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

We are trying to solve a classification problem of predicting the cause of the power outage - CAUSE.CATEGORY target variable. The evaluation metric we are using is the classification accuracy which is the ratio of the number of correct predictions to the total number of input samples.

Baseline Model

In our baseline model, we are only looking at categorical values since we believed that it is the best indicator for classifying the CAUSE.CATEGORY. The 6 features we used are: 'U.S._STATE', 'POSTAL.CODE', 'NERC.REGION', 'CLIMATE.REGION', 'CLIMATE.CATEGORY', 'HURRICANE.NAMES'

We use a SimpleImputer to fill missing values in the categorical columns with 'missing'. Then we use OneHotEncoder to assign float values to the categorical columns.

The model we used is LogisticRegression which is the most basic model for predicting categorical variables. Our accuracy came out to be 58% accuracy which is not good enough. This means that our model got a little more than half predictions correct on our test dataset.

Final Model

For our final models, we first took care of columns we knew we didn't care about:

HURRICANE.NAMES - this obviously did not have any correlation with cateogry

DEMAND.LOSS.MW - there were too many missing values so it was pointless to impute

CAUSE.CATEGORY.DETAIL - this column seemed nominal and had too much diversity to be useful.

For CLIMATE.REGION column, we filled missing values with some research online. We found out that the missing values - Hawaii and Alaska - were all part of the 'West' region.

'OUTAGE.START.DATE','OUTAGE.START.TIME','OUTAGE.RESTORATION.DATE','OUTAGE.RESTORATION.TIME' columns were pointless since we had OUTAGE.DURATION

We then used a correlation heatmap to see which columns had too close of a correlation. We wanted to reduce dimension by aggregating closely related columns into a single columns. So we added PRICE, SALES, and CUSTOMERS feature that aggregated all the columns closely related in regards to price, sales, and customers. Reducing dimension is good for our data since it improves the interpretation of the parameters of the machine learning model.

We collected all the columns that we want to focus on in the variable named 'columns'

For preprocessing, we first imputed all continuous variables with the mean, and performed a StandardScalar on them to normalize values including negatives. We also imputed all categorical variables with the category 'missing' and performed HotEncoder to assign numerical values to categories.

The best performing model that we found for our data was the decision tree classifier that was used alongside the GridSearchCV. GridSearchCV found the best parameters for the decision tree among [2,4,8] for max_depth, min_samples_split, and min_samples_leaf.

The model performance came out to be 74% accuracy - a 16% increase from our baseline model.

Fairness Evaluation

We chose 'OUTAGE.DURATION' as our interesting subset. We wanted to explore whether our model is fairer for lower outage durations or higher outage durations. The threshold we used was 800.

Null Hypothesis: The model performs similar on datasets with OUTAGE.DURATION above the threshold and below the threshold.

Alternative Hypothesis: The model performs differently on datasets with OUTAGE.DURATION above the threshold and below the threshold.

We performed a permutation test 100 times that sampled the OUTAGE.DURATION column and applied our final model to the subset > 800 and subset < 800. We then calculated the absolute difference to compare against our observed absolute difference. Our observed was 24% difference in accuracy in dataset with outage durations > 800 and dataset with durations < 800. We got a p-value of 0.04 which is less than our significance level of 0.05. Therefore, we can accept the null hypothesis.

In [1606]:
```
import warnings
warnings.filterwarnings('ignore')
```

Code

In [1069]:
```
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 figu
res
```

In [1620]:
```
#import sklearn packages
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
import sklearn.preprocessing as pp
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import BayesianRidge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, r2_score,
mean_squared_error
```

In [1071]:
```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Baseline Model

In [1584]:
```
#read the dataset
df = pd.read_excel('outage.xlsx')
df = df.drop([0,1,2,3,5])
df = df.drop(['Major power outage events in the continental U.S.'],axis=
1)
df.columns = df.iloc[0]
df = df.drop([4])
df = df.drop(['OBS'], axis =1)
df = df.reset_index(drop=True)
```

In [1585]:
```
df.head()
```

Out[1585]:

4	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVE
0	2011	7	Minnesota	MN	MRO	East North Central	-0.
1	2014	5	Minnesota	MN	MRO	East North Central	-0.
2	2010	10	Minnesota	MN	MRO	East North Central	-1.
3	2012	6	Minnesota	MN	MRO	East North Central	-0.
4	2015	7	Minnesota	MN	MRO	East North Central	1.

Do some basic data cleaning

In [1622]:
```
#fill all nulls with 0 for our base model
no_nulls = df.fillna(0)
```

In [1623]:
```
#choose the columns we want to observe
categorical_features = ['U.S._STATE',
                        'POSTAL.CODE',
                        'NERC.REGION',
                        'CLIMATE.REGION',
                        'CLIMATE.CATEGORY',
                        'HURRICANE.NAMES']

#create our Pipeline that imputes and encodes our categorical columns
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(missing_values=0,strategy='constant', fill
_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])
```

In [1624]:
```
#create ColumnTransformer that applies Pipeline to categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', categorical_transformer, categorical_features)
    ])

#create classifier pipeline that performs preprocessing and logistic reg
ression
classifier = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])
```

In [1215]:
```
#create training and testing dataset
X,y = no_nulls.drop(['CAUSE.CATEGORY'], axis=1), no_nulls[['CAUSE.CATEGO
RY']]
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
```

In [1216]:
```
#fit and score our training, testing
classifier.fit(X_tr, y_tr.values.ravel())
classifier.score(X_ts, y_ts.values.ravel())
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-pa
ckages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[1216]: 0.5859375

Final Model

In [1586]:
```
#create a copy of our data
clean = df
clean.head()
```

Out[1586]:

4	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVE
0	2011	7	Minnesota	MN	MRO	East North Central	-0.
1	2014	5	Minnesota	MN	MRO	East North Central	-0.
2	2010	10	Minnesota	MN	MRO	East North Central	-1.
3	2012	6	Minnesota	MN	MRO	East North Central	-0.
4	2015	7	Minnesota	MN	MRO	East North Central	1.

Drop HURRICANE.NAMES and DEMAND.LOSS.MW AND CAUSE.CATEGORY.DETAIL

In [1587]:
```
#dropping columns
clean = clean.drop(columns=['HURRICANE.NAMES', 'DEMAND.LOSS.MW','CAUSE.C
ATEGORY.DETAIL'])
clean.head()
```

Out[1587]:

4	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LEVE
0	2011	7	Minnesota	MN	MRO	East North Central	-0.
1	2014	5	Minnesota	MN	MRO	East North Central	-0.
2	2010	10	Minnesota	MN	MRO	East North Central	-1.
3	2012	6	Minnesota	MN	MRO	East North Central	-0.
4	2015	7	Minnesota	MN	MRO	East North Central	1.

In [1589]:
```
#observe that Hawaii and Alaska are null
clean[clean['CLIMATE.REGION'].isna()]
```

Out[1589]:

4	YEAR	MONTH	U.S._STATE	POSTAL.CODE	NERC.REGION	CLIMATE.REGION	ANOMALY.LI
1515	2008	12	Hawaii	HI	HI	NaN	
1516	2011	5	Hawaii	HI	PR	NaN	
1517	2006	10	Hawaii	HI	HECO	NaN	
1518	2006	6	Hawaii	HI	HECO	NaN	
1519	2006	10	Hawaii	HI	HECO	NaN	
1533	2000	NaN	Alaska	AK	ASCC	NaN	

Alaska and Hawaii are known to be in the West Region

In [1590]:
```
#fill missing with West
clean['CLIMATE.REGION'] = clean['CLIMATE.REGION'].fillna('West')
```

Drop outage start date/time and outage restoration date/time since we already have the duration column

In [1591]:
```
#drop columns
clean = clean.drop(columns=['OUTAGE.START.DATE','OUTAGE.START.TIME',
                            'OUTAGE.RESTORATION.DATE','OUTAGE.RESTORATIO
N.TIME'])
```

Create array of numerical columns and categorical columns. Convert numerical columns into floats Convert categorical columns into strings

