

CS 4780/5780 Final Project:

Election Result Prediction for US Counties

Names and NetIDs for your group members: Cole DeMeulemeester (cmd328), Reza Madhavan (rm855), Kunal Sheth (ks782)

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.

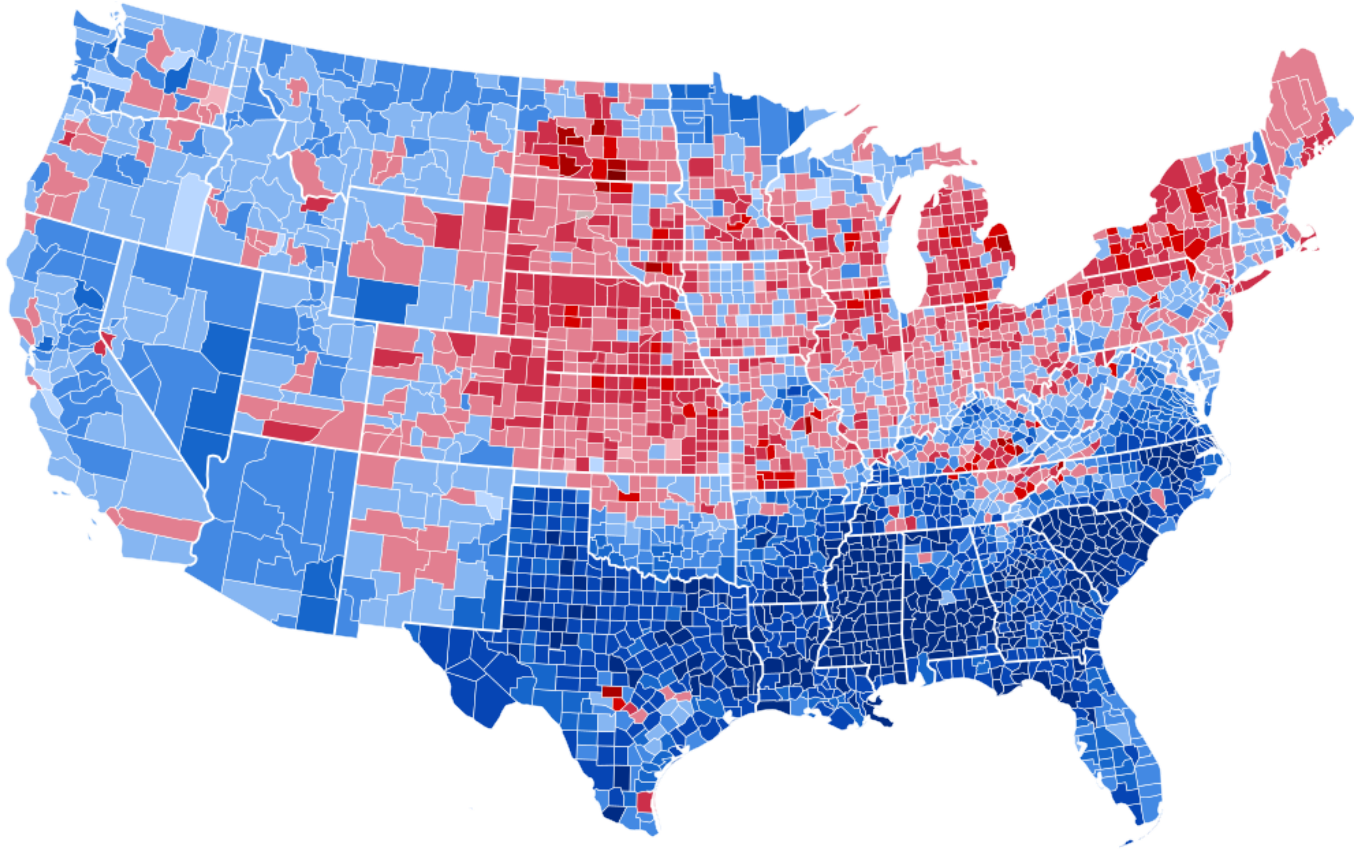
The task you will work on is forecasting election results. Economic and sociological factors have been widely used when making predictions on the voting results of US elections. Economic and sociological factors vary a lot among counties in the United States. In addition, as you may observe from the 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. </p>

Your Task:

Please 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.



Part 1: Basics

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 packages includes:

<https://scikit-learn.org/stable/tutorial/basic/tutorial.html> (<https://scikit-learn.org/stable/tutorial/basic/tutorial.html>)

<https://pytorch.org/tutorials/> (<https://pytorch.org/tutorials/>)

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

```
In [2]: import os
import pandas as pd
import numpy as np
# import SVC, KNN, preprocessing libraries for basic solution
from sklearn import preprocessing as pp
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

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.

```
In [3]: def weighted_accuracy(pred, true):
    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.

```
In [4]: # 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.  
def import_train_data(file):  
    df = pd.read_csv(file, sep=',', header=0, encoding='unicode_escape')  
    df["Result"] = df["DEM"] > df["GOP"]  
    df["Result"] = df["Result"].astype("int32")  
    df["MedianIncome"] = df["MedianIncome"].str.replace(",", "").astype("int3  
2")  
    return df  
  
# Make sure you comment your code clearly and you may refer to these comments  
in the part 2.4  
df = import_train_data("train_2016.csv")
```

```
In [7]: # normalizing the data
def scale_data(df,cols):
    topprocess = df[cols]
    scaler = pp.StandardScaler().fit(topprocess)
    return pd.DataFrame(scaler.transform(topprocess), columns = cols)

basic_cols = ['MedianIncome', 'MigraRate', 'BirthRate', 'DeathRate', 'Bachelor
Rate', 'UnemploymentRate']
dfProcessed = scale_data(df,basic_cols)
dfProcessed
```

Out[7]:

	MedianIncome	MigraRate	BirthRate	DeathRate	BachelorRate	UnemploymentRate
0	0.188697	0.416162	0.502519	0.056358	-0.065012	-0.518119
1	0.025339	-1.459045	-0.959060	-1.611216	-1.026568	0.943105
2	-0.364594	-0.082820	-0.065873	0.304720	-0.691644	0.077194
3	1.989870	0.762230	1.070911	-2.072460	2.009357	-1.221672
4	-1.799729	-1.008352	-0.715463	-1.966019	-1.599180	4.785582
...
1550	-0.259898	-0.259879	-0.147072	-0.014603	-0.259484	0.456030
1551	0.069056	-0.187446	-0.715463	-0.227485	0.269913	0.023075
1552	0.149536	0.150574	-0.593665	-0.156524	-0.745664	0.726627
1553	0.637052	-0.718620	0.543118	0.765964	-0.216268	-0.301642
1554	-0.269488	-0.613995	0.543118	0.446641	0.259109	-0.788716

1555 rows × 6 columns

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.

```
In [8]: modelSVM = SVC(C = 7, class_weight = "balanced")
modelKNN = KNeighborsClassifier(n_neighbors=1)
```

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.

```
In [78]: # Make sure you comment your code clearly and you may refer to these comments  
in the part 2.4  
def train_model(originaldf, scaled, model):  
    train = scaled.iloc[:1200]  
    valid = scaled.iloc[1200:]  
    trainy = originaldf['Result'].iloc[:1200]  
    validy = originaldf['Result'].iloc[1200:]  
  
    model.fit(train, trainy)  
    test = model.predict(valid)  
    return weighted_accuracy(test, validy),model  
  
SVMaccuracy, modelSVM = train_model(df, dfProcessed, modelSVM)  
KNNaccuracy, modelKNN = train_model(df, dfProcessed, modelKNN)  
print('SVM Accuracy: ',SVMaccuracy)  
print('KNN Accuracy: ',KNNaccuracy)
```

```
SVM Accuracy:  0.8455579157815819  
KNN Accuracy:  0.7333499937523429
```

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?

In the `import_train_data` function, we imported the entire csv and created a Pandas DataFrame from it. Then, we added a new column to this dataframe, "Result", which, given the voting statistics for DEM and GOP, was a 1 if DEMs had a higher vote in the county, and 0 otherwise. We also converted the Median Income column values from strings to integers, by replacing ',' in the string with '' and casting to an `int32`. After cleaning the data, we wrote the `scale_data` function which utilizes the `StandardScaler` from the preprocessing module (from `sklearn`) on the given numerical-valued columns to get means and standard deviations for each and normalized each column using the data points.

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

The first learning method from class that we chose was K-Nearest Neighbors. We inferred that the features that we were given would be informative enough that closely related values would indicate similar voting patterns. The second method we chose was SVM because we inferred that we could make a linear formulation of our data as a kernel which would also gain a relatively high accuracy fairly easily.

2.4.3 How did you do the model selection?

To start, in the train method, we split the data into a training set and a validation set in a roughly 80/20 split. For the KNN method, we performed model selection by iterating over values from 1-10 for number of neighbors and outputting the `weighted_accuracy` of the predictions on the validation set for each. We found that only taking the single nearest neighbor gave us the highest validation accuracy at around 73%. For the SVM method, we performed model selection by iterating over different C values from 2-15 and outputting the `weighted_accuracy` of the predictions on the validation set for each. We found that having a C value of 7 gave us the highest validation accuracy at around 84%. We tried various kernels and found that the default `rbf` kernel performed the best. We also used the 'balanced' `class_weight` parameter which enables the SVM to emphasize higher frequencied values to be weighted more.

