Lab ML For Data Science

Summer Semester 2023

14.07.2023

- Joslin Thomas (5566633)
- Manasi Acharya (5580192)
- Namrata De (5580793)

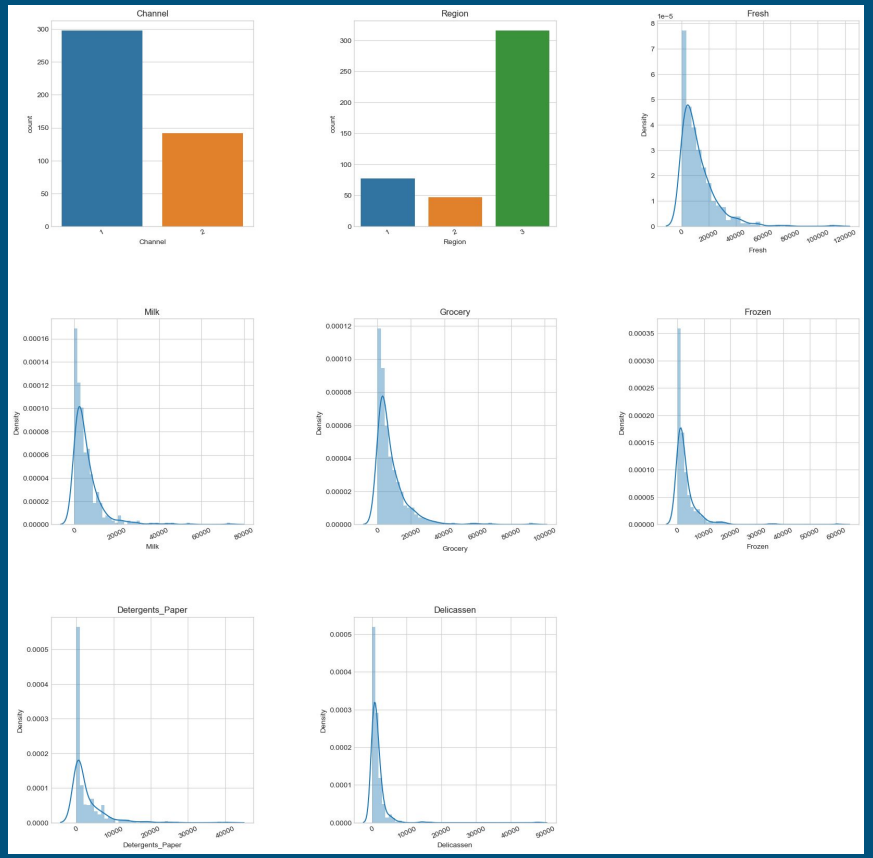
Group Name: **TAD**

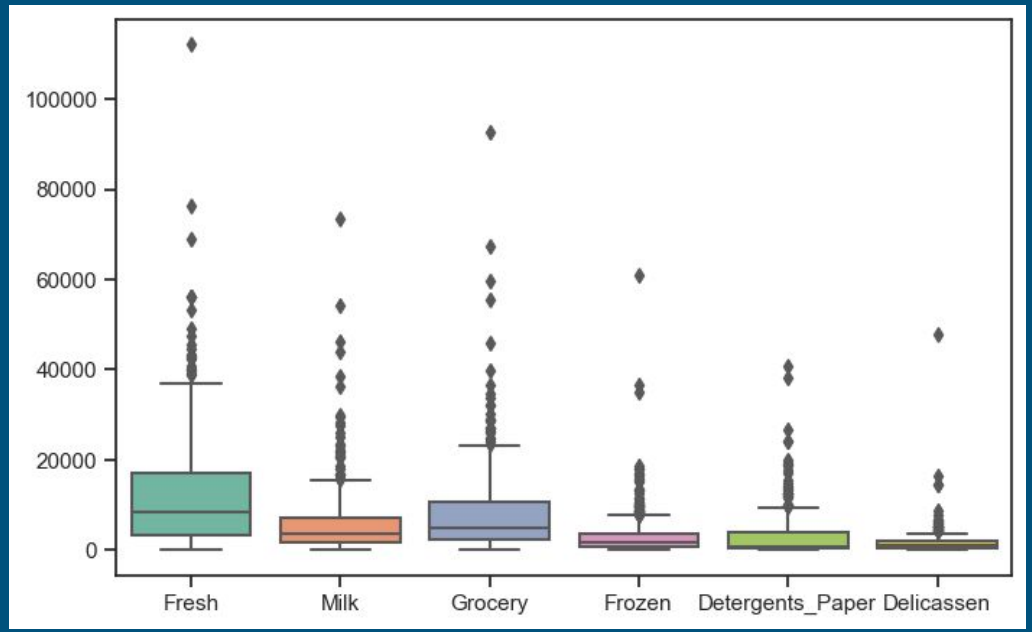
Lab ML For Data Science: Part I

Getting Insights into an Unsupervised Dataset

1.

Loading the Data, Preprocessing, Initial Data Analysis





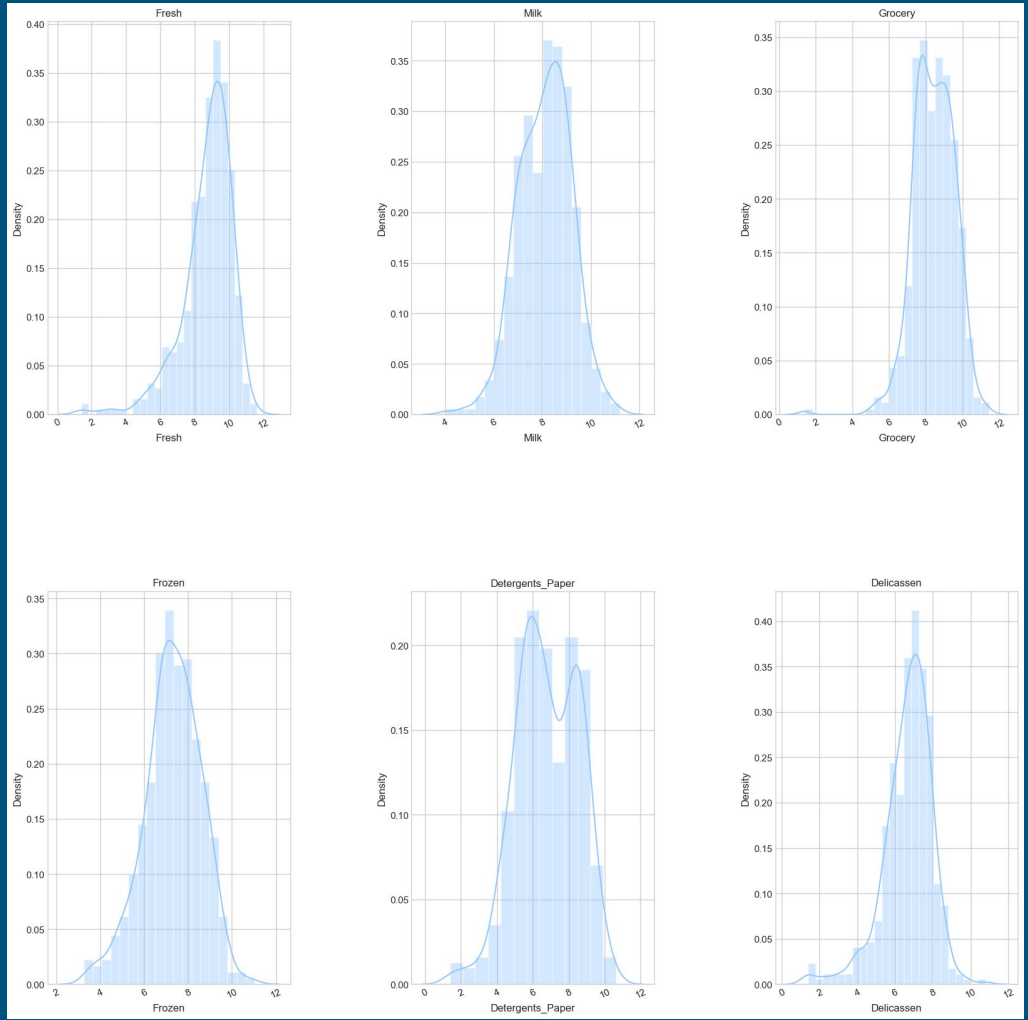
A general distribution shows that there is an imbalance between the number of instances in channel and in region.

For each category, there are many outliers towards the higher end indicating that data is highly skewed to the right.

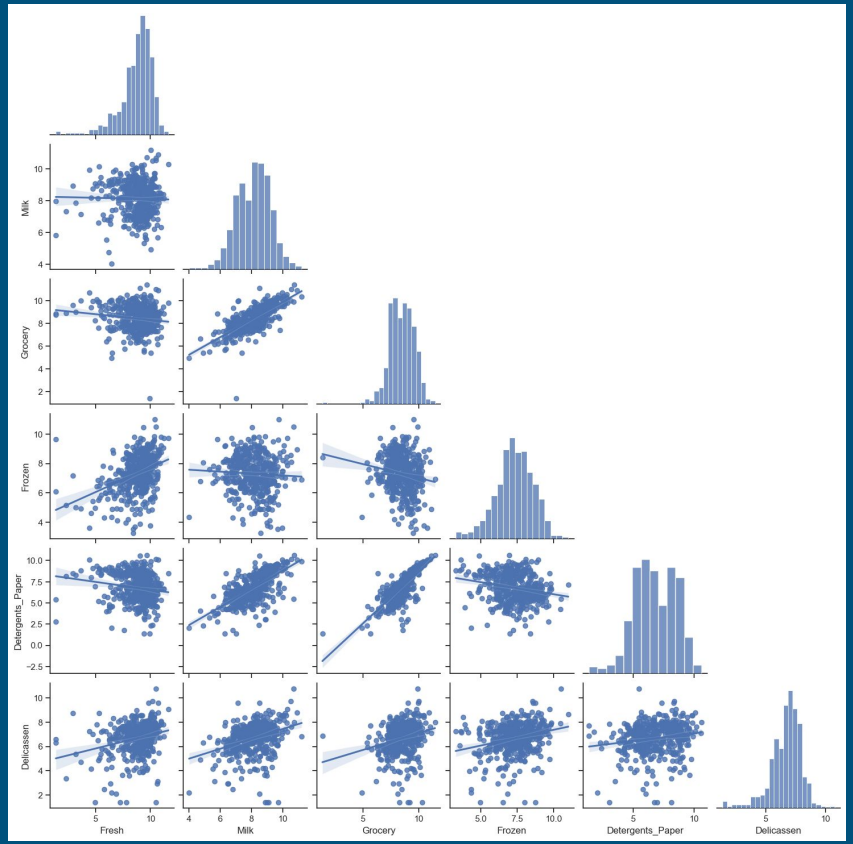
We drop the metadata Channel and Region and focus on numerical data to analyse anomalies.

Since data was highly skewed towards right, we apply log transform
X <- log(x + 1)

We add offset 1 in log so that 0 values get mapped to a specific value.



There is a high correlation between Detergents_Paper and Grocery. Also, Milk and Grocery are considerably correlated.



2.

Detecting Anomalies

Anomaly Scores from Distance Matrix (Hardmin)

```
dist_matrix = calculate_dist_matrix(df_log)
dist_matrix
```

	0	1	2	3	4	5	6	7	8	9	...	430	431	432	433	434
0	0.000000	2.224584	3.085157	4.366366	3.331161	1.307943	1.645392	2.330232	1.597930	2.345042	...	3.195456	4.266183	2.642341	3.901349	1.575182
1	2.224584	0.000000	1.541680	3.263750	1.988811	1.362131	2.174861	0.780828	2.092733	1.169105	...	3.126408	2.383102	3.582755	3.361482	1.780825
2	3.085157	1.541680	0.000000	3.431039	1.651504	2.294735	3.340546	1.337177	3.137447	1.921839	...	3.550237	2.656078	4.519018	3.651499	2.882593
3	4.366366	3.263750	3.431039	0.000000	2.406136	3.262152	3.557725	2.905404	3.407516	4.238947	...	3.455373	2.036131	3.668271	2.804716	3.128861
4	3.331161	1.988811	1.651504	2.406136	0.000000	2.411643	3.233237	1.699094	3.257427	2.746759	...	3.630900	2.048335	4.048988	3.706398	2.573983
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
435	5.069015	3.848109	4.030637	3.060203	2.953376	4.127355	4.921473	3.951802	4.778894	4.711158	...	4.008258	2.771055	5.387995	4.462180	4.273217
436	5.564116	5.161151	5.187257	2.677452	4.048232	4.616688	5.134254	4.925099	4.909501	6.240254	...	4.710604	4.049138	4.677306	3.630164	4.717930
437	2.390978	2.502427	3.148681	5.387722	3.702175	2.904145	2.933326	2.685797	3.298699	1.598576	...	4.633749	4.682201	4.327725	5.569987	2.807914
438	3.789719	3.735002	3.950812	2.296451	3.330583	2.943401	3.540550	3.484226	2.993919	4.704888	...	2.547515	3.588645	3.204004	1.992430	3.215658
439	4.616926	5.708060	6.821026	6.012160	6.799926	4.748969	4.066376	5.757638	3.757454	6.120177	...	4.797128	6.614062	3.424337	4.664636	4.576984

440 rows × 440 columns

338	4.945913
75	4.647444
154	4.201955
142	3.774788
95	3.746081

Top Anomalous Points

1. Calculate distance of one point from all the other points.

2. Get Min. distance of each point to any other point (i.e. taking the minimum value for each row, denoting a data point)

3. Check the data points with the Maximum of minimum distance.

Anomaly Scores from Distance Matrix (Softmin)

$$\operatorname{soft}\min_{k \neq j} \{z_{jk}\} = -\frac{1}{\gamma} \log \left(\frac{1}{N-1} \sum_{k \neq j} \exp(-\gamma z_{jk}) \right).$$

1. Apply Softmin function on distance scores instead of directly taking the minimum.

(Initially, we assume γ to be 0.5)

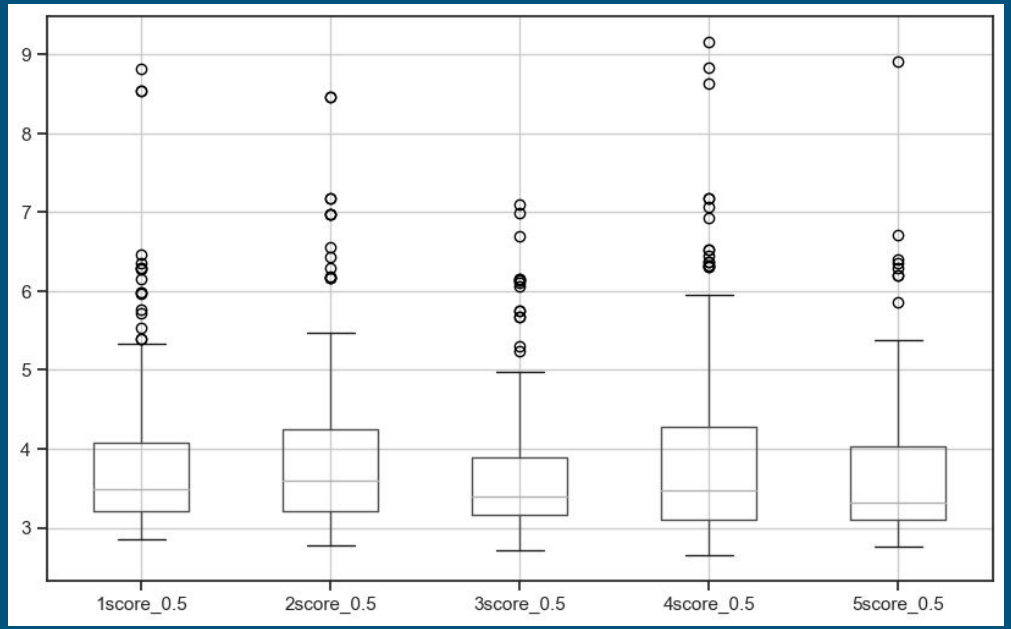
2. Check the data points with Maximum minimum distance.

338	9.151482
154	8.881429
75	8.698126
95	7.730110
66	7.595647

Top Anomalous Points

Anomaly Scores from Distance Matrix (Softmin)

Is the Softmin score Reproducible?



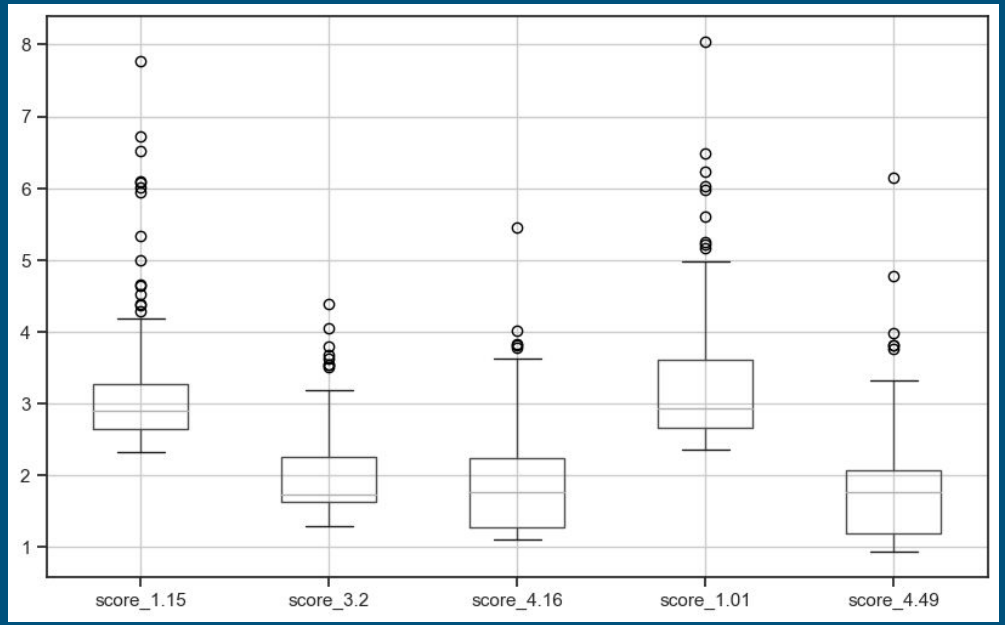
1. Apply Bootstrapping to the dataset (with a 50% sampling)

2. The Softmin scores usually fall in the range of 3 to 4 (for γ = 0.5).

3. There are some outliers for each case - deemed to be anomalous.

Anomaly Scores from Distance Matrix (Softmin)

Choosing γ :

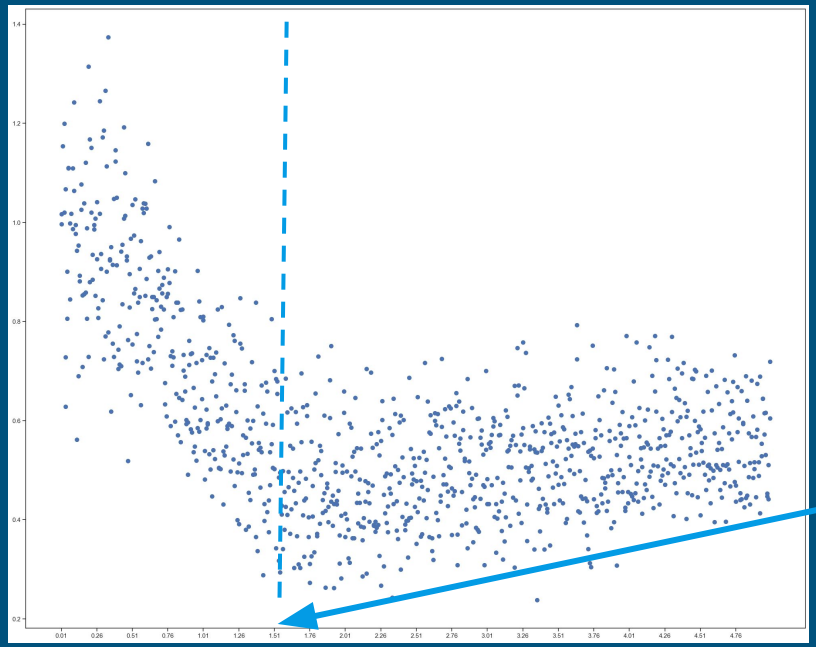


1. Use different γ values on bootstrapped data.

2. The range of Softmin scores change with different γ

Anomaly Scores from Distance Matrix (Softmin)

What is the best value of γ ?



1. Experiment with different γ in the range from 0 to 5.

2. Notice, after increasing γ to a certain limit, drop in Variance is stagnant.

3. We choose the cut-off value as our γ

We decide γ to be 1.5!

Analyzing Anomaly Points:

We apply softmin with γ = 1.5 and look at most anomalous points:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
338	3	333	7021	15601	15	550
75	20398	1137	3	4407	3	975
154	622	55	137	75	7	8
95	3	2920	6252	440	223	709
142	37036	7152	8253	2995	20	3
128	140	8847	3823	142	1062	3
187	2438	8002	9819	6269	3459	3
66	9	1534	7417	175	3468	27
109	1406	16729	28986	673	836	3
183	36847	43950	20170	36534	239	47943

How does these data points look like compared to the rest?

Comparing Anomaly Points:

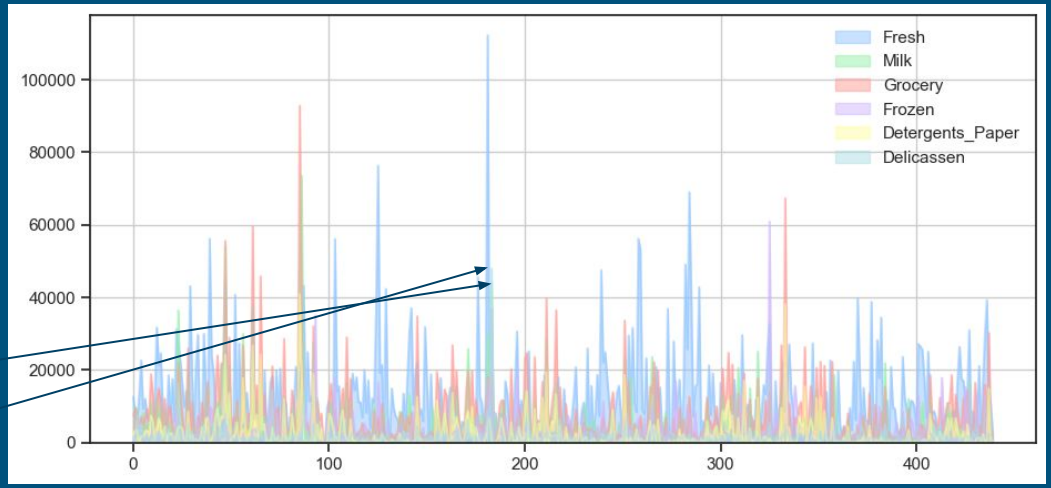
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
338	3	333	7021	15601	15	550
75	20398	1137	3	4407	3	975
154	622	55	137	75	7	8
95	3	2920	6252	440	223	709
142	37036	7152	8253	2995	20	3
128	140	8847	3823	142	1062	3
187	2438	8002	9819	6269	3459	3
66	9	1534	7417	175	3468	27
109	1406	16729	28986	673	836	3
183	36847	43950	20170	36534	239	47943

	Mean	Std.	25%	50%	75%
Fresh	12000.30	12647.33	3127.75	8504.0	16933.75
Milk	5796.27	7380.38	1533.00	3627.0	7190.25
Grocery	7951.28	9503.16	2153.00	4755.5	10655.75
Frozen	3071.93	4854.67	742.25	1526.0	3554.25
Detergents_Paper	2881.49	4767.85	256.75	816.5	3922.00
Delicassen	1524.87	2820.11	408.25	965.5	1820.25

Customer 338 seems to be a valid outlier based on their purchase of Fresh Foods, Milk etc.

Comparing Anomaly Points:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
338	3	333	7021	15601	15	550
75	20398	1137	3	4407	3	975
154	622	55	137	75	7	8
95	3	2920	6252	440	223	709
142	37036	7152	8253	2995	20	3
128	140	8847	3823	142	1062	3
187	2438	8002	9819	6269	3459	3
66	9	1534	7417	175	3468	27
109	1406	16729	28986	673	836	3
183	36847	43950	20170	36534	239	47943



	Mean	Std.	25%	50%	75%
Fresh	12000.30	12647.33	3127.75	8504.0	16933.75
Milk	5796.27	7380.38	1533.00	3627.0	7190.25
Grocery	7951.28	9503.16	2153.00	4755.5	10655.75
Frozen	3071.93	4854.67	742.25	1526.0	3554.25
Detergents_Paper	2881.49	4767.85	256.75	816.5	3922.00
Delicassen	1524.87	2820.11	408.25	965.5	1820.25

Comparing with the distribution of expenses for each customer, we can also validate our anomalies, especially when they overspend on a category.

3.

Explaining Anomalies

Explainable AI

It becomes important to explain the anomalies along with identifying them. In this task, we use the method of Layer-wise Relevance Propagation to explain high anomaly score.

$$R_k^{(j)} = \frac{\exp(-\gamma z_{jk})}{\sum_{k \neq j} \exp(-\gamma z_{jk})} \cdot y_j$$

For each instance j, the anomaly score is determined by softmin, and it is influenced by every other datapoint. Hence, as a first step, we identify how much each datapoint contributes to anomaly score of each instance.

	0	1	2	3	4	5	6	7	8	9	...	430	
0	0.000000e+00	2.671634e-03	2.818816e-06	1.701283e-12	2.641206e-07	3.437024e-01	7.708246e-02	1.298071e-03	9.710369e-02	1.170013e-03	...	9.972019e-07	6.225
1	2.330389e-04	0.000000e+00	1.103909e-02	4.488127e-08	1.034124e-03	2.413125e-02	3.235429e-04	1.563458e-01	5.473370e-04	5.021935e-02	...	1.674156e-07	7.791
2	3.072270e-06	1.379350e-01	0.000000e+00	1.045236e-07	8.151190e-02	1.809839e-03	2.620634e-07	3.335802e-01	1.885782e-06	1.913940e-02	...	2.999974e-08	1.236
3	3.754520e-13	1.135511e-07	2.116409e-08	0.000000e+00	1.670611e-04	1.153419e-07	5.608237e-09	3.128444e-06	2.693978e-08	1.944679e-12	...	1.646026e-08	1.966
4	1.078237e-07	4.839867e-03	3.053097e-02	3.090364e-04	0.000000e+00	2.969804e-04	2.827938e-07	2.403619e-02	2.234534e-07	2.221279e-05	...	4.712875e-09	3.375
...	...	...	...	...	...	...	...	...	...	...	...	...	
435	1.068199e-15	1.320921e-08	1.527715e-09	4.642375e-05	1.216846e-04	4.677782e-10	9.747337e-15	3.926401e-09	7.760250e-14	2.035311e-13	...	2.000966e-09	5.822
436	1.045218e-19	6.833979e-17	4.557073e-17	3.291693e-04	3.247086e-10	2.007871e-13	1.035302e-16	2.430392e-15	3.059199e-15	6.603872e-25	...	5.396174e-14	3.211
437	1.301597e-03	5.743844e-04	2.399415e-06	8.489178e-19	8.126344e-09	2.209650e-05	1.711411e-05	1.378613e-04	5.623617e-07	1.492442e-01	...	7.097483e-14	3.606
438	5.346820e-10	9.915559e-10	8.237261e-11	4.452303e-04	7.207491e-08	2.755808e-06	8.278268e-09	1.498615e-08	1.757323e-06	4.610923e-15	...	7.183499e-05	4.949
439	2.373244e-10	1.087025e-17	8.958749e-27	5.181062e-20	1.378695e-26	3.712847e-11	3.087601e-07	4.633684e-18	1.159115e-05	7.256126e-21	...	1.863010e-11	5.802

440 rows × 440 columns

For each instance, we calculate the relevance contribution of each datapoint as a 440x440 matrix, where each row represents each instance and contribution of each datapoint to softmin is given in columns.

Explainable AI

$$R_i^{(j)} = \sum_{k \neq j} \frac{[\boldsymbol{x}_k - \boldsymbol{x}_j]_i^2}{\|\boldsymbol{x}_k - \boldsymbol{x}_j\|^2} \cdot R_k^{(j)}$$

The relevance contribution of each datapoint can then further be backtracked to each features by observing that the (squared) Euclidean distance entering the anomaly score can be decomposed in terms of individual components:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Total_Relevance
0	0.904516	0.583854	0.250292	1.437025	0.468388	0.468318	4.112392
1	0.644680	0.661569	0.185157	0.631423	0.350877	0.279917	2.753624
2	0.589532	0.614826	0.175790	0.448208	0.342300	1.991170	4.161826
3	0.464723	0.687186	0.493128	0.437054	0.460333	0.717498	3.259922
4	0.472894	0.313277	0.237213	1.141064	0.595644	0.843142	3.603235
...	...	...	...	...	...	...	...
435	0.202129	1.107492	1.623817	0.656590	1.101971	0.926649	5.618648
436	0.600827	0.809939	1.203615	0.727913	1.038167	0.448858	4.829319
437	0.324045	0.274384	0.377892	1.894288	0.638277	0.852863	4.361749
438	0.708035	0.479280	0.290250	0.788470	0.299125	0.809403	3.374563
439	1.141659	1.065617	1.690130	2.347667	0.499341	2.379591	9.124005

440 rows × 7 columns

For each instance, with this split of relevance contribution, we can clearly see what feature contributes the most to the anomaly score, making it more explainable.

Explainable AI

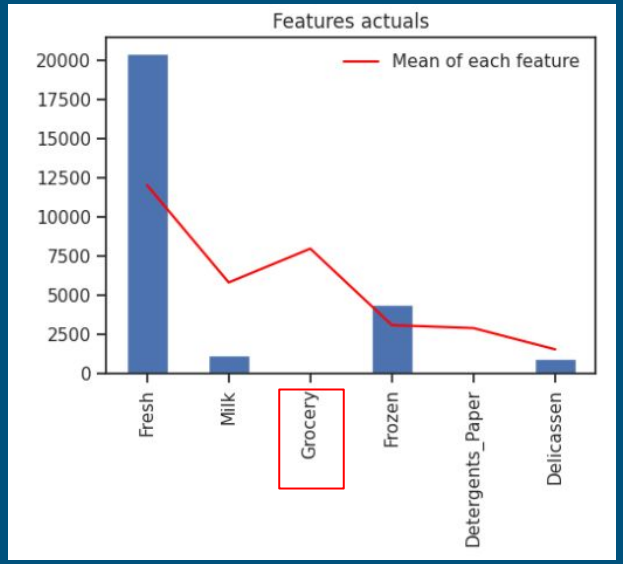


Fig: Actual Spending across categories

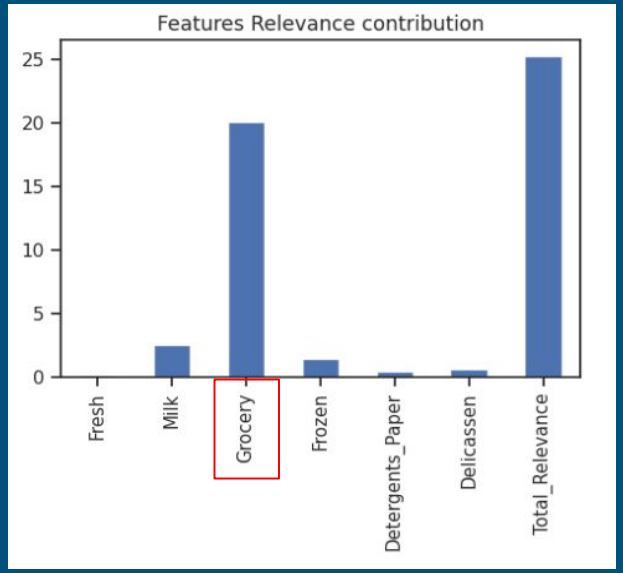


Fig: Relevance Contribution across categories

This is an example where we can see, the highest contributor to anomaly score is Grocery; where the spending on grocery is nil and the average spending is much higher.

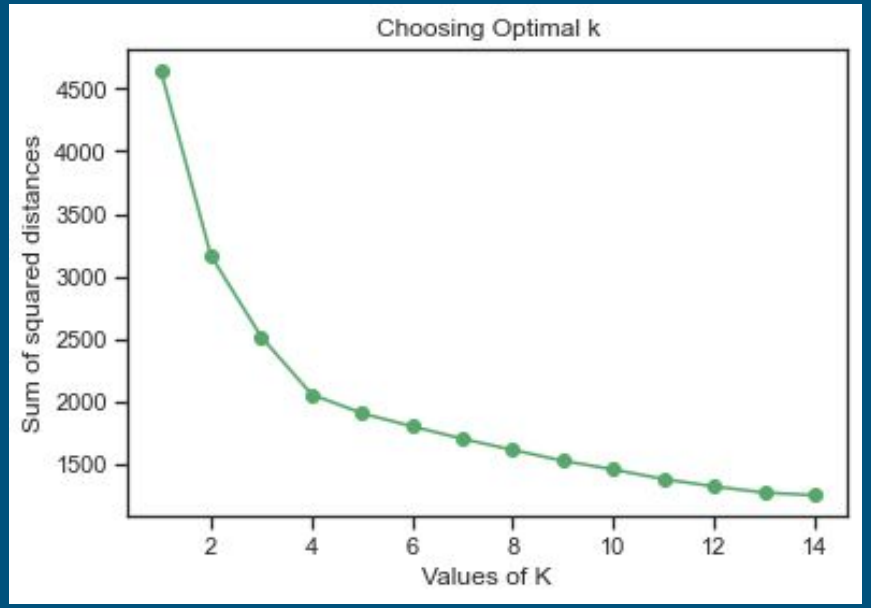
Only with anomaly score, it might not be intuitive to say if it is overspending or underspending which makes this an outlier.