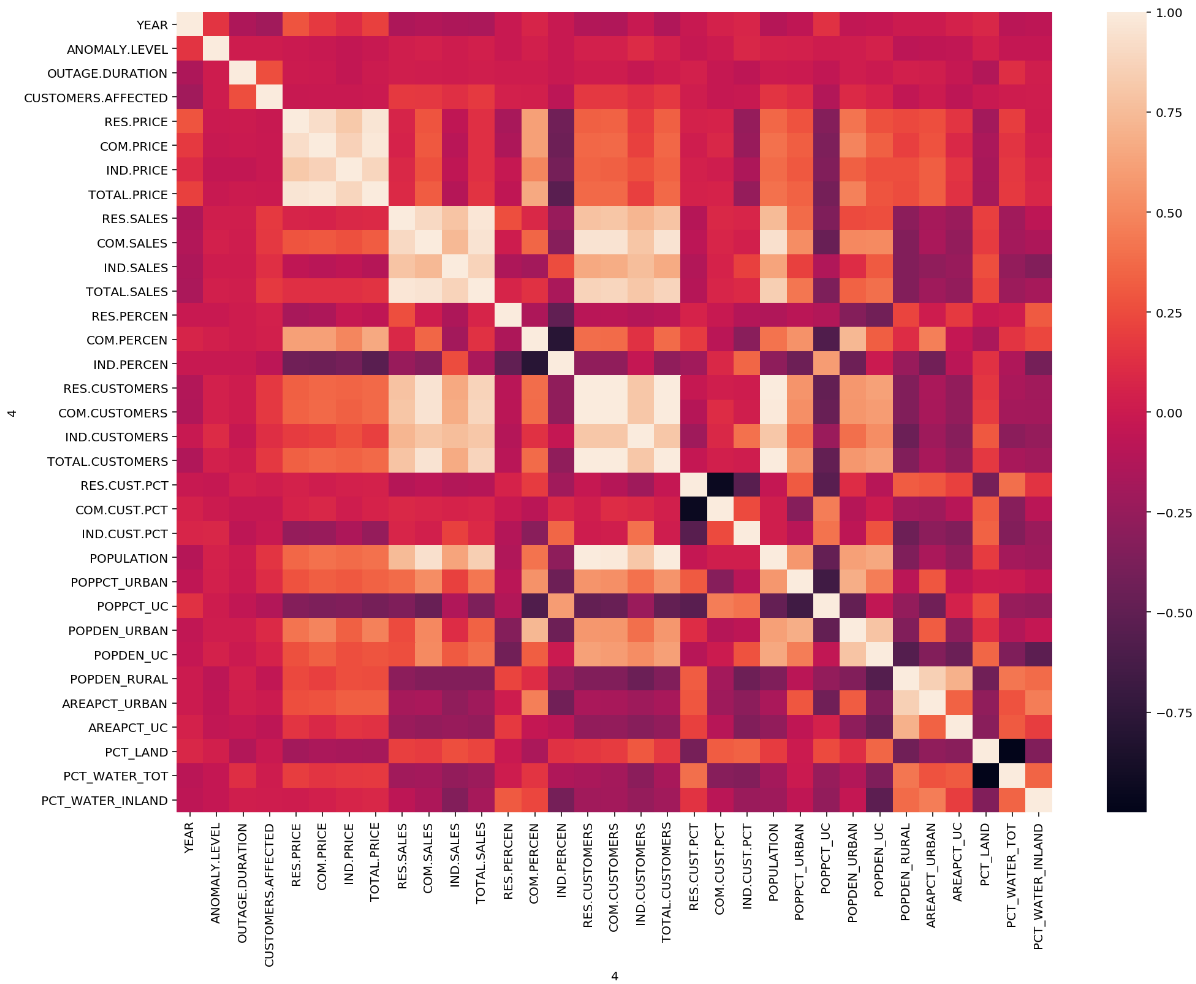
In [1592]:
```
categorical = ['MONTH','U.S._STATE','POSTAL.CODE','NERC.REGION','CLIMAT
E.REGION','CLIMATE.CATEGORY']
numerical = []
for i in clean.columns[0:11]:
    #convert numerical into floats
    if i not in categorical and i != 'CAUSE.CATEGORY':
        numerical.append(i)
        clean[i] = clean[i].apply(float)
for i in categorical:
    #convert categorical into strings
    clean[i] = clean[i].apply(str)
```

Map a correlation heatmap and combine columns that have very high correlations (reduce dimensions).

Choose the columns that we want to focus on

In [1479]:
```
f, ax = plt.subplots(figsize=(16,12))
corr = clean.corr()
#plot heatmap that shows correlations
sns.heatmap(corr)
```

Out[1479]: <matplotlib.axes._subplots.AxesSubplot at 0x13bc79250>



Create the PRICE column which aggregates RES.PRICE, COM.PRICE. IND.PRICE, and TOTAL.PRICE which all have very high correlations.

In [1593]:
```
clean['PRICE'] = (clean['RES.PRICE'] + clean['COM.PRICE'] + clean['IND.PRICE'] + clean['TOTAL.PRICE'])/4
```

Create the SALES column which aggregates RES.SALES, COM.SALES, IND.SALES, and TOTAL.SALES which all have very high correlations

In [1594]:
```
clean['SALES'] = (clean['RES.SALES'] + clean['COM.SALES'] + clean['IND.SALES'] + clean['TOTAL.SALES'])/4
```

Create the CUSTOMERS column which aggregates RES.CUSTOMERS, COM.CUSTOMERS, IND.CUSTOMERS, and TOTAL.CUSTOMERS which all have very high correlations

