US Accidents Severity Prediction – Data Mining Final Project

Topic: US Accidents Severity Prediction

Introduction: Road traffic crashes and accidents rank as the 9th leading cause of death and accounts for 2.2% of all deaths globally. They cost USD 518 billion globally, which is around 1-2% of nation's GDP. Around 20-50 million people get injured or disabled annually involved in the accidents, whereas on an average around 3287 deaths occur every day due to road accidents. While nearly 1.25 million people die in road crashes each year, in USA alone over 37000 people die in road crashes every year. Infact, road crashes are the single greatest cause of death of a healthy US citizens travelling abroad. Hence we wanted to conduct an experiment on US Accidents Dataset that consists of accident data from 2016 to 2020 to draw some key insights and perform a prediction model that could give us severity prediction on a real-time basis for the future accidents that occur.

Objectives:

1) Accident Severity Prediction
2) Key factors affecting these accidents using E.D.A and Feature Importance Analysis
3) Real-time accident severity prediction

To be specific, for a given accident, without any detailed information about itself, like driver attributes or vehicle type, this model is supposed to be able to predict the likelihood of this accident being a severe one. The accident could be the one that just happened and still lack of detailed information, or a potential one predicted by other models. Therefore, with the sophisticated real-time traffic accident prediction solution developed by the creators of the same dataset used in this project, this model might be able to further predict severe accidents in real-time.

Industry and Applications:

Insurance Industry: Accident Severity is the basis for both the premium for insurance that covers accidents and amount of money that an insurance company will release for the policyholder in the event of an accident.

Government: Helps them plan for better healthcare services, emergency planning, traffic control, but most importantly in prevention of accidents in the future.

Data Characteristics:

- Dataset contains data of US accidents during the period of February 2016 to December 2020
- Shape of the Data is (2845342,49) – 2845342 rows and 49 columns
- State-wise data is considered for model experiments (considering dataset is huge, and for better understanding in a state level)
- Feb 2019 to Dec 2020 Data Period is considered to remove irrelevant factors, whereas Overall Data is considered for Exploratory Data Analysis
- Data Attributes are divided into 5 categories
- Dataset Source - https://smoosavi.org/datasets/us_accidents

Data Cleaning and Pre-Processing:

Data cleaning was first performed to detect and handle corrupt or missing records. EDA (Exploratory Data Analysis) and feature engineering were then done over most features. Finally, Logistic regression, Random Forest Classifier, and EasyEnsemble were used to develop the predictive model. the final model is dependent on only **a small range of data attributes** that are **easily achievable** for all regions in the United States and before the accident really happened.

Key Findings

- Accident severity can be predicted with limited data attributes (location, time, weather, and POI).

- **Minute(frequency-encoding)** is the most useful feature. An accident is more likely to be a serious one when accidents happen less frequently at this time.
- Spatial patterns are also very important. For small areas like **street** and **zipcode**, severe accidents are more likely to happen at places having more accidents while for larger areas like **city** and **airport region**, at places having less accident.
- **Pressure** is top fourth important feature in the random-forest model and there is negative correlation between pressure and severity.
- An accident is much less likely to be severe if it happens near **traffic signal** while more likely if near **junction**.

Dataset Overview:

Traffic Attributes (12):

- **ID**: Accident record ID
- **Source**: API which reported the accident
- **TMC**: Traffic Message Channel (TMC) provides detailed description of the event.
- **Severity**: Severity of the accident, a number between 1 to 4
- **Start_Time**: Start time of the accident
- **End_Time**: End time of the accident
- **Start_Lat**: Latitude in GPS coordinate.
- **Start_Lng**: Longitude in GPS coordinate
- **End_Lat**: Latitude in GPS coordinate
- **End_Lng**: Longitude in GPS coordinate
- **Distance(mi)**: Length of the road extent affected by the accident.
- **Description**: Description of the accident.

Address Attributes (9):

- **Number**: Street number
- **Street**: Street name
- **Side**: Relative side of the street (Right/Left)
- **City**: City
- **County**: County
- **State**: State
- **Zipcode**: Zipcode
- **Country**: Country
- **Timezone**: Shows timezone based on the location of the accident (eastern, central, etc.).

Weather Attributes (11):

- **Airport_Code**: Airport-based weather station
- **Weather_Timestamp**: Time-stamp of weather observation record
- **Temperature(F)**: Temperature (F)
- **Wind_Chill(F)**: Wind chill (F)
- **Humidity(%)**: Humidity (%)
- **Pressure(in)**: Air pressure (in)
- **Visibility(mi)**: Visibility (mi)
- **Wind_Direction**: Wind direction
- **Wind_Speed(mph)**: Wind speed (mi/h)
- **Precipitation(in)**: Precipitation amount (in)
- **Weather_Condition**: Weather condition (rain, fog, etc)

POI Attributes (13):

- **Amenity**: Indicates presence of amenity.
- **Bump**: speed bump or hump
- **Crossing**: crossing
- **Give_Way**: give_way sign
- **Junction**: junction
- **No_Exit**: no_exit sign
- **Railway** railway
- **Roundabout**: roundabout
- **Station**: station (bus, train, etc.)
- **Stop** stop sign.
- **Traffic_Calming**: traffic_calming means
- **Traffic_Signal**: traffic_signal.
- **Turning_Loop**: turning_loop

Period-of-Day (4):

- **Sunrise_Sunset**: Based on sunrise/sunset.
- **Civil_Twilight**: civil twilight.
- **Nautical_Twilight**: nautical twilight.
- **Astronomical_Twilight**: astronomical twilight.

Useless Features

Features 'ID' doesn't provide any useful information about accidents themselves. 'TMC', 'Distance(mi)', 'End_Time' (we have start time), 'Duration', 'End_Lat', and 'End_Lng'(we have start location) can be collected only after the

accident has already happened and hence cannot be predictors for serious accident prediction. For 'Description', the POI features have already been extracted from it by dataset creators. We got rid of these features first.
df = df.drop(['ID','TMC','Description','Distance(mi)', 'End_Time', 'Duration', 'End_Lat', 'End_Lng'], axis=1)

Categorical Data:
Side, Country, Timezone, Amenity, Bump, Crossing, Give_way, Junction, No_Exit, Railway, Roundabout, Station, Stop, Traffic_Calming, Traffic_Signal, Turning_Loop, Sunrise_Sunset, Civil_Twilight, Nautical_Twilight, Astronomical Twilight

Fixing Datetime Format:
 # average difference between weather time and start time
print("Mean difference between 'Start_Time' and 'Weather_Timestamp': ",
(df.Weather_Timestamp - df.Start_Time).mean())
Mean difference between 'Start_Time' and 'Weather_Timestamp': 0 days 00:00:33.122457

Since the 'Weather_Timestamp' is almost as same as 'Start_Time', we can just keep 'Start_Time'. Then map 'Start_Time' to 'Year', 'Month', 'Weekday', 'Day' (in a year), 'Hour', and 'Minute' (in a day).

Handling Missing Data:
More than 60% percent of 'Number', 'Wind_Chill(F)', and 'Precipitation(in)' is missing. Drop na and value imputation wouldn't work for these features. 'Number' and 'Wind_Chill(F)' will be dropped because they are not highly related to severity according to previous research, whereas 'Precipitation(in)' could be a useful predictor and hence can be handled by separating feature.
Dropping Data:
NA Data: The counts of missing values in some features are much smaller compared to the total sample. It is convenient to drop rows with missing values in these columns. Drop NAs by these features:

1. 'City'
2. 'Zipcode'
3. 'Airport_Code'
4. 'Sunrise_Sunset'
5. 'Civil_Twilight'
6. 'Nautical_Twilight'
7. 'Astronomical_Twilight'

Value Imputation:
Continuous weather features with missing values:

1. Temperature(F)
2. Humidity(%)
3. Pressure(in)
4. Visibility(mi)
5. Wind_Speed(mph)

Before imputation, weather features will be grouped by location and time first, to which weather is naturally related. 'Airport_Code' is selected as location feature because the sources of weather data are airport-based weather stations. Then the data will be grouped by 'Start_Month' rather than 'Start_Hour' because using the former is computationally cheaper and remains less missing values. Finally, missing values will be replaced by median value of each group.

Code for Dropping and Imputation:

```
# group data by 'Airport_Code' and 'Start_Month' then fill NAs with median value
Weather_data=['Temperature(F)','Humidity(%)','Pressure(in)','Visibility(mi)','Wind_Speed(mph)']
print("The number of remaining missing values: ")
for i in Weather_data:
  df[i] = df.groupby(['Airport_Code','Month'])[i].apply(lambda x: x.fillna(x.median()))
  print( i + " : " + df[i].isnull().sum().astype(str))
```

Resampling:

The accidents with severity level 4 are much more serious than accidents of other levels, between which the division is far from clear-cut. Therefore, we decided to focus on level 4 accidents and regroup the levels of severity into level 4 versus other levels.

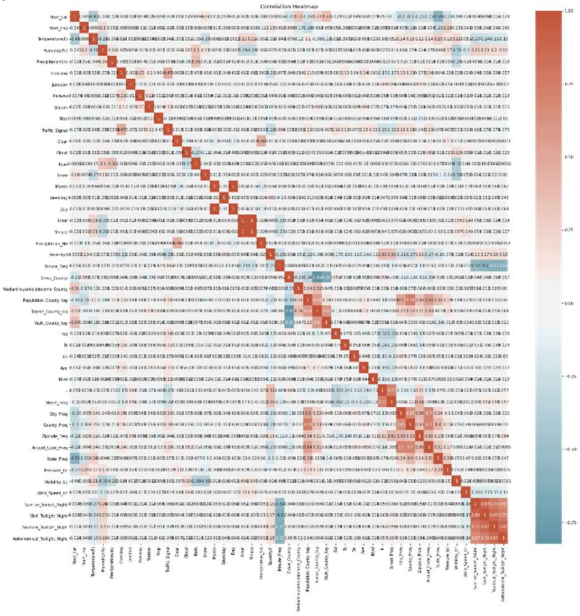
Code:
```
def resample(dat, col, n):
  return pd.concat([dat[dat[col]==1].sample(n, replace = True),
    dat[dat[col]==0].sample(n)], axis=0)
df_bl = resample(df, 'Severity4', 50000)
print('resampled data:', df_bl.Severity4.value_counts())
```

resampled data: 1 50000
0 50000
Name: Severity4, dtype: int64

Correlation:

```
# one-hot encoding
df[period_features] = df[period_features].astype('category')
df = pd.get_dummies(df, columns=period_features, drop_first=True)
# resample again
df_bl = resample(df, 'Severity4', 20000)

# plot correlation
df_bl['Severity4'] = df_bl['Severity4'].astype(int)
plt.figure(figsize=(25,25))
cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
sns.heatmap(df_bl.corr(), annot=True,cmap=cmap, center=0).set_title("Correlation Heatmap", fontsize=14)
plt.show()
```



One-Hot Encoding: One-hot encoding to encode categorical data

```
df = df.replace([True, False], [1,0])
cat = ['Side','Timezone','Weekday']
df[cat] = df[cat].astype('category')
df = pd.get_dummies(df, columns=cat, drop_first=True)
df_int = df.select_dtypes(include=['int']).apply(pd.to_numeric,downcast='unsigned')
df_float = df.select_dtypes(include=['float']).apply(pd.to_numeric,downcast='float')
df = pd.concat([df.select_dtypes(include=['uint8']),df_int,df_float],axis=1)
```

df.info()

Models:
To handle the class imbalance, which is about 100 and a key problem to deal with, we followed below methods

- **Under-sampling**
- **Modified loss function**
- **Ensemble Methods: (EasyEnsemble, BalanceCascade)**

Why classification over regression: Since the output of the model has to give the belongingness of data points in a dataset to a certain category, classification is preferred over regression. We picked the best models of classification according to result, out of which Random Forest gave the best result on which we performed further ensembling and

Train Test Split:

```
# split X, y
X = df.drop('Severity4', axis=1)
y= df['Severity4']
# split train, test
X_train, X_test, y_train, y_test = train_test_split(\
  X, y, test_size=0.30, random_state=42)
# List of classification algorithms
algo_lst=['Logistic Regression',' K-Nearest Neighbors','Decision Trees','Random Forest']
```

Under Sampling:

```
# Randomly undersample majority class to about 10 times of minority class
rus = RandomUnderSampler(sampling_strategy = 0.1, random_state=42)
X_train_res, y_train_res = rus.fit_sample(X_train, y_train)
print ("Distribution of class labels before resampling {}".format(Counter(y_train)))
print ("Distribution of class labels after resampling {}".format(Counter(y_train_res)))
```

Distribution of class labels before resampling Counter({0: 671001, 1: 5413})
Distribution of class labels after resampling Counter({0: 54130, 1: 5413})

```
# Select the state of Pennsylvania
state='PA'
df_state=df_sel.loc[df_sel.State==state].copy()
df_state.drop('State',axis=1, inplace=True)
df_state.info()
```

Logistic Regression:

```
# Logistic regression
lr = LogisticRegression(random_state=0)
lr.fit(X_train,y_train)
y_pred=lr.predict(X_test)

# Get the accuracy score
acc=accuracy_score(y_test, y_pred)
# Append to the accuracy list
accuracy_lst.append(acc)
print("[Logistic regression algorithm]
accuracy_score: {:.3f}.".format(acc))
```

[Logistic regression algorithm] accuracy_score: 0.954.

The K-Nearest Neighbors (KNN) algorithm:

```
knn = KNeighborsClassifier(n_neighbors=6)
# Fit the classifier to the data
knn.fit(X_train,y_train)
# Predict the labels for the training data X
y_pred = knn.predict(X_test)
# Get the accuracy score
acc=accuracy_score(y_test, y_pred)
# Append to the accuracy list
accuracy_lst.append(acc)
print('[K-Nearest Neighbors (KNN)] knn.score: {:.3f}.'.format(knn.score(X_test, y_test)))
print('[K-Nearest Neighbors (KNN)] accuracy_score: {:.3f}.'.format(acc))
```

[K-Nearest Neighbors (KNN)] knn.score: 0.935.
[K-Nearest Neighbors (KNN)] accuracy_score: 0.935.

Decision Tree:

```
# Decision tree algorithm
# Instantiate dt_entropy, set 'entropy' as the information criterion
dt_entropy = DecisionTreeClassifier(max_depth=8, criterion='entropy', random_state=1)
dt_entropy.fit(X_train, y_train)
y_pred= dt_entropy.predict(X_test)
# Evaluate accuracy_entropy
accuracy_entropy = accuracy_score(y_test, y_pred)
# Print accuracy_entropy
print('[Decision Tree -- entropy] accuracy_score: {:.3f}.'.format(accuracy_entropy))
# Instantiate dt_gini, set 'gini' as the information criterion
dt_gini = DecisionTreeClassifier(max_depth=8, criterion='gini', random_state=1)
dt_gini.fit(X_train, y_train)
y_pred= dt_gini.predict(X_test)
# Evaluate accuracy_entropy
accuracy_gini = accuracy_score(y_test, y_pred)
# Append to the accuracy list
acc=accuracy_gini
accuracy_lst.append(acc)
# Print accuracy_gini
print('[Decision Tree -- gini] accuracy_score: {:.3f}.'.format(accuracy_gini))
```

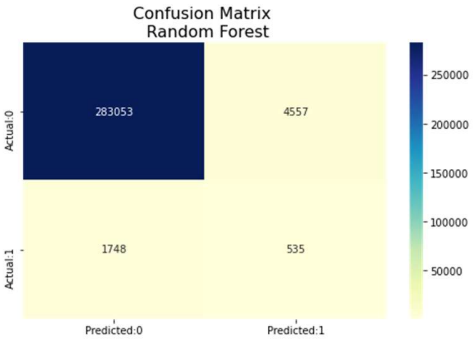
[Decision Tree -- entropy] accuracy_score: 0.967.
[Decision Tree -- gini] accuracy_score: 0.968.

Since Random Forest gave us best results, we did further ensembling activites and improved model functioning

Random Forest:

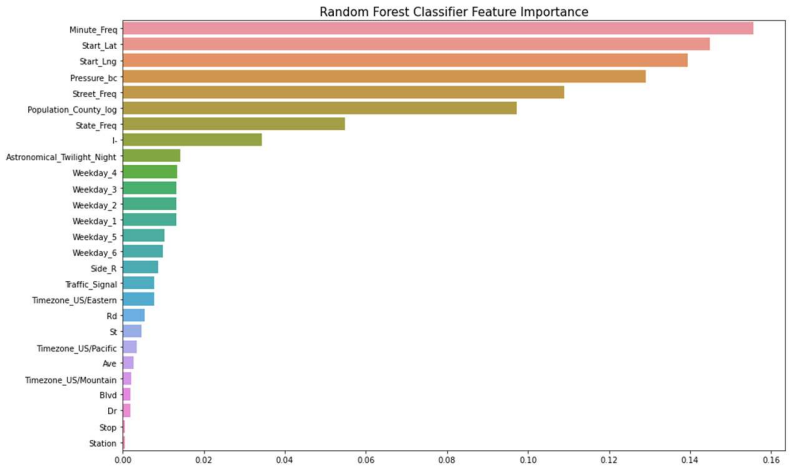
```
clf_base = RandomForestClassifier()
grid = {'n_estimators': [10, 50, 100],
      'max_features': ['auto','sqrt']}
clf_rf = GridSearchCV(clf_base, grid, cv=5, n_jobs=8, scoring='f1_macro')
clf_rf.fit(X_train_res, y_train_res)
y_pred = clf_rf.predict(X_test)
print (classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	287610
1	0.11	0.23	0.15	2283
accuracy			0.98	289893
macro avg	0.55	0.61	0.57	289893
weighted avg	0.99	0.98	0.98	289893



Random Forest Classifier Feature Importance:

```
importances = pd.DataFrame(np.zeros((X_train_res.shape[1], 1)), columns=['importance'],
index=df.drop('Severity4',axis=1).columns)
importances.iloc[:,0] = clf_rf.best_estimator_.feature_importances_
importances.sort_values(by='importance', inplace=True, ascending=False)
importances30 = importances.head(30)
plt.figure(figsize=(15, 10))
sns.barplot(x='importance', y=importances30.index, data=importances30)
plt.xlabel('')
plt.title('Random Forest Classifier Feature Importance', size=15)
plt.show()
```



EasyEnsemble:

```
X_train_res, y_train_res = multi_rus(X_train, y_train, 3, 0.1)
y_pred_proba = np.zeros(len(y_test))
for i in range(len(y_train_res)):
    clf = RandomForestClassifier(n_estimators=100, max_features='auto')
    clf.fit(X_train_res[i], y_train_res[i])
    y_pred_proba += clf.predict(X_test)
y_pred_proba = y_pred_proba/len(y_train_res)
y_pred = (y_pred_proba > 0.5).astype(int)
print (classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	287610
1	0.10	0.15	0.12	2283
accuracy			0.98	289893
macro avg	0.55	0.57	0.55	289893
weighted avg	0.99	0.98	0.98	289893

EasyEnsemble didn't improve the result very much.

BalanceCascade:

```
rus = RandomUnderSampler(sampling_strategy = 0.1, random_state=42)
X_train_res, y_train_res = rus.fit_sample(X_train, y_train)
X_train_res, y_train_res = BalanceCascadeSample(X = X_train_res.to_numpy(),
```

```
                        y = y_train_res.to_numpy(),
                        estimator=RandomForestClassifier(n_estimators=100, max_features='auto'),
                        n_max_subset = 5)
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    287610
           1       0.11      0.24      0.15      2283

    accuracy                           0.98    289893
   macro avg       0.55      0.61      0.57    289893
weighted avg       0.99      0.98      0.98    289893
```

Similar result to EasyEnsemble

Results:

- Data Cleaning and Pre-processing with good imbalance handling, ensembling, and treating different data attributes with different ways helped us achieve high accuracy in various models
- Initially Random Forest gave us 94% accuracy but it jumped to almost 98% accuracy upon performing various actions along with Random Forest
- We can build web interactive model to predict real-time accident severity (Did in Azure Studio)

What can we do with this data?: (Useful to Business Execs or CEO)

- Predict real-time accidents and reduce associated risk
- Improve traffic safety management and accident hotspots
- Area wise accidents to predict trends, hotspots, improve healthcare
- Preventing accidents and mitigate deaths, injuries and property loss

Future Scope:

- Find a better way to handle class imbalance.
- Incorporate this model in a real-time accident risk prediction model or develop a new real-time severe accident risk prediction.
- Detailed relations between some key factors and accident severity can be further studied.
- Explore Re-sampling and Frequency encoding

Team Contribution:
Dataset Selection: Vinay, Akhilesh
Data Attributes Identification: Vinay, Sandeep, Venkat
Data Cleaning & Pre-Processing: Vinay, Naveen, Venkat
Models & Optimization: Entire team
Feature Importance & Future Scope: Vinay, Akhilesh, Naveen
Data Visualization: Vinay, Sandeep
Report Curation: Vinay, Venkat