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

Yes, both our SVM and KNN reach the 68% baseline

Submission and Description	Public Score	Use for Final Score
knnv1.csv 2 days ago by Cole Basic solution with KNN and adjusted number of neighbors.	0.71124	<input type="checkbox"/>
predsv2.csv 3 days ago by Cole SVM with updated C value and balanced weights.	0.77046	<input type="checkbox"/>

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.

ADDING 2012 to 2016 CHANGE


```

In [64]: df_2016 = import_train_data("train_2016.csv")
df_2012 = import_train_data("train_2012.csv")

creative_df = df_2016.copy()
for item in ["MedianIncome", "MigraRate", "BirthRate", "DeathRate", "BachelorRate", "UnemploymentRate"]:
    creative_df["Change_"+item] = df_2012[item] + df_2016[item]

creative_df2 = df_2016.copy()
for item in ["MedianIncome", "MigraRate", "BirthRate", "DeathRate", "BachelorRate", "UnemploymentRate"]:
    creative_df2["Change_"+item] = df_2012[item] - df_2016[item]

creative_cols = ['MedianIncome', 'MigraRate', 'BirthRate', 'DeathRate', 'BachelorRate', 'UnemploymentRate',
                 'Change_MedianIncome',
                 'Change_MigraRate',
                 'Change_BirthRate',
                 'Change_DeathRate',
                 'Change_BachelorRate',
                 'Change_UnemploymentRate']

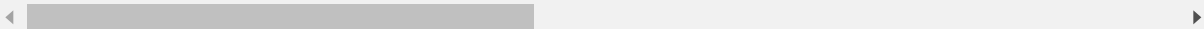
dfProcessedCreative = scale_data(creative_df, creative_cols)
dfProcessedCreative2 = scale_data(creative_df2, creative_cols)
dfProcessedCreative

```

Out[64]:

	MedianIncome	MigraRate	BirthRate	DeathRate	BachelorRate	UnemploymentRate	Chang
0	0.188697	0.416162	0.502519	0.056358	-0.065012	-0.518119	
1	0.025339	-1.459045	-0.959060	-1.611216	-1.026568	0.943105	
2	-0.364594	-0.082820	-0.065873	0.304720	-0.691644	0.077194	
3	1.989870	0.762230	1.070911	-2.072460	2.009357	-1.221672	
4	-1.799729	-1.008352	-0.715463	-1.966019	-1.599180	4.785582	
...	
1550	-0.259898	-0.259879	-0.147072	-0.014603	-0.259484	0.456030	
1551	0.069056	-0.187446	-0.715463	-0.227485	0.269913	0.023075	
1552	0.149536	0.150574	-0.593665	-0.156524	-0.745664	0.726627	
1553	0.637052	-0.718620	0.543118	0.765964	-0.216268	-0.301642	
1554	-0.269488	-0.613995	0.543118	0.446641	0.259109	-0.788716	

1555 rows × 12 columns



```
In [73]: # Initialize
modelSVMcreative = SVC(C = 2.6, class_weight = "balanced")
modelSVMcreative2 = SVC(C = 2.6, class_weight = "balanced")

creativeAccuracy, creativeModel = train_model(creative_df, dfProcessedCreative
, modelSVMcreative)
creativeAccuracy2, creativeModel2 = train_model(creative_df2, dfProcessedCreat
ive2, modelSVMcreative2)
print('Creative SVM Accuracy with Subtracting:',creativeAccuracy2)
print('Creative SVM Accuracy with Addition:',creativeAccuracy)

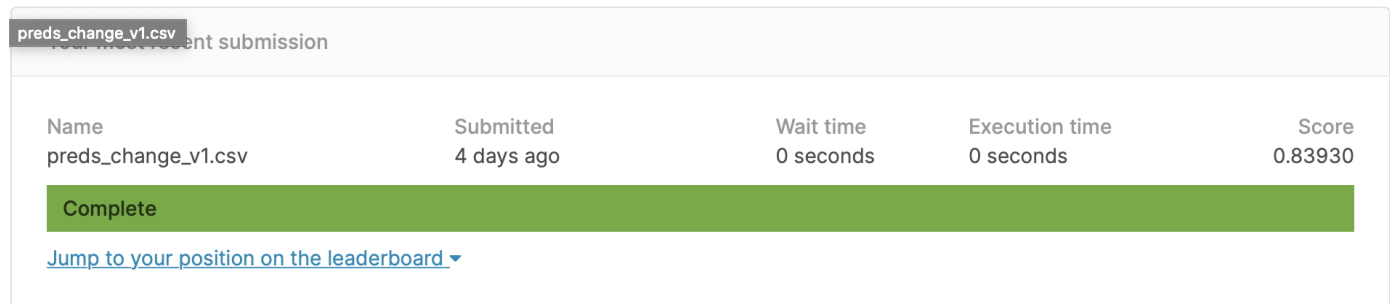
1200
1200
Creative SVM Accuracy with Subtracting: 0.8593027614644508
Creative SVM Accuracy with Addition: 0.9092840184930653
```

