Kyle Tranfaglia

COSC 311

Homework 02

28 Mar. 2024

Food Type Recognition Machine Learning Analysis

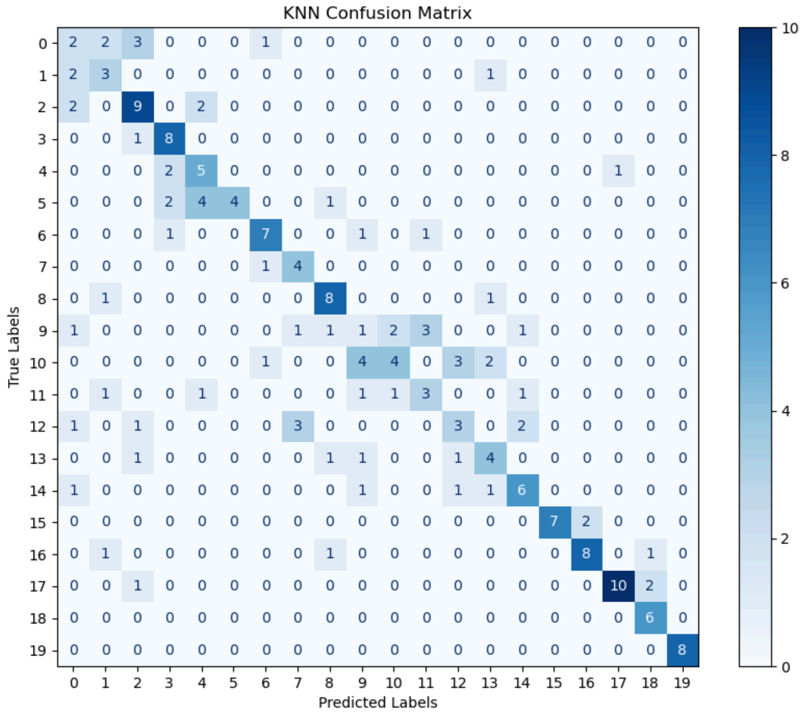
To preface, the random state used throughout the program is 7, that is, for train_test_split() and all training algorithms. All data was collected using the same random state. The following algorithms were tested: K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Support Vector Classifier (SVC), Random Forest (RF), Logistic Regression (LG), Pipelined Classifier combining Polynomial Feature Transformation and Logistic Regression (Pipe), and a Voting Classifier (VC) that considered the weighted average of the predicted probabilities.

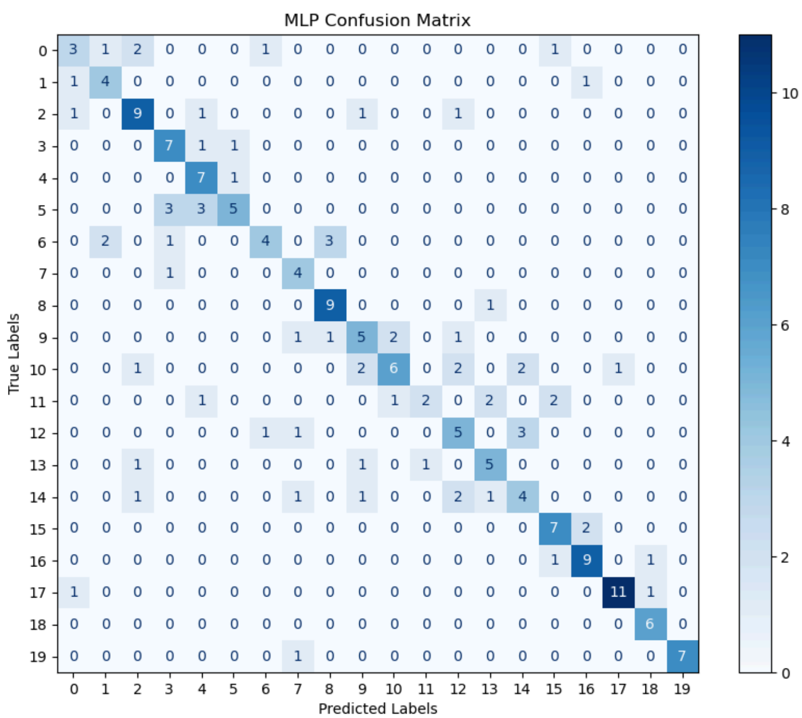
A function that condenses features by eliminating them based on importance using a logistic regression estimator and recursively selecting the top features was implemented. However, for all algorithms, the elimination of features harmed accuracy, and the amount that the accuracy decreased strongly correlated with the number of features removed.

