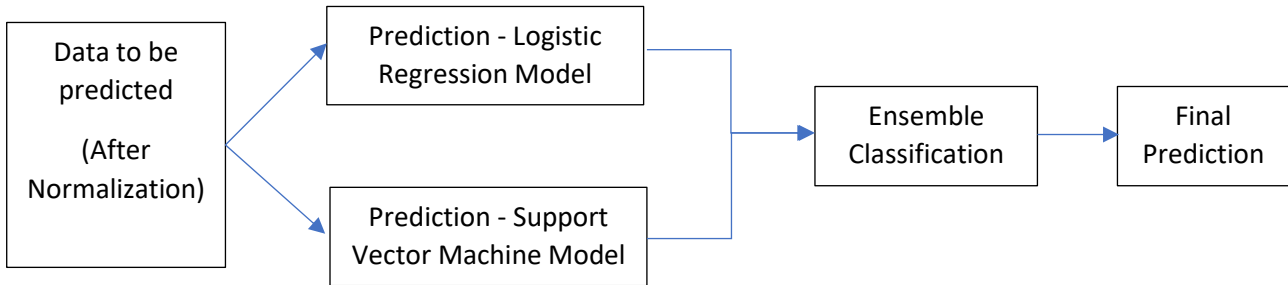


CT4101 MACHINE LEARNING - ASSIGNMENT 2

Prakhar Gurawa (20231064)

Yashitha Agarwal (20230091)

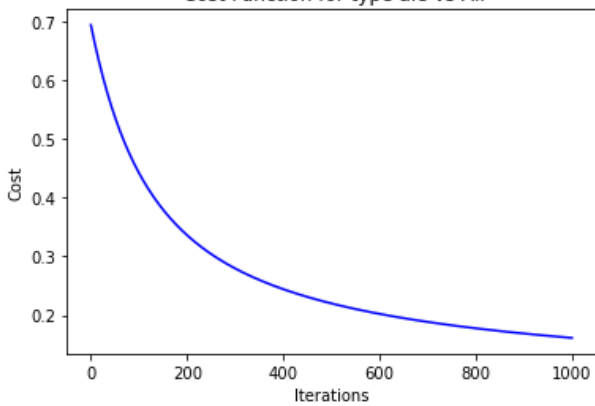
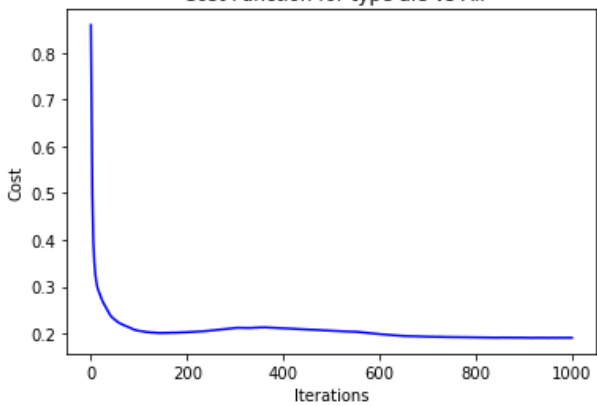
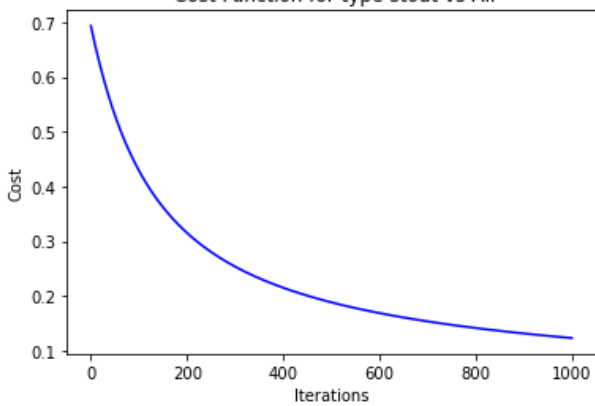
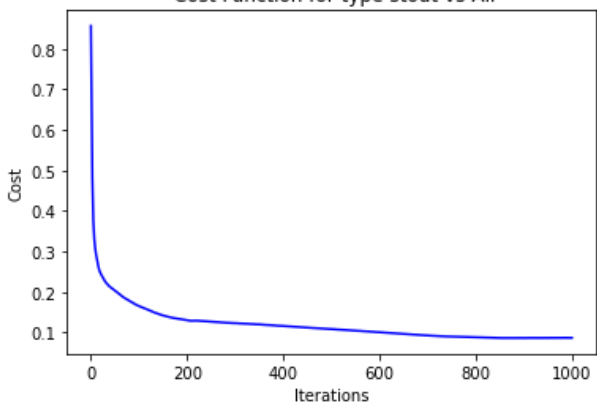
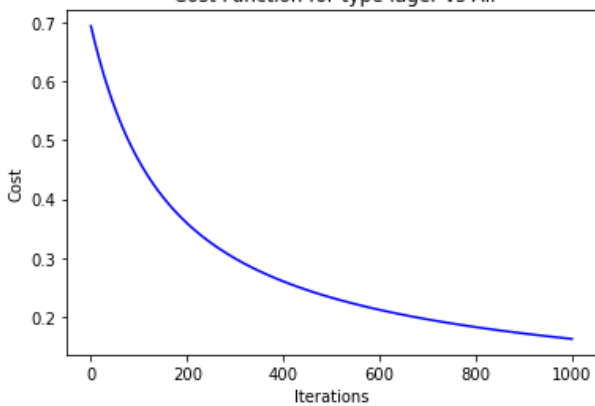
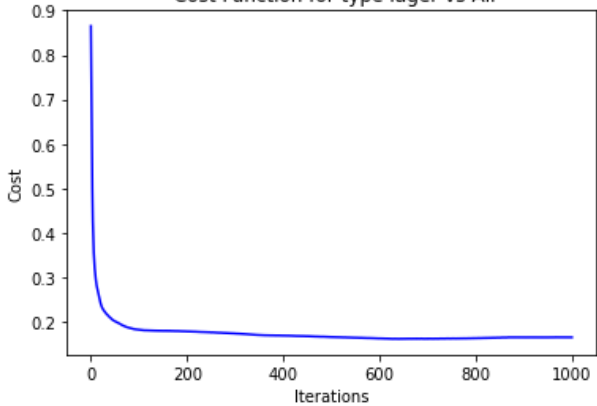
Algorithm description and design decisions: For this assignment, we have considered two machine learning models, namely Logistic Regression and Support Vector Machine classifiers, which have been implemented from scratch. Further, we have integrated both the models, based on their separate accuracy scores, to create an ensemble classification. We have also implemented standardization, which a feature scaling technique, from scratch.



