MLady-cs4780

December 15, 2020

CS 4780/5780 Final Project:

Election Result Prediction for US Counties

Names and NetIDs for your group members: coa22 and mia27

Introduction:

The final project is about conducting a real-world machine learning project on your own, with everything that is involved. Unlike in the programming projects 1-5, where we gave you all the scaffolding and you just filled in the blanks, you now start from scratch. The programming project provide templates for how to do this, and the most recent video lectures summarize some of the tricks you will need (e.g. feature normalization, feature construction). So, this final project brings realism to how you will use machine learning in the real world.

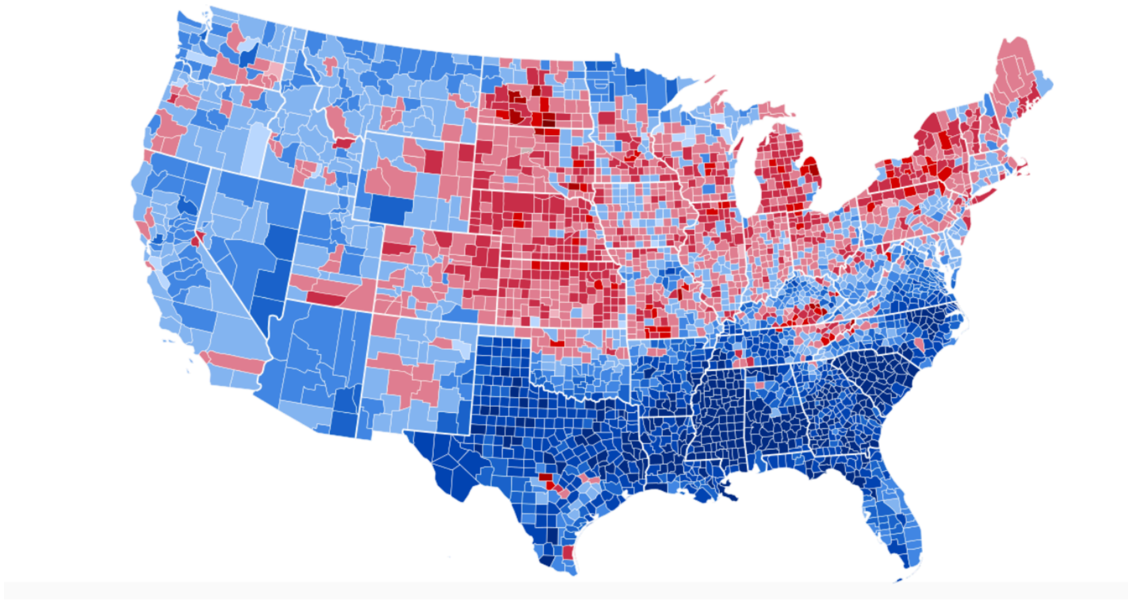
The task you will work on is forecasting election results. Economic and sociological factors have been widely used when making predictions on the voting results of US elections. Economic and sociological factors vary a lot among counties in the United States. In addition, as you may observe from the election map of recent elections, neighbor counties show similar patterns in terms of the voting results. In this project you will bring the power of machine learning to make predictions for the county-level election results using Economic and sociological factors and the geographic structure of US counties.

Your Task:

Plase read the project description PDF file carefully and make sure you write your code and answers to all the questions in this Jupyter Notebook. Your answers to the questions are a large portion of your grade for this final project. Please import the packages in this notebook and cite any references you used as mentioned in the project description. You need to print this entire Jupyter Notebook as a PDF file and submit to Gradescope and also submit the ipynb runnable version to Canvas for us to run.

Due Date:

The final project dataset and template jupyter notebook will be due on December 15th . Note that no late submissions will be accepted and you cannot use any of your unused slip days before.



1.1 Import:

Please import necessary packages to use. Note that learning and using packages are recommended but not required for this project. Some official tutorial for suggested packacges includes:

https://scikit-learn.org/stable/tutorial/basic/tutorial.html

https://pytorch.org/tutorials/

https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

[4]:
```
# Base Imports
import os
import pandas as pd
import numpy as np
import sklearn as sk
```

1.2 Weighted Accuracy:

Since our dataset labels are heavily biased, you need to use the following function to compute weighted accuracy throughout your training and validation process and we use this for testing on Kaggle.

