ and the vector $\mathbf{b}$ are parameters of the linear model.

2.2 Loss function

In the previous section we defined a function from the pixel values to class scores, which was parameterized by a set of weights $\mathbf{W}$. We want to set the weights so that the predicted class scores are consistent with the ground truth labels in the training data. We are going to measure our

unhappiness with the predictions with a loss function (or sometimes also referred to as the cost function or the objective). Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well. In this class, we will be considering two loss functions:

- Hinge loss (SVM)
- Cross-entropy loss (Softmax classifier)

In this discussion we mostly focus on the cross-entropy loss and have a practice problem on hinge loss.

2.2.1 Softmax classifier

In the softmax classifier we view the scores obtained from the linear mapping as unnormalized log probabilities for each class and use a cross-entropy loss that has the form:

$$L_i = -log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

where we are using the notation f_j to mean the j^{th} element of the vector of class scores f and the full loss for the dataset is the mean of L_i over all training examples. The function

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

is called the softmax function: It takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one.

Probabilistic interpretation: Looking at the expression, we see that

$$P(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}; \mathbf{W}) = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$$

can be interpreted as the (normalized) probability assigned to the correct label $\mathbf{y}^{(i)}$ given the image $\mathbf{x}^{(i)}$ and parameterized by $\mathbf{W}$. To see this, remember that the Softmax classifier interprets the scores inside the output vector f as the unnormalized log probabilities. Exponentiating these quantities therefore gives the (unnormalized) probabilities, and the division performs the normalization so that the probabilities sum to one. In the probabilistic interpretation, we are therefore minimizing the negative log likelihood of the correct class, which can be interpreted as performing Maximum Likelihood Estimation (MLE).

3 Practice problems

1. **Robustness of k-NN**
   In this question, we will look at how k-NN classifiers can be robust to noise. Suppose we have two labels 0 and 1. We are given a test point x, and its k nearest neighbors $z_1, z_2, \ldots, z_k$,

where z_i is the i^{th} closest neighbor of x (so z_1 is the closest neighbor, z_2 is the second closest neighbor and so on).

Suppose that the probability that the label of z_i is not equal to the label of x is p_i. Also assume that all distances are unique, so the i^{th} closest neighbor of x is unique for all i.

Answer the following questions:

(a) Now, suppose, for this question that $p_1 = 0.1$, and $p_i = 0.2$ for $i > 1$. What is the probability that the 1-nearest neighbor classifier makes a mistake on x?

(b) In the setting of part (a), what is the probability that the 3-nearest neighbor classifier makes a mistake on x?

(c) Based on your calculation, what can you conclude about the relative robustness of 1 and 3-nearest neighbor classifiers in this case?

2. **Picking the size of the neighborhood in a k-NN classifer**
As we have shown in the lecture that the size of the neighborhood (k) is a hyperparameter in the k-NN model. We can use cross validation to pick the optimal k. We plotted the 5-fold cross validation error as a function of k which is shown below

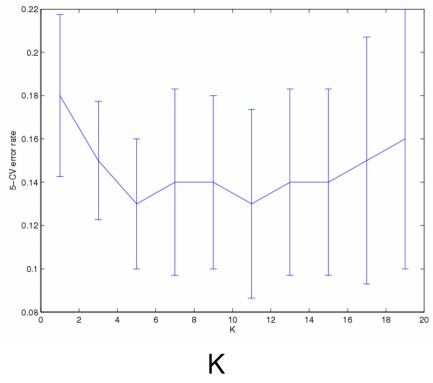

Figure 3: 5-fold cross validation error as a function of k

Answer the following questions:

(a) What is the optimal k? Justify your answer.

(b) Comment on the variance of the k-NN classifer as you vary k.

3. **Regularization through noise addition**
Let's consider the following loss function

$$\tilde{\mathcal{L}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - (x^{(i)} + \delta^{(i)})^T\theta)^2$$

where $x^{(i)} \in \mathbb{R}^d$ is a feature vector corresponding to the i^{th} training sample and $y^{(i)} \in \mathbb{R}$ is the corresponding label. We assume $\delta^{(i)}$ are i.i.d noise vectors with $\delta^{(i)} \sim \mathcal{N}(0, \sigma^2 I)$.

(a) Express the expectation of the loss function over the gaussian noise in the following form:

$$\mathbb{E}_{\delta \sim \mathcal{N}}[\tilde{\mathcal{L}}(\theta)] = \mathcal{L}(\theta) + R(\theta)$$

where R is not a function of the training data.

(b) Suppose $\sigma \longrightarrow 0$, what kind of effect would this have on the model?

(c) Suppose $\sigma \longrightarrow \infty$, what kind of effect would this have on the model?

4. **Derivative of composition of functions**

Let's consider the following composition of functions:

$$\begin{aligned} g(\mathbf{w}_j) &= \mathbf{w}_j^T \mathbf{x} = z_j, \;\; j = 1, \cdots, K \\ f(z_j) &= \frac{e^{z_j}}{\sum_{p=1}^{K} e^{z_p}} = r_j, \;\; j = 1, \cdots, K \\ c(r_j) &= -\log(r_j), \;\; j = 1, \cdots, K \end{aligned}$$

Compute the derivative for the following cases:

(a) $\nabla_{\mathbf{w}_i} c(r_j)$ for $i = j$

(b) $\nabla_{\mathbf{w}_i} c(r_j)$ for $i \neq j$

5. **Hinge Loss with L_1 regularization**

We want to design a model to predict if the received email is spam or not. Since we have only two labels, we can define this as a binary classification problem with Hinge loss as the loss function. The hinge loss for the i^{th} training sample, $(\mathbf{x}^{(i)}, y^{(i)})$, is defined as:

$$\text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) = max(0, 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)),$$

where $\mathbf{w}$ is the weight vector and b is a bias term. Also, we want the model to generalize well with as few parameters as possible. In order to impose this constraint in the optimization problem, we add a L_1 regularization term to the hinge loss. The L_1 regularization term promotes sparsity in the weight vector $\mathbf{w}$ and make the system robust to outliers (such as targeted promotions that user classifies as not spam). Hence, the average loss per training sample is given by:

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{K} \sum_{i=1}^{K} \text{hinge}_{y^{(i)}}(\mathbf{x}^{(i)}) + \lambda \|\mathbf{w}\|_1$$

Find the derivative of the loss function with respect to the weight vector $\mathbf{w}$ i.e. $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b)$.