Power Outages

Summary of Findings

Introduction

We are going to predict the cause category of power outage in the outages dataset. This is a classification question. Our target variable is "CAUSE.CATEGORY". There are seven potential cause category in the original dataset. Since we will classify each outage into 1 of 7 possible cause categories, and the cause category is not binary, we therefore use accuracy score as our evaluation metric for our two models.

Baseline Model

- We have 11 features:
  - 7 quantitative: OUTAGE.DURATION, ANOMALY.LEVEL, TOTAL.PRICE, TOTAL.CUSTOMERS, PC.REALGSP.STATE, POPPCT_URBAN
  - 4 nominal: MONTH, OUTAGE.START, U.S.STATE, CLIMATE.CATEGORY
- For our classification model, we choose accuracy score as our evaluation metric, because we have 7 possible causes instead of binary cause categories, so it is confusing to use f1score or True Negative/False Negative/True Positive/False Positive as the evaluation metric. Since accuracy score calculates the proportions of cause categories correctly identified by the model, we believe this is indicative of our model's prediction quality. In our baseline model, our average accuracy score after running 100 times is 0.568. We think this is a bad model performance(accuracy score), since this means that there is only about half of the outcomes are correctly predicted.

Final Model

- added features:
  - one engineered feature is the outage start hour, which we extract from outage start time, we choose to use hour insetad of datetime because minutes and seconds are too specific and hours are more correlated to cause category.(There is significant power usage in peak hours)
  - we also engineered ANOMALY.LEVEL to be "el nino", "la nina",and "normal", since accoording to our research, if anomaly level is greater than 0.5, then el nino is present, if the anomaly level is below -0.5, la nina is present. In-between value is indicative of normal condition.And this simplifies our data and give meaning to it.
- We choose RandomForestClassifier. We use GridSearchCv to find the parameters we are using,they are max_depth=10,max_leaf_nodes=10, min_samples_leaf=15, min_samples_split=2. The method of model selection we used is that we compared RandomForestClassifier with SVC and we found that RandomForestClassifier has a much higher accuracy score, RandomForestClassifier also generalize better than DecisionTreeClassifier. Since our major concern is accuracy score instead of how datas were classified(which SVC is good at), we choose RandomForestClassifier.

Fairness Evaluation

We want to investigate if the model we generated performs equally well across outage start hours during work hours(9am-5pm) and free hours(5pm-9am) and we intends to do an accuracy parity on our interesting subset. Since we do not care about the proportion of the population of the predictions generated, instead we care about the proportion of correctly classified predictions the model generated, we want to see if the accuracy score of our predictions differ significantly across groups, we therefore choose accuracy parity instead of demographic parity; We do not use True Positive Parity because there are more than 2 cause-category in our model, it is unable to determine what is positive in this case. And we found that our model achieves accuracy parity, there is no

Code

In [1]:
```
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 [2]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

Load dataset

Load the dataset from project 3 and simple cleaning for the first few rows where containing irrelevant informations.

In [18]:
```
#read the data frame
df = pd.read_excel("data/outage.xlsx")

#extract column names and reassign
cols = df.loc[4].to_dict()
df = df.loc[6:,:]
df = df.rename(columns =cols).reset_index(drop = True).drop(columns = ["variab
les"]).set_index("OBS")
```

Baseline Model

Choosing a few columns that we think might be relevant to the cause of an outage.

```
- U.S._STATE geographic influence
- CLIMATE.CATEGRY/ANOMALY.LEVEL climate influence
- OUTAGE.START.TIME time influence
- OUTAGE.DURATION extent of difficulty of fixing the outage might indicate the caus
e of an autage
- TOTAL.PRICE/TOTAL.SALES/TOTAL.CUSTOMERS electricity consumption influence
- PC.REALGSP.STATE regional economic influence
- POPPCT_URBAN urban population influence
```

In [4]:
```
#Filter columns to featured data frame
baseline_df = df[['U.S._STATE','CLIMATE.CATEGORY','CAUSE.CATEGORY', 'OUTAGE.ST
ART.TIME','OUTAGE.DURATION', 'ANOMALY.LEVEL', 'TOTAL.PRICE','TOTAL.SALES','TOT
AL.CUSTOMERS','PC.REALGSP.STATE', 'POPPCT_URBAN']]
```

Find Missingness

In [5]:
```
#Have a look at the missingness of baseline dataframe
baseline_df.isnull().sum()
```

Out[5]:
```
U.S._STATE           0
CLIMATE.CATEGORY     9
CAUSE.CATEGORY       0
OUTAGE.START.TIME    9
OUTAGE.DURATION     58
ANOMALY.LEVEL        9
TOTAL.PRICE         22
TOTAL.SALES         22
TOTAL.CUSTOMERS      0
PC.REALGSP.STATE     0
POPPCT_URBAN         0
dtype: int64
```

Dealing with missingness

From the table above, we decided to drop all nans.
Firstly, they don't constitute a large number of data. So they won't have a big influence on our model.
Secondly, we can't find relationship between the columns with missing values with other columns. Hence we can't do conditional imputation. And we think random sampling imputation or mode imputation is a bad idea because random imputation makes no sense to impute random date values or electricity consumption values to a certain area or time and mode imputation would create bias towards the largest group. Hence, we decide to drop all nan values.

In [7]:
```
# drop rows with nan values in the dataframe
baseline_df = baseline_df.dropna()

# convert columns with numerical values to the float data type
for x in baseline_df.columns[4:]:
    baseline_df[x] = baseline_df[x].astype(float)
```

```
C:\Users\owenz\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Build our baseline model

In [8]:
```
#create train and test set
X = baseline_df.drop('CAUSE.CATEGORY', axis = 1)
y = baseline_df['CAUSE.CATEGORY']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

#one hot encode for categorical columns and keeps the numerical columns
ohe = Pipeline([
    ('ohe', OneHotEncoder(sparse=False, handle_unknown = 'ignore'))
])
ohe_cols = ['U.S._STATE', 'CLIMATE.CATEGORY','OUTAGE.START.TIME']

ct = ColumnTransformer([('ohe', ohe, ohe_cols)], remainder = 'passthrough')

#form the pipeline
pl = Pipeline([('feat', ct),('tree', SVC(gamma='scale'))])
```

In [9]:
```
#look at the average accuracy score after running 100 times.
lst = []
for x in range(100):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
    pl.fit(X_train, y_train)
    lst.append(pl.score(X_test, y_test))
np.mean(lst)
```

Out[9]:
```
0.5686612021857924
```

Final Model

First Engineered Feature

The first engineered feature is the modification on the outage start time column. We extracted hour from the datetime object. We made this change because a start time with a year/month/day hour:minute:second format is too specific and our model might be confused. Hour is also the most representative feature of datetime on the cause category, so we kept this only time values.

In [19]:
```
#extract hour from outage.start.date
improved_df = df.copy()
df["OUTAGE.START.DATE"] = df["OUTAGE.START.DATE"].dt.date
outage_start = pd.to_datetime(df["OUTAGE.START.DATE"].astype(str).replace("Na
T", np.nan)+" "+df["OUTAGE.START.TIME"].astype(str)).dt.hour
improved_df["OUTAGE.START.TIME"] = outage_start.astype(float)
```

In [20]:
```
#select the same columns that we used in the baseline model.
improved_df = improved_df[['U.S._STATE','CLIMATE.CATEGORY','CAUSE.CATEGORY',
'OUTAGE.START.TIME','OUTAGE.DURATION', 'ANOMALY.LEVEL', 'TOTAL.PRICE','TOTAL.S
ALES','TOTAL.CUSTOMERS','PC.REALGSP.STATE', 'POPPCT_URBAN']]
```

In [21]:
```
#Convert the numerical columns to the float data type and drop nan values
for x in improved_df.columns[3:]:
    improved_df[x] = improved_df[x].astype(float)
improved_df = improved_df.dropna()
```

Second Engineered Feature

The second engineered feature is the modification on the ANOMALY.LEVEL column. We convert the original column with numerical datas to nominal datas. According to our research, ANOMALY.LEVEL is a indicator of el nino/la nina phenomenon with different anomaly level intervals. We generalized the orginal data and categorized it into el nina/la nina/normal.

In [376]:
```
#Create the function to convert the anomaly level to el nino/la nina/normal.
def trinarizer(outages):
    outages = pd.DataFrame(outages)

    def tri(outage):
        if outage>0.5:
            return 'el nino'
        elif outage<-0.5:
            return 'la nina'
        else:
            return 'normal'

    return outages[0].apply(tri).values.reshape(-1,1)


ft = FunctionTransformer(trinarizer)

ft.fit_transform(baseline_df[['ANOMALY.LEVEL']])
```

```
C:\Users\owenz\Anaconda3\lib\site-packages\sklearn\preprocessing\_function_tr
ansformer.py:97: FutureWarning: The default validate=True will be replaced by
validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
C:\Users\owenz\Anaconda3\lib\site-packages\sklearn\preprocessing\_function_tr
ansformer.py:97: FutureWarning: The default validate=True will be replaced by
validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
```

Out[376]:
```
array([['normal'],
       ['normal'],
       ['la nina'],
       ...,
       ['la nina'],
       ['normal'],
       ['normal']], dtype=object)
```

Create our improved model with the added two features

In [ ]:
```
#split orginal dataframe into train/test set.
X = improved_df.drop('CAUSE.CATEGORY', axis = 1)
y = improved_df['CAUSE.CATEGORY']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

#One hot encode for categorical data.
ohe = Pipeline([
    ('ohe', OneHotEncoder(sparse=False, handle_unknown = 'ignore')),
    ('pca', PCA(svd_solver='full')),
])
ohe_cols = ['U.S._STATE', 'CLIMATE.CATEGORY']

#Convert anomaly level into nominal data and one hot encode
def trinarizer(outages):
    outages = pd.DataFrame(outages)

    def tri(outage):
        if outage>0.5:
            return 'el nino'
        elif outage<-0.5:
            return 'la nina'
        else:
            return 'normal'

    return outages[0].apply(tri).values.reshape(-1,1)
ft = Pipeline([('transform',FunctionTransformer(trinarizer, validate = True)),
('ohe', OneHotEncoder(sparse = False, handle_unknown = 'ignore'))])
ft_col = ['ANOMALY.LEVEL']



ct = ColumnTransformer([('ohe', ohe, ohe_cols),('anomaly', ft, ft_col)], remai
nder = 'passthrough')

#Create the final pipeline
pl = Pipeline([('feat', ct),('tree', RandomForestClassifier(n_estimators = 100
))])
```

Decision on the model.

- Models that can be used: -SVC -DecisionTreeClassifier -RandomFoerestClassifier We finally determined to use RandomForestClassifier because our major concern is accuracy score instead of how datas were classified(what SVC is good at), we choose RandomForestClassifier over SVC. As for the DecisionTreeClassifier, RandomFoerestClassifier is consituted from multiple DecisionTreeClassifiers, and RandomForestClassifier is not likely to overfit the model, hence, RandomForestClassifier would be better than the DecisionTreeClassifier.

Decision on the parameters

We will use GridSearchCV to find the best parameters.

In [399]:
```
#Make a dictionary to be searched through.
parameters = {
    'max_depth': [2,5,10,13,15],
    'min_samples_split':[2,5,10,13,15],
    'min_samples_leaf':[2,3,5,7,10,15],
    'max_leaf_nodes' : [10,12,17,15]
}
clf = GridSearchCV(RandomForestClassifier(n_estimators =10), parameters, cv =
10)
```

In [409]:
```
#Find the best parameters
ct = ColumnTransformer([('ohe', ohe, ohe_cols),('anomaly', ft, ft_col)], remai
nder = 'passthrough')
clf.fit(ct.fit_transform(X), y)
clf.best_params_
```

```
C:\Users\owenz\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:814: DeprecationWarning: The default of the `iid` parameter will change fro
m True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[409]:
```
{'max_depth': 10,
 'max_leaf_nodes': 10,
 'min_samples_leaf': 15,
 'min_samples_split': 2}
```

Finalize our improved model with best parameters.

In [22]:
```
#Split the data into training and testing set
X = improved_df.drop('CAUSE.CATEGORY', axis = 1)
y = improved_df['CAUSE.CATEGORY']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

#One hot encode for categorical columns
ohe = Pipeline([
    ('ohe', OneHotEncoder(sparse=False, handle_unknown = 'ignore')),
    ('pca', PCA(svd_solver='full')),
])
ohe_cols = ['U.S._STATE', 'CLIMATE.CATEGORY']

#Convert anomaly level to nominal data and one hot encode
def trinarizer(outages):
    outages = pd.DataFrame(outages)

    def tri(outage):
        if outage>0.5:
            return 'el nino'
        elif outage<-0.5:
            return 'la nina'
        else:
            return 'normal'

    return outages[0].apply(tri).values.reshape(-1,1)
ft = Pipeline([('transform',FunctionTransformer(trinarizer, validate = True)),
('ohe', OneHotEncoder(sparse = False, handle_unknown = 'ignore'))])
ft_col = ['ANOMALY.LEVEL']



ct = ColumnTransformer([('ohe', ohe, ohe_cols),('anomaly', ft, ft_col)], remai
nder = 'passthrough')

#Make the pipeline with the best parameters.
pl = Pipeline([('feat', ct),('tree', RandomForestClassifier(n_estimators = 100
,
                                                           max_depth= 10,
                                                            max_leaf_nodes=10,
                                                            min_samples_leaf=1
5,
                                                            min_samples_split=
2))])
```

In [31]:
```
#Find the average score after running 100 times
for x in range(100):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
    pl.fit(X_train, y_train)
    lst.append(pl.score(X_test, y_test))
np.mean(lst)
```

Out[31]: 0.6455503512880563

We can see the score of our model increased from 0.568 to 0.645.

Fairness Evaluation

We want to investigate if the model we generated performs equally well across outage start hours during work hours(9am-5pm) and free hours(5pm-9am) and we intends to do an accuracy parity on our interesting subset.

In [333]:
```
#Append the prediction column into the dataframe.
results = X_test
results['prediction'] = pl.predict(X_test)

#Make a function to binarize hours into work time and free time and convert.
def bi(outage):
    if outage>=9 and outage <=17:
        return 'work'
    else:
        return 'free'

results['OUTAGE.START.TIME'] = results['OUTAGE.START.TIME'].apply(bi)
results['CAUSE.CATEGORY'] = y_test
```

```
C:\Users\owenz\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\owenz\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  if __name__ == '__main__':
C:\Users\owenz\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  # Remove the CWD from sys.path while we load stuff.
```

In [334]:
```
#Keeps the only columns we need for the accuracy parity test
results = results[['OUTAGE.START.TIME','prediction', 'CAUSE.CATEGORY']]
```

In [335]:
```
#Have a look at the accuracy scores for two groups
results.groupby('OUTAGE.START.TIME').apply(lambda x: accuracy_score(x['CAUSE.C
ATEGORY'], x.prediction)).rename('accuracy').to_frame()
```

Out[335]:

OUTAGE.START.TIME	accuracy
free	0.698864
work	0.684211

- Use a permutation test:
  - are the distributions of cause.category accuracy scores the same for work time/free time groups?
  - test-statistic: difference in accuracy scores
- Set a significance level of 0.05

In [336]:
```
#observed value
obs = results.groupby('OUTAGE.START.TIME').apply(lambda x: accuracy_score(x['C
AUSE.CATEGORY'], x.prediction)).rename('accuracy')
obs = obs.diff().iloc[-1]
obs
```

Out[336]: -0.014653110047846862

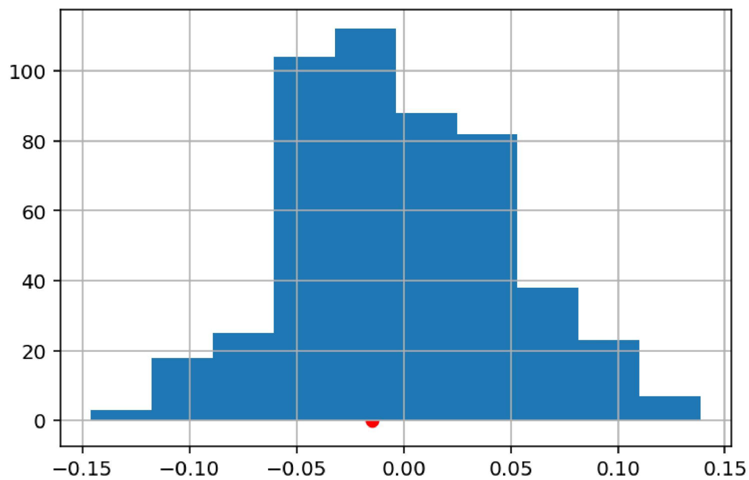
In [355]:
```
#start premutation test
lst = []

for x in range(500):
    results = results.reset_index(drop = True)
    results['OUTAGE.START.TIME'] = results['OUTAGE.START.TIME'].sample(frac =
1.0, replace = False).reset_index(drop = True)
    simulated_value = (results
                       .groupby('OUTAGE.START.TIME')
                       .apply(lambda x: accuracy_score(x['CAUSE.CATEGORY'], x.
prediction)).diff().iloc[-1])
    lst.append(simulated_value)
```

In [360]:
```
#plot the graph with p value
print((obs>=pd.Series(lst)).mean())
pd.Series(lst).hist()
plt.scatter(obs, 0, c = 'r')
```

```
0.432
```

Out[360]:
```
<matplotlib.collections.PathCollection at 0x195dc186248>
```



We found the pvalue to be 0.432 which is much higher than our significant level = 0.05. Therefore we conclude there is no significant difference in the accuracy scores for work and free time groups.