Written Questions

Recall the definition for a halfspace $h_w(x)$ given in lecture:

$$h_w(x) = \begin{cases} 
1 & \text{if } \langle w, x \rangle \geq 0, \\
0 & \text{otherwise}.
\end{cases}$$

Note that halfspaces may employ the use of bias terms. That is, each input has an additional constant '1' attribute, and a corresponding weight for that attribute. Given the bias weight has value $w_{bias}$, this has an effect equivalent to the following:

$$h_w(x) = \begin{cases} 
1 & \text{if } \langle w, x \rangle \geq -w_{bias}, \\
0 & \text{otherwise}.
\end{cases}$$

Problem 1

(16 points)
Define a halfspace for the following functions. Provide a description for how the weights are to be defined so that the classifier has the desired behavior. Weights may be expressed in terms of $d$.

a. Conjunction: Output is 1 if and only if all $d$ attributes are 1.

b. Majority: Output is 1 if and only if more than half of the $d$ attributes are 1.

Solution:

1. Since the conjunction treats every attribute the same, there will be a solution that provides equal weight for each attribute (aside from bias). Note that it is not necessary we apply this constraint, but it helps to simplify the problem. Denote this weight $w$. Our halfspace may be written as follows:

$$h_w(x) = \begin{cases} 
1 & \text{if } w \cdot \sum_{i=1}^{d} x_i \geq -w_{bias}, \\
0 & \text{otherwise}.
\end{cases}$$

The behavior of the conjunction may be restated such that:

$$h_w(x) = \begin{cases} 
1 & \text{if } \sum_{i=1}^{d} x_i = d, \\
0 & \text{otherwise}.
\end{cases}$$

Considering these two different statements of the halfspace together results in the following constraints:

$$w \cdot d \geq -w_{bias}$$
$$w \cdot (d - 1) < -w_{bias}$$

Combined, this may be stated as:

$$\frac{-w_{bias}}{d-1} > w \geq -\frac{w_{bias}}{d}$$

One such set of weights satisfying this is $w = 1, w_{bias} = -d$
2. Again, since the majority treats all attributes the same, you may simply denote a single weight \( w \) for the attributes. The behavior of the majority may be restated such that:

\[
h_w(x) = \begin{cases} 
1 & \text{if } \sum_{i=1}^d x_i \geq \frac{d+1}{2}, \\
0 & \text{otherwise}.
\end{cases}
\]

This generates the constraints:

\[
w \cdot \left\lceil \frac{d+1}{2} \right\rceil \geq -w_{bias}
\]

\[
w \cdot \left( \left\lceil \frac{d+1}{2} \right\rceil - 1 \right) < -w_{bias}
\]

These are both satisfied when \( w \cdot \frac{d+1}{2} = -w_{bias} \). One set of weights generating the desired behavior is \( w = 1, w_{bias} = -\frac{d+1}{2} \).

**Problem 2**

(8 points)
Consider the function \( h_{equiv}(x_1, x_2) \) (where \( x_1, x_2 \in \{0, 1\} \)) defined as

\[
h_{equiv}(x_1, x_2) = \begin{cases} 
1 & \text{if } x_1 = x_2, \\
0 & \text{otherwise}.
\end{cases}
\]

Show that \( h_{equiv} \) cannot be represented as a halfspace. *Hint:* In class, we showed that XOR cannot be represented with a halfspace. You can use a similar argument here.

**Solution:** As we are in a 2-D space, a halfspace may be written more explicitly as:

\[
h_w(x_1, x_2) = \begin{cases} 
1 & \text{if } w_1 \cdot x_1 + w_2 \cdot x_2 \geq w_3, \\
0 & \text{if } w_1 \cdot x_1 + w_2 \cdot x_2 < w_3
\end{cases}
\]

Using the behavior of \( h_{equiv} \), we can generate a set of constraints on \( w \).