[12]:
```
def weighted_accuracy(true, pred):
    assert(len(pred) == len(true))
    num_labels = len(true)
    num_pos = sum(true)
    num_neg = num_labels - num_pos
    frac_pos = num_pos/num_labels
    weight_pos = 1/frac_pos
    weight_neg = 1/(1-frac_pos)
    num_pos_correct = 0
    num_neg_correct = 0
    for pred_i, true_i in zip(pred, true):
        num_pos_correct += (pred_i == true_i and true_i == 1)
        num_neg_correct += (pred_i == true_i and true_i == 0)
    weighted_accuracy = ((weight_pos * num_pos_correct)
                         + (weight_neg * num_neg_correct))/((weight_pos *␣
 ↪num_pos) + (weight_neg * num_neg))
    return weighted_accuracy
```

Part 2: Baseline Solution

Note that your code should be commented well and in part 2.4 you can refer to your comments. (e.g. # Here is SVM, # Here is validation for SVM, etc). Also, we recommend that you do not to use 2012 dataset and the graph dataset to reach the baseline accuracy for 68% in this part, a basic solution with only 2016 dataset and reasonable model selection will be enough, it will be great if you explore thee graph and possibly 2012 dataset in Part 3.

2.1 Preprocessing and Feature Extraction:

Given the training dataset and graph information, you need to correctly preprocess the dataset (e.g. feature normalization). For baseline solution in this part, you might not need to introduce extra features to reach the baseline test accuracy.

[14]:
```
from sklearn import preprocessing
```

[15]:
```
# You may change this but we suggest loading data with the following code and␣
 ↪you may need to change
# datatypes and do necessary data transformation after loading the raw data to␣
 ↪the dataframe.
dataset_paths = [
    "./in/test_2016_no_label.csv",
    "./in/train_2016.csv",
    "./in/sampleSubmission.csv",
    "./in/graph.csv",
    "./in/test_2012_no_label.csv",
    "./in/train_2012.csv"
]

test = pd.read_csv(dataset_paths[0], sep=',', encoding='unicode_escape',␣
 ↪thousands=',')
train = pd.read_csv(dataset_paths[1], sep=',', encoding='unicode_escape',␣
 ↪thousands=',')


# Columns that make sense to standardize
data_range =␣
 ↪['MedianIncome','MigraRate','BirthRate','DeathRate','BachelorRate','UnemploymentRate']
```

```
# Make sure you comment your code clearly and you may refer to these comments␣
↪in the part 2.4
# TODO


# Get the Labels for the training data
yTr = (train["DEM"] > train["GOP"]).astype(int).values

# Scaling Training,Testing Features based on training data
scaler = preprocessing.StandardScaler()
xTr = scaler.fit_transform(train[data_range])
xTe = scaler.transform(test[data_range])
```

2.2 Use At Least Two Training Algorithms from class:

You need to use at least two training algorithms from class. You can use your code from previous projects or any packages you imported in part 1.1.

[19]:
```
from sklearn import svm,tree
```

[18]:
```
# Make sure you comment your code clearly and you may refer to these comments␣
↪in the part 2.4
# TODO

svmClf = svm.SVC()
svmClf.fit(xTr, yTr)
sv = svmClf.predict(xTe)
print(sv)


treeClf = tree.DecisionTreeClassifier()
treeClf.fit(xTr, yTr)
tr = treeClf.predict(xTe)
print(tr)
```

```
[0 0 0 … 0 0 0]
[0 0 0 … 0 0 0]
```

2.3 Training, Validation and Model Selection:

You need to split your data to a training set and validation set or performing a cross-validation for model selection.

[25]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import make_scorer
```

[29]:
```
#  Test svm c parmater and kernel combinations
#  Uses cross validation fucntion from scikit
#  score validation using weighted_accuracy and 5 setf of kfolds
#  take the mean score as the accuracy with those parameters.
#  save best kernel,c combination

n, = yTr.shape

# Loss Function/ Accuracy function
score = make_scorer(weighted_accuracy)
```

```
[33]: ####       SVM     #####

# K-Fold
k = 5

# Model Params
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# Through trial and error we limited out linspace to test high
# performing C parameters
cList = np.linspace(12,18,10)

# Acumulators
best_c = None
best_kernel = None
acc = None

#Tuning
for kernel in kernels:
    for c in cList :
        svmTemp = svm.SVC(C = c, kernel = kernel)
        curr_score = cross_val_score(svmTemp, xTr, yTr, scoring = score, cv=k).
 ↪mean()
        if (acc is None) or (acc < curr_score):
            acc = curr_score
            best_c = c
            best_kernel = kernel
print (acc, best_c, best_kernel)
# 0.7079030910609857 16.0 rbf

# Chosen SVM Model
svmClf = svm.SVC(C = best_c, kernel = best_kernel)
svmClf.fit(xTr,yTr)
sv = svmClf.predict(xTe)
# SVC(C=11.0)
```

```
0.7079030910609857 16.0 rbf
```

[37]:
```
#####       DECISION TREE       #####

# We care about pruning here actually so we need to add that as a tuning param␣
 ↪we can use a range and make it large
# max_depth and min sample split should be useful, I think max depth is what we␣
 ↪want tho

# K-Fold
k = n//50

# Model Params
criterion = ["gini", "entropy"]
splitter = ["best"]
depthList = np.linspace(1,1000000, 400).tolist() + [None]