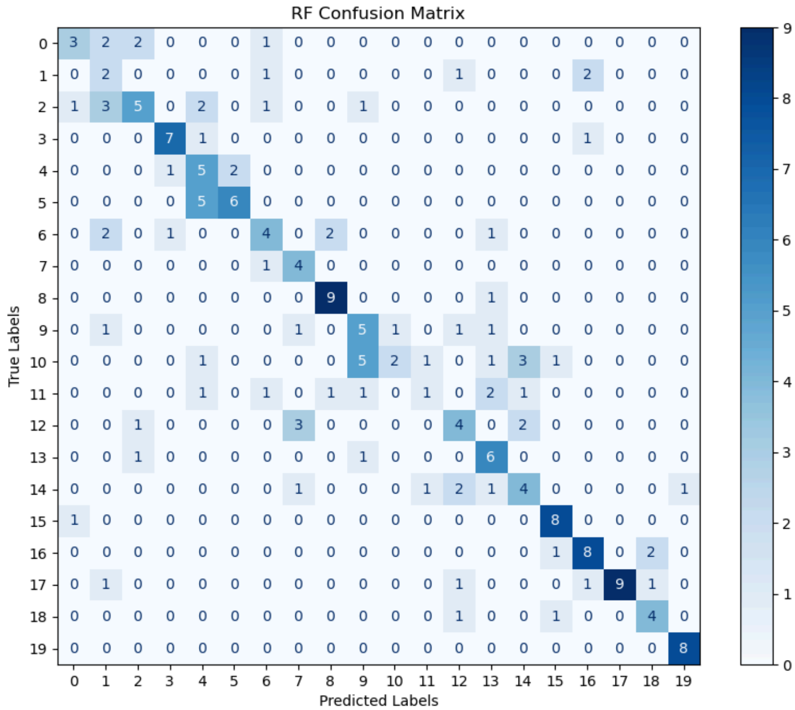
Accuracy results:

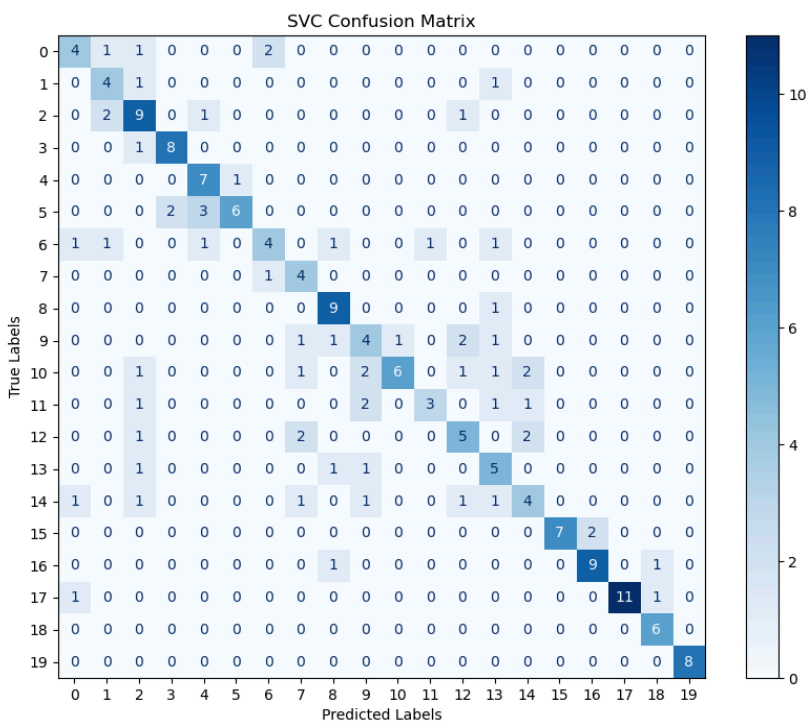
	KNN	MLP	SVC	RF	LG	Pipe	VC
Accuracy	0.59	0.64	0.66	0.56	0.54	0.64	0.67