\[
h_{equiv}(0, 0) = 1 \implies 0 \geq w_3
\]

\[
h_{equiv}(1, 1) = 1 \implies w_1 + w_2 \geq w_3
\]

\[
h_{equiv}(1, 0) = 0 \implies w_1 < w_3
\]

\[
h_{equiv}(0, 1) = 0 \implies w_2 < w_3
\]

Using the last 3 constraints, we can conclude that \( w_1, w_2 < w_3 < 0 \). As \( w_1, w_2 \) must be more negative than \( w_3 \), it is not also possible that \( w_1 + w_2 > w_3 \). As we cannot possibly satisfy all constraints on \( w \) necessary to mimic the behavior of \( h_{equiv} \), we can conclude that \( h_{equiv} \) cannot be represented as a halfspace.
Problem 3
(15 points)
Consider the set of lines in the plane that pass through the origin. All such lines may be expressed in the form \( w_1 x_1 + w_2 x_2 = 0 \) with \(|w| = 1\). Prove that the distance from a point \( x \) to a line defined by \( w \) is \(|\langle w, x \rangle|\). You should define a distance function and set the derivative to zero to find the minimum distance.

Solution: Assume our given point is \( x(x_1, x_2) \), and an arbitrary point \( y(y_1, y_2) \) on an arbitrary line in the form \( w_1 x_1 + w_2 x_2 = 0 \) with \(|w| = 1\). Thus we have \( w_1 y_1 + w_2 y_2 = 0 \) and \( w_1^2 + w_2^2 = 1 \).

The distance \( d \) between \( (x_1, x_2) \) and \( (y_1, y_2) \) is thus \( d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \). Plugging in \( y_2 = -\frac{w_1 y_1}{w_2} \), we have \( d = \sqrt{x_1^2 + y_1^2 - 2x_1 y_1 + x_2^2 + \frac{w_1^2 y_1^2}{w_2^2} + \frac{2w_1 w_2 y_1 y_2}{w_2} + \frac{2w_2 w_1 y_1 y_2}{w_2}} \).

Take the derivative with respect to \( y_1 \), and set it to zero, yields

\[
\frac{dd}{dy_1} = \frac{1}{2} (x_1^2 + y_1^2 - 2x_1 y_1 + x_2^2 + \frac{w_1^2 y_1^2}{w_2^2} + \frac{2w_1 w_2 y_1 y_2}{w_2} + \frac{2w_2 w_1 y_1 y_2}{w_2}) = 0
\]

\[
2y_1 - 2x_1 + \frac{2w_1^2 y_1}{w_2^2} + \frac{2w_2 w_1 y_1 y_2}{w_2} = 0
\]

\[
(w_2^2 + w_1^2)y_1 = w_2(w_2 x_1 - w_1 x_2)
\]

\[
y_1 = w_2 (w_2 x_1 - w_1 x_2)
\]

\[
y_2 = -\frac{w_1 y_1}{w_2} = w_1 (w_1 x_2 - w_2 x_1)
\]

Also check the second derivative to be positive, so \( y_1 \) and \( y_2 \) give local min. Then plug in \( y_1 \) and \( y_2 \) into \( d \) to minimize \( d \). We will compute \( d^2 \) (to simplify the expressions) and then take the square root:

\[
d^2 = (x_1 - w_2(w_2 x_1 - w_1 x_2))^2 + (x_2 - w_1(w_1 x_2 - w_2 x_1))^2
\]

\[
= (x_1^2 + w_2^2 x_1^2 + w_1^2 w_2^2 x_1^2 - 2w_2^2 x_1^2 + 2w_1 w_2 x_1 x_2 - 2w_1 w_2^2 x_1 x_2)
\]

\[
+ (x_2^2 + w_1^2 x_2^2 + w_1^2 w_2^2 x_2^2 - 2w_1^2 x_2^2 + 2w_1 w_2 x_1 x_2 - 2w_1^2 w_2 x_1 x_2)
\]

\[
= (x_1^2 - 2w_2^2 x_1^2 + 2w_1 w_2 x_1 x_2) + (x_2^2 - 2w_1^2 x_2^2 + 2w_1 w_2 x_1 x_2) + (w_2^2 + w_1^2)w_2^2 x_1^2 + (w_2^2 + w_1^2)w_1^2 x_2^2 - 2w_1 w_2 x_1 x_2(w_2^2 + w_1^2)
\]

