Mushroom Classification

Junjie Ge, Kejian Shi {jg5324,ks4765}@nyu.edu

I. Introduction

In this study we obtain the "mushroom" dataset presented by UCL Machine Learning. The problem is a binary classification task that aims to classify the mushrooms as either Edible (e) or Poisonous (p) using the 22 features including shape, gill color, size, etc. We first observe the data to have an overall picture and try to seek for meaningful patterns using some unsupervised techniques. Next, after we preprocess the data, we set up a comprehensive set of supervised experiments to see what types of machine learning models perform the best on the data. Our methodologies and the corresponding results are presented from section II to III, and we present the overall results and discussion in section IV. Specifically, we note that all of our models of Logistic Regression, SVM, and Neural Network have achieved > 95% accuracy.

II. Data Handling and Unsupervised Analysis

Observe and Preprocess the Data

Data: We first read the .csv file into a DataFrame and use basic DataFrame attributes to look at the data. The data is of shape (8124, 23). Figure below shows a snapshot of the structure of the data by taking the first 10 rows (samples).

As indicated in the figure, the "class" column will be the dependent variable, namely "y" which is the class type we are going to predict. Other 22 columns

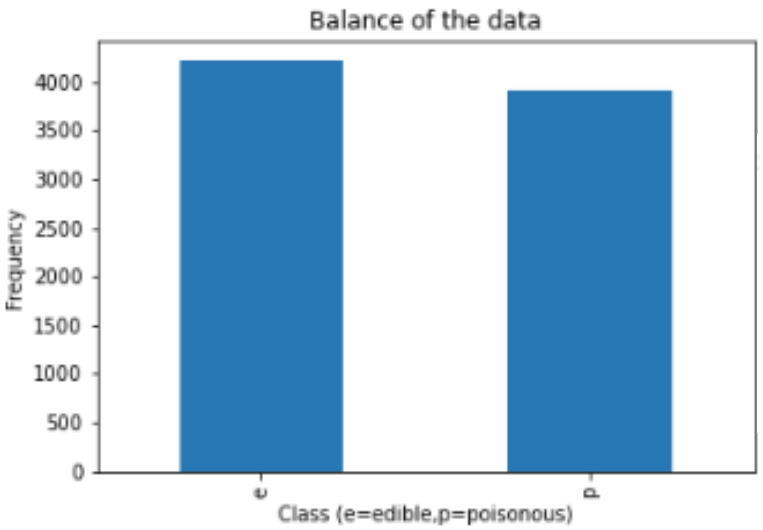
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w
5	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w
6	e	b	s	w	t	a	f	c	b	g	...	s	w	w	p	w
7	e	b	y	w	t	l	f	c	b	n	...	s	w	w	p	w
8	p	x	y	w	t	p	f	c	n	p	...	s	w	w	p	w
9	e	b	s	y	t	a	f	c	b	g	...	s	w	w	p	w

10 rows × 23 columns

In the data, there are 23 columns, including the label column and 22 features.

are independent variables, namely "X". Next, we further explore the data and the features by

1) Check for the balancedness of the data by counting the number of each label (p=poisonous and e=edible) occurred in the sample.




```
plt.figure()
series = pd.Series(df['class'])
counts = series.value_counts().sort_index()
plot = counts.plot(kind='bar')
plt.xlabel('Class (e=edible,p=poisonous)')
plt.ylabel('Frequency')
plt.title('Balance of the data')
```

2) Try to find patterns or anomalies in the features.

By printing out the counts of different values in each feature, we can see that the feature "veil-type" has only single value of "p", which makes it irrelevant in the classification problem. So, we will drop the column of veil-type.

```
for feature in df.columns:
    print(feature)
    print(df[feature].value_counts())
    print('=======\n')
```