# Accumulators
best_cri = None
best_spl = None
best_depth = None
acc = None

# Tuning
for cri in criterion:
    for split in splitter:
        for depth in depthList:
            treeTemp = tree.DecisionTreeClassifier(criterion = cri, splitter =␣
 ↪split, max_depth=depth)
            curr_score = cross_val_score(treeTemp, xTr, yTr, scoring = score,␣
 ↪cv=k).mean()
            if (acc is None) or (acc < curr_score):
                acc = curr_score
                best_cri = cri
                best_spl = split
                best_depth = depth
print (acc, best_cri, best_spl, best_depth)
# 0.7225814536340853 gini best 70708.0
# This does not perform as well on our dataset. The testing error isn't super␣
 ↪accurate (2% off)

# Chosen Tree Model
treeClf = tree.DecisionTreeClassifier(criterion = best_cri, splitter =␣
 ↪best_spl, max_depth=best_depth)
treeClf.fit(xTr,yTr)
tr = treeClf.predict(xTe)
```

```
0.7235201657557246 gini best 541353.8421052631
```

2.4 Explanation in Words:

You need to answer the following questions in the markdown cell after this cell:

2.4.1 How did you preprocess the dataset and features?

2.4.2 Which two learning methods from class did you choose and why did you made the choices?

2.4.3 How did you do the model selection?

2.4.4 Does the test performance reach a given baseline 68% performanc? (Please include a screenshot of Kaggle Submission)

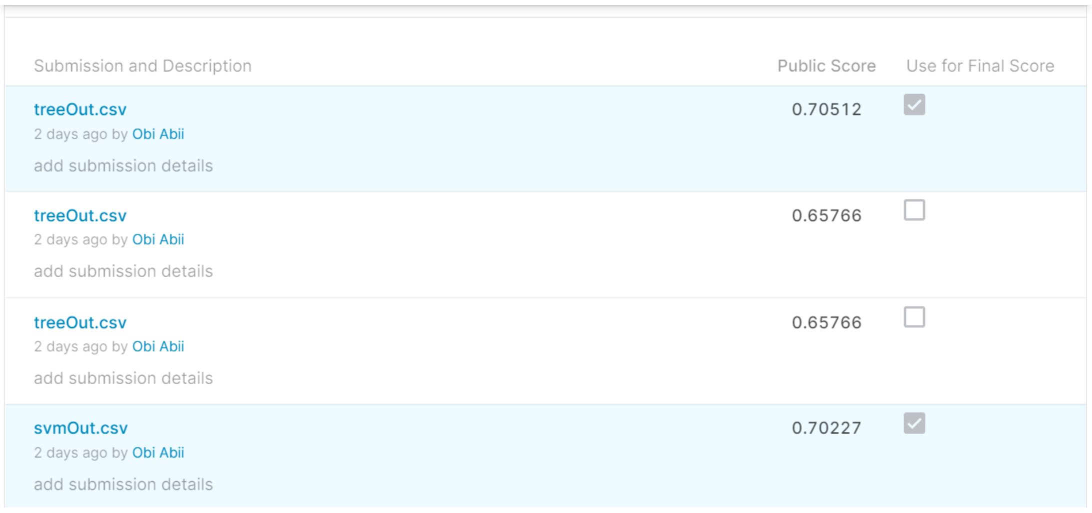
1 ANSWERS

2.4.1: We standardized the values of the real valued data. For instance, we didn't attempt to standardize the FIPS value because they were being used as indetifiers. We used the package standardScaler from scikit-learn and passed in the data frame of the relevant realvalued columns. We remembered to scale our testing data to the shape of of training data. This was easily faciliated by the functions from the package. We did not use the data from Dem of Gop number becuase we would not thave them for the testing data.

2.4.2: We use an SVM and a Decision Tree. We reasoned that the data would be some what linearly seprable hence the SVM. The Decision Tree made sense becuase we had relatively few features so over fitting would not be a major issue. We knew we neaded to learn a nonlinear rule and the decision trees and a kernelized svm would be able to do this.

2.4.3: We use scikit's functions to reason about model selection. We tuned multiple paramters looping and checking with 50-fold cross validation we tuned for the bet combinations.

2.4.4: Yes, our models reach the 68% baseline.



Submission and Description	Public Score	Use for Final Score
treeOut.csv 2 days ago by Obi Abii add submission details	0.70512	✓
treeOut.csv 2 days ago by Obi Abii add submission details	0.65766	☐
treeOut.csv 2 days ago by Obi Abii add submission details	0.65766	☐
svmOut.csv 2 days ago by Obi Abii add submission details	0.70227	✓

Part 3: Creative Solution

3.1 Open-ended Code:

You may follow the steps in part 2 again but making innovative changes like creating new features, using new training algorithms, etc. Make sure you explain everything clearly in part 3.2. Note that reaching the 75% creative baseline is only a small portion of this part. Any creative ideas will receive most points as long as they are reasonable and clearly explained.

[6]:
```
#### ADDITIONAL IDEAS ####

# We could compile a list of all DEM and all GOP and give ratings based on that
# We could see how the change in unemployment affects, change in birth rate␣
 ↪(high birth rate implies younger pop)
# Change in death rate
# Ratio of death to birth
# Migration also implies younger pop
# Income is important
# Get nearest neighbor from
```

[34]:
```
############### LOAD DATASETS ###############

graph = pd.read_csv(dataset_paths[3], sep=',', encoding='unicode_escape',␣
 ↪thousands=',')
test_2012 = pd.read_csv(dataset_paths[4], sep=',', encoding='unicode_escape',␣
 ↪thousands=',')
train_2012 = pd.read_csv(dataset_paths[5], sep=',', encoding='unicode_escape',␣
 ↪thousands=',')

# A hash for the county FIPS
counties = train["FIPS"].append(test["FIPS"]).to_frame().set_index("FIPS")
counties['i'] = list(range(len(counties.index)))
```

[35]:
```
################ FEATURE EXTRACTION FROM 2012 DATA ################

# We plan to use changes in the datasets between 2012 and 2016 as features

#### DATA ####
data_range =␣
 ↪['MedianIncome','MigraRate','BirthRate','DeathRate','BachelorRate','UnemploymentRate']
xTr_2012_delta =  train[data_range] - train_2012[data_range]
xTe_2012_delta = test[data_range] - test_2012[data_range]
### SCALING ###
scaler3 = preprocessing.StandardScaler()
xTr_2012_delta = scaler.fit_transform(xTr_2012_delta)
xTe_2012_delta = scaler.transform(xTe_2012_delta)

xTr_2012_use = np.concatenate((xTr, xTr_2012_delta), axis=1)
xTe_2012_use = np.concatenate((xTe, xTe_2012_delta), axis=1)
```

[36]:
```
############### FEATURE EXTRACTION FROM GRAPH.CSV ##################

#### PREDICTIONS SO FAR ####

# Best prediction so far
best_pred = sv

# Create indexes for training and testing data
train_i = train.set_index("FIPS")
train_i['i'] = list(range(len(train.index)))
test_i = test.set_index("FIPS")
test_i['i'] = list(range(len(test.index)))


## either returns the known label or our best prediction
"""
    A county's best known prediction
"""
def get_affiliation(county):
    in_training = county in train_i.index

    if in_training:
        label = yTr[train_i.i[county]]
    else:
        label = best_pred[test_i.i[county]]

    return -1 if label == 0 else 1
```

[38]:
```
#### EXTRACTING THE LABEL INFORMATION OF NEIGHBORS ####

# Accumulators
# a list of neighbors
counties['neighbors'] = [[] for x in range(len(counties.index))]
# a list of neighbors labels
counties['neighbor_labels'] = [0 for x in range(len(counties.index))]
# the number of neighbors we have information from
counties['confidence'] = [0 for x in range(len(counties.index))]


# Add in the data to our three new features
for ind in graph.index:
    # make sure the source and the destitnation are in our counties list
    # make sure we dont count the source as a destination to itself to avoid␣
 ↪corrupting the data
    if graph.SRC[ind] in counties.index and graph.DST[ind] in counties.index␣
 ↪and graph.DST[ind] != graph.SRC[ind]:
        # take the pair and append the neighbor to its list of neighbors
        # classify the neighbor off of the data or our previous sum
        # and add how many neighbors we saw to inform our model of the␣
↪confidence in this information and avoid false zeros
        if graph.SRC[ind] not in counties.neighbors[graph.DST[ind]]:
            counties.loc[graph.DST[ind], 'neighbors'].append(graph.SRC[ind])
            counties.loc[graph.DST[ind], 'neighbor_labels'] +=␣
↪get_affiliation(graph.SRC[ind])
            counties.loc[graph.DST[ind], 'confidence'] += 1

        if graph.DST[ind] not in counties.neighbors[graph.SRC[ind]]:
            counties.loc[graph.SRC[ind], 'neighbors'].append(graph.DST[ind])
            counties.loc[graph.SRC[ind],'neighbor_labels'] = counties.
↪neighbor_labels[graph.SRC[ind]] + get_affiliation(graph.DST[ind])
            counties.loc[graph.SRC[ind],'confidence'] += 1


# I think we don't need confidence
d_range = ['neighbor_labels', 'confidence']
xTr_counties_id = counties.i[train['FIPS']]
xTe_counties_id = counties.i[test['FIPS']]

xTr_gr = counties[d_range].values[xTr_counties_id]
xTe_gr = counties[d_range].values[xTe_counties_id]
# xTr_gr = counties[d_range][train['FIPS']]
# xTe_gr = counties[d_range][test['FIPS']]

### SCALING ###
scaler2 = preprocessing.StandardScaler()
xTr_gr = scaler.fit_transform(xTr_gr)
xTe_gr = scaler.transform(xTe_gr)

xTr_gr_use = np.concatenate((xTr, xTr_gr), axis=1)
xTe_gr_use = np.concatenate((xTe, xTe_gr), axis=1)

xTr_cr = np.concatenate((xTr_2012_use, xTr_gr), axis=1)
xTe_cr = np.concatenate((xTe_2012_use, xTe_gr), axis=1)
```

[39]:
```
##################### LOAD SCIPY ####################
from scipy.sparse.csgraph import shortest_path
```

[40]:
```
##################### GET ADJACENCY/DISTANCE MATRICES #########################

# Increase the index to accomodate for the counties that are not in the test␣
↪data
for ind in graph.index:
    if graph.SRC[ind] not in counties.index:
        counties.loc[graph.SRC[ind]] = ({'FIPS':graph.SRC[ind], 'i':␣
 ↪len(counties.index)})
    if graph.DST[ind] not in counties.index:
        counties.loc[graph.DST[ind]] = ({'FIPS':graph.DST[ind], 'i':␣
 ↪len(counties.index)})

# Create the adjacency matrix
n = len(counties.index)
adj_matrix = np.zeros((n,n))

# Construct Adjacency matrix
for ind in graph.index:
    adj_matrix[counties.i[graph.SRC[ind]]][counties.i[graph.DST[ind]]] =␣
 ↪adj_matrix[counties.i[graph.DST[ind]]][counties.i[graph.SRC[ind]]] = 1

# Distance matrix
dist_mat = shortest_path(adj_matrix,directed=True,method='auto',␣
 ↪unweighted=True)

# Find the neighbors that are 1 away and 2 away
two_away = (dist_mat<3).astype(int)
one_away = (dist_mat<2).astype(int)

# HUGE FEATURE SPACE
xTr_adj = np.concatenate((xTr, one_away[counties.i[train["FIPS"]]]), axis=1)
xTe_adj = np.concatenate((xTe, one_away[counties.i[test["FIPS"]]]), axis=1)
```

[ ]:
```
############       SVM -> Adjacency #FAILED     ###########
xTr_d = xTr_adj
xTe_d = xTe_adj
# K-Fold
k = 10

# Model Params
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# Through trial and error we limited out linspace to test high
# performing C parameters
cList = np.linspace(1,30,10)

# Acumulators
best_c = None
best_kernel = None
acc = None

# Tuning
for kernel in kernels:
    for c in cList :
        svmTemp = svm.SVC(C = c, kernel = kernel)
        curr_score = cross_val_score(svmTemp, xTr_d, yTr, scoring = score,␣
↪cv=k).mean()
        if (acc is None) or (acc < curr_score):
            acc = curr_score
            best_c = c
            best_kernel = kernel
print (acc, best_c, best_kernel)
# 0.7079030910609857 16.0 rbf

# Chosen SVM Model
svmClf = svm.SVC(C = 7, kernel = 'rbf')
svmClf.fit(xTr,yTr)
print(cross_val_score(svmClf, xTr_d, yTr, scoring = score, cv=k).mean())
sv_adj = svmClf.predict(xTe_d)
# SVC(C=11.0)
```

[155]:
```
############       SVM -> Graph DATA     ###########
xTr_d = xTr_gr_use
xTe_d = xTe_gr_use
# K-Fold
k = 10

# Model Params
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# Through trial and error we limited out linspace to test high
# performing C parameters
cList = np.linspace(1,10,13)

# Acumulators
best_c = None
best_kernel = None
acc = None

# Tuning
for kernel in kernels:
    for c in cList :
        svmTemp = svm.SVC(C = c, kernel = kernel)
        curr_score = cross_val_score(svmTemp, xTr_d, yTr, scoring = score,␣
↪cv=k).mean()
        if (acc is None) or (acc < curr_score):
            acc = curr_score
            best_c = c
            best_kernel = kernel
print (acc, best_c, best_kernel)
# 0.7079030910609857 16.0 rbf
# Chosen SVM Model
svmClf = svm.SVC(C = best_c, kernel = best_kernel)
svmClf.fit(xTr_d,yTr)
print(cross_val_score(svmClf, xTr_d, yTr, scoring = score, cv=k).mean())
sv_gr = svmClf.predict(xTe_d)
```

```
0.7367195161817588 7.0 rbf
0.7367195161817588
```

[156]:

```
############       SVM -> Change in Demographics     ###########
xTr_d = xTr_2012_use
xTe_d = xTe_2012_use
# K-Fold
k = 10

# Model Params
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# Through trial and error we limited out linspace to test high
# performing C parameters
cList = np.linspace(5,25,15)

