Power Outages

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Predict the severity (number of customers, duration, or demand loss) of a major power outage.
  - Predict the cause of a major power outage.
  - Predict the number and/or severity of major power outages in the year 2020.
  - Predict the electricity consumption of an area.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

Summary of Findings

Introduction

I am attempting to predict the severity of a major power outage. This is a regression problem. The target variable is OUTAGE.DURATION (mins) and the evaluation metric is accuracy.

Baseline Model

For the baseline model, my features are 'U.S._STATE', 'CAUSE.CATEGORY', 'CLIMATE.CATEGORY', and 'ANOMALY.LEVEL (numeric)'. The features 'U.S._STATE', 'CAUSE.CATEGORY', and 'CLIMATE.CATEGORY' are nominal while 'ANOMALY.LEVEL (numeric)' is quantitative. This model had a training score of 0.227 and a testing score of 0.145, which was not very good.

Final Model

For the final model, I engineered two features. The first is CLIMATE.CATEGORY_new, which is the original CLIMATE.CATEGORY except some values have been replaced with NaN if the cause of the outage is not severe weather. This is useful because outside of cases of severe weather, the climate category is not that relevant and can muddle the prediction if left in.

The second feature is CAUSE.CATEGORY_new, which is the original CAUSE.CATEGORY except some cause categories have been replaced with the detailed cause from CAUSE.CATEGORY.DETAILED where applicable. This is useful because the outage duration can vary a lot within a single cause category, and having a more detailed and specific cause categories can narrow them down.

I used a linear regression model and found that the best hyperparameter fit_intercept is False using GridSearchCV. This model performed better than the baseline model, with a training score of 0.352 and a testing score of 0.186.

Fairness Evaluation

I want to explore whether my model is fairer for older and newer power outages. I can binarize my data with a year threshold at 2008. Any outage in or before 2008 is considered old, while any outage after 2008 is considered new. For my parity measure, I chose R^2, and the statistic is the absolute difference in R^2 of the two subsets.

- Null Hypothesis: My model is fair. The R^2 for the two subsets are the same
- Alternative Hypothesis: My model is unfair. The R^2 for the two subsets are different

By shuffling the year column, I tested 1000 permutations and got a p-value of 0.909. This means that the difference in R^2 of the two subsets is likely due to random chance and I fail to reject the null hypothesis

Code

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

In [6]:
```
# Cleaned dataset from Project 3

df = pd.read_excel('outage.xlsx') # Dataset from website

# Cleaning excel rows & cols
cleaned = df.iloc[4:,:]
cleaned.columns = cleaned.loc[4] + ['' if i == ' ()' else i for i in (' (' + clea
cleaned = cleaned.drop(4).drop(5).drop('variables (Units)',axis=1).rename_axis(No
cleaned = cleaned.drop('OBS',axis=1)
cleaned.index += 1

# Combining start dates and times
times = cleaned['OUTAGE.START.DATE (Day of the week, Month Day, Year)'].apply(str
col = pd.to_datetime(times,errors='coerce')
cleaned = cleaned.rename(columns={'OUTAGE.START.DATE (Day of the week, Month Day,
cleaned = cleaned.drop(['OUTAGE.START.TIME (Hour:Minute:Second (AM / PM))'],axis=
cleaned['OUTAGE.START'] = col

# Combining restoration dates and times
times = cleaned['OUTAGE.RESTORATION.DATE (Day of the week, Month Day, Year)'].app
col = pd.to_datetime(times,errors='coerce')
cleaned = cleaned.rename(columns={'OUTAGE.RESTORATION.DATE (Day of the week, Mont
cleaned = cleaned.drop(['OUTAGE.RESTORATION.TIME (Hour:Minute:Second (AM / PM))']
cleaned['OUTAGE.RESTORATION'] = col

cleaned.head()
```

Out[6]:

	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVEL (numeric)
1	2011	7	Minnesota	MN	MRO	East North Central	-0.3
2	2014	5	Minnesota	MN	MRO	East North Central	-0.1
3	2010	10	Minnesota	MN	MRO	East North Central	-1.5
4	2012	6	Minnesota	MN	MRO	East North Central	-0.1
5	2015	7	Minnesota	MN	MRO	East North Central	1.2

5 rows × 53 columns

In [630]:
```
# Features are 'U.S._STATE','CAUSE.CATEGORY','CLIMATE.CATEGORY','ANOMALY.LEVEL (r
# Predicting 'OUTAGE.DURATION (mins)'
df = cleaned[['U.S._STATE','CAUSE.CATEGORY','CLIMATE.CATEGORY','ANOMALY.LEVEL (nu
df['ANOMALY.LEVEL (numeric)'] = df['ANOMALY.LEVEL (numeric)'].fillna(df['ANOMALY.
df['OUTAGE.DURATION (mins)'] = df['OUTAGE.DURATION (mins)'].fillna(df['OUTAGE.DUF
df_cat = df[['U.S._STATE','CAUSE.CATEGORY','CLIMATE.CATEGORY','ANOMALY.LEVEL (nur
df.head()
```