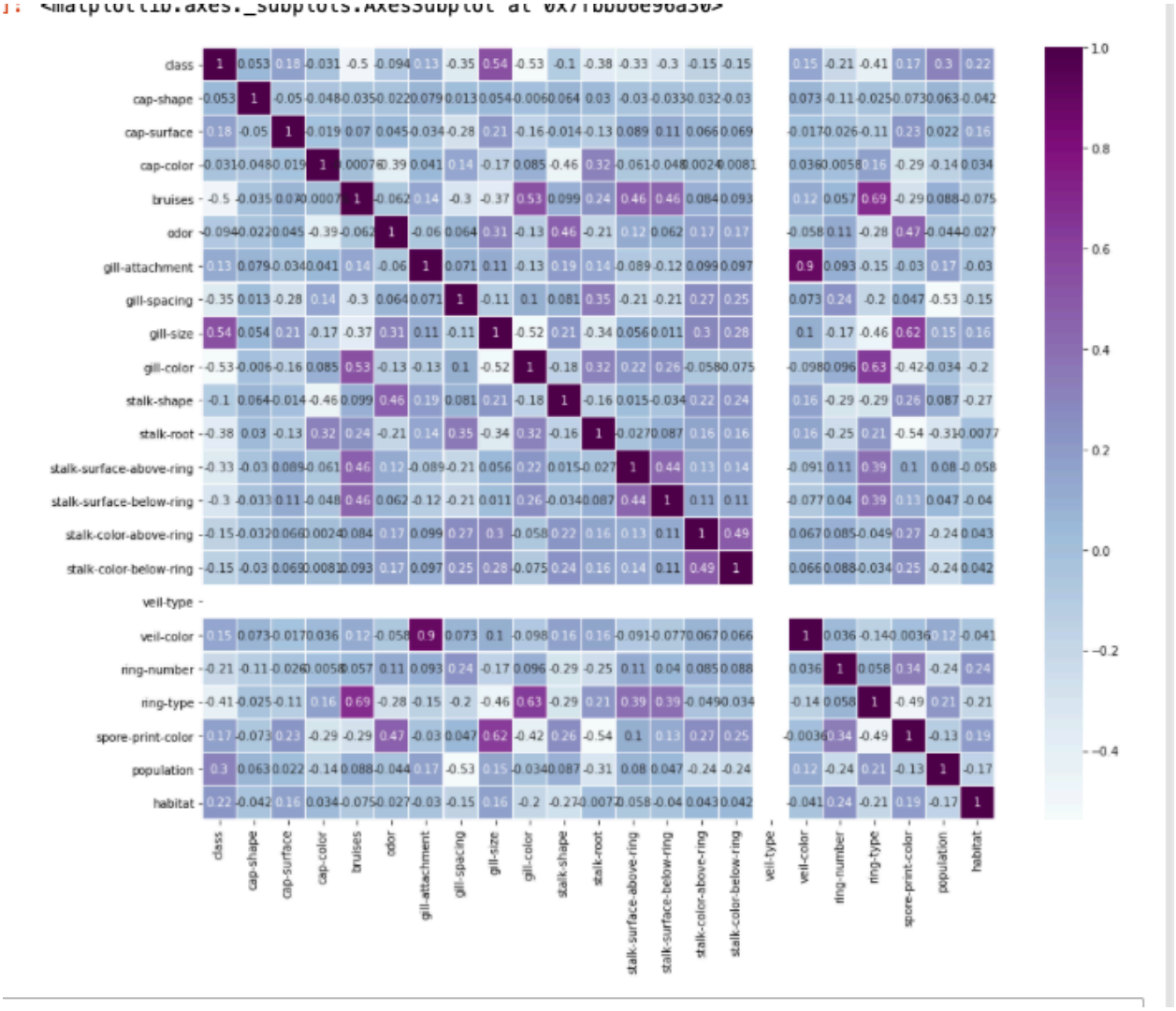
```
stalk-color-below-ring
w    4384
p    1872
g     576
n     512
b     432
o     192
e      96
c      36
y      24
Name: stalk-color-below-ring, dtype: int64
=======

veil-type
p    8124
Name: veil-type, dtype: int64
=======
```

Encoding: Next, we encode the data because the original data is non-numerical. In this case, we convert the categorical values to numbers using one-hot encoding provided by Sklearn library.

Correlation between Features

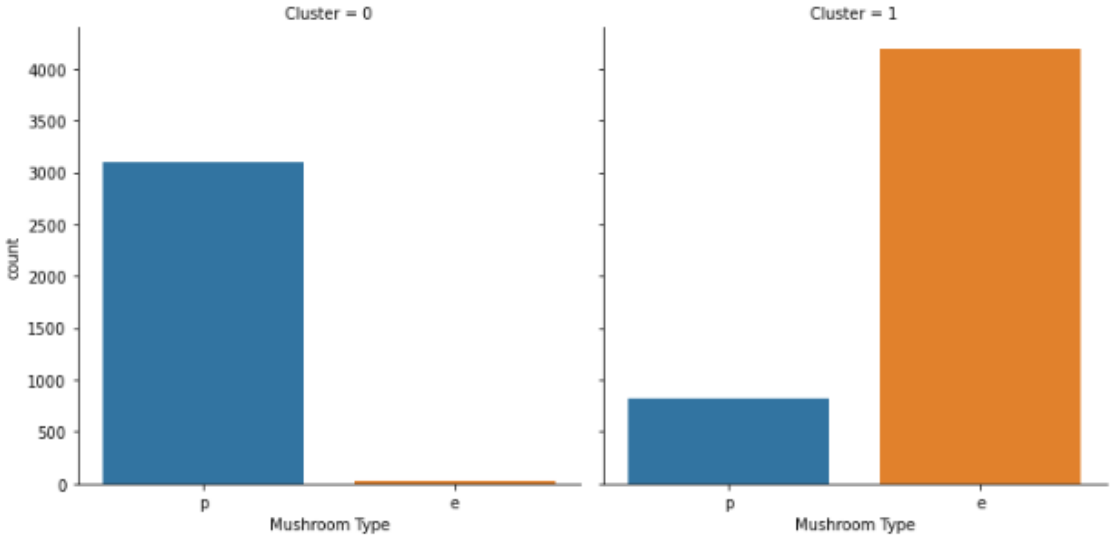
We then examined the correlation between the features using seaborn



We can see the feature "gill-color" might be the most useful feature for classification because it has the least correlation value of -0.53, making a "distinguishable" among all features.Also, by observing the correlation heatmap, we confirm that "veil-type" has no correlation with any other features. So, we drop this column in our future experiment.

K-mean Clustering

We then tried using K-mean to cluster the data into two distinguishable clusters. The results showed that One cluster contains mostly poisonous mushrooms and the other contains mostly edible ones.



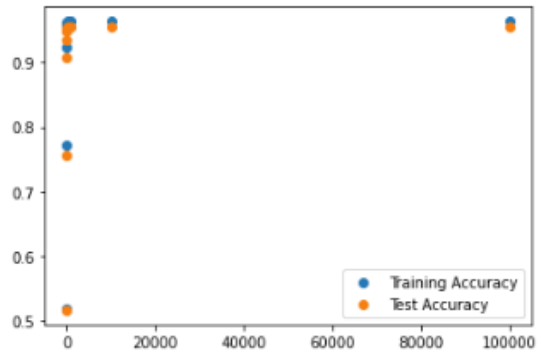
III. Supervised Analysis

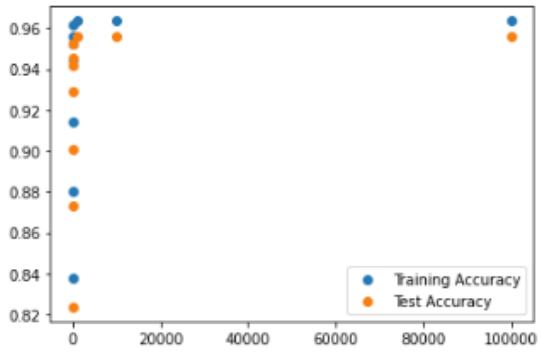
In this section, we illustrate our supervised experiment with three types of models: Logistic Regression, Support Vector Machines, and Neural Networks, with different parameter selections and feature transformations. We first split the data into training and testing set. For the majority of the experiments, unless specified otherwise, the ratio is 4:1 for training:testing.

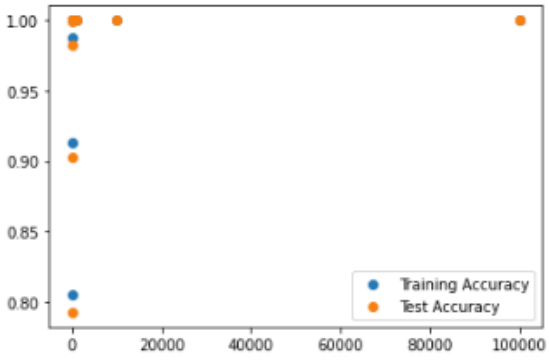
Logistic Regression

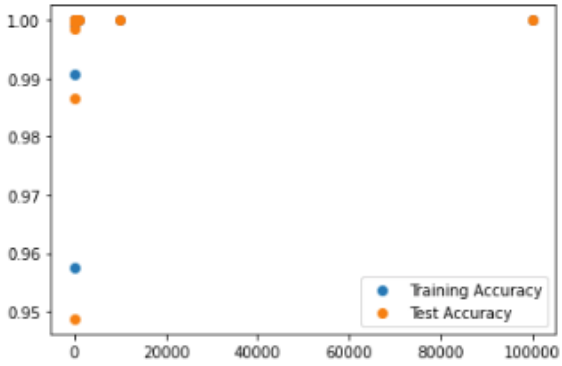
In Logistic Regression, we try to find a hyperplane that is mostly likely to have generated the dataset. We first use vanilla logistic regression without any handling of the data, which we managed to get 96% accuracy. Then we tried different regularization techniques (Ridge and Lasso) with varying c = $\frac{1}{\lambda}$ values to compare and to prevent overfitting. Lastly, we add feature transformation to the regression, using L1 and L2 each time. To our surprise, feature transformation with polynomial = 2 and L2 regularization almost gets 100% accuracy. We suspected that it is because the original data points formed a near quadratic surface. And the quadratic surface original feature space become linear in the transformed

space. We have produced four figures of Training/Testing accuracy and a table of results that include 1) Regression with L1 2) Regression with L2 3) Feature transformed (p=2) regression with L1 and 4) Feature transformed (p=2) regression with L2. The following four figures are arranged left to right, up to down as 1)2)3)4).





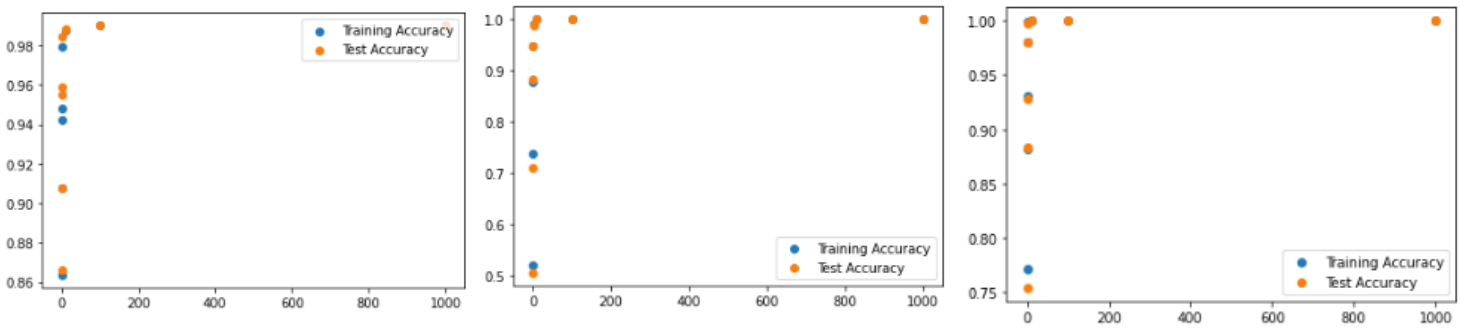




	0.0001	0.0010	0.0100	0.1000	1.0000	10.0000	100.0000	1000.0000	10000.0000	100000.0000
Training L1	0.518300	0.771213	0.923026	0.949943	0.957656	0.962580	0.963401	0.963565	0.963565	0.963565
Test L1	0.516987	0.756770	0.907927	0.935500	0.947317	0.953225	0.955687	0.955687	0.955687	0.955687
Training L2	0.837847	0.880519	0.914000	0.944855	0.952240	0.956015	0.961595	0.963401	0.963565	0.963565
Test L2	0.823732	0.872969	0.900542	0.929099	0.941901	0.945347	0.952240	0.955687	0.955687	0.955687
Train FT L1	0.805186	0.912851	0.988019	0.999836	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test FT L1	0.792713	0.902511	0.982275	0.999015	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Train FT L2	0.957656	0.990809	0.999179	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Test FT L2	0.948794	0.986706	0.998523	0.999508	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Support Vector Machines

In this section we compare hyperplanes of different variant of Support Vector Machines, illustrating how weights and errors has changed when we choose different parameters or regularization. Particularly, we used linear kernel, rbf kernel, and polynomial kernel, and for each one we tried to select the best hyperplane by varying the parameter C in sklearn.SVC(). The parameter controls how much to penalize model for misclassification. A high C parameter could give the model more freedom to select more samples as support vectors and allow few misclassifications while a low C makes a boundary smooth.

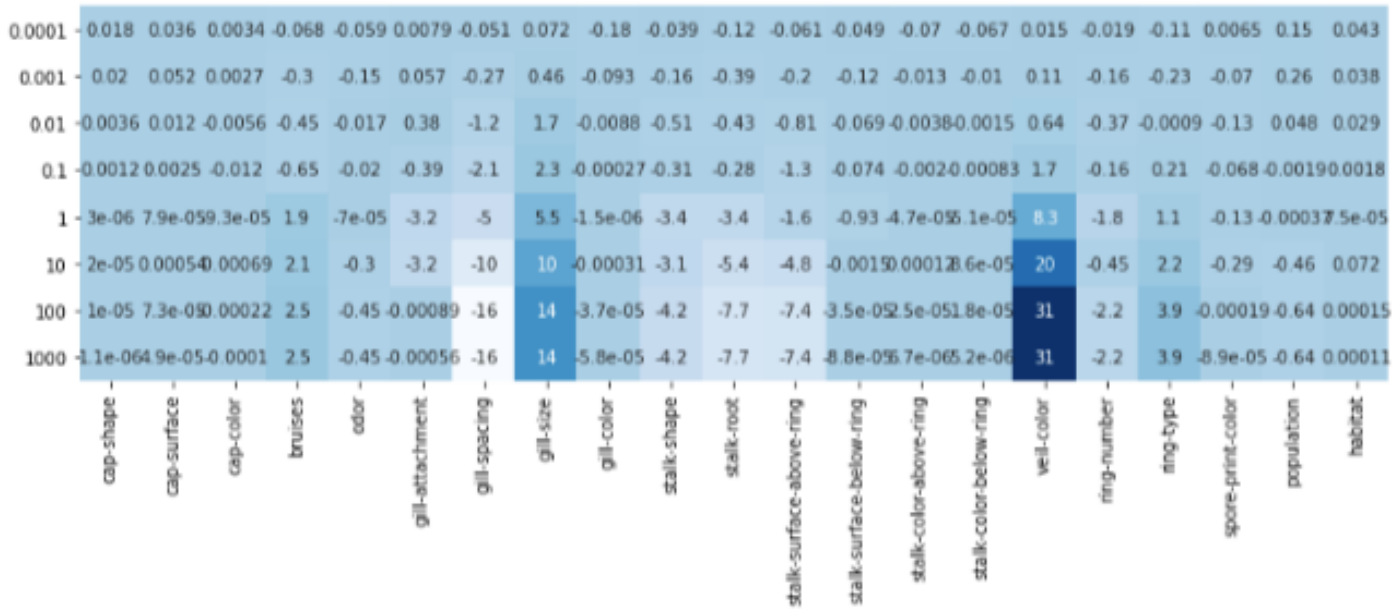


Figures: Training/Testing Accuracy with linear kernel, rbf kernel, and polynomial kernel

	0.0001	0.0010	0.0100	0.1000	1.0000	10.0000	100.0000	1000.0000
Training Linear	0.860167	0.905629	0.946004	0.951748	0.968653	0.987527	0.991466	0.989824
Test Linear	0.862629	0.906942	0.942885	0.947809	0.968981	0.989168	0.992122	0.991137
Training RBF	0.521910	0.737404	0.879206	0.948465	0.990973	1.000000	1.000000	1.000000
Test RBF	0.506155	0.710487	0.882324	0.947809	0.989168	1.000000	1.000000	1.000000
Training Poly	0.771869	0.882160	0.930904	0.980141	0.999015	1.000000	1.000000	1.000000
Test Poly	0.754308	0.883309	0.927622	0.979813	0.997538	1.000000	1.000000	1.000000

We can observe that for rbf kernel and polynomial kernel with C > 1, the accuracy converges to 100%. This means that a c value of 1 would be enough to penalize misclassified points. The following partial table show how the weights changed with varying c. (The full table can be accessed in the code)

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	...
0.000	0.0184	0.0361	0.00335	-0.0681	-0.0585	0.007900	-0.05064	0.07166	-0.1842	-0.0386	...
0.001	0.0196	0.0516	0.00266	-0.2950	-0.1549	0.057021	-0.27359	0.45983	-0.0929	-0.1587	...
0.01	0.0035	0.0122	-0.0056	-0.4480	-0.0166	0.380415	-1.16430	1.70892	-0.0088	-0.5058	...
0.1	0.0011	0.0024	-0.0116	-0.6519	-0.0198	-0.38903	-2.08445	2.33257	-0.0002	-0.3083	...
1	0.0000	0.0000	-0.0000	1.88897	-0.0000	-3.17984	-4.96520	5.49276	-0.0000	-3.3614	...
10	0.0000	0.0005	-0.0006	2.12447	-0.2957	-3.21226	-10.2932	9.97291	-0.0003	-3.0913	...
100	0.0000	0.0000	-0.0002	2.45480	-0.4543	-0.00088	-15.8986	13.7204	-0.0000	-4.1809	...
1000	0.0000	0.0000	-0.0001	2.45444	-0.4543	-0.00056	-15.9011	13.7217	-0.0000	-4.1810	...

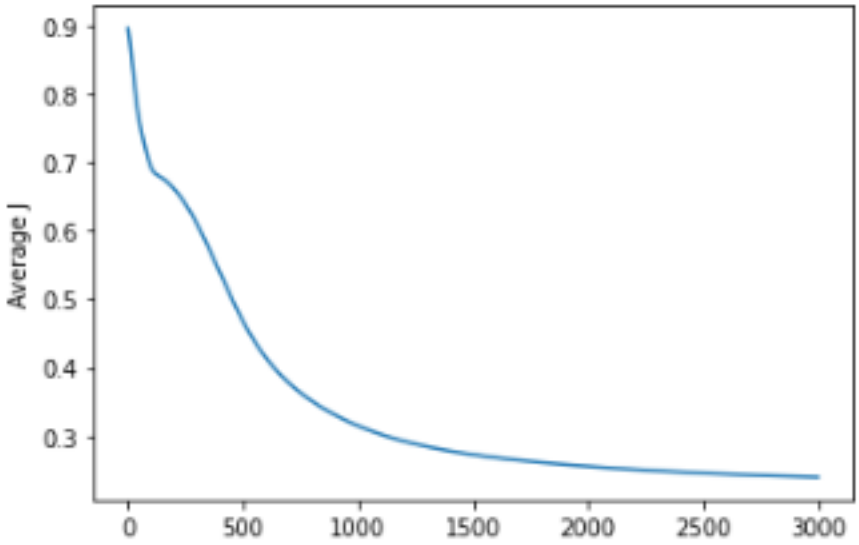


Neural Networks

In this section we apply the vanilla neural network model implemented in class with initial weights set to **0.** However, we found out using all 21 features as inputs impose computational burden on the machine. So for neural networks, we have reduced the number of inputs to the first 8 features. Even though the input size is reduced, tuning with different hidden layer sizes, alpha, and lambda has gave high accuracy.

We first encode the data, because we need to convert the categorical value to numerical ones. Particularly, we have converted label e=edible and p=poisonous to 0 and 1. Afterwards we scale the features to help us see the output, even though it is not necessary.

We design the neural network structure as [8, X, 2], where X represents the number of neurons the hidden layer. To have a quick look, we tested the model using network [8,6,2] and alpha=0.25, lambda(L2) = 0.001, iter_num = 3000. The following figure shows the average weights change increasing iteration number.




```
Prediction accuracy is 93.35384615384615%
```

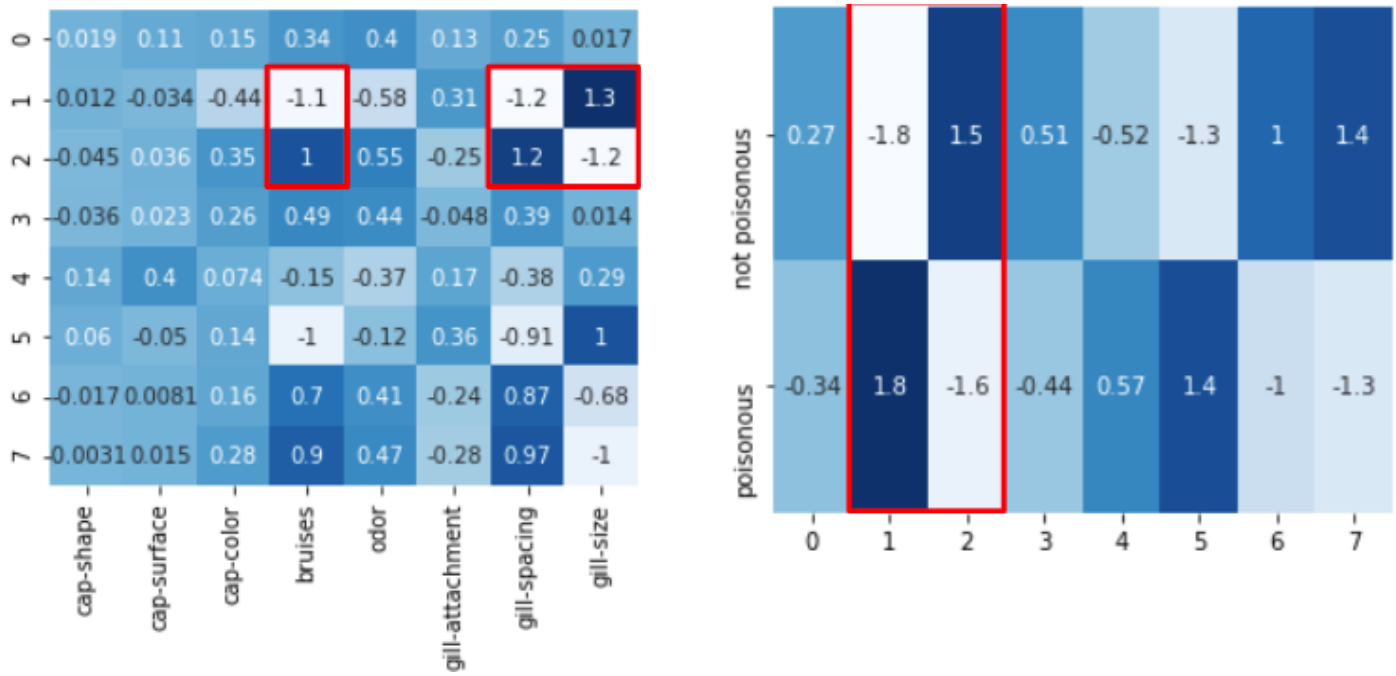
Moreover, we set up a parameter selecting function that attempts all combinations from lamb = [0.001,0.002,0.1], hidden_size = [4,8,12], alpha = [0.01,0.05, 0.1, 0.25, 0.5], and activation = [‘sigmoid’,’relu’,’tanh’]. The

results are given in the figure below (in consideration of length of the writeup, I omit the graphs for variations with ReLU and tanh activations, as they overall appear to be less applicable than sigmoid)

	Accuracy	Hidden-layer Size	Alpha	Lambda
0	73.292308	4.0	0.01	0.001
1	63.692308	4.0	0.01	0.002
2	52.615385	4.0	0.01	0.100
3	88.184615	4.0	0.05	0.001
4	91.692308	4.0	0.05	0.002
5	52.615385	4.0	0.05	0.100
6	93.415385	4.0	0.10	0.001
7	93.476923	4.0	0.10	0.002
8	52.615385	4.0	0.10	0.100
9	93.415385	4.0	0.25	0.001
10	93.415385	4.0	0.25	0.002
11	52.615385	4.0	0.25	0.100
12	93.476923	4.0	0.50	0.001
13	93.415385	4.0	0.50	0.002
14	52.615385	4.0	0.50	0.100
15	71.200000	8.0	0.01	0.001
16	52.615385	8.0	0.01	0.002
17	52.615385	8.0	0.01	0.100
18	90.769231	8.0	0.05	0.001
19	93.046154	8.0	0.05	0.002
20	52.615385	8.0	0.05	0.100
21	93.415385	8.0	0.10	0.001
22	93.415385	8.0	0.10	0.002
23	52.615385	8.0	0.10	0.100
24	93.476923	8.0	0.25	0.001
25	93.415385	8.0	0.25	0.002
26	52.615385	8.0	0.25	0.100
27	93.907692	8.0	0.50	0.001
28	93.415385	8.0	0.50	0.002
29	52.615385	8.0	0.50	0.100
30	48.861539	12.0	0.01	0.001
31	50.030769	12.0	0.01	0.002
32	52.615385	12.0	0.01	0.100
33	90.830769	12.0	0.05	0.001
34	93.723077	12.0	0.05	0.002
35	52.615385	12.0	0.05	0.100
36	93.415385	12.0	0.10	0.001
37	93.415385	12.0	0.10	0.002
38	52.615385	12.0	0.10	0.100
39	93.415385	12.0	0.25	0.001
40	93.415385	12.0	0.25	0.002
41	52.615385	12.0	0.25	0.100
42	93.415385	12.0	0.50	0.001
43	93.415385	12.0	0.50	0.002
44	52.615385	12.0	0.50	0.100

To conclude, the accuracies vary greatly (52% - 93%) . And the best tuning with a(z) = sigmoid, Hidden-layer size 8, $\alpha = 0.5$, iteration=3000, and $\lambda = 0.001$ achieves the highest accuracy of 93.91. The following graphs illustrate the weights and the impacts on the classification using heat map.

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size
0	0.018535	0.114422	0.145713	0.339548	0.402305	0.126600	0.253024	0.017056
1	0.011766	-0.034104	-0.442951	-1.139644	-0.575570	0.313861	-1.194285	1.328189
2	-0.045069	0.036132	0.347126	1.005224	0.551068	-0.248267	1.188486	-1.176447
3	-0.035600	0.023113	0.262598	0.486830	0.438311	-0.047629	0.388144	0.013907
4	0.139789	0.399026	0.073767	-0.146296	-0.370652	0.169625	-0.377714	0.289890
5	0.059810	-0.050459	0.144570	-1.020638	-0.122975	0.361693	-0.912796	1.006809
6	-0.016540	0.008061	0.163811	0.697212	0.409953	-0.239999	0.865816	-0.684508
7	-0.003140	0.014891	0.278187	0.900018	0.468183	-0.280534	0.971308	-1.022647



We note that Gill-color Gill spacing, gill size, and bruises have the most impact on the classification problem

IV. Overall Results and Discussion

For the unsupervised analysis, we particularly note that the k-mean clustering has produced two highly distinguishable clusters. That leads us to suppose the original data points form a non-linear quadratic shape. This actually also answers why our Logistic Regression with feature transformation with $p = 2$ works so well, which is because the data has been moved to a linear space from quadratic space.After comparing the results obtained by running logistic regression, SVM, and NN algorithm, we were able to draw a few conclusions based on our observations.

By looking at the results from logistic regression and SVM, we were able to conclude that the veil color is the most determining factor of whether a mushroom is poisonous or not. Specifically, in SVM, we see the weight for veil color increasing drastically as the regularization strength decreases. The other determining features is gill size and gill spacing, this is also reflected in both logistic regression algorithm and SVM. Interestingly, in logistic regression, we see that the gill attachment is the second most determining feature of the data, however this is not reflected in SVM.

Although we were not able to run NN algorithm using all available features, we were still able to achieve a high accuracy using the features we choose. We believe the reason is because we included features like gill size and gill spacing, which are two of the determining features as determined by logistic regression and SVM. By looking at the weights we obtained by running NN algorithm, we were able to confirm that this is indeed the case. Using our setting that yields the best accuracy, we can see that nodes one and two in the hidden layer has the two highest weights. And looking at the weights between the input layer and hidden layer, we indeed can see that gill spacing and gill size are the two nodes that has the biggest impact on nodes one and two in the hidden layer. Besides choosing the relevant features, we noticed that having the right set-up for NN is also crucial in achieving a high accuracy. In our result, we got accuracies from 52% to 93%. Looking that the variables (hidden layer, alpha, lambda) we see that no single variable is solely responsible for a high accuracy, it is by trial and error that we got the best variables for this specific data set.