- 1. Logistic Regression:** In logistic regression model, the primary aim is to maximise the probability of minimising the loss function. To achieve this, first the features with weights are passed to the sigmoid function, which returns a result between 0 and 1. The result from the sigmoid function is evaluated using the loss function and then using gradient descent, weight of the feature is modified, to the optimum value, based on the derivatives of the loss function.
- 2. Support Vector Machine:** In support vector machine model, the goal is to maximise the margin, which is the distance between the optimal hyperplane and the support vectors. The best line (hyperplane) for any given dataset separates the different classes with maximum margin. The algorithm does this by first finding the points that are closest to the line for each of the classes, called support vectors, and then calculates the distance between the line and support vectors. The line which has the maximum margin is chosen as the hyperplane.
- 3. Ensemble Classification:** The ensemble classification integrates the predictions from the Logistic Regression model and the Support Vector Machine model, which will help to get a better prediction and make the algorithm more robust. Data to be predicted is passed to the ensemble classification, where the individual prediction from Logistic Regression model and Support Vector Machine model is taken. If the individual prediction of both algorithms is the same, the final prediction is taken to be the same, otherwise if both models predict different results, the prediction from the model with higher individual score is considered as the final prediction.

Reference Implementation: The Logistic Regression model and Support Vector Machine model from scikit learn have been used as a reference implementation to assess the performance of our implemented algorithms and the observations are reported below. Additionally, the ensemble classifier has also been implemented using scikit learn and the performance has been compared with the own implementation.

Logistic Regression Model	Support Vector Machine Model
The sigmoid function has a s-shaped curve and exists between 0 and 1. Since the model needs to predict a probability as the output and probability values exist only between 0 to 1, the sigmoid function is used.	Hyperplanes are used as a decision boundary for the data points. Based on which side of the hyperplane a given data point is located, the class to which the data point belongs to is determined.
Cost Function: The loss function used by Logistic Regression model is the below Binary cross entropy loss function. $L(y, \hat{y}) = -\frac{1}{m} \sum_{i=1}^m [y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)]$	Cost Function: SVM uses Hinge loss function below. Regularization parameter is used to balance the cost function, which is set to $(1 / \text{Number of iterations})$. $\min_w w ^2 + \sum_{i=1}^n (1 - y_i \langle x_i, w \rangle)_+$

<p>Gradient descent in Logistic Regression is used to minimise the loss function. This is done by moving in the direction where the steepest (optimal) point is located on the given search landscape, using the below equation.</p> $w = w - x_i \cdot (\hat{y} - y)$	<p>Gradient update in SVM depends on whether the prediction from our model is correct or incorrect.</p> <p>In case of missclassification :</p> $w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$ <p>In case of no missclassification :</p> $w = w - \alpha \cdot (2\lambda w)$
<p>Number of iterations is set to 1000 for LR model.</p>	<p>Number of iterations is set to 1000 for SVM model.</p>
<p>Cost Function for type ale vs All</p> 	<p>Cost Function for type ale vs All</p> 
<p>Cost Function for type stout vs All</p> 	<p>Cost Function for type stout vs All</p> 
<p>Cost Function for type lager vs All</p> 	<p>Cost Function for type lager vs All</p> 

Implementation Details: Since the given problem is a multiclass classification problem, we have used the technique of one vs all in individual algorithms.

Tests: To ensure comprehensive tests are done, first the given data is imported into a dataset using pandas. The 'beer_id' column is removed from the dataset as it does not contribute towards identifying the type of beer. With the remaining dataset, we create dependent and independent (style) features, which is then split into training (two-thirds) and test (one-third) sets, shuffled over each iteration. Feature scaling is implemented for the dependent features, after which the data is fitted and scored over ten iterations, for each algorithm – Linear

Regression, Support Vector Machine and Ensemble Classifier of our own implementation and using scikit learn. The accuracy for each model over each iteration is shown in the images below.