Out[630]:

	U.S._STATE	CAUSE.CATEGORY	CLIMATE.CATEGORY	ANOMALY.LEVEL (numeric)	OUTAGE.DURATION (mins)
1	Minnesota	severe weather	normal	-0.3	3060.0
2	Minnesota	intentional attack	normal	-0.1	1.0
3	Minnesota	severe weather	cold	-1.5	3000.0
4	Minnesota	severe weather	normal	-0.1	2550.0
5	Minnesota	severe weather	warm	1.2	1740.0

Baseline Model

In [461]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
```

In [631]:
```
# Split into train and test
X = df_cat.values
y = df['OUTAGE.DURATION (mins)']
x_train,x_test,y_train,y_test = train_test_split(X,y,random_state=1)
```

In [632]:
```
# One hot encodes categorical features
preproc = ColumnTransformer(
    transformers = [
        ('cat', OneHotEncoder(), [0,1,2])
    ], remainder = 'passthrough')
```

In [633]:
```
# Performs linear regression after preproccessing
l = Pipeline([
    ('preprocessor', preproc),
    ('lin-reg', LinearRegression())
])
```

In [634]:
```
pl.fit(df_cat.values, df['OUTAGE.DURATION (mins)'])
```

Out[634]:
```
Pipeline
preprocessor: ColumnTransformer
cat                remainder
OneHotEncoder      passthrough
LinearRegression
```

In [635]:
```
pl.predict([['Minnesota','severe weather','cold',1]])
```

Out[635]:
```
array([2796.95810905])
```

In [549]:
```
# Training score
pl.score(x_train, y_train)
```

Out[549]:
```
0.22690779416221196
```

In [636]:
```
# Testing score
pl.score(x_test, y_test)
```

Out[636]:
```
0.14455998462406783
```

Final Model

In [637]:
```
# Engineered features
# CLIMATE.CATEGORY_new: the original CLIMATE.CATEGORY excect replaced with NaN wh
# CAUSE.CATEGORY_new: the original CAUSE.CATEGORY except replaced with the detail
df_final = df.copy()
df_final['CLIMATE.CATEGORY'] = df['CLIMATE.CATEGORY'][df['CAUSE.CATEGORY']=='seve
df_final = df_final.rename(columns={'CLIMATE.CATEGORY': 'CLIMATE.CATEGORY_new'})
a = cleaned[['CAUSE.CATEGORY']][cleaned['CAUSE.CATEGORY.DETAIL'].isna()]
b = cleaned[['CAUSE.CATEGORY.DETAIL']][~cleaned['CAUSE.CATEGORY.DETAIL'].isna()]
a = a.rename(columns={'CAUSE.CATEGORY': 'CAUSE.CATEGORY_new'})
b = b.rename(columns={'CAUSE.CATEGORY.DETAIL': 'CAUSE.CATEGORY_new'})
c = pd.concat([a,b]).sort_index()
df_final.insert(loc=1,column=c.columns[0],value=c)
df_final = df_final.drop(columns=['CAUSE.CATEGORY'])
df_final
```

Out[637]:

	U.S._STATE	CAUSE.CATEGORY_new	CLIMATE.CATEGORY_new	ANOMALY.LEVEL (numeric)	OUTAG
1	Minnesota	severe weather	normal	-0.300000	
2	Minnesota	vandalism	NaN	-0.100000	
3	Minnesota	heavy wind	cold	-1.500000	
4	Minnesota	thunderstorm	normal	-0.100000	
5	Minnesota	severe weather	warm	1.200000	
...	...	...	...	...	
1530	North Dakota	public appeal	NaN	-0.900000	
1531	North Dakota	Coal	NaN	-0.096852	
1532	South	islanding	NaN	0.500000	

In [638]:
```
df_final_cat = df_final[['U.S._STATE','CAUSE.CATEGORY_new','CLIMATE.CATEGORY_new'
```

In [668]:
```
X = df_final_cat
y = df_final['OUTAGE.DURATION (mins)']
x_train,x_test,y_train,y_test = train_test_split(X,y,random_state=1)
```

In [669]:
```
# Seeing how the new features perform with the old model
pl.fit(x_train, y_train)
```

Out[669]:
```
Pipeline
preprocessor: ColumnTransformer
cat                remainder
OneHotEncoder      passthrough
LinearRegression
```

In [670]:
```
pl.score(x_train, y_train)
```

Out[670]: 0.3523551903854485

In [671]:
```
pl.score(x_test, y_test)
```

Out[671]: 0.18579995665680005

In [672]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [673]:
```
preproc = ColumnTransformer(
    transformers = [
        ('cat', OneHotEncoder(handle_unknown='ignore'), [0,1,2])
    ], remainder = 'passthrough')

