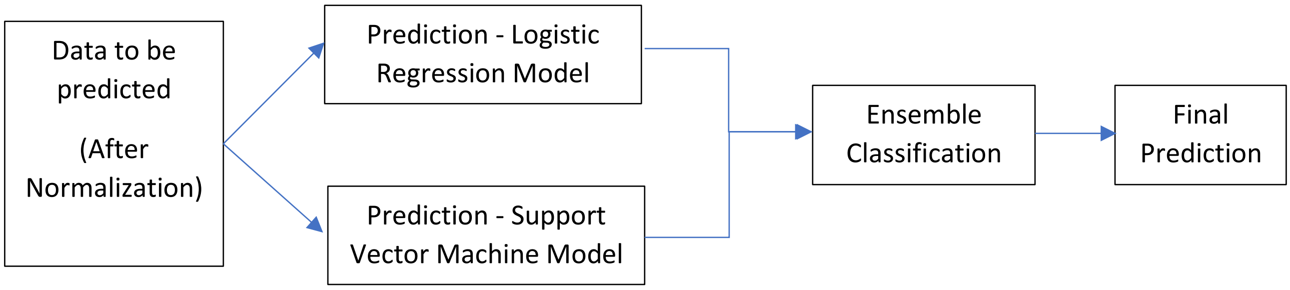
CT4101 MACHINE LEARNING - ASSIGNMENT 2

Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

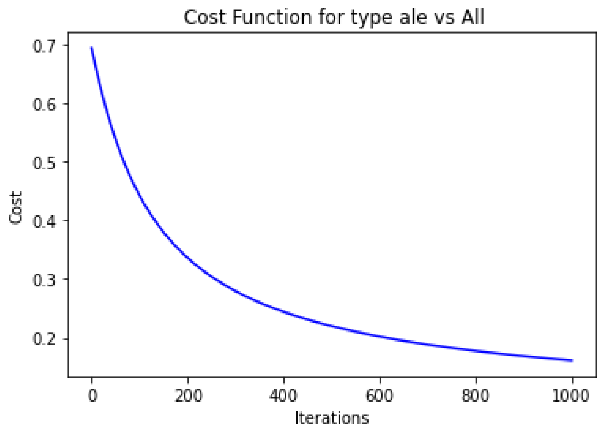
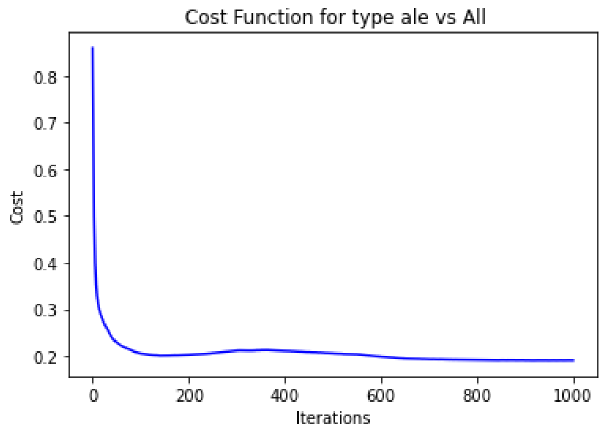
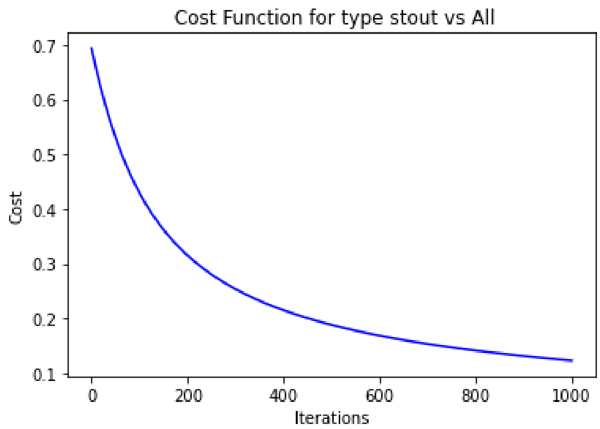
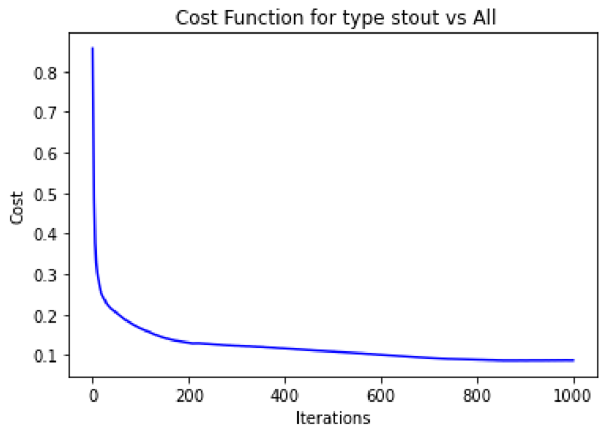
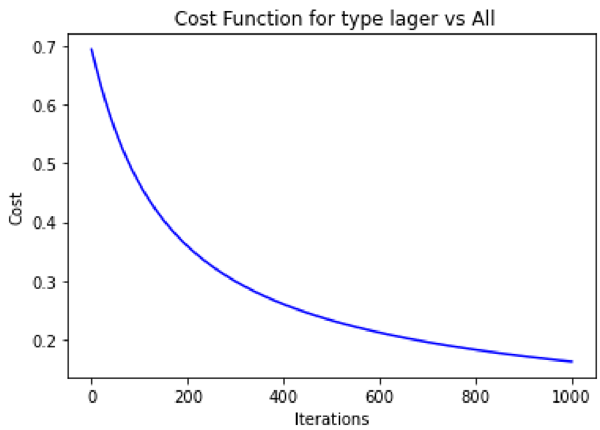
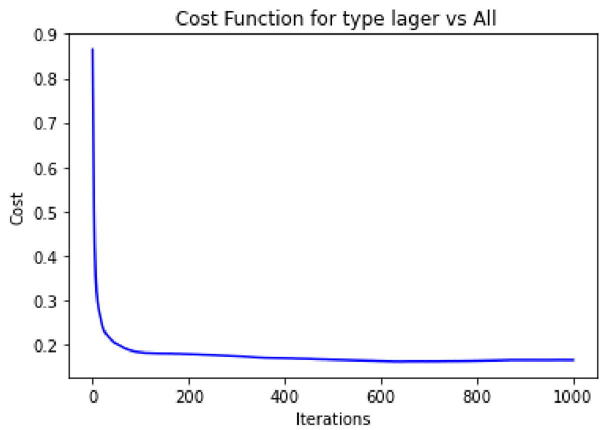
Algorithm description and design decisions: For this assignment, we have considered two machine learning models, namely Logistic Regression and Support Vector Machine classifiers, which have been implemented from scratch. Further, we have integrated both the models, based on their separate accuracy scores, 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 form 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 hyplerplane 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 / Number of iterations). $$min_w \, \|\| w \|\|^2 + \sum_{i=1}^{n}(1 - y_i \langle x_i , w\rangle)_+$$

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. $$w = w - x_i \cdot (\hat{y} - y)$$	Gradient update in SVM depends on whether the prediction from our model is correct or incorrect. In case of missclassification : $$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$ In case of no missclassification : $$w = w - \alpha \cdot (2\lambda w)$$
Number of iterations is set to 1000 for LR model.	Number of iterations is set to 1000 for SVM model.
	
	
	

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.