Results:

1. Logistic Regression Classifier – Own implementation vs. Scikit Learn:

```
Logistic Regression Classifier Learning

Accuracy 0 = 0.9807692307692307
Accuracy 1 = 0.9807692307692307
Accuracy 2 = 0.9423076923076923
Accuracy 3 = 0.9423076923076923
Accuracy 4 = 0.9615384615384616
Accuracy 5 = 1.0
Accuracy 6 = 0.9807692307692307
Accuracy 7 = 0.9807692307692307
Accuracy 8 = 0.9615384615384616
Accuracy 9 = 0.9615384615384616

Mean accuracy = 0.9692307692307691
```

```
Logistic Regression Classifier Learning - Scikit

Accuracy 0 = 0.9615384615384616
Accuracy 1 = 0.9807692307692307
Accuracy 2 = 0.9615384615384616
Accuracy 3 = 0.9807692307692307
Accuracy 4 = 1.0
Accuracy 5 = 0.9423076923076923
Accuracy 6 = 0.9615384615384616
Accuracy 7 = 0.9615384615384616
Accuracy 8 = 0.9807692307692307
Accuracy 9 = 1.0

Mean accuracy = 0.973076923076923
```

2. Support Vector Machine Classifier – Own implementation vs. Scikit Learn:

```
SVM Classifier Learning

Accuracy 0 = 0.9038461538461539
Accuracy 1 = 0.9423076923076923
Accuracy 2 = 1.0
Accuracy 3 = 0.9807692307692307
Accuracy 4 = 0.9615384615384616
Accuracy 5 = 0.9615384615384616
Accuracy 6 = 0.9615384615384616
Accuracy 7 = 0.9615384615384616
Accuracy 8 = 0.9230769230769231
Accuracy 9 = 0.9230769230769231

Mean accuracy = 0.951923076923077
```

```
SVM Classifier Learning - Scikit

Accuracy 0 = 0.9807692307692307
Accuracy 1 = 0.9807692307692307
Accuracy 2 = 0.9615384615384616
Accuracy 3 = 0.9615384615384616
Accuracy 4 = 0.9615384615384616
Accuracy 5 = 0.9423076923076923
Accuracy 6 = 0.9423076923076923
Accuracy 7 = 0.9615384615384616
Accuracy 8 = 0.9230769230769231
Accuracy 9 = 0.9423076923076923

Mean accuracy = 0.9557692307692307
```

3. Ensemble Classifier – Own implementation vs. Scikit Learn

```
Ensemble Classifier Learning

Accuracy 0 = 0.9615384615384616
Accuracy 1 = 0.9615384615384616
Accuracy 2 = 0.9615384615384616
Accuracy 3 = 1.0
Accuracy 4 = 0.9807692307692307
Accuracy 5 = 0.9807692307692307
Accuracy 6 = 0.9807692307692307
Accuracy 7 = 1.0
Accuracy 8 = 0.9423076923076923
Accuracy 9 = 1.0

Mean accuracy = 0.976923076923077
```

```
Ensemble Classifier Learning - Scikit

Accuracy 0 = 0.9807692307692307
Accuracy 1 = 0.9807692307692307
Accuracy 2 = 0.9230769230769231
Accuracy 3 = 0.9807692307692307
Accuracy 4 = 0.9807692307692307
Accuracy 5 = 0.9615384615384616
Accuracy 6 = 0.9807692307692307
Accuracy 7 = 0.9423076923076923
Accuracy 8 = 0.9615384615384616
Accuracy 9 = 0.9807692307692307

Mean accuracy = 0.9673076923076922
```

Observations: From the results above, starting with the Logistic Regression classifier, it can be observed that the mean accuracy of the own implementation 96.9%, is just 0.4% lower than the scikit learn version of 97.3%. Similarly, comparing the mean accuracy for SVM model, again the own implementation, 95.2% performs just 0.4% lower than the scikit implementation value of 95.6%. However, for the ensemble classifier, there is an improvement by 1% in the own implementation, which scored 97.7% when compared to the 96.7%.

On analysing the three classifiers from the own implementation alone, it can be seen that the Logistic Regression (96.9%) model performs better than the Support Vector Machine (95.2%) model by 1.7%, while the ensemble classifier (97.7%) model gives the best performance among the three.

Conclusion: Overall, it can be seen that the Logistic Regression and Support Vector Machine models that have been implemented from scratch performs almost as good as the implementations using scikit learn. Also, the results above show that the performance of the ensemble classifier from our own implementation is slightly better when compared to the scikit learn implementation.

Future Work: The ensemble model implemented here, uses just two algorithms to make a final prediction. For further improvements, this model can be extended to include more than two algorithms, in which case it will become a voting classifier. For instance, if we decide to include five different algorithms, and three algorithms predict 'Class A' and the remaining algorithms predict 'Class B', then the final prediction would be 'Class A' as it has the highest vote among the five models.

Contribution of each team member: This assignment has been done as a group of two members, with the overview of the contribution of each member, towards the coding part, given below. Additionally, more specific code contributions are also mentioned in comments on the source code files.

Prakhar Gurawa (20231064):

- Own implementation: *SupportVectorMachine.py*, *EnsembleClassifier.py* (partially)
- Scikit implementation: *SupportVectorMchine_Scikit.py*, *EnsembleClassifier_Scikit.py* (partially)
- Report: Reference Implementation, Comparison Table, Implementation Details, Observation, Future work, References

Yashitha Agarwal (20230091):

- Own implementation: *LogisticRegression.py*, *EnsembleClassifier.py* (partially)
- Scikit implementation: *LogisticRegression_Scikit.py*, *EnsembleClassifier_Scikit.py* (partially)
- Report: Algorithm description and design decisions, Tests, Results, Conclusion, Contribution of each team member, Code execution

Code execution: All the python files required to compile the solutions and the dataset are zipped into a single folder. In order to execute the ensemble classification model (implementation from scratch), run the *EnsembleClassifier.py*, after ensuring that *SupportVectorMachine.py* and *LogisticRegression.py* are present in the same folder, as it imports those python files along with the *beer.txt* dataset. To execute our own implementation of individual classification models, Logistic Regression or Support Vector Machine, run the *LogisticRegression.py* or *SupportVectorMachine.py* file respectively.

On executing the *LogisticRegression.py*, *SupportVectorMachine.py* and *EnsembleClassifier.py* individually, corresponding **output files** showing the predicted and actual value for each model over the 10 iterations are created with the file names *LR_Results.csv*, *SVM_Results.csv* and *Ensemble_Results.csv* respectively. Scikit implementation of both the algorithms, and the ensemble classification are also present in the same folder.

References:

1. <https://towardsdatascience.com/logistic-regression-from-scratch-69db4f587e17>
2. <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc>
3. <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>
4. <https://stackoverflow.com/questions/47966728/how-to-fix-float-object-has-no-attribute-exp?noredirect=1&lq=1>
5. <https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-numpy-array-without-looping/56594688>
6. <https://datascience.stackexchange.com/questions/22470/python-implementation-of-cost-function-in-logistic-regression-why-dot-multiplic>
7. <https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-numpy-array-without-looping/56594688>
8. <https://towardsdatascience.com/svm-implementation-from-scratch-python-2db2fc52e5c2>
9. <https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>
10. https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html
11. <https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b>

```

1  """Final Source Code - Appendix"""
2
3  '''
4  LOGISTIC REGRESSION - OWN IMPLEMENTATION
5  '''
6
7  # -*- coding: utf-8 -*-
8  '''
9  CT4101 MACHINE LEARNING - ASSIGNMENT 2
10  Prakhar Gurawa (20231064)
11  Yashitha Agarwal (20230091)
12
13  Code by: Yashitha Agarwal (20230091)
14  '''
15
16  # References:
17  https://towardsdatascience.com/logistic-regression-from-scratch-69db4f587e17 (A nice
18  explanation of logistic regression and its mathematics)
19
20  # We are importing all necessary libraries to implement our model
21  import matplotlib.pyplot as plt
22  import numpy as np
23  import pandas as pd
24  from sklearn.model_selection import train_test_split
25
26  class LogisticRegression:
27
28      def __init__(self, alpha=0.01, iterations=1000): # constructor function to
29      initialize learning rate alpha and number of iterations
30          self.alpha = alpha
31          self.iterations = iterations
32
33      def sigmoid(self, z): # utility function to find sigmoid values of input
34      # Reference :
35      https://stackoverflow.com/questions/47966728/how-to-fix-float-object-has-no-at-tribute-exp?noredirect=1&lq=1
36      z = np.array(z, dtype=float)
37      return 1 / (1 + np.exp(-z))
38
39      def transformMultiClass(self, y, c): # utility function to convert given
40      muticlass vector to 2 class 0/1 vector
41      # Reference :
42      https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-num-py-array-without-looping/56594688
43      y_copy = y.copy()
44      indices_C = y==c # all indices with class 'c'
45      indices_notC = y!=c # all indices with classes other than 'c'
46      y_copy[indices_C]=1 # convert values to 1 for all class 'c'
47      y_copy[indices_notC]=0 # convert values to 0 for all class other than 'c'
48      return y_copy
49
50      def gradientDescent(self, X, y, weight, h): # function to calculate gradient descent
51      for logistic regression
52      # By calculus the derivative of cost function wrt weights comes out to be
53      xi*(y_pred - y)
54      """
55      # Reference :
56      https://datascience.stackexchange.com/questions/22470/python-implementation-of-cost-function-in-logistic-regression-why-dot-multiplic
57      Cost function for Logistic Regression = -1/m * np.sum(np.dot(Y, np.log(A)) +
58      np.dot(1-Y, np.log(1-A)))
59      dw = 1/m * np.dot(X, dz.T)
60      """
61      costGradient = np.dot(X.T, (h - y)) / len(y) # gradient of cost function for
62      logistic regression (calculated using chain rule of calculus)
63      weight = weight - self.alpha * costGradient # gradient descent process
64      (updating weight on basis of cost gradients)
65      return weight
66
67      def costFunction(self, y, weight, h): # defines cost function for logic regression
68      costValue = (1 / len(y)) * (np.sum(-y.T.dot(np.log(h)) - (1 -

```

```

58     y).T.dot(np.log(1 - h))) # Binary cross entropy loss function
59     return costValue
60
61 def fit(self,X,y): # starts the logistic regression model by fitting our given
dataset
62     self.costs = []
63     self.weights = []
64     X = np.insert(X, 0, 1, axis=1) # adding 1 for bias term
65     classes = set(y) # storing unique classes (our predicted
output will be one of them)
66     # Using concept of one-vs-all where a particular class is treated as 1 and
all other 0 and this process is repeated for all classes
67     for c in classes:
68         # Gradient descent for class 'c'
69         y_onevall = self.transformMultiClass(y,c)
70         weight = np.zeros(X.shape[1]) # initializing weights with 0 at beginning
of logistic regression
71         cost = list()
72         for itr in range(self.iterations):
73             z = X.dot(weight)
74             h = self.sigmoid(z)
75             weight = self.gradientDescent(X,y_onevall,weight,h)
76             cst = self.costFunction(y_onevall,weight,h)
77             cost.append(cst)
78         self.weights.append((weight,c))
79         self.costs.append((cost,c))
80     return self
81
82 def predict(self,X): # predict class values for given independent features
83     X = np.insert(X, 0, 1, axis=1)
84     X_prediction = list() # storing predicted classes
85     for x in X:
86         class_predictions = [(self.sigmoid(x.dot(weight)),c) for weight,c in
self.weights] # This loop runs n times for n classes (multi class
logistic regression one vs all)
87
88     X_prediction.append(max(class_predictions)[1])
89     # append the class with maximum prediction value (probability)
90     return X_prediction
91
92 def score(self,X,y): # function to calculate number of matches between actual
classes and predicted classes by our model
93     size = len(y)
94     return sum(self.predict(X)==y)/size # number of matches divided by total
inputs
95
96 def plotCost(self,costs): # utility function to plot cost value per class
97     for cost,c in costs:
98         plt.plot(range(len(cost)),cost,'blue')
99         plt.title(" Cost Function for type " + str(c) + " vs All")
100         plt.xlabel("Iterations")
101         plt.ylabel("Cost")
102         plt.show()
103
104 if __name__=="__main__":
105     # importing dataset using pandas
106     filename = 'beer.txt'
107     header_list =
108     ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','be
er_id','colour','degree_of_fermentation']
109     data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
110
111     # creating dependent and independent features
112     X = data.drop(['style','beer_id'],axis=1).values
113     y = data['style'].values
114
115     # creating a pandas dataframe for storing results
116     predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual
Value'])
117
118     # data standardization pre-processing

```



```

116 def feature_scaling(X):
117     X = X.astype(np.float)
118     mean = np.mean(X, axis=0)
119     sd = np.std(X, axis=0)
120     X_scaled= (X - mean) / sd
121     return X_scaled
122
123 X = feature_scaling(X)
124
125 scores=list()
126 print("Logistic Regression Classifier Learning\n")
127 for i in range(10):
128     X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3,
129     shuffle = True) # split data
130     model = LogisticRegression(alpha=0.01,iterations=1000)
131     model.fit(X_train, y_train)
132     prediction = model.predict(X_test)
133     score = model.score(X_test,y_test)
134     # storing all the predictions and actual values in the dataframe
135     for (p , a) in zip(prediction, y_test):
136         predictions_final = predictions_final.append({'Iteration': i, 'Predicted
137         Value': p, 'Actual Value': a}, ignore_index=True)
138     print("Accuracy ",i," = ",score)
139     scores.append(score)
140
141     print("\nMean accuracy = ",np.mean(scores))
142     model.plotCost(model.costs)
143
144     # output the results to csv file
145     predictions_final.to_csv('LR_Results.csv', index=False)
146
147
148
149
150 LOGISTIC REGRESSION - SCIKIT LEARN
151
152
153 # -*- coding: utf-8 -*-
154
155 CT4101 MACHINE LEARNING - ASSIGNMENT 2
156 Prakhar Gurawa (20231064)
157 Yashitha Agarwal (20230091)
158
159 Code by: Yashitha Agarwal (20230091)
160
161
162 # We are importing all necessary libraries to implement our model
163 import pandas as pd
164 import numpy as np
165 from sklearn.linear_model import LogisticRegression
166 from sklearn.preprocessing import StandardScaler
167 from sklearn.model_selection import train_test_split
168
169 # importing dataset using pandas
170 filename = 'beer.txt'
171 header_list =
172     ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','beer_i
173     d','colour','degree_of_fermentation']
174 data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
175
176 # creating dependent and independent features
177 X = data.drop(['style','beer_id'],axis=1).values
178 y = data['style'].values
179
180 # data stardalization pre-processing
181 scaler = StandardScaler()
182 X = scaler.fit_transform(X)
183
184 scores=list()

```

```

183 print("Logistic Regression Classifier Learning - Scikit\n")
184 for i in range(10):
185     X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3,
186         shuffle = True) # split data
187     model = LogisticRegression()
188     model.fit(X_train, y_train)
189     prediction = model.predict(X_test)
190     score = model.score(X_test,y_test)
191     print("Accuracy ",i," = ",score)
192     scores.append(score)
193
194 print("\nMean accuracy = ",np.mean(scores))
195
196 -----
197
198
199 '''
200 SUPPORT VECTOR MACHINE - OWN IMPLEMENTATION
201 '''
202
203 # -*- coding: utf-8 -*-
204 '''
205 CT4101 MACHINE LEARNING - ASSIGNMENT 2
206 Prakhar Gurawa (20231064)
207 Yashitha Agarwal (20230091)
208
209 Code by: Prakhar Gurawa (20231064)
210 '''
211
212 #We are importing all necessary libraries to implement our model
213 import matplotlib.pyplot as plt
214 import numpy as np
215 import pandas as pd
216 from sklearn.model_selection import train_test_split
217
218 class SVM:
219
220     def __init__(self, alpha=0.001,iterations=1000): # constructor function to
221         initialize learning rate alpha, lambda param and number of iterations
222         self.alpha = alpha
223         self.iterations = iterations
224         # self.lambda_param = lambda_param (if in future user wants to give some
225         well defined lambda param say for example : lambda_param=0.01 , currently
226         taking as 1/epoch)
227
228     def transformMultiClass(self,y,c): # utility function to convert given
229         multiclass vector to 2 class 0/1 vector
230         # Reference :
231         https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-num-py-array-without-looping/56594688
232         y_copy = y.copy()
233         indices_C = y==c # all indices with class 'c'
234         indices_notC = y!=c # all indices with classes other than 'c'
235         y_copy[indices_C]=1 # convert values to 1 for all class 'c'
236         y_copy[indices_notC]=-1 # convert values to 0 for all class other than 'c'
237         return y_copy
238
239     def compute_cost(self,W, X, Y):
240         # Reference :
241         https://towardsdatascience.com/svm-implementation-from-scratch-python-2db2fc52e5c2
242         # calculate hinge loss - SVM uses hinge loss function
243         N = X.shape[0]
244         distances = 1 - Y * (np.dot(X, W))
245         distances[distances < 0] = 0 # equivalent to max(0, distance)
246         hinge_loss = 10000 * (np.sum(distances) / N) # 10000 is regularization
247         parameter (fixed for now)
248         # calculate cost
249         cost = 1 / 2 * np.dot(W, W) + hinge_loss
250         return cost

```



```

244
245
246 def fit(self,X,y):
247     n_samples, n_features = X.shape
248     self.costs = []
249     self.weights = []
250     classes = set(y) # storing unique classes (our predicted
output will be one of them)
251     # Using concept of one-vs-all where a particular class is treated as 1 and
all other 0 and this process is repeated for all classes
252     for c in classes:
253         # weight updation for class 'c'
254         y_onevall = self.transformMultiClass(y,c)
255         weight = np.zeros(n_features) # initializing weights with 0 at beginning
of SVM Classifier
256         bias = 0 # initailizing bias as 0 at beginning of
SVM Classifier
257         cost = list()
258         for itr in range(self.iterations):
259             for idx, x_i in enumerate(X): # iterating through full dataset
260                 lambda_param = 1/(itr+1) # regularization param ( will
decrease with iteration)
261                 condition = y_onevall[idx] * (np.dot(x_i, weight) - bias) >= 1 #
standard condition of SVM Classifier
262                 if condition:
263                     weight = weight - self.alpha * (2 * lambda_param * weight) #
gradient update when no misclassification
264                 else:
265                     weight = weight - self.alpha * (2 * lambda_param * weight -
np.dot(x_i, y_onevall[idx])) # gradient update when
misclassification
266                     bias = bias - self.alpha * y_onevall[idx]
267
268                 cst=self.compute_cost(weight,X,y_onevall)
269                 cost.append(cst)
270                 self.weights.append((weight,bias,c))
271                 self.costs.append((cost,c))
272         return self
273
274 def predict(self,X):
275     X_prediction = list() # storing predicted classes
276     for x in X:
277         class_predictions = [(np.dot(x, weight) - bias ,c) for weight,bias,c in
self.weights] # This loop runs n times for n classes (multi class
logistic regression one vs all)
278         X_prediction.append(max(class_predictions)[1])
279     return X_prediction
280
281 def score(self,X,y): # function to calculate number of matches between actual
classes and predicted classes by our model
282     size = len(y)
283     return sum(self.predict(X)==y)/size # number of matches divided by total
inputs
284
285 def plotCost(self,costs): # utility function to plot cost value per class
286     for cost,c in costs:
287         scaledCost = [c/10000 for c in cost ] # Dividing by 10000 for scaled
output
288         plt.plot(range(len(cost)),scaledCost,'blue')
289         plt.title(" Cost Function for type " + str(c) + " vs All")
290         plt.xlabel("Iterations")
291         plt.ylabel("Cost")
292         plt.show()
293
294 if __name__=="__main__":
295     # importing dataset using pandas
296     filename = 'beer.txt'
297     header_list =
['caloric_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','be
er_id','colour','degree_of_fermentation']
298     data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
299     data.dtypes

```

```

300
301     # creating dependent and independent features
302     y = data['style'].values
303     X = data.drop(['style','beer_id'],axis=1).values
304
305     # creating a pandas dataframe for storing results
306     predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual
Value'])
307
308     # data stardalization pre-processing
309     def feature_scaling(X):
310         X = X.astype(np.float)
311         mean = np.mean(X, axis=0)
312         sd = np.std(X, axis=0)
313         X_scaled= (X - mean) / sd
314         return X_scaled
315
316     X = feature_scaling(X)
317
318     scores=list()
319     print("SVM Classifier Learning\n")
320     for i in range(10):
321         X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3,
shuffle = True) # split data
322         model = SVM(alpha=0.001,iterations=1000)
323         model.fit(X_train, y_train)
324         prediction = model.predict(X_test)
325         score = model.score(X_test,y_test)
326         # storing all the predictions and actual values in the dataframe
327         for (p , a) in zip(prediction, y_test):
328             predictions_final = predictions_final.append({'Iteration': i, 'Predicted
Value': p, 'Actual Value': a}, ignore_index=True)
329         print("Accuracy ",i," = ",score)
330         scores.append(score)
331
332
333     print("\nMean accuracy = ",np.mean(scores))
334     model.plotCost(model.costs)
335
336     # output the results to csv file
337     predictions_final.to_csv('SVM_Results.csv', index=False)
338
339     '''
340     -----
341     '''
342
343     '''
344     SUPPORT VECTOR MACHINE - SCIKIT LEARN
345     '''
346
347     # -*- coding: utf-8 -*-
348     '''
349     CT4101 MACHINE LEARNING - ASSIGNMENT 2
350     Prakhar Gurawa (20231064)
351     Yashitha Agarwal (20230091)
352
353     Code by: Prakhar Gurawa (20231064)
354     '''
355
356     # We are importing all necessary libraries to implement our model
357     import pandas as pd
358     import numpy as np
359     from sklearn.svm import SVC
360     from sklearn.preprocessing import StandardScaler
361     from sklearn.model_selection import train_test_split
362
363     # importing dataset using pandas
364     filename = 'beer.txt'
365     header_list =
['caloric_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','beer_i
d','colour','degree_of_fermentation']

```

```

366 data = pd.read_csv(filename,sep='\t', header=None, dtype=str, names=header_list)
367
368 # creating dependent and independent features
369 X = data.drop(['style','beer_id'],axis=1).values
370 y = data['style'].values
371
372 # data stardalization pre-processing
373 scaler = StandardScaler()
374 X = scaler.fit_transform(X)
375
376 scores=list()
377 print("SVM Classifier Learning - Scikit\n")
378 for i in range(10):
379     X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3, shuffle
380                                                     = True) # split data
381     model = SVC()
382     model.fit(X_train, y_train)
383     prediction = model.predict(X_test)
384     score = model.score(X_test,y_test)
385     print("Accuracy ",i," = ",score)
386     scores.append(score)
387
388 print("\nMean accuracy = ",np.mean(scores))
389
390 '''
391 -----
392 -----
393
394 '''
395 ENSEMBLE CLASSIFIER - OWN IMPLEMENTATION
396 '''
397
398 # -*- coding: utf-8 -*-
399 '''
400 CT4101 MACHINE LEARNING - ASSIGNMENT 2
401 Prakhar Gurawa (20231064)
402 Yashitha Agarwal (20230091)
403
404 Code by: Combined effort (specific parts mentioned in comments)
405 '''
406
407 # We are importing all necessary libraries to implement our model
408 import numpy as np
409 import pandas as pd
410 from sklearn.model_selection import train_test_split
411 from LogisticRegression import LogisticRegression
412 from SupportVectorMachine import SVM
413
414 class EnsembleClassifier:
415
416     # SVM and LR classifiers are considered to make final model more robust
417     # Scratch implementation of SVM : SupportVectorMachine.py
418     # Scratch implementation of Logistic Regression : LogisticRegression.py
419
420     # Code by: Yashitha Agarwal (20230091)
421     def __init__(self,lrAlpha=0.01,svmAlpha=0.01,iterations=1000): # Constructor
422         function to initialize individual hyperparameters for LR and SVM
423         self.lrAlpha = lrAlpha
424         self.svmAlpha = svmAlpha
425         self.iterations = iterations
426         self.lrModel = None
427         self.svmModel = None
428
429     # Code by: Yashitha Agarwal (20230091)
430     def fit(self,X,y):
431         self.lrModel = LogisticRegression(self.lrAlpha,self.iterations)
432         self.svmModel = SVM(self.svmAlpha,self.iterations)
433         self.lrModel.fit(X,y) # Fitting independent and dependent feature in
434                               # Logistic Regression
435         self.svmModel.fit(X,y) # Fitting independent and dependent feature in

```

Support Vector Machine Classifier

```
434
435 # Code by: Prakhar Gurawa (20231064)
436 def predict(self,X,y):
437     lrScore = self.lrModel.score(X,y)
438     svmScore = self.svmModel.score(X,y)
439     lrPrediction = self.lrModel.predict(X)      # Prediction of Logistic
440     svmPrediction = self.svmModel.predict(X)    # Prediction of Support Vector
441     # Currently we are considering only two algorithms and considering
442     # prediction of that higher score classifier in case of disagreement
443     finalPrediction = list() # Storing predicted classes
444     for i in range(len(lrPrediction)):
445         if lrPrediction[i] == svmPrediction[i]:
446             finalPrediction.append(lrPrediction[i]) # Case 1: Both LR ans SVM
447             # predict to same class
448         else:
449             # Case 2: Disagreement
450             # between LR and SVM classifiers
451             if lrScore > svmScore:
452                 finalPrediction.append(lrPrediction[i])
453             else:
454                 finalPrediction.append(svmPrediction[i])
455     # Future work : If we have more than two algorithms we will make this as a
456     # voting classifier.
457     # Mutiple classifer are considered and majority of prediction is taken as
458     # final prediction.
459     return finalPrediction # Final Predictions using ensemble
460
461 # Code by: Prakhar Gurawa (20231064)
462 def score(self,X,y): # Function to calculate number of matches between actual
463     # classes and predicted classes by our model
464     size = len(y)
465     return sum(self.predict(X,y)==y)/size # Number of matches divided by total
466     # inputs
467
468 # Code by: Yashitha Agarwal (20230091)
469 # importing dataset using pandas
470 filename = 'beer.txt'
471 header_list =
472     ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','bitterness','beer_id',
473     'colour','degree_of_fermentation']
474 data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
475
476 # creating dependent and independent features
477 X = data.drop(['style','beer_id'],axis=1).values
478 y = data['style'].values
479
480 # creating a pandas dataframe for storing results
481 predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual
482 Value'])
483
484 # Code by: Prakhar Gurawa (20231064)
485 # data stardalization pre-processing
486 def feature_scaling(X):
487     X = X.astype(np.float)
488     mean = np.mean(X, axis=0)
489     sd = np.std(X, axis=0)
490     X_scaled= (X - mean) / sd
491     return X_scaled
492
493 X = feature_scaling(X)
494
495 scores=list()
496 print("Ensemble Classifier Learning\n")
497 for i in range(10):
498     X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3,
499     shuffle = True) # split data
500     model = EnsembleClassifier()
501     model.fit(X_train, y_train)
502     # passing y_test to predict since it is needed to get the score of LR and SVM
```

```

model
492 prediction = model.predict(X_test, y_test)
493 score = model.score(X_test, y_test)
494 # storing all the predictions and actual values in the dataframe
495 for (p , a) in zip(prediction, y_test):
496     predictions_final = predictions_final.append({'Iteration': i, 'Predicted
        Value': p, 'Actual Value': a}, ignore_index=True)
497 print("Accuracy ",i," = ",score)
498 scores.append(score)
499
500 print("\nMean accuracy = ",np.mean(scores))
501
502 # output the results to csv file
503 predictions_final.to_csv('Ensemble_Results.csv', index=False)
504
505 '''
506 -----
507 -----
508
509 '''
510 ENSEMBLE CLASSIFIER - SCIKIT LEARN
511 '''
512
513 # -*- coding: utf-8 -*-
514 '''
515 CT4101 MACHINE LEARNING - ASSIGNMENT 2
516 Prakhar Gurawa (20231064)
517 Yashitha Agarwal (20230091)
518
519 Code by: Combined effort (specific parts mentioned in comments)
520 '''
521
522 # We are importing all necessary libraries to implement our model
523 import numpy as np
524 import pandas as pd
525 from sklearn.model_selection import train_test_split
526 from sklearn.preprocessing import StandardScaler
527 from sklearn.linear_model import LogisticRegression
528 from sklearn.svm import SVC
529
530 class EnsembleClassifier:
531
532     # SVM and LR classifiers are considered to make final model more robust
533     # Code by: Yashitha Agarwal (20230091)
534     def fit(self,X,y):
535         self.lrModel = LogisticRegression()
536         self.svmModel = SVC()
537         self.lrModel.fit(X,y) # Fitting independent and dependent feature in
            Logistic Regression
538         self.svmModel.fit(X,y) # Fitting independent and dependent feature in
            Support Vector Machine Classifier
539
540     # Code by: Prakhar Gurawa (20231064)
541     def predict(self,X,y):
542         lrScore = self.lrModel.score(X,y)
543         svmScore = self.svmModel.score(X,y)
544         lrPrediction = self.lrModel.predict(X) # Prediction of Logistic
            Regression model
545         svmPrediction = self.svmModel.predict(X) # Prediction of Support Vector
            Machine model
546         # Currently we are considering only two algorithms and considering
            prediction of that higher score classifier in case of disagreement
547         finalPrediction = list() # Storing predicted classes
548         for i in range(len(lrPrediction)):
549             if lrPrediction[i] == svmPrediction[i]:
550                 finalPrediction.append(lrPrediction[i]) # Case 1: Both LR ans SVM
                    predict to same class
551             else: # Case 2: Disagreement
                    between LR and SVM classifiers
552                 if lrScore > svmScore:
553                     finalPrediction.append(lrPrediction[i])

```

```

554         else:
555             finalPrediction.append(svmPrediction[i])
556             # Future work : If we have more than two algorithms we will make this as a
                    voting classifier.
557             # Mutiple classifier are considered and majority of prediction is taken as
                    final prediction.
558             return finalPrediction # Final Predictions using ensemble
559
560     # Code by: Prakhar Gurawa (20231064)
561     def score(self,X,y): # Function to calculate number of matches between actual
                    classes and predicted classes by our model
562         size = len(y)
563         return sum(self.predict(X,y)==y)/size # Number of matches divided by total
                    inputs
564
565
566     # Code by: Yashitha Agarwal (20230091)
567     # importing dataset using pandas
568     filename = 'beer.txt'
569     header_list =
        ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id',
        'colour','degree_of_fermentation']
570     data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
571
572     # creating dependent and independent features
573     X = data.drop(['style','beer_id'],axis=1).values
574     y = data['style'].values
575
576     scaler = StandardScaler()
577     X = scaler.fit_transform(X)
578
579     # Code by: Prakhar Gurawa (20231064)
580     scores=list()
581     print("Ensemble Classifier Learning - Scikit\n")
582     for i in range(10):
583         X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3,
                    shuffle = True) # split data
584         model = EnsembleClassifier()
585         model.fit(X_train, y_train)
586         # passing y_test to predict since it is needed to get the score of LR and SVM
                    model
587         prediction = model.predict(X_test, y_test)
588         score = model.score(X_test,y_test)
589         print("Accuracy ",i," = ",score)
590         scores.append(score)
591
592     print("\nMean accuracy = ",np.mean(scores))

```