\[
= x_1^2 - w_2^2 x_1^2 + x_2^2 - w_1^2 x_2^2 + 2w_1 w_2 x_1 x_2
\]

\[
= x_1^2 (1 - w_2^2) + x_2^2 (1 - w_1^2) + 2w_1 w_2 x_1 x_2
\]

\[
= x_1^2 w_1^2 + x_2^2 w_2^2 + 2w_1 w_2 x_1 x_2
\]

\[
= (w_1 x_1 + w_2 x_2)^2
\]

\[
= |w_1 x_1 + w_2 x_2|^2
\]

\[
= |\langle w, x \rangle|^2
\]

Therefore, \( d = |\langle w, x \rangle| \)
Problem 4

(12 points)
Describe an efficient algorithm for a string kernel comparison. Your algorithm should output the similarity score between two given strings. Your algorithm should use the similarity score defined in lecture. For example, \( \langle v_{\text{fame}}, v_{\text{family}} \rangle = 6 \) and \( \langle v_{\text{fame}}, v_{\text{farm}} \rangle = 4 \).

Compute the runtime of your algorithm, and justify that your algorithm produces the desired comparison. Hint: Consider the vectorization of a string as described in lecture. While it is infinite-dimensional, the number of attributes with non-zero values is finite.

Solution:
Given strings a and b, we are looking for an algorithm that can compute the function \( <v_a, v_b> \) where \( v_s \) is the infinite-dimensional vector derived from string s.

We can make a representation for \( v_s \) in time quadratic in the length of s by looping through all the substrings and adding them to a hash table. Then, we can compute the dot product by looping through all the substrings and summing up the corresponding products. The time is quadratic in the length of the longer string. Here’s working python code:

```python
def makev(s):
    v = {}
    for i in range(0,len(s)):
        for j in range(i+1,len(s)+1):
            if s[i:j] not in v: v[s[i:j]] = 0
            v[s[i:j]] = v[s[i:j]] + 1
    return v

def stringkernel(a,b):
    va = makev(a)
    vb = makev(b)
    val = 0
    for str in va:
        if str in vb: val = val + va[str] * vb[str]
    return val
```

# Tests
print stringkernel('banana','financial')
# prints 15
print stringkernel('implementable','multimillionaire')
# prints 21
print stringkernel('machine','learning')
# prints 6

Proof of Correctness: A substring is defined as a string contained in word that begins at character \( i \) and ends at character \( j \) of of a string with length \( n \), such that \( i \leq j \) and \( 0 \leq i, j \leq n \). We can then see the algorithm for the vectorization clearly follows this exact definition.

Base Case: In the case where a string is empty, the vectorization gives an empty dictionary. In the case where a string is one character, it provides a dictionary with the single character as a substring.

Inductive Step: Let us assume that this works for a string of length \( n \). In the case of a string of length \( n + 1 \), the algorithm increases the length of each loop by one. This means that the secondary loop now encompasses the \( n + 1 \) character for each of the preceding character, so we know each new substring is included.
For the increased outer loop, the only substring considered in the inner loop is only the new character itself. This means that the output now includes the new single character substring. Thus this algorithm holds for calculating all possible substrings.

Now let us consider the algorithm for computing the dot product. Let us assume that the algorithm \textit{stringkernel} does not correctly compute the dot product. This would mean that for some element, without loss of generality, in $v_a$ is not multiplied by the element with the corresponding index in $v_b$. However, this is clearly in contradiction, as every element in $v_a$ is looked up $v_b$ and multiplied by the value if it exists. If a string in $v_b$ is not looked up in $v_a$ then this implies that it does not exist in $v_a$. Thus we have demonstrated that the dot product is correctly determined.

Given that we have shown the correctness of the vectorization and of the dot product, then we have achieved our goal as defined by the similarity score and have thus proven a method for calculating it.