Q1. Generalization

We consider the following different classifiers for classification of samples in a 2-dimensional feature space.

**PNoBias** Linear perceptron *without* a bias term (features $[x_1 \ x_2]^T$)

**PBias** Linear perceptron with a bias term (features $[1 \ x_1 \ x_2]^T$)

**PQuad** Kernel perceptron with the quadratic kernel function $K(x, z) = (1 + x \cdot z)^2$

**PCutoff** Kernel perceptron with the kernel function $K(x, z) = \max\{0, 0.01 - \|x - z\|_2\}$ ($\|a - b\|_2$ is the Euclidean distance between $a$ and $b$)

**1NN** 1-nearest neighbor classifier

**3NN** 3-nearest neighbor classifier

(a) In each of the plots below you are given points from two classes, shown as filled rectangles and open circles. For each plot, fill in the bubble next to each classifier that will be able to perfectly classify all of the training data (or, if none, mark “None of these will classify the data perfectly”).

Note that when computing the nearest neighbors for a training data point, the training data point will be its own nearest neighbor.

- PNoBias
- PBias
- PQuad
- PCutoff
- 1NN
- 3NN
- None of these will be able to classify the training data perfectly.

3NN misclassifies the bottom-most circle.

- PNoBias
- PBias
- PQuad
- PCutoff
- 3NN
- None of these will be able to classify the training data perfectly.

PNoBias is restricted to separators through the origin.

- PNoBias
- PBias
- PQuad
- PCutoff
- 3NN
- None of these will be able to classify the training data perfectly.

The data are not linearly separable.

- PNoBias
- PBias
- PQuad
- PCutoff
- 3NN
- None of these will be able to classify the training data perfectly.

The decision boundary is complicated and in particular neither linear, nor quadratic. 1NN and PCutoff classify locally.
(b) Suppose you are given the choice between using the normal perceptron algorithm, which directly works with \( \phi(x) \), and the dual (kernelized) perceptron algorithm, which does not explicitly compute \( \phi(x) \) but instead works with the kernel function \( K \). Keeping space and time complexities in consideration, when would you prefer using the kernelized perceptron algorithm over the normal perceptron algorithm.

*Note:* Here \( N \) denotes the total number of training samples and \( d \) is the dimensionality of \( \phi(x) \).

- \( d \gg N \)
- \( d \ll N \)
- Always
- Never

For this question, the rationale was when we use a Kernel function, we typically store a Kernel matrix \( K \) with \( K_{ij} = \phi(x_i) \cdot \phi(x_j) \) where \( x_i \) and \( x_j \) are the \( i^{th} \) and \( j^{th} \) training instances. This results in an \( N \times N \) matrix. If we were to use the transformed \( d \)-dimensional feature representation, we would have to store \( Nd \) values instead of \( N^2 \) values in the Kernel matrix. Thus space-wise, we would prefer kernels when \( d \gg N \).

Looking at time complexity, (at test time), if we use kernels (e.g., the kernelized perceptron) we need to compute \( \sum_{i=1}^{N} \alpha_i y K(x', x_i) \) for a test sample \( x' \). Assuming the kernel function computation takes \( O(1) \) time, we need to do \( N \) such computations. In case of using \( \phi(x) \), we have the precomputed weight vector as \( w = \sum \alpha_i y \phi(x_i) \) which is \( d \)-dimensional and the computation of \( w \cdot \phi(x') \) takes \( d \cdot O(1) \) computations. So again we would prefer kernels if \( d \gg N \).

(c) (i) Suppose you train a classifier and test it on a held-out validation set. It gets 80% classification accuracy on the training set and 20% classification accuracy on the validation set.

From what problem is your model most likely suffering?

- Underfitting
- Overfitting

Fill in the bubble next to any measure of the following which could reasonably be expected to improve your classifier’s performance on the validation set.

- Add extra features
- Remove some features

Briefly justify: Either answer was accepted with justification. Add extra features – adding some really good features could better capture the structure in the data. Remove some features – the model may be using the noise in the abundant feature set to overfit to the training data rather than learning any meaningful underlying structure.

- Collect more training data
- Throw out some training data

More data should yield a more representative sample of the true distribution of the data. Less data is more susceptible to overfitting.

Assuming features are outcome counts (\( k \) is the Laplace smoothing parameter controlling the number of extra times you “pretend” to have seen an outcome in the training data):

- Increase \( k \)
- Decrease \( k \) (assuming \( k > 0 \) currently)

Increasing \( k \) reduces the impact of any one training instance to make the classifier less sensitive to overfitting of rare (= low count) patterns.

Assuming your classifier is a Bayes’ net:

- Add edges
- Remove edges

Removing edges reduces the class of distributions the Bayes’ net can represent. Adding edges introduces more parameters so that the model could further overfit.

(ii) Suppose you train a classifier and test it on a held-out validation set. It gets 30% classification accuracy on the training set and 30% classification accuracy on the validation set.

From what problem is your model most likely suffering?

- Underfitting
- Overfitting

Fill in the bubble next to any measure of the following which could reasonably be expected to improve your classifier’s performance on the validation set.

- Add extra features
- Remove some features

Briefly justify: Under the current feature representation, we are unable to accurately model the training
data for the purpose of the classification task we’re interested in. The classifier may be able to deduce more information about the connections between data points and their classes from additional features, allowing it to better model the data for the classification task. For example, a linear perceptron could not accurately model two classes separated by a circle in a 2-dimensional feature space, but by using quadratic features in a kernel perceptron, we can find a perfect separating hyperplane.

- Collect more training data
- Throw out some training data

More training data can only be a good thing. Marking neither of the bubbles was accepted, too, as given that train and hold-out validation already achieve the same performance, likely the underlying problem is not a lack of training data.
Q2. Clustering

In this question, we will do $k$-means clustering to cluster the points $A, B \ldots F$ (indicated by $\times$’s in the figure on the right) into 2 clusters. The current cluster centers are $P$ and $Q$ (indicated by the $\blacksquare$ in the diagram on the right). Recall that $k$-means requires a distance function. Given 2 points, $A = (A_1, A_2)$ and $B = (B_1, B_2)$, we use the following distance function $d(A, B)$ that you saw from class,

$$d(A, B) = (A_1 - B_1)^2 + (A_2 - B_2)^2$$

(a) **Update assignment step:** Select all points that get assigned to the cluster with center at $P$:

- $A$
- $B$
- $C$
- $D$
- $E$
- $F$
- No point gets assigned to cluster $P$

(b) **Update cluster center step:** What does cluster center $P$ get updated to?

The cluster center gets updated to the point, $P'$ which minimizes, $d(P', B) + d(P', C) + d(P', D)$, which in this case turns out to be the centroid of the points, hence the new cluster center is

$$\left(\frac{-1 - 2 - 1}{3}, \frac{2 + 1 - 2}{3}\right) = \left(\frac{-4}{3}, \frac{1}{3}\right)$$

Changing the distance function: While $k$-means used Euclidean distance in class, we can extend it to other distance functions, where the assignment and update phases still iteratively minimize the total (non-Euclidian) distance. Here, consider the Manhattan distance:

$$d'(A, B) = |A_1 - B_1| + |A_2 - B_2|$$

We again start from the original locations for $P$ and $Q$ as shown in the figure, and do the update assignment step and the update cluster center step using Manhattan distance as the distance function:

(c) **Update assignment step:** Select all points that get assigned to the cluster with center at $P$, under this new distance function $d'(A, B)$.

- $A$
- $B$
- $C$
- $D$
- $E$
- $F$
- No point gets assigned to cluster $P$

(d) **Update cluster center step:** What does cluster center $P$ get updated to, under this new distance function $d'(A, B)$?

The cluster center gets updated to the point, $P'$ which minimizes, $d'(P', A) + d'(P', C) + d'(P', D)$, which in this case turns out to be the point with X-coordinate as the median of the X-coordinate of the points in the cluster and the Y-coordinate as the median of the Y-coordinate of the points in the cluster. Hence the new cluster center is

$$(-1, 0)$$
Q3. Decision Trees

You are given points from 2 classes, shown as +'s and ·'s. For each of the following sets of points,

1. Draw the decision tree of depth at most 2 that can separate the given data completely, by filling in binary predicates (which only involve thresholding of a single variable) in the boxes for the decision trees below. If the data is already separated when you hit a box, simply write the class, and leave the sub-tree hanging from that box empty.

2. Draw the corresponding decision boundaries on the scatter plot, and write the class labels for each of the resulting bins somewhere inside the resulting bins.

If the data can not be separated completely by a depth 2 decision tree, simply cross out the tree template. We solve the first part as an example.