In [1595]:
```
clean['CUSTOMERS'] = (clean['RES.CUSTOMERS'] + clean['COM.CUSTOMERS'] +
clean['IND.CUSTOMERS'] + clean['TOTAL.CUSTOMERS'])/4
```

In [1596]:
```
columns = list(clean.columns[0:11])
columns = columns + ['PRICE','SALES','CUSTOMERS','POPULATION']
```

In [1597]:
```
#our new cleaned dataset we will use for our pipeline
few = clean[columns]
```

In [1505]:
```
#create training and testing datasets
X_tr, X_ts, y_tr, y_ts = train_test_split(few.drop(columns=['CAUSE.CATEG
ORY']), few['CAUSE.CATEGORY'], test_size=0.7)
```

In [1506]:
```
#create pipeline to transform our numerical columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

#create pipeline to transform our categorical columns
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing'
)),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

#create columntransformer to transform our columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical),
        ('cat', categorical_transformer, categorical)])

#create final pipeline with preprocessing and classifier
pipe = Pipeline(steps=[('preprocessor', preprocessor),
                  ('classifier', clf)])
```

In [1507]:
```
#create GridSearchCV with DecisionTree
parameters = {
    'max_depth': [2,4,8],
    'min_samples_split':[2,4,8],
    'min_samples_leaf':[2,4,8]
}
clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv = 5)
```

In [1508]:
```
#fit and score our testing, training
pipe.fit(X_tr,y_tr)
pipe.score(X_ts, y_ts)
```

Out[1508]: 0.7402234636871509

Fairness Evaluation

In [1536]:

```
#function that takes in a df and applies final model pipeline, returning the accuracy score
def predict(df):
    X_tr, X_ts, y_tr, y_ts = train_test_split(df.drop(columns=['CAUSE.CATEGORY']), df['CAUSE.CATEGORY'], test_size=0.7)
    numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numerical),
            ('cat', categorical_transformer, categorical)])

    parameters = {
    'max_depth': [2,4,8],
    'min_samples_split':[2,4,8],
    'min_samples_leaf':[2,4,8]
    }
    clf = GridSearchCV(DecisionTreeClassifier(), parameters, cv = 5)

    pipe = Pipeline(steps=[('preprocessor', preprocessor),
                    ('classifier', clf)])
    pipe.fit(X_tr,y_tr)
    return pipe.score(X_ts, y_ts)
```

In [1540]:

```
#above threshold
X = few[few['OUTAGE.DURATION'] > 800]
```

In [1538]:

```
#below threshold
Y = few[few['OUTAGE.DURATION'] <= 800]
```

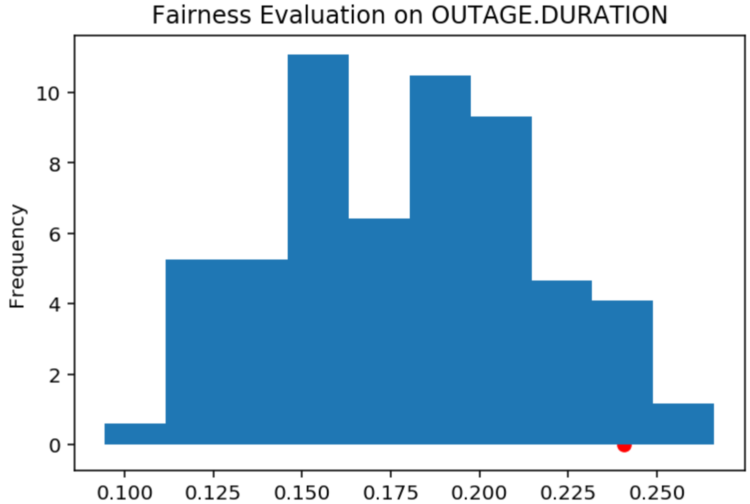
In [1611]:

```
#absolute difference for our observed stat
obs = abs(predict(X) - predict(Y))
```

In [1615]:

```
stats = []
#sample 100 times and append stat to our stats array
for i in range(100):
    copy = few.copy()
    copy['OUTAGE.DURATION'] = few['OUTAGE.DURATION'].sample(frac=1, replace=False)
    X = copy[copy['OUTAGE.DURATION'] > 800]
    Y = copy[copy['OUTAGE.DURATION'] < 800]
    stats.append(abs(predict(X) - predict(Y)))
```

In [1616]:
```
#plot our stats vs obs
pd.Series(stats).plot(kind='hist', density=True, title='Fairness Evaluat
ion on OUTAGE.DURATION')
plt.scatter(obs,0,color='red',s=40);
```



In [1618]:
```
#p-value
print((stats>=obs).mean())
```

0.04