# Acumulators
best_c = None
best_kernel = None
acc = None

# Tuning
for kernel in kernels:
    for c in cList :
        svmTemp = svm.SVC(C = c, kernel = kernel)
        curr_score = cross_val_score(svmTemp, xTr_d, yTr, scoring = score,␣
 ↪cv=k).mean()
        if (acc is None) or (acc < curr_score):
            acc = curr_score
            best_c = c
            best_kernel = kernel
print (acc, best_c, best_kernel)
# 0.7079030910609857 16.0 rbf

# Chosen SVM Model
svmClf = svm.SVC(C = best_c, kernel = best_kernel)
svmClf.fit(xTr_d,yTr)
print(cross_val_score(svmClf, xTr_d, yTr, scoring = score, cv=k).mean())
sv_2012 = svmClf.predict(xTe_d)
```

```
0.7774436090225564 19.285714285714285 rbf
0.7774436090225564
```

[157]:
```
############       SVM -> all changes     ###########
xTr_d = xTr_cr
xTe_d = xTe_cr
# K-Fold
k = 10

# Model Params
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# Through trial and error we limited out linspace to test high
# performing C parameters
cList = np.linspace(4,16,10)

# Acumulators
best_c = None
best_kernel = None
acc = None

# Tuning
for kernel in kernels:
    for c in cList :
        svmTemp = svm.SVC(C = c, kernel = kernel)
        curr_score = cross_val_score(svmTemp, xTr_d, yTr, scoring = score,␣
 ↪cv=k).mean()
        if (acc is None) or (acc < curr_score):
            acc = curr_score
            best_c = c
            best_kernel = kernel
print (acc, best_c, best_kernel)
# 0.7079030910609857 16.0 rbf

# Chosen SVM Model
svmClf = svm.SVC(C = best_c, kernel = best_kernel)
svmClf.fit(xTr_d,yTr)
print(cross_val_score(svmClf, xTr_d, yTr, scoring = score, cv=k).mean())
sv_all = svmClf.predict(xTe_d)
```

```
0.7766917293233083 13.333333333333332 rbf
0.7766917293233083
```

[114]:
```
#####       DECISION TREE ->  #Really LONG Training time      #####

xTr_d = xTr_adj
xTe_d = xTe_adj

# We care about pruning here actually so we need to add that as a tuning param␣
 ↪we can use a range and make it large
```

```
# max_depth and min sample split should be useful, I think max depth is what we␣
↪want tho

# K-Fold
k = n//50

# Model Params
criterion = ["gini"]
splitter = ["best"]
depthList = [541353.8421052631] + [None]

# Accumulators
best_cri = None
best_spl = None
best_depth = None
acc = None

# Tuning
for cri in criterion:
    for split in splitter:
        for depth in depthList:
            treeTemp = tree.DecisionTreeClassifier(criterion = cri, splitter =␣
↪split, max_depth=depth)
            curr_score = cross_val_score(treeTemp, xTr_d, yTr, scoring = score,␣
↪cv=k).mean()
            if (acc is None) or (acc < curr_score):
                acc = curr_score
                best_cri = cri
                best_spl = split
                best_depth = depth
print (acc, best_cri, best_spl, best_depth)
# 0.7225814536340853 gini best 70708.0
# This does not perform as well on our dataset. The testing error isn't super␣
↪accurate (2% off)

# Chosen Tree Model
treeClf = tree.DecisionTreeClassifier(criterion = best_cri, splitter =␣
↪best_spl, max_depth=best_depth)
treeClf.fit(xTr_d,yTr)
tr_cr = treeClf.predict(xTe_d)
```

```
0.7409598214285713 gini best 541353.8421052631
```

[160]:
```
##### NEURAL NETWORK #####
from sklearn.neural_network import MLPClassifier

#### INPUT FORMATS
```

```
# xTr_adj -> adjacency list
# xTr_2012_use -> 2012 change + xTr
# xTr_gr_use -> graph + xTr
# xTr_cr -> graph + 2012 change + xTr
```

```
[178]: ####      MLP NN -> all data   #####
xTr_d = xTr_cr
xTe_d = xTe_cr
k = 5



# We are not sure how exactly to tue the shape of the neural network
# We are using a random assortment of layer shapes and testing on that
hidden_layer_shapes = [(100,), (50,20),(100, 70, 50, 20), (200, 100, 50)]
activation_funcs = ['identity', 'logistic', 'tanh', 'relu']
solvers = ['lbfgs', 'adam']

# Accumulators
best_act = None
best_solver = None
best_shape = None
acc = None

for solver in solvers:
    for act_func in activation_funcs:
        for shape in hidden_layer_shapes:
            nnTemp = MLPClassifier(max_iter = 400, activation= act_func,␣
 ↪hidden_layer_sizes=shape, solver = solver, random_state=1)
            curr_score = cross_val_score(nnTemp, xTr_d, yTr, scoring = score,␣
 ↪cv=k).mean()
            if (acc is None) or (acc < curr_score):
                    acc = curr_score
                    best_act = act_func
                    best_solver = solver
                    best_shape = shape


nnClf = MLPClassifier(max_iter = 400, activation= best_act,␣
 ↪hidden_layer_sizes=best_shape, solver = best_solver, random_state=1)
nnClf.fit(xTr_d,yTr)
nn_cr = nnClf.predict(xTe_d)

print(best_act)
print(best_solver)
print(best_shape)
print(cross_val_score(nnClf, xTr_d, yTr, scoring = score, cv=k).mean())
```

```
0.8052046783625733
```

3.2 Explanation in Words:

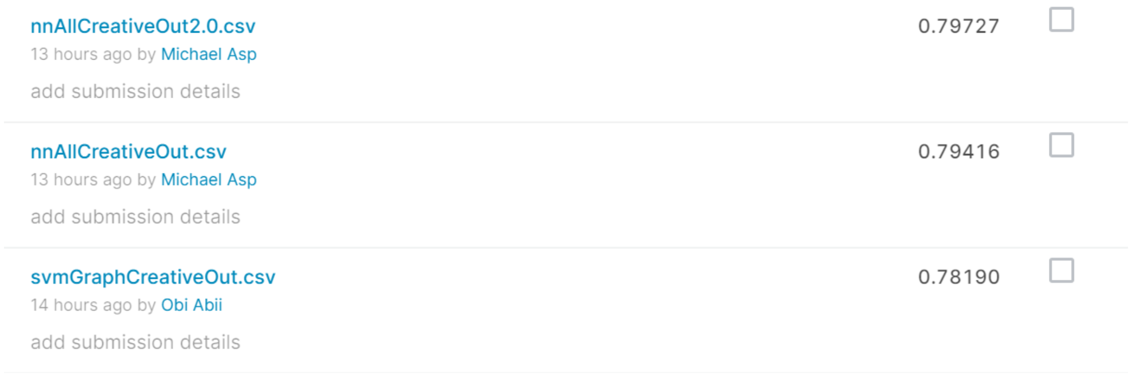
You need to answer the following questions in a markdown cell after this cell:

3.2.1 How much did you manage to improve performance on the test set compared to part 2? Did you reach the 75% accuracy for the test in Kaggle? (Please include a screenshot of Kaggle Submission)

3.2.2 Please explain in detail how you achieved this and what you did specifically and why you tried this.

2 ANSWERS

3.2.1: We managed to increase our accuracy by roughly 10 percent! We met the 75% accuracy rating



nnAllCreativeOut2.0.csv
13 hours ago by Michael Asp
add submission details
0.79727

nnAllCreativeOut.csv
13 hours ago by Michael Asp
add submission details
0.79416

svmGraphCreativeOut.csv
14 hours ago by Obi Abii
add submission details
0.78190

3.2.2: Intially we tried creating an adjacency matrix from the graph and passing that into an SVM as a feature. This caused a massive increase in the feature space. This did not improve our accuracy. Then we re-formulated the graph data more intelligently. We used the known labels of the neighbors as well as our best predictions of the unknown neighbors to create new features for each county. Additionally we computed the diffrences in the given six features from 2012 until 2016.

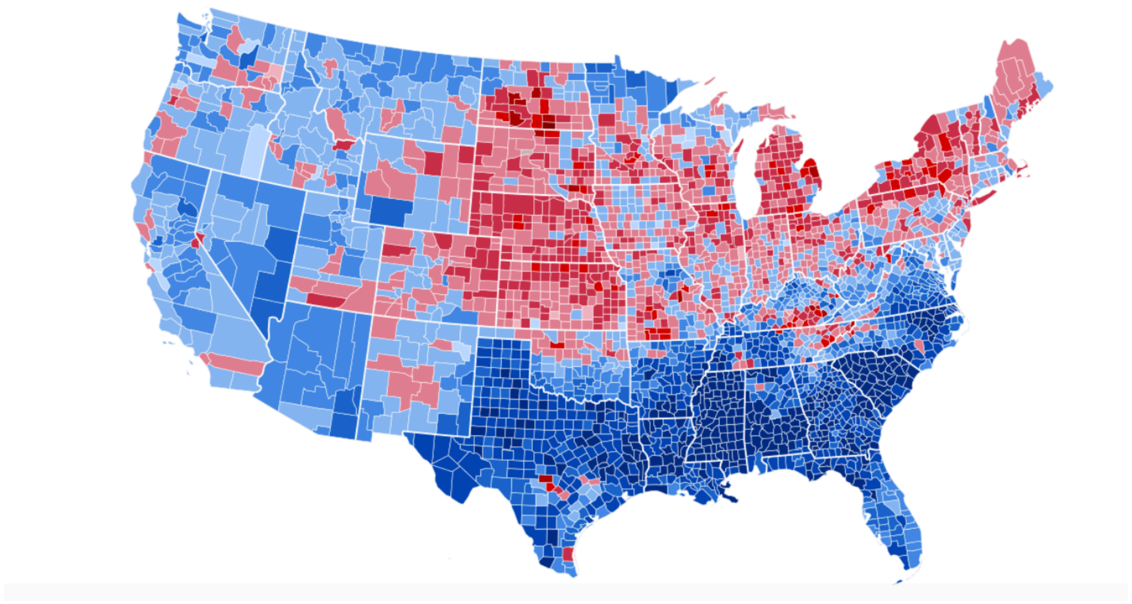
2.1 Feature Spaces

The list of creative feature combinations are as follows:

Feature	Contents
xTr__adj	adjacency list
xTr__2012__use	2012 change + xTr
xTr__gr__use	graph + xTr
xTr__cr	graph + 2012 change + xTr

2.1.1 Adjacency List

The rational behind getting graph data in the form of an adjacency list was that we believed that neihboring counties tend to vote alike (clustering) as can be seen in the graph provided. We thought that knowing what counties were next to each other regardless of its affilition would be a great indicator. We also tried creating a **distance matrix** and finding neighbors that were two away. The pitfall of this method was that it clogged up the feature space. we went from 6 -> ~3110 features. It was hard to train on and did not significantly raise the accuracy of our model when run on an SVM and Decision Tree.



2.1.2 Graph

This refers to the sum of known labels for the immediate neighbors of each county (we used prior svm predictions ~70% accuracy on the test data and true labels on the training data). The thought process behind this was similar to the adjacency matrix but by compiling what we knew into one feature, we both improved the accuracy and training time of our models compared to the adjacency list. **Important consideration** - We made republican counties count as a (-1) and Dems count as a (+1). WE believe there is more to be learned from *5* republican neighbors than from *2*. The same goes for democrats. This is a stylistic choice that we believe provides our models with more information that simply the majority label. - We noticed that some counties appear in the graph that we know nothing about, these counties count as *0* as they do not improve our confidence at all. - We also added a confidence variable. This variable distigushed between no knowledge situations. For instance we may have a zero in the case where we know nothing about our neighbors but this is not the same as the casee where we have seen two neighboring dem and rep counties. Intuitively, where there is no-knowledge about our neighbors our model may may skew toward a particular classification, however when the decision splits 50-50 and results in a zero this may skew in the other direction. Additionally the 5-4 case and 3-2 case both result in ones but the were arrived at by diffrent means. Therefore, we provide the number of neighbors (standardized) we have informtion about as a feature to our models.

2.1.3 2012 Change

The change in our original features from 2012 -> 2016: These are our hypothese on the data. Look at the initial comments at section **3.1**

2.1.4 xTr

Our original feature space

2.2 Results

Our two best models with these new feature were an svm trained and validated on ***xTr_gr*** and a neural network trained on ***xTr_cr***. These models were able to meet the bench mark handly. For both of the models we did extensive validation and parameter tuning.

Part 4: Kaggle Submission

You need to generate a prediction CSV using the following cell from your trained model and submit the direct output of your code to Kaggle. The CSV shall contain TWO column named exactly "FIPS" and "Result" and 1555 total rows excluding the column names, "FIPS" column shall contain FIPS of counties with same order as in the test_2016_no_label.csv while "Result" column shall contain the 0 or 1 prdicaitons for corresponding columns. A sample predication file can be downloaded from Kaggle.

[184]:
```
# TODO