<u>Prakhar Gurawa (20231064):</u>

- 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

<u>Yashitha Agarwal (20230091):</u>

- 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

```python
"""Final Source Code - Appendix"""

'''
LOGISTIC REGRESSION - OWN IMPLEMENTATION
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Yashitha Agarwal (20230091)
'''

# References: https://towardsdatascience.com/logistic-regression-from-scratch-69db4f587e17 (A nice explaination of logistic regression and its mathematics)

# We are importing all necessary libraries to implement our model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split


class LogisticRegression:

    def __init__(self,alpha=0.01,iterations=1000): # constructor function to intialize learning rate alpha and number of iterations
        self.alpha = alpha
        self.iterations = iterations

    def sigmoid(self, z): # utility function to find sigmoid values of input
        # Reference : https://stackoverflow.com/questions/47966728/how-to-fix-float-object-has-no-attribute-exp?noredirect=1&lq=1
        z = np.array(z,dtype=float)
        return 1 / (1 + np.exp(-z))

    def transformMultiClass(self,y,c): # utility function to convert given mutilclass vector to 2 class 0/1 vector
        # Reference : https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-numpy-array-without-looping/56594688
        y_copy = y.copy()
        indices_C = y==c            # all indices with class 'c'
        indices_notC = y!=c         # all indices with classes other than 'c'
        y_copy[indices_C]=1         # convert values to 1 for all class 'c'
        y_copy[indices_notC]=0      # convert values to 0 for all class other than 'c'
        return y_copy

    def gradientDescent(self,X,y,weight,h): # function to calculate gradient descent for logistic regression
        # By calculus the derivative of cost function wrt weights comes out to be xi*(y_pred - y)
        """
        # Reference : https://datascience.stackexchange.com/questions/22470/python-implementation-of-cost-function-in-logistic-regression-why-dot-multiplic
        Cost function for Logistic Regression  = -1/m * np.sum(np.dot(Y,np.log(A)) + np.dot(1-Y, np.log(1-A)))
        dw = 1/m * np.dot(X, dz.T)
        """
        costGradient = np.dot(X.T, (h - y)) / len(y) # gradient of cost function for logistic regression (calculated using chain rule of calculus)
        weight = weight - self.alpha * costGradient  # gradient descent process (updating weight on basis of cost gradients)
        return weight

    def costFunction(self,y,weight,h): # defines cost function for logic regression
        costValue = (1 / len(y)) * (np.sum(-y.T.dot(np.log(h)) - (1 -
```

```
        y).T.dot(np.log(1 - h)))) # Binary cross entropy loss function
        return costValue

    def fit(self,X,y): # starts the logistic regression model by fitting our given dataset
        self.costs = []
        self.weights = []
        X = np.insert(X, 0, 1, axis=1) # adding 1 for bias term
        classes = set(y)               # storing unique classes (our predicted output will be one of them)
        # 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
        for c in classes:
            # Gradient descent for class 'c'
            y_onevall = self.transformMultiClass(y,c)
            weight = np.zeros(X.shape[1]) # initializing weights with 0 at beginning of logistic regression
            cost = list()
            for itr in range(self.iterations):
                z = X.dot(weight)
                h = self.sigmoid(z)
                weight = self.gradientDescent(X,y_onevall,weight,h)
                cst = self.costFunction(y_onevall,weight,h)
                cost.append(cst)
            self.weights.append((weight,c))
            self.costs.append((cost,c))
        return self

    def predict(self,X): # predict class values for given independent features
        X = np.insert(X, 0, 1, axis=1)
        X_prediction = list() # storing predicted classes
        for x in X:
            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)

            X_prediction.append(max(class_predictions)[1])
                        # append the class with maximum prediction value (probablity)
        return X_prediction

    def score(self,X,y): # function to calculate number of matches between actual classes and predicted classes by our model
        size = len(y)
        return sum(self.predict(X)==y)/size # number of matches divided by total inputs

    def plotCost(self,costs): # utility function to plot cost value per class
        for cost,c in costs   :
                plt.plot(range(len(cost)),cost,'blue')
                plt.title(" Cost Function for type " + str(c) +" vs All")
                plt.xlabel("Iterations")
                plt.ylabel("Cost")
                plt.show()


if __name__=="__main__":
    # importing dataset using pandas
    filename = 'beer.txt'
    header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
    data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)

    # creating dependent and independent features
    X = data.drop(['style','beer_id'],axis=1).values
    y = data['style'].values

    # creating a pandas dataframe for storing results
    predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual Value'])

    # data stardaization pre-processing
```

```
    def feature_scaling(X):
        X = X.astype(np.float)
        mean = np.mean(X, axis=0)
        sd = np.std(X, axis=0)
        X_scaled= (X -  mean) / sd
        return X_scaled

    X = feature_scaling(X)

    scores=list()
    print("Logistic Regression Classifier Learning\n")
    for i in range(10):
        X_train,X_test,y_train,y_test  = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
        model = LogisticRegression(alpha=0.01,iterations=1000)
        model.fit(X_train, y_train)
        prediction = model.predict(X_test)
        score = model.score(X_test,y_test)
        # storing all the predictions and actual values in the dataframe
        for (p , a) in zip(prediction, y_test):
            predictions_final = predictions_final.append({'Iteration': i, 'Predicted Value': p, 'Actual Value': a}, ignore_index=True)
        print("Accuracy ",i," = ",score)
        scores.append(score)

    print("\nMean accuracy = ",np.mean(scores))
    model.plotCost(model.costs)

    # output the results to csv file
    predictions_final.to_csv('LR_Results.csv', index=False)

'''
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
'''

'''
LOGISTIC REGRESSION - SCIKIT LEARN
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Yashitha Agarwal (20230091)
'''

# We are importing all necessary libraries to implement our model
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# importing dataset using pandas
filename = 'beer.txt'
header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)

# creating dependent and independent features
X = data.drop(['style','beer_id'],axis=1).values
y = data['style'].values

# data stardaization pre-processing
scaler = StandardScaler()
X = scaler.fit_transform(X)

scores=list()
```

```python
print("Logistic Regression Classifier Learning - Scikit\n")
for i in range(10):
    X_train,X_test,y_train,y_test  = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
    model = LogisticRegression()
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    score = model.score(X_test,y_test)
    print("Accuracy ",i," = ",score)
    scores.append(score)

print("\nMean accuracy = ",np.mean(scores))

'''
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
'''

'''
SUPPORT VECTOR MACHINE - OWN IMPLEMENTATION
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Prakhar Gurawa (20231064)
'''

#We are importing all necessary libraries to implement our model
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

class SVM:

    def __init__(self, alpha=0.001,iterations=1000): # constructor function to intialize learning rate alpha, lambda param and number of iterations
        self.alpha = alpha
        self.iterations = iterations
        # self.lambda_param = lambda_param (if in future user wants to give some well defined lambda param say for example : lambda_param=0.01 , currently taking as 1/epoch)

    def transformMultiClass(self,y,c): # utility function to convert given mutilclass vector to 2 class 0/1 vector
        # Reference : https://stackoverflow.com/questions/56594598/change-1s-to-0-and-0s-to-1-in-numpy-array-without-looping/56594688
        y_copy = y.copy()
        indices_C = y==c            # all indices with class 'c'
        indices_notC = y!=c         # all indices with classes other than 'c'
        y_copy[indices_C]=1         # convert values to 1 for all class 'c'
        y_copy[indices_notC]=-1     # convert values to 0 for all class other than 'c'
        return y_copy

    def compute_cost(self,W, X, Y):
        # Reference : https://towardsdatascience.com/svm-implementation-from-scratch-python-2db2fc52e5c2
        # calculate hinge loss - SVM uses hinge loss function
        N = X.shape[0]
        distances = 1 - Y * (np.dot(X, W))
        distances[distances < 0] = 0  # equivalent to max(0, distance)
        hinge_loss = 10000 * (np.sum(distances) / N) # 10000 is regularization parameter (fixed for now)
        # calculate cost
        cost = 1 / 2 * np.dot(W, W) + hinge_loss
        return cost
```

```python


    def fit(self,X,y):
        n_samples, n_features = X.shape
        self.costs = []
        self.weights = []
        classes = set(y)               # storing unique classes (our predicted output will be one of them)
        # 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
        for c in classes:
            # weight updation for class 'c'
            y_onevall = self.transformMultiClass(y,c)
            weight = np.zeros(n_features) # initializing weights with 0 at beginning of SVM Classifier
            bias = 0                      # initailizing bias as 0 at beginning of SVM Classifier
            cost = list()
            for itr in range(self.iterations):
                for idx, x_i in enumerate(X):   # iterating through full dataset
                    lambda_param = 1/(itr+1)    # regularization param ( will decrease with iteration)
                    condition = y_onevall[idx] * (np.dot(x_i, weight) - bias) >= 1 # standard condition of SVM Classifier
                    if condition:
                        weight = weight - self.alpha * (2 * lambda_param * weight) # gradient update when no misclassification
                    else:
                        weight = weight - self.alpha * (2 * lambda_param * weight - np.dot(x_i, y_onevall[idx])) # gradient update when misclassification
                        bias = bias - self.alpha * y_onevall[idx]

                cst=self.compute_cost(weight,X,y_onevall)
                cost.append(cst)
            self.weights.append((weight,bias,c))
            self.costs.append((cost,c))
        return self

    def predict(self,X):
        X_prediction = list() # storing predicted classes
        for x in X:
            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)
            X_prediction.append(max(class_predictions)[1])
        return X_prediction

    def score(self,X,y): # function to calculate number of matches between actual classes and predicted classes by our model
        size = len(y)
        return sum(self.predict(X)==y)/size # number of matches divided by total inputs

    def plotCost(self,costs): # utility function to plot cost value per class
         for cost,c in costs:
                scaledCost = [c/10000 for c in cost ] # Dividing by 10000 for scaled output
                plt.plot(range(len(cost)),scaledCost,'blue')
                plt.title(" Cost Function for type " + str(c) +" vs All")
                plt.xlabel("Iterations")
                plt.ylabel("Cost")
                plt.show()

if __name__=="__main__":
    # importing dataset using pandas
    filename = 'beer.txt'
    header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
    data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)
    data.dtypes
```

```
    # creating dependent and independent features
    y = data['style'].values
    X = data.drop(['style','beer_id'],axis=1).values

    # creating a pandas dataframe for storing results
    predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual Value'])

    # data stardaization pre-processing
    def feature_scaling(X):
        X = X.astype(np.float)
        mean = np.mean(X, axis=0)
        sd = np.std(X, axis=0)
        X_scaled= (X -  mean) / sd
        return X_scaled

    X = feature_scaling(X)

    scores=list()
    print("SVM Classifier Learning\n")
    for i in range(10):
        X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
        model = SVM(alpha=0.001,iterations=1000)
        model.fit(X_train, y_train)
        prediction = model.predict(X_test)
        score = model.score(X_test,y_test)
        # storing all the predictions and actual values in the dataframe
        for (p , a) in zip(prediction, y_test):
            predictions_final = predictions_final.append({'Iteration': i, 'Predicted Value': p, 'Actual Value': a}, ignore_index=True)
        print("Accuracy ",i," = ",score)
        scores.append(score)


    print("\nMean accuracy = ",np.mean(scores))
    model.plotCost(model.costs)

    # output the results to csv file
    predictions_final.to_csv('SVM_Results.csv', index=False)

'''
--------------------------------------------------------------------------------------------------------------------------------------------------------------------
'''

'''
SUPPORT VECTOR MACHINE - SCIKIT LEARN
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Prakhar Gurawa (20231064)
'''

# We are importing all necessary libraries to implement our model
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# importing dataset using pandas
filename = 'beer.txt'
header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
```

```
data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)

# creating dependent and independent features
X = data.drop(['style','beer_id'],axis=1).values
y = data['style'].values

# data stardaization pre-processing
scaler = StandardScaler()
X = scaler.fit_transform(X)

scores=list()
print("SVM Classifier Learning - Scikit\n")
for i in range(10):
    X_train,X_test,y_train,y_test = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
    model = SVC()
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    score = model.score(X_test,y_test)
    print("Accuracy ",i," = ",score)
    scores.append(score)


print("\nMean accuracy = ",np.mean(scores))

'''
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
'''

'''
ENSEMBLE CLASSIFIER - OWN IMPLEMENTATION
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Combined effort (specific parts mentioned in comments)
'''

# We are importing all necessary libraries to implement our model
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from LogisticRegression import LogisticRegression
from SupportVectorMachine import SVM

class EnsembleClassifier:

    # SVM and LR classifers are considered to make final model more robust
    # Scratch implementation of SVM : SupportVectorMachine.py
    # Scratch implementation of Logistic Regression : LogisticRegression.py

    # Code by: Yashitha Agarwal (20230091)
    def __init__(self,lrAlpha=0.01,svmAlpha=0.01,iterations=1000): # Constructor function to initalize individual hyperparameters for LR and SVM
        self.lrAlpha = lrAlpha
        self.svmAlpha =svmAlpha
        self.iterations = iterations
        self.lrModel = None
        self.svmModel = None

    # Code by: Yashitha Agarwal (20230091)
    def fit(self,X,y):
        self.lrModel = LogisticRegression(self.lrAlpha,self.iterations)
        self.svmModel = SVM(self.svmAlpha,self.iterations)
        self.lrModel.fit(X,y)    # Fitting independent and dependent feature in Logistic Regression
        self.svmModel.fit(X,y)   # Fitting independent and dependent feature in
```

```
        Support Vector Machine Classifier

    # Code by: Prakhar Gurawa (20231064)
    def predict(self,X,y):
        lrScore = self.lrModel.score(X,y)
        svmScore = self.svmModel.score(X,y)
        lrPrediction = self.lrModel.predict(X)      # Prediction of Logistic Regression model
        svmPrediction = self.svmModel.predict(X)    # Prediction of Support Vector Machine model
        # Currently we are considering only two algorithms and considering prediction of that higher score classifier in case of disagreement
        finalPrediction = list() # Storing predicted classes
        for i in range(len(lrPrediction)):
            if lrPrediction[i] == svmPrediction[i]:
                finalPrediction.append(lrPrediction[i]) # Case 1: Both LR ans SVM predict to same class
            else:                                       # Case 2: Disagreement between LR and SVM classifiers
                if lrScore > svmScore:
                    finalPrediction.append(lrPrediction[i])
                else:
                    finalPrediction.append(svmPrediction[i])
        # Future work : If we have more than two algorithms we will make this as a voting classifier.
        # Mutiple classifer are considered and majority of prediction is taken as final prediction.
        return finalPrediction # Final Predictions using ensemble

    # Code by: Prakhar Gurawa (20231064)
    def score(self,X,y): # Function to calculate number of matches between actual classes and predicted classes by our model
        size = len(y)
        return sum(self.predict(X,y)==y)/size # Number of matches divided by total inputs

# Code by: Yashitha Agarwal (20230091)
# importing dataset using pandas
filename = 'beer.txt'
header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)

# creating dependent and independent features
X = data.drop(['style','beer_id'],axis=1).values
y = data['style'].values

# creating a pandas dataframe for storing results
predictions_final = pd.DataFrame(columns=['Iteration','Predicted Value','Actual Value'])


# Code by: Prakhar Gurawa (20231064)
# data stardaization pre-processing
def feature_scaling(X):
    X = X.astype(np.float)
    mean = np.mean(X, axis=0)
    sd = np.std(X, axis=0)
    X_scaled= (X -  mean) / sd
    return X_scaled

X = feature_scaling(X)

scores=list()
print("Ensemble Classifier Learning\n")
for i in range(10):
    X_train,X_test,y_train,y_test  = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
    model = EnsembleClassifier()
    model.fit(X_train, y_train)
    # passing y_test to predict since it is needed to get the score of LR and SVM
```

```
    model
    prediction = model.predict(X_test, y_test)
    score = model.score(X_test,y_test)
    # storing all the predictions and actual values in the dataframe
    for (p , a) in zip(prediction, y_test):
        predictions_final = predictions_final.append({'Iteration': i, 'Predicted Value': p, 'Actual Value': a}, ignore_index=True)
    print("Accuracy ",i," = ",score)
    scores.append(score)

print("\nMean accuracy = ",np.mean(scores))

# output the results to csv file
predictions_final.to_csv('Ensemble_Results.csv', index=False)

'''
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
'''

'''
ENSEMBLE CLASSIFIER - SCIKIT LEARN
'''

# -*- coding: utf-8 -*-
'''
CT4101 MACHINE LEARNING - ASSIGNMENT 2
Prakhar Gurawa (20231064)
Yashitha Agarwal (20230091)

Code by: Combined effort (specific parts mentioned in comments)
'''

# We are importing all necessary libraries to implement our model
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

class EnsembleClassifier:

    # SVM and LR classifers are considered to make final model more robust
    # Code by: Yashitha Agarwal (20230091)
    def fit(self,X,y):
        self.lrModel = LogisticRegression()
        self.svmModel = SVC()
        self.lrModel.fit(X,y)   # Fitting independent and dependent feature in Logistic Regression
        self.svmModel.fit(X,y)  # Fitting independent and dependent feature in Support Vector Machine Classifier

    # Code by: Prakhar Gurawa (20231064)
    def predict(self,X,y):
        lrScore = self.lrModel.score(X,y)
        svmScore = self.svmModel.score(X,y)
        lrPrediction = self.lrModel.predict(X)      # Prediction of Logistic Regression model
        svmPrediction = self.svmModel.predict(X)    # Prediction of Support Vector Machine model
        # Currently we are considering only two algorithms and considering prediction of that higher score classifier in case of disagreement
        finalPrediction = list() # Storing predicted classes
        for i in range(len(lrPrediction)):
            if lrPrediction[i] == svmPrediction[i]:
                finalPrediction.append(lrPrediction[i]) # Case 1: Both LR ans SVM predict to same class
            else:                                       # Case 2: Disagreement between LR and SVM classifiers
                if lrScore > svmScore:
                    finalPrediction.append(lrPrediction[i])
```

```
                else:
                    finalPrediction.append(svmPrediction[i])
        # Future work : If we have more than two algorithms we will make this as a voting classifier.
        # Mutiple classifer are considered and majority of prediction is taken as final prediction.
        return finalPrediction # Final Predictions using ensemble

    # Code by: Prakhar Gurawa (20231064)
    def score(self,X,y): # Function to calculate number of matches between actual classes and predicted classes by our model
        size = len(y)
        return sum(self.predict(X,y)==y)/size # Number of matches divided by total inputs


# Code by: Yashitha Agarwal (20230091)
# importing dataset using pandas
filename = 'beer.txt'
header_list = ['caloric_value','nitrogen','turbidity','style','alcohol','sugars','biterness','beer_id','colour','degree_of_fermentation']
data = pd.read_csv(filename,sep='\t', header=None,dtype=str,names=header_list)

# creating dependent and independent features
X = data.drop(['style','beer_id'],axis=1).values
y = data['style'].values

scaler = StandardScaler()
X = scaler.fit_transform(X)

# Code by: Prakhar Gurawa (20231064)
scores=list()
print("Ensemble Classifier Learning - Scikit\n")
for i in range(10):
    X_train,X_test,y_train,y_test  = train_test_split(X, y, train_size = 2/3, shuffle = True) # split data
    model = EnsembleClassifier()
    model.fit(X_train, y_train)
    # passing y_test to predict since it is needed to get the score of LR and SVM model
    prediction = model.predict(X_test, y_test)
    score = model.score(X_test,y_test)
    print("Accuracy ",i," = ",score)
    scores.append(score)

print("\nMean accuracy = ",np.mean(scores))
```