pl = Pipeline([
    ('preprocessor', preproc),
    ('lin-reg', LinearRegression())
])
```

In [674]:
```
# Using GridSearchCV to find the best hyperparameter
hyperparameters = {'lin-reg__fit_intercept':[True,False]}
grid = GridSearchCV(pl, param_grid=hyperparameters,return_train_score=True)
```

In [675]:
```
grid.fit(x_train,y_train)
```

Out[675]:
```
GridSearchCV
estimator: Pipeline
preprocessor: ColumnTransformer
cat                remainder
OneHotEncoder      passthrough
LinearRegression
```

In [676]:
```
# Best value for fit_intercept is False
grid.best_params_
```

Out[676]:
```
{'lin-reg__fit_intercept': False}
```

In [677]:
```
grid.predict(x_train)
```

Out[677]:
```
array([  81.53780573, 2660.31517787,  296.660515  , ..., -960.56301349,
        367.89903738,   92.37398867])
```

In [678]:
```
grid.score(x_train,y_train)
```

Out[678]:
```
0.35235519010388205
```

In [679]:
```
grid.score(x_test,y_test)
```

Out[679]:
```
0.1859513531554564
```

Fairness Evaluation

In [680]:
```
df_final.head()
```

Out[680]:

	U.S._STATE	CAUSE.CATEGORY_new	CLIMATE.CATEGORY_new	ANOMALY.LEVEL (numeric)	OUTAGE.DUR.
1	Minnesota	severe weather	normal	-0.3	
2	Minnesota	vandalism	NaN	-0.1	
3	Minnesota	heavy wind	cold	-1.5	
4	Minnesota	thunderstorm	normal	-0.1	
5	Minnesota	severe weather	warm	1.2	

In [681]:
```
# Add year column
with_year = df_final.copy()
with_year.insert(loc=0,column='YEAR',value=cleaned['YEAR'])
with_year.head()
```

Out[681]:

	YEAR	U.S._STATE	CAUSE.CATEGORY_new	CLIMATE.CATEGORY_new	ANOMALY.LEVEL (numeric)	OUTA
1	2011	Minnesota	severe weather	normal	-0.3	
2	2014	Minnesota	vandalism	NaN	-0.1	
3	2010	Minnesota	heavy wind	cold	-1.5	
4	2012	Minnesota	thunderstorm	normal	-0.1	
5	2015	Minnesota	severe weather	warm	1.2	

In [686]:
```
# helper function to separate the old and new
# old is when year <= 2008, new is when year > 2008
# returns the absolute difference in R^2
def get_stat(df):
    old = df[df['YEAR']<=2008]
    new = df[df['YEAR']>2008]
    old_stat = grid.score(old[['U.S._STATE','CAUSE.CATEGORY_new','CLIMATE.CATEGOR
                      old['OUTAGE.DURATION (mins)'])
    new_stat = grid.score(new[['U.S._STATE','CAUSE.CATEGORY_new','CLIMATE.CATEGOR
                      new['OUTAGE.DURATION (mins)'])
    return abs(old_stat-new_stat)
```

In [687]:
```
# Original stat
orig_stat = get_stat(with_year)
orig_stat
```

Out[687]: 0.014023094390654078

In [688]:
```
# 1000 permutations of the year column
# list of stats of each
lst = []
for i in np.arange(1000):
    temp_df = with_year.copy()
    temp_df['YEAR'] = with_year['YEAR'].sample(frac=1).reset_index(drop=True)
    lst += [get_stat(temp_df)]
lst
```

Out[688]:
```
[0.13152865958653936,
 0.02251978648542785,
 0.044869074234212936,
 0.0568243985857787,
 0.24632074226478617,
 0.05180346616645792,
 0.04744493701634667,
 0.21449469213307515,
 0.0855083047336519,
 0.028215764069699834,
 0.0852385033446077,
 0.01681012856354791,
 0.07055201106725306,
 0.046779442370747826,
 0.032390615462214334,
 0.04279757238169013,
 0.14081922756974585,
 0.03536231888239971,
 0.03269060573890359,
```

In [690]:
```
# p-value
np.mean(lst > orig_stat)
```

Out[690]:
```
0.909
```

In [ ]: