Introduction to Machine Learning (Fall 2022)

Recitation attendance check

Type in your section passcode to get attendance credit (within the first fifteen minutes of your scheduled section).

Passcode: Enter

100.00%

Feature Transformations

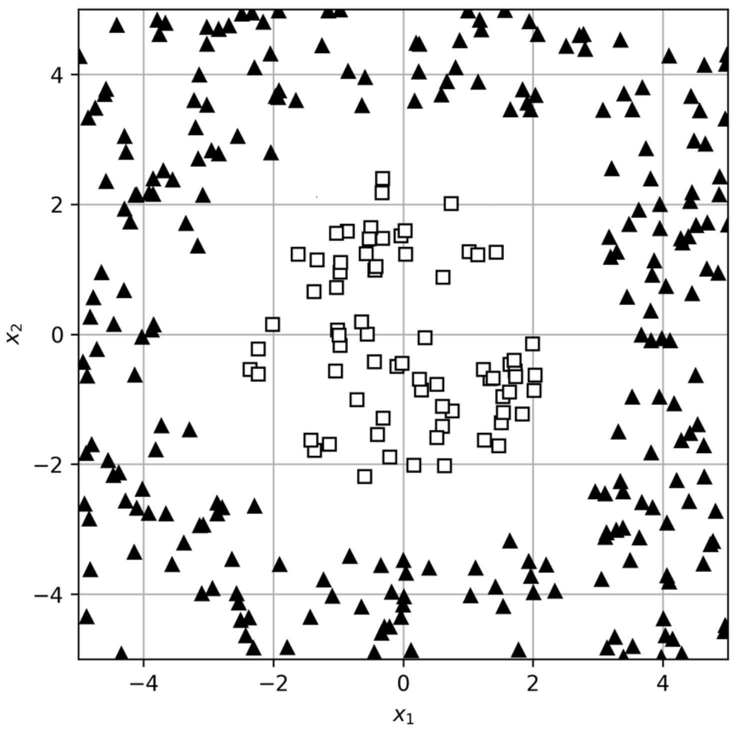
Due: Wednesday, October 12, 2022 at 11:00 PM

In the first part of this assignment, we will consider several general issues in representing features and their impact on classification. In the second part of the assignment, we will experiment with these strategies in the context of realistic data sets. Please make sure your read the course notes on Features for this assignment.

A colab notebook for this homework can be found here.

1) Separating a dataset we couldn't separate last week

Consider the two-dimensional data points shown below. Instead of using the original features x_1 and x_2 we will consider transformations of them that might be able to make the data separable in the new feature space.



Consider the following new features:

- $r_1 = \sqrt{x_1^2 + x_2^2}$
- $r_2 = x_1 - x_2$
- $r_3 = |x_1 + x_2|$

For each of the features r_1, r_2, and r_3, indicate whether it can be used individually to construct a one-dimensional linear classifier that perfectly separates the data.

r_1: separable

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

r_2: not separable

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

r_3: not separable

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

2) Linear Separators and Features

We consider a binary classification problem. We are given a training dataset of n data points, each data point having d features. Each data point has a corresponding binary label.

2.1)

Let’s assume the data set is linearly separable. If one permutes the order of features, will the data set still be linearly separable?

[✓ Yes]

No

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

2.2)

On the contrary, if the data set is not linearly separable, is it possible that rearranging the feature order can make this data set linearly separable?

Yes

[✓ No]

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

2.3)

Let's assume that a problem has a feature vector with three features (x_1,x_2, x_3). For the following datasets, consider whether they are separable in either the feature space (x_1,x_2), (x_3) or (x_1,x_2,x_3)? Consider drawing the labelled data points in 3D.

Consider the dataset $[[0,0,0],[1,1,0],[1,0,1],[0,1,1]]$ with the labels for the four points:$[0,0,1,1]$.

Select all that apply.

- [ ] Separable in (x_1,x_2)
- [✓] Separable in (x_3)
- [✓] Separable in (x_1,x_2,x_3)

100.00%

You have infinitely many submissions remaining.

Solution:

✗ Separable in (x_1,x_2)

✓ Separable in (x_3)

✓ Separable in (x_1,x_2,x_3)

Explanation:

It is not separable in (x_1, x_2) as this is the classic XOR problem we saw in the feature representation notes. It is separable if we include x_3 because all the points with label 0 have $x_3 = 0$ and all the points with label 1 have $x_3 = 1$.

Consider the dataset $[[0, 0, 0], [1, 1, 1], [1, 0, 0], [0, 1, 0]]$ with the labels for the four points:$[0, 0, 1, 1]$.

Select all that apply.

☐ Separable in (x_1,x_2)

☐ Separable in (x_3)

☑ Separable in (x_1,x_2,x_3)

100.00%

You have infinitely many submissions remaining.

Solution:

✗ Separable in (x_1,x_2)

✗ Separable in (x_3)

✓ Separable in (x_1,x_2,x_3)

Explanation:

It is not separable in (x_1, x_2) as this is the classic XOR problem we saw in the feature representation notes. It is not separable in (x_3) because points with the same x_3 value have different labels. It is separable in (x_1, x_2, x_3) with $\theta = [2, 2, -4], \theta_0 = -1$. It's also somewhat possible to mentally visualize this dataset and see the separability directly, since all points lie on the unit cube.

Consider the dataset $[[0, 0, 1], [0, 1, 0], [1, 1, 1], [1, 0, 0]]$ with the labels for the four points:$[0, 0, 1, 1]$.

Select all that apply.

☑ Separable in (x_1,x_2)

☐ Separable in (x_3)

☑ Separable in (x_1,x_2,x_3)

Submit View Answer 100.00%

You have infinitely many submissions remaining.

Consider the dataset $[[1, 0, 0], [1, 1, 0], [0, 1, 1], [0, 0, 1]]$ with the labels for the four points:$[0, 0, 1, 1]$.

Select all that apply.

- [✓ Separable in (x_1,x_2)]
- [✓ Separable in (x_3)]
- [✓ Separable in (x_1,x_2,x_3)]

100.00%

You have infinitely many submissions remaining.

Solution:

✓ Separable in (x_1,x_2)

✓ Separable in (x_3)

✓ Separable in (x_1,x_2,x_3)

Explanation:

Any feature space with x_1 in it will be separable because all points with $x_1 = 1$ have label 0 and all points with $x_1 = 0$ have label 1. Any feature space with x_3 in it will be separable because all points with $x_3 = 0$ have label 0 and all points with $x_3 = 1$ have label 1.

2.4)

Let's consider a new data set, where each data point is described using a feature vector x = (x_1,x_2). Alex claims that for some non-separable problems, it is possible to use a **linear** transformation of the features and make the problem separable. The new feature vector is $\phi(x) = Ax$, where A is a 2×2 matrix.

Is Alex correct?

Yes [✓ No]

Submit View Answer 100.00%

You have infinitely many submissions remaining.

2.5)

On the other hand, Alex also claims that if the problem is separable, it is possible to construct a linear transformation of the feature space to make the problem not separable.

Provide a 2×2 matrix A such that using the feature vector $\phi(x) = Ax$ would make the data not linearly separable. Provide your answer as a list, e.g. `[[a, b], [c, d]]`, or enter `None` if it is not possible.

[[1,2],[1,2]]

100.00%

You have infinitely many submissions remaining.

Solution: `[[0, 0], [0, 0]]`

Explanation:

Zero matrix will collapse everything to a point.

2.6)

Drew has obtained a new dataset. Each data point in this dataset is expressed by a feature vector (x_1, x_2). Upon closer inspection, data points with the positive label have the following relationship between their features: $x_2 = x_1^2 + 2$. For points with the negative label, we observe that $x_2 = x_1^2 - 1$. Which of the following transformations of the original feature space make sure that any data set satisfying these relationships between x_1 and x_2 are linearly separable.

Select feature transformations that make the problem linearly separable

- [✓] $[x_1^2 + x_2, x_2]$
- [✓] $[-x_1^2 + x_2]$
- [ ] $[x_1^2, x_2^2]$
- [✓] $[x_1, x_1^2 - x_2]$
- [ ] $[x_1^2 - 1, x_1^2 + 2]$
- [✓] $[x_1, x_2, x_1 - x_1^2]$

100.00%

You have infinitely many submissions remaining.

Solution:

✓ $[x_1^2 + x_2, x_2]$

✓ [-$x_1^2 + x_2$]

✗ [x_1^2, x_2^2]

✓ [x_1,$x_1^2 - x_2$]

✗ [$x_1^2 - 1$, $x_1^2 + 2$]

✓ [x_1, x_2, $x_1 - x_1^2$]

Explanation:

Because $x_2 = x_1^2 + 2$ if the data point has a positive label, and $x_2 = x_1^2 - 1$ if the data point has a negative label, then $x_2 - x_1^2 = 2$ for positive points and $x_2 - x_1^2 = -1$ for negative points. So, if $x_2 - x_1^2$, or any linear combination of it, is included as a feature, then the data will be linearly separable. If $x_2 - x_1^2$ is a feature, we can separate the data via a separator with the equation $1 \cdot (x_2 - x_1^2) = 0$. The features in the two incorrect options don't give us enough information to tell which of these equations (the equation the positive points satisfy, and the equation the negative points satisfy) hold. For all the correct options, you can get $x_2 - x_1^2$ from some linear combination of the features:

For the first choice: We can take any linear combination of the two features, $x_1^2 + x_2, x_2$. Consider $2 * (x_2) - 1 * (x_1^2 + x_2) = x_2 - x_1^2$ as desired.

For the second choice: The second feature is already in the form $x_2 - x_1^2$.

For the fourth choice: The linear combination to obtain the desired $x_2 - x_1^2$ is simply -1*(second feature).

For the sixth choice: We can take any linear combination of the three features, $x_1, x_2, x_1 - x_1^2$. Consider $-1 * (x_1) + 1 * (x_2) + 1 * (x_1 - x_1^2) = x_2 - x_1^2$ as desired.

You can also think about this graphically: plot all of the possible positive labeled and negative labeled points given the feature set and see if it is linearly separable. For option 3, we can place x_1^2 on the x-axis and x_2^2 on the y-axis. For positive labels, we have points described by $(x_1^2, (x_1^2 + 2)^2)$, which when plotted is the right half of plotting $y = (x + 2)^2$. Similarly, for negative labels, we get the right half of the plot of $y = (x - 1)^2$. There is no linear classifier that separates the two curves described by these two plots. A similar process can be done for option 5.

3) Encoding Discrete Values

Some data sets have features that take on discrete values drawn from a set. Examples might be:

- which section of a class a student is in (1, 2, 3, 4)
- manufacturer of a cell phone (Samsung, Xiaomi, Sony, Apple, LG, Nokia)
- which laboratory performed a particular medical test

Sometimes they already have an obvious encoding into integers; other times, they don't but it's easy to make one (e.g., Samsung = 1, Xiaomi = 2, Sony = 3, Apple = 4, LG = 5, Nokia = 6)

3.1)

Let's consider the case of the cell phones, using the encoding above, and imagine there is some prediction problem, such as predicting whether the phone will last three years, for which we have the data set:

```
data =   [[2, 3,  4,  5]]
labels = [[1, 1, -1, -1]]
```

There is a binary linear classifer (θ, θ_0) that perfectly classifies this data. θ_0 is 7 and θ is an integer. What is θ? Provide your answer as an **integer**. -2

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

3.2)

What prediction would we make about other phone types based on this classifier?

Enter a Python list with two labels (1 or -1), the first one for a Samsung phone and the second for a Nokia phone: [1, -1]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

3.3)

Are these predictions meaningful given the training data we used? No

100.00%

You have infinitely many submissions remaining.

Solution: No

Explanation:

It is based on the completely arbitrary ordering of the phones as integers

3.4)

It is common to encode a feature which takes on a value from a set of discrete values, not as a single multi-valued feature, but using a *one hot* encoding.

Here, assume you have a feature f which can take on any value from the set $\{1, 2, \ldots, k\}$. If f takes on value i, then we represent it as a vector of length k of all 0, except for a $+1$ at the ith coordinate.

Write a function `one_hot` that takes as input x, a single feature value (between 1 and k), and k, the total possible number of values this feature can take on, and transform it to a numpy column vector of k binary features using a one-hot encoding (remember vectors have zero-based indexing).

For example, `one_hot(3,7)` should return a column vector of length 7 with the entry at index 2 taking value 1 (indices start at 0) and other entries taking value 0.

```
import numpy as np

def one_hot(x, k):
    temp = [0]*k
    temp[x-1] = 1
    return np.array([temp]).T

```

Run Code Submit View Answer 100.00%

You have infinitely many submissions remaining.

3.5)

What happens if we use one-hot encoding on the phone data set from part 3.1 above, and put it into our binary linear classifier?

3.5.1)

Assuming the same training data as for part 3.1 but using a one-hot encoding with 6 dimensions, provide a parameter vector θ such that when offset θ_0 is 0, it would cause a logistic regression classifier to classify the training data correctly.

Enter a Python list with six floats θ: [0,1,1,-1,-1,0]

Check Formatting Submit View Answer 100.00%

You have infinitely many submissions remaining.

3.5.2)

Assuming the first and last entries of θ are equal to 0 due to regularization, and that θ_0 is 0, what would be the (continuous-valued) output of the logistic regression hypothesis for the Samsung and Nokia data points?

Enter a Python list with two numbers, the first one for a Samsung phone and the second for a Nokia phone:

[1/2, 1/2]

100.00%

You have infinitely many submissions remaining.

Solution: `[0.5, 0.5]`

Explanation:

The distances for both of the points, which is $\theta^T x + \theta_0$, is 0. Thus the classification, $\sigma(\theta^T x + \theta_0)$, is 0.5.

3.5.3)

Mark all that are true about this hypothesis:

- ☐ It is bad because it doesn’t make a definitive classification of the Samsung and Nokia phones.
- ☑ It is good because it doesn’t make a definitive classification of the Samsung and Nokia phones.
- ☑ The training data did not provide enough information to make a definitive classification of Samsung and Nokia phones.
- ☑ With $\theta = (-1000, 1, 1, -1, -1, 1000)$, we would have had 0 training error.

100.00%

You have infinitely many submissions remaining.

Solution:

✗ It is bad because it doesn’t make a definitive classification of the Samsung and Nokia phones.

✓ It is good because it doesn’t make a definitive classification of the Samsung and Nokia phones.

✓ The training data did not provide enough information to make a definitive classification of Samsung and Nokia phones.

✓ With $\theta = (-1000, 1, 1, -1, -1, 1000)$, we would have had 0 training error.

Explanation:

Only knowing that a phone is Samsung or Nokia, with no other information about it, there is no reason for us to definitely classify it as +1 or -1 when we have no data about them; definitively classifying them would be an arbitrary choice, which we don't want to do.

3.6)

Now, what if we have this dataset:

```
data =   [[1, 2, 3, 4, 5, 6]]
labels = [[1, 1, -1, -1, 1, 1]]
```

Is it linearly separable in the original encoding? No

Submit View Answer 100.00%

You have infinitely many submissions remaining.

3.7)

Is it linearly separable in the one-hot encoding, assuming that $\theta_0 = 0$? If so, provide a valid θ, otherwise submit `None`.

Enter a Python list with six floats θ, or `None`: [1,1,-1,-1,1,1]

100.00%

You have infinitely many submissions remaining.

Solution: `[1, 1, -1, -1, 1, 1]`

Explanation:

Since we use a one-hot encoding on a single variable, our data is clearly separable.

3.8)

Enter an assignment of data values to labels (with distinct data points) that is not linearly separable using the one-hot encoding, or enter `None` if no such assignment exists.

Enter a Python list with six tuples `(value, label)`, or `None`: None

100.00%

You have infinitely many submissions remaining.

Solution: None

Explanation:

One-hot encoding leads to a dimension for each different data point. Therefore, by assigning different weights to each dimension, we can always construct a separator to separate the data. Therefore, it is always possible to find a solution that separates this data.

4) Factoring in Feature Representation

Consider a situation in which you have presents that come in boxes of different types:

- small, medium, large, and
- red, blue, green.

and you want to predict whether you'll like what's in them, based on what they look like. You could encode the input using:

- **Encoding A**: a one-hot vector with 9 possible classes (one for each combination of size and color), or
- **Encoding B**: two independent one-hot vectors, one with three classes for size and one with three classes for color.

4.1)

What is the dimension of encoding B? Enter your answer as a single integer, e.g. `4`.

6

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

4.2)

For a dataset with distinct points, is there any possible classification of objects in this space that cannot be linearly separated using **encoding A**? If so, provide such a classification; if not, write `None`.

Specify an example as a list of tuples of inputs and outputs. Inputs can be specified like 'sr' (for a small, red present), 'mb' (for a medium, blue present), etc.; outputs are 0 or 1. For example, `[('sr', 1), ('mb', 0), ('lg', 1)]`.

None

100.00%

You have infinitely many submissions remaining.

Solution: None

Explanation:

One-hot encoding with encoding A leads to a dimension for each different data point. Therefore, by assigning different weights to each dimension, we can always construct a separator to separate the data. Therefore, it is always possible to find a solution that separates this data.

4.3)

For a dataset with distinct points, is there any possible classification of objects in this space that cannot be linearly separated using **encoding B**? If so, provide such a classification; if not, write `None`.

Specify an example as a list of tuples of inputs and outputs. Inputs can be specified like 'sr' (for a small, red present), 'mb' (for a medium, blue present), etc.; outputs are 0 or 1. For example, `[('sr', 1), ('mb', 0), ('lg', 1)]`.

[('sr', 0), ('mr', 1), ('lr', 0),('sb', 1), ('mb', 0), ('lb', 0)]

100.00%

You have infinitely many submissions remaining.

Solution: `[('sr', 1), ('lg', 1), ('sg', 0), ('lr', 0)]`

Explanation:

Consider each datapoint to be [is_red, is_green, is_blue, is_small, is_medium, is_large], where each dimension holds an indicator in {0, 1} for that characteristic. Then the equation of our separator would be $\theta^T x + \theta_0 = 0$, with $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$. If a separator existed for this dataset, the following four inequalities would be satisfied:

$$\theta_1 + \theta_4 + \theta_0 > 0$$

$$\theta_2 + \theta_6 + \theta_0 > 0$$

$$\theta_2 + \theta_4 + \theta_0 < 0$$

$$\theta_1 + \theta_6 + \theta_0 < 0$$

The first and third inequalities imply $\theta_1 > \theta_2$, and the second and fourth imply $\theta_2 > \theta_1$. This is impossible, therefore a separator for this data cannot exist.

4.4)

Imagine you have received training examples `[('sr', 1), ('sg', 1), ('lr', 0), ('lg', 0)]`. What prediction would a regularized LLC likely make on example 'sb' with **encoding A**? a low-confidence guess

100.00%

You have infinitely many submissions remaining.

Solution: a low-confidence guess

Explanation:

One-hot encoding with encoding A leads to a dimension for each different data point. Since we have not seen the datapoint corresponding to 'sb' in our dataset, with regularization, the weight assigned to that dimension will be zero. So we will classify 'sb' as

$$\sigma(0) = 0.5$$

.

4.5)

Imagine you have received training examples `[('sr', 1), ('sg', 1), ('lr', 0), ('lg', 0)]`. What prediction would a regularized LLC likely make on example 'sb' with **encoding B**? [1, confidently]

100.00%

You have infinitely many submissions remaining.

Solution: 1, confidently

Explanation:

Consider each datapoint to be [is_red, is_green, is_blue, is_small, is_medium, is_large], where each dimension holds an indicator in {0, 1} for that characteristic. Then the equation of our separator would be $\theta^T x + \theta_0 = 0$, with $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$.

A separator for this dataset would satisfy the following inequalities:

$$\theta_1 + \theta_4 + \theta_0 > 0$$

$$\theta_2 + \theta_4 + \theta_0 > 0$$

$$\theta_1 + \theta_6 + \theta_0 < 0$$

$$\theta_2 + \theta_6 + \theta_0 < 0$$

If we wanted to classify a point "sb", we would calculate $\theta_3 + \theta_4 + \theta_0$ and pass it through the sigmoid function.

Note that because our dataset does not have any data with blue or medium, and we are regularizing, θ_3 and θ_5 will be zero. Also, notice that θ_1, θ_2, and θ_0 appear in all four inequalities. Because of this, and that we are regularizing, θ_1, θ_2, and θ_0 should be close to zero, making θ_4 and θ_6 determine the classification. Because the classification for 'sr' and 'sg' is determined by θ_4, and LLC will aim to make the probability assigned to these points close to 1, because 'sb' is also determined by θ_4, its assigned probability will also be close to 1.

4.6)

Which encoding offers more ability to generalize? Encoding B

100.00%

You have infinitely many submissions remaining.

Solution: Encoding B

Explanation:

Generalization refers to the ability to react/adapt/respond to unseen data. In this case: how "accurately" we can classify data not seen in the training set. Encoding A, because it assigns a weight to each data point, with regularization, the weights assigned to data points not seen in the training set will be zero. So any new data point will be assigned probability 0.5 for each class. Encoding B does not do this: an object's color and an object's size will separately contribute to the classification. A new data point may share the same color or the same size as a data point seen in the training set, and that will make it possible to classify the point accurately. Note that we may still do a bad job at generalizing if the true data generating process is not linear.

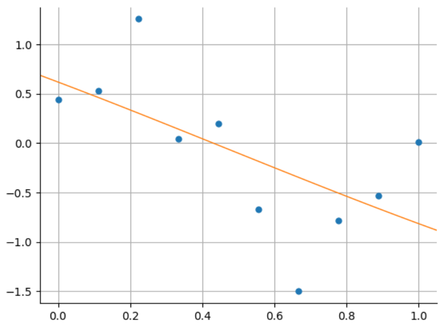
5) Regression with Radial Basis Functions

Read about radial basis functions in the lecture notes! They have a parameter β that governs how "broad" the influence of each point is.

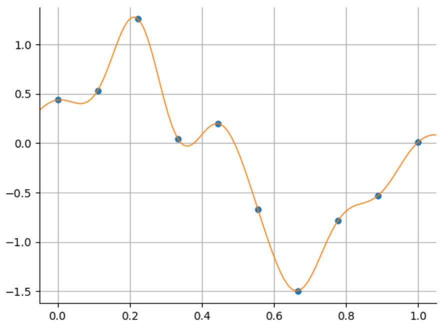
We did linear regression using a feature transformation based on the data points in the plots below.

We used β values of 0.01, 0.1, 1, 10, 100, and 1000. These values correspond to the plots in some order.

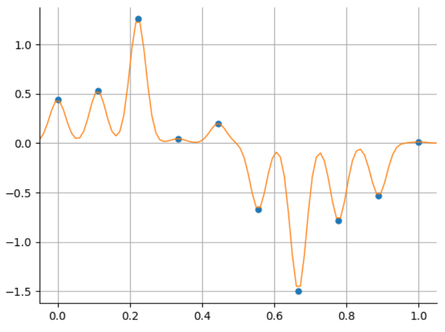
(a)



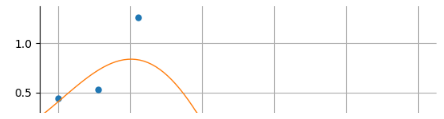
(b)



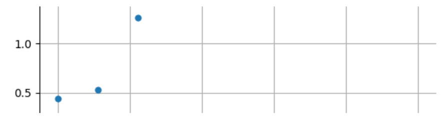
(c)



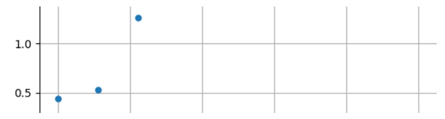
(d)

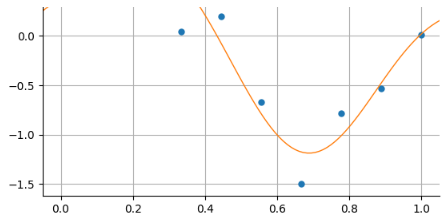


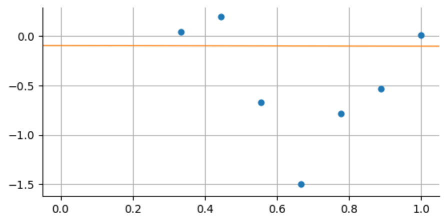
(e)

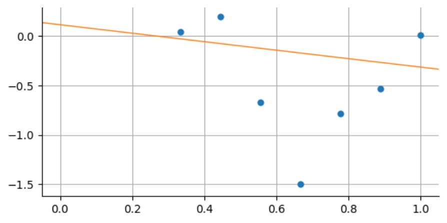


(f)

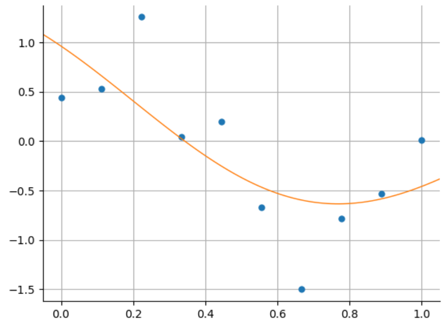








(g)



Which plots correspond to the β values 0.001, 0.01, 0.1, 1, 10, 100, and 1000?

Provide a Python list of seven strings, e.g. `["a", "b", "c", "d", "e", "f", "g"]`, in the order 0.001, 0.01, 0.1, 1, 10, 100, 1000.

["e", "f", "a", "g", "d", "b", "c"]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6) Polynomial Features

One systematic way of generating non-linear transformations of your input features is to consider the polynomials of increasing order. Given a feature vector $x = [x_1, x_2, ..., x_d]^T$, we can map it into a new feature vector that contains all the monomials in a polynomial of order d. For example, for $x = [x_1, x_2]^T$ and order 2, we get

$$\phi(x) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

and for order 3, we get

$$\phi(x) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2, x_1^2 x_2, x_1 x_2^2, x_1^3, x_2^3]^T.$$

In the Colab (linked on top), we have defined `make_polynomial_feature_fun` that, given the order, returns a feature transformation function (analogous to ϕ in the description). You should use it in doing this problem, and you can find it by examining the `hw05_tests.py` file in the Colab after clicking on the folder icon in the left-hand-side.

6.1)

Enter a list of 6 integers indicating the number of polynomial features of degrees [1, 10, 20, 30, 40, 50] for a 2-dimensional feature vector: [3, 66, 231, 496, 861, 1326]

100.00%

You have infinitely many submissions remaining.

Solution: `[3, 66, 231, 496, 861, 1326]`

Explanation:

In the code, we provide `make_polynomial_feature_fun`, which returns a function that computes the d^{th} order polynomial features of your 2-dimensional input. Let f_d be the output of `make_polynomial_feature_fun(d)`. f_d takes as input, a $2 \times n$ array of data and returns a $k \times n$ array, where k is the number of features. So you may run `make_polynomial_feature_fun` for each of the desired dimensions and report the resultant k for each d.

6.2) Polynomials for Classification

Consider this data-set of four points in two-dimensional space:

```
data = ([[1, 1, 2, 2],
         [1, 2, 1, 2]])
labels = [[-1, 1, 1, -1]]
```

It is standardly called the "exclusive-or" or "xor" problem. These points are not linearly separable, and you could interpret each point as being a pair of truth values, with their label being the XOR of the values.

In the Colab we have defined 4 sample data sets, (1) `super_simple_separable_through_origin`, (2) `super_simple_separable`, (3) `xor`, and (4) `xor_more`. Run the code we have provided (`test_with_features`) for various orders of polynomial features and enter below the smallest n such that an order-n polynomial feature set can linearly separate the data. We will use the `sklearn.linear_model.SGDClassifier` with `loss=log` so we can watch the model optimize. You can find the code to use by clicking on the folder icon on the left-hand side of the Colab page and double-clicking the `hw05_tests.py` file.

The decision boundaries are displayed when the code runs; it's instructive to watch them to see the range of separators that these non-linear transformations produce. Note that the separators are drawn by evaluating the feature transformations on a grid of points in the feature space and using the separator to classify them.

Enter a Python list of integers indicating the smallest polynomial order for which a separator exists for each of the four datasets in the Colab (in order): [1, 1, 2, 3]

Check Formatting Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6.3) Polynomials for regression

In least squares regression problems, we assume that our objective function $J(\theta, \theta_0)$ comes without a regularization term; in other words,

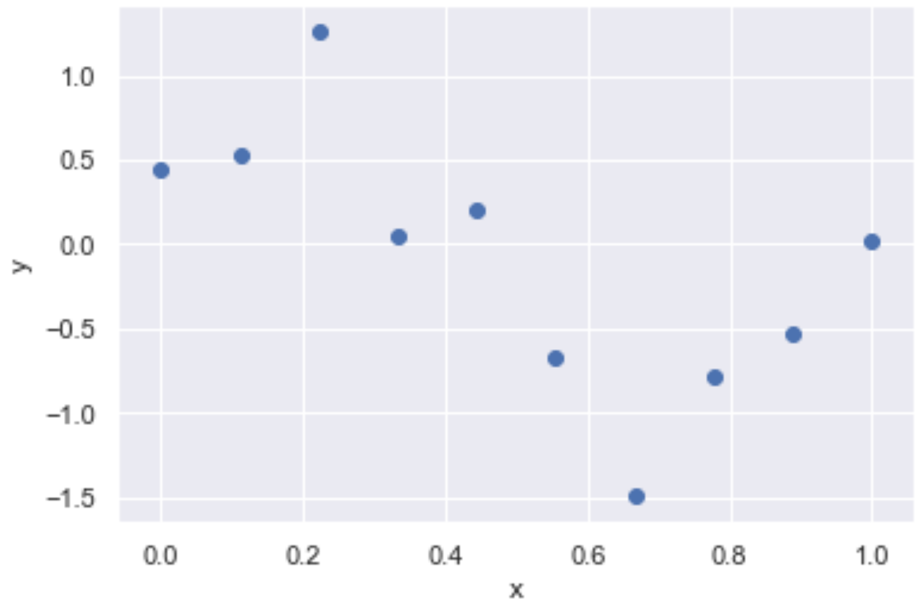
$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0) \ .$$

Recall from the lecture and notes that it is possible to solve a least-squares regression problem directly via the matrix algebra expression for the parameter vector θ in terms of the input data X and desired output vector Y. In this section, we will explore the θ, θ_0 solutions that this *analytic* strategy produces.

We will be computing solutions by transforming our one-dimensional input features with a polynomial basis. Specifically, we will be transforming our single input feature, which we will call x, into the vector $\phi(x) = (x^1, ..., x^k)$ if we are using a k-th order basis. This means that solutions to our regression problem will take the form

$$\theta^T\phi(x) + \theta_0 = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k.$$

We will be using polynomial features to perform regression on the following dataset.



To answer the next set of problems, you are given the function `analytic_regress` to experiment with, which:

- Takes as input `order`, which is the order of the polynomial basis.
- Outputs θ and θ_0 that minimizes the regression objective $J(\theta, \theta_0)$ on the above dataset.
- Finds θ and θ_0 **analytically**.

In the codebox below, experiment with `analytic_regress` and try different values of order k to examine the effects of the order of the polynomial basis on the resulting solution. Based on your observations, answer the questions below the codebox.

```
def run():
    return analytic_regress(order=10)

```

Run Code Submit **100.00%**

You have infinitely many submissions remaining.

6.3.1)

Consider the solution with the 0th-order basis (the regression polynomial described by only θ_0). What will be the shape of the solution?

- [ ] Straight line of best fit with slope m
- [✓] Horizontal line
- [ ] Vertical line
- [ ] Parabola

Submit View Answer **100.00%**

You have infinitely many submissions remaining.

6.3.2)

What happens if the polynomial order gets too big, e.g. what happens if the polynomial order becomes equal to $n-1$, where n is the number of data points.

- [✓] The model overfits to the training data
- [ ] We would expect a new point to be close to the found regression fit
- [✓] The regression may not apply generally to new data points
- [ ] Higher orders are better since they have lower error on the training set

100.00%

You have infinitely many submissions remaining.

Solution:

✓ The model overfits to the training data

✗ We would expect a new point to be close to the found regression fit

✓ The regression may not apply generally to new data points

✗ Higher orders are better since they have lower error on the training set

Explanation:

Regressions of higher order tend to run into overfitting. It may fit the training data perfectly, but you would not expect it to apply generally to new data.

6.3.3)

How do the magnitudes of θ and θ_0 change as the polynomial orders increase? Refer to the values of θ as shown in the results. Why do you think this is the case?

[✓] θ and θ_0 increase in magnitude at higher orders

[ ] θ and θ_0 decrease in magnitude at higher orders

[ ] θ and θ_0 are generally the same magnitude at higher orders

100.00%

You have infinitely many submissions remaining.

Solution:

✓ θ and θ_0 increase in magnitude at higher orders

✗ θ and θ_0 decrease in magnitude at higher orders

✗ θ and θ_0 are generally the same magnitude at higher orders

Explanation:

Higher order functions tend to have larger values of θ and θ_0 of the higher order values (x^3, etc.) and the large shifts in slope that are required to perfectly fit the data points.

6.3.4)

Our code fails if the order is 10 or higher. Why?

- ☐ It takes too long to run
- ☐ It gets stuck in a local optimum
- ☑ The analytic solution is not well-defined

100.00%

You have infinitely many submissions remaining.

Solution:

✗ It takes too long to run

✗ It gets stuck in a local optimum

✓ The analytic solution is not well-defined

Explanation:

Higher orders than your number of data points have no exact solution, so it may not be possible to invert the matrix to obtain the analytical solution.

6.4) There's more than one way to regularize: Polynomial Order vs. Lambda

Recall the *ridge regression* objective:

$$J_{ridge}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n} L_s(x^{(i)}, y^{(i)}, \theta, \theta_0) + \lambda\|\theta\|^2.$$

The procedure `regularized_analytic_regress(order, lam)` performs ridge regression after the inputs have been transformed using polynomial features of order `order` with regularization coefficient `lam`. In the resulting plots, `regularized_analytic_regress` draws the original least-squares solution in a dotted orange line, and draws the regularized version in a solid green.

Experiment with different pairings of order and λ. Look at the detailed results after submitting your code, paying attention to the visualization of the polynomial that was fit using your chosen λ.

```python
def run():
    return regularized_analytic_regress(order=1, lam = 3)

```

Run Code Submit **100.00%**

You have infinitely many submissions remaining.

6.4.1)

What trends do you observe when changing the values of order and λ?

- [✓ Higher λ's look similar to lower order polynomial fits]
- Higher λ's look similar to higher order polynomial fits
- Neither.

100.00%

You have infinitely many submissions remaining.

Solution: Higher λ's look similar to lower order polynomial fits

Explanation:

Adding regularization to a polynomial fit can have the same effect as lowering the order of the polynomial.

7) Molecules!

Refer to lab 5's Molecules! section for a refresher on molecular structures.

7.1) Featurizing molecules to exploit invariances

There are interesting advance machine-learning models, such as graph neural networks, that can represent some kinds of structured inputs with varying dimension and handle some types of invariances. But we can also approach the problem by using our understanding of the underlying problem and designing a fixed-dimensional representation that retains the important properties of the molecules for making the predictions we need. Such featurizations of molecules are often called molecular fingerprints or descriptors. There are many types of molecular descriptors. You can see a list of examples in the popular cheminformatics and machine learning package RDKit.

For this problem, we will use a specific descriptor called the **radial distribution function (RDF)** to describe molecules and predict whether or not a given molecule contains a benzene ring.

In its simplest form, a RDF is a histogram of all the pairwise distances between particles of a given system (in our case, atoms in a molecule). Pairwise distances don't change under translation, rotation, or mirroring. And a histogram of the pairwise distances doesn't depend on putting the atoms in any particular order and has the same dimension no matter how many atoms a particular molecule has.

We will be using a descriptor rdf, which is a vector of length B, with component b defined as:

$$\mathrm{rdf}_b = \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} e^{-100\times(R_b-D_{ij})^2}$$

which is the same one that is implemented in `rdkit.Chem.rdMolDescriptors.CalcRDF`. We can understand this as a kind of "softened" histogram, where the vector R_b specifies the centers of the "bins" of the histogram. Values D_{ij} are the Euclidean distances between atoms i and j. The factor of -100 in the exponent is to avoid numeric underflow (these distances are small!). The factor of $\frac{1}{2}$ probably doesn't make a real difference, but is there to account for double counting of each pair (the order doesn't matter).

We will implement our RDF featurizer in three steps in the questions below.

7.1.1)

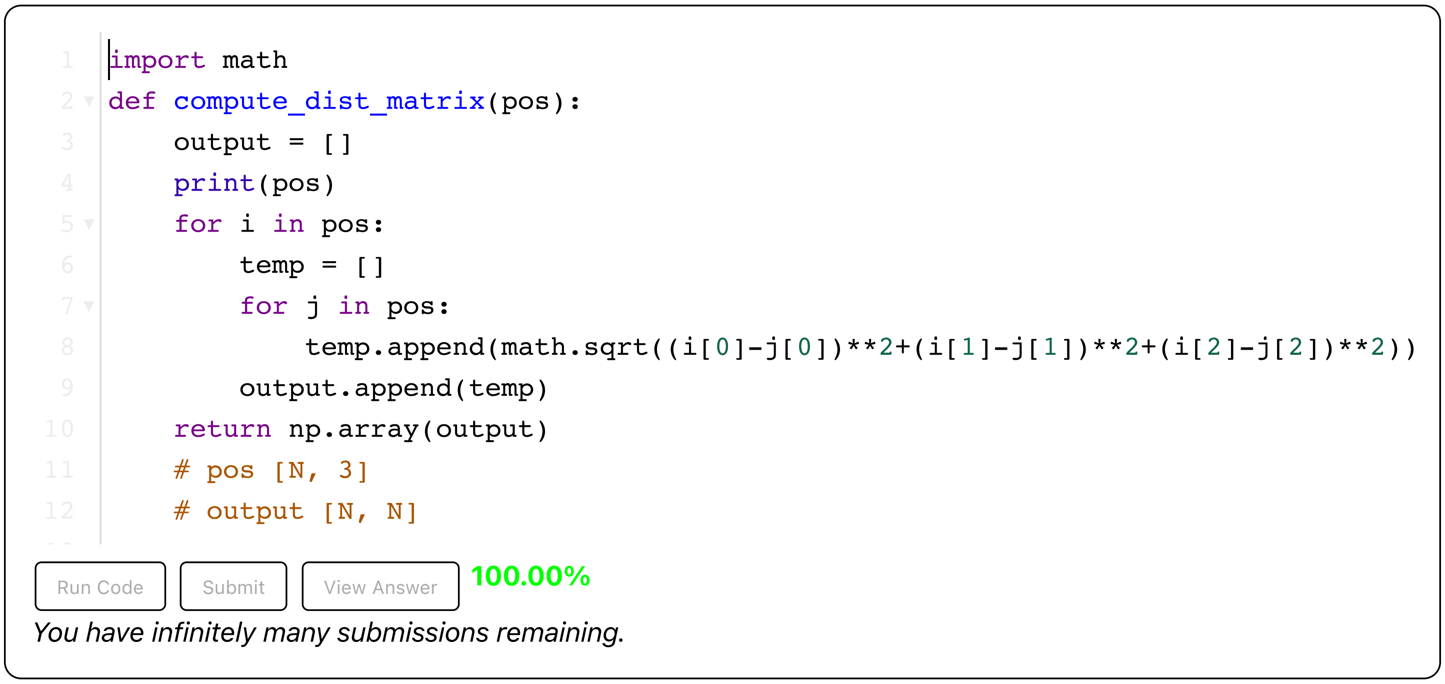
Our first job is to compute the relative distance matrix of a given set of atoms given their 3D coordinates (`pos`) in an array with dimensions `[N, 3]`. The distance matrix can be written component by component as the magnitude of the difference in position of atom i and atom j

$$D_{ij} = ||\vec{\mathrm{pos}}_i - \vec{\mathrm{pos}}_j||.$$

Implement a function to compute the pairwise distance matrix which takes in 3D coordinates `pos` of shape `[N, 3]` and gives back pairwise distances of `[N, N]`.

It's possible to do this straightforwardly with a `for` loop, but this is a good opportunity to practice using the power of numpy!

▶ **Useful notes on numpy (click me!)**



```python
import math
def compute_dist_matrix(pos):
    output = []
    print(pos)
    for i in pos:
        temp = []
        for j in pos:
            temp.append(math.sqrt((i[0]-j[0])**2+(i[1]-j[1])**2+(i[2]-j[2])**2))
        output.append(temp)
    return np.array(output)
    # pos [N, 3]
    # output [N, N]
```

100.00%

You have infinitely many submissions remaining.

7.1.2)

Now, we're going to work toward our "soft" histogram of distances. We will be given a vector, `R` of values that correspond to the centers of our histogram bins. For instance, if our distances could range from 0 to 10, then `R` might be 20 equally spaced values covering that range.

Now that we have our matrix of distances, we're going to figure out the difference between each of these bin centers (components of `R`) and each of our inter-atom distances (components of `D`). The vector R has dimensions `[B]` and D has dimensions `[N, N]`.

Compute as output a tensor of shape `[B, N, N]`, where element `b,i,j` contains the difference between bin-center `b` and the Euclidean distance between atoms `i` and `j`.

```python
def get_R_minus_D(R, D):
    # R [B]
    # D [N, N]
    # output [B, N, N]
    output = []
    for i in R:
        part = []
        for first in D:
            temp = []
            for second in first:
                temp.append(i - second)
            part.append(temp)
```

Run Code Submit View Answer **100.00%**

You have infinitely many submissions remaining.

7.1.3)

Then we will put this all together to compute the RDF formula above. The formula is given again below for your reference. You should use the function `get_R_minus_D`. Remember, you will need to sum over both atom dimensions `[N, N]`.

For each pair of atoms, we increment each histogram bin by $e^{-100\delta^2}$ where δ is the difference between that bin's center and the measured distance between the two atoms. So, bin centers that are near distances that occur frequently in the distance distribution will have high values and bin centers that represent distances that don't occur often will have low values.

$$\mathrm{rdf}_b = \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} e^{-100\times(R_b - D_{ij})^2}$$

```python
import math
def get_rdf(R, D):
    # R [B]
    # D [N, N]
    # output [B]
    distances = get_R_minus_D(R,D)
    output = [0 for i in range(np.shape(R)[0])]
    sum = 0
    for i in range(np.shape(R)[0]):
        for first in range(np.shape(D)[0]):
            for second in range(np.shape(D)[0]):
                output[i] += 1/2*math.e**(-100*distances[i,first,second]**2)
```

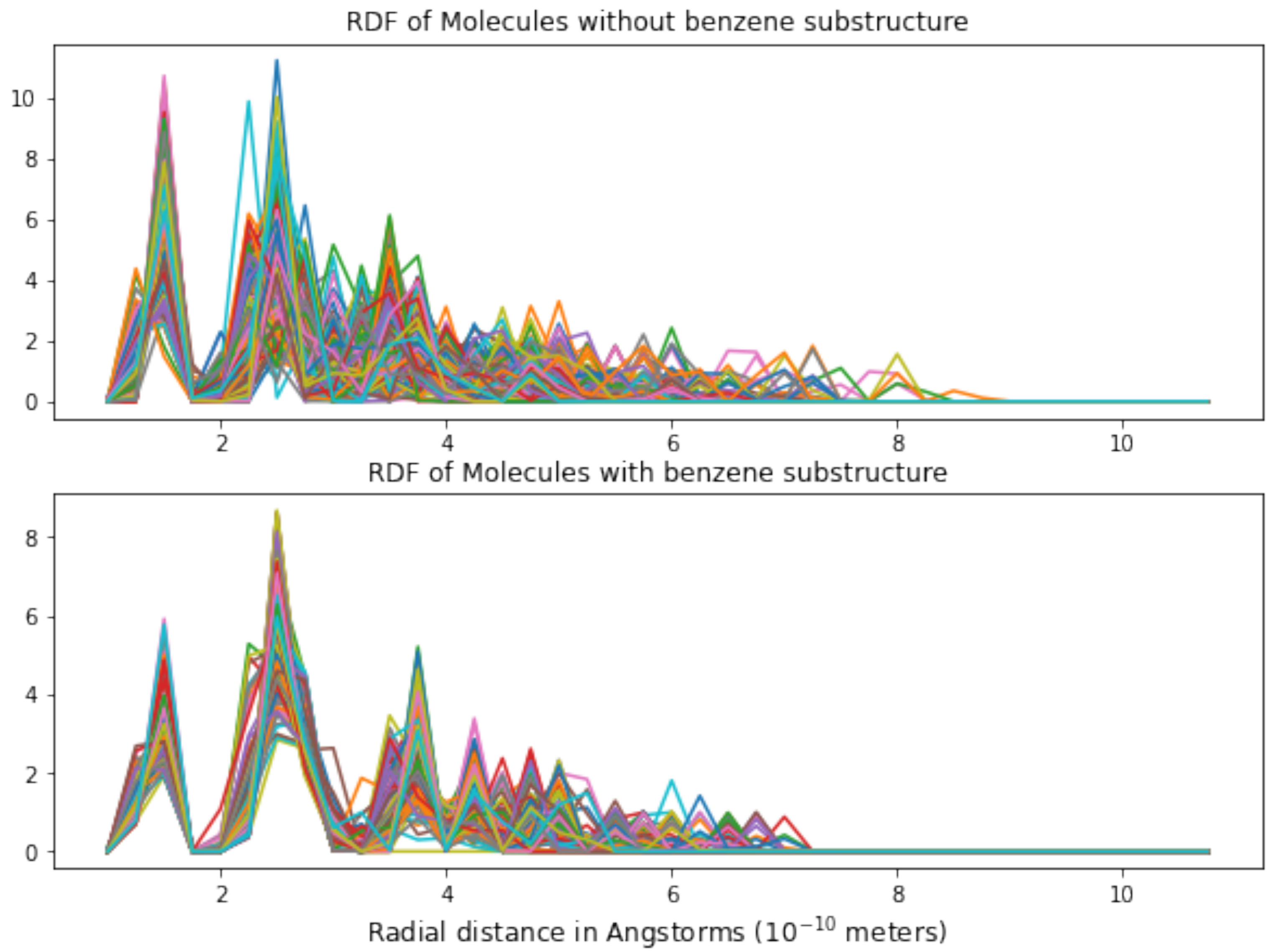
Run Code Submit View Answer **100.00%**

You have infinitely many submissions remaining.

Now we are ready to train a classifier. We have a dataset of 250 molecules with a benzene substructure and 250 molecules without a benzene substructure. We are going to split them into train and test datasets.

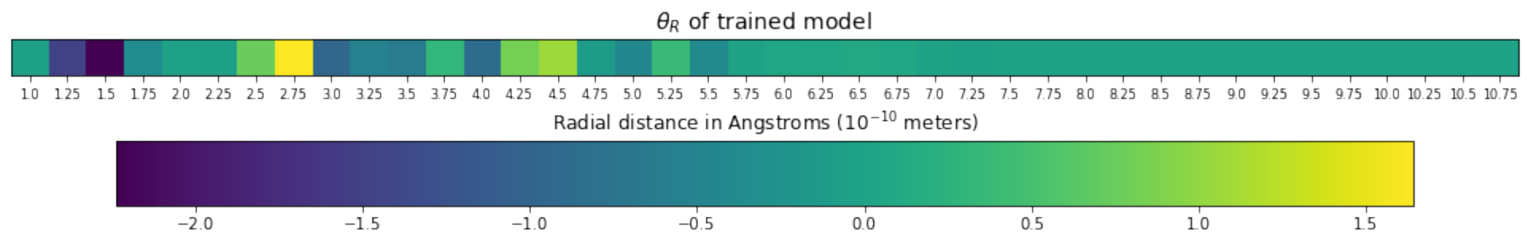
```
data_benzene = np.load('data_benzene.npy', allow_pickle=True)
data_no_benzene = np.load('data_no_benzene.npy', allow_pickle=True)

rs = data_no_benzene[0]['rs']

features0 = np.stack([d['rdf'] for d in data_no_benzene], axis=0)
features1 = np.stack([d['rdf'] for d in data_benzene], axis=0)

n_examples = len(features0)

split = 200

train_rdf = np.concatenate([features0[:split], features1[:split]], axis=0)
test_rdf = np.concatenate([features0[split:], features1[split:]], axis=0)

train_labels = np.array([0] * split + [1] * split)
test_labels = np.array([0] * (n_examples - split) + [1] * (n_examples - split))
```

Before we train, let's plot the RDF data. Each color below corresponds to the rdf representation of one molecule, where R is a list of bin centers ranging from 1 to 11 in increments of 0.25. For example, in all the molecules it looks like there are big peaks, so, many atom pairs, at a distance of about 2.5 angstroms.



```
# Train with logistic regression
...
>> Model train accuracy: 0.993
>> Model test accuracy: 0.99
```

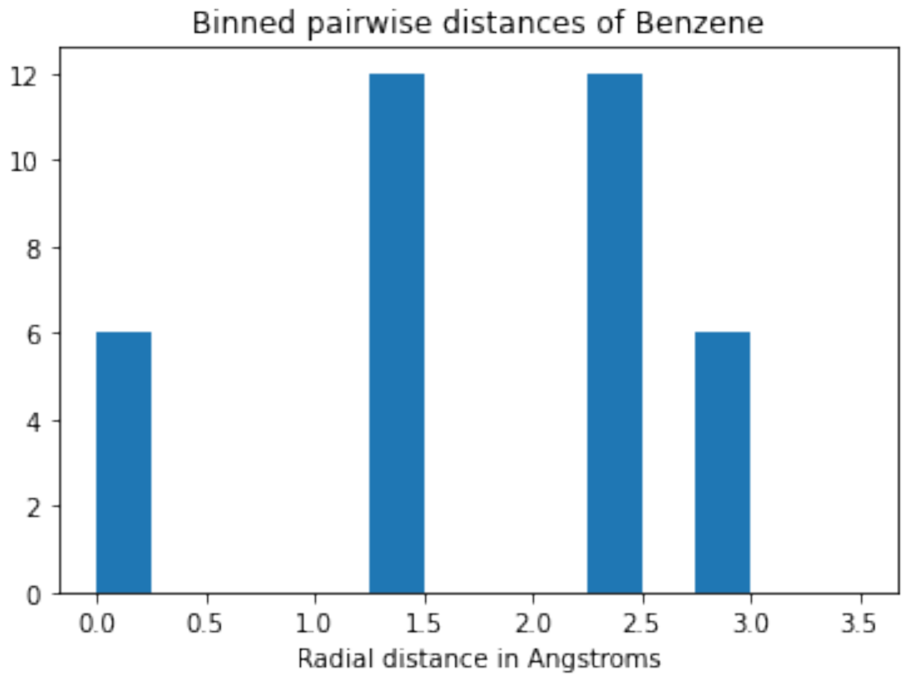
This plot shows the θ values recovered by logistic regression, with dark blue corresponding to large-magnitude negative θ_i values and bright yellow corresponding to large-magnitude positive θ_i values.



Based on this plot, it looks like features 1 through 9 (1.25 - 3.0 Angstroms) are most strongly contributing to the decisions. Let's try re-training with just those features.

```
# Train with logistic regression on features 1-9
...
>> Model train accuracy: 0.988
>> Model test accuracy: 0.96
```

Okay, so clearly we can get fairly good accuracy just by looking up to 3.0 Angstroms away. Parameters θ_i associated with radial distances near 1.5 and 3.0 pushed examples to being classified as having no benzene substructure, whereas radial distances near 2.5 pushed examples to being classified as having a benzene substructure. Why is this? Let's plot a histogram of the pairwise distances of benzene.



7.1.4)

Which of the following are true about each carbon in benzene based on this histogram? You may find it helpful to know that carbon-carbon bonds are typically between 1.3-1.6 Angstroms.

- ☐ They have 2 nearest neighbors approximately 1.4 Angstroms away.
- ☐ They have 2 next nearest neighbors approximately 2.4 Angstroms away.
- ☐ They have 1 next next nearest neighbor approximately 2.8 Angstroms away.
- ☐ Zeros appear on this histogram because pairwise distance matrices have zeros on the diagonals since atoms are next to themselves.

Submit View Answer

You have infinitely many submissions remaining.

7.1.5)

Which of the following are true based on the learned parameters of theta?

- ☐ Carbon-carbon bonds are more prevalent in benzene substructures than other organic molecules with carbon.
- ☐ Having both pairwise distances of approximately 2.4 and 2.8 Angstroms are important for identifying benzene.

Submit View Answer

You have infinitely many submissions remaining.

7.2) Something to make you SMILE

SMILES are the simplified molecular-input line-entry system which allows us to write down molecules as strings. We can take a look at the SMILES of the molecules that were classified improperly by using the online SMILES Explorer. Just copy and paste the following misclassified SMILES strings to that website to generate molecular diagrams. Note, molecular diagrams do NOT contain 3D coordinate information, just bonding information that can be used to suggest 3D positions.

Train Misclassifications

```
Cc1cc(N)nc(F)c1
O=C1CNC2CC1C2O
[NH-]c1cc(O)[nH+]cc1O
c1ccc(C2CC2)cc1
c1ccc2c(c1)COC2


Correct labels for misclassified train examples:  [0 0 0 1 1]
```

Test Misclassificiations

```
N#CC1C(N)C=CC1O
O=C1NCC2CC1C2O
C1=Cc2ccccc2C1
c1ccc(C2CN2)cc1


Correct labels for misclassified test examples:  [0 0 1 1]
```

7.2.1)

Take a look at the misclassified molecules in the SMILES Explorer.

What do you notice about molecules that are misclassified?

- ☐ Examples misclassified as having benzene don't have a 6-member ring.
- ☐ Examples misclassified as having benzene have a 6-member ring that include non-carbon elements.
- ☐ Examples misclassified as not having benzene have many more ~1.4-1.6 carbon-carbon bonds than ~2.8 nearest neighbors.
- ☐ Examples misclassified as not having benzene have distorted benzene structures.

Submit View Answer

You have infinitely many submissions remaining.

7.2.2)

Our featurization does not weigh the radial distance function by atom type and only looks at the geometry of the molecule.

How might we update our featurization to better distinguish benzene rings vs. azine (benzene rings with a carbon replaced with a nitrogen)?

- ☐ Include an additional feature that keeps track of the total number of bonds in the molecule.
- ☐ Include an additional one-hot encoded feature defining which atoms are present in the molecule.

Submit View Answer

You have infinitely many submissions remaining.