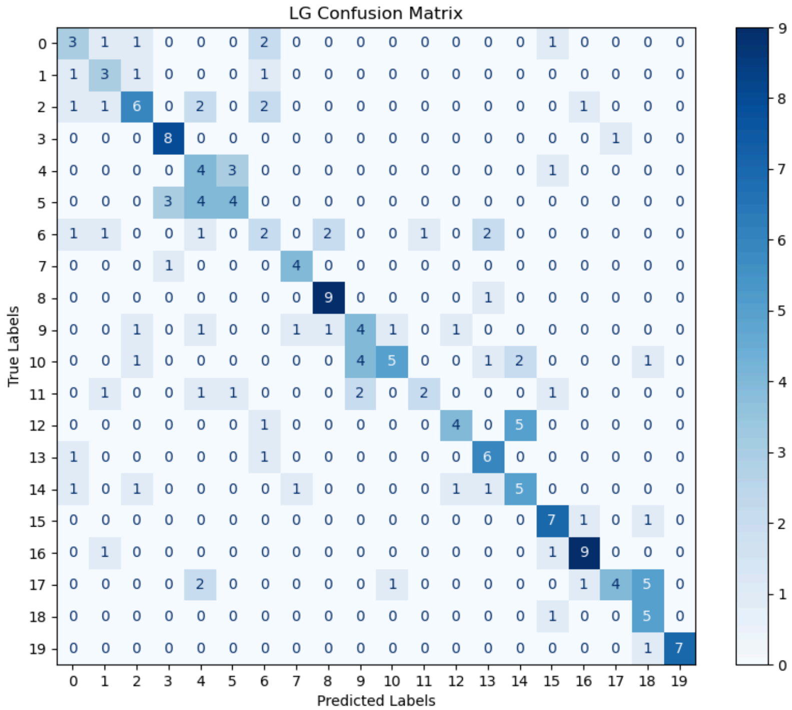
Heatmaps / Confusion Matrices:

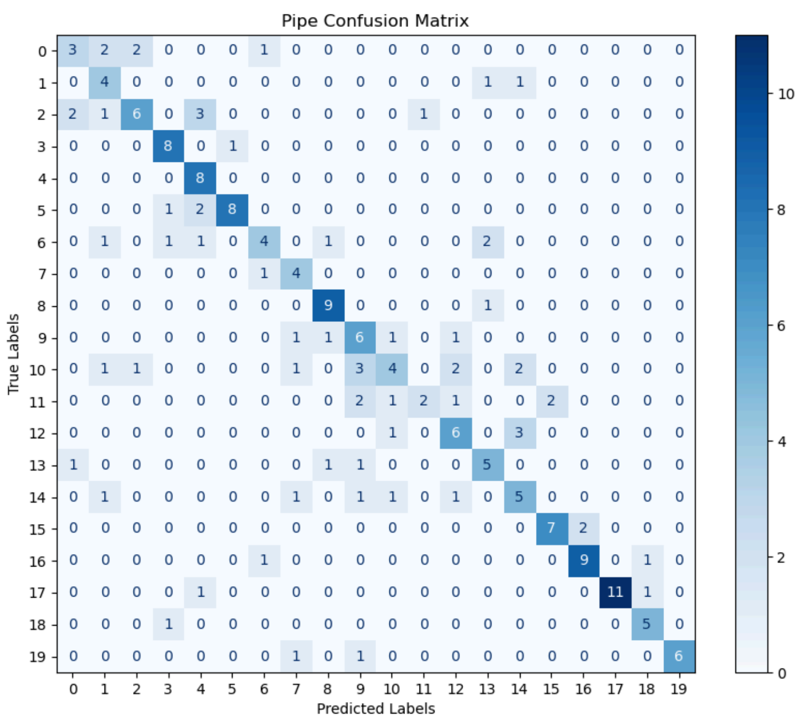

KNN Confusion Matrix
True Labels
Predicted Labels

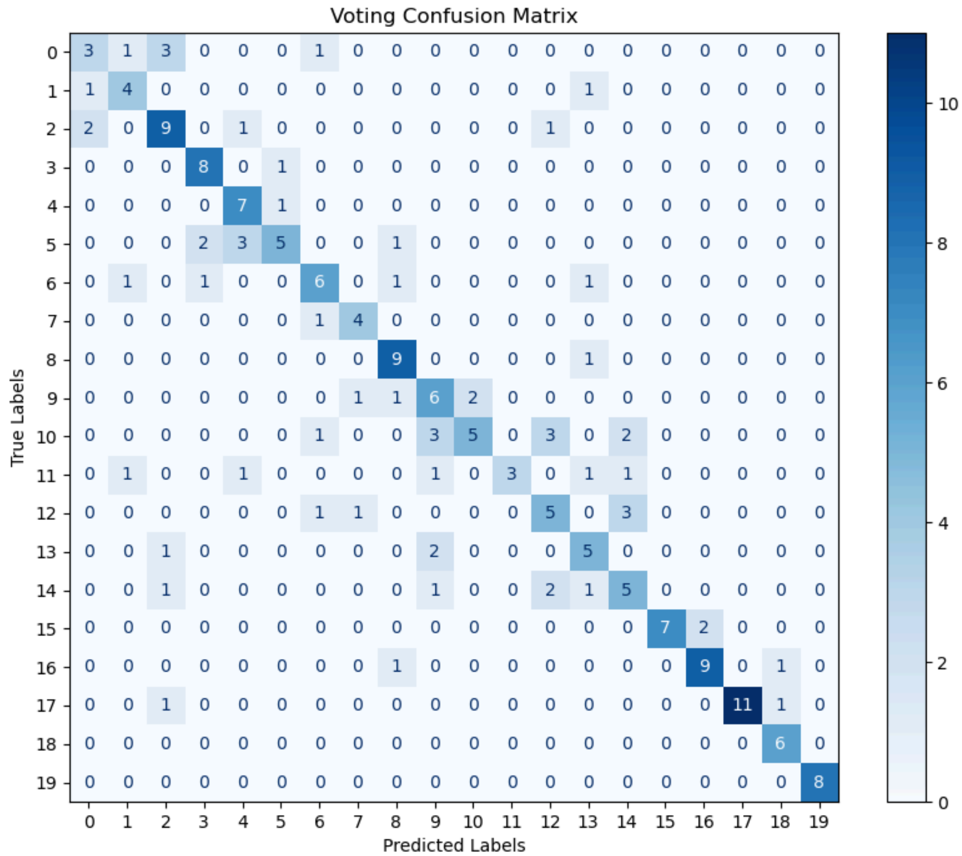

MLP Confusion Matrix
True Labels
Predicted Labels


RF Confusion Matrix
True Labels
Predicted Labels


SVC Confusion Matrix
True Labels
Predicted Labels







Each cell in the heatmap represents the misclassifications between the true and predicted labels for each class. non-diagonal cells represent misclassifications, where the predicted label does not match the true label. Lower numbers in these cells suggest fewer misclassifications. Overall, the models are fairly accurate but have some asymmetric qualities, sporadic cells off the main diagonal that denote misclassification.

Code:

```
'''
Kyle Tranfaglia
COSC311 - Homework02
Last updated 03/25/24
```

```
This program reads in a food type recognition dataset, Randomly splits the
dataset into two parts: 80% for training and 20% for testing,
and uses various classification algorithms in attempt to obtain the
highest testing accuracy on the testing data. A classification Report
is used to show the classification performance and a heatmap is used to
show the classification confusion matrix.
'''
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import RFE
from sklearn.pipeline import make_pipeline

# KNN (K-Nearest Neighbors) Algorithm to fit the classifier to the
training data and target labels and return the predictions for the test
data
def knnClassifier(attributeTrain, attributeTest, targetTrain):
    knn = KNeighborsClassifier(n_neighbors = 1)  # Instantiate a
KNeighborsClassifier object with 1 neighbors, optimal for data set
    knn.fit(attributeTrain, targetTrain)  # Fit the classifier to the
training data and target labels

    return knn.predict(attributeTest)  # Return the predictions for the
test data

# MLP (Multi-Layer Perceptron) Algorithm to fit the classifier to the
training data and target labels and return the predictions for the test
data
def mlpClassifier(attributeTrain, attributeTest, targetTrain):
```

```
    mlp = MLPClassifier(hidden_layer_sizes = 100, activation = 'tanh',
solver = 'adam', alpha = 1e-5, batch_size = 36, tol = 1e-6,
learning_rate_init=0.01,
                               learning_rate='constant', max_iter = 10000,
random_state = 7)  # Instantiate an MLPClassifier object with optimized
parameters
    mlp.fit(attributeTrain, targetTrain)  # Fit the classifier to the
training data and target labels

    return mlp.predict(attributeTest)  # Return the predictions for the
test data

# RF (Random Forest) Algorithm to fit the classifier to the training data
and target labels and return the predictions for the test data
def rfClassifier(attributeTrain, attributeTest, targetTrain):
    rf = RandomForestClassifier(n_estimators = 315, criterion = 'gini',
max_depth = 14, min_samples_split = 3,
                                        min_samples_leaf = 3, max_features
= 'sqrt', random_state = 7)  # Instantiate an RFClassifier object with
optimized parameters
    rf.fit(attributeTrain, targetTrain)  # Fit the classifier to the
training data and target labels

    return rf.predict(attributeTest)  # Return the predictions for the
test data

# SVC (Support Vector Classifier) Algorithm to fit the classifier to the
training data and target labels and return the predictions for the test
data
def svcClassifier(attributeTrain, attributeTest, targetTrain):
    svc = SVC(kernel = 'rbf', C = 13, gamma = 'scale', random_state = 7)
# Instantiate an SVCClassifier object with optimized parameters
    svc.fit(attributeTrain, targetTrain)  # Fit the classifier to the
training data and target labels

    return svc.predict(attributeTest)  # Return the predictions for the
test data

# Logistic Regression Algorithm to fit the classifier to the training data
and target labels and return the predictions for the test data
```

```
def logRegClassifier(attributeTrain, attributeTest, targetTrain):
    logReg = LogisticRegression(solver='liblinear', random_state = 7)  # 
Instantiate an LogisticRegressionClassifier object with optimized 
parameters
    logReg.fit(attributeTrain, targetTrain)  # Fit the classifier to the 
training data and target labels

    return logReg.predict(attributeTest)  # Return the predictions for the 
test data

# Pipelined Logistic Regression and polynomial feature transformation 
Algorithm to fit the classifier to the training data and target labels and 
return predictions
def pipelineClassifier(attributeTrain, attributeTest, targetTrain):
    # Note: Increasing polynomial features count improves accuracy but 
significantly decreases performance
    # Instantiate a PiplinedClassifier object to combine polynomial 
feature transformation with logistic regression with optimized parameters
    pipeline = make_pipeline(PolynomialFeatures(2), 
LogisticRegression(solver='liblinear', random_state = 7))
    pipeline.fit(attributeTrain, targetTrain)  # Fit the classifier to the 
training data and target labels

    return pipeline.predict(attributeTest)  # Return the predictions for 
the test data

# Voting Algorithm to fit the classifier to the training data and target 
labels and return the predictions for the test data
# Algorithm uses all the other classifers and uses a weighted average of 
predicted probabilities for create a prediction
def votingClassifier(attributeTrain, attributeTest, targetTrain):
    # All Classifiers for voting
    knn = KNeighborsClassifier(n_neighbors = 1)
    mlp = MLPClassifier(hidden_layer_sizes = 100, activation = 'tanh', 
solver = 'adam', alpha = 1e-5, batch_size = 36, tol = 1e-6, 
learning_rate_init=0.01,
                               learning_rate='constant', max_iter = 10000, 
random_state = 7)
    rf = RandomForestClassifier(n_estimators = 315, criterion = 'gini', 
max_depth = 14, min_samples_split = 3,
```

```
                                        min_samples_leaf = 3, max_features
= 'sqrt', random_state = 7)
    svc = SVC(kernel = 'rbf', C = 13, gamma = 'scale', probability = True,
random_state = 7)
    pipeline = make_pipeline(PolynomialFeatures(2),
LogisticRegression(solver='liblinear', random_state = 7))

    # Create a voting ensemble of the classifiers with soft voting.
Weighted average of predicted probabilities
    votingSystem = VotingClassifier(estimators=[('mlp', mlp), ('knn',
knn), ('svc', svc), ('rf', rf), ('pipeline', pipeline)], voting='soft')
    votingSystem.fit(attributeTrain, targetTrain)  # Fit the classifier to
the training data and target labels

    return votingSystem.predict(attributeTest)  # Return the predictions
for the test data

# Condense features by eliminating them based on importance
def condenseFeatures(attributeTrain, targetTrain, numFeatures):
    estimator = LogisticRegression(max_iter=1000)  # Instantiate logistic
regression as the estimator

    # Instantiate RFE with logistic regression estimator and recursively
select top _ features, then fit RFE to the data
    rfe = RFE(estimator, n_features_to_select = numFeatures)
    rfe.fit(attributeTrain, targetTrain)

    return rfe.support_  # Return selected features

# Print the confusion matrix, classification report, accuracy score,
heatmap, and matrix display
def printResults(targetTest, prediction):
    confusionMatrix = confusion_matrix(targetTest, prediction)  # Create
confusion matrix

    # Display the confusion matrix, a classification report, and the
overall accuracy score of the prediction
    print("\nConfusion Matrix:\n", confusionMatrix)
    print("\nClassification Report:\n", classification_report(targetTest,
prediction))
```

```
    print("Accuracy Score:", accuracy_score(targetTest, prediction))

    # Display a heatmap using matplotlib and the sklearn toolset to
display a confusion matrix
    matrixDisplay = ConfusionMatrixDisplay(confusion_matrix =
confusionMatrix)
    fig, ax = plt.subplots(figsize=(10, 8))  # Create layout and structure
figure
    matrixDisplay.plot(ax = ax, cmap = 'Blues')  # Create Plot
    # Plot labels
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Voting Confusion Matrix')
    plt.savefig('confusion_matrix.png')  # Save plot as png
    # plt.show()  # Display plot

    # Display Heatmap using Seaborn
    # sb.heatmap(confusionMatrix, annot = False, fmt = 'd', cmap =
'Blues', cbar = False)  # Create heatmap with Seaborn
    # plt.savefig('heatmap.png')  # Save plot as png
    # plt.show()  # Display plot

# main
foodData = pd.read_csv('FoodTypeDataset.csv')  # Read in data form csv
file

foodData.columns = ["column" + str(i + 1) for i in range(0,
len(foodData.columns) - 1)] + ["Target"]  # Set up a list to store column
labels (numbered columns)

attributes = foodData.iloc[:, :-1]  # Stores all the attributes, as in,
every column except target
target = foodData.iloc[:, -1]  # Stores last column (Target)

# Declare train and test variables, then split the datasets into training
and testing subsets
attributeTrain, attributeTest, targetTrain, targetTest =
train_test_split(attributes, target, test_size = .2, shuffle = True,
random_state = 7)
```

```
scaler = StandardScaler()  # Instantiate a StandardScaler object
scaler.fit(attributeTrain)  # Fit the scaler to the training data to
compute the mean and standard deviation
# Scale the training data and testing data using the mean and standard
deviation
attributeTrain = scaler.transform(attributeTrain)
attributeTest = scaler.transform(attributeTest)

# Get and Use only the top features for the data set, as in, the most
correlated and informative features
# topFeatures = condenseFeatures(attributeTrain, targetTrain, 10)

# attributeTrain = attributeTrain[:, topFeatures]
# attributeTest = attributeTest[:, topFeatures]

# Get predictions for classifiers and print the results
# prediction = knnClassifier(attributeTrain, attributeTest, targetTrain)
# printResults(targetTest, prediction)

# prediction = mlpClassifier(attributeTrain, attributeTest, targetTrain)
# printResults(targetTest, prediction)

# prediction = rfClassifier(attributeTrain, attributeTest, targetTrain)
# printResults(targetTest, prediction)

# prediction = svcClassifier(attributeTrain, attributeTest, targetTrain)
# printResults(targetTest, prediction)

# prediction = logRegClassifier(attributeTrain, attributeTest,
targetTrain)
# printResults(targetTest, prediction)

# prediction = pipelineClassifier(attributeTrain, attributeTest,
targetTrain)
# printResults(targetTest, prediction)

prediction = votingClassifier(attributeTrain, attributeTest, targetTrain)
printResults(targetTest, prediction)
```

Results:

```
Classification Report:
              precision    recall  f1-score   support

           1       0.22      0.25      0.24         8
           2       0.38      0.50      0.43         6
           3       0.56      0.69      0.62        13
           4       0.62      0.89      0.73         9
           5       0.42      0.62      0.50         8
           6       1.00      0.36      0.53        11
           7       0.70      0.70      0.70        10
           8       0.50      0.80      0.62         5
           9       0.67      0.80      0.73        10
          10       0.11      0.10      0.11        10
          11       0.57      0.29      0.38        14
          12       0.43      0.38      0.40         8
          13       0.38      0.30      0.33        10
          14       0.44      0.50      0.47         8
          15       0.60      0.60      0.60        10
          16       1.00      0.78      0.88         9
          17       0.80      0.73      0.76        11
          18       0.91      0.77      0.83        13
          19       0.67      1.00      0.80         6
          20       1.00      1.00      1.00         8

    accuracy                           0.59       187
   macro avg       0.60      0.60      0.58       187
weighted avg       0.61      0.59      0.58       187

Accuracy Score: 0.5882352941176471
```

```
Classification Report:
              precision    recall  f1-score   support

           1       0.50      0.38      0.43         8
           2       0.40      0.67      0.50         6
           3       0.67      0.46      0.55        13
           4       0.73      0.89      0.80         9
           5       0.53      1.00      0.70         8
           6       0.89      0.73      0.80        11
           7       0.57      0.40      0.47        10
           8       0.50      0.80      0.62         5
           9       0.75      0.90      0.82        10
          10       0.43      0.60      0.50        10
          11       0.50      0.29      0.36        14
          12       0.67      0.25      0.36         8
          13       0.55      0.60      0.57        10
          14       0.56      0.62      0.59         8
          15       0.45      0.50      0.48        10
          16       0.78      0.78      0.78         9
          17       0.82      0.82      0.82        11
          18       1.00      0.85      0.92        13
          19       0.71      0.83      0.77         6
          20       1.00      0.75      0.86         8

    accuracy                           0.64       187
   macro avg       0.65      0.66      0.63       187
weighted avg       0.66      0.64      0.63       187

Accuracy Score: 0.6417112299465241
```

Classification Report:

	precision	recall	f1-score	support
1	0.50	0.38	0.43	8
2	0.57	0.67	0.62	6
3	0.60	0.69	0.64	13
4	0.73	0.89	0.80	9
5	0.58	0.88	0.70	8
6	0.71	0.45	0.56	11
7	0.60	0.60	0.60	10
8	0.67	0.80	0.73	5
9	0.69	0.90	0.78	10
10	0.46	0.60	0.52	10
11	0.71	0.36	0.48	14
12	1.00	0.38	0.55	8
13	0.45	0.50	0.48	10
14	0.50	0.62	0.56	8
15	0.45	0.50	0.48	10
16	1.00	0.78	0.88	9
17	0.82	0.82	0.82	11
18	1.00	0.85	0.92	13
19	0.75	1.00	0.86	6
20	1.00	1.00	1.00	8
accuracy			0.67	187
macro avg	0.69	0.68	0.67	187
weighted avg	0.69	0.67	0.66	187

Accuracy Score: 0.6684491978609626

Classification Report:

	precision	recall	f1-score	support
1	0.50	0.38	0.43	8
2	0.57	0.67	0.62	6
3	0.64	0.69	0.67	13
4	0.58	0.78	0.67	9
5	0.54	0.88	0.67	8
6	0.71	0.45	0.56	11
7	0.67	0.40	0.50	10
8	0.50	0.80	0.62	5
9	0.69	0.90	0.78	10
10	0.50	0.50	0.50	10
11	0.67	0.43	0.52	14
12	0.67	0.25	0.36	8
13	0.45	0.50	0.48	10
14	0.56	0.62	0.59	8
15	0.44	0.40	0.42	10
16	0.64	0.78	0.70	9
17	0.75	0.82	0.78	11
18	0.92	0.85	0.88	13
19	0.75	1.00	0.86	6
20	1.00	0.88	0.93	8
accuracy			0.64	187
macro avg	0.64	0.65	0.63	187
weighted avg	0.65	0.64	0.63	187

Accuracy Score: 0.6363636363636364

Classification Report:

	precision	recall	f1-score	support
1	0.57	0.50	0.53	8
2	0.50	0.67	0.57	6
3	0.53	0.69	0.60	13
4	0.80	0.89	0.84	9
5	0.58	0.88	0.70	8
6	0.86	0.55	0.67	11
7	0.57	0.40	0.47	10
8	0.44	0.80	0.57	5
9	0.69	0.90	0.78	10
10	0.40	0.40	0.40	10
11	0.86	0.43	0.57	14
12	0.75	0.38	0.50	8
13	0.50	0.50	0.50	10
14	0.42	0.62	0.50	8
15	0.44	0.40	0.42	10
16	1.00	0.78	0.88	9
17	0.82	0.82	0.82	11
18	1.00	0.85	0.92	13
19	0.75	1.00	0.86	6
20	1.00	1.00	1.00	8
accuracy			0.66	187
macro avg	0.67	0.67	0.65	187
weighted avg	0.69	0.66	0.66	187

Accuracy Score: 0.6577540106951871

Classification Report:

	precision	recall	f1-score	support
1	0.60	0.38	0.46	8
2	0.18	0.33	0.24	6
3	0.56	0.38	0.45	13
4	0.78	0.78	0.78	9
5	0.33	0.62	0.43	8
6	0.75	0.55	0.63	11
7	0.44	0.40	0.42	10
8	0.44	0.80	0.57	5
9	0.75	0.90	0.82	10
10	0.38	0.50	0.43	10
11	0.67	0.14	0.24	14
12	0.33	0.12	0.18	8
13	0.40	0.40	0.40	10
14	0.46	0.75	0.57	8
15	0.40	0.40	0.40	10
16	0.73	0.89	0.80	9
17	0.67	0.73	0.70	11
18	1.00	0.69	0.82	13
19	0.57	0.67	0.62	6
20	0.89	1.00	0.94	8
accuracy			0.56	187
macro avg	0.57	0.57	0.54	187
weighted avg	0.59	0.56	0.55	187

Accuracy Score: 0.5561497326203209

Classification Report:

	precision	recall	f1-score	support
1	0.38	0.38	0.38	8
2	0.38	0.50	0.43	6
3	0.55	0.46	0.50	13
4	0.67	0.89	0.76	9
5	0.27	0.50	0.35	8
6	0.50	0.36	0.42	11
7	0.22	0.20	0.21	10
8	0.67	0.80	0.73	5
9	0.75	0.90	0.82	10
10	0.40	0.40	0.40	10
11	0.71	0.36	0.48	14
12	0.67	0.25	0.36	8
13	0.67	0.40	0.50	10
14	0.55	0.75	0.63	8
15	0.42	0.50	0.45	10
16	0.58	0.78	0.67	9
17	0.75	0.82	0.78	11
18	0.80	0.31	0.44	13
19	0.38	0.83	0.53	6
20	1.00	0.88	0.93	8
accuracy			0.54	187
macro avg	0.56	0.56	0.54	187
weighted avg	0.58	0.54	0.53	187

Accuracy Score: 0.5401069518716578