PROPERTY DAMAGE PREDICTION

Individual Final Report

Author: Ishan Kuchroo

Email: ishankuchroo@gwu.edu

Overview of Project

NOAA (National Oceanic and Atmospheric Administration) records the occurrence of storms and other significant weather phenomena having sufficient intensity to cause loss of life, injuries, significant property damage, and/or disruption to commerce.

What is a Storm?

According to Wikipedia, a storm is any disturbed state of an environment or in an astronomical body's atmosphere especially affecting its surface, and strongly implying severe weather. It may be marked by significant disruptions to normal conditions such as strong wind, tornadoes, hail, thunder, and lightning (a thunderstorm), heavy precipitation (snowstorm, rainstorm), heavy freezing rain (ice storm), strong winds (tropical cyclone, windstorm), or wind transporting some substance through the atmosphere as in a dust storm, blizzard, sandstorm, etc.

What are we predicting?

NOAA stores the observations of storm events in a database of csv files (https://www.ncei.noaa.gov/pub/data/swdi/stormevents/csvfiles/). We are using the features and observations from this data to predict the property damage caused by any of the storm events in United States

Roles and Responsibility

Team Member	Area of Work	Shared Responsibility
Siddharth Das	Extraction and Preprocessing	EDA
Kartik Das	EDA	Modelling
Hemangi Kinger	PyQt5 and Visualization	EDA
Ishan Kuchroo	Modelling	Preprocessing

What is my responsibility?

Once my teammates are done with extraction and data pre-processing, I'll be using their learning of the data to do further feature engineering, train and build regression models. Additionally, based on model performance, I'll suggest what changes should be done in the previous steps to increase the model's prediction accuracy.

Modelling

After Data Cleaning and Data Transformation, I started to do following steps:

Advanced feature engineering:

1. **Encoding categorical columns**:

   For categorical columns, based on type of data available, I did label encoding and one-hot encoding.

   Columns for label encoding:
   `'CZ_NAME', 'BEGIN_LOCATION', 'END_LOCATION', 'TOR_OTHER_CZ_STATE', 'TOR_OTHER_CZ_NAME'`

   Columns for one-hot encoding:
   `'STATE', 'MONTH_NAME', 'EVENT_TYPE', 'CZ_TYPE', 'CZ_TIMEZONE', 'BEGIN_AZIMUTH', 'MAGNITUDE_TYPE', 'FLOOD_CAUSE', 'TOR_F_SCALE', 'END_AZIMUTH'`

2. **Imputation of logically important columns**:

   For column "`DAMAGE_CROPS`", we believed instead of simply removing all NAN's it is better to impute them with the average value of DAMAGE_CROPS per EVENT.

3. **Split the data into training and validation sets**:

   Using `from sklearn. model_selection import train_test_split`, I was able to create the training data and validation data sets.

4. **Standardize and normalize the data**:

   Using, `from sklearn. preprocessing import StandardScaler`, I was able to standardize the training data before running the regression models. In addition to this, using mean and standard deviation I normalized the training data.

Training Models:

1. **Linear Regression**:

   Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

```
model = LinearRegression()
model.fit(X_train, y_train)

print(model)

# make predictions for test data
y_pred = model.predict(X_test)
# y_pred_score = [round(value) for value in y_pred]

MSE = mean_squared_error(y_pred,y_test)
print("Mean Square Value", MSE)

print("Training R-square value", model.score(X_train,y_train))

print("R-Square Value", r2_score(y_test, y_pred))
```

Linear Regression Results:

2. **Random Forest:**

   A supervised learning algorithm that is based on the ensemble learning method and many Decision Trees. Random Forest uses a Bagging technique, so all calculations are run in parallel and there is no interaction between the Decision Trees when building them.

```
clf_rf = RandomForestRegressor(n_estimators= 100, oob_score = 'TRUE', n_jobs = -1,random_state =50, max_features = "auto", min_samples_leaf = 50)
# RandomForestRegressor(max_depth=10, random_state=0)

# perform training
clf_rf.fit(X_train, y_train)

# make predictions

# prediction on test using all features
y_pred = clf_rf.predict(X_test)

MSE = mean_squared_error(y_pred,y_test)
print("Mean Square Value", MSE)

print("Training R-square value", clf_rf.score(X_train,y_train))

print("R-Square Value", r2_score(y_test, y_pred))
```

Random Forest Results:

3. **XGBoost Regressor**:

Gradient boosting refers to a class of ensemble machine learning algorithms constructed from decision tree models. Models are fit using loss function and gradient descent algorithm. This gives the name, “gradient boosting,” as the loss gradient is minimized as the model is fitted. Extreme Gradient Boosting, or XGBoost, is an efficient implementation of the gradient boosting algorithm and is a powerful approach for building supervised regression models.

```
xgb = XGBRegressor(learning_rate =0.01,subsample =0.7, max_depth=5, n_estimators=100, colsample_bytree=0.8, booster = "gblinear")
xgb.fit(X_train, y_train, eval_set=[(X_train, y_train)])

# print(xgb.feature_importances_)

# make predictions for test data
y_pred = xgb.predict(X_test)
y_pred_score = [round(value) for value in y_pred]

MSE = mean_squared_error(y_pred,y_test)
print("Mean Square Value", MSE)

print("Training R-square value", xgb.score(X_train,y_train))

print("R-Square Value", r2_score(y_test, y_pred))
```

XGBoost Results:

<u>Additional options tried to increase model efficiency</u>:

Outlier Removal:

Using the Inter-Quartile Range method, I was able to identify the outliers and remove them. This method helped improve the R-square of the model by 5%.

```
# Removing Outliers

NOAA_df.shape

Quart1 = NOAA_df.quantile(0.25)
Quart3 = NOAA_df.quantile(0.75)
Range = Quart3 - Quart1

NOAA_df = NOAA_df[~((NOAA_df < (Q1 - 1.5 * Range)) |(NOAA_df > (Q3 + 1.5 * Range))).any(axis=1)]

NOAA_df.shape
```

Principal Component Analysis:

Principal Component Analysis, or PCA, is a very popular dimensionality reduction technique. PCA is trying to rearrange the features by their linear combinations. One characteristic of PCA is that the first principal component holds the most information about the dataset. The second principal component is more informative than the third, and so on.

```
# PCA code if we use one-hot encoding

pca = PCA()
# pca = PCA(n_components=25)
pca.fit(X_train)


X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# print(X_train.shape)
# pca.components_
```

Conclusion

Referenced Code %:

(210 – 164)/ 210 + 52 * 100 = 17%

References

http://en.wikipedia.org/wiki/Storm

sklearn.linear_model.LinearRegression — scikit-learn 1.0.1 documentation

https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097

https://machinelearningmastery.com/principal-component-analysis-for-visualization/

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html