	Mean	Std.	25%	50%	75%
Fresh	12000.30	12647.33	3127.75	8504.0	16933.75
Milk	5796.27	7380.38	1533.00	3627.0	7190.25
Grocery	7951.28	9503.16	2153.00	4755.5	10655.75
Frozen	3071.93	4854.67	742.25	1526.0	3554.25
Detergents_Paper	2881.49	4767.85	256.75	816.5	3922.00
Delicassen	1524.87	2820.11	408.25	965.5	1820.25

4.

Cluster Analysis

K-Means Clustering on Remaining Data:




```
Cluster 1 has 66 members
Cluster 2 has 129 members
Cluster 3 has 105 members
Cluster 4 has 130 members
```

```
df_log['cluster'] = kmeans.labels_
df_log
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	cluster
0	9.446992	9.175438	8.930891	5.370638	7.891705	7.199678	2
1	8.861917	9.191259	9.166284	7.474772	8.099858	7.482682	2
2	8.756840	9.083529	8.947026	7.785721	8.165364	8.967632	2
3	9.492960	7.087574	8.348064	8.764834	6.230481	7.489412	3
4	10.026413	8.596189	8.881697	8.272826	7.483244	8.553718	2
...	...	...	...	...	...	...	...
435	10.299037	9.396986	9.682092	9.483112	5.209486	7.698483	3
436	10.577172	7.266827	6.639876	8.414274	4.543295	7.760893	3
437	9.584108	9.647885	10.317053	6.082219	9.605216	7.532624	2
438	9.239025	7.591862	7.711101	6.946014	5.129899	7.661998	1
439	7.933080	7.437795	7.828436	4.189655	6.169611	3.970292	1

430 rows × 7 columns

We decide 4 clusters to be optimal. We introduce a new column, “cluster”, denoting the membership of a point.

Analyzing Results of K-Means Clustering

These are our 4 cluster centres (how a “typical” customer looks like) in our scaled dataset

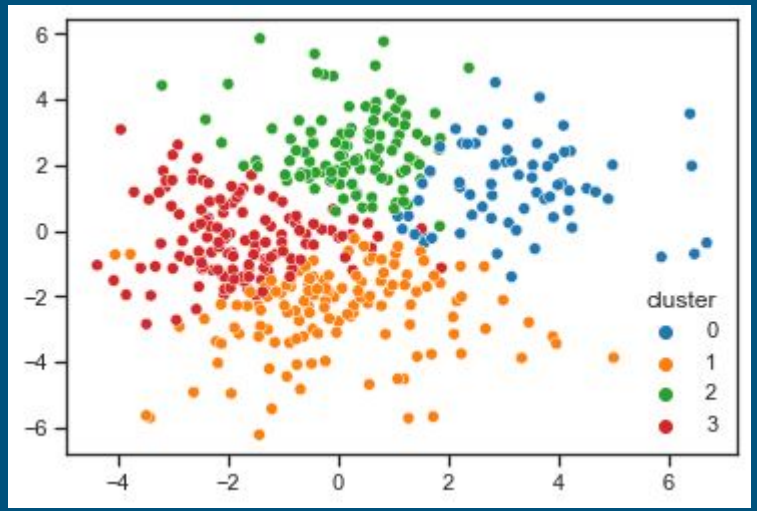
```
array([[7.071, 8.782, 9.372, 5.859, 8.503, 6.262],
       [8.628, 7.224, 7.516, 6.913, 5.315, 6.024],
       [9.143, 9.095, 9.457, 7.351, 8.515, 7.391],
       [9.54 , 7.904, 8.108, 8.389, 6.133, 7.133]])
```

vs.

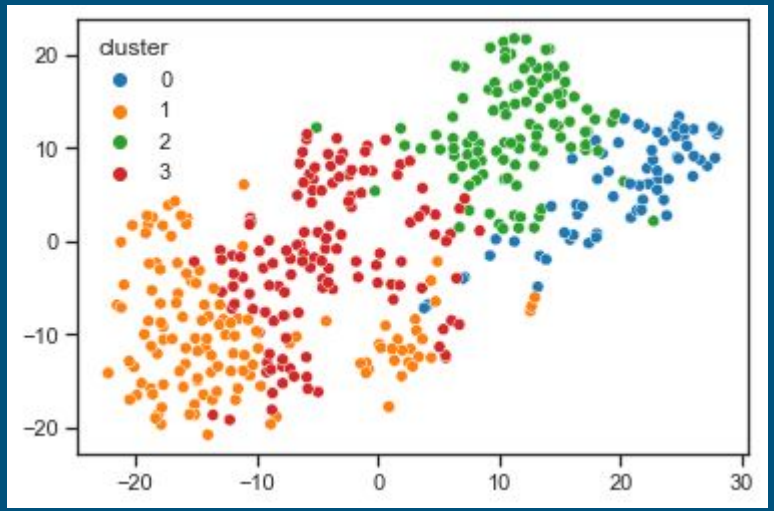
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	430.000000	430.000000	430.000000	430.000000	430.000000	430.000000
mean	8.790636	8.125476	8.453723	7.304672	6.833045	6.729462
std	1.338203	1.054783	1.050306	1.259815	1.667777	1.156351
min	2.944439	4.727388	5.389072	3.258097	1.386294	2.079442
25%	8.109368	7.360222	7.673222	6.646058	5.593719	6.044349
50%	9.057013	8.196435	8.464846	7.331043	6.711132	6.894670
75%	9.735449	8.868730	9.275776	8.166423	8.295236	7.510567
max	11.627610	11.205027	11.437997	11.016496	10.617123	9.712569

Analyzing Results of K-Means Clustering

We also use different embedding methods and check if K-Means cluster membership gives a good result.



MDS



T-SNE

Lab ML For Data Science: Part II

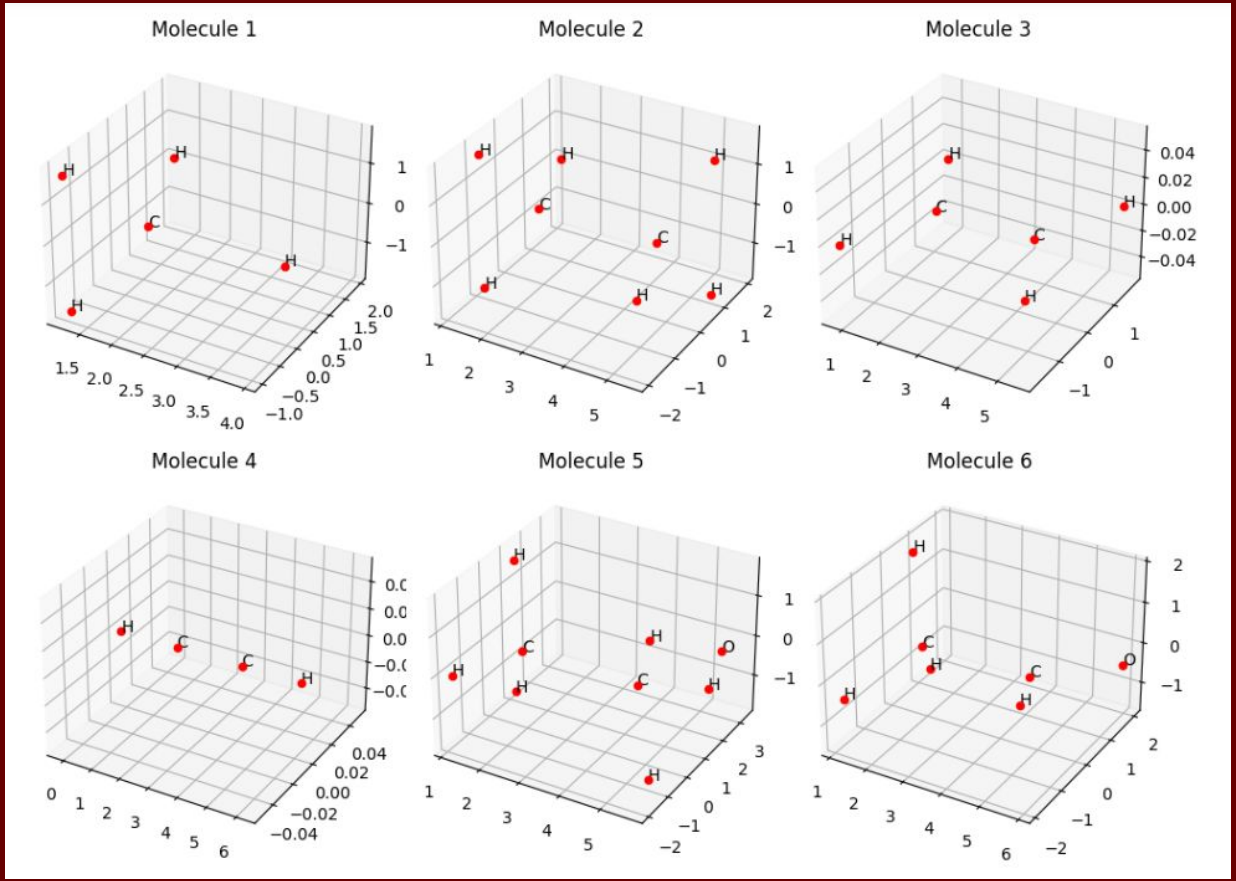
Getting Insights into Quantum-Chemical Relations

1.1.

Visualizing Molecules

QM7 DataSet

- Consists of 7165 molecules and its atomic structure, each molecule comprises 23 atoms
- Variable R:
  - It has the 3D coordinates of each of the 23 atoms in a single molecule.
  - Shape: 7165x23x3
  -
- Variable Z:
  - It has the atomic number of each atom in a molecule.
  - Shape: 7165x23
  - Atomic Number 0 signifies no atom
- Variable T :
  - It has the atomization energy.
  - Shape: 1x7165
  - Signifies energy required to break bonds



Plain scatter plot of the atoms of first 6 molecules in 3D space

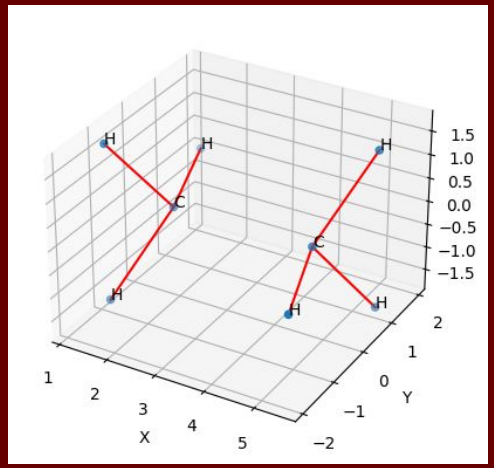
Finding an existing bond

Out of the 23 atom coordinates, first molecule has 5 atoms.
To find out an existing bond, based on a distance threshold, we consider only the non-zero coordinates.
We form a list of 7165 molecules having the shape: (Number of atoms)x3

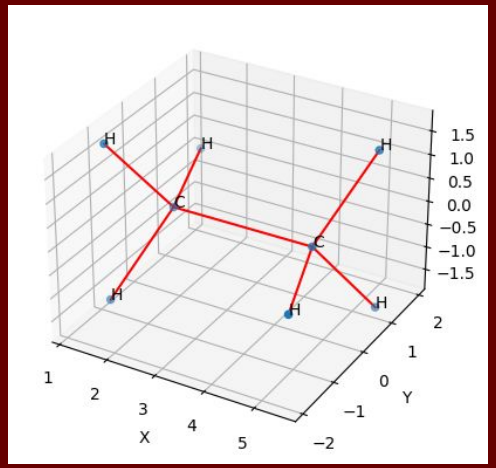
```
First molecule has only 5 valid atoms, and the coordinates are given by :

array([[ 1.886438  , -0.00464873, -0.00823921],
       [ 3.9499245 , -0.00459203,  0.00782347],
       [ 1.1976895 ,  1.9404842 ,  0.00782347],
       [ 1.1849339 , -0.99726516,  1.6593875 ],
       [ 1.2119948 , -0.9589793 , -1.710958  ]], dtype=float32)
```

Plotting the second molecule C2H6

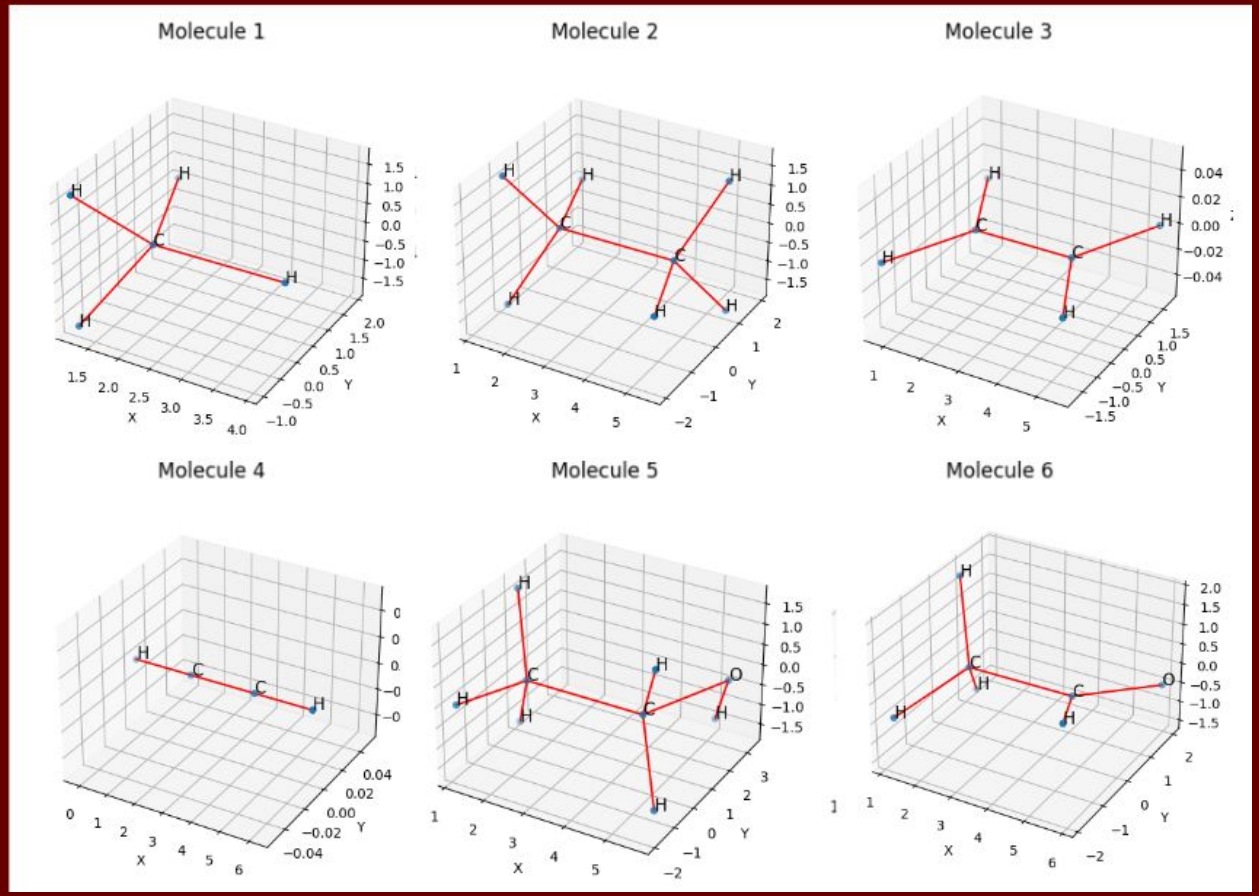
Threshold distance=2.5



Threshold distance=3



We know that in C2H6, there is a bond between the two Carbon atoms. So we decided to move the distance threshold to 3 for a more accurate representation.



Scatterplot of the atoms of first 6 molecules with bond threshold=3

2.1

Data Representation

One-hot Encoding

```
unique_atoms = np.unique(qm7_data_atomic_number, axis=None)
unique_atoms
```

```
array([ 0.,  1.,  6.,  7.,  8., 16.], dtype=float32)
```

- The atomic number array (Z) has 6 unique numbers out of which 0 is invalid representing no atom. (We remove it from the list.)
- To do regression on the atoms, as a function of each unique atom in a molecule, we need to have uniform representation of each molecule.
- Hence we one-hot encode each atom and then sum it over unique atoms.

One-hot Encoding

Example:
In the first molecule (CH4) we have 1 C and 4 H and rest are invalid atoms. Since 0 is invalid representing no atom, we remove it.

```
unique_atoms_list [1.0, 6.0, 7.0, 8.0, 16.0]
```

- Now, the first position of the encoder is for atomic number 1 (H) and the second for 6 (C) and so on
- Therefore CH4 can be represented as:
- Shape: 23x5
- Now we sum over one hot encoding of each atom to get uniform representation by each unique atom

Molecular Structure of first 5 molecules:

```
array([[4, 1, 0, 0, 0],
       [6, 2, 0, 0, 0],
       [4, 2, 0, 0, 0],
       [2, 2, 0, 0, 0],
       [6, 2, 0, 1, 0]])
```

```
[[0, 1, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

2.2.

Ridge Regression Model

Data Preparation for Ridge Regression Model

```
X=one_hot_df_array # Features: One hot representing molecular structure
Y=qm7_data_atomization_energy.reshape(-1, 1) # Target: Atomization Energy

#Splitting the data in train and test and validation
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42,)
X_train, X_val, y_train, y_val   = train_test_split(X_train, y_train, test_size=0.25, random_state=1)

# Scaling after splitting
X_train_scaled= scaler_X.fit_transform(X_train)
X_test_scaled= scaler_X.transform(X_test)
X_val_scaled = scaler_X.transform(X_val)

y_train_scaled=scaler_Y.fit_transform(y_train)
y_test_scaled=scaler_Y.transform(y_test)
y_val_scaled = scaler_Y.transform(y_val)
```

Determining the Regularization Parameter (lambda)

We optimize the regularization parameter based on the improvement in mean squared error.
We get the optimised lambda to be 0.28
After running the Ridge model, we get the weights as:

```
array([[-0.78542092, -0.63865221, -0.36023638, -0.33025999, -0.07239034]])
```

```
MSE: 0.008387
Lambda: 0.01
MSE: 0.0083869
Lambda: 0.05
MSE: 0.0083868
Lambda: 0.13
MSE: 0.0083867
Lambda: 0.2
MSE: 0.0083866
Lambda: 0.28
Best MSE: 0.0083866
Best Lambda: 0.28
```

Predictions from ridge seem to match the actual values (both scaled) with the following error values:

```
Maximum error between predicted and actual atomization energy: 0.2697
Average error in predicted and actual atomization energy: -0.0049
```

Closed Form Solution

- The Ridge weights can be calculated using the closed form solution given by: $w = (\Sigma_{xx} + \lambda I)^{-1}\Sigma_{xt}$

where $\Sigma_{xx} = \mathbb{E}[xx^\top]$ and $\Sigma_{xt} = \mathbb{E}[xt]$ are auto- and cross-covariance matrices respectively.

Weights from closed form solution:

```
array([[-0.7853891 ],
       [-0.63850458],
       [-0.36009454],
       [-0.33012226],
       [-0.07235176]])
```

Weights from Ridge Regression Model:

```
array([[-0.78542092, -0.63865221, -0.36023638, -0.33025999, -0.07239034]])
```

2.3.

Deeper Insights with Explanations

Contribution of each atom in Prediction:

```
oh= onehot_encoding[molecule_index]
oh
```

```
[[0, 0, 1, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 1, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
```

```
one_hot_df.iloc[molecule_index]
```

```
1.0     5
6.0     4
7.0     1
8.0     2
16.0    0
```

This molecule has 5 H, 4C, 1N and 2 O, i.e. a total of 12 atoms

Multiplying the One Hot representation with our Ridge Regression Weights, gives contribution of each atom:

```
molecule_relevance_by_atom = ridge_weights @ np.array(oh).T
#Relevance of each atom in predicting y (target feature) = w.T . onehotencoding
molecule_relevance_by_atom
```

```
array([[-0.36023638, -0.63865221, -0.63865221, -0.63865221, -0.63865221,
        -0.33025999, -0.33025999, -0.78542092, -0.78542092, -0.78542092,
        -0.78542092, -0.78542092,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ]])
```

The first atom in this molecule is N which contributes -0.360236 to the total atomization energy of -7.5.

Contribution of each atom in Prediction:

	1.0	6.0	7.0	8.0	16.0	relevance
0	0	0	1	0	0	-0.360236
1	0	1	0	0	0	-0.638652
2	0	1	0	0	0	-0.638652
3	0	1	0	0	0	-0.638652
4	0	1	0	0	0	-0.638652
5	0	0	0	1	0	-0.330260
6	0	0	0	1	0	-0.330260
7	1	0	0	0	0	-0.785421
8	1	0	0	0	0	-0.785421
9	1	0	0	0	0	-0.785421
10	1	0	0	0	0	-0.785421
11	1	0	0	0	0	-0.785421
12	0	0	0	0	0	0.000000
13	0	0	0	0	0	0.000000
14	0	0	0	0	0	0.000000
15	0	0	0	0	0	0.000000
16	0	0	0	0	0	0.000000
17	0	0	0	0	0	0.000000
18	0	0	0	0	0	0.000000
19	0	0	0	0	0	0.000000
20	0	0	0	0	0	0.000000
21	0	0	0	0	0	0.000000
22	0	0	0	0	0	0.000000

$$
\begin{aligned}
f(x) &= w^T \cdot (Xscaled) \\
&= w^T \cdot (X - X_{\text{mean}}) \\
&= (w^T \cdot X) - (w^T \cdot X_{\text{mean}}) \\
&= (w^T \cdot X) - AverageAtomizationEnergy \\
Also, \\
f(x) &= w^T \cdot \left(\sum (\text{one hot encodings}) - X_{\text{mean}}\right) \\
&= w^T \cdot \sum (\text{one hot encodings}) - w^T \cdot X_{\text{mean}} \\
&= \sum \left(w^T \cdot (\text{one hot encodings})\right) - w^T \cdot X_{\text{mean}} \\
&= \sum \left(w^T \cdot (\text{one hot encodings})\right) - AverageAtomizationEnergy
\end{aligned}
$$

We can show that $w^T \cdot X$=∑($w^T \cdot$ one hot encodings).

```
ridge_weights @ X[5732]  # y for 5732nd molecule = w.T . X
```

```
array([-7.50246979])
```

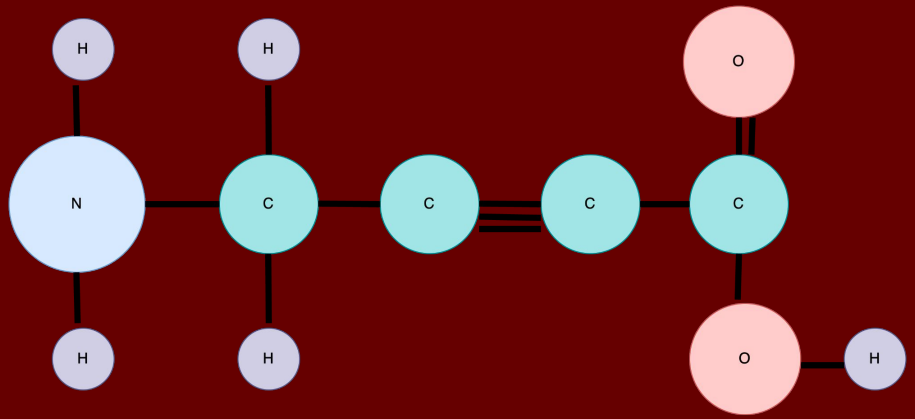
```
(ridge_weights @ np.array(oh).T).sum() #Sum of each individual atom's contribution = Sum(w.T . onehotencoding)
# which is same as w.T . X which we see in the cell above
```

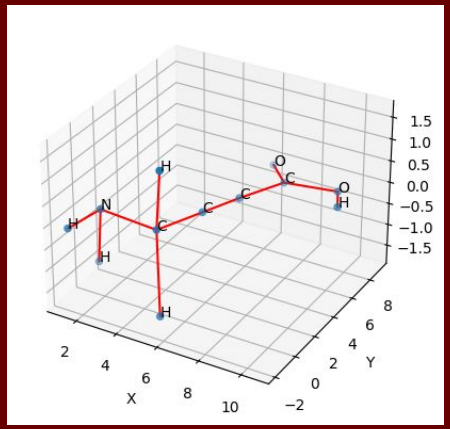
```
-7.502469785542177
```

Contribution of each atom in Prediction:

	1.0	6.0	7.0	8.0	16.0	relevance
0	0	0	1	0	0	-0.360236
1	0	1	0	0	0	-0.638652
2	0	1	0	0	0	-0.638652
3	0	1	0	0	0	-0.638652
4	0	1	0	0	0	-0.638652
5	0	0	0	1	0	-0.330260
6	0	0	0	1	0	-0.330260
7	1	0	0	0	0	-0.785421
8	1	0	0	0	0	-0.785421
9	1	0	0	0	0	-0.785421
10	1	0	0	0	0	-0.785421
11	1	0	0	0	0	-0.785421
12	0	0	0	0	0	0.000000
13	0	0	0	0	0	0.000000
14	0	0	0	0	0	0.000000
15	0	0	0	0	0	0.000000
16	0	0	0	0	0	0.000000
17	0	0	0	0	0	0.000000
18	0	0	0	0	0	0.000000
19	0	0	0	0	0	0.000000
20	0	0	0	0	0	0.000000
21	0	0	0	0	0	0.000000
22	0	0	0	0	0	0.000000

However, considering the scatterplot it might have the following atomic structure:





So far, we do not account for relevance by type of bond, it just helps us understand the contribution by each type of atom.

3.1.

Simple atom-based Representation

Mapping each molecule to Vector and applying Regression

```
'''
As explained above, for each molecule, now we sum over one hot encoding of each atom to get uniform representation
'''
one_hot_encoded_array = np.array(onehot_encoding)
one_hot_df = pd.DataFrame(np.sum(one_hot_encoded_array, axis=1), columns=unique_atoms_list)
one_hot_df_array = one_hot_df.to_numpy()
one_hot_df_array.shape
one_hot_df_array[:5]
```

```
array([[4, 1, 0, 0, 0],
       [6, 2, 0, 0, 0],
       [4, 2, 0, 0, 0],
       [2, 2, 0, 0, 0],
       [6, 2, 0, 1, 0]])
```

From One Hot Encodings of each atom, we get their Vector representation by summing up the type of atoms. For example, the first molecule (CH4) is represented as {4,1,0,0,0} as shown above.

We then perform Ridge Regression and see contribution of each type of atom into predicting the atomization energy as shown before.

3.2. Models with Pairs of Atoms

Molecule Representation based on Pairwise Distance between atoms (Hard Indicator)

Building ϕ^A:

We represent each molecule based on the pairwise distances between each combination of atoms. Here is an example of the first molecule CH4:

```
distance_info[0]
```

```
[{'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.063549041748047},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635344982147217},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635828971862793},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0651566982269287},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.37018084526062},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3701982498168945},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706564903259277},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370192527770996},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706531524658203},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706717491149902}]
```

We initially take the interval 0-1, 1-2, .., 5-6 of unit distance. ϕ^A for CH4 then looks like :

```
intervals = [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
results = create_one_hot_encoding(non_zero_coordinates, intervals)
```

```
results[0] #for 1st molecules with 5 atoms
```

```
array([[0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0, 0],

      [0, 0, 0, 1, 0, 0],
      [0, 0, 0, 1, 0, 0],
      [0, 0, 0, 1, 0, 0]])
```

Molecule Representation based on Pairwise Distance between atoms (Soft Indicator)

Building ϕ^A:

To avoid unnatural discontinuation, we replace the previous method by a Gaussian Function with mean at the centre of the interval with a fixed variance.

```
distance_info[0]
```

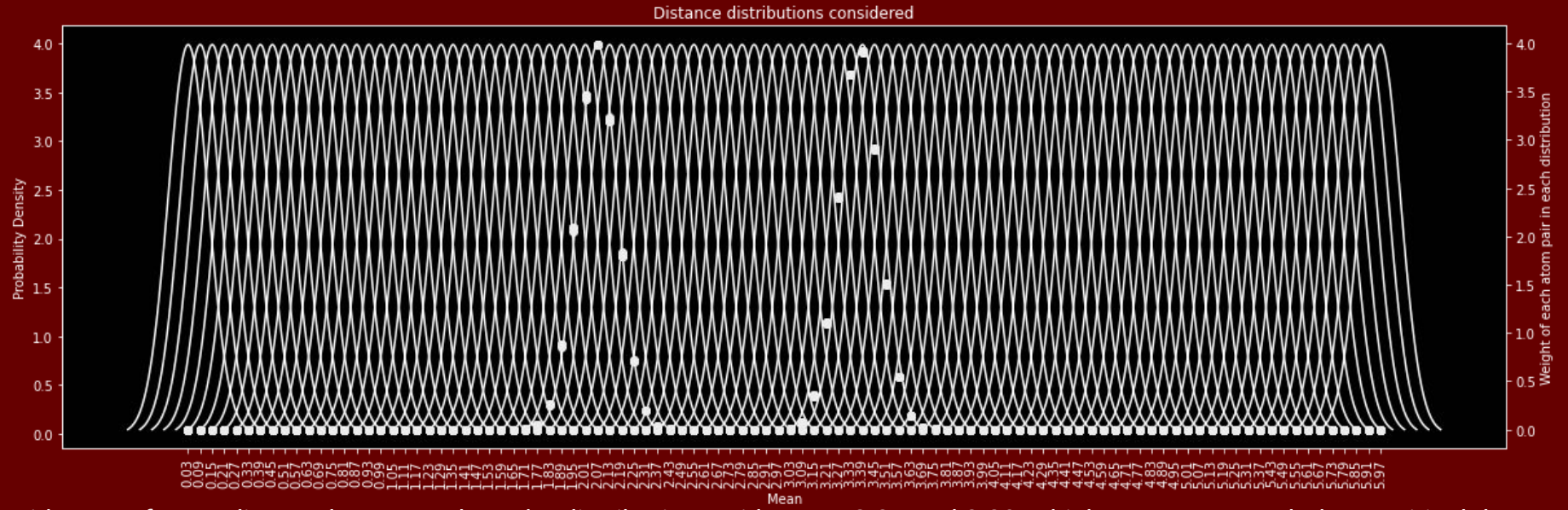
```
[{'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.063549041748047},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635344982147217},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635828971862793},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0651566982269287},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.37018084526062},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3701982498168945},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706564903259277},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370192527770996},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706531524658203},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706717491149902}]
```

We make the intervals much more continuous, taking 100 intervals between 0 to 6 (each of length 0.06). We choose a variance of 0.01 for the Gaussian Distribution.

ϕ^A for CH4 now has the shape 10 * 100

Molecule Representation based on Pairwise Distance between atoms (Soft Indicator)

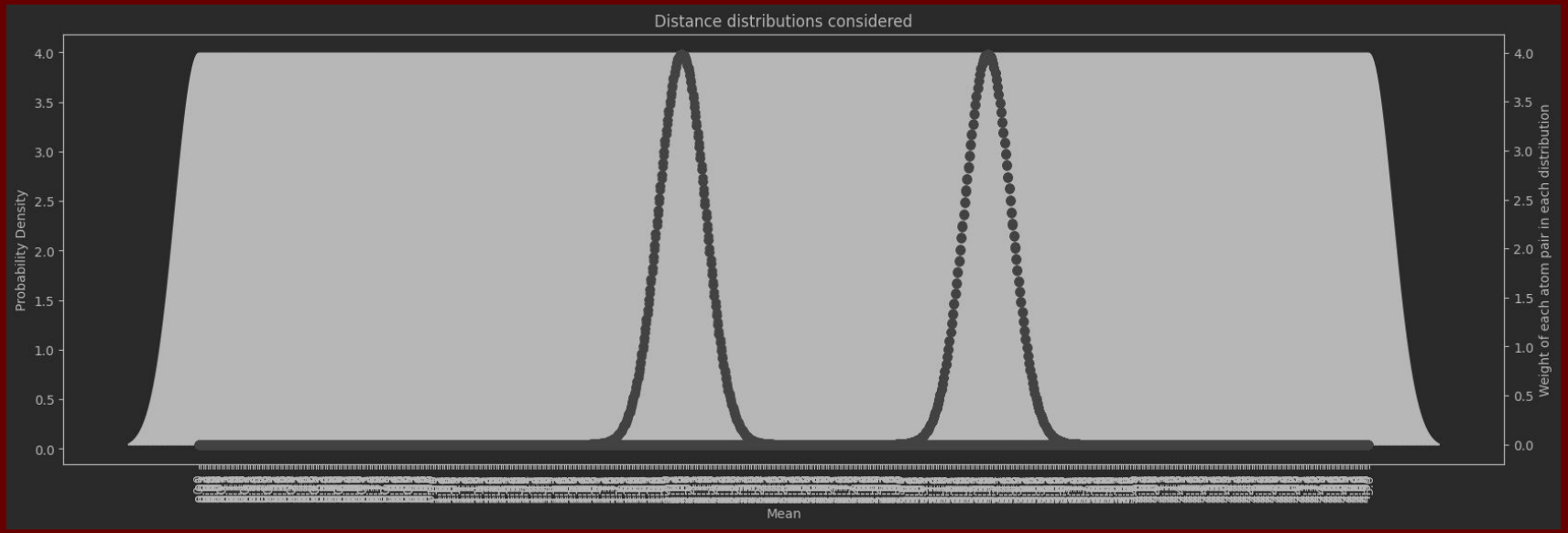
Visualizing ϕ^A for CH4:



With our soft encoding we have a peak on the distributions with mean 2.07 and 3.39, which seems to match the empirical data.

Molecule Representation based on Pairwise Distance between atoms (Soft Indicator)

We further experimented with different parameters.



Here, we chose the intervals much smaller and got more continuous soft indicator scores.

Molecule Representation based on Atom Bonds

Building ϕ^{B}:

We have five unique type of atoms, H, C, N, O and S. Therefore, there are 15 different type of unique bonds (unordered) between them:

```
['HH', 'HN', 'HO', 'HS', 'CH', 'CC', 'CN', 'CO', 'CS', 'NN', 'NO', 'NS', 'OO', 'OS', 'SS']
```

```
distance_info[0]
```

```
[{'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.063549041748047},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635344982147217},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635828971862793},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0651566982269287},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.37018084526062},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3701982498168945},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706564903259277},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370192527770996},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706531524658203},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706717491149902}]
```

After trial and error, we decide if the pairwise distance between two atoms are less than 2.5 unit, they have a bond. For example, in CH4, there are four bonds between C and H but no bonds between the H atoms.

Molecule Representation based on Atom Bonds

Building ϕ^B:

(Example for CH4)

```
distance_info[0]
```

```
[{'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.063549041748047},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635344982147217},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0635828971862793},
 {'Molecule': 0, 'Atom Pair': 'C-H', 'Distance': 2.0651566982269287},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.37018084526062},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370198249816894 5},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706564903259277},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370192527770996},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.370653152465820 3},
 {'Molecule': 0, 'Atom Pair': 'H-H', 'Distance': 3.3706717491149902}]
```

HH	HN	HO	HS	CH	CC	CN	CO	CS	NN	NO	NS	OO	OS	SS
[[0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	1,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0],
[0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0]]

$\phi^A * \phi^B$

To improvise the regression model, now we merge both ϕ^A and ϕ^B

100 rows | 100 rows × 15 columns | np.ndarray

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
23	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
26	0.0	0.0	0.0	0.0	0.00020	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
27	0.0	0.0	0.0	0.0	0.00303	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28	0.0	0.0	0.0	0.0	0.03038	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
29	0.0	0.0	0.0	0.0	0.21217	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	0.0	0.0	0.0	1.03385	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
31	0.0	0.0	0.0	0.0	3.51472	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
32	0.0	0.0	0.0	0.0	8.33657	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
33	0.0	0.0	0.0	0.0	13.79578	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
34	0.0	0.0	0.0	0.0	15.92819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
35	0.0	0.0	0.0	0.0	12.83063	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
36	0.0	0.0	0.0	0.0	7.21092	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
37	0.0	0.0	0.0	0.0	2.82746	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
38	0.0	0.0	0.0	0.0	0.77351	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
39	0.0	0.0	0.0	0.0	0.14763	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40	0.0	0.0	0.0	0.0	0.01966	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
41	0.0	0.0	0.0	0.0	0.00183	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
42	0.0	0.0	0.0	0.0	0.00012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
43	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
44	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The product feature map has shape of 100x15

- 100 is the dimension from ϕ^A - 100 intervals
- 15 is the dimension of ϕ^B - 15 distance atom bonds

Here is an example of CH4, where the feature values are high for (34,4) where 34th interval represents the closest bond distance in the molecule - CH

$\phi^A * \phi^B$ - Ridge Regression - again!

WIth the updated feature map of ϕ^A and ϕ^B we again run the ridge regression to model the atomization energy.

```
ridge_pair = Ridge(alpha=lambda_val,fit_intercept=False)
ridge_pair.fit(X_train_scaled, y_train_scaled)
```

Executed at 2023.07.12 15:05:55 in 371ms

Ridge

```
Ridge(alpha=0.01, fit_intercept=False)
```

```
print(f"Average error in predicted and actual atomization energy: {round((ridge_result_on_y_test_scaled - y_test_scaled).mean(),4)}")
```

Executed at 2023.07.12 20:13:07 in 64ms

```
Average error in predicted and actual atomization energy: 0.0165
```

Pairwise Potentials

The atomisation energy that we modelled now is a function of both atomic bond distance and the bond type.

```
1  X_one_molecule = flattened_X[0]
2  molecule_relevance_by_each_feature = ridge_pair_weights * np.array(X_one_molecule) #=w·phiA*phiB
3  molecule_relevance_by_each_feature = molecule_relevance_by_each_feature.reshape(phi_A_phi_B.shape)
4  molecule_relevance_by_each_feature.shape
```

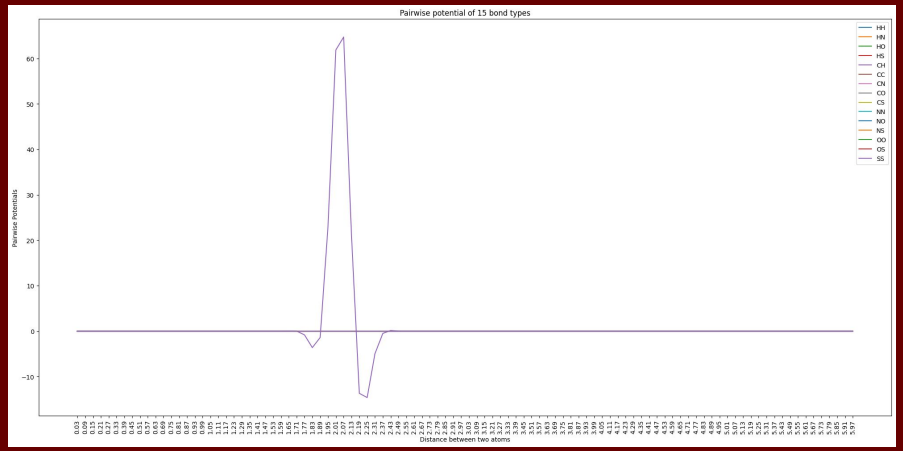
Executed at 2023.07.12 15:05:56 in 107ms

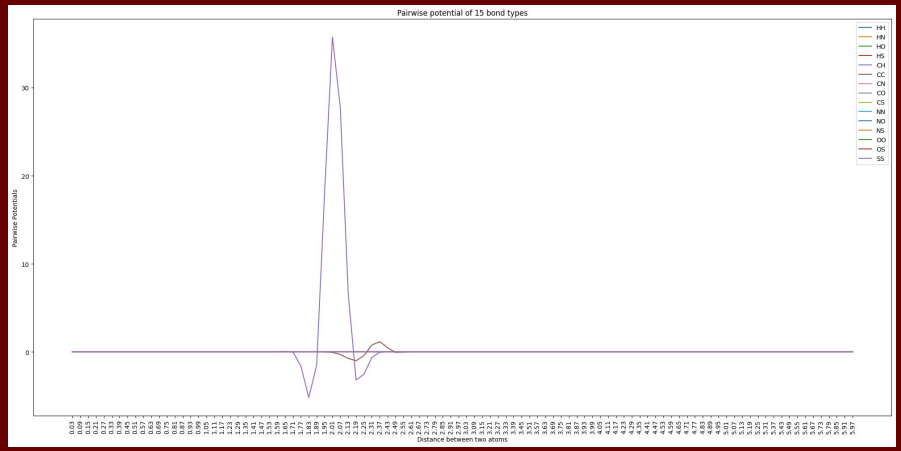
```
(100, 15)
```

To visualize it further, we calculate the relevance by each feature

- Multiply ridge regression weights/parameters with the atom representation
- Reshape the result to get a matrix where each value is a function of both atom bond distance and bond type

Pairwise Potentials





First figure is for CH4, where the atomization energy required to break the bond is higher around the distance 2.06 units, which is roughly the actual distance between C and H.

Second figure is C2H6, where the distance is 2.05 between C-H and 2.52 between C-C, since the threshold we have taken is 2.5

If the distance is higher, it probably does not take much energy or releases energy for the bond to break.

Lab ML For Data Science: Part III

Getting Insights into Images and their Metadata

1.

The Dataset

Dataset Description:

We work with image data of leaves, which are of two types : “Healthy” and “Black Rot”

Sample image from class “Black Rot”

Sample image from class “Healthy”

2.

Pretrained Models for Image Recognition

Preprocessing and calculating the direction of difference of means

- 200 images for each class in the train data and
- 50 images for each class in the test data.
- Transform the data and resize it as required for modelling.
- The preprocessed image tensor is passed through the features part of the pretrained VGG-16 model.
- The shape of features: [1, 512, 7, 7].
- Flatten the features to calculate the mean of each class given by the formula:

$$\boldsymbol{\mu}_1 = \frac{1}{|\mathcal{C}_1|} \sum_{i \in \mathcal{C}_1} \Phi(\boldsymbol{x}_i)$$

$$\boldsymbol{\mu}_2 = \frac{1}{|\mathcal{C}_2|} \sum_{i \in \mathcal{C}_2} \Phi(\boldsymbol{x}_i)$$

- The size of flattened mean: 25088.
-

The direction w of difference of means is given by:

$$\boldsymbol{w} = \frac{\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1}{\|\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\|}$$

Then we calculate the discriminant function giving us the score for any instance

$$g(\boldsymbol{x}) = \boldsymbol{w}^\top \Phi(\boldsymbol{x})$$

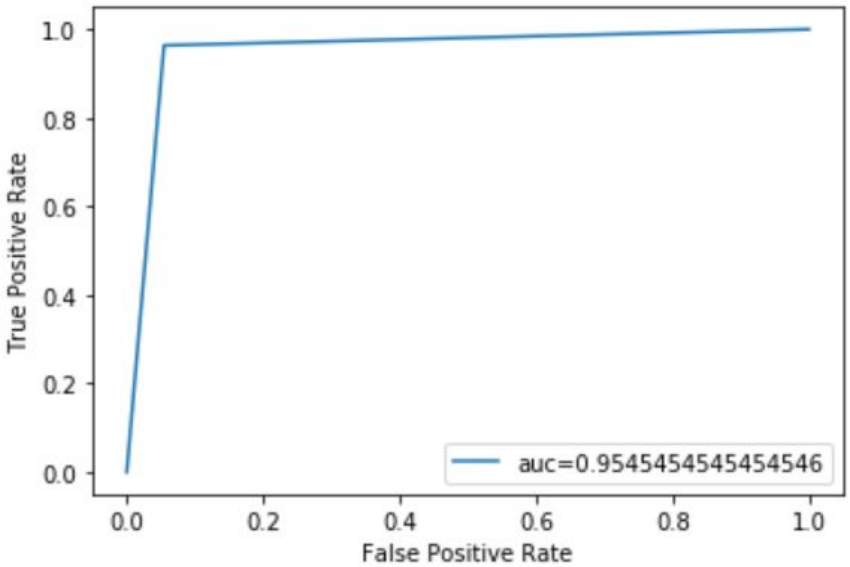
3.

