outages

December 8, 2019

1 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.

2 Summary of Findings

2.0.1 Introduction

In this notebook, the prediction question that I will be addressing relates to predicting major power outages being from severe weather. This is a classification problem, because we are trying to predict the cause of an outage, being either due to severe weather or not, which is a nominal attribute. My target variable is the column 'CAUSE.CATEGORY' from the dataset which as stated from the website explaining each of the individual column in the dataset (https://www.sciencedirect.com/science/article/pii/S2352340918307182), states the column holds the 'Categories of all the events causing the major power outages,' which hold the values for what I am trying to predict. The evaluation metric that I will use is accuracy, since our target labels (severe weather or not) are fairly balanced.

2.0.2 Baseline Model

Before evaluating what features I wanted in my dataset, I first checked the missingness of each column and discovered that 9 of the rows in the column MONTHS, were NMAR and dependent on both the missingness of CLIMATE.CATEGORY and OUTAGE.START. Since it wasn't too much data to completely ignore [9/1534(length of the dataframe) being roughly 0.6% of the data], I decided to remove these rows entirely so that I was able to distinguish between what rows were categorical and numerical, since the missingness of the data made almost every column have the datatype of Object, which would wrongly place all of the columns into the categorical section. I then created a helper function which altered each columns dtype to be either int or float where applicable or stay as object types, (this helper function does have flaws though, since it can bias the dataset depending on the type of missingness of each individual column). After this I decided to choose the features YEAR, MONTH, STATE, CLIMATE.CATEGORY, having the attributes Ordinal, Ordinal, Nominal, Nominal, to test my target ('CAUSE.CATEGORY', changed to an array with 1 representing severe weather and 0 otherwise). In the baseline model, I used One-Hot-Encoding to alter the categorical columns (those with Nominal attributes) changing the columns into numerical values per row, then applied PCA, dropping correlated features from the OHE values. I also used the transformer LinearRegression, which works for the pipeline due to the transformations on the categorical columns. I then fit the pipeline and got an accuracy of ~0.21 or 21%, which makes sense for this model because there isn't that strong of a relationship between these columns for what I am solving for and also due to problems with Multicollinearity negatively affecting LinearRegression, meaning we would potentially need to adjust our PCA or use something other than LinearRegression to try and improve the model.

2.0.3 Final Model

For the final model, two additional features that I decided to use were CAUSE.CATEGORY.DETAIL and HURRICANE.NAMES (both having Nominal attributes). These two columns seemed like good choices because CAUSE.CATEGORY.DETAIL provides the description of the cause of an outage, which for specific values like heavy wind or thunderstorm, would always match with severe weather in the CAUSE.CATEGORY, and HURRICANE.NAMES was also relevant since whenever the column was not null, the column CAUSE.CATEGORY was always severe weather. For the model type, I decided to use RandomForestClassifier since its generally the best option to get for accuracy. For finding the parameters that worked best, I used GridSearchCV and tested for the best choices for n_components for PCA, and for the max depth and number of estimators for RandomForestClassifier. With this new final model, I got an average accuracy/score of 0.8966492146596861 or roughly ~90%, which is much better than the baseline model and a good accuracy value.

2.0.4 Fairness Evaluation

For this section I decided to use a permutation test based off the YEAR column, seperating the different years into two parts, 2000s and 2010s to see whether the distribution of severe weather changes were significantly different. This question is interesting when viewing severe weather as being a potential factor due to the effects of global warming, or since technology has greatly developed past 2009 in terms of computing power and mobile phones, making outages possibly even more prevalent with more people using more electricity. For the permutation test, I tested the question of whether the distributions of outages due to severe weather were the same or not in the 2000s and 2010s, for the test statistic I used the accuracy parity, with a significance level of 0.05. From this permutation test, the p-value was much greater than 0.05, meaning that we cannot reject the null hypothesis being that differences in accuracy were not significant.

[1]:
```python
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

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

[2]:
```python
#Read the dataset
df = pd.read_excel('outage.xlsx')
```

[3]:
```python
#Set the correct columns in the Excel file, ignoring the previous statements␣
↪explaining the dataset
new_col = df.loc[4:].loc[4].values
df.columns = new_col
df = df.loc[6:].set_index('OBS').drop('variables', axis = 1)

#Cleaning the Dataset

#Combine OUTAGE.START.DATE and OUTAGE.START.TIME into a new column OUTAGE.START
#Drop the previous columns
df['OUTAGE.START.DATE'] = pd.to_datetime(df['OUTAGE.START.DATE'])
df['OUTAGE.START.TIME'] = pd.to_timedelta(df['OUTAGE.START.TIME'].astype(str))
df['OUTAGE.START'] = df['OUTAGE.START.DATE'] + df['OUTAGE.START.TIME']
df = df.drop('OUTAGE.START.DATE', axis = 1)
df = df.drop('OUTAGE.START.TIME', axis = 1)

#Combine OUTAGE.RESTORATION.DATE and OUTAGE.RESTORATION.TIME into a new column␣
↪OUTAGE.RESTORATION
#Drop the previous columns
df['OUTAGE.RESTORATION.DATE'] = pd.to_datetime(df['OUTAGE.RESTORATION.DATE'])
df['OUTAGE.RESTORATION.TIME'] = pd.to_timedelta(df['OUTAGE.RESTORATION.TIME'].
↪astype(str))
```

```
df['OUTAGE.RESTORATION'] = df['OUTAGE.RESTORATION.DATE'] + df['OUTAGE.
 ↪RESTORATION.TIME']
df = df.drop('OUTAGE.RESTORATION.DATE', axis = 1)
df = df.drop('OUTAGE.RESTORATION.TIME', axis = 1)
df.rename(columns = {'U.S._STATE': 'STATE'}, inplace = True) #Rename the␣
 ↪atrocious given state
```

[4]:

```
#Look at the missingness of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1534 entries, 1 to 1534
Data columns (total 53 columns):
YEAR                        1534 non-null object
MONTH                       1525 non-null object
STATE                       1534 non-null object
POSTAL.CODE                 1534 non-null object
NERC.REGION                 1534 non-null object
CLIMATE.REGION              1528 non-null object
ANOMALY.LEVEL               1525 non-null object
CLIMATE.CATEGORY            1525 non-null object
CAUSE.CATEGORY              1534 non-null object
CAUSE.CATEGORY.DETAIL       1063 non-null object
HURRICANE.NAMES             72 non-null object
OUTAGE.DURATION             1476 non-null object
DEMAND.LOSS.MW              829 non-null object
CUSTOMERS.AFFECTED          1091 non-null object
RES.PRICE                   1512 non-null object
COM.PRICE                   1512 non-null object
IND.PRICE                   1512 non-null object
TOTAL.PRICE                 1512 non-null object
RES.SALES                   1512 non-null object
COM.SALES                   1512 non-null object
IND.SALES                   1512 non-null object
TOTAL.SALES                 1512 non-null object
RES.PERCEN                  1512 non-null object
COM.PERCEN                  1512 non-null object
IND.PERCEN                  1512 non-null object
RES.CUSTOMERS               1534 non-null object
COM.CUSTOMERS               1534 non-null object
IND.CUSTOMERS               1534 non-null object
TOTAL.CUSTOMERS             1534 non-null object
RES.CUST.PCT                1534 non-null object
COM.CUST.PCT                1534 non-null object
IND.CUST.PCT                1534 non-null object
PC.REALGSP.STATE            1534 non-null object
PC.REALGSP.USA              1534 non-null object
```

```
PC.REALGSP.REL           1534 non-null object
PC.REALGSP.CHANGE        1534 non-null object
UTIL.REALGSP             1534 non-null object
TOTAL.REALGSP            1534 non-null object
UTIL.CONTRI              1534 non-null object
PI.UTIL.OFUSA            1534 non-null object
POPULATION               1534 non-null object
POPPCT_URBAN             1534 non-null object
POPPCT_UC                1534 non-null object
POPDEN_URBAN             1534 non-null object
POPDEN_UC                1524 non-null object
POPDEN_RURAL             1524 non-null object
AREAPCT_URBAN            1534 non-null object
AREAPCT_UC               1534 non-null object
PCT_LAND                 1534 non-null object
PCT_WATER_TOT            1534 non-null object
PCT_WATER_INLAND         1534 non-null object
OUTAGE.START             1525 non-null datetime64[ns]
OUTAGE.RESTORATION       1476 non-null datetime64[ns]
dtypes: datetime64[ns](2), object(51)
memory usage: 647.2+ KB
```

[5]:

```
#Find the null values in month
drp_idx = df[df['MONTH'] != df['MONTH']].index
print(df[df['MONTH'] != df['MONTH']].index)
print(df[df['CLIMATE.CATEGORY'] != df['CLIMATE.CATEGORY']].index)
print(df[df['OUTAGE.START'] != df['OUTAGE.START']].index)
```

```
Int64Index([240, 340, 366, 767, 888, 1319, 1507, 1531, 1534], dtype='int64',
name='OBS')
Int64Index([240, 340, 366, 767, 888, 1319, 1507, 1531, 1534], dtype='int64',
name='OBS')
Int64Index([240, 340, 366, 767, 888, 1319, 1507, 1531, 1534], dtype='int64',
name='OBS')
```

[6]:

```
#Temporary df dropping the found missing indices
t_df = df.drop(drp_idx)
```

[7]:

```
#Code to identify which columns are floats/int(Numerical) and set their dtype␣
↪as so

def col_fixer(df):
    for x in df:
        t = type(df[~df[x].isnull()][x].values[0])
        if t != float and t != int:
            continue
        df[x] = df[x].fillna(0)
```

```
        df[x].astype(t)
    return df


#Dataframe with the missing indices dropped
n_df = col_fixer(t_df)
```

Find the best features

[8]:
```
#Build the pipeline and see the score/accuracy
X = n_df[['YEAR', 'MONTH', 'STATE', 'CLIMATE.CATEGORY', 'CAUSE.CATEGORY.
 ↪DETAIL', 'HURRICANE.NAMES']]
y = (n_df['CAUSE.CATEGORY'] == 'severe weather').astype(int)

types = X.dtypes
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index

cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value= 'NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=0.99))
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
])

pl = Pipeline([('feats', ct), ('rfc', RandomForestClassifier())])

X_tr, X_ts, y_tr, y_ts = train_test_split(X, y)

pl.fit(X_tr, y_tr)
pl.score(X_ts, y_ts)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[8]: 0.9031413612565445

[9]:
```
#Find the keys
pl.get_params().keys()
```

[9]:
```
dict_keys(['memory', 'steps', 'verbose', 'feats', 'rfc', 'feats__n_jobs',
'feats__remainder', 'feats__sparse_threshold', 'feats__transformer_weights',
```

```
'feats__transformers', 'feats__verbose', 'feats__catcols',
'feats__catcols__memory', 'feats__catcols__steps', 'feats__catcols__verbose',
'feats__catcols__imp', 'feats__catcols__ohe', 'feats__catcols__pca',
'feats__catcols__imp__add_indicator', 'feats__catcols__imp__copy',
'feats__catcols__imp__fill_value', 'feats__catcols__imp__missing_values',
'feats__catcols__imp__strategy', 'feats__catcols__imp__verbose',
'feats__catcols__ohe__categorical_features', 'feats__catcols__ohe__categories',
'feats__catcols__ohe__drop', 'feats__catcols__ohe__dtype',
'feats__catcols__ohe__handle_unknown', 'feats__catcols__ohe__n_values',
'feats__catcols__ohe__sparse', 'feats__catcols__pca__copy',
'feats__catcols__pca__iterated_power', 'feats__catcols__pca__n_components',
'feats__catcols__pca__random_state', 'feats__catcols__pca__svd_solver',
'feats__catcols__pca__tol', 'feats__catcols__pca__whiten', 'rfc__bootstrap',
'rfc__class_weight', 'rfc__criterion', 'rfc__max_depth', 'rfc__max_features',
'rfc__max_leaf_nodes', 'rfc__min_impurity_decrease', 'rfc__min_impurity_split',
'rfc__min_samples_leaf', 'rfc__min_samples_split',
'rfc__min_weight_fraction_leaf', 'rfc__n_estimators', 'rfc__n_jobs',
'rfc__oob_score', 'rfc__random_state', 'rfc__verbose', 'rfc__warm_start'])
```

[10]:
```
#Set our parameters for the GridSearch
params = {
    'feats__catcols__pca__n_components':[None, 0.90, 0.99],
    'rfc__n_estimators': [1, 5, 10, 25, 100],
    'rfc__max_depth': [1, 5, None],
}
```

[11]:
```
grids = GridSearchCV(pl, param_grid=params, cv=5)
grids.fit(X_tr, y_tr)
```

[11]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('feats',
                                        ColumnTransformer(n_jobs=None,
                                                          remainder='drop',
                                                          sparse_threshold=0.3,
transformer_weights=None,
transformers=[('catcols',
Pipeline(memory=None,
  steps=[('imp',
          SimpleImputer(add_indicator=False,
                        copy=True,
                        fill_value='NULL',
                        missing_values=nan,
                        strategy='constant'…
min_weight_fraction_leaf=0.0,
                                                               n_estimators=10,
                                                               n_jobs=None,
```

```
                                                                oob_score=False,
random_state=None,
                                                                verbose=0,
warm_start=False))],
                                verbose=False),
             iid='warn', n_jobs=None,
             param_grid={'feats__catcols__pca__n_components': [None, 0.9, 0.99],
                         'rfc__max_depth': [1, 5, None],
                         'rfc__n_estimators': [1, 5, 10, 25, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[12]:
```
grids.best_score_
```

[12]:
```
0.8993875765529309
```

[13]:
```
grids.best_params_
```

[13]:
```
{'feats__catcols__pca__n_components': None,
 'rfc__max_depth': None,
 'rfc__n_estimators': 100}
```

[14]:
```
grids.cv_results_
```

[14]:
```
{'mean_fit_time': array([5.46930542, 5.24065871, 4.83871403, 5.2793715 ,
5.23009953,
        4.98724399, 5.376862  , 5.21782417, 5.33169875, 5.29284458,
        4.95562253, 5.01538992, 4.96391377, 5.53307986, 5.56037598,
        5.31108327, 5.03484793, 5.21353502, 5.3568718 , 5.04870019,
        4.90875373, 5.13854537, 5.17374606, 5.09781532, 5.60271163,
        5.17159791, 4.99415927, 5.21980481, 5.11563077, 5.87043152,
        5.33636136, 5.27868643, 5.87287111, 5.5955092 , 5.19070883,
        4.86525507, 5.33607526, 5.11750836, 4.93959031, 5.65180202,
        4.99591393, 5.43601794, 5.20169716, 5.45998573, 5.714361  ]),
 'std_fit_time': array([0.2602886 , 0.28795169, 0.53559855, 0.14736496, 0.44668
,
        0.34789237, 0.54071773, 0.44126358, 0.30629677, 0.31341436,
        0.18595049, 0.17965313, 0.49977121, 0.34601561, 0.36037139,
        0.47556089, 0.18530735, 0.46736669, 0.31564107, 0.51738101,
        0.77089156, 0.59360494, 0.33106287, 0.43381385, 0.3240596 ,
        0.31069297, 0.2918804 , 0.63216772, 0.16986008, 0.44885885,
        0.18126986, 0.41689101, 0.4050715 , 0.36221556, 0.49292549,
        0.35812211, 0.31624186, 0.29142151, 0.33177828, 0.6986289 ,
        0.17039756, 0.19764688, 0.3515477 , 0.40035675, 0.41887709]),
 'mean_score_time': array([0.0435524 , 0.03957901, 0.03721886, 0.00606837,
0.01351271,
        0.07742243, 0.0224227 , 0.00636163, 0.01066213, 0.01419592,
```

```
        0.07699428, 0.00434093, 0.01825352, 0.00674715, 0.01552129,
        0.07802048, 0.00453057, 0.00450516, 0.00592017, 0.01356354,
        0.07726488, 0.00460215, 0.00474372, 0.00613546, 0.01351566,
        0.05870838, 0.00422845, 0.00457363, 0.01132388, 0.01680398,
        0.07810335, 0.05771279, 0.00538406, 0.0080296 , 0.0139595 ,
        0.07739329, 0.02086892, 0.00481024, 0.00644975, 0.01409245,
        0.02270079, 0.00479183, 0.01847658, 0.00705237, 0.01720309]),
 'std_score_time': array([4.52749680e-02, 4.30955377e-02, 3.94049229e-02,
6.73922240e-05,
        4.44721751e-04, 3.61221268e-02, 3.51494311e-02, 1.37589666e-03,
        8.25962119e-03, 2.88377576e-04, 3.66937412e-02, 1.75604268e-04,
        2.64474603e-02, 2.51905295e-04, 7.94926115e-04, 3.69951107e-02,
        4.55171193e-04, 1.25567259e-04, 1.31042885e-04, 5.29237491e-04,
        3.65605361e-02, 4.56362268e-04, 4.50371537e-04, 8.16446314e-05,
        5.58383031e-05, 4.48352571e-02, 1.17391130e-04, 3.57298381e-05,
        9.91787131e-03, 2.14592571e-03, 3.71126486e-02, 4.31639939e-02,
        3.92630322e-04, 2.91972628e-03, 7.88238961e-04, 3.63397903e-02,
        3.33253747e-02, 2.89893330e-04, 2.47389276e-04, 4.84612046e-04,
        3.69900151e-02, 6.26303736e-04, 2.68246636e-02, 2.14577007e-04,
        2.07327076e-03]),
 'param_feats__catcols__pca__n_components': masked_array(data=[None, None, None,
None, None, None, None, None, None,
                    None, None, None, None, None, None, 0.9, 0.9, 0.9, 0.9,
                    0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9,
                    0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99,
                    0.99, 0.99, 0.99, 0.99, 0.99, 0.99],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_rfc__max_depth': masked_array(data=[1, 1, 1, 1, 1, 5, 5, 5, 5, 5, None,
None, None, None,
                    None, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, None, None, None,
                    None, None, 1, 1, 1, 1, 1, 5, 5, 5, 5, 5, None, None,
                    None, None, None],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
```

```
 'param_rfc__n_estimators': masked_array(data=[1, 5, 10, 25, 100, 1, 5, 10, 25,
100, 1, 5, 10, 25,
                    100, 1, 5, 10, 25, 100, 1, 5, 10, 25, 100, 1, 5, 10,
                    25, 100, 1, 5, 10, 25, 100, 1, 5, 10, 25, 100, 1, 5,
                    10, 25, 100],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 1,
   'rfc__n_estimators': 1},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 1,
   'rfc__n_estimators': 5},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 1,
   'rfc__n_estimators': 10},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 1,
   'rfc__n_estimators': 25},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 1,
   'rfc__n_estimators': 100},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 5,
   'rfc__n_estimators': 1},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 5,
   'rfc__n_estimators': 5},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 5,
   'rfc__n_estimators': 10},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 5,
   'rfc__n_estimators': 25},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': 5,
   'rfc__n_estimators': 100},
  {'feats__catcols__pca__n_components': None,
   'rfc__max_depth': None,
   'rfc__n_estimators': 1},
  {'feats__catcols__pca__n_components': None,
```

```
 'rfc__max_depth': None,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': None,
 'rfc__max_depth': None,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': None,
 'rfc__max_depth': None,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': None,
 'rfc__max_depth': None,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': None,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': None,
 'rfc__n_estimators': 5},
```

```
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': None,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': None,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': 0.9,
 'rfc__max_depth': None,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 1,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 10},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 25},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': 5,
 'rfc__n_estimators': 100},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': None,
 'rfc__n_estimators': 1},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': None,
 'rfc__n_estimators': 5},
{'feats__catcols__pca__n_components': 0.99,
 'rfc__max_depth': None,
```

```
  'rfc__n_estimators': 10},
 {'feats__catcols__pca__n_components': 0.99,
  'rfc__max_depth': None,
  'rfc__n_estimators': 25},
 {'feats__catcols__pca__n_components': 0.99,
  'rfc__max_depth': None,
  'rfc__n_estimators': 100}],
'split0_test_score': array([0.66375546, 0.75545852, 0.78165939, 0.76419214,
0.78165939,
       0.78165939, 0.79475983, 0.86026201, 0.86026201, 0.8558952 ,
       0.81659389, 0.86462882, 0.86899563, 0.86899563, 0.88646288,
       0.63318777, 0.74235808, 0.79039301, 0.75982533, 0.78165939,
       0.8209607 , 0.82969432, 0.80786026, 0.83406114, 0.8209607 ,
       0.80349345, 0.84279476, 0.85152838, 0.84716157, 0.84716157,
       0.60262009, 0.78165939, 0.7860262 , 0.77292576, 0.75982533,
       0.81659389, 0.84716157, 0.83406114, 0.84716157, 0.84716157,
       0.83406114, 0.86462882, 0.83842795, 0.8558952 , 0.8558952 ]),
'split1_test_score': array([0.71179039, 0.82532751, 0.81659389, 0.79039301,
0.82532751,
       0.75982533, 0.81659389, 0.89082969, 0.90393013, 0.89519651,
       0.87772926, 0.89519651, 0.88646288, 0.90829694, 0.92139738,
       0.79912664, 0.84716157, 0.81222707, 0.81222707, 0.82969432,
       0.75545852, 0.88646288, 0.86899563, 0.88209607, 0.89956332,
       0.86026201, 0.89519651, 0.89956332, 0.89956332, 0.89956332,
       0.68122271, 0.70742358, 0.8209607 , 0.82532751, 0.84279476,
       0.84716157, 0.86462882, 0.87772926, 0.89519651, 0.90829694,
       0.8558952 , 0.89082969, 0.90393013, 0.90393013, 0.91703057]),
'split2_test_score': array([0.57641921, 0.77292576, 0.81222707, 0.81222707,
0.81659389,
       0.8209607 , 0.89082969, 0.87772926, 0.88209607, 0.89082969,
       0.83406114, 0.86026201, 0.89956332, 0.89519651, 0.89082969,
       0.62445415, 0.78165939, 0.7860262 , 0.79475983, 0.79039301,
       0.78165939, 0.86462882, 0.86026201, 0.86899563, 0.86026201,
       0.85152838, 0.87336245, 0.88646288, 0.89519651, 0.89082969,
       0.56768559, 0.68995633, 0.7510917 , 0.75545852, 0.80786026,
       0.79039301, 0.8558952 , 0.88646288, 0.89519651, 0.86899563,
       0.87772926, 0.88209607, 0.89519651, 0.89519651, 0.90829694]),
'split3_test_score': array([0.57017544, 0.8377193 , 0.75      , 0.80263158,
0.8245614 ,
       0.82017544, 0.86403509, 0.85526316, 0.87719298, 0.88157895,
       0.8245614 , 0.89035088, 0.89035088, 0.9122807 , 0.90789474,
       0.63596491, 0.80701754, 0.78070175, 0.82894737, 0.82894737,
       0.85964912, 0.88157895, 0.86403509, 0.88596491, 0.89035088,
       0.89035088, 0.90350877, 0.90350877, 0.90789474, 0.89912281,
       0.79385965, 0.69298246, 0.73684211, 0.82017544, 0.83333333,
       0.83333333, 0.87280702, 0.89912281, 0.89473684, 0.89035088,
       0.88157895, 0.88157895, 0.88596491, 0.90789474, 0.92105263]),
```

```
 'split4_test_score': array([0.64035088, 0.8245614 , 0.8377193 , 0.82894737,
0.85526316,
        0.81578947, 0.88157895, 0.89473684, 0.89035088, 0.89473684,
        0.79824561, 0.88596491, 0.89912281, 0.89473684, 0.89035088,
        0.67105263, 0.82017544, 0.83333333, 0.79824561, 0.85964912,
        0.81140351, 0.85964912, 0.87280702, 0.85087719, 0.85526316,
        0.83333333, 0.87280702, 0.87719298, 0.87719298, 0.88157895,
        0.64473684, 0.71052632, 0.86403509, 0.84210526, 0.86842105,
        0.71052632, 0.86842105, 0.90350877, 0.86842105, 0.88596491,
        0.84210526, 0.85526316, 0.89473684, 0.89912281, 0.89035088]),
 'mean_test_score': array([0.63254593, 0.80314961, 0.79965004, 0.79965004,
0.82064742,
        0.79965004, 0.84951881, 0.87576553, 0.88276465, 0.88363955,
        0.83027122, 0.87926509, 0.88888889, 0.89588801, 0.89938758,
        0.6727909 , 0.79965004, 0.80052493, 0.79877515, 0.81802275,
        0.80577428, 0.86439195, 0.85476815, 0.86439195, 0.86526684,
        0.84776903, 0.87751531, 0.88363955, 0.88538933, 0.88363955,
        0.65791776, 0.71653543, 0.79177603, 0.80314961, 0.8223972 ,
        0.79965004, 0.86176728, 0.88013998, 0.88013998, 0.88013998,
        0.85826772, 0.87489064, 0.88363955, 0.89238845, 0.89851269]),
 'std_test_score': array([0.05359619, 0.03266621, 0.03057583, 0.02176562,
0.02354362,
        0.02468982, 0.03750904, 0.01581252, 0.01445563, 0.01472239,
        0.02650832, 0.01412756, 0.011163  , 0.01515607, 0.01326359,
        0.06520113, 0.03563134, 0.01957595, 0.0229147 , 0.02854138,
        0.03542928, 0.02005552, 0.02386381, 0.01950961, 0.02791858,
        0.02881166, 0.02113798, 0.01860175, 0.02160411, 0.01940158,
        0.07793371, 0.03355207, 0.04639385, 0.03313276, 0.03683067,
        0.04834979, 0.00918865, 0.02480691, 0.01945843, 0.02071677,
        0.01883913, 0.01296795, 0.02333285, 0.0187642 , 0.02379914]),
 'rank_test_score': array([45, 32, 35, 35, 29, 35, 25, 17, 11,  7, 27, 15,  5,
3,  1, 43, 35,
        34, 40, 30, 31, 20, 24, 20, 19, 26, 16,  7,  6,  7, 44, 42, 41, 32,
        28, 35, 22, 12, 12, 12, 23, 18,  7,  4,  2], dtype=int32)}
```

[15]:
```
grids.best_estimator_.score(X_ts, y_ts)
```

[15]: 0.918848167539267

2.0.5 Baseline Model

[16]:
```
#Features and Training array
X = n_df[['YEAR', 'MONTH', 'STATE', 'CLIMATE.CATEGORY']]
y = (n_df['CAUSE.CATEGORY'] == 'severe weather').astype(int)

types = X.dtypes
```

```
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index
```

[17]:
```
#Create the Pipeline

cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value= 'NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=0.99))
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
])

pl = Pipeline([('feats', ct), ('reg', LinearRegression())])
```

[18]:
```
#Get the accuracy
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y)

pl.fit(X_tr, y_tr)
pl.score(X_ts, y_ts)
```

[18]: 0.20590601453459534

2.0.6 Final Model

[19]:
```
#Create the Pipeline with the new features

X = n_df[['YEAR', 'MONTH', 'STATE', 'CLIMATE.CATEGORY', 'CAUSE.CATEGORY.
 ↪DETAIL', 'HURRICANE.NAMES']]
y = (n_df['CAUSE.CATEGORY'] == 'severe weather').astype(int)

types = X.dtypes
catcols = types.loc[types == np.object].index
numcols = types.loc[types != np.object].index

cats = Pipeline([
    ('imp', SimpleImputer(strategy='constant', fill_value= 'NULL')),
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
    ('pca', PCA(svd_solver='full', n_components=None))
])

ct = ColumnTransformer([
    ('catcols', cats, catcols),
])
```

```
pl = Pipeline([('feats', ct), ('rfc', RandomForestClassifier())])
```

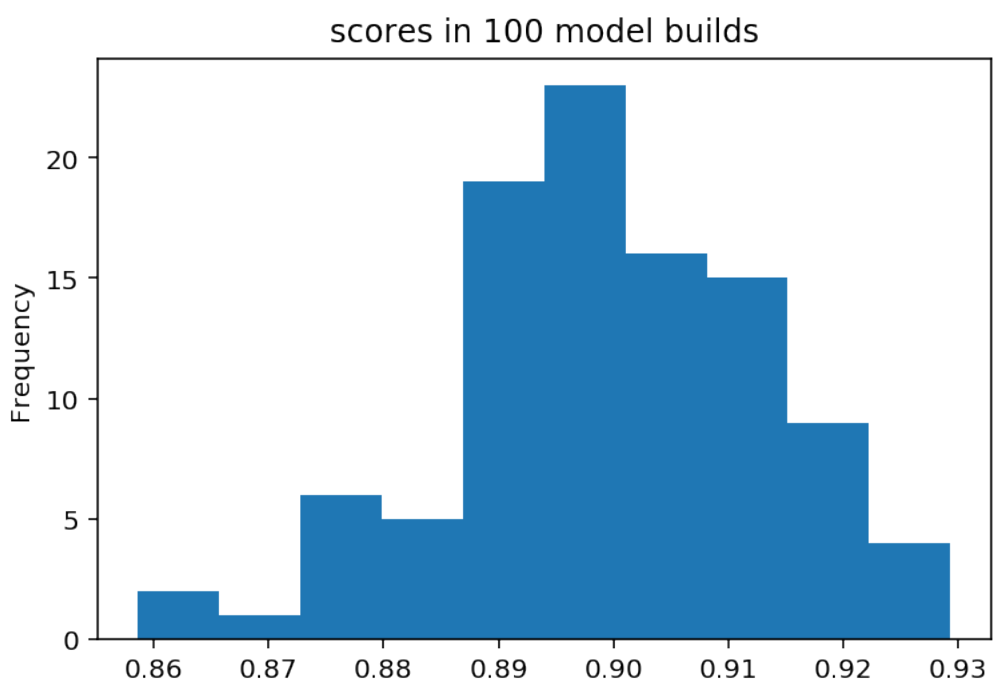
[20]:
```
#Get the accuracy
X_tr, X_ts, y_tr, y_ts = train_test_split(X, y)

pl.fit(X_tr, y_tr)
pl.score(X_ts, y_ts)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[20]: 0.8979057591623036

[21]:
```
#Run 100 times to get an average of accuracy

out = []
for _ in range(100):
    X_tr, X_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.25)
    pl.fit(X_tr, y_tr)
    out.append(pl.score(X_ts, y_ts))
```

[22]:
```
#Plot these averages
pd.Series(out).plot(kind='hist', title='scores in 100 model builds');
```



[23]:
```
np.mean(out)
```

[23]: 0.8994764397905758

2.0.7 Fairness Evaluation

[27]:
```
states = n_df['STATE'].unique()
cause_cat = n_df['CAUSE.CATEGORY.DETAIL'].unique()
hurr = n_df['HURRICANE.NAMES'].unique()
climate_cat = n_df['CLIMATE.CATEGORY'].unique()
```

[30]:
```
#Create the dataframe with the nominal columns transformed into quantitative␣
↪values
tmp = pd.concat([n_df['YEAR'], n_df['MONTH'], n_df['STATE'].apply(lambda x: pd.
↪Series(x == states, index=states, dtype=float)), n_df['CAUSE.CATEGORY.
↪DETAIL'].apply(lambda x: pd.Series(x == cause_cat, index=cause_cat,␣
↪dtype=float)), n_df['HURRICANE.NAMES'].apply(lambda x: pd.Series(x == hurr,␣
↪index=hurr, dtype=float)), n_df['CLIMATE.CATEGORY'].apply(lambda x: pd.
↪Series(x == climate_cat, index=climate_cat, dtype=float))], axis = 1)

y = (n_df['CAUSE.CATEGORY'] == 'severe weather').astype(int)
```

[31]:
```
#Find the accuracy

X_tr, X_ts, y_tr, y_ts = train_test_split(tmp, y)

clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X_tr, y_tr)
preds = clf.predict(X_ts)

metrics.accuracy_score(y_ts, preds)
```

[31]: 0.8115183246073299

2.0.8 Parity Measure (accuracy)

A = { years affected with year <= 2010 }

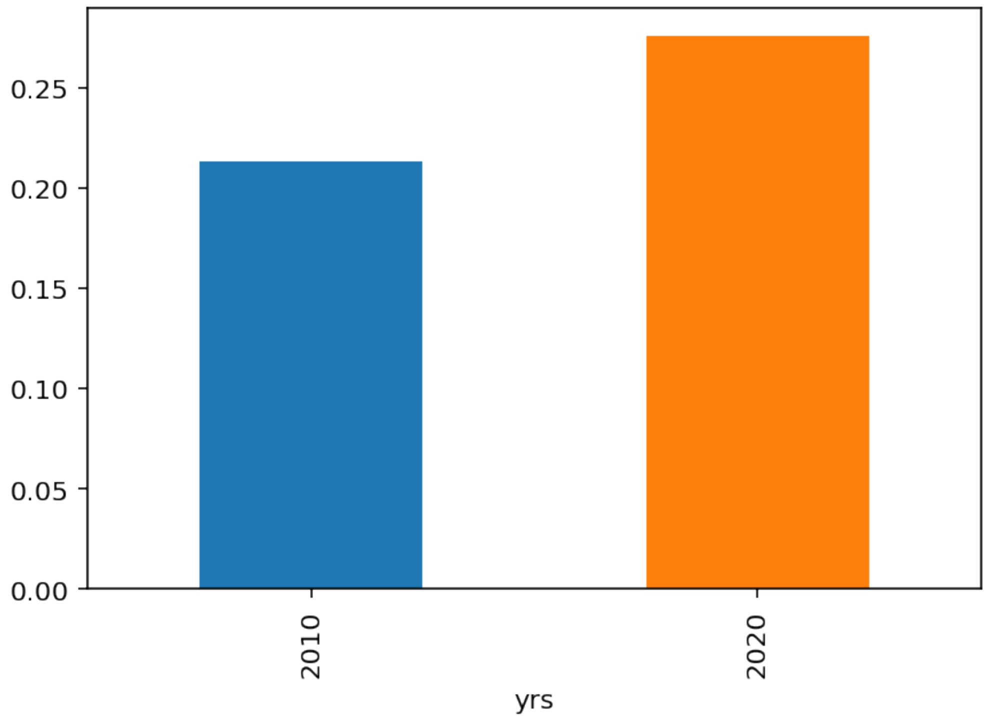
Y = Severe weather (1.0) or other causes (0.0)

C = was there severe weather (1.0) or not (0.0)

[32]:
```
#Get the accuracy

results = X_ts
results['yrs'] = results['YEAR'].apply(lambda x:10*(x//10 + 1))
results['prediction'] = preds
results['tag'] = y_ts

(
    results
    .groupby('yrs')
    .apply(lambda x:1 - metrics.recall_score(x.tag, x.prediction))
    .plot(kind='bar')
)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
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/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
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/stable/indexing.html#indexing-view-versus-copy
  """
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
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/stable/indexing.html#indexing-view-versus-copy
```

[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b5d954d68>
```



[33]:
```
results['YEAR'] = (results.YEAR <= 2009).replace({True:'2010s', False:'2000s'})
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
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/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[34]:
```
results.groupby('YEAR').prediction.mean().to_frame()
```

[34]:
```
       prediction
YEAR
2000s    0.359833
2010s    0.643357
```

[35]:
```
# Accuracy Parity

(
    results
    .groupby('YEAR')
    .apply(lambda x: metrics.accuracy_score(x.tag, x.prediction))
```

```
    .rename('accuracy')
    .to_frame()
)
```

[35]:

```
       accuracy
YEAR
2000s  0.824268
2010s  0.790210
```

2.1 Is this difference in accuracy significant?

2.1.1 Are the distributions of severe weather-scores "the same" in the 2000 as they are in the 2010 groups?

2.1.2 Test-statistic: Accuracy

Significance level - 0.05

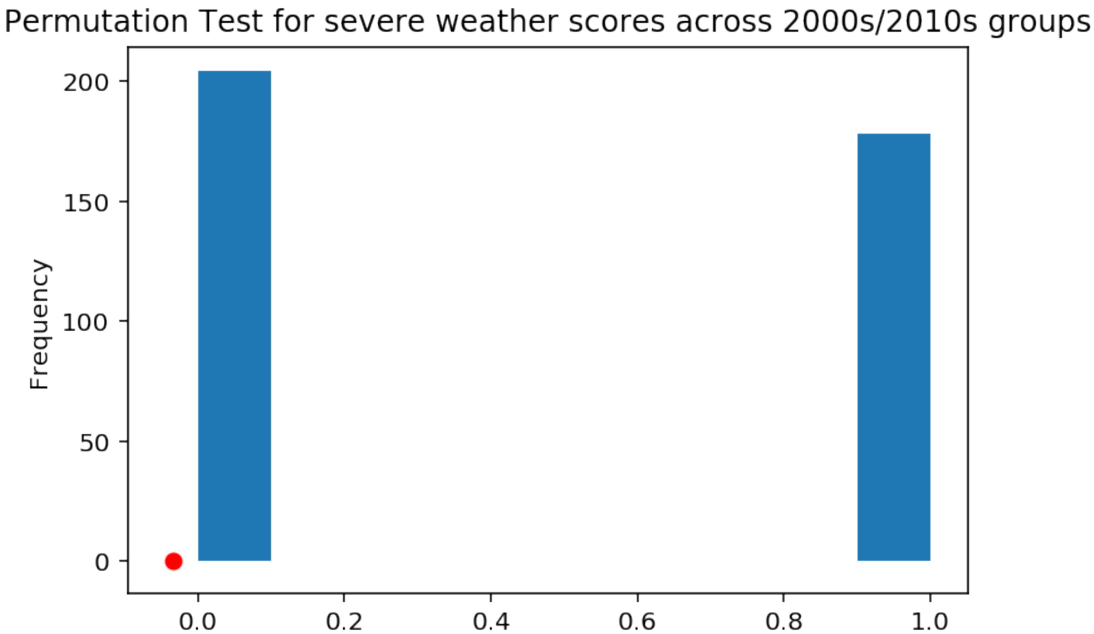
[39]:

```
obs = results.groupby('YEAR').apply(lambda x: metrics.accuracy_score(x.tag, x.
 ↪prediction)).diff().iloc[-1]

pred = []
for _ in range(100):
    s = (
        results[['YEAR', 'prediction', 'tag']]
        .assign(YEAR=results.YEAR.sample(frac=1.0, replace=False).
 ↪reset_index(drop=True))
        .groupby('YEAR')
        .apply(lambda x: metrics.accuracy_score(x.tag, x.prediction))
        .diff()
        .iloc[-1]
    )

    pred.append(s)
```

[40]:

```
#Print the p-value and plot
print(pd.Series(pred <= obs).mean())
pd.Series(preds).plot(kind='hist', title='Permutation Test for severe weather␣
 ↪scores across 2000s/2010s groups')
plt.scatter(obs, 0, c='r');
```

```
0.35
```



We fail to reject the null hypothesis, that the difference in the accuracy of outages due to severe weather in the decade 2000s and 2010s are not significant

[ ]:

```

```