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)

Our creative solution is at 83.9% accuracy, which is 6.9% higher than our basic SVM solution and 12.8% higher than our basic KNN solution. We clearly reached the 75% benchmark for the Kaggle test.



Name	Submitted	Wait time	Execution time	Score
preds_change_v1.csv	4 days ago	0 seconds	0 seconds	0.83930

Complete

[Jump to your position on the leaderboard ▾](#)

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

We first started by combining the datasets from 2016 and 2012 to make new features that encapsulated change between election years in important statistics. These statistics were "MedianIncome", "MigraRate", "BirthRate", "DeathRate", "BachelorRate", and "UnemploymentRate". The main idea behind this was to factor in trends in socioeconomic metrics. We tried combining these datasets in a few ways. First, we tried subtracting the 2012 values from the 2016 values and then normalized all numerical columns. We also kept the original 2016 values in this new dataset. Later, we found out that adding the 2012 values and the 2016 values led to a much higher weighted accuracy than the subtraction method. This led to an improvement on our original SVM by a considerable amount. We also performed a model selection similar to that of the basic solution in that we split our data in the same fashion and iterated over multiple C values to select the optimal model. For our train/validation split, we originally chose 1200 as a divider because it is somewhat near the 80% conventional split, as our dataset has 1555 examples. Later, we tried adjusting the sizes of our sets, but realized that none resulted in higher testing accuracy so we reverted back to the original split of 1200.

Some other ideas we had that we did not have time to implement were:

Pooling neighboring county data using graph.csv to take into account the idea that geographically close counties are similar in voting.

Creating a convolutional neural network

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 predictions for corresponding columns. A sample prediction file can be downloaded from Kaggle.

```
In [76]: # Kaggle Submission for basic solution
testset = pd.read_csv("test_2016_no_label.csv", sep=',', header=0, encoding='unicode_escape', usecols = ['FIPS', 'MedianIncome', 'MigraRate', 'BirthRate', 'DeathRate', 'BachelorRate', 'UnemploymentRate'])
testset["MedianIncome"] = testset["MedianIncome"].str.replace(",", "").astype("int32")
testScaled = scale_data(testset, basic_cols)

testset["Result"] = modelSVM.predict(testScaled)
out = testset[["FIPS", "Result"]]
pd.DataFrame.to_csv(out, "basic_svm.csv", index = False)

testset["Result"] = modelKNN.predict(testScaled)
out2 = testset[["FIPS", "Result"]]
pd.DataFrame.to_csv(out2, "basic_knn.csv", index = False)
```

```
In [45]: # Kaggle Submission for creative solution
testset2012 = pd.read_csv("test_2012_no_label.csv", sep=',', header=0, encoding='unicode_escape', usecols = ['FIPS', 'MedianIncome', 'MigraRate', 'BirthRate', 'DeathRate', 'BachelorRate', 'UnemploymentRate'])
testset2012["MedianIncome"] = testset2012["MedianIncome"].str.replace(",", "").astype("int32")

creative_test = testset
for item in ["MedianIncome", "MigraRate", "BirthRate", "DeathRate", "BachelorRate", "UnemploymentRate"]:
    creative_test["Change_" + item] = testset[item] + testset2012[item]

testScaled = scale_data(creative_test, creative_cols)

creative_test["Result"] = modelSVMcreative.predict(testScaled)
out3 = creative_test[["FIPS", "Result"]]
pd.DataFrame.to_csv(out3, "creative_svm.csv", index = False)
```

Part 5: Resources and Literature Used

Packages used:

scikit-learn, Pandas We used Pandas DataFrames to work with the csv files.

In scikit-learn, we used the preprocessing, SVC, and KNearestNeighbors modules. The preprocessing module was used to normalize the numerical columns. The SVC module was used to train an SVM on our data. Similarly, the KNearestNeighbors module was used to train a KNN on our data.

Literature used:

Pandas documentation and scikit-learn documentation.

In []: