Machine & Deep Learning Documentation

Machine and Deep learning Documentation - Improve the Road Safety in Breda

Ron L. Tabuchov
Mohamed K. A. M. Elshami
Daria Vlăduţu
Peter Husen

Data Science and Artificial Intelligence, Breda University of Applied Science

Index

1 Background

This document outlines architecture of models used within an AI system developed to improve road safety by offering safer roads instead of faster routes. This documentation is compliant with regulations set out by the EU AI act, and therefore comprises the model architecture, intended use case, and a step by step walkthrough of the system.

2 Goal

The goal of this file is to document the steps taken to build the system for road safety, this is both to fulfil legal obligations as well as to inform users of how models operate, and therefore provide assurance that the model will help them drive more safely.

3 Method

Model Selection: Choose the appropriate ML and deep learning algorithms based on the problem and data characteristics.

Model Architecture: Define the architecture of the deep learning model, including the number of layers, neurons, activation functions, etc.

Hyperparameter Tuning: Set initial hyperparameters and use techniques like grid search or random search to find the optimal values.

Model Training: Train the model using the training dataset, monitoring the training process for convergence and performance.

Validation: Validate the model on the validation dataset to tune hyperparameters and prevent overfitting.

Performance Metrics: Evaluate the model using relevant metrics (e.g., accuracy, precision, recall, F1-score).

Review and Follow-Up: Regularly review the developed ML in GitHub and the progress made on the actionable steps, adjusting the code and methods as needed to ensure continuous improvement.

4 Data collection and Usage

Our data has been collected from several sources. First is a dataset provided to us by ANWB, this dataset takes personal information as well as data collected by their behaviour on the road. ANWB only collects this data from members that sign up to their safe driving insurance, which is a higher tier which is optional. Consent into data collection is therefore present. For the purposes of machine learning, all personal data has been removed from the dataset and only an event id is used.

The second dataset that has been used is the KNMI dataset regarding precipitation. This data is freely available from the KNMI website. The data on precipitation is collected based on the water that falls in specific weather stations which measure the amount of rain that falls in that location. For our purposes we have used the nearest weather station to Breda for analysis.

The third dataset we used is the open-meteo dataset, which is an open-source system that also looks at the weather similarly to KNMI. Open-meteo has 80 years of data meaning that analysis based on it is reliable for the purposes of machine learning.

The preprocessing steps of both models are present in this document.

Using this data, we aim to detect which roads have a disproportionate number of accidents and mark these as high-risk zones. The goal of marking roads as such is to inform people of the increased risk, get them to drive safer or choose an alternative route. By doing this the number of accidents occurring in a year should decrease.

5 Random Forest

Selecting columns and creating a copy to avoid modifying the original DF

```
            df_selected = df_joined[['date', 'eventid',
    'duration_seconds', 'road_name', 'maxwaarde',
                              'incident_severity',
    'risk_category_encoded', 'weather_code', 'risk_level_wmo']].copy()
```

grouping the data frame by road name

```
        X = df_selected.drop(columns=['road_name'])  # Features
        y = df_selected['road_name']  # Target
```

Resets the index of the resulting Series and converts it into a df with a new column named.

those actions resulting a new 2 columns df, where ‘incident count’ column represent the number of incidents per road.

merging the columns to the selected columns on ‘road_name’ to combine the data of the new column (‘incident count’)

Label encoder converts the categorical ‘road_name’ column into numerical labels suitable for machine learning algorithms

X represents the selected columns to use an input: 'road_name', 'weather_code', 'risk_level_wmo', and 'incident_count'.

Y is the target variable, where 'risk_category_encoded' is used as the target variable that model will try to predict based on the input features.

Divide the dataset into training and testing sets, ensuring that the model can be trained on one part of the data and evaluated on another part to test its performance (20%).

Plotting class distribution before and after SMOTE and using the SMOTE method to, ensure balanced representation of the classes in the target variable column ('risk_category_encoded').

Creating and training Random Forest Classifier because of its ability to handle various of data type, balanced and imbalanced data sets. In addition, Random Forest offers benefits such as reducing overfitting, handling missing values, and providing feature importance, making it a suitable choice for classification task.

Making predictions on the test data and evaluate the predictions by comparing the true labels (y_test) with the predicted labels by the machine learning algorithm (y_pred_test_rf).
Generating classification report that includes precision, recall, f1-score, and support for each class.
Precision: The ratio of correctly predicted positive observations to the total predicted positives.
Recall: The ratio of correctly predicted positive observations to all observations in the actual class.
F1-Score: The weighted average of precision and recall. It considers both false positives and false negatives.
Support: The number of actual occurrences of the class in the dataset.

40.37% of the predictions made by the model are correct. This low accuracy indicates that the model may not be performing well on this classification task.

Plotting confusion matrix and examining the error analysis point out a large proportion of Class 0 and Class 2 instances are being confused with each other and with Class 1. This suggests that the features used aren't capturing the distinctions between these classes. Data preprocessing and better features selecting will provide effective and reliable machine learning model.

Iteration 2

Trying different method to achieve higher accuracy by adding additional preprocessing step, weights, and fine tuning for the model. In iteration 2, we count the total number of accidents per road, assign weights to each incident type according to its severity and use the machine learning model to classify risk level according to the new selected features. By calculating the total number of incidents per road, we can identify roads with high incident frequencies and potentially investigate the causes or implement measures to reduce incidents.

This line counts the number of occurrences of each unique value in the road_name column. It returns 2 columns where the index is the unique road names and the values are the counts of incidents.

```
incident_counts = df['road_name'].value_counts().reset_index()
incident_counts.columns = ['road_name', 'total_incidents']
```

By calculating the total number of incidents per road, you can identify roads with high incident frequencies and potentially investigate the causes or implement measures to reduce incidents.

```
incident_types_per_road = df.groupby(['road_name',
'incident_severity']).size().unstack(fill_value=0).reset_index()
```

create a data frame that shows the counts of each type of incident severity for each road.

```
road_incident_data = pd.merge(incident_counts,
incident_types_per_road, on='road_name')
```

This code merges two data frames, incident_counts and incident_types_per_road, based on the road_name column.

```
  severity_weights = {

'HA1': 1, 'HA2': 2, 'HA3': 3, 'HB1': 1, 'HB2': 2, 'HB3': 3, 'HC1': 1,
'HC2': 2, 'HC3': 3,

'HC4': 2, 'HC5': 3, 'HC6': 4, 'HC7': 3, 'HC8': 4, 'HC9': 5, 'HC10': 4,
'HC11': 5, 'HC12': 6,

'HC13': 2, 'HC14': 3, 'HC15': 4, 'HC16': 3, 'HC17': 4, 'HC18': 5,
'HC19': 4, 'HC20': 5, 'HC21': 5

}
```

Defines a dictionary called severity_weights which assigns a weight to each type of incident severity. The weights indicate the severity of each type of incident, with higher weights representing more severe incidents.

```
    for severity, weight in severity_weights.items():
if severity in road_incident_data.columns:
        road_incident_data[f"{severity}_weighted"] =
    road_incident_data[severity] * weight
```

iterates through the severity_weights dictionary and applies the weights to the corresponding columns in the road_incident_data data frame to calculate the weighted counts for each incident severity type.

```
incident_columns = [col for col in road_incident_data.columns if
'weighted' in col]
road_incident_data['severity_score'] =
road_incident_data[incident_columns].sum(axis=1)
```

This code snippet selects the columns that contain weighted incident counts and calculates a severity_score for each row in the road_incident_data data frame by summing these weighted counts.

```
# Calculate the 25th and 75th percentiles
q25 = road_incident_data['severity_score'].quantile(0.25)
q75 = road_incident_data['severity_score'].quantile(0.75)

# Adjust bins to avoid duplicates
bins = [road_incident_data['severity_score'].min() - 1, q25, q75,
road_incident_data['severity_score'].max()]
labels = ['low', 'mid', 'high']
```

This code snippet categorizes the severity_score for each road into risk levels (low – min score to first 25%, mid – 25% to 75% , high – 75% and above) based on predefined bins. It uses the pd.cut function from pandas to achieve this.

```
top_risk_roads = road_incident_data.sort_values(by='severity_score',
ascending=False).head(10)
print("\nTop 10 roads with highest severity scores:")
print(top_risk_roads[['road_name', 'severity_score', 'risk_level']])
```

This code sorts the road_incident_data data frame by the severity_score in descending order and selects the top 10 roads with the highest severity scores. It then prints these top 10 roads along with their severity_score and risk_level.

```
X2 = road_incident_data.drop(columns=['road_name', 'total_incidents',
'risk_level'])
y2 = road_incident_data['risk_level']
```

This code prepares the feature `X2` and the target `y2` for training a machine learning model, as it selects the relevant features and the target variable.

Dropping 'road_name', 'total_incidents', 'risk_level' columns ensures that the model is trained on relevant, non-redundant features while avoiding data leakage and overfitting. This leads to a more accurate and generalizable model when predicting the risk level of roads based on incident data.

```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2,
test_size=0.2, random_state=42)
```

Splitting the dataset into training and testing sets. The training set is used to train the machine learning model, and the testing set is used to evaluate the model's performance on unseen data.

```
rf2 = RandomForestClassifier(random_state=42)
rf2.fit(X2_train, y2_train)
```

This code snippet trains a RandomForestClassifier using the training dataset.

```
y2_pred_test_rf = rf2.predict(X2_test)
print("RandomForest Test Accuracy for Risk Level Classification:",
accuracy_score(y2_test, y2_pred_test_rf))
print("RandomForest Test Classification Report for Risk Level
Classification:\n", classification_report(y2_test, y2_pred_test_rf))
```

evaluates the trained RandomForestClassifier model on the test dataset by predicting the target values and calculating performance metrics such as accuracy and classification report.

6 K-means clustering

Our second model is a K-means clustering model.

Similarly to the previous model, we start by making a copy of the dataframe.

```
df = df_joined.copy()
```

The next lines we pull the date from the DF

```
df['day_of_week'] = df['date'].dt.dayofweek

df['month'] = df['date'].dt.month

df['year'] = df['date'].dt.year

df.drop(columns=['date'], inplace=True)
```

then we cluster based on latitude and longitude.

```
df_clustering = df[['latitude', 'longitude']]
```

We then cluster 5 different groups together with the model and plot this on a scatterplot, because this is based on latitude and longitude it resembles the way it would look like on a map.

```
n_clusters = 5  # Adjust the number of clusters as needed

kmeans = KMeans(n_clusters=n_clusters, random_state=42)

df['cluster'] = kmeans.fit_predict(df_clustering_scaled)


plt.figure(figsize=(12, 8))

plt.scatter(df['latitude'], df['longitude'], c=df['cluster'],
cmap='viridis', marker='o', edgecolor='k')

plt.xlabel('Latitude')

plt.ylabel('Longitude')

plt.title('Clustering of Incidents')
```

```
      plt.colorbar(label='Cluster Label')

plt.show()
```

Then we have a hierarchical clustering. Once again we take the same dates as before, and cluster latitude and longitude. We standardise he values with

```
      scaler = StandardScaler()
      df_clustering_scaled=scaler.fit_transform(df_clustering)
```

Then we check the model based on 20000 samples and parse through those 20000.

```
sample_size = 20000  # Adjust based on your memory capacity

if len(df_clustering_scaled) > sample_size:

    sample_indices = np.random.choice(df_clustering_scaled.shape[0],
sample_size, replace=False)

    df_clustering_sampled = df_clustering_scaled[sample_indices]

else:

    df_clustering_sampled = df_clustering_scaled
```

Then we use the ward method for clustering, this minimises variance within clusters, then the dendogram is plotted. Dendograms plot based on hierarchy, which allows one way of seeing the clusters. Next we establish a scatterplot on a map like we did before.

7 Decision tree model

Our third model is a decision tree model. The model uses road_name, risk_category_encoded, weather_code and risk_level_wmo, to calculate incidents based on road_name. This is done by train_test_split. Then after that a bar chart is plotted to show the distribution of classes. Class 1 has significantly fewer classes than other classes so SMOTE is applied to equalise the distribution. Then we set up the random forest for predicting y_test for the dataset.

```
      y_pred_test_rf = rf.predict(X_test)
```

The model then shows accuracy score, as well as full classification with precision, recall and f1-score. We then plot a confusion matrix for the risk level classification to gain insight on how things are plotted, and have a feature importance chart.

8 DNN

After the basic models, we implemented a deep neural network.

In the first part of the code we select the columns that will be used for the DNN.

Then selected categorical variables are converted to numerical variables.

```
encoder = LabelEncoder()

df_selected['incident_severity'] =
encoder.fit_transform(df_selected['incident_severity'])

df_selected['road_name'] =
encoder.fit_transform(df_selected['road_name'].astype(str))

df_selected['risk_level_wmo'] =
encoder.fit_transform(df_selected['risk_level_wmo'])
```

After that we extract the date information by having days, months and years separately from each other. After this we drop the risk_category_encoded column for X as it will be used for the Y value. It is then one-hot encoded as it is more suitable for classification with multiple classes.

```
X = df_selected.drop(columns=['risk_category_encoded'])  # Features

y = df_selected['risk_category_encoded']  # Target

# One-hot encode the target variable

y_encoded = to_categorical(y)
```

Then we use train_test_split with an 80% training and 20% testing split, standardise the features, and build the model.

The input layer is 128 neurons with ReLU activation, then a dropout layer of 0.5 to prevent overfitting, then a 64 neuron layer, another dropout layer, and another layer with 32 neurons, before reaching the output layer with softmax activation.

The model is then compiled using categorical crossentropy, adam optimiser and accuracy as metric so we can compare it with the other models we made. We print the loss and accuracy and a heatmap.

Second Iteration

After this first iteration we made an L2 version of the model, L2 penalises large weights which prevents overfitting. Early stopping is introduced to reduce computing time, as well as reducing overfitting. Both of these changes lead to an improvement in the output of the model.

9 RNN

Lastly, we made an RNN

For the RNN we converted the times to the date format, sorted our values by the date, and put a label encoder in place to convert categorical value of incident severity into numeric values.

```
label_encoder = LabelEncoder()

df_joined['incident_severity'] =
label_encoder.fit_transform(df_joined['incident_severity'])
```

After this we selected all the features that would be relevant for analysis, used MinMaxScaler() for normalisation. After this we created a sequence for time series analysis.

Data is our input data
Target is the target values based on the data

```
features = ['latitude', 'longitude', 'duration_seconds', 'maxwaarde',
'temperature_2m',
     'rain', 'snowfall', 'snow_depth', 'risk_category_encoded',
'risk_level_wmo']
  target = 'incident_severity'
```

Sequence_length is 10 as a default
Sequences is a list storing the sequences of features
Targets is a list which sorts target values at the end of each sequence.
This then iterates over the data and creates sequences of length 10.
After that we generate sequences and the targets from the data to use in train_test_split.

```
  model = Sequential()
model.add(LSTM(50, return_sequences=True,
input_shape=(sequence_length,
len(features))))
  model.add(Dropout(0.2))
  model.add(LSTM(50, return_sequences=False))
  model.add(Dropout(0.2))
  model.add(Dense(1, activation='linear'))

  model.compile(optimizer='adam', loss='mse')
  model.summary()
```

The next code builds the model with the first LSTM layer having 50 units, outputs are sequenced to the next LSTM layer. The dropout drops 20% of neurons randomly to prevent overfitting, then there is the second LSTM layer which does not sequence outputs to he next layer as there is no 3rd layer. Then another equivalent dropout layer before going to the output with a dense of 1, as the RNN predicts continuous values.
The model is then compiled using adam optimser and MSE loss, and a summary is printed. The model is then trained using 20 epochs, and then evaluated based on the loss, and predictions are made and plotted on the table.

Then future incident severity is predicted, and transformed back to the original scale for interpretability.

Second Iteration

In the second iteration of the RNN has some fewer features to focus on those and see if that makes a difference. Event_id, road_name, risk_level_wmo have been cut. Then we establish model architecture and train like we did in the previous one, then we print actual and predicted values, with one actual value being much higher than the predicted value. We then look at mae, mse, rmse and mape and print those values and plot the distribution of residuals.

Third Iteration

In the third iteration the target variable is risk_level_wmo instead of incident_severity. We switched to using MinMaxScaler instead of StandardScaler, we also use forward filling in case a value shows up as missing somehow.

10 System usage

The system is intended to be used before or whilst driving using a navigation system, navigation systems give the option to drive the fastest route, our model will also give an option for the least risky route. This can be done before beginning with driving but will also offer to change whilst driving. When a user approaches a dangerous road, the system will mention this as well as offering a safer alternative. Users can choose to follow what our system suggest, opt to use the standard navigation, or follow their own road. Our system makes clear to users before usage that it is an AI system, that AI can mistakes, and user discretion is advised. Users should drive with care even if they enter low risk zones, accidents can still occur even if the risk is low.