Predicting Classes From Images

Computing the AUC score

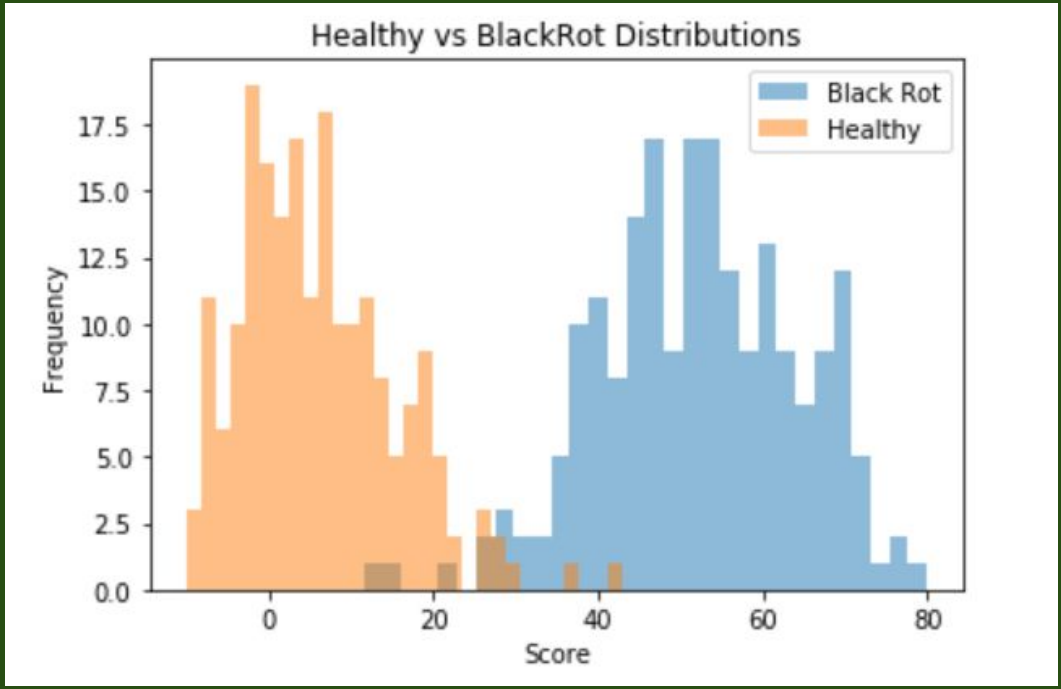
```
Threshold: 15.0
AUC Score: 0.8818181818181818
Threshold: 17.22222222222222
AUC Score: 0.9090909090909092
Threshold: 19.444444444444443
AUC Score: 0.9181818181818182
Threshold: 21.666666666666668
AUC Score: 0.9272727272727271
Threshold: 23.88888888888889
AUC Score: 0.9272727272727272
Threshold: 26.11111111111111
AUC Score: 0.9545454545454545
Threshold: 30.555555555555557
AUC Score: 0.9545454545454546
Best Threshold: 30.555555555555557
Best AUC Score: 0.9545454545454546
```

```
The number of instances used for class Healthy:  56
The number of instances used for class Black Rot:  54
```



We computed the optimum AUC score by iterating over a range of thresholds separating the 2 classes.
The AUC curve for the best threshold is as shown above.

Classification based on scores



We classify the instances by comparing their scores to the threshold value.

4.1.

Sensitivity Analysis

Getting Pixel Wise Contributions:

We now apply our model on a test image and understand which pixels contribute the most to the classification of the image.

```
# Preprocess the input image
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Load and preprocess the image
image_path = 'data_TAD/test/Black_Rot/image (542).JPG'
image = Image.open(image_path).convert('RGB')
input_tensor = preprocess(image)
input_batch = input_tensor.unsqueeze(0)
print(input_batch.shape)
```

```
torch.Size([1, 3, 224, 224])
```

We first preprocess the image, so we have 224 * 224 pixels for each of the three channels (R, G, B)

We then apply the existing model to get an output.

Getting Pixel Wise Contributions:

```
# Calculate the gradients
output[0, predicted_idx].backward()

# Get the gradients of the input tensor
gradients = input_batch.grad[0]

# print("input_batch",input_batch)
print("input_batch shape",input_batch.shape)
# print("gradients",gradients)
print("gradients shape",gradients.shape)
```

```
input_batch shape torch.Size([1, 3, 224, 224])
gradients shape torch.Size([3, 224, 224])
```

```
importance_scores = torch.norm(gradients, p=2, dim=0,keepdim=False)**2

importance_scores.shape
```

```
torch.Size([224, 224])
```

```
np.round(importance_scores, decimals=4)
```

```
tensor([[1.0000e-04, 5.0000e-04, 7.0000e-04,  ..., 1.9000e-03, 3.0000e-04,
         1.0000e-04],
        [0.0000e+00, 5.0000e-04, 3.6000e-03,  ..., 5.2000e-03, 6.0000e-04,
         1.0000e-04],
        [1.0000e-04, 5.1000e-03, 1.4000e-03,  ..., 2.4000e-03, 3.0000e-04,
         1.0000e-04],
        ...,
        [0.0000e+00, 3.0000e-04, 4.1000e-03,  ..., 1.0000e-03, 4.0000e-04,
         1.0000e-04],
        [1.9000e-03, 2.1000e-03, 3.6000e-03,  ..., 8.0000e-04, 7.0000e-04,
         0.0000e+00],
        [3.0000e-04, 2.0000e-03, 1.0000e-03,  ..., 2.0000e-04, 1.0000e-04,
         4.0000e-04]])
```

We then calculate the gradient vector across the three channels (R,G,B).

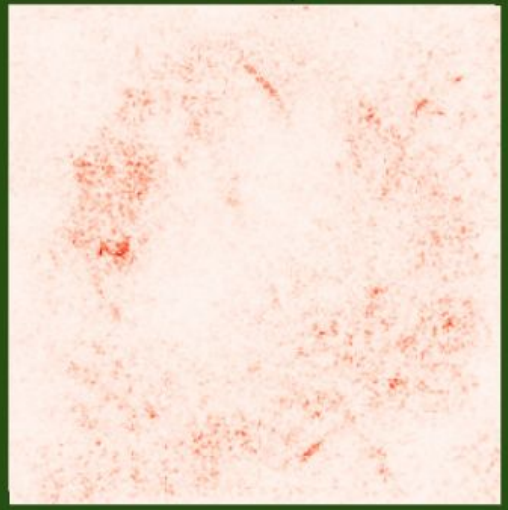
Then we accumulate the gradient vector along each of our three directions (R,G,B) by calculating their square norm, so we can have a final importance score of size 224 * 224

These scores represent the importance of each pixel in the input image.

Getting Pixel Wise Contributions:

Original Image

Heatmap



To visualize the importance scores, we normalize them and convert them to a heatmap image.

The heatmap image highlights important features in the original image.

However, sensitivity analysis using gradients tends to produce noisy and not fully satisfactory explanations.

4.2.

More Robust Explanations

Robustify gradient-based explanations

To robustify the features , we now focus on excitatory effects and less on inhibitory effects in the network.

$$z_k = \left(\sum_j a_j w_{jk}^{\uparrow} + b_k^{\uparrow}\right) \cdot \left[\frac{\sum_j a_j w_{jk} + b_k}{\sum_j a_j w_{jk}^{\uparrow} + b_k^{\uparrow}}\right]_{\text{cst.}}$$

$$w_{jk}^{\uparrow} = w_{jk} + 0.25 \max(0, w_{jk})$$
$$b_k^{\uparrow} = b_k + 0.25 \max(0, b_k).$$

- To achieve this, we rewrite specific layers in a way that the forward function remains the same locally but the gradient is modified to implement the asymmetry.
- By biasing the gradient in this way, the network tends to prioritize the learning of features and patterns that have a positive impact on the task at hand, while downplaying the influence of features that may hinder its performance.

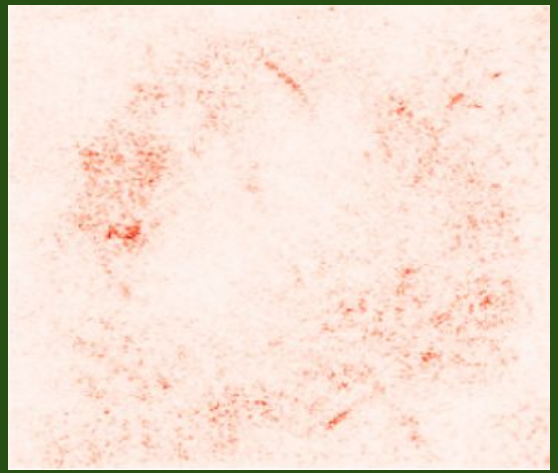
Excitatory effects over inhibitory effects

```python
class BiasedLayer(nn.Module):
    def __init__(self, original_layer,gamma:float):
        super(BiasedLayer, self).__init__()

        # Clone the original layer
        self.layer = original_layer
        self.gamma = gamma
        # self.biased_layer = self.clone_layer_with_bias(original_layer)
        self.biased_layer = deepcopy(self.layer)
        self.biased_layer.bias = nn.Parameter(self.layer.bias+ self.gamma * torch.relu(self.layer.bias))
        self.biased_layer.weight = nn.Parameter(self.layer.weight + self.gamma * torch.relu(self.layer.weight))


    def forward(self, x):

        original_output = self.layer(x)
        biased_output = self.biased_layer(x)
        biased_output_detached = biased_output.detach()
        original_output_detached = original_output.detach()

        scaling_factor = original_output_detached / biased_output_detached

        output = biased_output * scaling_factor

        return output

#
# Modify the VGG-16 model by replacing certain layers with biased versions
modified_model = model_vgg16_features
for name, module in modified_model.named_children():
    if isinstance(module, nn.Linear) or isinstance(module, nn.Conv2d):
        modified_model._modules[name] = BiasedLayer(module,gamma=0.25)
```

- As explained before, we modify the layers in a way that we get the gradient which favors the excitatory effects keeping the output same as the original VGG features model.

```
Sequential(
  (0): BiasedLayer(
    (layer): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (biased_layer): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (1): ReLU(inplace=True)
  (2): BiasedLayer(
    (layer): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (biased_layer): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): BiasedLayer(
    (layer): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

Feature comparisons



- The initial heatmap of the image is noisy while after modifying the gradient, we get a sharper image which accentuates the boundaries/features of the leaf. Although considering the task at hand, this does not distinguishes the part which has the disease.
- One idea to improve is to have an semantic segmentation model to identify individual objects within an image and assign a separate mask to each object.

Thank You!