Regression, Classification and Clustering

Introduction to Data Science – CS 418

Project 2

Submitted by:

Yukthi Papanna Suresh

Emily Lin

Caglar Kurtkaya

INDEX

Abstract:

Used merged data from previous Exploratory data analysis to predict Democratic votes and Republican votes for each county. Tried and explored different regression, classification and clustering techniques. Finally, predicted classes for given test data.

Tasks:

Task 1:

Partition the merged dataset into a training set and a validation set using the holdout method or the cross-validation method :

Used cross-validation method to partition the dataset into 10 folds.

Cross-validation is preferred method because it gives the model the opportunity to train on multiple train-test splits. This gives you a better indication of how well your model will perform on unseen data. Hold-out, on the other hand, is dependent on just one train-test split.

These are the 10 folds:

```
[1075, 120]
[1075, 120]
[1075, 120]
[1075, 120]
[1075, 120]
[1076, 119]
[1076, 119]
[1076, 119]
[1076, 119]
[1076, 119]
```

Task 2:

Standardize the training set and the validation set:

Dropped columns ‘State’ and ‘County’ as they are of object type and cannot be used for standardization and then standardized the data.

Task 3:

Regression:

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for simple linear regression model, it is highest for the model with 'Total Population' as the predictor.
Hence, for simple linear regression, to predict number of votes cast for Democratic party, the best performance is given by the model with predictor variable as 'Total Population'.

```
#Evaluating with predictor 'Total Population'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,1].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96233051 0.62586705 0.9337941  0.93559838 0.94522807 0.91697893
 0.86543006 0.8268442  0.85791933 0.8712226 ]
[0.8741213225980164, 0.09285786260366721]
```

To select the predictor variable, used **correlation matrix** where each column is compared with the column having Democratic votes.

Used the same correlation matrix to try different combinations of variables to predict number of votes for Democratic party for each county.

Also, used **OLS regression** results to identify significant predictor variables for multiple linear regression assuming a significance level of 0.05.

Tried LASSO regression but did not perform well as none of the coefficients were estimated to exactly zero and for few gave negative coefficients.

The best performing model for predicting number of votes for Democratic party is multiple linear model with the predictor variables - 'Total Population', "Percent Less than bachelor's degree", 'Percent Age 29 and Under'.

```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree','Percent Age 29 and Under'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12,7]].reshape(-1,3), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96768806 0.65080786 0.93369853 0.85900243 0.95002191 0.93012499
 0.88725169 0.83674902 0.86704169 0.88410078]
[0.8766486980131882, 0.08541305723112652]
```

The performance of the model is given by R-squared(coefficient of determination). As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for multiple linear regression model is 87.66% i.e. this percentage of variance in observations is explained by the model.

Repeated the same tasks for Republican party and got the below results.

```
#Evaluating with predictor 'Total Population'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,1].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.87526991  0.645852    0.90664131  0.78602157 -0.01733154  0.91508408
  0.90662245  0.88928016  0.93398776  0.80359756]
[0.7645025255819977, 0.27330831959314233]
```

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for simple linear regression model is highest for the model with 'Total Population' as the predictor. Hence, for simple linear regression, to predict number of votes cast for Republican party, the best performance is given by the model with predictor variable as 'Total Population'

The best performing model for predicting number of votes for Republican party is multiple linear model with the predictor variables - as 'Total Population', "Percent Less than Bachelor's Degree", 'Percent Age 29 and Under', 'Percent Rural', 'Percent Foreign Born', 'Median Household Income'*.

```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree',
#'Percent Age 29 and Under','Percent Rural','Percent Foreign Born','Median Household Income'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,5,9,11,12,13]].reshape(-1,6), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

The performance of the model is given by R-squared(coefficient of determination). As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for multiple linear regression model is 78.72% i.e. this percentage of variance in observations is explained by the model.

Task 4:

Classification:

Tried different classifiers and results are as shown:

1. Decision Tree classifier:

```
#For identifying best performing parameters using GridSearchCV
parameters={'min_samples_split' : range(10,500,20),'max_depth': range(1,20,2), 'criterion':['entropy', 'gini']}
clf_tree=DecisionTreeClassifier()
gridsearch = GridSearchCV(estimator=clf_tree, param_grid = parameters, scoring='accuracy', cv=folds)
gridSearch = gridsearch.fit(X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.802510460251046 {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10}
```

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_DT)                    #Accuracy
error = 1 - accuracy                                              #Error
precision = metrics.precision_score(y, y_pred_DT, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_DT, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_DT, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.802510460251046, 0.197489539748954, array([0.83580508, 0.67729084]), array([0.90689655, 0.52307692]), array([0.86990077,
0.59027778])]
```

2. SVM classifier:

```
Svm = SVC(random_state=0)
#Find out the best model for SVM
parameters = [{'C':[1,10,100], 'kernel': ['rbf', 'linear']}]
gridsearch = GridSearchCV(estimator = Svm, param_grid = parameters, scoring = 'accuracy', cv = folds)
gridSearch = gridsearch.fit(X=x[:,[1,13,12,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.8100418410041841 {'C': 10, 'kernel': 'rbf'}
```

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_SVM)                    #Accuracy
error = 1 - accuracy                                               #Error
precision = metrics.precision_score(y, y_pred_SVM, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_SVM, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_SVM, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8100418410041841, 0.18995815899581592, array([0.82117882, 0.75257732]), array([0.94482759, 0.44923077]), array([0.8786745
1, 0.56262042])]
```

3. kNN classifier:

```
#Find out the best model for KNN
parameters = [{'n_neighbors':range(10,500,20)}]
gridsearch = GridSearchCV(estimator = knn_cv, param_grid = parameters, scoring = 'accuracy', cv = folds)
gridSearch = gridsearch.fit(X=x[:,[1,13,12,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.796652719665272 {'n_neighbors': 10}
```

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_KNN)                    #Accuracy
error = 1 - accuracy                                               #Error
precision = metrics.precision_score(y, y_pred_KNN, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_KNN, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_KNN, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.796652719665272, 0.20334728033472804, array([0.81008902, 0.72282609]), array([0.94137931, 0.40923077]), array([0.8708134
, 0.52259332])]
```

4. Naïve Bayes classifier:

```
#Using 5 most correlated variables as predictors
NB_cv = GaussianNB()
#train model with cv of 10
cv_scores = cross_val_score(NB_cv, X=x[:,[1,12,13,5,2]].reshape(-1,5), y = y, cv = folds)
#print each cv score (accuracy) and average them
print(cv_scores)
print([scores.mean(), scores.std()])
```

```
[0.84166667 0.80833333 0.61666667 0.875      0.8        0.89915966
 0.77310924 0.84033613 0.72268908 0.65546218]
[0.7872002729610357, 0.23348847469633535]
```

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_NB)                    #Accuracy
error = 1 - accuracy                                              #Error
precision = metrics.precision_score(y, y_pred_NB, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_NB, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_NB, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7832635983263598, 0.21673640167364017, array([0.81989529, 0.6375    ]), array([0.9       , 0.47076923]), array([0.8580821
9, 0.54159292])]
```

The best performing classification model is SVM with parameters 'C' -10 and kernel 'rbf' with predictor variables 'Percent White, not Hispanic or Latino', 'Percent Foreign Born', 'Total Population', 'Percent Rural', "Percent Less than bachelor's degree.

The performance of the model is with accuracy of 81% and F1-score of 0.878 for Democr atic and 0.563 for Republican.

We must look at both Accuracy and F1-scores while evaluating the models, as we have imbalanced classes in the data, and we chose the model that has higher F1-scores for both classes.

In order to hyper tune and select the parameters of the model, used **GridSearchCV** to return b oth best score and best parameters.

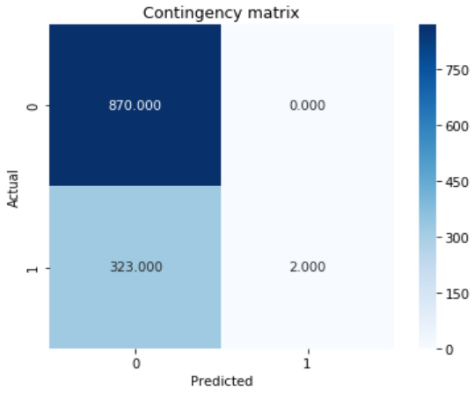
In order to select the variables, used the **correlation matrix** and used the 5 most correlated va riables with the 'Party' variable as the response variable.

Task 5:

Clustering:

Tried various techniques and results are as shown:

1. Hierarchical clustering – single linkage




```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

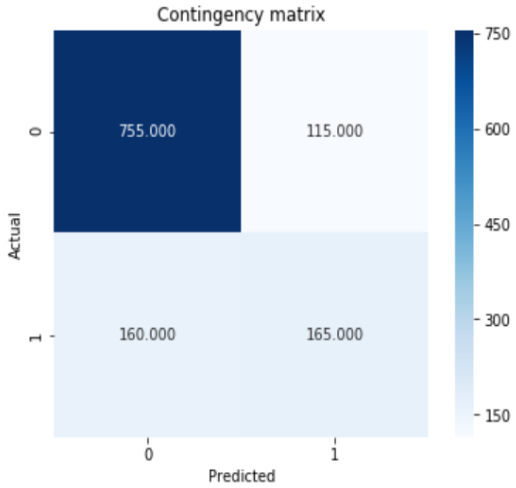
```
0.005608925119335567
0.8040606718628662
```

2. Hierarchical clustering – complete linkage

   Got similar results as above.

In general, hierarchical clustering performed well in separating the observations into two distinct separable clusters as we observe silhouette coefficient is 0.8 close to 1.

3. KMeans clustering:

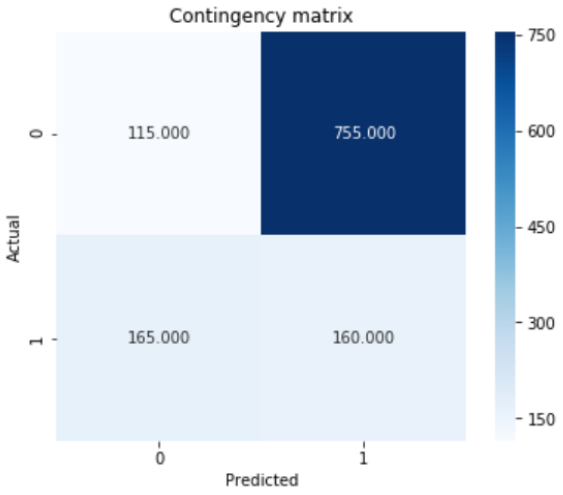


```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.24672990896646144
0.45221355231267646
```

KMeans clustering with multiple iterations gave better results in matching observations to true clusters. But still seems quite random in nature meaning the observations were assigned to clusters randomly.

4. DBSCAN:



Performed bad in terms of both supervised and unsupervised evaluation metrics.

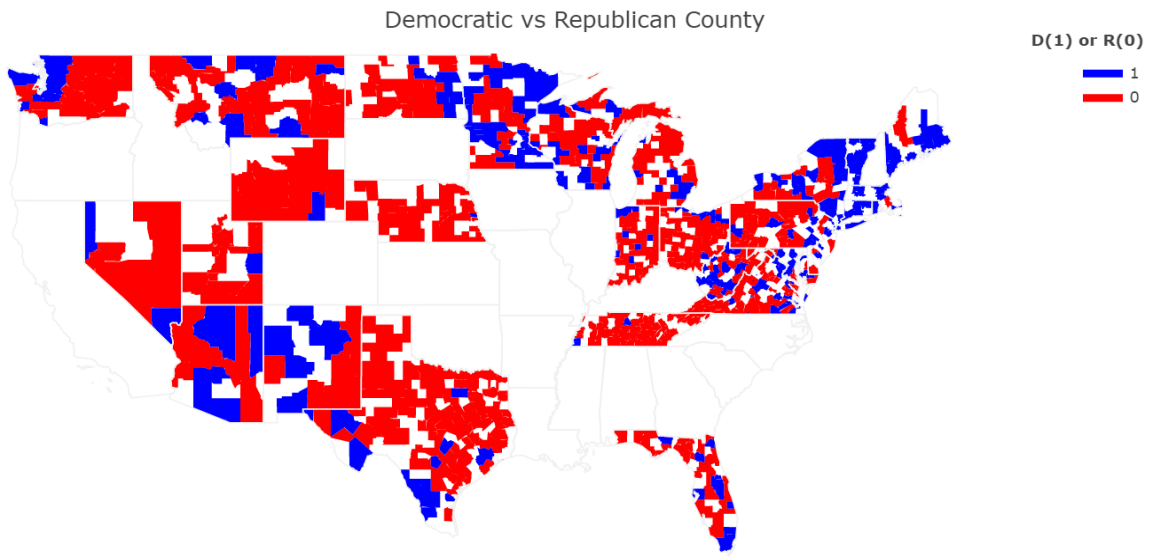
The parameters were selected on a trial and error basis.

The variables were selected from previous tasks – Task 3 & Task 4 and then experimented.

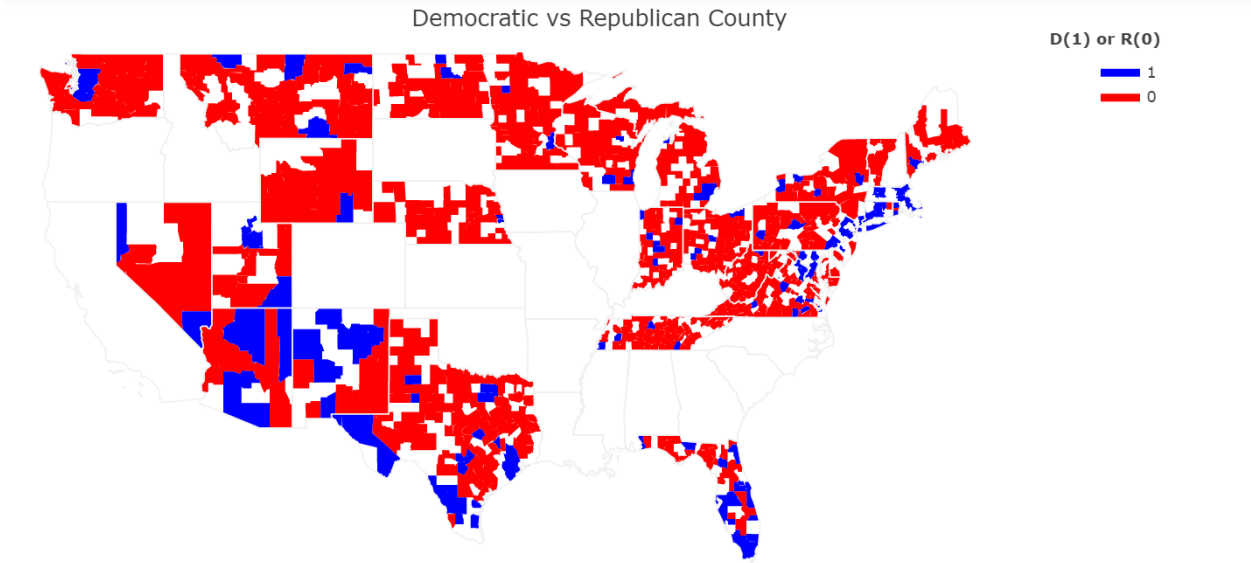
None of the clustering techniques gave good results with respect to matching observations with their true class labels , so this data is not good for clustering.

Task 6:

Map with true values:



Map with predicted values using SVM (best classifier):



Comparing the maps, we see that SVM classifier misclassified a little more for Democratic party than Republican party within the given error rate.

Task 7:

Use your best performing regression and classification models to predict :

Replaced negative values with zeros.

	State	County	Democratic	Republican	Party
0	NV	eureka	0.000000	0.000000	0
1	TX	zavala	0.000000	0.000000	0
2	VA	king george	9180.291751	4111.663448	0
3	OH	hamilton	242020.204400	153849.117662	1
4	TX	austin	336.646009	0.000000	0
...	...	...	...	...	...
395	VT	chittenden	61017.458161	42756.939923	1
396	OH	butler	107678.994980	70178.618865	0
397	NE	franklin	0.000000	7598.262280	0
398	MD	cecil	22984.988641	16177.600272	0
399	NY	yates	0.000000	11165.006159	0