# You may use pandas to generate a dataframe with FIPS and your predictions␣
↪first
# and then use to_csv to generate a CSV file.

out = pd.read_csv(dataset_paths[2], sep=',', encoding='unicode_escape',␣
↪thousands=',')
out['FIPS'] = test['FIPS']
out['Result'] = sv
out.to_csv("./out/svmOut.csv", index=False)
out['Result'] = tr
out.to_csv("./out/treeOut.csv", index=False)
out['Result'] = sv_adj
out.to_csv("./out/svmAdjCreativeOut.csv", index=False)
out['Result'] = sv_all
out.to_csv("./out/svmAllCreativeOut.csv", index=False)
out['Result'] = sv_gr
out.to_csv("./out/svmGraphCreativeOut.csv", index=False)
out['Result'] = sv_2012
out.to_csv("./out/svm2012CreativeOut.csv", index=False)
out['Result'] = nn_cr
out.to_csv("./out/nnAllCreativeOut.csv", index=False)
```

Part 5: Resources and Literature Used

```
[0]: https://scikit-learn.org/stable/index.html#

     CS 4780/5780 Introduction to Machine Learning (2020FA) lectures and slides

     https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

     https://stackoverflow.com/questions/10628262/
      ↪inserting-image-into-ipython-notebook-markdown

     https://stackoverflow.com/questions/15998491/
      ↪how-to-convert-ipython-notebooks-to-pdf-and-html/25942111

     https://datalore.jetbrains.com
```